

# tModMaker – Terraria modding tool

## Contents

Analysis .....	4
Background .....	4
Current Methods .....	5
Definitions .....	6
Survey Results.....	10
A Summary of Current Problems .....	10
Clients and users.....	11
Objectives .....	11
Core .....	11
Extension .....	12
Modeled Solutions.....	13
Current Solution .....	13
Proposed Solution.....	14
Proposed Solution .....	15
Data transfer between components .....	15
Limitations.....	16
Design .....	17
Inventory structure .....	17
Types of mod.....	18
Overall system design .....	20
Inputs .....	20
Processes .....	21
Outputs.....	22
Block design .....	22
Weapon .....	22
Accessories.....	27
Tiles.....	31
Consumables .....	33
Projectiles .....	35
Visual effects.....	35

CanUseItem .....	35
Recipes.....	35
NPCs .....	36
Als .....	38
UI design .....	38
Start Screen.....	39
Create Mod Screen.....	39
Mod Overview Screen .....	40
Details .....	41
Per Item Management.....	42
Item Type Select Window .....	43
Template Selection .....	44
Code Editing Window .....	44
File structure.....	45
Algorithm design.....	49
Mod class .....	50
Item class.....	50
Sprite class.....	50
Code class .....	50
Export and code generation algorithm .....	50
Example code generation function .....	52
Example block code .....	52
Test Strategy .....	57
Technical solution.....	57
C# Solution .....	58
Main form – Main.cs .....	58
Code generator class – CodeGenerator.cs .....	82
Generic block class – GenericBlock.cs .....	96
Mod class – Mod.cs .....	102
Item class – Item.cs.....	103
Create item form – CreateItemDialog.cs.....	105
Edit project details form – EditDetailsDialog.cs .....	106
Help form – HelpDialog.cs .....	108

Name form – NameDialog.cs.....	110
Other sprites form – OtherSprites.cs .....	111
Recipe editor form – RecipeEditor.cs.....	113
Recipe item class – Recipeltem.cs.....	115
Settings form – Settings.cs .....	116
Blockly Scripts .....	117
Tool editor – tool.js .....	117
NPC editor – npc.js.....	134
Projectile editor – projectile.js .....	144
Tile editor – tile.js .....	152
HTML code for editors .....	155
Tool editor – tool_editor.html.....	155
NPC editor – npc_editor.html .....	156
Projectile editor – projectile_editor.html.....	156
Tile editor – tile_editor.html.....	156
Help Pages.....	156
Adding items page – AddItemHelp.html.....	157
Editing items page – EditItemHelp.html .....	158
Editing mod page – EditModHelp.html.....	161
Exporting mod page – ExportModHelp.html .....	162
Testing.....	163
0 Saving, loading and exporting .....	163
1 Adding and deleting items.....	170
2 Editing items.....	173
3 Editing the project .....	177
4 Validation in Blockly .....	179
Blockly input issues .....	186
Evaluation .....	186
Changes in user interface design .....	186
Feedback.....	187
Survey Results .....	187
Verbal Feedback .....	187
Objectives .....	188

Core Objectives .....	188
Extension Objectives .....	190
Improvements .....	190
Closing comments .....	191

## Analysis

### Background

Terraria is a 2011 2D sandbox game, developed by Re-Logic. The gameplay loop is based around defeating bosses, which in turn opens access to new progression and bosses. Crafting is an essential part of gameplay, as it allows for the creation of the vast majority of the game's items. Resources are gathered from killing enemies, or harvesting resources from the blocks which the world is made up of. The game uses a 16-bit art style, inspired by the fourth generation of consoles.

Due to the game's 2-dimensional nature, combined with its simplistic art style and predictable physics, modding has become very popular. The main method for distribution of mods has been tModLoader, since 2015. While it had existed as a third-party application for some time, in 2020, tModLoader was officially released as free downloadable content for the game on Steam, although it was previously endorsed by Re-Logic.



An example of a boss fight added by the Calamity Mod



The new UI added by Recipe Browser

Mods available on tModLoader range from huge scale game overhauls, like the popular Calamity mod or the upcoming Starlight River, which both add huge amounts of content and change base game mechanics, to the addition of a single item. Mods tend to fall into one of two categories, content mods, like the aforementioned Calamity, or quality of life (QoL) mods like Recipe Browser or Boss Checklist. Content mods are significantly easier to develop than many of the QoL mods, as they are adding new versions of things that already exist (items, NPCS, etc.), rather than modifying the base game, or interacting with other mods. This is one of the reasons that content mods are far more abundant, the other being that they are generally more interesting to make.

I intend to create a tool to simplify the creation of content mods for the game. It will provide a block-based interface for creating these mods, removing most of the required coding experience by significantly abstracting the process.

## Current Methods

Currently, the only way to write a mod for Terraria is to write directly into the .cs file. It is recommended that this is done in an IDE such as Visual Studio, as they benefit from IntelliSense, and other quality of life features. This is done using the Terraria library. The documentation for the library is available here: <https://docs.tmodloader.net/>. There are several levels of guide available on the GitHub page here: <https://github.com/tModLoader/tModLoader/wiki#easy-guides>. These are of use for new modders, as they walk you through the process of making most types of mods, and the more advanced sections even touch on some very advanced techniques. The GitHub page also provides several useful files that allow users to address certain things by name, and not numeric ID. However, it is worth noting that the game is typically downloaded via Steam, so few people use or know about the GitHub page.

Spriting, the process of creating the visual elements of a mod can be done in any graphics application. The image format used is a .PNG, with the name of the file corresponding with the .cs file. In the case of an animated sprite one extended sprite is used, with the size of each frame defined in code.

For a content mod, the components of its file are as follows:

- [ModName].cs – the central file for the mod. It is rarely used in simpler mods. Only one instance of this file can exist per mod.
- description.txt and build.txt – these contain the mod description and mod author, version, and display name of the mod, respectively.
- [ModName].csproj – a Visual Studio project set up for debugging.
- Properties/launchSettings.json – contains tModLoader files for debugging.
- Items\[ItemName].cs – exists for each item. Contains the code for its properties.
- Items\[ItemName].png – Contains the corresponding sprite.

These files are best created in tModLoader's mod sources menu, to ensure the structure is correct. This is also where your mod can be built into a .tmod file, which is the format in which mods are used and shared.

For any given item, there are several mandatory properties, and some specific to the type of item it is.

For any item (not an NPC), the follow properties must be defined in the SetDefaults() subroutine:

- Height and width: these define the size of the item's hitbox in-game. The sprite is not scaled to these values, so they should match the dimensions of it.
- MaxStack: this is the maximum number of this item that can be stored in one slot in an inventory. For blocks, this is normally 9999, and for other items like weapons, it is one.

- rare: this defines the items in game rarity. This is used to signal the value of the item but has very little gameplay impact.

There is a second default subroutine, AddRecipes(). It defines the way the item can be crafted. The method recipe.AddIngredient(item, amount) is used to add each ingredient in the recipe.

## Definitions

Mod	<p>Short for modification. A change or addition to a game's code that somehow alters the experience.</p> <p>Mods tend to fall into 3 categories:</p> <ul style="list-style-type: none"> <li>• Quality of life: mods that do not add new content to a game, but improve the experience through small changes, such as to menus.</li> <li>• Content: mods that add new content or features to a game to extend or alter the gameplay experience. For example, adding new enemies or items.</li> <li>• Overhauls: mods that fundamentally change how the game works by replacing large amounts of existing content.</li> </ul>
tModLoader	<p>A modified client for the game Terraria. It allows the game to load mods in the .tmod format. It also provides mod management tools, and an interface with the Steam Workshop to allow users to share and download mods.</p> <p>It also provides tools to create framework for and compile mods.</p>
Sprite	<p>A 2D graphic which defines how something appears on screen in a game. Terraria uses .PNG files to store its sprites.</p> <p>Terraria sprites are in a 16-bit style, though it is worth noting that they are drawn with each "square" using 2*2 pixels.</p> <p>Some items in Terraria have multiple sprites, dictating how it appears in the inventory and</p>

	when equipped, for example.
Item	<p>Anything in Terraria that can be stored in a player's inventory. Items fall into several different categories and must have different properties accordingly. They are as follows:</p> <ul style="list-style-type: none"> <li>• Blocks, which can be placed as tiles in the world. Their sprites appear the same in the inventory as they do in the world.</li> <li>• Weapons and tools, which operate in a similar manner to each other. Each has a use animation, a damage value, and a use time (defining how long the player must wait between swings). Tools also have a value that defines their ability to break certain tiles (represented as a percentage).</li> <li>• Consumables, which have a use animation similar to weapons or tools, and an on-consumption effect.</li> <li>• Equipable items, which can be placed in an equipment slot, and provide an effect from doing so. They can be limited to specific slots, like armor or wings.</li> <li>• Other items, which cannot be used or equipped. These are often parts of crafting recipes.</li> </ul>
NPC	<p>An abbreviation of non-player character. In Terraria, this refers to 3 separate things:</p> <p>Friendly NPCs: these can live in houses provided by the player. They sell items and offer quests to the player. They will spawn after certain criteria have been met in-game. For example, the nurse can spawn when a player has over 200 health.</p> <p>Mobs: short for mobiles. These fall into 2 categories: critters and enemies. Critters do very little except provide atmosphere. Enemies will attack the player. Different enemies and</p>

	<p>critters spawn in different places, and may be dependent on other in game events, such as weather and time of day.</p> <p>Bosses: bosses are special enemies, summoned through a specific process (usually a summoning item). They provide a significant challenge to the player and unlock new content to the player on their defeat.</p>
Class	<p>In Terraria, there exists a system of classes. It is not strongly enforced like in some RPG games, where a class is one of the first things you decide about your character. It is instead based on the 4 (5 if you include Throwing damage, a remnant of a cancelled rogue class) damage types in the game. Players base their equipment around one of these, to make their weapons of that class as strong as possible. The four classes are:</p> <ul style="list-style-type: none"> <li>• <b>Melee:</b> Melee relies on close range weapons such as swords or spears (though later in the game, they will receive swords which fire projectiles). Melee armor tends to focus on high defense, as melee players are generally closest to the enemy. Melee is considered the strongest class, as it has access to the Zenith, the strongest and second most difficult to obtain weapon in the game.</li> <li>• <b>Ranger:</b> Ranger uses ranged weapons, predominantly bows and guns. Most ranged weapons consume ammunition. Ranged armor focuses on reducing ammo consumption, and rewards the player for standing still, as rangers are typically furthest from the enemy. Rangers are very strong in pre-hardmode, the struggle in the later game.</li> <li>• <b>Mage:</b> Mages use various weapons that consume mana, with a focus on area of</li> </ul>



	<p>effect (AoE). They have the least defense of any class, but very high damage output. Mage armor reduces mana usage. The viability of playing mage varies from enemy to enemy.</p> <ul style="list-style-type: none"> <li>• Summoner: Summoners create summons, creatures that will fight alongside the player. Summoners tend to avoid direct combat and have a low defense as a result. Their armor is instead focused on increasing the number of creatures they can summon. Summoners are strong, and the easiest class to play.</li> </ul>
Rarity	<p>Rarity is a system in Terraria that vaguely defines the value of items. Each one has a colour and value, which are as follows:</p> <ul style="list-style-type: none"> <li>• Grey (-1) – reserved for “junk” items.</li> <li>• White (0) – the default value in code. Used for most blocks.</li> <li>• Blue (1) – Items this tier and above cannot be destroyed in lava.</li> <li>• Green (2)</li> <li>• Orange (3)</li> <li>• Light Red (4)</li> <li>• Pink (5)</li> <li>• Light Purple (6)</li> <li>• Lime (7)</li> <li>• Yellow (8)</li> <li>• Cyan (9)</li> <li>• Red (10)</li> <li>• Purple (11)</li> <li>• Rainbow (-12) – reserved for expert</li> </ul>

	<p>mode-only items.</p> <ul style="list-style-type: none"> <li>• Fiery Red (-13) – reserved for master mode-only items.</li> <li>• Amber (-11) – reserved for certain quest items.</li> </ul> <p>These rarities only provide an idea of how difficult an item is to obtain and its value and have little effect on gameplay. However, they are a necessary property of an item when developing a mod.</p>
--	---

## Survey Results

Most people have never made a Terraria mod: among the fifteen respondents, only 1 had previous experience making mods. This is a result of the people I surveyed. I posted the survey on the Reddit community r/Terraria, which is about the game in general, and not specifically about modding. However, as the project is aimed at new users, I wanted the opinions of people who hadn't necessarily made mods before.

People struggle to find documentation: one person with modding experience claimed finding documentation was the most difficult part of learning to mod. Additionally, a few people cited lack of knowledge as the main reason they had not made a mod. This is in line with my earlier observations, and something I hope to address with my project. Unfortunately, as there is documentation available, I do believe that a lot of people were not unable to find documentation, but simply found the modding process harder than expected, and did not seek the necessary resources, through lack of time or interest.

Nearly everyone would be more likely to make a mod if a block-based solution was available: 10 of the 12 people who answered this question said yes. In fact, the idea of a block-based tool seems to make people more interested in modding. 3 people said they were not interested in modding currently, while only one said they were not interested if a block-based tool was available. I suspect this is because it, with a basic understanding of programming, would broadly remove the need for outside resources, and reduce the effort involved in learning to make mods.

## A Summary of Current Problems

- The current method is reliant on an existing knowledge of C#. While most of the properties a modder uses are exclusive to the Terraria library, a knowledge of C# is essential. This creates a barrier to entry, both through the knowledge requirement and the daunting prospect of learning a new programming language, especially for someone with no programming experience, like most Terraria players according to the survey results.

- The documentation for the latest version is incomplete. In 2022, tModLoader moved to support the latest major release of Terraria, 1.4. Unfortunately, as the wiki is community driven, sections are yet to be updated to reflect the changes. This makes some parts of the wiki useless to people developing mods for the latest version. Moreover, the community forums suffer worse from the update. Many help threads are now outdated, removing a huge pool of helpful advice and answers to frequently asked questions from the community. This has made navigating the documentation very difficult for inexperienced modders, as it is now important to check the publication date on every thread or page they read. This is highlighted as a problem by the results of the survey.
- There are reliability issues with the Terraria library. Some users have reported that IntelliSense does not work in the Terraria library. This is another added inconvenience for new users, as they cannot view all the available features, and must rely solely on external resources. While there are fixes suggested online, and there is no official solution from the developers.
- Some aspects of programming a mod are unnecessarily complicated. For example, it is recommended that every user includes a number of files in their project which translate the numeric IDs for each projectile, AI, use animation or status effect to the corresponding name. If you do not consult the wiki, or even not read the page regarding this, as it is not made obvious, you will struggle a lot with referencing a lot of fundamental features.

## Clients and users

It is important to note that, given the nature of the software, the clients and users will be the same people.

Judging by the results of the surveys, the modding tool appeals to people who have an interest in modding, but not the skills or time to do so currently. I am making the assumption that people who answered the survey had some existing interest in modding, as they voluntarily responded.

The majority of respondents have no experience making mods for Terraria, while programming experience is split 50 – 50. Therefore, I plan to assume no programming experience, which should not create too much of an issue as the blocks are specifically designed to be easy. The tool is aimed at new modders to create basic content mods without the steep learning curve of learning to program using a specific, under-documented library.

## Objectives

### Core

1. The program should allow an inexperienced user to create a mod without using outside documentation.
2. Under most circumstances, it should be faster to use the program to create a mod than programming in C#.
3. It should be impossible to make a non-functional mod using the tool. A player could make a mod that works badly, but the game should be able to compile and run it.

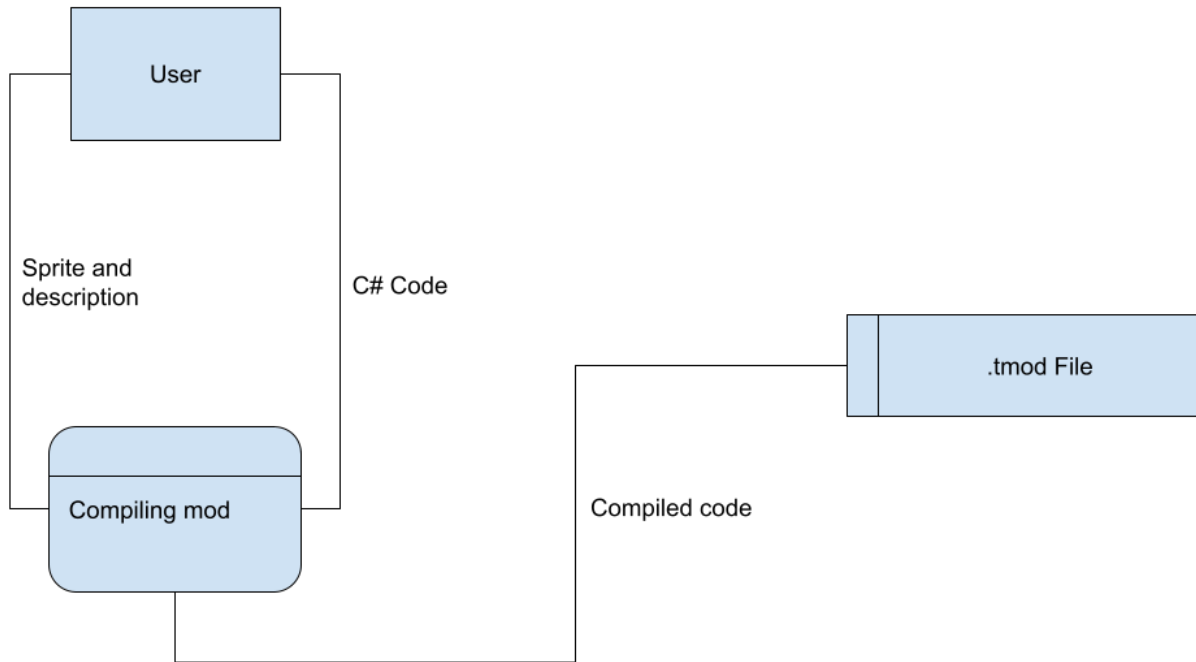
4. The program must allow the user to create the following things in Terraria; items (including equipment and armor), projectiles, NPCs, and tiles.
5. The program should hide much of the file management for the mod to make the process simpler for inexperienced users.
6. The program should produce fully readable C# code, with correct formatting and indentation, to make editing it directly as easy as possible.
7. The program should include an example mod, which highlights how each type of item can be used, and how the code is written.
8. The output of the program should be ready to compile and in the correct directory, so no further input is needed from the user.
9. The blocks should be clear in what they do, by using syntax very close to English.

## Extension

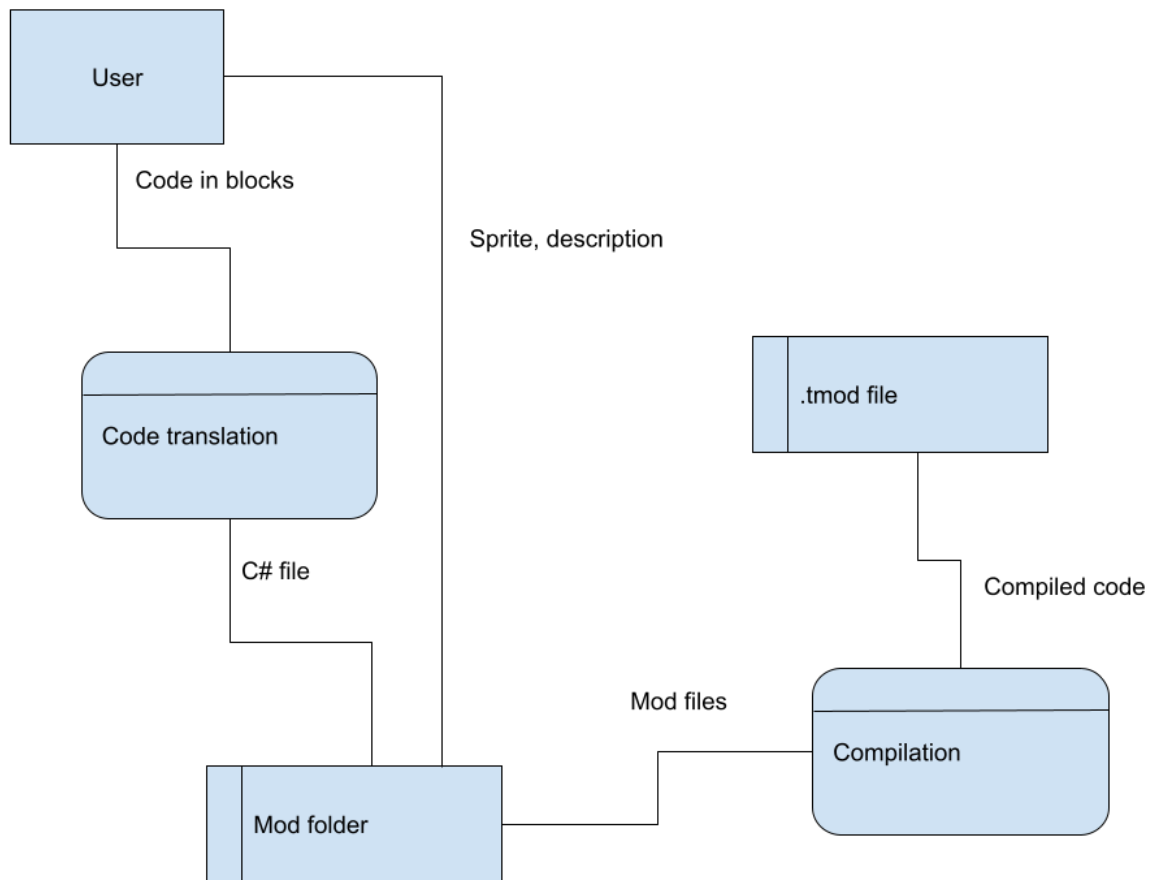
10. The tool should contain some form of walkthrough for new users. It should highlight the basic properties and how they apply to each type of object, as well as explain their in-game effect.
11. The program should offer some interaction with other mods.  
For instance, you could create a new item crafted from items that feature in another mod. This could be done through hard coding for the most popular mods, such as Calamity and Thorium, or allowing the user to create recipes using items that may or may not exist. This second option creates the issue of stability but would allow the user greater control.

## Modeled Solutions

### Current Solution



## Proposed Solution



The current solution is, in terms of the number of steps, very simple. The files are provided by the user and compiled by the software. In comparison, my proposed solution appears more complicated, as it has the additional step of code translation.

Unfortunately, these data flow diagrams have no way of showing the complexity of creating each piece of data. In the current solution, the user must use multiple different pieces of software, namely an IDE, graphics program, and text editor, to create the necessary files for the mod. My proposed solution will have all these features included, reducing the complexity of the process for the user.

Additionally, writing the code should be made significantly easier by my program. While the program must translate the code to C#, for the user, the ability to create the code in blocks will make the process notably quicker, and far more accessible for inexperienced and younger users.

I would like the user to be able to compile the mods within my application. However, I have found little to no information online about how to compile mods without using tModLoader. This would be a nice feature, so I hope I can find a solution, however, as the user will use tModLoader anyway, I do not see it as a huge loss if the mod must be compiled using it.

## Proposed Solution

I intend to use [Google Blockly](#) library as the framework for my project. The program will work in two parts: the web based Blockly interface, and a C# program for file managed and final export. I am looking into having the program build the mod, rather than it being done in tModLoader, but documentation on this is sparse, and I am not certain it is possible.

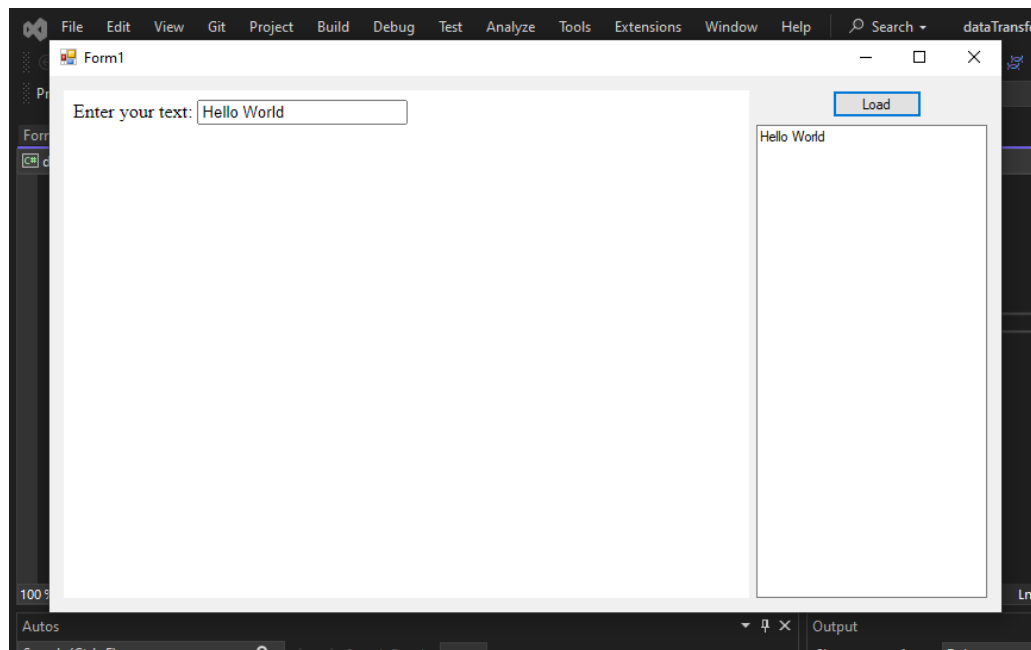
The Blockly interface will be JavaScript based. It will have custom blocks, with custom code generation to C#. By making use of [Block Factory](#), I will be able to streamline the Block design process, which will be valuable over the course of the project. This interface will run in a browser component in C# (using WebView2 for its JavaScript support).

The translated code will be imported into the C# program, where it will be compiled into the correct format alongside the other files needed for a mod, such as sprites and descriptions.

I hope that I will be able to find a way to build the mods within the program, rather than in tModLoader, but, though I have researched the topic, I have struggled to find anything on the subject.

## Data transfer between components

A crucial part of my design is the transfer of data between the browser component and the C# program. I have created a proof of concept, with a basic html form with a text box, and a C# program that loads the contents of the text box into the application.



In the screenshot above, you can see that the text in the web component (left), also appears in the text box in the C# application (right). When the “load” button is pressed, the text is updated.

This program uses WebView2’s web message features. The HTML page contains the following JavaScript script.

```

1. function sendDataToWinForm() {
2.     var text = document.getElementById("theText").value;
3.     window.chrome.webview.postMessage(text);
4. }
5.

```

When this script is executed, the contents of the text box are sent as a message. The C# program contains the following event to receive the text and display it in the text box.

```

1. private void webView21_WebMessageReceived(object sender,
Microsoft.Web.WebView2.Core.CoreWebView2WebMessageReceivedEventArgs e)
2. {
3.     txt_Text.Text = e.TryGetWebMessageAsString();
4. }
5.

```

The WebView2 component is constantly listening for a message, and this event is triggered when one is received.

In the context of my project, it is important for the data transfer to be started by the local application. In order to prompt the html page to send the data, the following code is sent to execute a script within the web content.

```

1. public async void requestData()
2. {
3.     await webView21.ExecuteScriptAsync("sendDataToWinForm()");
4. }
5.

```

This calls the script that returns the data. This procedure is called when the “load” button is pressed.

This functionality will serve as an important part of my project, as it will allow the user to work in Blockly, then have file handling and compiling take place in C#.

## Limitations

My project will be limited to only content mods, as quality of life mods are simply too irregular in terms of what they do, and how they interact with other elements, to create a block-based interface for. However, I do not see this as a serious problem, as my project is aimed at new users looking to make content mods.

A technical limitation of the program will be the Blockly integration. It will have to run in a third-party web browser component. The included web browser component in WinForms is significantly outdated. It runs Internet Explorer 7, which does not support Blockly. I will be using Microsoft’s WebView2 instead, as it is the closest thing to an officially supported Chromium component available. It behaves very similarly to the included web browser, though it is under-documented. However, it is the best option as most alternatives are either paid for or no longer supported. As evident above, I have managed to create the data transfer system between the browser and application already. This was the main issue with compatibility that worried me, so I am confident that using WebView2 is the way forward for this program.



# Design

## Inventory structure



The inventory is where the player will interact with the vast majority of modded and non-modded items. It is important to establish the functionality of some slots in the inventory, as it will allow me to better explain the behaviour of some types of items.

- The storage slots are the main inventory. Any item can be stored here, although the player does not benefit from items stored here unless explicitly specified. For example, the PDA will provide its stats screen when in the inventory, even if it is not equipped. These slots are accessed by pressing Escape.
- The hot bar contains 10 items that are easily accessible. They can be accessed when the inventory is closed, using the scroll wheel or number keys.
- Armor/Accessory/Equipment slots behave similarly to each other. They appear on the right-hand side of the screen while in the full inventory. There are 3 armour slots, head, body, and legs, which can only take the corresponding armour. There are then 7 (8 after consuming the Demon Heart in Expert or Master mode) generic accessory slots. These can equip any accessory, notably wings and boots, which do not have a dedicated slot. There are then 5 specific equipment slots in the equipment menu, which can only contain a specific type of item in each, namely, pet, light pet, mine cart, mount, and hook.
- There is a vanity slot that corresponds to each armour and accessory slot. They are subject to the same restrictions as to what you can put in them. Items in these slots provide no

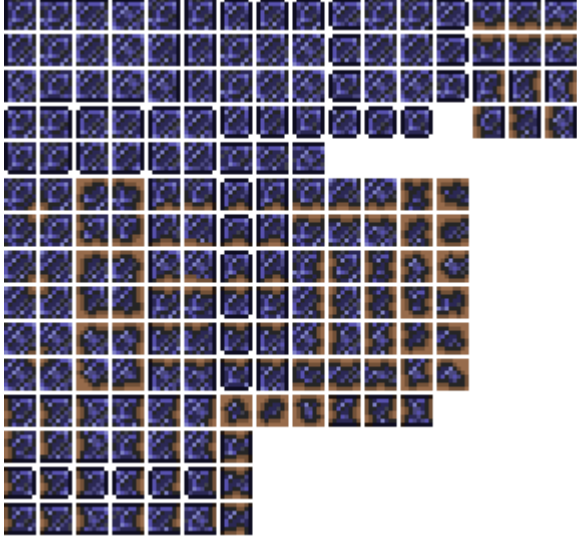

benefit but appear on the character. They are used to decide how your character looks, hence, vanity slots.

- There is a dye slot that corresponds to each armour, accessory, and equipment slot. They allow the player to change the colour of the item in that slot.
- The trash slot is the easiest way to dispose of items. If an item is in this slot, and another is placed on top of it, the first item is destroyed.
- There are 4 slots each for ammo and coins. These slots are automatically filled by the game, with excess items being put into the storage slots.

## Types of mod

Item (Equipable)	<p>This refers to an item that can be equipped to the player's armour, accessory, or equipment slots. Their behaviour is as follows.</p> <ul style="list-style-type: none"> <li>• These items should be stacked to one in the inventory. Additionally, only one of each item may be equipped to a player at any one time.</li> <li>• They provide their benefits only when equipped in the correct slot. It is important to note that they may appear on the character when equipped. This would require there to be 2 sprites for such an item, with one potentially being animated.</li> <li>• They cannot be used outside the inventory – left clicking with the item in hand will do nothing.</li> </ul>
Item (Other)	<p>Items that cannot be equipped or used outside of crafting are the simplest item type.</p> <ul style="list-style-type: none"> <li>• They have one sprite – they appear the same in inventory and when dropped on the ground.</li> <li>• They are stored in the storage slots and cannot be equipped.</li> <li>• Most of these item's stack to 9999, with a few exceptions, like the Broken Hero's Sword, which is a special crafting ingredient, and only stacks to one.</li> </ul>
Tool	<p>Tools refers to weapons (i.e., a sword), and utility tools (i.e., a pickaxe). They share the same base properties as each other and have some properties the differentiate them.</p> <ul style="list-style-type: none"> <li>• Some weapons have an associated projectile. This is not limited to ranged weapons, as weapons like the Star Fury are swords with an additional projectile.</li> <li>• They can be clicked on and used from the storage slots but are normally stored in the hot bar.</li> </ul> <p>Weapons correspond to the 4 classes in Terraria and deal one of the 4 types of damage. Tools deal melee damage if used to hit something.</p> <p>There are 3 essential types of tools in Terraria, though many others exist.</p> <ul style="list-style-type: none"> <li>• Pickaxe – used to break blocks. Better pickaxes have a</li> </ul>

	<p>higher pickaxe power, which defines the speed at which they can break blocks, and what blocks they can break.</p> <ul style="list-style-type: none"> <li>• Axe – used to specifically break trees. Axe power defines how quickly they cut.</li> <li>• Hammer – used to break background walls, and certain other tiles such as the Shadow Orbs in the Corruption.</li> </ul>
Tile	<p>Tiles are specifically the blocks that exist in the world. I say this to avoid confusion, as many items place tiles in the world. A block is an item that places a tile in the world. Most tiles are 1 x 1 “blocks”, however walls, tombstones, foliage, and naturally occurring tiles like trees are all tiles.</p> <p>These are the basic properties for a tile.</p> <pre> Main.tileSolid[Type] = true; Main.tileMergeDirt[Type] = true; Main.tileBlockLight[Type] = true; Main.tileLighted[Type] = true; DustType = ModContent.DustType&lt;Sparkle&gt;(); ItemDrop = ModContent.ItemType&lt;ExampleBlock&gt;(); AddMapEntry(new Color(200, 200, 200)); </pre> <p>Tiles fall into 2 types: Framed and FramelImportant. Framed tiles are “normal.” They are the blocks that make up the world. Notably, they change when certain tiles are placed next to them. Assuming the spritesheet is correct, the line <code>Main.tileMergeDirt[Type] = true</code> defines whether it will merge with dirt. This is an example of how the sprite of a Framed tile looks, illustrating how it changes as other blocks are placed.</p>

	 <p>You can note from this sprite that there is a sprite for each combination of surrounding blocks, and that there are several versions of each sprite to create variation in the world.</p> <p>FramelImportant tiles tend to be larger than Framed tiles and cannot change sprite. Tombstones are a good example of this.</p>  <p>You can see here that there is one sprite for each tombstone. Placing something next to one will not cause it to change. They are sometimes referred to as furniture tiles, as furniture is the most common type of item that uses FramelImportant tiles.</p>
NPC (non-Player Character)	<p>This includes town NPCs, enemies, bosses, and critters. Each NPC has a set of properties similar to that of a player or projectile, and either a reference to an existing AI or a custom AI.</p> <p>Town NPCs will have more complex interactions like dialogue and shop menus, whereas enemies will have more complex attacks. Bosses are like enemies, though they may have multiple AIs as they can have more than one phase based on certain conditions. They may also be made up of multiple independent parts, each with their own code.</p>

## Overall system design

### Inputs

Input Name	Description
------------	-------------

<b>Code Creation</b>	The code will be created by the user in the Blockly interface. They will have access to a library of blocks that will allow them to assemble a mod.
<b>Mod Details</b>	There will be a series of text boxes on the mod overview page to edit the details about the mod, such as name, description, and author. A similar menu like this will exist for each item within a mod, allowing the user to edit the name and tooltip of the item.
<b>Sprite Selection</b>	The user will be able to open a file explorer window and select a file in the appropriate format to be used as a sprite.
<b>Item Type Selection</b>	When adding a new item to the mod, the user will be prompted to select from 3 type options: item, NPC, or AI.
<b>Load Template</b>	On the item type selection screen, there will be a button to load a template for a mod of that type.

## Processes

Translating Code	A script will be executed on the JavaScript end of the solution, prompting the code to be translated. Each value is taken from the fields, and each block is converted to a line of code. These are compiled into one string, then returned to the C# component, where they are written to a file.
Compiling Files	The files will be moved into their correct locations. A copy will be made, and it will be compiled into a usable file for tModLoader.
Adding Items	This will add a new item object to the array of objects. The majority of the properties will be blank or placeholders until the user manually edits them.
Adding Code	The code object in an item will be created, and a file created in the specified location.
Adding Sprites	The sprite object will be created, with a path to the selected sprite.

## Outputs

Output Name	Description
<b>Save File</b>	“Finished” parts of the mod, for example sprites and descriptions, will be saved in the mod’s file structure. The code for each mod will be saved as a Blockly save file, so the user can return to editing it later.
<b>Export Mod</b>	<p>The mod will be made into its correct file structure. If they are not already in place, sprites and descriptions will be saved to the correct file. The Blockly code will be translated to C# and stored in the correct location with a name corresponding to the item it defines. Any other necessary file will be created.</p> <p>If I am able to find a way to compile a mod outside of tModLoader, this will also take place during this process. This would mean the output is a .tmod file. Otherwise, the user will be given the complete mod folder, ready to be compiled in tModLoader.</p>

## Block design

For items, I am planning to use a core block, which defines the obligatory properties of an item, with connectors to add additional blocks which define extra functionality. Please note that throughout this section, I will use code snippets from the example mod on the tModLoader GitHub page.

### Weapon

The basic properties of a weapon are as follows:

```
item.damage = 50;
```

This is the amount of damage the weapon deals.

```
item.noMelee = false;
```

This defines whether the weapon itself swing deals damage. It is set to true on most weapons, as weapons like bows should only deal projectile damage.

```
item.melee = true;
```

This refers to the type of damage dealt by the weapon. The five types are melee, magic, ranged, thrown, and summon. The one that is used is set as true, while the rest are false by default. It is only necessary to specify the type of damage it does deal.

```
item.width = 40;  
item.height = 40;
```

These properties define the size of the weapons on screen. They are not related to the size of the sprite, which is resized to match these dimensions.

```
item.useTime = -100;  
item.useAnimation = -100;
```

The properties define how long the item takes to use. The first line defines how long it takes to swing the weapon, so how long it takes until you can use it again. The second defines how long the animation takes. This should correspond to the useTime property, for the sake of gameplay experience.

```
item.useStyle = 1;
```

This defines the way the item is used. This effects how it looks, but also how it behaves. For example, a great sword, which is swung will deal damage in a wider area than a short sword, which is thrust. The value is an integer, which corresponds to a value on the following list:

0	None	Any item that does not have a useStyle defined
1	Swing	Most usable items, including blocks, background walls and broadswords
2	DrinkOld	Food
3	Thrust	Umbrella, Tragic Umbrella
4	HoldUp	Permanent boosters, summoning items, Magic Mirrors
5	Shoot	Most ranged and magic weapons, yoyos, spears
6	DrinkLong	Recall Potion, Potion of Return
7	EatFood	n/a (unused)
8	GolfPlay	Golf Clubs
9	DrinkLiquid	Potions, drinks, flasks, Hair Dyes
10	HiddenAnimation	Whoopie Cushion
11	MowTheLawn	Lawn Mower
12	Guitar	Stellar Tune, Ivy, Rain Song
13	Rapier	Shortswords, Ruler, Starlight
14	RaiseLamp	Nightglow

```
item.knockBack = 6;
```

This defines how far something hit with the item is knocked back.

```
item.value = 10000;
```

This property, defined as above, defines the value of the item when bought from a merchant. It is important to remember that it will be sold for a fifth of this value. The value is in copper coins. There are 100 copper coins in 1 silver coin, 100 silver coins in a gold coin, and 100 gold coins in a platinum coin, the highest denomination. Accordingly, the item defined here costs 1 gold coin. There are 2 more user friendly methods of defining the price of an item. By making use of `Item.buyPrice()`, then entering values for each value of coin, such as `Item.buyPrice(0, 1, 4, 62)`, it is a lot easier to define the value without doing conversions. `Item.sellPrice()` is used to set the sell price instead of the buy price. So, the lines `item.value = Item.buyPrice(0, 0, 10, 55)`; and `item.value = Item.sellPrice(0, 0, 2, 11)` would have the same effect, as the sell price is a fifth of the buy price.

```
item.rare = 2;    int
```

This defines the rarity of the item. In the game the rarities appear as colours but are referenced in the code by number (see definitions section for a full table). It is an integer the corresponds to the chosen colour.

```
item.UseSound = SoundID.Item1;
```

This defines the sound the item makes when swung. There is a table of sounds available in the base game, and more can be added by the user.

```
item.autoReuse = true;
```

This defines if the item continues swinging when the mouse is held down.

### *Tools*

Tools make use of most of the same properties as a weapon, with an additional property that defines its ability to break blocks. For a pickaxe, this property is `item.pick`, for an axe, `item.axe`, and for a hammer, `item.hammer`. For `item.pickaxe` and `item.hammer`, these values are percentages, but `item.axe` is multiplied by 5 to get the values displayed in game. So, if `item.axe` is equal to 30, it would appear as 150% in game. An item can have more than one of these properties: hammers exist in the vanilla game and combine the properties of a hammer and an axe, meaning they can break both trees and walls. Fishing rods have a similar value `item.fishingPole`, though a fishing pole generally behaves more like a ranged weapon.

### *Firing projectiles*

Various items in Terraria produce projectiles. The obvious examples are guns and bows, but bobbbers and summons are also considered projectiles. `item.shoot` followed by a reference to a projectile, for example `ModContent.ProjectileType<ExampleBobber>()`; `item.shootSpeed` is used to define the speed at which the projectile is launched. Other behaviours are defined in the projectile itself.



### *Boomerangs*

Boomerangs also fire projectiles. However, they have additional code to ensure that only one projectile is in the air at any one time.

### *Mana/ammo/summon slots usage*

Some items use a resource when firing. Ranged weapons tend to consume arrows or bullets. The code `item.useAmmo = AmmoID.Arrow;` is used to assign an ammo type to weapons. This can be any existing item in the game. There is a precedent for this in the base game, as the Sand Gun and Coin Gun fire sand and coins respectively, which are not generally treated as ammunition except in this case.

Magic weapons use mana to as their form of ammunition. The line `item.mana = 100;` sets the mana usage of a weapon.

### *Dual use weapons*

If a weapon is a dual use weapon, meaning the left and right clicks do different things, it contains 2 separate versions of the basic properties. There is a function to check which function is being used. When the weapon is used, the game checks which mode has been used, and assigns the damage values and other behaviour accordingly.

### *Whips*

Whips are similar in functionality to ranged weapons. They have a damaging projectile, while the weapon itself does not deal damage. They use the property `Item.DefaultToWhip` to reference the projectile they use. All further properties are defined within the projectile itself.

### *Consumable weapons*

Some weapons, like javelins, are consumable, meaning they are used up when thrown. This is defined by the property `item.consumable`. When set to true, the item will be removed from the inventory when used.

### *Inflicting status effects*

### *Implementation as blocks*

The core block for a weapon should allow the user to assign the following values:

<code>item.damage</code>	This should appear as an integer field. The user should be prevented from entering a negative value.
<code>item.noMelee</code>	This should be a checkbox, as it is a Boolean value. It should be reworded to something like “deal contact damage,” to avoid confusion with the property <code>item.melee</code> .
<code>item.melee</code> <code>item.magic</code> <code>item.ranged</code>	These 5 values are all assigned as Booleans, though typically only one is true for a given item. So, I will use a dropdown menu to allow

<code>item.thrown</code> <code>item.summon</code>	the user to select their chosen damage type. This will remove the confusion of having all 5 values displayed at once.
<code>item.width</code> <code>item.height</code>	I would like to have these both assigned in one line on the block. They are both integers and must be positive. I will need to assign an upper value, though I am currently unsure of how large an item can be before it becomes unstable.
<code>item.useTime</code>	This will be an integer field. There should be an upper limit to ensure the user does not create something that will break the game.
<code>item.useAnimation</code>	This is much the same as the <code>item.useTime</code> property.
<code>item.useStyle</code>	This should appear as a dropdown of the available animations, listed by name, not number.
<code>item.knockback</code>	This is another integer field, which should have an upper limit to prevent stability issues.
<code>item.value</code>	This should appear as 4 fields, one for each denomination of currency. They should be positive integers, limited at 100.
<code>item.rare</code>	This should appear as a dropdown menu of the colours, listed in rarity order. It would also be useful to have a number displayed for clarity.
<code>item.UseSound</code>	I will leave a connector here. The user should be able to add either an integer (which will appear as default), or an mp3 file containing a custom sound.
<code>item.autoReuse</code>	This will be a checkbox, as the value is a Boolean.

The main block will have a bottom connector to allow additional properties to be added. These properties are as follows, though there may be more added later:

<code>item.pick</code> <code>item.axe</code> <code>item.hammer</code> <code>item.fishingPole</code>	These properties are all Booleans. As an item can have more than one of these values, they will each be a separate block. I may scale up the value for <code>item.axe</code> , so that the proportions are in line with the other similar values. <code>item.fishingPole</code> may have to have an attached property to define a bobber as the projectile, as it will not function without.
<code>item.shoot</code> <code>item.shootSpeed</code>	These values should be assigned as one block, as <code>shootSpeed</code> is necessary for <code>shoot</code> to work properly. <code>Item.shoot</code> is a reference to an existing projectile, while <code>item.shootSpeed</code> is a float.

Boomerangs	This is a fairly niche scenario, but it has to be taken into consideration. Given the code for a boomerang is fairly complex, I may simply create an 'isBoomerang' property, and have it be something that is just turned on or off.
item.useAmmo item.mana	These should be 2 separate blocks with similar functionality. The item.useAmmo block should allow the user to reference any existing item. This will most likely take the form of a dropdown menu, though I may investigate other solutions as the available list is large. item.mana will be an integer field. The input must be positive.
Dual use weapons	The code behind dual use weapons is complex. I intend to provide a second block that defines most of the damaging properties of a weapon. The user can have this in addition to the main block and assign it as the alternate function.
item.consumable	This is a Boolean value, so a checkbox will be used.

## Accessories

The basic properties of an accessory are as follows. Most of them are shared with weapons, so I will not reiterate some explanations.

```

item.width = 22;
item.height = 20;
item.value = 10000;
item.rare = ItemRarityID.Green;
item.accessory = true;

```

This is the property that makes something an accessory. When it is true, the item can be equipped into an accessory slot.

## Wings

They are three important functions when creating a set of wings, which are as follows:

```

public override void UpdateAccessory(Player player, bool hideVisual) {
    player.wingTimeMax = 180;
}

```

This defines how long the player can fly for. 180, as used in the example, is a very high value. Most wings would be much lower.

```

public override void VerticalWingSpeeds(Player player, ref float
ascentWhenFalling, ref float ascentWhenRising, ref float

```

```

maxCanAscendMultiplier, ref float maxAscentMultiplier, ref float
constantAscend) {
    ascentWhenFalling = 0.85f;
    ascentWhenRising = 0.15f;
    maxCanAscendMultiplier = 1f;
    maxAscentMultiplier = 3f;
    constantAscend = 0.135f;
}

```

```

public override void HorizontalWingSpeeds(Player player, ref float speed, ref
float acceleration) {
    speed = 9f;
    acceleration *= 2.5f;
}

```

This defines the horizontal properties of the player's movement. The values are self-explanatory, with speed defining the top speed, and acceleration defining the rate at which they reach it.

### *Mount summons*

It is important to note here that I am describing the item that summons the mount here, not the mount itself. A mount summon is both a tool and an accessory. They can be swung like a weapon or equipped in the mount slot. When equipped, the mount can be summoned with the r key.

All mount summons contain the line `item.mountType = ModContent.MountType<summonName>();`.

### *Pets and light pets*

Pets follow the player, but do not interact with the world. Light pets do the same while providing light. There are enough pets in the base game, and such limited variation, that modded pets tend to clone existing behaviour, and only replace the sprite.

### *Hooks*

Hooks are effectively another ranged weapon. They fire a projectile. The addition code for them is as follows:

```

public override float GrappleRange() {
    return 200f;
}

```

This is the furthest distance the projectile will travel before returning. The highest value in the base game for this is 600.

```

public override void NumGrappleHooks(Player player, ref int numHooks) {
    numHooks = 2;
}

```

This defines the number of hooks that can be fired simultaneously. Any further hooks over this limit will cause the oldest existing hook to disappear.

```
public override void GrappleRetreatSpeed(Player player, ref float speed) {
    speed = 14f;
}
```

This assigns the speed at which the hook returns to the player. This varies per hook.

```
public override void GrapplePullSpeed(Player player, ref float speed) {
    speed = 4;
}
```

When the hook hits a tile, this is the speed at which the player will be pulled towards the hook.

There are other properties which control how the chain of the hook is drawn, for example, and I intend to handle these without the interaction of the user, as they are complicated and need to be done correctly.

### *Movement items*

#### *Stat buffs*

There are many different stats that items can increase. They are simply assigned as variables like the following: `player.allDamage += 19f;`. This code snippet increases all damage the player deals by 19%. To apply a fixed value for a buff, the values can just be assigned, for example, `player.allDamage = 19f;`.

#### *Status effect buffs*

To inflict a status effect, `HitEffect` is used. It uses a reference to an existing status effect.

Status effects applied to the player make use of `player.AddBuff()`. This takes 2 arguments, buff ID and duration.

#### *Boots*

Boots are an accessory and are not limited to a specific slot in the inventory. However, they do contain the line `[AutoLoadEquip(EquipType.Shoes)]` to ensure it appears in the correct place on the player. They tend to grant an increase in running speed for the player. The function `UpdateAccessory()` is used to apply the effects of an accessory. The lines `player.accRunSpeed = 6f;` and `player.moveSpeed += 0.05f;` are used to assign the players maximum running speed and acceleration, respectively.

#### *Armour/clothes*

Armour is effectively an accessory that grants buffs. The only difference is that it fits in only a specific slot. The line `[AutoLoadEquip(EquipType.Body)]` ensures this.

#### *Implementation as blocks*

Wings	This will be a sizeable block, with a name along
-------	--

	<p>the lines of AllowFlight. It will take the following values:</p> <ul style="list-style-type: none"> <li>• <code>player.wingTimeMax</code> as an integer.</li> <li>• <code>ascentWhenFalling</code>,</li> <li>• <code>ascentWhenRising</code>,</li> <li>• <code>maxCanAscendMultiplier</code>,</li> <li>• <code>maxAscentMultiplier</code>,</li> <li>• <code>constantAscend</code>,</li> <li>• <code>speed</code>,</li> <li>• <code>acceleration</code>, all as floats.</li> </ul> <p>I may have to reword some of these properties to make them more comprehensible to the user.</p>
Mount summons	There would be a single input called <code>AddMount</code> , which would be a reference to an existing mount in the game.
Pets and light pets	I will provide a drop-down menu of existing pet AIs and allow them to select from among those.
Hooks	<p>This is a slightly more challenging feature to implement, as being a hook is mutually exclusive with a lot of other functionality. I will have to ensure that the vast majority of blocks cannot be connected to this.</p> <p>The values a hook needs are the floats <code>GrappleRange</code>, <code>GrappleRetreatSpeed</code> and <code>GrapplePullSpeed</code>, and the integer <code>NumGrappleHooks</code>. These will be four values on a single block.</p>
Movement items	
Stat buffs	This a something that can be done by just about any accessory. There will be 2 blocks, <code>IncreaseStat</code> and <code>SetStat</code> . The names are self-explanatory. They will both have 2 inputs, a dropdown list of the players stats, and an integer input for the value.
Status effect buffs	This should provide a drop-down of the existing status effects that can be inflicted.
Boots/armour/clothes	All of these are accessories limited to a specific slot. There will be a block <code>LimitToSlot</code> , which will provide a dropdown of the available slots for an item to be placed in.

## Tiles

Tiles are placed by block items. They use `item.createTile` followed by a reference to a tile to place a tile.

### *Ores*

Ores are a special type of tile. They are defined as an ore with the line `TileID.Sets.Ore[Type] = true;`. A handful of items highlight ores in the world. For this to happen these 2 lines are needed:  
`Main.tileSpelunker[Type] = true;`

`Main.tileValue[Type] = 410;`

The former means the tile is highlighted as an ore when the player is using a Spelunker Potion, or another item with a similar effect. The latter sets the rarity value of the item high enough that the metal detector accessory, which displays the rarest nearby item, will detect it.

### *Doors*

Doors have 2 sprites and code files – open and closed. The file for open contains the line `closeDoorID = ModContent.TileType<ExampleDoorClosed>();`, and vice versa.

### *Walls*

Walls are one of the simplest tiles to implement. The line `Main.wallHouse[Type] = true;` defines them as a wall.

### *Plants*

There are several different types of plants in Terraria.

- Saplings, which call `growSuccess = WorldGen.GrowTree(i, j);` to grow into a tree. `i` and `j` are the saplings coordinates.
- Herbs, which use `tile.frameX += FrameWidth;` to shift along the spritesheet and display the next growth stage.

### *Interactive furniture*

Some furniture, like chairs and beds can be interacted with. To enable this the tile is added to a group, which defines its functions. The property that does this is `adjTiles = new int[] { TileID.Chairs };`, where the `TileID` is that of the correct set.

### *Chests*

There is a large amount of code for a chest, so I will not explain it in its entirety here. Importantly, all chests behave the same. So, creating a chest will be as simple as adding the same block of code. This is also where locked keys are dealt with. There is a check to see if the player has the correct key and will consume it and unlock the chest if they do.

### Platforms

Platforms are a set of blocks, with the notable characteristic that the player can press down/s to fall through them. As this feature is present in all platforms, the line `TileID.Sets.Platforms[Type] = true;` is used to implement this.

### Gravity affected blocks

Gravity affected blocks are blocks that require a block underneath to prevent them from falling. The most common example of this is sand. The line `TileID.Sets.Falling[Type] = true;` is used to give them this characteristic.

### Banners

Banners grant players near to them a buff against the enemy they correspond to. They make use of the `NearbyEffects()` function. The buffs they give are assigned through the lines `player.NPCBannerBuff[mod.NPCType(type)] = true;` and `player.hasBanner = true;`. It is worth noting that nothing about a banner is exclusive to a banner – these effects could be applied by any tiles in the same way.

### Drops

The drop property defines what the tile will drop when destroyed. This is just a reference to any item in the game: it does not actually have to correspond in any way to the tile. A tile also does not have to have a drop. Some items, like the demon altar, do not drop an item when broken.

### Implementation as blocks

Light blocks	
Ores	When defining any tile, the tile set is important. There will be a dropdown for all tiles which allows the users to select a tile set. Additionally, a Boolean to decide whether it is highlighted by a spelunker potion, and an integer to define its value should be available to all tiles.
Doors	Doors have several properties, but the main one is the reference to the open and closed tile. There will be a Boolean for whether it is a door, and a dropdown to reference the other part.
Walls	There will be a Boolean input to decide if something is a wall. This will have to disable a lot of other functionality for the item, as their functionality is limited.
Plants	The 2 types of plant are saplings and herbs. For saplings, there will be a block that will define the tree it will grow into. For herbs, the user will have to input the frame width to allow the program to move between stages correctly.



Interactive furniture	This will be covered in the tile set selection block.
Chests	While there is a lot of code for this, only 2 pieces of information are needed from the user. There will be a Boolean to decide if it is a chest, and another to add a key if necessary.
Platforms	Platforms characteristics are defined by their tile group. This will be handled by the generic tile group block as mentioned above.
Gravity affected blocks	Gravity affect blocks are again defined by their block group. However, they also need a damage value for when they fall on a player or NPC. This is simply an integer.
Banners	Banner is a term for a type of tile that buffs the player against certain enemies. They have no property that inherently makes them banners. They will need a “provide buff” block, which would allow the user to enter the type of buff provided, and optionally a creature type, as this functionality would not be limited to banners.
Drops	This block must provide a drop-down of every item in the game. Obviously, this is inefficient, so I will also provide a text input block, that may cause errors from incorrect inputs, but will be more efficient for experienced modders.

## Consumables

### *Potions*

Applying the effect of a potion is done in the `UseItem()` event. A healing potion uses an `item.healLife` value greater than zero to identify it as a healing item. However, this value can be overwritten in the `GetHealLife()` event to calculate a different value for the amount of life gained.

To apply a status effect from a potion, the lines `item.buffType = ModContent.BuffType<Bufs.ExampleBuff>();` and `item.buffTime = 5400;` are used. The first assigns the buff from the existing buffs, and the second assigns the duration in game ticks. There are 60 ticks in a second, so this buff lasts for 90 seconds.

### *Permanent buffs*

Permanent buffs are handled a lot like regular consumables, with an addition line to make the change permanent. The line `player.GetModPlayer<ExamplePlayer>().itemName += 1;` is used in the `UseItem()` event, and increases the number of the item used by the player by one.

## Ammo

The process of firing ammo takes place in the weapon's code, but a few properties must be defined for the ammo itself. The ammunition has most of the same properties as a weapon, to define damage, damage type, and other basic properties. The additional properties include `item.shoot`, which assigns a corresponding projectile to the ammo, `item.shootSpeed`, which defines the velocity of the projectile, and `item.consumable`, which means the ammo will be used up. Importantly, the property `item.ammo` is used to define the ammo group it is part of, like arrows or bullets.

## Boss summoning items

Boss summoning items behave like any other usable item. In the `UseItem` event, they use the function `NPC.SpawnOnPlayer` to spawn the corresponding NPC at the player's location.

## Explosives

Explosives fire an explosion projectile. Aside from this, they are effectively normal items. The projectile deals damage in a set area. The bulk of the code for it is just the visual effects. A useful function is `projectile.timeLeft`. When it reaches zero, the projectile is destroyed.

Potions	There will be a "consume on use block", with a Boolean input, that can be applied to any item. There will also be an apply status effect block, with a dropdown to select an existing effect. A potion would make use of both to create something that gives a buff and is used up. This same functionality could be used for food.
Permanent buffs	The apply status effect block will have a second Boolean value, which will decide if the effect is permanent or not.
Ammo	Ammo will be an extension of a weapon, with the additional shoot speed and projectile properties. It will also make use of the aforementioned "consume on use" property.
Boss summoning items	For boss summoning items, a summon NPC block will be necessary. It will be a dropdown list of existing NPCs.
Explosives	This will just be a weapon, firing a unique projectile. The <code>projectile.timeLeft</code> function will be implemented as a block.

## Projectiles

There are a lot of different additional properties regarding projectiles, but the main thing to note is that they are effectively just NPCs. They have an, albeit basic, AI. They also have a `SetDefaults` event which defines various characteristics such as collisions. Depending on the projectile, it may also have “on hit” effects in `OnHitPlayer` event. Most of the detail on how the AI works will be covered in the AI section.

## Visual effects

### *Dusts*

Many items make use of particles for visual effects. This could be anything from the dust from mining a block to the smoke from an explosion. Dust can be created in two ways. The item could define its own simple dust using `Dust.NewDust`. This takes a series of parameters to define things like velocity, position and colour. This is used by items like the magic mirror, that only need generic particles for their visual effects. The second way of creating dust is to reference an existing dust. This will simply make use of a more advanced dust that may have more complex behaviour, or a custom sprite. These are referenced by setting the `dustType` property to an exist dust.

## CanUseItem

`CanUseItem` is an event used in some items to check if the player meets the criteria to use it. It contains a return statement, which will return true only if certain conditions. For example, to make an item usable only after killing Plantera, the line `return NPC.downedPlantBoss` would be used. Typically, a statement would be more complex than this. It is normally used by boss summoning items, and checks you are in the correct biome, at the right time of day, and other similar criteria.

## Recipes

Recipes are a property of any item that can be crafted. They typically specify the items needed and the crafting station for the item. The following example is taken from the example mod on GitHub. It is worth clarifying that a recipe is not a mandatory feature of an item. There are plenty of items in the base game that cannot be crafted.

```
public override void AddRecipes() {
    ModRecipe recipe = new ModRecipe(mod);
    recipe.AddIngredient(ItemID.BeetleHusk);
    recipe.AddIngredient(ModContent.ItemType<ScytheBlade>());
    recipe.AddIngredient(ModContent.ItemType<Icicle>());
    recipe.AddIngredient(ModContent.ItemType<Bubble>());
    recipe.AddIngredient(ItemID.Ectoplasm, 5);
}
```

This is an example of where multiple of the same item are used in a recipe. It is simply the ID as normal, then a comma followed by the quantity.

```
recipe.AddTile(TileID.MythrilAnvil);
```

This is where the necessary tile is defined. Typically, this will be a crafting station, like the above anvil, but it could be any tile. For example, the crafting station for a bottle of water is water.

```
recipe.SetResult(this);  
recipe.AddRecipe();
```

This adds the recipe to the game's recipe list. It also shows the definition of this recipe has ended.

```
recipe = new ModRecipe(mod);
```

This is the beginning of the definition of an alternate recipe. There is no limit to the number of different recipes an item can have.

```
recipe.AddIngredient(ModContent.ItemType<BossItem>(), 10);  
recipe.AddTile(ModContent.TileType<ExampleWorkbench>());  
recipe.SetResult(this, 20);  
recipe.AddRecipe();  
}
```

## NPCs

When it comes to NPCs, there are 2 distinct components: the NPC itself and the AI. The NPC is defined somewhat like an item, as it makes use of the same `SetDefaults()` function. The code below is an example of the defaults for a modded NPC, in this case, an octopus.

```
npc.lifeMax = 1100;  
npc.damage = 160;  
npc.defense = 90;  
npc.knockBackResist = 0.3f;  
npc.width = 28;  
npc.height = 44;  
npc.aiStyle = -1;  
npc.noGravity = true;  
npc.HitSound = SoundID.NPCHit1;  
npc.DeathSound = SoundID.NPCDeath1;  
npc.value = Item.buyPrice(0, 0, 15, 0);  
npc.hide = true;  
banner = npc.type;  
bannerItem = ModContent.ItemType<OctopusBanner>();
```

An important line in this is `npc.aiStyle`. In this case it is -1, referring to a custom AI, but there are over 100 existing styles. The bulk of these are specific for bosses, but there is also an AI style for all regular mobs, for instance worms or flying creatures. For most mobs, these AIs will be enough. It is only custom bosses that are likely to need a new AI.

There are far too many properties to cover in full here, so I will highlight a few of the most important and most used features. The bulk of NPCs fall into one of three main categories – hostile mob, town NPC or boss. There are a couple of other categories like mini-boss or critter, but they are not particularly relevant, and the properties described here will still describe them.

### *Hostile mob*

Hostile mobs are simply not declared as friendly. This means that the mob will deal contact damage to the player. They will also typically make use of a hostile AI which will chase the player.

### *Town NPC*

There are 2 properties a town NPC must have to work as intended:

- [AutoLoadHead]

This function loads the head of the NPC from a separate file to display on the map. This means there must be a second file attached to the NPC, which contains the head.

- `npc.townNPC = true;`

This is what defines the NPC as a town NPC. This defines how it interacts with housing, furniture, other NPCs, and the player.

Town NPCs must have a suitable house to live in. There is a default method, but it can be overwritten if the NPC needs different conditions.

For names, the NPC uses the `TownNPCName` function. It is a randomised value used in a switch-case statement of names.

NPC dialogue is handled in a similar manner. The bulk of it will be a switch-case, but there may be other dialogues that are only used in certain circumstances, such as the presence of a specific NPC. It is also possible to weight messages so some will be said more than others.

Some NPCs have up to two chat buttons. The names of the buttons are defined in the `SetChatButtons` event. In the `OnChatButtonClicked` event, their functionality is defined. A common function is to open a shop menu.

This is done in the `SetupShop` event. This event allows the user to add an item associated with a numeric ID. The typical way to this is like this:

```
shop.item[nextSlot].SetDefaults(ModContent.ItemType<ExampleItem>());  
nextSlot++;  
shop.item[nextSlot].SetDefaults(ModContent.ItemType<EquipMaterial>());  
nextSlot++;
```

It adds an item in the slot, then increments the slot ID. There may be more items defined in IF statements, which are only available under some circumstances.

Friendly NPCs can have more properties, but they are more relevant to hostile mobs, so will be covered there.

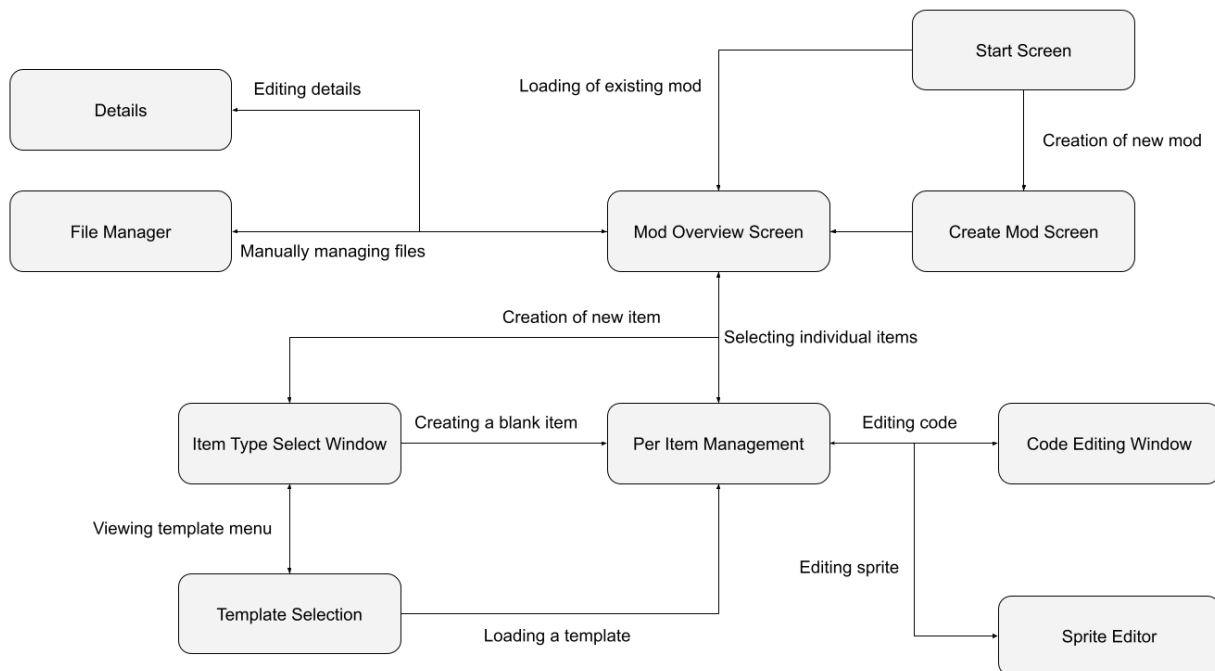
## Boss

## AI

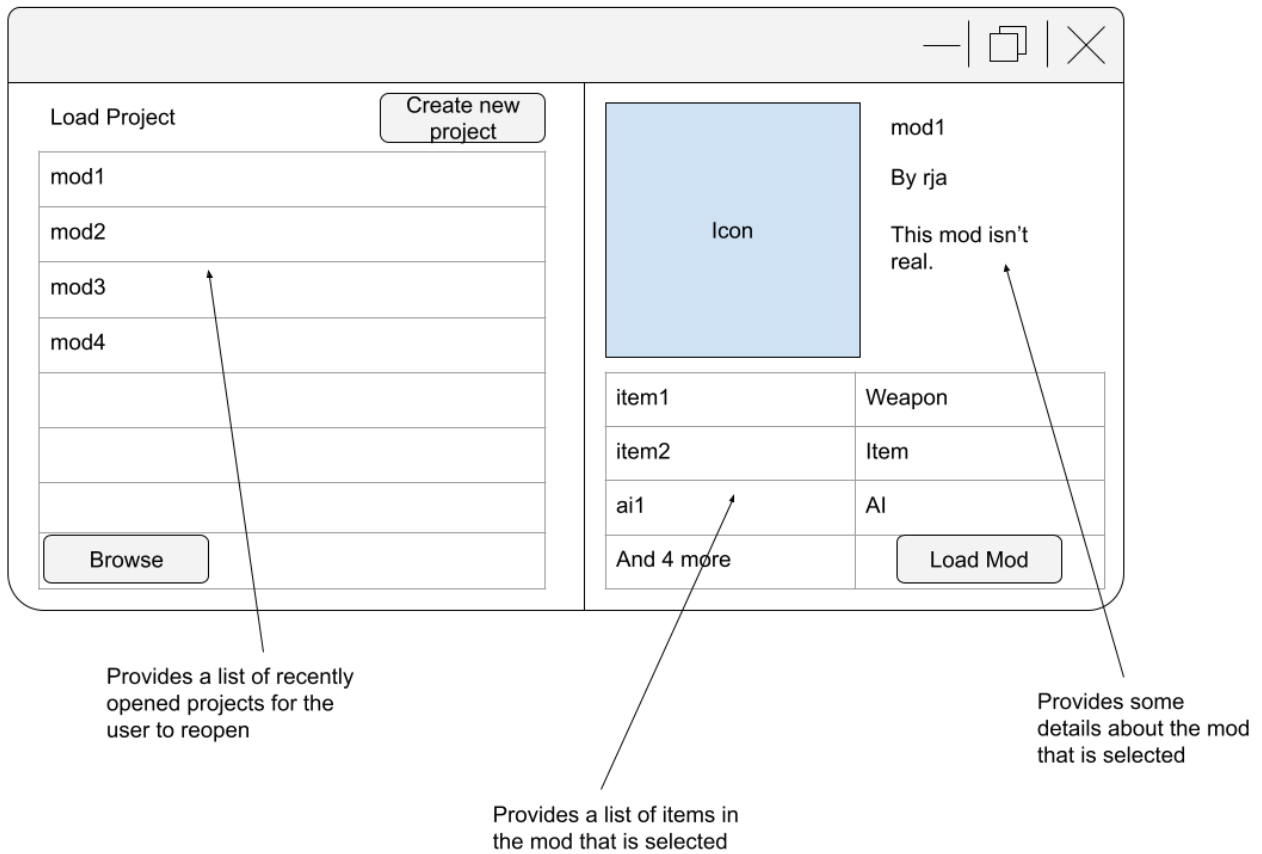
AI is used not only for NPCs, but also for projectiles. They are effectively the same, though projectiles tend to be significantly simpler. An AI is essentially a collection of IF statements. They define what the AI does in what circumstances, normally regarding distance to the player. A boss may have several AI, as they can have different phases based on their remaining health, which each have unique behaviours. A segmented boss may also have multiple parts with their own AI, such as the different arms of Skeletron Prime.

## UI design

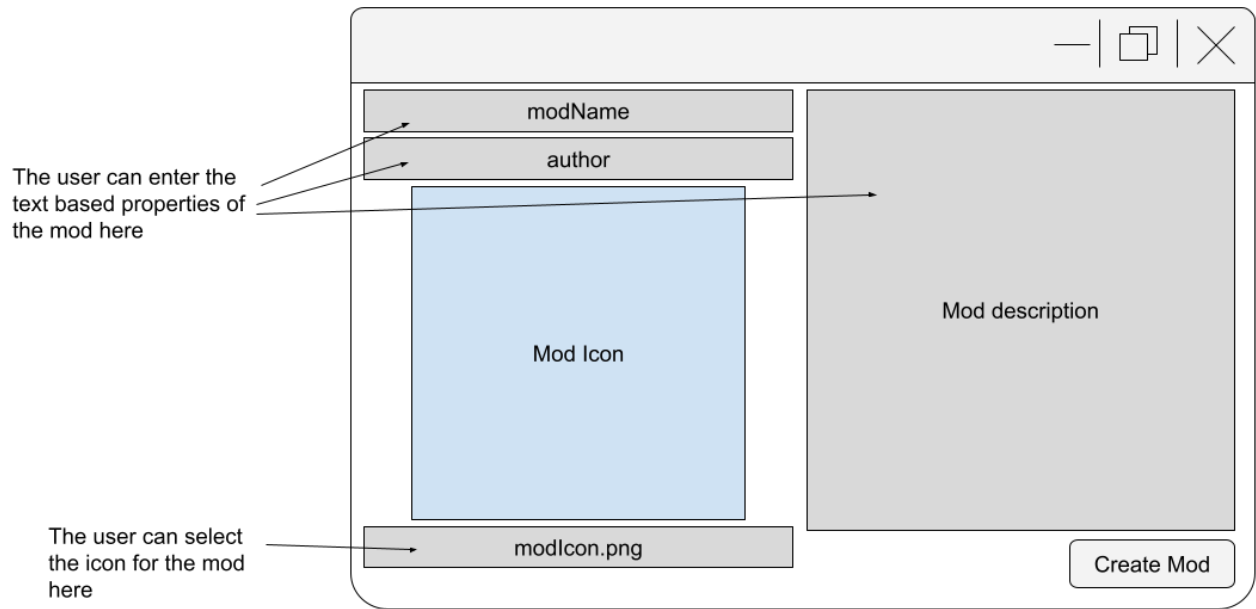
This is my planned design for the structure of my program. I will fall into 2 main menus, one as an overview of the mod, and one to edit an individual item within the mod. Each of these screens will branch into menus that allow the user to edit each part of each item.



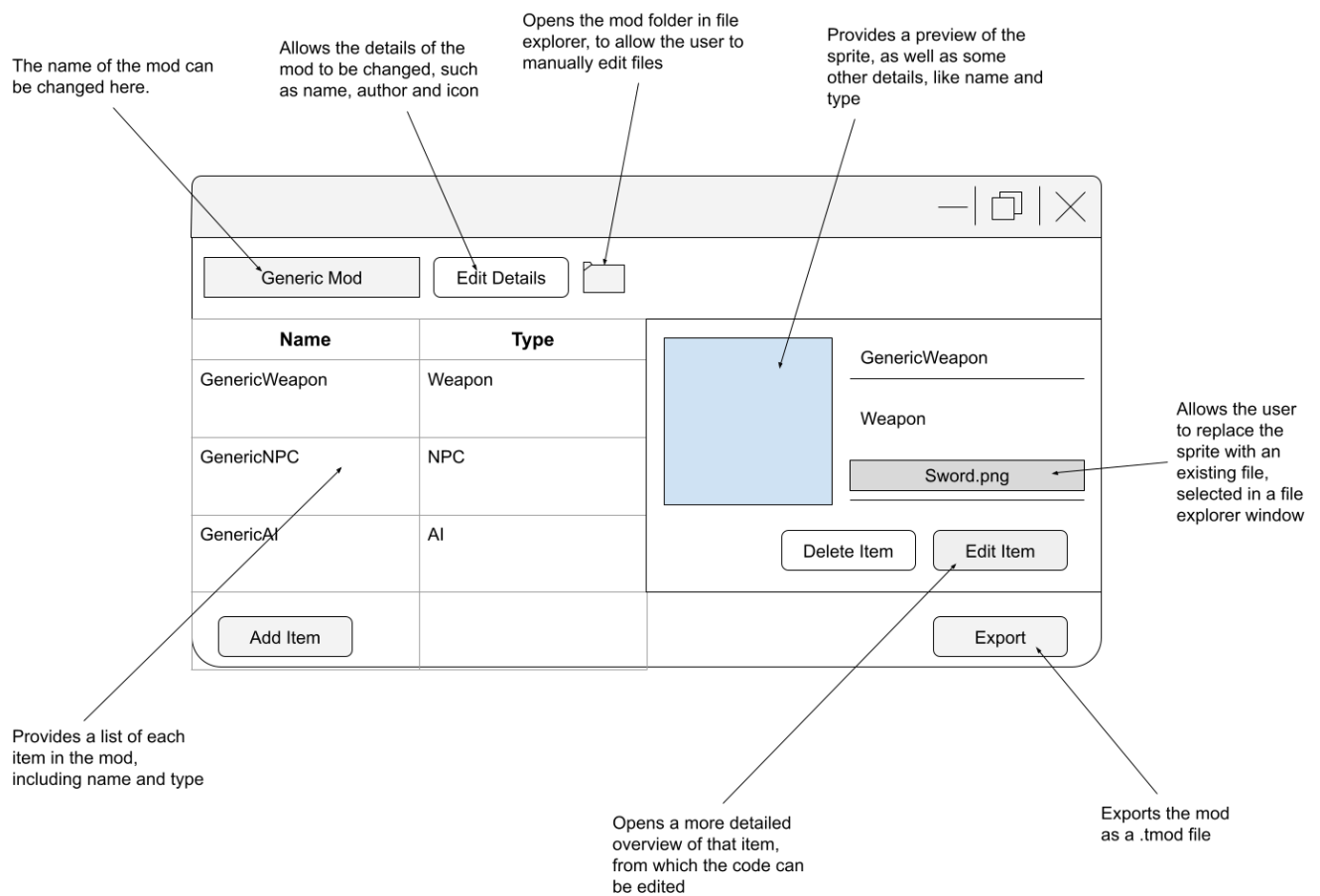
## Start Screen



## Create Mod Screen



## Mod Overview Screen



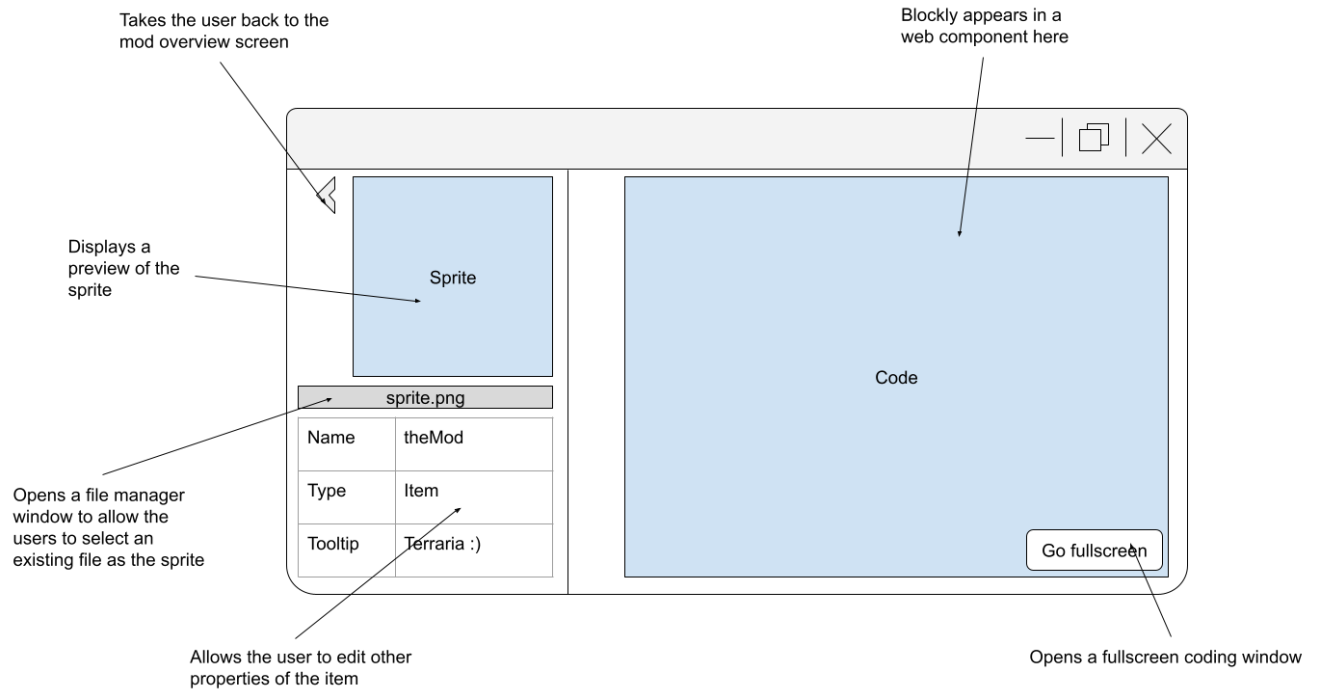


## Details

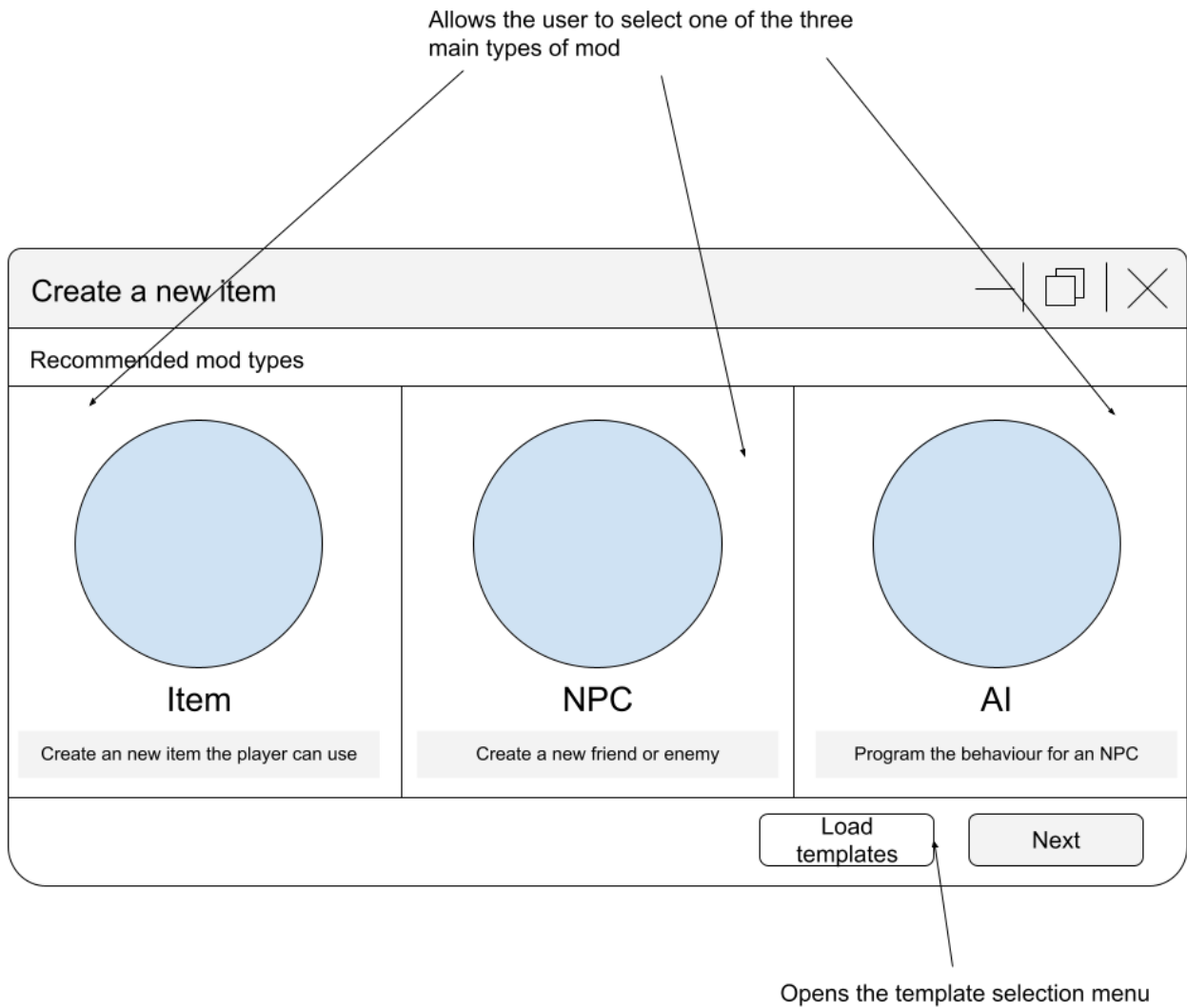
The edit details screen will be nearly identical to the mod creation screen, as the same properties are being changed.

The image shows a UI mockup of an 'edit details' screen. It features a light gray header bar with standard window controls (minimize, maximize, close) on the right. The main content area is divided into two columns. The left column contains a light gray rectangular input field at the top labeled 'modName', a large light blue square placeholder in the center labeled 'Mod Icon', and another light gray rectangular input field at the bottom labeled 'author'. The right column is a large gray rectangular area labeled 'Mod description'. At the bottom right of the screen, there are two rounded rectangular buttons: 'Go Back' and 'Save Changes'.

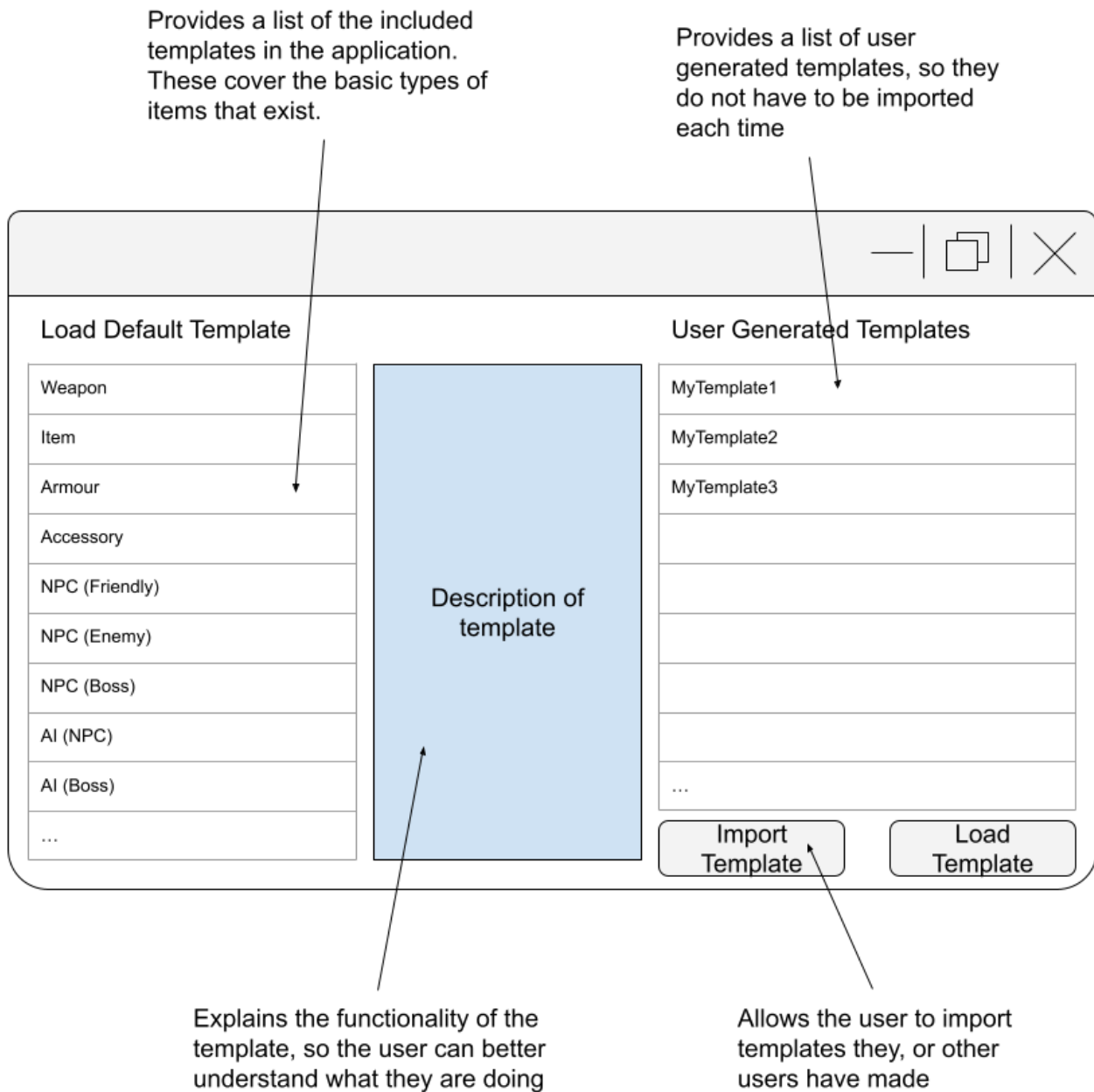
## Per Item Management



## Item Type Select Window

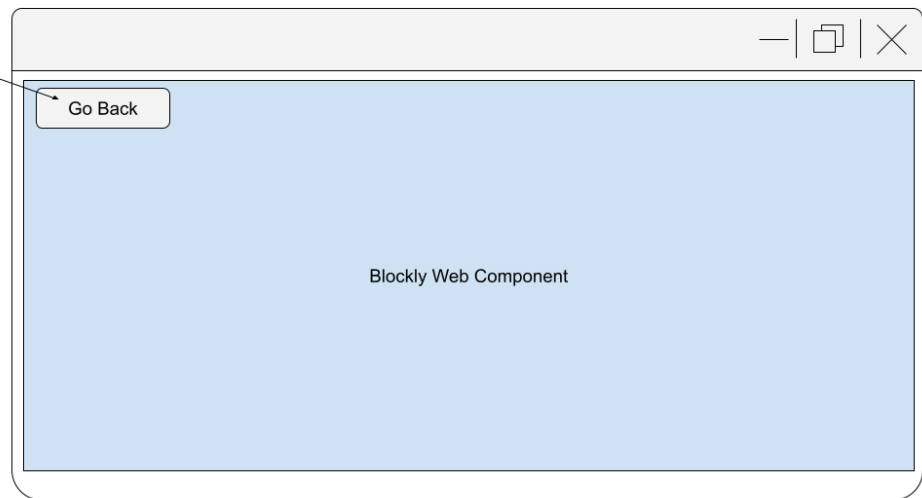


## Template Selection



## Code Editing Window

Returns the user to the per item management screen for that item

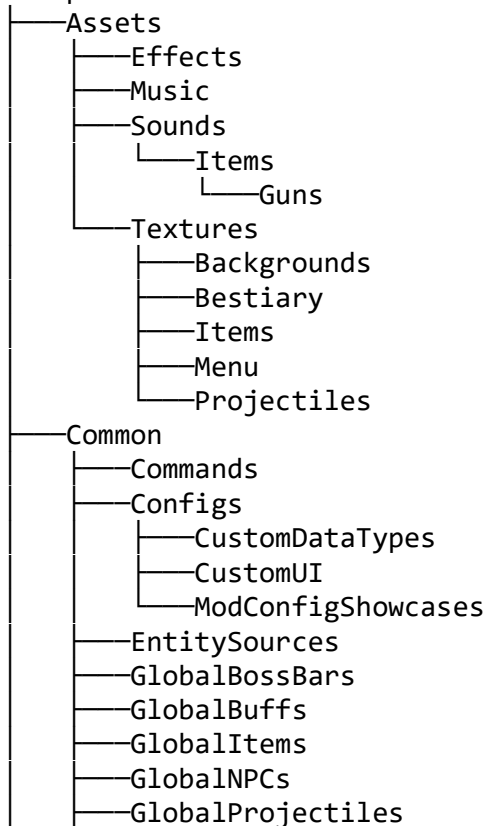


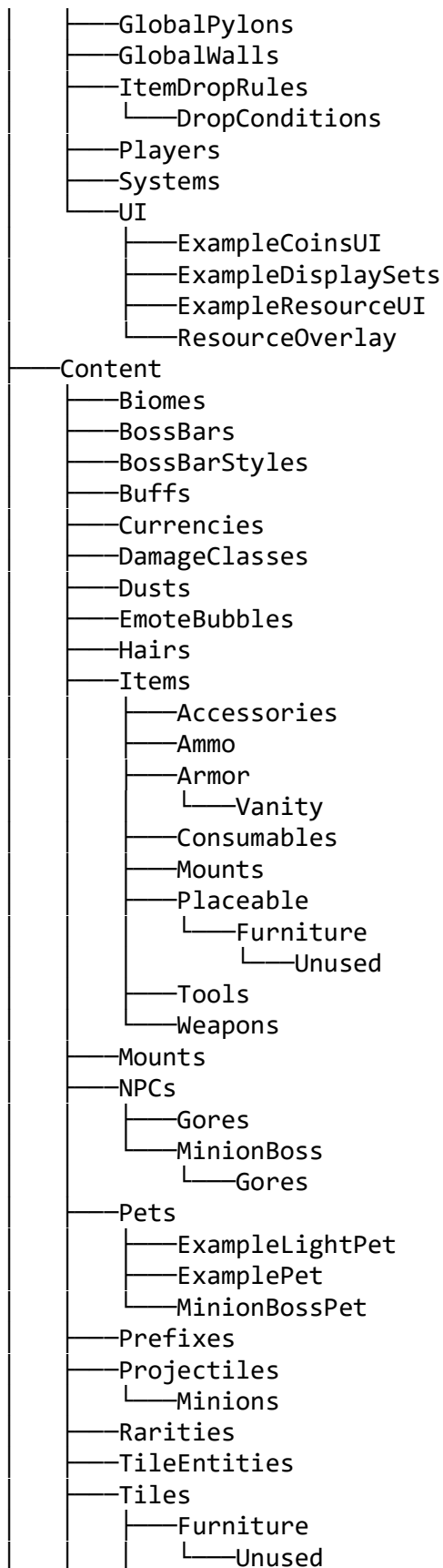
## File structure

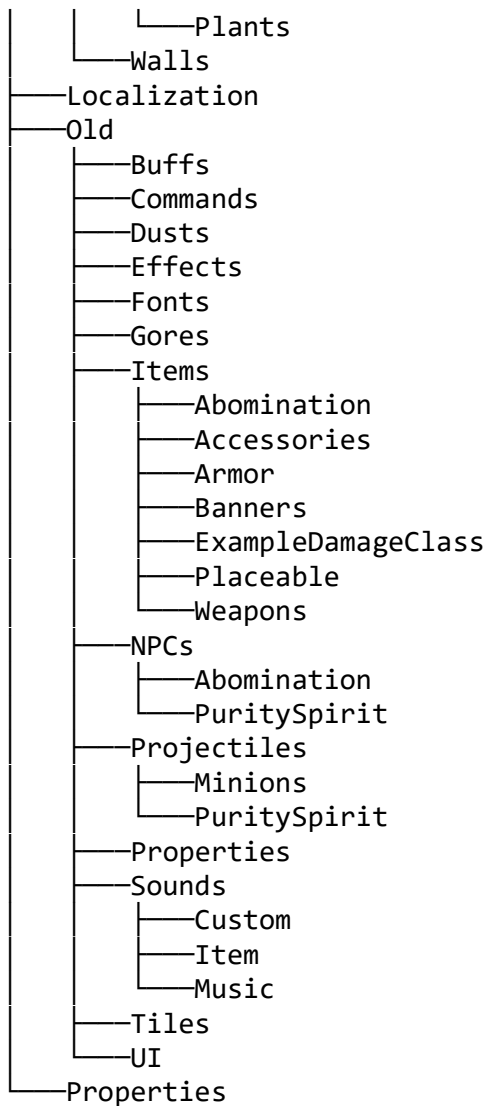
Much of the file structure used in this project will simply be what is used by tModLoader.

Depending on the size of a mod, the file structure can vary drastically. Taking the example of the example mod available on the GitHub page, which offers an example of each possible feature of a mod, the file tree is as follows:

### ExampleMod

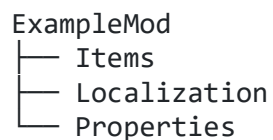






It is worth noting that it is unlikely a user will make a mod of this scale, so this is effectively the worst-case scenario.

In contrast, the mod template created by tModLoader, which contains a single example item, looks as follows:



Realistically, an average mod would be somewhere between these two in terms of size, as most mods do not make use of a lot of features. The folders that will likely appear are Items, NPCs, Projectiles, and Tiles.

I will assemble a file structure like this as items are added to the mod. The majority of this will take place in the Items folder. The .png sprite files can be stored in their correct locations as soon as they are created, and when the code for each item is translated from Blockly, it will also be placed in the corresponding folder.

The Blockly saves files are effectively a text file. They define what blocks appear where on the workspace. The file size is negligible: a file filled with hundreds of blocks was just over 50KB and is an unrealistic situation for regular usage.

The .cs files that contain the code for the mod are also insubstantial. The largest files in the example mod are around 5KB.

Due to the art style of the game, sprites are also of a very small size. The majority of sprites in the example mod are rounded to 1KB, though the majority are smaller than this in reality, with some being as small as 200 bytes.

The various text files for descriptions are also less than 1KB.

If the user were to create a mod containing 100 items, which is a fairly large number considering the target users for this project, the amount of space it takes up could be estimated.

- The template tModLoader creates is roughly 6KB.
- Assuming each item contains roughly 20 blocks in its code, then the Blockly file for each item would be around 3KB. The corresponding C# code is about another 3KB.
- Each sprite could be up to about 1KB.
- This makes each item take up about 7KB. 100 of these would be 700KB.

This leaves the total size at around 0.7MB, an almost negligible file size. It is fair to conclude that storage will not be an issue for this project.



## Algorithm design

Mod
<ul style="list-style-type: none"><li>-items: item[]</li><li>-name: string</li><li>-author: string</li><li>-description: string</li><li>-modPath: string</li></ul>
<ul style="list-style-type: none"><li>+load_item(int:itemID):void</li><li>+add_item(string:name):void</li><li>+delete_item(int:itemID):void</li><li>+get_details():string</li><li>+get_items_for_display():string[,]</li><li>+get_name():string</li></ul>

Item
<ul style="list-style-type: none"><li>-name: string</li><li>-type: string</li><li>-displayName: string</li><li>-tooltip: string</li><li>-code: code</li><li>-sprite: sprite</li></ul>
<ul style="list-style-type: none"><li>+write_item_to_file():void</li><li>+get_name():string</li><li>+get_type():string</li><li>+get_details():string</li><li>+set_name(string:name):void</li><li>+set_display_name(string:name):void</li><li>+set_tooltip(string:tooltip):void</li><li>+set_sprite(string:sprite):void</li><li>+set_code(Code:code):void</li></ul>

Sprite
<ul style="list-style-type: none"><li>-imageLocation: string</li><li>-isAnimated: bool</li><li>-frameTime: int</li><li>-sizeX: int</li><li>-sizeY: int</li></ul>
<ul style="list-style-type: none"><li>+get_image_location():string</li><li>+get_is_animated():bool</li><li>+get_frametime():int</li><li>+get_sizeX():int</li><li>+get_sizeY():int</li><li>+set_item_location(string:path):void</li><li>+set_animation(bool:isAnimated, int:frameTime):void</li><li>+set_dimensions(int:sizeX, int:sizeY):void</li></ul>

Code
<ul style="list-style-type: none"><li>-saveLocation: string</li><li>-exportedCode: string</li></ul>
<ul style="list-style-type: none"><li>+get_save_location():string</li><li>+get_exported_code():string</li><li>+export_code():void</li><li>+load_code_to_blockly():void</li><li>+set_code_location(string:path):void</li></ul>

## Mod class

The mod class will be the container for the items in the mod, as well as the properties of the project like name and author. The items are stored in a list of items, which the mod class will be used to add and remove items from, as well as load them to be displayed.

## Item class

The item class will contain one item in the mod, specifically its properties, sprites, and code. Its methods will mostly be getters and setters, with an additional function to save the item to a file. Each item will contain a code and a sprite object.

## Sprite class

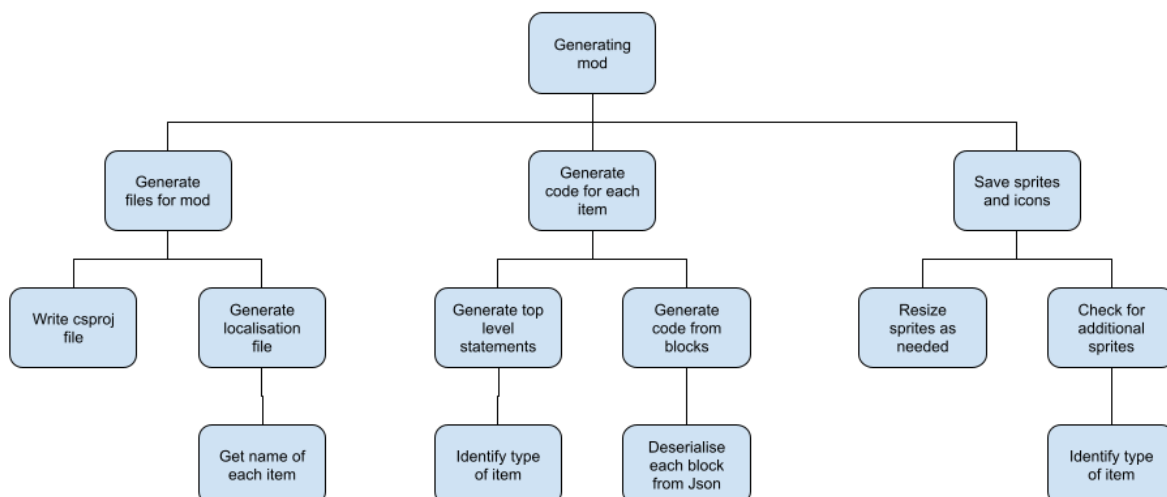
The sprite class manages the sprite for an item and its properties. It contains a reference to the sprite's location, and its height and width. It also contains the data needed to animate the sprite, being a Boolean to mark it as animated and a value for the length of each frame. Each item will have exactly one sprite.

## Code class

The code class contains the code for an item. It will contain the method that will generate the code for the item. Additionally, it will handle communication with Blockly, being able to send and receive the code from the WebView2 component.

## Export and code generation algorithm

I have opted to use a hierarchy diagram to express the functionality of the export and code generation algorithm. This is because I am currently unsure as to how many blocks will exist in the program, and there is not any real value in writing pseudocode for a code translator in detail, as it is not logically complex. Additionally, much of the process of generating the mod is writing data to files, which is not particularly useful to express as pseudocode.



This diagram outlines the process of generating the mod. The bulk of the processes involve writing files. The code generation will make use of Json deserialization to convert the Blockly workspace into something easily handled.

I have written pseudocode for the algorithm that will break the Blockly code into individual blocks as Json arrays. This is going to be one of the most important algorithms, as it forms the basis for the code translator.

```
function find_blocks(string contents)

    blocklist = {}
    reader = ""
    reading = false
    count = 0

    for i = 0 to contents.length

        reader = reader + contents[i]

        if reader.contains("\"block\":") then
            reader = ""
            reading = true
        else if reader.contains(",") AND reading == true then
            reading = false
            blocklist.add(reader)
        endif

    endfor

    for i = 0 to contents.length

        reader = reader + contents[i]

        if reader.contains("\"fields\":{\"") then
            reader = ""
            reading = true
        else if reader.contains("}") then
            blockslist[count] = blocklist[count] + reader
            reading = false
            count = count + 1
        endif

    endfor

endfunction
```

The first pass identifies each block in the code and adds them to the list. The second pass identifies the fields for each block and adds them to the block in the list. This creates a Json array for each block, that the program can make use of to generate the code from and easily access the variables.

## Example code generation function

While not particularly complex, the code generator is an integral part of the project. It will get values from the block and concatenate them into a string with the appropriate formatting. The following is an example to highlight how this process could look.

```
function convert_to_code(block)
  damage <- block.getFieldValue(damage)
  melee <- block.getFieldValue(melee)
  width <- block.getFieldValue(width)
  height <- block.getFieldValue(height)
  useTime <- block.getFieldValue(useTime)
  useAnimation <- block.getFieldValue(useAnimation)
  useStyle <- block.getFieldValue(useStyle)
  knockback <- block.getFieldValue(knockback)
  value <- block.getFieldValue(value)
  rarity <- block.getFieldValue(rarity)
  UseSound <- block.getFieldValue(UseSound)
  autoReuse <- block.getFieldValue(autoReuse)
  code <- ""
  code <- code + "item.damage = " + damage + ";\n"
  code <- code + "item.melee = " + melee + ";\n"
  code <- code + "item.width = " + width + ";\n"
  code <- code + "item.height = " + height + ";\n"
  code <- code + "item.useTime = " + useTime + ";\n"
  code <- code + "item.useAnimation = " + useAnimation + ";\n"
  code <- code + "item.useStyle = " + useStyle + ";\n"
  code <- code + "item.knockback = " + knockback + ";\n"
  code <- code + "item.value = " + value + ";\n"
  code <- code + "item.rare = " + rarity + ";\n"
  code <- code + "item.UseSound = " + UseSound + ";\n"
  code <- code + "item.autoReuse = " + autoReuse + ";\n"
  return code
end function
```

## Example block code

I have created the JSON array for the core weapon block, making use of Blockly block factory.

```
{
  "type": "define_weapon_essential",
  "message0": "Damage: %1 %2 Melee: %3 %4 Width: %5 Height: %6 %7 Use Time: %8 %9 Animation Time: %10 %11 Use Style: %12 %13 Knockback: %14 %15 Value %16 %17 Rarity %18 %19 Sound ID %20 %21 Auto Reuse %22",
  "args0": [
    {
```

```

    "type": "field_number",
    "name": "damage",
    "value": 0,
    "min": 0
  },
  {
    "type": "input_dummy"
  },
  {
    "type": "field_checkbox",
    "name": "melee",
    "checked": true
  },
  {
    "type": "input_dummy"
  },
  {
    "type": "field_number",
    "name": "width",
    "value": 0,
    "min": 0
  },
  {
    "type": "field_number",
    "name": "height",
    "value": 0,
    "min": 0
  },
  {
    "type": "input_dummy"
  },
  {
    "type": "field_number",
    "name": "useTime",
    "value": 0
  },
  {
    "type": "input_dummy"
  },
  {
    "type": "field_number",
    "name": "useAnimation",
    "value": 0
  },
  {
    "type": "input_dummy"
  },
  {
    "type": "field_dropdown",
    "name": "useStyle",

```

```
"options": [  
  [  
    "None",  
    "0"  
  ],  
  [  
    "Swing",  
    "1"  
  ],  
  [  
    "DrinkOld",  
    "2"  
  ],  
  [  
    "Thrust",  
    "3"  
  ],  
  [  
    "HoldUp",  
    "4"  
  ],  
  [  
    "Shoot",  
    "5"  
  ],  
  [  
    "DrinkLong",  
    "6"  
  ],  
  [  
    "EatFood",  
    "7"  
  ],  
  [  
    "GolfPlay",  
    "8"  
  ],  
  [  
    "DrinkLiquid",  
    "9"  
  ],  
  [  
    "HiddenAnimation",  
    "10"  
  ],  
  [  
    "MowTheLaw",  
    "11"  
  ],  
  [  

```

```

        "Guitar",
        "12"
    ],
    [
        "Rapier",
        "13"
    ],
    [
        "RaiseLamp",
        "14"
    ]
]
},
{
    "type": "input_dummy"
},
{
    "type": "field_number",
    "name": "knockback",
    "value": 0
},
{
    "type": "input_dummy"
},
{
    "type": "field_number",
    "name": "value",
    "value": 0
},
{
    "type": "input_dummy"
},
{
    "type": "field_dropdown",
    "name": "rare",
    "options": [
        [
            "Grey",
            "-1"
        ],
        [
            "White",
            "0"
        ],
        [
            "Blue",
            "1"
        ],
        [
            "Green",

```

```
    "2"
  ],
  [
    "Orange",
    "3"
  ],
  [
    "Light Red",
    "4"
  ],
  [
    "Pink",
    "5"
  ],
  [
    "Light Purple",
    "6"
  ],
  [
    "Lime",
    "7"
  ],
  [
    "Yellow",
    "8"
  ],
  [
    "Cyan",
    "9"
  ],
  [
    "Red",
    "10"
  ],
  [
    "Purple",
    "11"
  ],
  [
    "Rainbow",
    "-12"
  ],
  [
    "Fiery Red",
    "-13"
  ],
  [
    "Amber",
    "-11"
  ]
]
```



```

    ],
    {
      "type": "input_dummy"
    },
    {
      "type": "field_number",
      "name": "UseSound",
      "value": 0,
      "min": 0,
      "max": 172
    },
    {
      "type": "input_dummy"
    },
    {
      "type": "field_checkbox",
      "name": "autoReuse",
      "checked": true
    }
  ],
  "nextStatement": null,
  "colour": 230,
  "tooltip": "Define a basic weapon",
  "helpUrl": "no"
}

```

This Json array defines the fields for the block, as well as setting the default state for them and validation. These will be referenced in the toolbox, which is where the user can select blocks from different categories.

## Test Strategy

In terms of testing, it will fall into two clear parts. I will have to ensure that the C# program is stable, and well validated, and ensure that the code generated by block is fully functional.

Testing the C# component will be about ensuring the user cannot crash the program. As most of the values entered will be strings, validation may not be as important. Something very important will be the way the program handles incorrect files being placed in the folder.

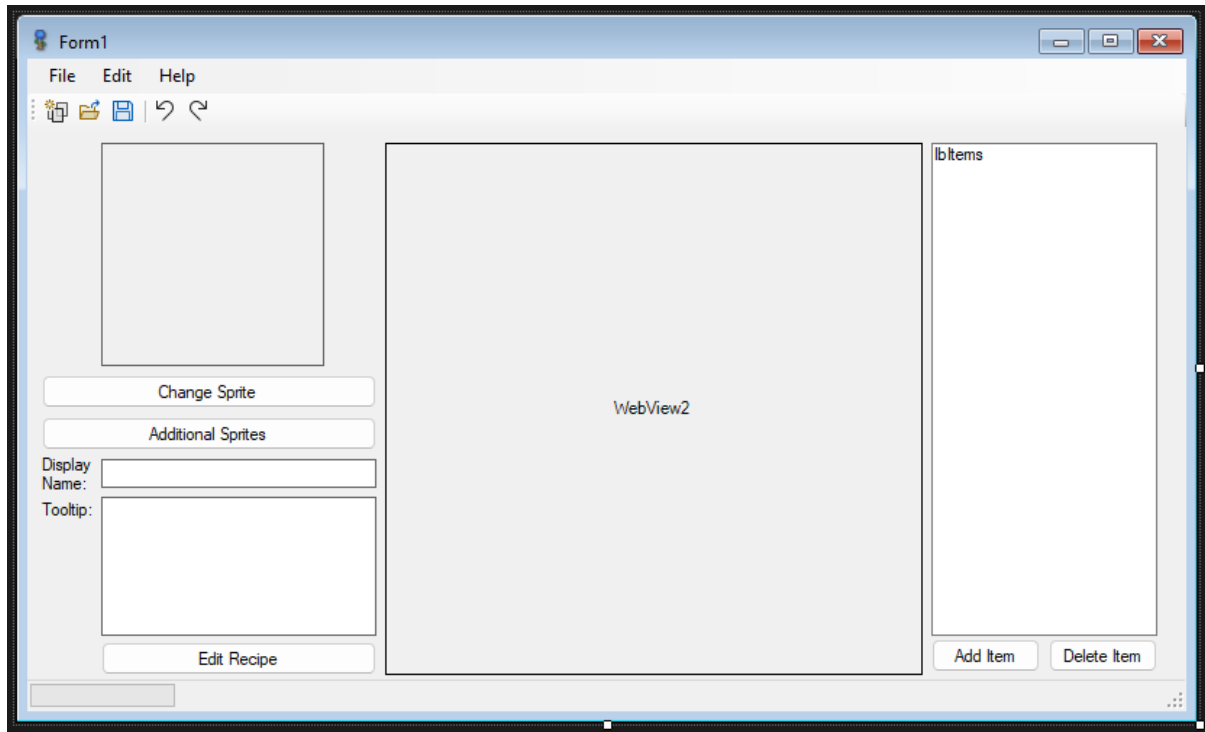
For the Blockly component, the goal is to meet objective 3. This means that the fields in the blocks will have to be strictly validated. As my tool is aimed at new users, I cannot rely on them knowing what inputs are expected in the same as you could of an experienced programmer. As ease of use is a goal of the project (objective 1), it must also be clear when and why an input is wrong.

## Technical solution

This is the full code listing for the project. It is listed as relevant.

## C# Solution

### Main form – Main.cs



The main form handles the bulk of the user's interaction with the program, as well as saving, loading, and exporting the project.

While code is generated in the *CodeGenerator* class, the file handling for the export process happens here. The first part of the export process is to ensure the output will be valid. It ensures that all sprites are present as appropriate for the item. If not, an appropriate message is generated to explain to the user the issues.

Next, the files defining properties of the whole mod are generated. Notably, the icon must be resized to be exactly 80 by 80 pixels, as this is the only size tModLoader will accept. The localisation file, which defines names and tooltips for a language, must also be generated here. I made the decision to always generate the English file, as my project does not support other languages. Additionally, UI design would become challenging to accommodate multiple inputs for each piece of text.

Each item in the mod then has code generated for it, using the *CodeGenerator* class. Then, each possible sprite is checked for and saved if present. Finally, the user is prompted to view the mod in tModLoader.

An important part of the main form's design is the locking of certain controls. As much of the program is asynchronous, it could be possible to make inputs while other processes are running. As such, the controls must be locked at certain times when an input could cause incorrect data to

be written. While the main feature that must be disabled is saving, all data enter fields are disabled to prevent any unpredictable behaviour.

```
1. using Microsoft.Web.WebView2.Core;
2. using System;
3. using System.Collections.Generic;
4. using System.ComponentModel;
5. using System.Data;
6. using System.Diagnostics;
7. using System.Drawing;
8. using System.Drawing.Drawing2D;
9. using System.Drawing.Imaging;
10. using System.IO;
11. using System.Linq;
12. using System.Net.Http.Headers;
13. using System.Reflection;
14. using System.Reflection.Emit;
15. using System.Security.Permissions;
16. using System.Text;
17. using System.Threading.Tasks;
18. using System.Windows.Forms;
19. using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;
20. using static System.Windows.Forms.VisualStyles.VisualStyleElement.Tab;
21.
22. //This program is confirmed working for 1.4.4.9
23. namespace NEA_solution
24. {
25.     public partial class Main : Form
26.     {
27.         Mod loadedMod;
28.         Item loadedItem;
29.         bool hasExportPath;
30.         string workspace;
31.         Item theItem;
32.         bool returned;
33.         bool saveReturned;
34.         bool wvready = false;
35.         bool wvSaveReady = false;
36.
37.         public Main()
38.         {
39.             InitializeComponent();
40.             Text = "tModMaker";
41.
42.             loadedMod = new Mod("", "");
43.
44.             //This checks to seem if the user has set a path to export to.
45.             if (File.ReadAllText(Environment.CurrentDirectory + "\\userConfig.txt") != "")
46.             {
47.                 hasExportPath = true;
48.             }
49.             else { hasExportPath = false; }
50.
51.             initialise_editor();
52.             load_recents();
53.             lock_controls();
54.         }
55.
56.         private void load_recents()
57.         {
58.             string[] recents = File.ReadAllLines(Environment.CurrentDirectory +
"\\recents.txt");
59.             openRecentToolStripMenuItem.DropDownItems.Clear();
60.             //Each recent project is added to the dropdown.
```

```

61.         for (int i = 0; i < recents.Length; i++)
62.         {
63.             openRecentToolStripMenuItem.DropDownItems.Add(recents[i]);
64.             openRecentToolStripMenuItem.DropDownItems[i].Click += recent_click;
65.         }
66.     }
67.
68.     private void recent_click(object sender, EventArgs e)
69.     {
70.         //The name of the sender is the path of the mod, so it can be opened like this.
71.         open_mod(sender.ToString());
72.     }
73.
74.     private void initialise_editor()
75.     {
76.         //The webView component containing the editor is set up.
77.         InitWebView();
78.         wwCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
Editor\\start.html");
79.     }
80.
81.     private void btnAddItem_Click(object sender, EventArgs e)
82.     {
83.         //A dialog for the users to enter the details of the mod is opened.
84.         CreateItemDialog createItemDialog = new CreateItemDialog();
85.         DialogResult result = createItemDialog.ShowDialog();
86.         if (result == DialogResult.OK)
87.         {
88.             loadedMod.add_item(createItemDialog.newItem);
89.
90.             //The list of items is updated to contain the new item.
91.             update_item_list();
92.             loadedItem = loadedMod.get_item(loadedMod.get_item_number() - 1);
93.             displayItem(loadedItem);
94.         }
95.     }
96.
97.     private void update_item_list()
98.     {
99.         //Each name is added to the list box.
100.        lbItems.Items.Clear();
101.        string[,] displayText = loadedMod.get_items_for_display();
102.        for (int i = 0; i < displayText.GetLength(0); i++)
103.        {
104.            lbItems.Items.Add(displayText[i, 0]);
105.        }
106.    }
107.
108.    private async void update_loaded_item(int index)
109.    {
110.        if (loadedItem != null)
111.        {
112.            /*
113.             * The save data of the load item is requested before it is changed.
114.             * This means that when the save button is pressed, all items have
115.             * their updated code written to file.
116.             */
117.            requestData();
118.            returned = false;
119.            do
120.            {
121.                await Task.Delay(100);
122.            }
123.            while (returned == false);
124.

```

```

125.         loadedItem.set_code(workspace);
126.     }
127.     loadedItem = loadedMod.get_item(index);
128.     displayItem(loadedItem);
129.     update_item_list();
130.
131. }
132.
133.
134. private void fileSaveModAs_Click(object sender, EventArgs e)
135. {
136.     save_mod_as();
137. }
138.
139. private void fileSaveMod_Click(object sender, EventArgs e)
140. {
141.     save_mod();
142. }
143.
144. private void save_mod_as()
145. {
146.     NameDialog nameDialog = new NameDialog();
147.     DialogResult result = nameDialog.ShowDialog();
148.     if (result == DialogResult.OK)
149.     {
150.         loadedMod.set_name(nameDialog.name);
151.     }
152.     else
153.     {
154.         //When save mod is called, the controls are locked, so they must be unlocked
here if the mod is not going to be saved.
155.         unlock_controls();
156.
157.         return;
158.     }
159.     FolderBrowserDialog dialog = new FolderBrowserDialog();
160.     DialogResult dialogResult = dialog.ShowDialog();
161.     if (dialogResult == DialogResult.OK)
162.     {
163.         loadedMod.set_modPath(dialog.SelectedPath + "\\\" + loadedMod.get_name());
164.
165.         //Once the path and name are specified, it is saved as normal.
166.         add_recent_path(loadedMod.get_modPath());
167.         save_mod();
168.     }
169.     else
170.     {
171.         unlock_controls();
172.
173.         return;
174.     }
175.     unlock_controls();
176.     Text = "tModMaker - " + loadedMod.get_name();
177. }
178.
179. //This procedure is asynchronous as it waits to recieve the data from the web
component.
180. private async Task save_mod()
181. {
182.     lock_controls();
183.     //The properties of the progress bar are set up.
184.     pbSave.Step = 1;
185.     pbSave.Minimum = 1;
186.     if (loadedMod.get_item_number() > 0)
187.     {

```

```

188.         pbSave.Maximum = loadedMod.get_item_number() * 2;
189.     }
190.     else
191.     {
192.         pbSave.Maximum = 1;
193.     }
194.     pbSave.Value = 1;
195.     //If an item is loaded, it is saved.
196.     if (txtDisplayName.Text != "")
197.     {
198.         theItem.set_display_name(txtDisplayName.Text);
199.     }
200.     if (txtTooltip.Text != "")
201.     {
202.         theItem.set_tooltip(txtTooltip.Text);
203.     }
204.     if (loadedItem != null)
205.     {
206.         //The program must wait here to recieve the code.
207.         requestData();
208.         returned = false;
209.         do
210.         {
211.             await Task.Delay(100);
212.         }
213.         while (returned == false);
214.         loadedItem.set_code(workspace);
215.     }
216.     for (int i = 0; i < loadedMod.get_item_number(); i++)
217.     {
218.         pbSave.PerformStep();
219.     }
220.
221.
222.
223.     string thePath = loadedMod.get_modPath();
224.     string modFile = "";
225.     string tempItem;
226.     string recipeItems;
227.
228.     /*
229.     * The details of the mod are compiled into one string, separated with pipes,
230.     * as it is a fairly uncommon character, and is unlikely to appear in the mod's
231.     * details.
232.     */
233.     modFile += loadedMod.get_name() + "|";
234.     modFile += loadedMod.get_description() + "|";
235.     modFile += loadedMod.get_author() + "|";
236.     modFile += loadedMod.get_version();
237.
238.     /*
239.     * The progress bar is set up to make a step for each item in the mod.
240.     * It is worth noting that the progress bar doesn't appear to work on
241.     * windows 11 - this issue is inconsistent, and seems to be happening
242.     * less
243.     */
244.
245.     /*
246.     * If the mod is missing a path i.e. the placeholder is loaded, the save_mod_as()
247.     * then returns nothing for this procedure.
248.     */
249.     if (thePath == "")
250.
is called,
251.
252.
253.

```

```

254.         {
255.             save_mod_as();
256.             return;
257.         }
258.
259.         Directory.CreateDirectory(thePath);
260.
261.         //The details of the mod are written to a file here.
262.         File.WriteAllText(thePath + "\\\" + loadedMod.get_name() + ".mod", modFile);
263.
264.         Bitmap icon = loadedMod.get_icon();
265.         if (icon != null)
266.         {
267.             icon.Save(thePath + "\\\" + loadedMod.get_name() + ".png", ImageFormat.Png);
268.         }
269.
270.         //This checks for and creates other directories that need to exist.
271.         if (!Directory.Exists(thePath + "\\Items"))
272.         {
273.             Directory.CreateDirectory(thePath + "\\Items");
274.         }
275.         if (!Directory.Exists(thePath + "\\Items\\Code"))
276.         {
277.             Directory.CreateDirectory(thePath + "\\Items\\Code");
278.         }
279.         if (!Directory.Exists(thePath + "\\Items\\Sprites"))
280.         {
281.             Directory.CreateDirectory(thePath + "\\Items\\Sprites");
282.         }
283.         if (!Directory.Exists(thePath + "\\Items\\Recipes"))
284.         {
285.             Directory.CreateDirectory(thePath + "\\Items\\Recipes");
286.         }
287.
288.         if (loadedMod.get_item_number() != 0)
289.         {
290.             //This loops through and saves each item.
291.             for (int i = 0; i < loadedMod.get_item_number(); i++)
292.             {
293.                 //The details of the item and put into a string, then saved.
294.                 tempItem = "";
295.                 tempItem += loadedMod.get_item(i).get_name() + "\r\n";
296.                 tempItem += loadedMod.get_item(i).get_displayName() + "\r\n";
297.                 tempItem += loadedMod.get_item(i).get_tooltip() + "\r\n";
298.                 tempItem += loadedMod.get_item(i).get_type();
299.                 File.WriteAllText(thePath + "\\Items\\" + loadedMod.get_item(i).get_name()
+ ".item", tempItem);
300.
301.                 //The code is written to a separate file.
302.                 File.WriteAllText(thePath + "\\Items\\Code\\" +
loadedMod.get_item(i).get_name() + "_code.code", loadedMod.get_item(i).get_code());
303.
304.                 recipeItems = loadedMod.get_item(i).get_craftingStationID() + "\r\n";
305.                 for (int j = 0; j < loadedMod.get_item(i).get_ingredients().Length; j++)
306.                 {
307.                     recipeItems += loadedMod.get_item(i).get_ingredients()[j].itemName +
"|" + loadedMod.get_item(i).get_ingredients()[j].quantity + "\r\n";
308.                 }
309.                 File.WriteAllText(thePath + "\\Items\\Recipes\\" +
loadedMod.get_item(i).get_name() + "_recipe.recipe", recipeItems);
310.
311.                 Bitmap bmp = loadedMod.get_item(i).get_sprite();
312.
313.                 //If a sprite exists for the item, it is saved as a .png.
314.                 if (bmp != null)

```

```

315.         {
316.             bmp.Save(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + ".png", ImageFormat.Png);
317.         }
318.
319.         bmp = loadedMod.get_item(i).get_wingSprite();
320.         File.Delete(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Wings.png");
321.         if (bmp != null)
322.         {
323.             bmp.Save(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Wings.png", ImageFormat.Png);
324.         }
325.
326.         string itemType = GetTypeOfItem(loadedMod.get_item(i));
327.
328.         bmp = loadedMod.get_item(i).get_headSprite();
329.         File.Delete(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Head.png");
330.         if (bmp != null && itemType == "head")
331.         {
332.             bmp.Save(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Head.png", ImageFormat.Png);
333.         }
334.
335.         bmp = loadedMod.get_item(i).get_bodySprite();
336.         File.Delete(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Body.png");
337.         if (bmp != null && itemType == "body")
338.         {
339.             bmp.Save(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Body.png", ImageFormat.Png);
340.         }
341.
342.         bmp = loadedMod.get_item(i).get_legsSprite();
343.         File.Delete(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Legs.png");
344.         if (bmp != null && itemType == "legs")
345.         {
346.             bmp.Save(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_Legs.png", ImageFormat.Png);
347.         }
348.
349.         bmp = loadedMod.get_item(i).get_mapHead();
350.         File.Delete(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_MapHead.png");
351.         if (bmp != null && itemType == "boss")
352.         {
353.             bmp.Save(thePath + "\\Items\\Sprites\\" +
loadedMod.get_item(i).get_name() + "_MapHead.png", ImageFormat.Png);
354.         }
355.
356.         //The step for the progress bar is performed.
357.         pbSave.PerformStep();
358.     }
359.     //This delay is purely visual, but I found the user experience was much better
as a result.
360.     await Task.Delay(500);
361.     pbSave.Value = 1;
362.     tbSave.Enabled = true;
363.     fileSaveMod.Enabled = true;
364.     wvCode.Enabled = true;
365.     unlock_controls();
366. }
367. }

```



```

368.
369.     private void fileOpenMod_Click(object sender, EventArgs e)
370.     {
371.         open_mod();
372.     }
373.     private void load_items_for_mod()
374.     {
375.         string[] existingItems;
376.         string[] existingCode;
377.         string[] existingSprites;
378.         string[] existingRecipes;
379.         string[] recipeItems;
380.         List<RecipeItem> tmpRecipe;
381.         Item currentItem;
382.         string[] tmpProperties;
383.         {
384.             /*
385.              * This ensures the correct structure exists in the specified directory.
386.              *
387.              * This isn't foolproof, but it is unlikely someone would try to open another
388.              * just happened to have the same structure, without doing it on purpose.
389.              */
390.             if (Directory.Exists(loadedMod.get_modPath() + "\\Items")
391.                 && Directory.Exists(loadedMod.get_modPath() + "\\Items\\Code")
392.                 && Directory.Exists(loadedMod.get_modPath() + "\\Items\\Sprites")
393.                 && Directory.Exists(loadedMod.get_modPath() + "\\Items\\Recipes"))
394.             {
395.                 //If it does, arrays of paths are created for each item, sprite, and code.
396.                 existingItems = Directory.GetFiles(loadedMod.get_modPath() + "\\Items");
397.                 existingCode = Directory.GetFiles(loadedMod.get_modPath() +
398. "\\Items\\Code");
399.                 existingSprites = Directory.GetFiles(loadedMod.get_modPath() +
400. "\\Items\\Sprites");
401.                 existingRecipes = Directory.GetFiles(loadedMod.get_modPath() +
402. "\\Items\\Recipes");
403.                 //If there are no items in the mod, an empty workspace is loaded.
404.                 if (existingItems.Length == 0)
405.                 {
406.                     wvCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
407. Editor\\start.html");
408.                     lbItems.Items.Clear();
409.                     return;
410.                 }
411.                 else
412.                 {
413.                     lbItems.Items.Clear();
414.                     for (int i = 0; i < existingItems.Length; i++)
415.                     {
416.                         tmpProperties = File.ReadAllLines(existingItems[i]);
417.                         //The properties in the array created are used to assemble the
418.                         item.
419.                         currentItem = new Item(tmpProperties[0], tmpProperties[3]);
420.                         currentItem.set_display_name(tmpProperties[1]);
421.                         currentItem.set_tooltip(tmpProperties[2]);
422.                         currentItem.set_code(File.ReadAllText(existingCode[i]));
423.                         /*
424.                          * Sprites are assigned to their item by sharing a file name,
425.                          * so this checks if a sprite corresponds to the item that has

```

```

426.             for (int j = 0; j < existingSprites.Length; j++)
427.             {
428.                 if (loadedMod.get_modPath() + "\\Items\\Sprites\\" +
currentItem.get_name() + ".png" == existingSprites[j])
429.                 {
430.                     FileStream fileHandler = File.Open(existingSprites[j],
FileMode.Open);
431.                     currentItem.set_sprite(new Bitmap(fileHandler));
432.                     fileHandler.Close();
433.                 }
434.                 else if (loadedMod.get_modPath() + "\\Items\\Sprites\\" +
currentItem.get_name() + "_Wings.png" == existingSprites[j])
435.                 {
436.                     FileStream fileHandler = File.Open(existingSprites[j],
FileMode.Open);
437.                     currentItem.set_wingSprite(new Bitmap(fileHandler));
438.                     fileHandler.Close();
439.                 }
440.                 else if (loadedMod.get_modPath() + "\\Items\\Sprites\\" +
currentItem.get_name() + "_Head.png" == existingSprites[j])
441.                 {
442.                     FileStream fileHandler = File.Open(existingSprites[j],
FileMode.Open);
443.                     currentItem.set_headSprite(new Bitmap(fileHandler));
444.                     fileHandler.Close();
445.                 }
446.                 else if (loadedMod.get_modPath() + "\\Items\\Sprites\\" +
currentItem.get_name() + "_Body.png" == existingSprites[j])
447.                 {
448.                     FileStream fileHandler = File.Open(existingSprites[j],
FileMode.Open);
449.                     currentItem.set_bodySprite(new Bitmap(fileHandler));
450.                     fileHandler.Close();
451.                 }
452.                 else if (loadedMod.get_modPath() + "\\Items\\Sprites\\" +
currentItem.get_name() + "_Legs.png" == existingSprites[j])
453.                 {
454.                     FileStream fileHandler = File.Open(existingSprites[j],
FileMode.Open);
455.                     currentItem.set_legsSprite(new Bitmap(fileHandler));
456.                     fileHandler.Close();
457.                 }
458.                 else if (loadedMod.get_modPath() + "\\Items\\Sprites\\" +
currentItem.get_name() + "_MapHead.png" == existingSprites[j])
459.                 {
460.                     FileStream fileHandler = File.Open(existingSprites[j],
FileMode.Open);
461.                     currentItem.set_mapHead(new Bitmap(fileHandler));
462.                     fileHandler.Close();
463.                 }
464.             }
465.
466.             //If the item has a recipe, it is assigned here.
467.             for (int j = 0; j < existingRecipes.Length; j++)
468.             {
469.                 if (loadedMod.get_modPath() + "\\Items\\Recipes\\" +
currentItem.get_name() + "_recipe.recipe" == existingRecipes[j])
470.                 {
471.                     tmpRecipe = new List<RecipeItem>();
472.                     recipeItems = File.ReadAllLines(existingRecipes[j]);
473.
currentItem.set_craftingStationID(Convert.ToInt32(recipeItems[0]));
474.                     for (int k = 1; k < recipeItems.Length; k++)
475.                     {

```

```

476.                                tmpRecipe.Add(new
RecipeItem(recipeItems[k].Split('|')[0], Convert.ToInt32(recipeItems[k].Split('|')[1])));
477.                                }
478.                                currentItem.set_ingredients(tmpRecipe.ToArray());
479.                            }
480.                        }
481.                        //The item is added to the list, then the displayed list is
updated.
482.                        loadedMod.add_item(currentItem);
483.                        update_item_list();
484.                        loadedItem = loadedMod.get_item(0);
485.                        displayItem(loadedItem);
486.                    }
487.                }
488.            }
489.            //This is reached if the directory does not have the right structure.
490.            else
491.            {
492.                MessageBox.Show("Please select a valid folder");
493.            }
494.        }
495.    }
496.
497.    private void modDetailsToolStripMenuItem_Click(object sender, EventArgs e)
498.    {
499.        //This opens a dialog where the user can edit the details of the mod.
500.        EditDetailsDialog editDetailsDialog = new EditDetailsDialog(loadedMod.get_name(),
loadedMod.get_author(), loadedMod.get_description(), loadedMod.get_version(), loadedMod.get_icon());
501.        editDetailsDialog.ShowDialog();
502.        loadedMod.set_name(editDetailsDialog.name);
503.        loadedMod.set_author(editDetailsDialog.author);
504.        loadedMod.set_description(editDetailsDialog.description);
505.        loadedMod.set_version(editDetailsDialog.version);
506.        loadedMod.set_icon(editDetailsDialog.icon);
507.        Text = loadedMod.get_name();
508.    }
509.
510.    private void btnDeleteItem_Click(object sender, EventArgs e)
511.    {
512.        if (lbItems.SelectedIndex != -1)
513.        {
514.            //A confirmation dialog will appear.
515.            DialogResult result = MessageBox.Show("Are you sure?", "Confirm action",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
516.            if (result == DialogResult.Yes)
517.            {
518.                //This ensures they have selected a valid item to delete.
519.
520.                /*
521.                 * A list one shorter than the current one is created,
522.                 * and every item except the deleted one is written to it.
523.                 */
524.                Item[] tmpItems = new Item[loadedMod.get_item_number() - 1];
525.                int indexToDelete = lbItems.SelectedIndex;
526.                int count = 0;
527.                for (int i = 0; i < loadedMod.get_item_number(); i++)
528.                {
529.                    if (i != indexToDelete)
530.                    {
531.                        tmpItems[count] = loadedMod.get_item(i);
532.                        count++;
533.                    }
534.                }
535.                //The files for the item are deleted, and the list of items is
overwritten.

```

```

536.             if (File.Exists(loaderMod.get_modPath() + "\\Items\\" +
loaderMod.get_item(indexToDelete).get_name() + ".item"))
537.             {
538.                 File.Delete(loaderMod.get_modPath() + "\\Items\\" +
loaderMod.get_item(indexToDelete).get_name() + ".item");
539.             }
540.             if (File.Exists(loaderMod.get_modPath() + "\\Items\\Code\\" +
loaderMod.get_item(indexToDelete).get_name() + "_code.code"))
541.             {
542.                 File.Delete(loaderMod.get_modPath() + "\\Items\\Code\\" +
loaderMod.get_item(indexToDelete).get_name() + "_code.code");
543.             }
544.             if (File.Exists(loaderMod.get_modPath() + "\\Items\\Recipes\\" +
loaderMod.get_item(indexToDelete).get_name() + "_recipe.code"))
545.             {
546.                 File.Delete(loaderMod.get_modPath() + "\\Items\\Recipes\\" +
loaderMod.get_item(indexToDelete).get_name() + "_recipe.code");
547.             }
548.             loaderMod.set_items(tmpItems);
549.             loaderItem = null;
550.             displayItem(new Item("", ""));
551.             wvCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
Editor\\start.html");
552.             update_item_list();
553.         }
554.     }
555. }
556.
557. private void newToolStripMenuItem_Click(object sender, EventArgs e)
558. {
559.     new_project();
560. }
561.
562. private void tbNew_Click(object sender, EventArgs e)
563. {
564.     new_project();
565. }
566.
567. private void new_project()
568. {
569.     //A confirmation dialog will appear.
570.     DialogResult result = MessageBox.Show("Are you sure? Unsaved work will be lost.",
"Confirm action", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
571.     if (result == DialogResult.OK)
572.     {
573.         //A blank mod is created.
574.         loaderMod = new Mod("", "");
575.         displayItem(new Item("", ""));
576.         clearBlockly();
577.         lock_controls();
578.
579.         //The ui is reset.
580.         update_item_list();
581.         update_loaded_item(-1);
582.     }
583. }
584.
585. private void tbOpen_Click(object sender, EventArgs e)
586. {
587.     open_mod();
588. }
589.
590. private void open_mod()
591. {
592.     Mod theMod;

```

```

593.         string modDetails;
594.         string[] modDetailsSplit;
595.         string[] existingFiles;
596.
597.         FolderBrowserDialog folderDialog = new FolderBrowserDialog();
598.         DialogResult result = folderDialog.ShowDialog();
599.         if (result == DialogResult.OK)
600.         {
601.             existingFiles = Directory.GetFiles(folderDialog.SelectedPath);
602.
603.             //This ensures the directory is the correct structure.
604.             if (!(existingFiles.Length == 2 || existingFiles.Length == 1))
605.             {
606.                 MessageBox.Show("Please select a valid folder");
607.                 return;
608.             }
609.             else
610.             {
611.                 //The details of the mod are loaded for the files.
612.                 if (existingFiles[0].Contains(".mod"))
613.                 {
614.                     modDetails = File.ReadAllText(existingFiles[0]);
615.                 }
616.                 else
617.                 {
618.                     modDetails = File.ReadAllText(existingFiles[1]);
619.                 }
620.                 modDetailsSplit = modDetails.Split('|');
621.                 theMod = new Mod(modDetailsSplit[0], folderDialog.SelectedPath);
622.                 theMod.set_description(modDetailsSplit[1]);
623.                 theMod.set_author(modDetailsSplit[2]);
624.                 theMod.set_version(Convert.ToDouble(modDetailsSplit[3]));
625.                 if (existingFiles[0].Contains(".png"))
626.                 {
627.                     FileStream fileHandler = File.Open(existingFiles[0], FileMode.Open);
628.                     theMod.set_icon(new Bitmap(fileHandler));
629.                     fileHandler.Close();
630.                 }
631.                 else
632.                 {
633.                     try
634.                     {
635.                         FileStream fileHandler = File.Open(existingFiles[1],
FileMode.Open);
636.                         theMod.set_icon(new Bitmap(fileHandler));
637.                         fileHandler.Close();
638.                     }
639.                     catch { }
640.                 }
641.                 loadedMod = theMod;
642.                 Text = "tModMaker - " + loadedMod.get_name();
643.                 displayItem(new Item("", ""));
644.                 clearBlockly();
645.                 lock_controls();
646.             }
647.             //The procedure to load the items is called.
648.             load_items_for_mod();
649.             add_recent_path(folderDialog.SelectedPath);
650.         }
651.     }
652.
653.     private void open_mod(string path)
654.     {
655.         Mod theMod;
656.         string modDetails;

```

```

657.         string[] modDetailsSplit;
658.         string[] existingFiles;
659.         try
660.         {
661.             existingFiles = Directory.GetFiles(path);
662.
663.
664.             //This ensures the directory is the correct structure.
665.             if (!(existingFiles.Length == 2 || existingFiles.Length == 1))
666.             {
667.                 MessageBox.Show("Please select a valid folder");
668.                 return;
669.             }
670.             else
671.             {
672.                 //The details of the mod are loaded for the files.
673.                 if (existingFiles[0].Contains(".mod"))
674.                 {
675.                     modDetails = File.ReadAllText(existingFiles[0]);
676.                 }
677.                 else
678.                 {
679.                     modDetails = File.ReadAllText(existingFiles[1]);
680.                 }
681.                 modDetailsSplit = modDetails.Split('|');
682.                 theMod = new Mod(modDetailsSplit[0], path);
683.                 theMod.set_description(modDetailsSplit[1]);
684.                 theMod.set_author(modDetailsSplit[2]);
685.                 theMod.set_version(Convert.ToDouble(modDetailsSplit[3]));
686.                 if (existingFiles[0].Contains(".png"))
687.                 {
688.                     FileStream fileHandler = File.Open(existingFiles[0], FileMode.Open);
689.                     theMod.set_icon(new Bitmap(fileHandler));
690.                     fileHandler.Close();
691.                 }
692.                 else
693.                 {
694.                     try
695.                     {
696.                         FileStream fileHandler = File.Open(existingFiles[1],
FileMode.Open);
697.                         theMod.set_icon(new Bitmap(fileHandler));
698.                         fileHandler.Close();
699.                     }
700.                     catch { }
701.                 }
702.                 loadedMod = theMod;
703.                 Text = "tModMaker - " + loadedMod.get_name();
704.                 displayItem(new Item("", ""));
705.                 clearBlockly();
706.                 lock_controls();
707.             }
708.             //The procedure to load the items is called.
709.             load_items_for_mod();
710.             add_recent_path(path);
711.         }
712.         catch (Exception e)
713.         {
714.             //This should tell the user that the path does not exist, as this is the error
this is designed to catch.
715.             MessageBox.Show(e.Message);
716.         }
717.     }
718.
719.     private void add_recent_path(string path)

```

```

720.         {
721.             string[] recents = File.ReadAllLines(Environment.CurrentDirectory +
"\\recents.txt");
722.             string pathsToWrite = path;
723.
724.             //This loop ensures there are at maximum 5 unique paths listed.
725.             int checkPaths = 0;
726.             while (checkPaths < recents.Length && checkPaths < 4)
727.             {
728.                 if (recents[checkPaths] != path)
729.                 {
730.                     pathsToWrite += "\r\n" + recents[checkPaths];
731.                 }
732.                 checkPaths++;
733.             }
734.             File.WriteAllText(Environment.CurrentDirectory + "\\recents.txt", pathsToWrite);
735.             load_recents();
736.         }
737.
738.         private void tbSave_Click(object sender, EventArgs e)
739.         {
740.             save_mod();
741.         }
742.
743.         private void lbItems_DoubleClick(object sender, EventArgs e)
744.         {
745.             if (lbItems.SelectedIndex != -1)
746.             {
747.                 update_loaded_item(lbItems.SelectedIndex);
748.             }
749.         }
750.
751.         private async void exportToolStripMenuItem_Click(object sender, EventArgs e)
752.         {
753.             CodeGenerator codeGenerator = new CodeGenerator();
754.             string path;
755.             string tmpCode;
756.             bool canExport = true;
757.             List<string> incompleteItems = new List<string>();
758.             string localizationString = "";
759.             string slot;
760.
761.             if (File.ReadAllText(Environment.CurrentDirectory + "\\userConfig.txt") != "")
762.             {
763.                 hasExportPath = true;
764.             }
765.             else { hasExportPath = false; }
766.             //This checks for the export directory being set. If not, the user is prompted to
set one.
767.
768.             if (loadedMod.get_name() != "")
769.             {
770.                 if (hasExportPath)
771.                 {
772.                     DialogResult result = MessageBox.Show("Save before exporting?", "Save?",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);
773.                     if (result == DialogResult.Yes)
774.                     {
775.                         await save_mod();
776.                     }
777.                     path = File.ReadAllText(Environment.CurrentDirectory + "\\userConfig.txt")
+ "\\\" + loadedMod.get_name();
778.                     Bitmap bmp;
779.
780.                     //An array is created containing all the items to export.

```

```

781.         Item[] itemsToExport = new Item[loadedMod.get_item_number()];
782.         for (int i = 0; i < itemsToExport.Length; i++)
783.         {
784.             itemsToExport[i] = loadedMod.get_item(i);
785.         }
786.
787.         for (int i = 0; i < itemsToExport.Length; i++)
788.         {
789.             //This ensures each item has all the sprites they require.
790.             slot = codeGenerator.get_slot(itemsToExport[i]);
791.             bmp = itemsToExport[i].get_sprite();
792.             if (bmp == null)
793.             {
794.                 incompleteItems.Add(itemsToExport[i].get_name() + " - no sprite");
795.                 canExport = false;
796.             }
797.             if (slot == "Head" && itemsToExport[i].get_headSprite() == null)
798.             {
799.                 incompleteItems.Add(itemsToExport[i].get_name() + " - no head
800. sprite");
801.                 canExport = false;
802.             }
803.             if (slot == "Body" && itemsToExport[i].get_bodySprite() == null)
804.             {
805.                 incompleteItems.Add(itemsToExport[i].get_name() + " - no body
806. sprite");
807.                 canExport = false;
808.             }
809.             if (slot == "Legs" && itemsToExport[i].get_legsSprite() == null)
810.             {
811.                 incompleteItems.Add(itemsToExport[i].get_name() + " - no legs
812. sprite");
813.                 canExport = false;
814.             }
815.             if (GetTypeOfItem(itemsToExport[i]) == "boss" &&
816. itemsToExport[i].get_mapHead() == null)
817.             {
818.                 incompleteItems.Add(itemsToExport[i].get_name() + " - no map
819. sprite");
820.                 canExport = false;
821.             }
822.         }
823.
824.         //At this point, all the checks to ensure the mod is valid have been run,
825.         so it will return if it is invalid.
826.         if (!canExport)
827.         {
828.             string tmpString = "The following items cannot be exported:\n";
829.             for (int i = 0; i < incompleteItems.Count; i++)
830.             {
831.                 tmpString += "\u2022 " + incompleteItems[i] + "\n";
832.             }
833.             tmpString += "Please ensure all items have sprites, details, and
834. code.";
835.
836.             //The user is told which item is the problem.
837.             MessageBox.Show(tmpString);
838.             return;
839.         }
840.
841.         //The necessary directories are created.
842.         if (Directory.Exists(path))
843.         {
844.             Directory.Delete(path, true);
845.         }

```



```

839.         Directory.CreateDirectory(path);
840.         Directory.CreateDirectory(path + "\\Items");
841.         Directory.CreateDirectory(path + "\\Localization");
842.         Directory.CreateDirectory(path + "\\Properties");
843.         Directory.CreateDirectory(path + "\\Projectiles");
844.         Directory.CreateDirectory(path + "\\NPCs");
845.         Directory.CreateDirectory(path + "\\Tiles");
846.
847.         Bitmap icon = loadedMod.get_icon();
848.         if (icon != null)
849.         {
850.             Bitmap exportIcon = new Bitmap(80, 80);
851.             Graphics g = Graphics.FromImage(exportIcon);
852.             g.DrawRectangle(new Pen(Color.Transparent), 0, 0, 80, 80);
853.             g.InterpolationMode = InterpolationMode.HighQualityBicubic;
854.             double picBoxWidth = 80;
855.             double picBoxHeight = 80;
856.             double height = icon.Height;
857.             double width = icon.Width;
858.             if (height > width)
859.             {
860.                 g.DrawImage(icon, (int)(picBoxWidth - (picBoxHeight / height *
width)) / 2, 0, (int)(picBoxHeight / height * width), (int)(picBoxHeight));
861.             }
862.             else if (height < width)
863.             {
864.                 g.DrawImage(icon, 0, (int)(picBoxHeight - (picBoxWidth / width *
height)) / 2, (int)picBoxWidth, (int)(picBoxWidth / width * height));
865.             }
866.             else
867.             {
868.                 g.DrawImage(icon, 0, 0, 80, 80);
869.             }
870.             exportIcon.Save(path + "\\icon.png", ImageFormat.Png);
871.         }
872.
873.         File.WriteAllText(path + "\\description.txt",
loadedMod.get_description());
875.         DialogResult messageResult = MessageBox.Show("Increment version number?",
"Increment version", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
876.         if (messageResult == DialogResult.Yes)
877.         {
878.             //It the version number is incremented, the change is saved here.
879.             loadedMod.set_version(loadedMod.get_version() + 0.1);
880.             string modFile = "";
881.             modFile += loadedMod.get_name() + "|";
882.             modFile += loadedMod.get_description() + "|";
883.             modFile += loadedMod.get_author() + "|";
884.             modFile += loadedMod.get_version();
885.
886.             File.WriteAllText(loadedMod.get_modPath() + "\\" +
loadedMod.get_name() + ".mod", modFile);
887.
888.         }
889.
890.         File.WriteAllText(path + "\\build.txt", "displayName = " +
loadedMod.get_name() + "\nauthor = " + loadedMod.get_author() + "\nversion = " +
loadedMod.get_version());
891.
892.         File.WriteAllText(path + "\\" + loadedMod.get_name() + ".cs", "using
Terraria.ModLoader;\r\n\r\nnamespace " + loadedMod.get_name() + "\r\n{\r\n\tpublic class " +
loadedMod.get_name() + " : Mod\r\n\t{\r\n\t\t\r\n\t}\r\n}");
893.

```

```

894.         File.WriteAllText(path + "\\\" + loadedMod.get_name() + ".csproj",
File.ReadAllText(@"projectConfig.txt"));
895.
896.         /*
897.         * The localisation string contains user facing data, mainly display
names.
898.         * While it would be nice to add functionality to support multiple
languages,
899.         * this is a fairly unlikely scenario for my target users, so I have not
implemented it.
900.         */
901.         localizationString += "Mods: {\r\n" + loadedMod.get_name() + ":
{\r\nItems: {\r\n";
902.         for (int i = 0; i < itemsToExport.Length; i++)
903.         {
904.             if (itemsToExport[i].get_type() == "Item")
905.             {
906.                 localizationString += itemsToExport[i].get_name() + ": {";
907.                 if (itemsToExport[i].get_tooltip() != "")
908.                 {
909.                     localizationString += "\r\nTooltip: " +
itemsToExport[i].get_tooltip();
910.                 }
911.                 if (itemsToExport[i].get_displayName() != "")
912.                 {
913.                     localizationString += "\r\nDisplayName: " +
itemsToExport[i].get_displayName();
914.                 }
915.                 localizationString += "\r\n}\r\n";
916.             }
917.         }
918.         localizationString += "}\r\n";
919.         for (int i = 0; i < itemsToExport.Length; i++)
920.         {
921.             if (itemsToExport[i].get_type() == "Projectile")
922.             {
923.                 localizationString += "Projectiles." + itemsToExport[i].get_name()
+ ".DisplayName: " + itemsToExport[i].get_displayName() + "\r\n";
924.             }
925.         }
926.         localizationString += "NPCs: {\r\n";
927.         for (int i = 0; i < itemsToExport.Length; i++)
928.         {
929.             if (itemsToExport[i].get_type() == "NPC")
930.             {
931.                 localizationString += "\r\n" + itemsToExport[i].get_name() + ":
{\r\nDisplayName: " + itemsToExport[i].get_displayName() + "\r\n";
932.             }
933.         }
934.         localizationString += "\r\n";
935.         localizationString += "\r\n}\r\n";
936.         File.WriteAllText(path + "\\Localization\\en-US.hjson",
localizationString);
937.
938.         /*
939.         * The code is generated by the code generator, then the correctly named
code and sprite
940.         * are saved to the export directory.
941.         */
942.         for (int i = 0; i < itemsToExport.Length; i++)
943.         {
944.             tmpCode = codeGenerator.generate_code(itemsToExport[i],
loadedMod.get_name());
945.
946.             if (itemsToExport[i].get_type() == "Item")

```

```

947.         {
948.             File.WriteAllText(path + "\\Items\\" + itemsToExport[i].get_name()
+ ".cs", tmpCode);
949.         }
950.         else if (itemsToExport[i].get_type() == "Projectile")
951.         {
952.             File.WriteAllText(path + "\\Projectiles\\" +
itemsToExport[i].get_name() + ".cs", tmpCode);
953.         }
954.         else if (itemsToExport[i].get_type() == "NPC")
955.         {
956.             File.WriteAllText(path + "\\NPCs\\" + itemsToExport[i].get_name()
+ ".cs", tmpCode);
957.         }
958.         else if (itemsToExport[i].get_type() == "Tile")
959.         {
960.             File.WriteAllText(path + "\\Tiles\\" + itemsToExport[i].get_name()
+ ".cs", tmpCode);
961.         }
962.
963.         /*
964.         * The sprites are written to files here. Each item has a main sprite,
then
965.         * other sprites are checked for and saved.
966.         */
967.         bmp = itemsToExport[i].get_sprite();
968.         if (bmp == null)
969.         {
970.             incompleteItems.Add(itemsToExport[i].get_name());
971.             canExport = false;
972.         }
973.         else
974.         {
975.             if (itemsToExport[i].get_type() == "Item")
976.             {
977.                 bmp.Save(path + "\\Items\\" + itemsToExport[i].get_name() +
".png", ImageFormat.Png);
978.             }
979.             else if (itemsToExport[i].get_type() == "Projectile")
980.             {
981.                 bmp.Save(path + "\\Projectiles\\" +
itemsToExport[i].get_name() + ".png", ImageFormat.Png);
982.             }
983.             else if (itemsToExport[i].get_type() == "NPC")
984.             {
985.                 bmp.Save(path + "\\NPCs\\" + itemsToExport[i].get_name() +
".png", ImageFormat.Png);
986.             }
987.             else if (itemsToExport[i].get_type() == "Tile")
988.             {
989.                 bmp.Save(path + "\\Tiles\\" + itemsToExport[i].get_name() +
".png", ImageFormat.Png);
990.             }
991.         }
992.         bmp = itemsToExport[i].get_wingSprite();
993.         if (bmp != null)
994.         {
995.             if (itemsToExport[i].get_type() == "Item")
996.             {
997.                 bmp.Save(path + "\\Items\\" + itemsToExport[i].get_name() +
"_Wings.png", ImageFormat.Png);
998.             }
999.         }
1000.         bmp = itemsToExport[i].get_headSprite();
1001.         if (bmp != null)

```

```

1002.        {
1003.            if (itemsToExport[i].get_type() == "Item")
1004.            {
1005.                bmp.Save(path + "\\Items\\" + itemsToExport[i].get_name() +
1006.                "_Head.png", ImageFormat.Png);
1007.            }
1008.            bmp = itemsToExport[i].get_bodySprite();
1009.            if (bmp != null)
1010.            {
1011.                if (itemsToExport[i].get_type() == "Item")
1012.                {
1013.                    bmp.Save(path + "\\Items\\" + itemsToExport[i].get_name() +
1014.                    "_Body.png", ImageFormat.Png);
1015.                }
1016.                bmp = itemsToExport[i].get_legsSprite();
1017.                if (bmp != null)
1018.                {
1019.                    if (itemsToExport[i].get_type() == "Item")
1020.                    {
1021.                        bmp.Save(path + "\\Items\\" + itemsToExport[i].get_name() +
1022.                        "_Legs.png", ImageFormat.Png);
1023.                    }
1024.                }
1025.                bmp = itemsToExport[i].get_mapHead();
1026.                if (bmp != null)
1027.                {
1028.                    if (GetTypeOfItem(itemsToExport[i]) == "boss")
1029.                    {
1030.                        bmp.Save(path + "\\NPCs\\" + itemsToExport[i].get_name() +
1031.                        "_Head_Boss.png", ImageFormat.Png);
1032.                    }
1033.                }
1034.            }
1035.            MessageBox.Show("Export complete: the mod will be available in the develop
1036.            mods section of tModLoader");
1037.        }
1038.        else
1039.        {
1040.            MessageBox.Show("Please set an export directory in Edit>Settings");
1041.        }
1042.    }
1043.    else
1044.    {
1045.        MessageBox.Show("Please save the mod before exporting");
1046.    }
1047. }
1048.
1049. private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
1050. {
1051.     Settings settings = new Settings();
1052.     settings.ShowDialog();
1053. }
1054.
1055. public async void displayItem(Item loadedItem)
1056. {
1057.     lock_controls();
1058.     //If the Blockly editor needs to be changed, it is. The correct buttons are
1059.     enabled or disabled.
1060.     if (loadedItem != null)
1061.     {

```

```

1061.         string prevSource = wvCode.Source.ToString();
1062.         if (loadedItem.get_type() == "Item")
1063.         {
1064.             wvCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
Editor\\tool_editor.html");
1065.             txtTooltip.Enabled = true;
1066.             btnRecipe.Enabled = true;
1067.         }
1068.         else if (loadedItem.get_type() == "Projectile")
1069.         {
1070.             wvCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
Editor\\projectile_editor.html");
1071.             txtTooltip.Enabled = false;
1072.             btnRecipe.Enabled = false;
1073.         }
1074.         else if (loadedItem.get_type() == "NPC")
1075.         {
1076.             wvCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
Editor\\npc_editor.html");
1077.             txtTooltip.Enabled = false;
1078.             btnRecipe.Enabled = false;
1079.         }
1080.         else if (loadedItem.get_type() == "Tile")
1081.         {
1082.             wvCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
Editor\\tile_editor.html");
1083.             txtTooltip.Enabled = false;
1084.             btnRecipe.Enabled = false;
1085.         }
1086.
1087.         //Here, it checks to see if the item has an additional sprite, and enables the
button to change it.
1088.         string type = GetTypeOfItem(loadedItem);
1089.         if (type == "body")
1090.         {
1091.             btnAdditionalSprites.Enabled = true;
1092.         }
1093.         else if (type == "head")
1094.         {
1095.             btnAdditionalSprites.Enabled = true;
1096.         }
1097.         else if (type == "legs")
1098.         {
1099.             btnAdditionalSprites.Enabled = true;
1100.         }
1101.         else if (type == "wings")
1102.         {
1103.             btnAdditionalSprites.Enabled = true;
1104.         }
1105.         else
1106.         {
1107.             btnAdditionalSprites.Enabled = false;
1108.         }
1109.
1110.
1111.         //This waits for the webView component to be ready before displaying the
item.
1112.         if (prevSource != wvCode.Source.ToString())
1113.         {
1114.             wvready = false;
1115.         }
1116.
1117.         while (!wvready)
1118.         {
1119.             await Task.Delay(100);

```

```

1120.         }
1121.
1122.         clearBlockly();
1123.         theItem = loadedItem;
1124.         txtDisplayName.Text = theItem.get_displayName();
1125.         txtTooltip.Text = theItem.get_tooltip();
1126.         pbSprite.Refresh();
1127.         sendData();
1128.     }
1129.     if (loadedItem == null)
1130.     {
1131.         wvCode.Source = new Uri(Environment.CurrentDirectory + "\\Blockly
Editor\\start.html");
1132.     }
1133.     unlock_controls();
1134. }
1135.
1136. async void InitWebview()
1137. {
1138.     await wvCode.EnsureCoreWebView2Async(null);
1139. }
1140.
1141. async void requestData()
1142. {
1143.     await wvCode.ExecuteScriptAsync("sendDataToWinForm()");
1144. }
1145.
1146. async void sendData()
1147. {
1148.     await wvCode.ExecuteScriptAsync("loadData('" + theItem.get_code() + "')");
1149. }
1150.
1151. public async void clearBlockly()
1152. {
1153.     await wvCode.ExecuteScriptAsync("clear()");
1154. }
1155.
1156. /*
1157.  * These are needed to lock the controls at certain points where access
1158.  * to the buttons would cause issues, predominantly during saving.
1159.  */
1160. public void lock_controls()
1161. {
1162.     btnChangeSprite.Enabled = false;
1163.     txtDisplayName.Enabled = false;
1164.     txtTooltip.Enabled = false;
1165.     wvCode.Enabled = false;
1166.     btnRecipe.Enabled = false;
1167.     btnAdditionalSprites.Enabled = false;
1168.     lbItems.Enabled = false;
1169.     tbSave.Enabled = false;
1170.     fileSaveMod.Enabled = false;
1171.     wvCode.Enabled = false;
1172.     toolStrip1.Enabled = false;
1173.     btnUndo.Enabled = false;
1174.     btnRedo.Enabled = false;
1175.     exportToolStripMenuItem.Enabled = false;
1176. }
1177.
1178. public void unlock_controls()
1179. {
1180.     wvCode.Enabled = true;
1181.     lbItems.Enabled = true;
1182.     tbSave.Enabled = true;
1183.     fileSaveMod.Enabled = true;

```

```

1184.         wvCode.Enabled = true;
1185.         toolStrip1.Enabled = true;
1186.         btnUndo.Enabled = true;
1187.         btnRedo.Enabled = true;
1188.         exportToolStripMenuItem.Enabled = true;
1189.
1190.         //This has to check whether or not the additional sprite button should be enabled.
1191.         CodeGenerator codeGenerator = new CodeGenerator();
1192.         if (loadedItem != null)
1193.         {
1194.             btnChangeSprite.Enabled = true;
1195.             txtDisplayName.Enabled = true;
1196.             if (loadedItem.get_type() == "Item")
1197.             {
1198.                 txtTooltip.Enabled = true;
1199.                 btnRecipe.Enabled = true;
1200.                 if (codeGenerator.get_slot(loadedItem) != string.Empty)
1201.                 {
1202.                     btnAdditionalSprites.Enabled = true;
1203.                 }
1204.             }
1205.             if (loadedItem.get_type() == "Item")
1206.             {
1207.                 btnRecipe.Enabled = true;
1208.                 txtTooltip.Enabled = true;
1209.             }
1210.             if (GetTypeOfItem(loadedItem) == "boss")
1211.             {
1212.                 btnAdditionalSprites.Enabled = true;
1213.             }
1214.         }
1215.     }
1216.
1217.     private void pbSprite_Paint(object sender, PaintEventArgs e)
1218.     {
1219.         if (theItem != null)
1220.         {
1221.             Bitmap theImage = theItem.get_sprite();
1222.             Graphics g = e.Graphics;
1223.
1224.             //As this will be using mostly pixel art, this prevents blurring instead of
1225.             sharp edges.
1226.             e.Graphics.InterpolationMode = InterpolationMode.NearestNeighbor;
1227.
1228.             if (theItem.get_sprite() != null)
1229.             {
1230.                 double picBoxWidth = pbSprite.Width;
1231.                 double picBoxHeight = pbSprite.Height;
1232.                 double height = theItem.get_sprite().Height;
1233.                 double width = theItem.get_sprite().Width;
1234.                 if (height > width)
1235.                 {
1236.                     e.Graphics.DrawImage(theImage, (int)(picBoxWidth - (picBoxHeight /
1237. height * width)) / 2, 0, (int)(picBoxHeight / height * width), (int)(picBoxHeight));
1238.                 }
1239.                 else if (height < width)
1240.                 {
1241.                     e.Graphics.DrawImage(theImage, 0, (int)(picBoxHeight - (picBoxWidth /
1242. width * height)) / 2, (int)picBoxWidth, (int)(picBoxWidth / width * height));
1243.                 }
1244.                 else
1245.                 {
1246.                     e.Graphics.DrawImage(theImage, 0, 0, pbSprite.Width, pbSprite.Height);
1247.                 }
1248.             }
1249.         }
1250.     }

```

```

1246.     }
1247. }
1248.
1249. private void btnAdditionalSprites_Click(object sender, EventArgs e)
1250. {
1251.     if (theItem != null)
1252.     {
1253.         OtherSprites otherSprites = new OtherSprites(theItem, GetTypeOfItem(theItem));
1254.         otherSprites.ShowDialog();
1255.         theItem = otherSprites.theItem;
1256.     }
1257. }
1258.
1259. private void wvCode_WebMessageReceived(object sender,
CoreWebView2WebMessageReceivedEventArgs e)
1260. {
1261.     workspace = e.TryGetWebMessageAsString();
1262.     returned = true;
1263. }
1264.
1265. private void wvCode_NavigationCompleted(object sender,
CoreWebView2NavigationCompletedEventArgs e)
1266. {
1267.     wvready = true;
1268.     if (wvready == false)
1269.     {
1270.         displayItem(new Item("", ""));
1271.         lock_controls();
1272.     }
1273. }
1274.
1275. private void btnChangeSprite_Click(object sender, EventArgs e)
1276. {
1277.     //This opens a file dialog to let the user select a new sprite, then refreshes the
picture box.
1278.     if (theItem != null)
1279.     {
1280.         OpenFileDialog openSpriteDialog = new OpenFileDialog();
1281.         openSpriteDialog.InitialDirectory = "c:\\";
1282.         openSpriteDialog.Filter = "png files (*.png)|*.png|All files (*.*)|*.*";
1283.         if (openSpriteDialog.ShowDialog() == DialogResult.OK)
1284.         {
1285.             theItem.set_sprite(new Bitmap(@openSpriteDialog.FileName));
1286.             pbSprite.Refresh();
1287.         }
1288.     }
1289. }
1290.
1291. private string GetTypeOfItem(Item item)
1292. {
1293.     //If the below strings were entered as a value, problems would occur.
1294.
1295.     if (item.get_code().Contains("{\\slot\\":\\"Body\\"}"))
1296.     {
1297.         return "body";
1298.     }
1299.     else if (item.get_code().Contains("{\\slot\\":\\"Head\\"}"))
1300.     {
1301.         return "head";
1302.     }
1303.     else if (item.get_code().Contains("{\\slot\\":\\"Legs\\"}"))
1304.     {
1305.         return "legs";
1306.     }
1307.     else if (item.get_code().Contains("{\\type\\":\\"create_wings\\"}"))

```



```

1308.         {
1309.             return "wings";
1310.         }
1311.         else if (item.get_code().Contains("\"boss\":true"))
1312.         {
1313.             return "boss";
1314.         }
1315.
1316.         return "null";
1317.     }
1318.
1319.     private void openExportDirectoryToolStripMenuItem_Click(object sender, EventArgs e)
1320.     {
1321.         string path = File.ReadAllText(Environment.CurrentDirectory + "\\userConfig.txt")
1322. + "\\\" + loadedMod.get_name();
1323.         Process.Start("explorer.exe", @path);
1324.     }
1325.
1326.     private void btnRecipe_Click(object sender, EventArgs e)
1327.     {
1328.         RecipeEditor recipeEditor = new RecipeEditor(loadedItem);
1329.         recipeEditor.ShowDialog();
1330.         loadedItem.set_ingredients(recipeEditor.outputArray);
1331.         loadedItem.set_craftingStationID(recipeEditor.station);
1332.     }
1333.
1334.     private void wvSave_WebMessageReceived(object sender,
CoreWebView2WebMessageReceivedEventArgs e)
1335.     {
1336.         workspace = e.TryGetWebMessageAsString();
1337.         saveReturned = true;
1338.     }
1339.
1340.     private void wvSave_NavigationCompleted(object sender,
CoreWebView2NavigationCompletedEventArgs e)
1341.     {
1342.         wvSaveReady = true;
1343.     }
1344.
1345.     protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
1346.     {
1347.         if (keyData == (Keys.Control | Keys.S | Keys.Shift))
1348.         {
1349.             save_mod_as();
1350.         }
1351.         if (keyData == (Keys.Control | Keys.S))
1352.         {
1353.             save_mod();
1354.         }
1355.         if (keyData == (Keys.Control | Keys.O))
1356.         {
1357.             open_mod();
1358.         }
1359.         if (keyData == (Keys.Control | Keys.Z))
1360.         {
1361.             undo();
1362.         }
1363.         if (keyData == (Keys.Control | Keys.Y))
1364.         {
1365.             redo();
1366.         }
1367.         return base.ProcessCmdKey(ref msg, keyData);
1368.     }
1369.

```

```

1370.         async Task undo()
1371.         {
1372.             await wwCode.ExecuteScriptAsync("undo()");
1373.         }
1374.
1375.         async Task redo()
1376.         {
1377.             await wwCode.ExecuteScriptAsync("redo()");
1378.         }
1379.
1380.         private void btnUndo_Click(object sender, EventArgs e)
1381.         {
1382.             undo();
1383.         }
1384.
1385.         private void btnRedo_Click(object sender, EventArgs e)
1386.         {
1387.             redo();
1388.         }
1389.
1390.         private void viewHelpToolStripMenuItem_Click(object sender, EventArgs e)
1391.         {
1392.             HelpDialog helpDialog = new HelpDialog();
1393.             helpDialog.ShowDialog();
1394.         }
1395.     }
1396. }
1397.

```

## Code generator class – CodeGenerator.cs

This class is used to translate code from the Json provided by Blockly to C#. It also includes a function to return the slot which the item can be equipped to. A single code generator is instantiated during the export process. The procedure *generate\_code()* is called for each item, also sending the mod name as it is needed to generate the correct namespace.

Due to the nature of the Json files, they are very difficult to deserialise in a typical way, as each block is a property of the last. This is why I opted to initially handle the the code as a string first, breaking it down into smaller strings of Json which are more manageable to work with. The blocks are stored in an array of strings.

Each block is first deserialised into the *GenericBlock* class. This class has only a type field, which allows the block to be identified. It is then deserialised into the correct class, so its values can be read, and the code generated accordingly.

Each method that the item could use exists as a string in the code generator. Code is added to each of these strings by certain blocks. Additionally, there are a number of Booleans which indicate properties of the item that are not explicitly set. For example, the value *isWing* will be set to true by certain blocks. This value effects how the code for flight is written. After each block is deserialised, the strings that are not empty concatenated with the correct formatting. The completed code is returned as a string and written to a file by the main form.

```

1. using System;
2. using System.Collections.Generic;
3. using System.IO;
4. using System.Linq;

```

```

5. using System.Text;
6. using System.Threading.Tasks;
7. using System.Text.Json;
8. using System.Windows.Forms;
9. using System.Numerics;
10.
11. namespace NEA_solution
12. {
13.     internal class CodeGenerator
14.     {
15.         //This algorithm will only return the final slot that is assigned in code.
16.         public string get_slot(Item item)
17.         {
18.             string slot = string.Empty;
19.             string blockType;
20.
21.             string[] blocksAsStrings = findBlocksInline(item.get_code());
22.
23.             for (int i = 0; i < blocksAsStrings.Length; i++)
24.             {
25.                 blockType = JsonSerializer.Deserialize<GenericBlock>(blocksAsStrings[i]).type;
26.
27.                 //If a block that assigns it to a slot is found, the value is read from it.
28.                 if (blockType == "equip_slot")
29.                 {
30.                     slot = JsonSerializer.Deserialize<equip_slot>(blocksAsStrings[i]).slot;
31.                 }
32.             }
33.
34.             return slot;
35.
36.         }
37.         public string generate_code(Item item, string modName)
38.         {
39.             string setDefaults = "";
40.             string blockType;
41.             string[] blocksAsStrings;
42.             string UpdateAccessory = "";
43.             bool isEquipable = false;
44.             bool isWing = false;
45.             string SetStaticDefaults = "";
46.             string verticalWingsSpeeds;
47.             float[] wingStats = new float[5];
48.             bool canHover = false;
49.             string slot = null;
50.             List<string> chatOptions = new List<string>();
51.             List<string> nameOptions = new List<string>();
52.             List<string> shopItems = new List<string>();
53.             List<int> shopValues = new List<int>();
54.             bool hasShop = false;
55.             string spawnrate = "";
56.             string NPCloot = "";
57.             string onHit = "";
58.             string useItem = "";
59.             string bossLoot = "";
60.             string bestiary = "";
61.             bool isBoss = false;
62.             string useItemReturn = "\r\n\t\t\t\treturn true;";
63.             string[] vanillaItems = File.ReadAllLines(Environment.CurrentDirectory +
"\\itemIDs.txt");
64.             string[] vanillaMobs = File.ReadAllLines(Environment.CurrentDirectory +
"\\npcIDs.txt");
65.             string[] vanillaTiles = File.ReadAllLines(Environment.CurrentDirectory +
"\\tileIDs.txt");
66.

```

```

67.
68.
69.      /*
70.      * Sets the string to be returned to the namespace setup, with the correct names in
place.
71.      *
72.      * Everything is on a new line to aid readability for test and the end user.
73.      */
74.      string generatedCode = "using Terraria;" +
75.          "\r\nusing System.Linq;" +
76.          "\r\nusing Terraria;" +
77.          "\r\nusing Terraria.DataStructures;" +
78.          "\r\nusing Terraria.ID;" +
79.          "\r\nusing Terraria.ModLoader;" +
80.          "\r\nusing Terraria.ID;" +
81.          "\r\nusing Terraria.ModLoader;" +
82.          "\r\nusing System;" +
83.          "\r\nusing System.Collections.Generic;" +
84.          "\r\nusing Terraria.ModLoader.Utilities;" +
85.          "\r\nusing Terraria.GameContent.ItemDropRules;" +
86.          "\r\nusing Microsoft.Xna.Framework;" +
87.          "\r\nusing Terraria.GameContent.Bestiary;" +
88.          "\r\nusing Terraria.ObjectData;";
89.
90.      string itemType = item.get_type();
91.      if (itemType == "Item")
92.      {
93.          generatedCode += "\r\nnamespace " + modName + ".Items";
94.      }
95.      else if (itemType == "Projectile")
96.      {
97.          generatedCode += "\r\nnamespace " + modName + ".Projectiles";
98.      }
99.      else if (itemType == "NPC")
100.     {
101.         generatedCode += "\r\nnamespace " + modName + ".NPCs";
102.     }
103.     else if (itemType == "Tile")
104.     {
105.         generatedCode += "\r\nnamespace " + modName + ".Tiles";
106.     }
107.     generatedCode += "\r\n{";
108.
109.
110.     //Each block is found in the code, and put in an array of strings.
111.     blocksAsStrings = findBlocksInline(item.get_code());
112.
113.     for (int i = 0; i < blocksAsStrings.Length; i++)
114.     {
115.         /*
116.         * Each block is a small json array, which is deserialised into a class with
only type
117.         * as a field.
118.         *
119.         * This allows the type of block to be identified and handled accordingly.
120.         */
121.         blockType = JsonSerializer.Deserialize<GenericBlock>(blocksAsStrings[i]).type;
122.         /*
123.         * The switch case takes the type of block, and deserialises the block into the
correct
124.         * child class of GenericBlock.
125.         *
126.         * The values and code are then added to the code for the correct method
127.         *

```

```

128.         * the value isEquipable may be set to true. This means it must be made into an
accessory
129.         * to have these properties function.
130.         */
131.     switch (blockType)
132.     {
133.         //Item class blocks
134.         case "define_item":
135.             define_item define_Item =
JsonSerializer.Deserialize<define_item>(blocksAsStrings[i]);
136.             setDefaults += "\r\n\t\t\tItem.value = " + define_Item.value + ";";
137.             setDefaults += "\r\n\t\t\tItem.rare = " + define_Item.rare + ";";
138.             break;
139.
140.         case "define_weapon_essential":
141.             define_weapon_essential define_Weapon_Essential =
JsonSerializer.Deserialize<define_weapon_essential>(blocksAsStrings[i]);
142.
143.             setDefaults += "\r\n\t\t\tItem.damage = " +
define_Weapon_Essential.damage + ";";
144.             setDefaults += "\r\n\t\t\tItem.DamageType = DamageClass." +
define_Weapon_Essential.damageType + ";";
145.             setDefaults += "\r\n\t\t\tItem.knockBack = " +
define_Weapon_Essential.knockback + ";";
146.             setDefaults += "\r\n\t\t\tItem.crit = " + define_Weapon_Essential.crit
+ ";";
147.             break;
148.
149.         case "define_tool":
150.             define_tool define_Tool =
JsonSerializer.Deserialize<define_tool>(blocksAsStrings[i]);
151.             setDefaults += "\r\n\t\t\tItem.useTime = " + define_Tool.useTime + ";";
152.             setDefaults += "\r\n\t\t\tItem.useAnimation = " + define_Tool.useTime +
";";
153.             setDefaults += "\r\n\t\t\tItem.UseSound = SoundID.Item" +
define_Tool.UseSound + ";";
154.             setDefaults += "\r\n\t\t\tItem.autoReuse = " +
define_Tool.autoReuse.ToString().ToLower() + ";";
155.             setDefaults += "\r\n\t\t\tItem.useStyle = " + define_Tool.useStyle +
";";
156.             break;
157.
158.         case "tool_power":
159.             tool_power tool_Power =
JsonSerializer.Deserialize<tool_power>(blocksAsStrings[i]);
160.             setDefaults += "\r\n\t\t\tItem." + tool_Power.tool_type + " = " +
tool_Power.power + ";";
161.             break;
162.
163.         case "is_consumable":
164.             is_consumable is_Consumable =
JsonSerializer.Deserialize<is_consumable>(blocksAsStrings[i]);
165.             setDefaults += "\r\n\t\t\tItem.consumable = " +
is_Consumable.consumable.ToString().ToLower() + ";";
166.             break;
167.
168.         case "no_melee":
169.             no_melee no_Melee =
JsonSerializer.Deserialize<no_melee>(blocksAsStrings[i]);
170.             setDefaults += "\r\n\t\t\tItem.noMelee = " +
(!no_Melee.melee).ToString().ToLower() + ";";
171.             break;
172.
173.         case "shoot_existing_ammo":

```

```

174.         shoot_existing_ammo shoot_Existing_Ammo =
JsonSerializer.Deserialize<shoot_existing_ammo>(blocksAsStrings[i]);
175.         setDefaults += "\r\n\t\t\tItem.shoot = 10;";
176.         + "\r\n\t\t\tItem.shootSpeed = " + shoot_Existing_Ammo.shoot_speed
+ ";";
177.         + "\r\n\t\t\tItem.useAmmo = AmmoID." +
shoot_Existing_Ammo.ammo_type + ";";
178.         if (shoot_Existing_Ammo.ammo_type == "Rocket")
179.         {
180.             setDefaults += "\r\n\t\t\tItem.shoot = ProjectileID.RocketI;";
181.         }
182.         break;
183.
184.         case "change_class_stat":
185.             change_class_stat change_Class_Stat =
JsonSerializer.Deserialize<change_class_stat>(blocksAsStrings[i]);
186.             UpdateAccessory += "\r\n\t\t\ttplayer." + change_Class_Stat.stat +
"(DamageClass." + change_Class_Stat.class_name + ") += " + change_Class_Stat.value + ";";
187.             isEquipable = true;
188.             break;
189.
190.         case "use_mana":
191.             use_mana use_Mana =
JsonSerializer.Deserialize<use_mana>(blocksAsStrings[i]);
192.             setDefaults += "\r\n\t\t\tItem.mana = " + use_Mana.useMana + ";";
193.             break;
194.
195.         case "grant_ability":
196.             grant_ability grant_Ability =
JsonSerializer.Deserialize<grant_ability>(blocksAsStrings[i]);
197.             UpdateAccessory += "\r\n\t\t\ttplayer." + grant_Ability.ability + " =
true;";
198.             isEquipable = true;
199.             break;
200.
201.         case "change_player_stat":
202.             change_player_stat change_Player_Stat =
JsonSerializer.Deserialize<change_player_stat>(blocksAsStrings[i]);
203.             UpdateAccessory += "\r\n\t\t\ttplayer." + change_Player_Stat.stat + " +=
" + change_Player_Stat.value + ";";
204.             isEquipable = true;
205.             break;
206.
207.         case "set_player_stat":
208.             set_player_stat set_Player_Stat =
JsonSerializer.Deserialize<set_player_stat>(blocksAsStrings[i]);
209.             UpdateAccessory += "\r\n\t\t\ttplayer." + set_Player_Stat.stat + " = " +
set_Player_Stat.value + ";";
210.             isEquipable = true;
211.             break;
212.
213.         case "set_class_stat":
214.             set_class_stat set_Class_Stat =
JsonSerializer.Deserialize<set_class_stat>(blocksAsStrings[i]);
215.             UpdateAccessory += "\r\n\t\t\ttplayer." + set_Class_Stat.stat +
"(DamageClass." + set_Class_Stat.class_name + ") = " + set_Class_Stat.value + ";";
216.             isEquipable = true;
217.             break;
218.
219.         case "set_all_player_bools":
220.             set_all_player_bools set_All_Player_Bool =
JsonSerializer.Deserialize<set_all_player_bools>(blocksAsStrings[i]);
221.             UpdateAccessory += "\r\n\t\t\ttplayer." + set_All_Player_Bool.property
+ " = true;";
222.             isEquipable = true;

```

```

223.             break;
224.
225.         case "create_wings":
226.             create_wings create_Wings =
227. JsonSerializer.Deserialize<create_wings>(blocksAsStrings[i]); ;
228.             wingStats[0] = create_Wings.flight_time;
229.             wingStats[1] = create_Wings.flight_speed;
230.             wingStats[2] = create_Wings.acceleration;
231.             isEquipable = true;
232.             isWing = true;
233.             slot = "Wings";
234.             break;
235.
236.         case "wing_hover":
237.             wing_hover wing_Hover =
238. JsonSerializer.Deserialize<wing_hover>(blocksAsStrings[i]);
239.             wingStats[3] = wing_Hover.hover_speed;
240.             wingStats[4] = wing_Hover.acceleration;
241.             canHover = true;
242.             break;
243.
244.         case "use_custom_projectile":
245.             use_custom_projectile use_Custom_Projectile =
246. JsonSerializer.Deserialize<use_custom_projectile>(blocksAsStrings[i]);
247.             setDefaults += "\r\n\t\t\tItem.shoot = 10;" + "\r\nItem.shootSpeed = "
248. + use_Custom_Projectile.shoot_speed + ";" + "\r\nItem.shoot = "
249. ModContent.ProjectileType<Projectiles."> + use_Custom_Projectile.projectile + ">()";
250.             break;
251.
252.         case "equip_slot":
253.             equip_slot equip_Slot =
254. JsonSerializer.Deserialize<equip_slot>(blocksAsStrings[i]);
255.             slot = equip_Slot.slot;
256.             break;
257.
258.         case "create_tile":
259.             create_tile create_Tile =
260. JsonSerializer.Deserialize<create_tile>(blocksAsStrings[i]);
261.             if (vanillaTiles.Contains(create_Tile.tileName))
262.             {
263.                 setDefaults += "\r\n\t\t\tItem.createTile = TileID." +
264. create_Tile.tileName + ";";
265.             }
266.             else
267.             {
268.                 setDefaults += "\r\n\t\t\tItem.createTile = "
269. ModContent.TileType<Tiles."> + create_Tile.tileName + ">()";
270.             }
271.             break;
272.
273.         case "grant_effect":
274.             grant_effect grant_Effect =
275. JsonSerializer.Deserialize<grant_effect>(blocksAsStrings[i]);
276.             setDefaults += "\r\n\t\t\tItem.buffType = BuffID." +
277. grant_Effect.effect + ";";
278.             setDefaults += "\r\n\t\t\tItem.buffTime = " + grant_Effect.time * 60 +
279. ";";
280.             break;
281.
282.         case "hit_effect":
283.             hit_effect hit_Effect =
284. JsonSerializer.Deserialize<hit_effect>(blocksAsStrings[i]);
285.             onHit += "\r\n\t\t\t\ttarget.AddBuff(BuffID." + hit_Effect.effect + ", "
286. + hit_Effect.time * 60 + ");";
287.             break;

```

```

274.
275.         case "spawn_enemy":
276.             spawn_enemy spawn_Energy =
277.             JsonSerializer.Deserialize<spawn_enemy>(blocksAsStrings[i]);
278.             if (vanillaMobs.Contains(spawn_Energy.enemy_name))
279.             {
280.                 useItem += "\r\n\t\t\tNPC.SpawnOnPlayer(player.whoAmI, NPCID." +
281.                 spawn_Energy.enemy_name + ");";
282.             }
283.             else
284.             {
285.                 useItem += "\r\n\t\t\tNPC.SpawnOnPlayer(player.whoAmI,
286.                 ModContent.NPCType<NPCs." + spawn_Energy.enemy_name + ">());";
287.             }
288.             break;
289.
290.         case "max_stack":
291.             max_stack max_Stack =
292.             JsonSerializer.Deserialize<max_stack>(blocksAsStrings[i]);
293.             setDefaults += "\r\n\t\t\tItem.maxStack = " + max_Stack.max + ";";
294.             break;
295.
296.         //Projectile class blocks
297.         case "projectile_basic":
298.             projectile_basic projectile_Basic =
299.             JsonSerializer.Deserialize<projectile_basic>(blocksAsStrings[i]);
300.             setDefaults += "\r\n\t\t\tProjectile.timeLeft = " +
301.             projectile_Basic.time_left * 60 + ";";
302.             break;
303.
304.         case "use_ai":
305.             use_ai use_Ai = JsonSerializer.Deserialize<use_ai>(blocksAsStrings[i]);
306.             setDefaults += "\r\n\t\t\tProjectile.aiStyle = " + use_Ai.style + ";";
307.             break;
308.
309.         case "set_value":
310.             Set_value set_Value =
311.             JsonSerializer.Deserialize<Set_value>(blocksAsStrings[i]);
312.             setDefaults += "\r\n\t\t\tProjectile." + set_Value.property + " = " +
313.             set_Value.value + ";";
314.             break;
315.
316.         case "declare_friendly":
317.             declare_friendly declare_Friendly =
318.             JsonSerializer.Deserialize<declare_friendly>(blocksAsStrings[i]);
319.             setDefaults += "\r\n\t\t\tProjectile.friendly = " +
320.             declare_Friendly.friendly.ToString().ToLower() + ";";
321.             break;
322.
323.         case "declare_hostile":
324.             declare_hostile declare_Hostile =
325.             JsonSerializer.Deserialize<declare_hostile>(blocksAsStrings[i]);
326.             setDefaults += "\r\n\t\t\tProjectile.hostile = " +
327.             declare_Hostile.hostile.ToString().ToLower() + ";";
328.             break;
329.
330.         case "hide_projectile":
331.             hide_projectile hide_Projectile =
332.             JsonSerializer.Deserialize<hide_projectile>(blocksAsStrings[i]);
333.             setDefaults += "\r\n\t\t\tProjectile.hide = " +
334.             hide_Projectile.hide.ToString().ToLower() + ";";
335.             break;
336.
337.         case "collide_with_tiles":

```



```

325.                 collide_with_tiles collide_With_Tiles =
JsonSerializer.Deserialize<collide_with_tiles>(blocksAsStrings[i]);
326.                 setDefaults += "\r\n\t\t\tProjectile.tileCollide = " +
collide_With_Tiles.collide.ToString().ToLower() + ";";
327.                 break;
328.
329.                 case "emit_light":
330.                     emit_light emit_Light =
JsonSerializer.Deserialize<emit_light>(blocksAsStrings[i]);
331.                     setDefaults += "\r\n\t\t\tProjectile.light = " + emit_Light.light +
"f;";
332.                     break;
333.
334.
335.                 //NPC Class Blocks
336.                 case "npc_basic":
337.                     npc_basic npc_Basic =
JsonSerializer.Deserialize<npc_basic>(blocksAsStrings[i]);
338.                     setDefaults += "\r\n\t\t\tNPC.damage = " + npc_Basic.damage + ";";
339.                     setDefaults += "\r\n\t\t\tNPC.defense = " + npc_Basic.defense + ";";
340.                     setDefaults += "\r\n\t\t\tNPC.lifeMax = " + npc_Basic.life + ";";
341.                     setDefaults += "\r\n\t\t\tNPC.knockBackResist = " +
npc_Basic.knockResist + "f;";
342.                     break;
343.
344.                 case "use_npc_ai":
345.                     use_ai use_Npc_Ai =
JsonSerializer.Deserialize<use_ai>(blocksAsStrings[i]);
346.                     setDefaults += "\r\n\t\t\tNPC.aiStyle = " + use_Npc_Ai.style + ";";
347.                     break;
348.
349.                 case "chat_option":
350.                     chat_options chat_Options =
JsonSerializer.Deserialize<chat_options>(blocksAsStrings[i]);
351.                     chatOptions.Add(chat_Options.chat);
352.                     break;
353.
354.                 case "name_option":
355.                     name_options name_Options =
JsonSerializer.Deserialize<name_options>(blocksAsStrings[i]);
356.                     nameOptions.Add(name_Options.name);
357.                     break;
358.
359.                 case "set_spawn_rate":
360.                     set_spawn_rate set_Spawn_Rate =
JsonSerializer.Deserialize<set_spawn_rate>(blocksAsStrings[i]);
361.                     spawnrate += "\r\n\t\t\tspawnChance = " + set_Spawn_Rate.rate + "f;";
362.                     break;
363.
364.                 case "set_spawn_condition":
365.                     set_spawn_condition set_Spawn_Condition =
JsonSerializer.Deserialize<set_spawn_condition>(blocksAsStrings[i]);
366.                     spawnrate += "\r\n\t\t\tspawnChance = SpawnCondition." +
set_Spawn_Condition.condition + ".Chance;";
367.                     spawnrate += "\r\n\t\t\tspawnChance *= " +
set_Spawn_Condition.multiplier + "f;";
368.                     break;
369.
370.                 case "npc_friendly":
371.                     npc_friendly npc_Friendly =
JsonSerializer.Deserialize<npc_friendly>(blocksAsStrings[i]);
372.                     setDefaults += "\r\n\t\t\tNPC.friendly = " +
npc_Friendly.friendly.ToString().ToLower() + ";";
373.                     break;
374.

```

```

375.         case "add_shop_item":
376.             add_shop_item add_Shop_Item =
JsonSerializer.Deserialize<add_shop_item>(blocksAsStrings[i]);
377.             shopItems.Add(add_Shop_Item.item);
378.             shopValues.Add(add_Shop_Item.value);
379.             hasShop = true;
380.             break;
381.
382.         case "add_loot_drop":
383.             add_loot_drop add_Loot_Drop =
JsonSerializer.Deserialize<add_loot_drop>(blocksAsStrings[i]);
384.             if (vanillaItems.Contains(add_Loot_Drop.item))
385.             {
386.                 NPCloot += "\r\n\t\t\tNPCLoot.Add(new CommonDrop(ItemID." +
add_Loot_Drop.item + "," + add_Loot_Drop.denominator + "," + add_Loot_Drop.min + "," +
add_Loot_Drop.max + "," + add_Loot_Drop.numerator + "));";
387.             }
388.             else
389.             {
390.                 NPCloot += "\r\n\t\t\tNPCLoot.Add(new
CommonDrop(ModContent.ItemType<Items." + add_Loot_Drop.item + ">(), " + add_Loot_Drop.denominator +
"," + add_Loot_Drop.min + "," + add_Loot_Drop.max + "," + add_Loot_Drop.numerator + "));";
391.             }
392.             break;
393.
394.         case "is_boss":
395.             is_boss is_Boss =
JsonSerializer.Deserialize<is_boss>(blocksAsStrings[i]);
396.             setDefaults += "\r\n\t\t\tNPC.boss = " +
is_Boss.boss.ToString().ToLower() + ";";
397.             if (is_Boss.boss)
398.             {
399.                 isBoss = true;
400.             }
401.             else
402.             {
403.                 isBoss = false;
404.             }
405.             break;
406.
407.         case "set_boss_value":
408.             set_boss_value set_Boss_Value =
JsonSerializer.Deserialize<set_boss_value>(blocksAsStrings[i]);
409.             setDefaults += "\r\n\t\t\tNPC.value = " + set_Boss_Value.value + ";";
410.             break;
411.
412.         case "set_npc_property":
413.             set_npc_property set_Npc_Property =
JsonSerializer.Deserialize<set_npc_property>(blocksAsStrings[i]);
414.             setDefaults += "\r\n\t\t\tNPC." + set_Npc_Property.property + " =
true;";
415.             break;
416.
417.         case "set_flavour_text":
418.             set_flavour_text set_Flavour_Text =
JsonSerializer.Deserialize<set_flavour_text>(blocksAsStrings[i]);
419.             bestiary += "\r\n\t\t\tnew FlavorTextBestiaryInfoElement(\"" +
set_Flavour_Text.text + "\")";
420.             break;
421.
422.         case "drop_potion":
423.             drop_potion drop_Potion =
JsonSerializer.Deserialize<drop_potion>(blocksAsStrings[i]);
424.             bossLoot += "\r\n\t\t\ttpotionType = ItemID." + drop_Potion.potion +
";";

```

```

425.                 break;
426.
427.
428.                 //Tile blocks
429.                 case "define_tile":
430.                     define_tile define_Tile =
431. JsonSerializer.Deserialize<define_tile>(blocksAsStrings[i]);
432.                     SetStaticDefaults += "\r\n\t\t\tMain.tileSolid[Type] = " +
433. define_Tile.solid.ToString().ToLower() + ";";
434.                     SetStaticDefaults += "\r\n\t\t\tMain.tileMergeDirt[Type] = " +
435. define_Tile.mergeDirt.ToString().ToLower() + ";";
436.                     SetStaticDefaults += "\r\n\t\t\tMain.tileBlockLight[Type] = " +
437. define_Tile.blockLight.ToString().ToLower() + ";";
438.                     SetStaticDefaults += "\r\n\t\t\tAddMapEntry(new Color(" +
439. Convert.ToInt32(define_Tile.colour[1].ToString() + define_Tile.colour[2].ToString(), 16) + ", " +
440. Convert.ToInt32(define_Tile.colour[3].ToString() + define_Tile.colour[4].ToString(), 16) + ", " +
441. Convert.ToInt32(define_Tile.colour[5].ToString() + define_Tile.colour[6].ToString(), 16) + "));";
442.                     break;
443.
444.                 case "min_pickaxe":
445.                     min_pickaxe min_Pickaxe =
446. JsonSerializer.Deserialize<min_pickaxe>(blocksAsStrings[i]);
447.                     SetStaticDefaults += "\r\n\t\t\tMinPick = " + min_Pickaxe.power + ";";
448.                     break;
449.
450.                 case "mine_resist":
451.                     mine_resist mine_Resist =
452. JsonSerializer.Deserialize<mine_resist>(blocksAsStrings[i]);
453.                     SetStaticDefaults += "\r\n\t\t\tMineResist = " + mine_Resist.resist +
454. ";";
455.                     break;
456.
457.                 case "have_solid_top":
458.                     have_solid_top have_Solid_Top =
459. JsonSerializer.Deserialize<have_solid_top>(blocksAsStrings[i]);
460.                     SetStaticDefaults += "\r\n\t\t\tMain.tileSolidTop[Type] = " +
461. have_Solid_Top.solid.ToString().ToLower() + ";";
462.                     break;
463.
464.                 case "frame_important":
465.                     frame_important frame_Important =
466. JsonSerializer.Deserialize<frame_important>(blocksAsStrings[i]);
467.                     SetStaticDefaults += "\r\n\t\t\tMain.tileFrameImportant[Type] = " +
468. frame_Important.connect.ToString().ToLower() + ";";
469.                     break;
470.             }
471.         }
472.
473.         //All these if statements ensure the formatting is correct.
474.         if (item.get_type() == "Item")
475.         {
476.             setDefaults += "\r\n\t\t\tItem.width = " + item.get_sprite().Width + ";";
477.             setDefaults += "\r\n\t\t\tItem.height = " + item.get_sprite().Height + ";";
478.         }
479.         if (item.get_type() == "NPC")
480.         {
481.             setDefaults += "\r\n\t\t\tNPC.width = " + item.get_sprite().Width + ";";
482.             setDefaults += "\r\n\t\t\tNPC.height = " + item.get_sprite().Height + ";";
483.         }
484.         if (item.get_type() == "Projectile")
485.         {
486.             setDefaults += "\r\n\t\t\tProjectile.width = " + item.get_sprite().Width + ";";
487.             setDefaults += "\r\n\t\t\tProjectile.height = " + item.get_sprite().Height +
488. ";";
489.         }

```

```

475.
476.         if (isBoss)
477.         {
478.             generatedCode += "\r\n\t[AutoloadBossHead]";
479.         }
480.
481.         if (slot != null)
482.         {
483.             generatedCode += "\r\n\t[AutoloadEquip(EquipType." + slot + ")]";
484.         }
485.
486.         if (itemType == "Item")
487.         {
488.             generatedCode +=
489.                 "\r\n\tpublic class " + item.get_name() + " : ModItem" +
490.                 "\r\n\t{";
491.         }
492.         else if (itemType == "Projectile")
493.         {
494.             generatedCode +=
495.                 "\r\n\tpublic class " + item.get_name() + " : ModProjectile" +
496.                 "\r\n\t{";
497.         }
498.         else if (itemType == "NPC")
499.         {
500.             generatedCode +=
501.                 "\r\n\tpublic class " + item.get_name() + " : ModNPC" +
502.                 "\r\n\t{";
503.         }
504.         else if (itemType == "Tile")
505.         {
506.             generatedCode +=
507.                 "\r\n\tpublic class " + item.get_name() + " : ModTile" +
508.                 "\r\n\t{";
509.         }
510.
511.         if (isEquipable)
512.         {
513.             setDefaults += "\r\n\t\t\tItem.accessory = true;";
514.         }
515.
516.         if (isWing)
517.         {
518.             if (canHover)
519.             {
520.                 SetStaticDefaults = "\r\n\t\t\tArmorIDs.Wing.Sets.Stats[Item.wingSlot] =
521. new WingStats(" + (int)wingStats[0] + ", " + wingStats[1] + "f, " + wingStats[2] + "f, true, " +
522. wingStats[3] + "f, " + wingStats[4] + "f);";
523.             }
524.             else
525.             {
526.                 SetStaticDefaults = "\r\n\t\t\tArmorIDs.Wing.Sets.Stats[Item.wingSlot] =
527. new WingStats(" + (int)wingStats[0] + ", " + wingStats[1] + "f, " + wingStats[2] + "f);";
528.             }
529.             generatedCode += "\r\n\tpublic override void
530. SetStaticDefaults()\r\n\t\t\t{ \r\n" + SetStaticDefaults + "\r\n\t\t}";
531.         }
532.
533.         verticalWingsSpeeds = "\r\n\tpublic override void VerticalWingSpeeds(Player
534. player, ref float ascentWhenFalling, ref float ascentWhenRising, " +
535. "\r\n\t\t\tref float maxCanAscendMultiplier, ref float maxAscentMultiplier, ref
536. float constantAscend)" +
537. "\r\n\t\t\t{" +
538. "\r\n\t\t\t\tascentWhenFalling = 0.85f;" +
539. "\r\n\t\t\t\tascentWhenRising = 0.15f;" +

```

```

534.         "\r\n\t\tmaxCanAscendMultiplier = 1f;" +
535.         "\r\n\t\tmaxAscentMultiplier = 3f;" +
536.         "\r\n\t\tconstantAscend = 0.135f;" +
537.         "\r\n\t\t";
538.
539.     if (isWing)
540.     {
541.         generatedCode += verticalWingsSpeeds;
542.     }
543.
544.     if (chatOptions.Count > 0)
545.     {
546.         generatedCode += "\r\n\t\tpublic override string GetChat() {" ;
547.         generatedCode += "\r\n\t\t\tRandom rand = new Random();" ;
548.         generatedCode += "\r\n\t\t\tint result = rand.Next(" + chatOptions.Count +
549.             ");";
550.         for (int i = 0; i < chatOptions.Count; i++)
551.         {
552.             generatedCode += "\r\n\t\t\tif (result == " + i + ") { return \"\" +
553.                 chatOptions[i] + "\"; }";
554.             generatedCode += "\r\n\t\t\telse { return \"oh no\"; }";
555.             generatedCode += "\r\n\t\t\t";
556.             setDefaults += "\r\n\t\t\tNPC.townNPC = true;";
557.         }
558.     }
559.     if (hasShop)
560.     {
561.         generatedCode += "\r\n\t\tpublic override void SetChatButtons(ref string
562. button, ref string button2) {" ;
563.         generatedCode += "\r\n\t\t\tbutton = \"Shop\";";
564.         generatedCode += "\r\n\t\t\t";
565.         setDefaults += "\r\n\t\t\tNPC.townNPC = true;";
566.     }
567.
568.     if (spawnrate != "")
569.     {
570.         generatedCode += "\r\n\t\tpublic override float SpawnChance(NPCSpawnInfo
571. spawnInfo)";
572.         generatedCode += "\r\n\t\t\t{\r\n\t\t\t\tfloat spawnChance;";
573.         generatedCode += spawnrate;
574.         generatedCode += "\r\n\t\t\t\treturn spawnChance;\r\n\t\t\t}";
575.     }
576.
577.     //The generated methods are compiled into one string here.
578.     if (item.get_type() != "Tile")
579.     {
580.         generatedCode += "\r\n\t\tpublic override void SetDefaults()\r\n\t\t\t{\r\n" +
581.             setDefaults + "\r\n\t\t\t}";
582.     }
583.     if (isEquipable)
584.     {
585.         generatedCode += "\r\n\t\tpublic override void UpdateAccessory(Player player,
586. bool hideVisual)\r\n\t\t\t{\r\n" + UpdateAccessory + "\r\n\t\t\t}";
587.     }
588.
589.     string[] stations = File.ReadAllLines(Environment.CurrentDirectory +
590. "\\stationIDs.txt");
591.
592.     //The recipes are added here.
593.     if (itemType == "Item")
594.     {
595.         generatedCode += "\r\n\t\tpublic override void AddRecipes() \r\n\t\t\t{" +
596.             "\r\n\t\t\t\tRecipe recipe = CreateRecipe();" ;
597.         for (int i = 0; i < item.ingredients.Length; i++)

```

```

592.        {
593.            generatedCode += "\r\n\t\t\t\trecipe.AddIngredient(ItemID." +
item.get_ingredients()[i].itemName + "," + item.get_ingredients()[i].quantity + ");";
594.        }
595.        if (item.get_craftingStationID() != 0)
596.        {
597.            if (item.get_craftingStationID() != -1)
598.            {
599.                generatedCode += "\r\n\t\t\t\trecipe.AddTile(TileID." +
stations[item.get_craftingStationID()] + ");";
600.            }
601.
602.        }
603.        generatedCode += "\r\n\t\t\t\trecipe.Register();";
604.        generatedCode += "\r\n\t\t\t\t}";
605.    }
606.
607.    if (shopItems.Count > 0)
608.    {
609.        generatedCode += "\r\n\t\t\t\tpublic override void OnChatButtonClicked(bool
firstButton, ref string shop)";
610.        generatedCode += "\r\n\t\t\t\t{";
611.        generatedCode += "\r\n\t\t\t\t\tif (firstButton) { shop = \"Shop\"; }";
612.        generatedCode += "\r\n\t\t\t\t\t}";
613.
614.        generatedCode += "\r\n\t\t\t\t";
615.
616.        generatedCode += "\r\n\t\t\t\t\tpublic override void AddShops()";
617.        generatedCode += "\r\n\t\t\t\t\t{";
618.
619.        generatedCode += "\r\n\t\t\t\t\t\tNPCShop Shop = new NPCShop(Type)";
620.
621.        for (int i = 0; i < shopItems.Count; i++)
622.        {
623.            if (vanillaItems.Contains(shopItems[i]))
624.            {
625.                generatedCode += "\r\n\t\t\t\t\t\t\tShop.Add(ItemID." + shopItems[i] + ");";
626.            }
627.            else
628.            {
629.                generatedCode += "\r\n\t\t\t\t\t\t\tShop.Add(ModContent.ItemType<Items." +
shopItems[i] + ">());";
630.            }
631.        }
632.
633.        generatedCode += "\r\n\t\t\t\t\t\tShop.Register()";
634.
635.        generatedCode += "\r\n\t\t\t\t\t\t}";
636.    }
637.
638.    if (NPCloot != "")
639.    {
640.        generatedCode += "\r\n\t\t\t\t\t\tpublic override void ModifyNPCloot(NPCloot npcLoot)";
641.        generatedCode += "\r\n\t\t\t\t\t\t{";
642.        generatedCode += "\r\n\t\t\t\t\t\t\tNPCloot";
643.        generatedCode += "\r\n\t\t\t\t\t\t\t}";
644.    }
645.
646.    if (onHit != "")
647.    {
648.        generatedCode += "\r\n\t\t\t\t\t\tpublic override void OnHitNPC(Player player, NPC
target, NPC.HitInfo hit, int damageDone)";
649.        generatedCode += "\r\n\t\t\t\t\t\t{";
650.        generatedCode += "\r\n\t\t\t\t\t\t\tonHit";
651.        generatedCode += "\r\n\t\t\t\t\t\t\t}";

```

```

652.         }
653.
654.         if (useItem != "")
655.         {
656.             generatedCode += "\r\n\t\tpublic override bool? UseItem(Player player)";
657.             generatedCode += "\r\n\t\t{";
658.             generatedCode += useItem;
659.             generatedCode += useItemReturn;
660.             generatedCode += "\r\n\t\t}";
661.         }
662.
663.         if (bestiary != "")
664.         {
665.             generatedCode += "\r\n\t\tpublic override void SetBestiary(BestiaryDatabase
666. database, BestiaryEntry bestiaryEntry)";
667.             generatedCode += "\r\n\t\t{";
668.             generatedCode += "\r\n\t\t\tbestiaryEntry.Info.AddRange(new
669. IBestiaryInfoElement[] {";
670.             generatedCode += bestiary;
671.             generatedCode += "\r\n\t\t\t});\r\n\t\t}";
672.         }
673.
674.         if (bossLoot != "")
675.         {
676.             generatedCode += "\r\n\t\tpublic override void BossLoot(ref string name, ref
677. int potionType)";
678.             generatedCode += "\r\n\t\t{";
679.             generatedCode += bossLoot;
680.             generatedCode += "\r\n\t\t}";
681.         }
682.
683.         if (item.get_type() == "Tile")
684.         {
685.             generatedCode += "\r\n\t\tpublic override void SetStaticDefaults()\r\n\t\t{" +
686. SetStaticDefaults + "\r\n\t\t}";
687.         }
688.
689.         generatedCode += "\r\n\t}\r\n";
690.
691.         return generatedCode;
692.     }
693.
694.     private static string[] findBlocksInline(string contents)
695.     {
696.         List<string> blocksList = new List<string>();
697.
698.         string reader = "";
699.         bool reading = false;
700.         int count = 0;
701.
702.         //An ID can contain '}', which causes the program to misidentify blocks, so they
703.         must be removed.
704.         while (contents.Contains("\nid\""))
705.         {
706.             contents = contents.Remove(contents.IndexOf("\nid\""), 28);
707.         }
708.
709.         //This first loop checks for the type of each block and add them to an array.
710.         for (int i = 0; i < contents.Length; i++)
711.         {
712.             reader += contents[i];
713.             if (reader.Contains("\"blocks\":[") || reader.Contains("\"block\":[")
714.             {
715.                 reader = "";
716.                 reading = true;

```

```

712.         }
713.
714.         if (reader.Contains(",") && reading)
715.         {
716.             reading = false;
717.             blocksList.Add(reader);
718.         }
719.     }
720.
721.     /*
722.     * This loop finds the other properties of each block, and adds them to the
723.     * item in the array.
724.     */
725.     for (int i = 0; i < contents.Length; i++)
726.     {
727.         reader += contents[i];
728.         if (reader.Contains("\"fields\":{\"")
729.         {
730.             reader = "";
731.             reading = true;
732.         }
733.
734.         if (reader.Contains("}") && reading)
735.         {
736.             blocksList[count] += reader;
737.             reading = false;
738.             count++;
739.         }
740.     }
741.
742.     for (int i = 0; i < blocksList.Count; i++)
743.     {
744.         blocksList[i].Trim();
745.         if (blocksList[i][blocksList[i].Length - 1] != '}')
746.         {
747.             blocksList[i] = blocksList[i].Trim(',');
748.             blocksList[i] += '}';
749.         }
750.     }
751.
752.     string[] blocks = blocksList.ToArray();
753.
754.     return blocks;
755. }
756. }
757. }
758.

```

## Generic block class – GenericBlock.cs

These classes are what the blocks are deserialised into. They are all children of *GenericBlock*, which contains only the type field needed to identify the block and the correct class for it. It is worth noting the fields for these classes are public, as the getters and setters are needed for deserialization to take place.

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6.

```



```

7. namespace NEA_solution
8. {
9.     //These are definitions of blocks that can have data deserialised into them.
10.    internal class GenericBlock
11.    {
12.        public string type { get; set; }
13.    }
14.
15.    internal class define_item : GenericBlock
16.    {
17.        public int width { get; set; }
18.        public int height { get; set; }
19.        public int value { get; set; }
20.        public string rare { get; set; }
21.    }
22.
23.    internal class define_weapon_essential : GenericBlock
24.    {
25.        public int damage { get; set; }
26.        public string damageType { get; set; }
27.        public int knockback { get; set; }
28.        public int crit { get; set; }
29.    }
30.
31.    internal class define_tool : GenericBlock
32.    {
33.        public int useTime { get; set; }
34.        public string useStyle { get; set; }
35.        public int UseSound { get; set; }
36.        public bool autoReuse { get; set; }
37.    }
38.
39.    internal class tool_power : GenericBlock
40.    {
41.        public string tool_type { get; set; }
42.        public int power { get; set; }
43.    }
44.
45.    internal class shoot_existing_ammo : GenericBlock
46.    {
47.        public string ammo_type { get; set; }
48.        public int shoot_speed { get; set; }
49.    }
50.
51.    internal class change_class_stat : GenericBlock
52.    {
53.        public string stat { get; set; }
54.        public string class_name { get; set; }
55.        public int value { get; set; }
56.    }
57.
58.    internal class use_mana : GenericBlock
59.    {
60.        public int useMana { get; set; }
61.    }
62.
63.    internal class increase_life : GenericBlock
64.    {
65.        public int life { get; set; }
66.    }
67.
68.    internal class increase_move_speed : GenericBlock
69.    {
70.        public float value { get; set; }
71.    }

```

```

72.
73.     internal class grant_ability : GenericBlock
74.     {
75.         public string ability { get; set; }
76.     }
77.
78.     internal class change_player_stat : GenericBlock
79.     {
80.         public string stat { get; set; }
81.         public int value { get; set; }
82.     }
83.
84.     internal class set_player_stat : GenericBlock
85.     {
86.         public string stat { get; set; }
87.         public int value { get; set; }
88.     }
89.
90.     internal class set_class_stat : GenericBlock
91.     {
92.         public string stat { get; set; }
93.         public string class_name { get; set; }
94.         public int value { get; set; }
95.     }
96.
97.     internal class set_all_player_bools : GenericBlock
98.     {
99.         public string property { get; set; }
100.     }
101.
102.     internal class create_wings : GenericBlock
103.     {
104.         public float flight_time { get; set; }
105.         public float flight_speed { get; set; }
106.         public float acceleration { get; set; }
107.     }
108.
109.     internal class wing_hover : GenericBlock
110.     {
111.         public float hover_speed { get; set; }
112.         public float acceleration { get; set; }
113.     }
114.
115.     internal class use_custom_projectile : GenericBlock
116.     {
117.         public string projectile { get; set; }
118.         public int shoot_speed { get; set; }
119.     }
120.
121.     internal class projectile_basic : GenericBlock
122.     {
123.         public int width { get; set; }
124.         public int height { get; set; }
125.         public int time_left { get; set; }
126.     }
127.
128.     internal class use_ai : GenericBlock
129.     {
130.         public string style { get; set; }
131.     }
132.
133.     internal class Set_value : GenericBlock
134.     {
135.         public string property { get; set; }
136.

```

```

137.         public int value { get; set; }
138.     }
139.
140.     internal class declare_friendly : GenericBlock
141.     {
142.         public bool friendly { get; set; }
143.     }
144.
145.     internal class declare_hostile : GenericBlock
146.     {
147.         public bool hostile { get; set; }
148.     }
149.
150.     internal class no_melee : GenericBlock
151.     {
152.         public bool melee { get; set; }
153.     }
154.
155.     internal class is_consumable : GenericBlock
156.     {
157.         public bool consumable { get; set; }
158.     }
159.
160.     internal class hide_projectile : GenericBlock
161.     {
162.         public bool hide { get; set; }
163.     }
164.
165.     internal class collide_with_tiles : GenericBlock
166.     {
167.         public bool collide { get; set; }
168.     }
169.
170.     internal class ignore_water : GenericBlock
171.     {
172.         public bool ignore { get; set; }
173.     }
174.
175.     internal class emit_light : GenericBlock
176.     {
177.         public int light { get; set; }
178.     }
179.
180.     internal class equip_slot : GenericBlock
181.     {
182.         public string slot { get; set; }
183.     }
184.
185.     internal class npc_basic : GenericBlock
186.     {
187.         public int width { get; set; }
188.         public int height { get; set; }
189.         public int damage { get; set; }
190.         public int defense { get; set; }
191.         public int life { get; set; }
192.         public float knockResist { get; set; }
193.     }
194.
195.     internal class chat_options : GenericBlock
196.     {
197.         public string chat { get; set; }
198.     }
199.
200.     internal class name_options : GenericBlock
201.     {

```

```

202.     public string name { get; set; }
203. }
204.
205. internal class add_buttons : GenericBlock
206. {
207.     public string button1 { get; set; }
208.     public string button2 { get; set; }
209. }
210.
211. internal class use_custom_ai : GenericBlock
212. {
213.     public string aiName { get; set; }
214. }
215.
216. //AI Blocks
217. internal class chase_player_X : GenericBlock
218. {
219.     public int xVelocity { get; set; }
220.     public float xAcceleration { get; set; }
221. }
222.
223. internal class chase_player_Y : GenericBlock
224. {
225.     public int yVelocity { get; set; }
226.     public float yAcceleration { get; set; }
227.     public int distance { get; set; }
228. }
229.
230. internal class set_spawn_rate: GenericBlock
231. {
232.     public float rate { get; set; }
233. }
234.
235. internal class set_spawn_condition : GenericBlock
236. {
237.     public string condition { get; set; }
238.     public float multiplier { get; set; }
239. }
240.
241.
242. internal class create_tile : GenericBlock
243. {
244.     public string tileName { get; set; }
245. }
246.
247. internal class tile_default : GenericBlock
248. {
249.     public bool solid { get; set; }
250.     public bool merge { get; set; }
251.     public bool block_light { get; set; }
252.     public string dust_type { get; set; }
253. }
254.
255. internal class npc_friendly : GenericBlock
256. {
257.     public bool friendly { get; set; }
258. }
259.
260. internal class add_shop_item : GenericBlock
261. {
262.     public string item { get; set; }
263.     public int value { get; set; }
264. }
265.
266. internal class grant_effect : GenericBlock

```

```

267.     {
268.         public string effect { get; set; }
269.         public int time { get; set; }
270.     }
271.
272.     internal class add_loot_drop : GenericBlock
273.     {
274.         public int min { get; set; }
275.         public int max { get; set; }
276.         public string item { get; set; }
277.         public string amount { get; set; }
278.         public int numerator { get; set; }
279.         public int denominator { get; set; }
280.     }
281.
282.     internal class hit_effect : GenericBlock
283.     {
284.         public string effect { get; set; }
285.         public int time { get; set; }
286.     }
287.
288.     internal class spawn_enemy : GenericBlock
289.     {
290.         public string enemy_name { get; set; }
291.     }
292.
293.     internal class is_boss : GenericBlock
294.     {
295.         public bool boss { get; set; }
296.     }
297.
298.     internal class max_stack : GenericBlock
299.     {
300.         public int max { get; set; }
301.     }
302.
303.     internal class set_boss_value : GenericBlock
304.     {
305.         public int value { get; set; }
306.     }
307.
308.     internal class set_npc_property : GenericBlock
309.     {
310.         public string property { get; set; }
311.     }
312.
313.     internal class define_tile : GenericBlock
314.     {
315.         public bool solid { get; set; }
316.         public bool mergeDirt { get; set; }
317.         public bool blockLight { get; set; }
318.         public string colour { get; set; }
319.     }
320.
321.     internal class min_pickaxe : GenericBlock
322.     {
323.         public int power { get; set; }
324.     }
325.
326.     internal class mine_resist : GenericBlock
327.     {
328.         public int resist { get; set; }
329.     }
330.
331.     internal class set_flavour_text : GenericBlock

```

```

332.     {
333.         public string text { get; set; }
334.     }
335.     internal class drop_potion : GenericBlock
336.     {
337.         public string potion { get; set; }
338.     }
339.
340.     internal class have_solid_top : GenericBlock
341.     {
342.         public bool solid { get; set; }
343.     }
344.
345.     internal class frame_important : GenericBlock
346.     {
347.         public bool connect { get; set; }
348.     }
349. }
350.

```

## Mod class – Mod.cs

This class handles the list of items in the mod, as well as containing the properties like the name of the project.

```

1. using System;
2. using System.Collections.Generic;
3. using System.Drawing;
4. using System.Linq;
5. using System.Text;
6. using System.Threading.Tasks;
7. using System.Windows.Forms;
8.
9. namespace NEA_solution
10. {
11.     public class Mod
12.     {
13.         public Item[] items;
14.         private string name;
15.         private string description;
16.         private string author;
17.         private string modPath;
18.         private double version;
19.         private Bitmap icon;
20.
21.         public Mod(string name, string modPath)
22.         {
23.             this.name = name;
24.             this.modPath = modPath;
25.             items = new Item[0];
26.             author = string.Empty;
27.             description = string.Empty;
28.             version = 0;
29.         }
30.
31.         //This creates a new array one item larger to add the new item to.
32.         public void add_item(Item item)
33.         {
34.             Item[] tempItems = new Item[items.Length + 1];
35.             for (int i = 0; i < items.Length; i++)
36.             {
37.                 tempItems[i] = items[i];

```

```

38.         }
39.         tempItems[items.Length] = item;
40.         items = tempItems;
41.     }
42.
43.     public Item get_item(int index)
44.     {
45.         try
46.         {
47.             return items[index];
48.         }
49.         catch
50.         {
51.             { return null; }
52.         }
53.     }
54.
55.     public string get_name() { return name; }
56.     public string get_description() { return description; }
57.     public string get_author() { return author; }
58.     public string get_modPath() { return modPath; }
59.     public int get_item_number() { return items.Length; }
60.     public double get_version() { return version; }
61.     public Bitmap get_icon() { return icon; }
62.     public void set_name(string name) { this.name = name.Replace('\u0020', '_'); }
63.     public void set_author (string author) { this.author = author; }
64.     public void set_description(string description) { this.description = description; }
65.     public void set_modPath(string modPath) { this.modPath = modPath; }
66.     public void set_items(Item[] items) { this.items = items; }
67.     public void set_version(double version) { this.version = version; }
68.     public void set_icon(Bitmap icon) { this.icon = icon; }
69.
70.     public string[,] get_items_for_display()
71.     {
72.         string[,] itemsDisplay = new string[items.Length, items.Length+1];
73.         for (int i = 0; i < items.Length; i++)
74.         {
75.             itemsDisplay[i, 0] = items[i].get_name();
76.             itemsDisplay[i, 1] = items[i].get_type();
77.         }
78.         return itemsDisplay;
79.     }
80. }
81. }
82.

```

## Item class – Item.cs

This class stores items in the mod. It is mostly getters and setters, but also contains some input sanitisation, specifically to remove spaces from names that are going to be used in code.

```

1. using System;
2. using System.Collections.Generic;
3. using System.Drawing;
4. using System.Linq;
5. using System.Text;
6. using System.Threading.Tasks;
7. using System.Windows.Forms;
8.
9. namespace NEA_solution
10. {
11.     public class Item

```

```

12.     {
13.         private string name;
14.         private string displayName;
15.         private string tooltip;
16.         private string code;
17.         private string type;
18.         private Bitmap sprite;
19.         private Bitmap wingSprite;
20.         private Bitmap headSprite;
21.         private Bitmap bodySprite;
22.         private Bitmap legsSprite;
23.         private RecipeItem[] ingredients = new RecipeItem[1];
24.         private Bitmap mapHead;
25.         private int craftingStationID = -1;
26.
27.
28.         public Item(string name, string type)
29.         {
30.             //The spaces in the name must be replaced with underscores to make it a valid class
name.
31.             ingredients[0] = new RecipeItem("DirtBlock", 10);
32.             this.name = name.Replace('\u0020', '_');
33.             this.type = type;
34.             displayName = name;
35.             tooltip = "";
36.             code = "";
37.             type = "";
38.             sprite = null;
39.         }
40.
41.         public string get_name() { return name; }
42.         public string get_displayName() { return displayName; }
43.         public string get_tooltip() { return tooltip; }
44.         public string get_type() { return type; }
45.         public string get_code() { return code; }
46.         public Bitmap get_sprite() { return sprite; }
47.         public RecipeItem[] get_ingredients() { return ingredients; }
48.         public Bitmap get_mapHead() { return mapHead; }
49.         public int get_craftingStationID() { return craftingStationID; }
50.         public void set_tooltip(string tooltip)
51.         {
52.             this.tooltip = tooltip;
53.         }
54.         public void set_display_name(string displayName)
55.         {
56.             this.displayName = displayName;
57.         }
58.         public void set_code(string code)
59.         {
60.             this.code = code;
61.         }
62.         public void set_type(string type)
63.         {
64.             this.type = type;
65.         }
66.         public void set_sprite(Bitmap sprite)
67.         {
68.             this.sprite = sprite;
69.         }
70.         public Bitmap get_wingSprite()
71.         {
72.             return wingSprite;
73.         }
74.         public void set_wingSprite(Bitmap wingSprite)
75.         {

```

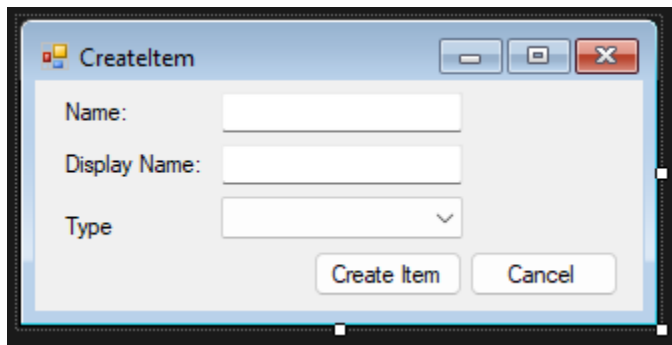


```

76.         this.wingSprite = wingSprite;
77.     }
78.     public Bitmap get_headSprite()
79.     {
80.         return headSprite;
81.     }
82.     public void set_headSprite(Bitmap headSprite)
83.     {
84.         this.headSprite = headSprite;
85.     }
86.     public Bitmap get_bodySprite()
87.     {
88.         return bodySprite;
89.     }
90.     public void set_bodySprite(Bitmap bodySprite)
91.     {
92.         this.bodySprite = bodySprite;
93.     }
94.     public Bitmap get_legsSprite()
95.     {
96.         return legsSprite;
97.     }
98.     public void set_legsSprite(Bitmap legsSprite)
99.     {
100.        this.legsSprite = legsSprite;
101.    }
102.
103.    public void set_ingredients(RecipeItem[] ingredients)
104.    {
105.        this.ingredients = ingredients;
106.    }
107.    public void set_mapHead(Bitmap mapHead)
108.    {
109.        this.mapHead = mapHead;
110.    }
111.    public void set_craftingStationID (int craftingStationID)
112.    {
113.        this.craftingStationID = craftingStationID;
114.    }
115. }
116. }
117.

```

## Create item form – CreateItemDialog.cs



The screenshot shows a Windows-style dialog box titled "CreateItem". It has standard window controls (minimize, maximize, close) in the top right corner. The dialog contains three input fields: "Name:" with a text box, "Display Name:" with a text box, and "Type" with a dropdown menu. At the bottom of the dialog are two buttons: "Create Item" and "Cancel".

This form is shown when the user creates a new item. It ensures that the necessary values are entered and creates a new item using those values.

```

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Linq;
7. using System.Security.Cryptography.X509Certificates;
8. using System.Text;
9. using System.Threading.Tasks;
10. using System.Windows.Forms;
11.
12. namespace NEA_solution
13. {
14.     public partial class CreateItemDialog : Form
15.     {
16.         public Item newItem;
17.         public CreateItemDialog()
18.         {
19.             InitializeComponent();
20.             btnOK.DialogResult = DialogResult.None;
21.             MaximizeBox = false;
22.             MinimizeBox = false;
23.         }
24.
25.         private void btnOK_Click(object sender, EventArgs e)
26.         {
27.             //This ensures the needed values are entered before creating a new item using them.
28.             if (txtName.Text.Length != 0 && cbType.SelectedIndex != -1)
29.             {
30.                 newItem = new Item(txtName.Text, cbType.Text);
31.                 newItem.set_display_name(txtDisplayName.Text);
32.                 DialogResult = DialogResult.OK;
33.             }
34.             else
35.             {
36.                 MessageBox.Show("Please enter all values");
37.             }
38.         }
39.     }
40. }

```

## Edit project details form – EditDetailsDialog.cs

The screenshot shows a Windows application window titled "EditDetailsDialog". The window contains the following elements:

- Name:** A text input field.
- Author:** A text input field.
- Description:** A large text area for entering details.
- Version:** A text input field currently displaying "0.0".
- Change Icon:** A button located at the bottom right of the form.
- Right Panel:** A large, empty rectangular area on the right side of the window, possibly for a preview or additional settings.

This form is shown when the user edits the details of the project. It mainly contains text inputs, but also has a picture box that ensures the aspect ratio of the icon is maintained.

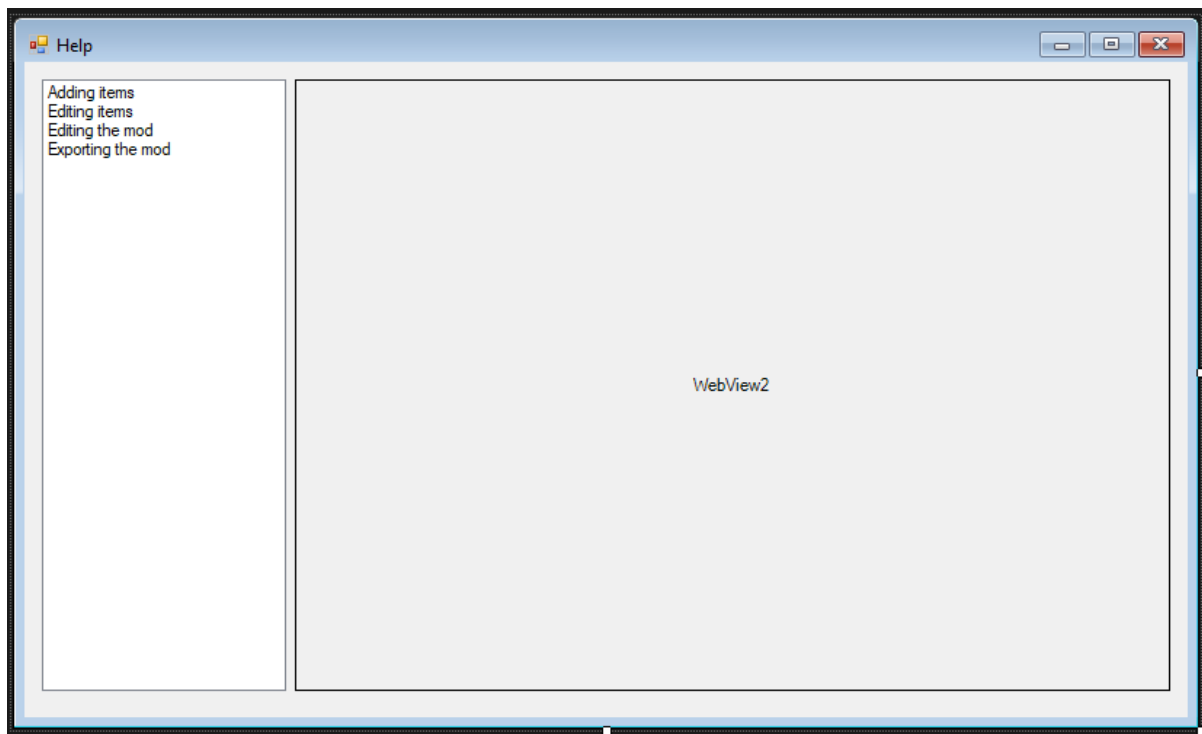
```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Drawing.Drawing2D;
7. using System.Linq;
8. using System.Text;
9. using System.Threading.Tasks;
10. using System.Windows.Forms;
11. using static System.Windows.Forms.VisualStyles.VisualStyleElement.Tab;
12.
13. namespace NEA_solution
14. {
15.     public partial class EditDetailsDialog : Form
16.     {
17.         public string name;
18.         public string description;
19.         public string author;
20.         public double version;
21.         public Bitmap icon;
22.         public EditDetailsDialog(string name, string author, string description, double version,
23.         Bitmap icon)
24.         {
25.             InitializeComponent();
26.             txtAuthor.Text = author;
27.             txtDescription.Text = description;
28.             txtName.Text = name;
29.             numVersion.Value = (decimal)version;
30.             this.icon = icon;
31.             pictureBox1.Refresh();
32.             MaximizeBox = false;
33.             MinimizeBox = false;
34.
35.             //This maintains the correct aspect ratio for the icon, while resizing it to fit in the
36.             picture box.
37.             private void pictureBox1_Paint(object sender, PaintEventArgs e)
38.             {
39.                 Bitmap theImage = icon;
40.                 Graphics g = e.Graphics;
41.
42.                 //As this will be using mostly pixel art, this prevents blurring instead of sharp
43.                 edges.
44.                 e.Graphics.InterpolationMode = InterpolationMode.NearestNeighbor;
45.
46.                 if (icon != null)
47.                 {
48.                     double picBoxWidth = pictureBox1.Width;
49.                     double picBoxHeight = pictureBox1.Height;
50.                     double height = icon.Height;
51.                     double width = icon.Width;
52.                     if (height > width)
53.                     {
54.                         e.Graphics.DrawImage(theImage, (int)(picBoxWidth - (picBoxHeight / height *
55.                         width)) / 2, 0, (int)(picBoxHeight / height * width), (int)(picBoxHeight));
56.                     }
57.                     else if (height < width)
58.                     {
59.                         e.Graphics.DrawImage(theImage, 0, (int)(picBoxHeight - (picBoxWidth / width
60.                         * height)) / 2, (int)picBoxWidth, (int)(picBoxWidth / width * height));
61.                     }
62.                 }
63.             }
64.         }
65.     }
66. }
```

```

57.         }
58.     else
59.     {
60.         e.Graphics.DrawImage(theImage, 0, 0, pictureBox1.Width, pictureBox1.Height);
61.     }
62. }
63.
64.
65. private void btnChangeIcon_Click(object sender, EventArgs e)
66. {
67.     OpenFileDialog openSpriteDialog = new OpenFileDialog();
68.     openSpriteDialog.InitialDirectory = "c:\\\\";
69.     openSpriteDialog.Filter = "png files (*.png)|*.png|All files (*.*)|*.*";
70.     if (openSpriteDialog.ShowDialog() == DialogResult.OK)
71.     {
72.         icon = new Bitmap(@openSpriteDialog.FileName);
73.         pictureBox1.Refresh();
74.     }
75. }
76.
77. //As the form closes the values are set.
78. private void EditDetailsDialog_FormClosing(object sender, FormClosingEventArgs e)
79. {
80.     name = txtName.Text;
81.     description = txtDescription.Text;
82.     author = txtAuthor.Text;
83.     version = (double)numVersion.Value;
84. }
85. }
86. }
87.

```

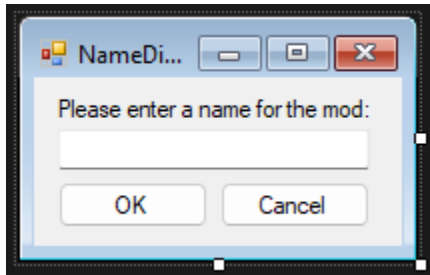
## Help form – HelpDialog.cs



This form is displayed when the user presses the help button. It switches between 4 HTML pages, making use of a WebView2 component.

```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Linq;
7. using System.Text;
8. using System.Threading.Tasks;
9. using System.Windows.Forms;
10.
11. namespace NEA_solution
12. {
13.     public partial class HelpDialog : Form
14.     {
15.         public HelpDialog()
16.         {
17.             InitializeComponent();
18.             webView21.Source = new Uri(Environment.CurrentDirectory + "\\Help
Pages\\AddItemHelp.html");
19.         }
20.
21.         //When the item is double clicked, the corresponding page is loaded.
22.         private void lbMenu_DoubleClick(object sender, EventArgs e)
23.         {
24.             if (lbMenu.SelectedIndex == 0)
25.             {
26.                 webView21.Source = new Uri(Environment.CurrentDirectory + "\\Help
Pages\\AddItemHelp.html");
27.             }
28.             else if (lbMenu.SelectedIndex == 1)
29.             {
30.                 webView21.Source = new Uri(Environment.CurrentDirectory + "\\Help
Pages\\EditItemHelp.html");
31.             }
32.             else if (lbMenu.SelectedIndex == 2)
33.             {
34.                 webView21.Source = new Uri(Environment.CurrentDirectory + "\\Help
Pages\\EditModHelp.html");
35.             }
36.             else if (lbMenu.SelectedIndex == 3)
37.             {
38.                 webView21.Source = new Uri(Environment.CurrentDirectory + "\\Help
Pages\\ExportModHelp.html");
39.             }
40.         }
41.     }
42. }
43.
```

## Name form – NameDialog.cs

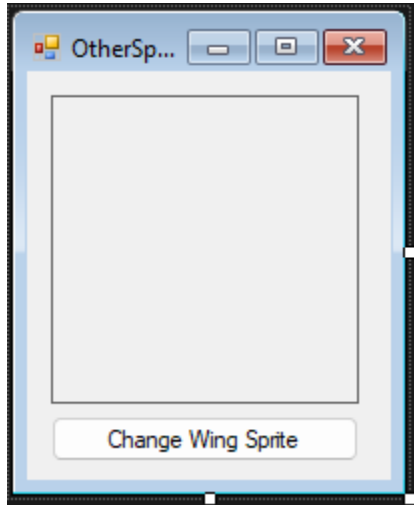


This form is opened when the mod is saved as. It prompts them to enter a name for the mod so it has a name to be saved under.

```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Linq;
7. using System.Text;
8. using System.Threading.Tasks;
9. using System.Windows.Forms;
10.
11. namespace NEA_solution
12. {
13.     public partial class NameDialog : Form
14.     {
15.         public string name;
16.         bool isCanceled;
17.         public NameDialog()
18.         {
19.             InitializeComponent();
20.             this.MaximizeBox = false;
21.             this.MinimizeBox = false;
22.         }
23.
24.         private void NameDialog_FormClosing(object sender, FormClosingEventArgs e)
25.         {
26.             //If the creating the item is not cancelled, then it checks that the correct details
27.             //are entered.
28.             if (!isCanceled)
29.             {
30.                 if (txtName.Text != "")
31.                 {
32.                     name = txtName.Text;
33.                 }
34.                 else
35.                 {
36.                     //If the details are not present, the closing is canceled.
37.                     MessageBox.Show("Please enter a valid name");
38.                     e.Cancel = true;
39.                 }
40.             }
41.
42.             private void btnCancel_Click(object sender, EventArgs e)
43.             {
44.                 isCanceled = true;
45.             }
46.         }
```

```
47. }  
48.
```

## Other sprites form – OtherSprites.cs



This is where the user can change the other sprites for the item. It ensures that the correct sprite is loaded, and the button is appropriately labelled.

```
1. using System;  
2. using System.Collections.Generic;  
3. using System.ComponentModel;  
4. using System.Data;  
5. using System.Drawing;  
6. using System.Drawing.Drawing2D;  
7. using System.Linq;  
8. using System.Text;  
9. using System.Threading.Tasks;  
10. using System.Windows.Forms;  
11. using static System.Windows.Forms.VisualStyles.VisualStyleElement.Tab;  
12.  
13. namespace NEA_solution  
14. {  
15.     public partial class OtherSprites : Form  
16.     {  
17.         public Item theItem;  
18.         string type;  
19.         public OtherSprites(Item item, string type)  
20.         {  
21.             InitializeComponent();  
22.             theItem = item;  
23.             this.MaximizeBox = false;  
24.             this.MinimizeBox = false;  
25.             this.type = type;  
26.             if (type == "body")  
27.             {  
28.                 btnChangeSprite.Text = "Change Body Sprite";  
29.             }  
30.             else if (type == "head")  
31.             {  
32.                 btnChangeSprite.Text = "Change Head Sprite";  
33.             }  
34.             else if (type == "legs")
```

```

35.         {
36.             btnChangeSprite.Text = "Change Legs Sprite";
37.         }
38.         else if (type == "wings")
39.         {
40.             btnChangeSprite.Text = "Change Wings Sprite";
41.         }
42.         else if (type == "boss")
43.         {
44.             btnChangeSprite.Text = "Change Map Head Sprite";
45.         }
46.     }
47.
48.     private void btnChangeSprite_Click(object sender, EventArgs e)
49.     {
50.         if (theItem != null)
51.         {
52.             OpenFileDialog openSpriteDialog = new OpenFileDialog();
53.             openSpriteDialog.InitialDirectory = "c:\\\\";
54.             openSpriteDialog.Filter = "png files (*.png)|*.png|All files (*.*)|*.*";
55.             if (openSpriteDialog.ShowDialog() == DialogResult.OK)
56.             {
57.                 if (type == "body")
58.                 {
59.                     theItem.set_bodySprite(new Bitmap(@openSpriteDialog.FileName));
60.                 }
61.                 else if (type == "head")
62.                 {
63.                     theItem.set_headSprite(new Bitmap(@openSpriteDialog.FileName));
64.                 }
65.                 else if (type == "legs")
66.                 {
67.                     theItem.set_legsSprite(new Bitmap(@openSpriteDialog.FileName));
68.                 }
69.                 else if (type == "wings")
70.                 {
71.                     theItem.set_wingSprite(new Bitmap(@openSpriteDialog.FileName));
72.                 }
73.                 else if (type == "boss")
74.                 {
75.                     theItem.set_mapHead(new Bitmap(@openSpriteDialog.FileName));
76.                 }
77.
78.                 pbSprite.Refresh();
79.             }
80.         }
81.     }
82.
83.     private void pbSprite_Paint(object sender, PaintEventArgs e)
84.     {
85.         if (theItem != null)
86.         {
87.             Bitmap theImage = null;
88.             if (type == "body")
89.             {
90.                 theImage = theItem.get_bodySprite();
91.             }
92.             else if (type == "head")
93.             {
94.                 theImage = theItem.get_headSprite();
95.             }
96.             else if (type == "legs")
97.             {
98.                 theImage = theItem.get_legsSprite();
99.             }

```

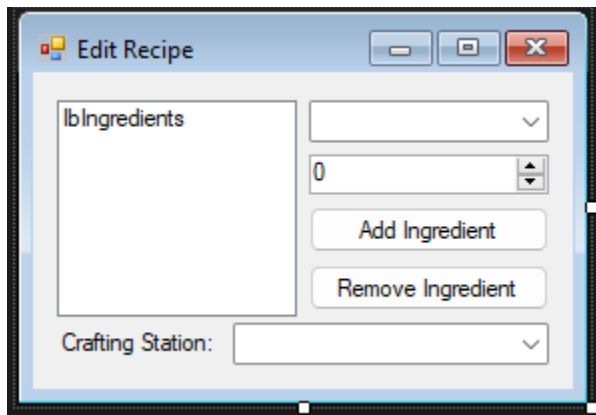


```

100.         else if (type == "wings")
101.         {
102.             theImage = theItem.get_wingSprite();
103.         }
104.         else if (type == "boss")
105.         {
106.             theImage = theItem.get_mapHead();
107.         }
108.         else
109.         {
110.             this.Close();
111.         }
112.
113.         //This ensures the sprite's aspect ratio is maintained.
114.         Graphics g = e.Graphics;
115.         e.Graphics.InterpolationMode = InterpolationMode.NearestNeighbor;
116.         if (theImage != null)
117.         {
118.             double picBoxWidth = pbSprite.Width;
119.             double picBoxHeight = pbSprite.Height;
120.             double height = theImage.Height;
121.             double width = theImage.Width;
122.             if (height > width)
123.             {
124.                 e.Graphics.DrawImage(theImage, (int)(picBoxWidth - (picBoxHeight /
height * width)) / 2, 0, (int)(picBoxHeight / height * width), (int)(picBoxHeight));
125.             }
126.             else if (height < width)
127.             {
128.                 e.Graphics.DrawImage(theImage, 0, (int)(picBoxHeight - (picBoxWidth /
width * height)) / 2, (int)picBoxWidth, (int)(picBoxWidth / width * height));
129.             }
130.             else
131.             {
132.                 e.Graphics.DrawImage(theImage, 0, 0, pbSprite.Width, pbSprite.Height);
133.             }
134.         }
135.     }
136. }
137. }
138. }
139.

```

## Recipe editor form – RecipeEditor.cs



This form loads the list of items in the game from a file. The recipe items and quantities are added to a list which is assigned to the loaded item.

```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.Linq;
7. using System.Text;
8. using System.Threading.Tasks;
9. using System.Windows.Forms;
10. using System.IO;
11.
12. namespace NEA_solution
13. {
14.     public partial class RecipeEditor : Form
15.     {
16.         RecipeItem currentItem;
17.         List<RecipeItem> IngredientsList = new List<RecipeItem>();
18.         public RecipeItem[] outputArray;
19.         public int station;
20.         public RecipeEditor(Item item)
21.         {
22.             InitializeComponent();
23.             this.MaximizeBox = false;
24.             this.MinimizeBox = false;
25.             string[] items = File.ReadAllLines(Environment.CurrentDirectory + "\\itemIDs.txt");
26.             //The list of items is loaded from the file into the combo box.
27.             for (int i = 0; i < items.Length; i++)
28.             {
29.                 cbIngredient.Items.Add(items[i]);
30.             }
31.             //The items are added to the list box.
32.             if (item.get_ingredients() != null)
33.             {
34.                 for (int i = 0; i < item.get_ingredients().Length; i++)
35.                 {
36.                     lbIngredients.Items.Add(item.get_ingredients()[i].itemName);
37.                     IngredientsList.Add(item.get_ingredients()[i]);
38.                 }
39.             }
40.             lbIngredients.SelectedIndex = 0;
41.             numQuantity.Value = IngredientsList[0].quantity;
42.             cbStation.SelectedIndex = item.get_craftingStationID();
43.         }
44.
45.         private void lbIngredients_SelectedIndexChanged(object sender, EventArgs e)
46.         {
47.             if (lbIngredients.SelectedIndex != -1)
48.             {
49.                 currentItem = IngredientsList[lbIngredients.SelectedIndex];
50.                 cbIngredient.Text = currentItem.itemName;
51.                 numQuantity.Value = currentItem.quantity;
52.             }
53.         }
54.
55.         private void btnAdd_Click(object sender, EventArgs e)
56.         {
57.             //A "blank" item is added.
58.             lbIngredients.Items.Add("NEW ITEM");
59.             numQuantity.Value = 1;
60.             IngredientsList.Add(new RecipeItem("NEW ITEM", 1));
61.         }
62.     }
63. }
```

```

62.
63.     private void cbIngredient_TextChanged(object sender, EventArgs e)
64.     {
65.         currentItem.itemName = cbIngredient.Text;
66.         lbIngredients.Items[lbIngredients.SelectedIndex] = cbIngredient.Text;
67.         lbIngredients.Refresh();
68.     }
69.
70.     private void numQuantity_ValueChanged(object sender, EventArgs e)
71.     {
72.         currentItem.quantity = (int)numQuantity.Value;
73.     }
74.
75.     private void RecipeEditor_FormClosing(object sender, FormClosingEventArgs e)
76.     {
77.         outputArray = IngredientsList.ToArray();
78.         station = cbStation.SelectedIndex;
79.     }
80.
81.     private void btnRemove_Click(object sender, EventArgs e)
82.     {
83.         IngredientsList.Remove(currentItem);
84.         lbIngredients.Items.Clear();
85.         foreach (RecipeItem item in IngredientsList)
86.         {
87.             lbIngredients.Items.Add(item.itemName);
88.         }
89.         lbIngredients.Refresh();
90.     }
91. }
92. }
93.

```

## Recipe item class – RecipeItem.cs

This is a container for an item and quantity.

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6.
7. namespace NEA_solution
8. {
9.     public class RecipeItem
10.    {
11.        public string itemName { get; set; }
12.        public int quantity { get; set; }
13.        public RecipeItem(string name, int number)
14.        {
15.            itemName = name;
16.            quantity = number;
17.        }
18.    }
19. }
20.

```

## Settings form – Settings.cs

This allows the user to change the export path.

```
1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Data;
5. using System.Drawing;
6. using System.IO;
7. using System.Linq;
8. using System.Text;
9. using System.Threading.Tasks;
10. using System.Windows.Forms;
11. using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;
12. using System.Xml.Linq;
13. using System.Reflection;
14.
15. namespace NEA_solution
16. {
17.     public partial class Settings : Form
18.     {
19.         string path;
20.         string filePath = Environment.CurrentDirectory + "\\userConfig.txt";
21.         public Settings()
22.         {
23.             InitializeComponent();
24.             txtPath.Text = File.ReadAllText(filePath);
25.             this.MaximizeBox = false;
26.             this.MinimizeBox = false;
27.             btnApply.Enabled = false;
28.         }
29.
30.         private void btnApply_Click(object sender, EventArgs e)
31.         {
32.             Console.WriteLine(filePath);
33.             File.WriteAllText(filePath, txtPath.Text);
34.             //Disabling the button makes it clear to the user that the changes have been
35.             applied.
36.             btnApply.Enabled = false;
37.         }
38.
39.         private void btnFolderDialog_Click(object sender, EventArgs e)
40.         {
41.             FolderBrowserDialog folderDialog = new FolderBrowserDialog();
42.             DialogResult result = folderDialog.ShowDialog();
43.             if (result == DialogResult.OK)
44.             {
45.                 path = folderDialog.SelectedPath;
46.                 txtPath.Text = path;
47.             }
48.         }
49.
50.         private void txtPath_TextChanged(object sender, EventArgs e)
51.         {
52.             btnApply.Enabled = true;
53.         }
54.     }
55. }
```

## Blockly Scripts

Each of these editors contains definitions for the blocks they contain, as well as functions to manage import and export.

The blocks are defined in a Json array. The toolbox is a second Json array which places the blocks into categories.

There are a number of functions for managing the workspace, and functions to send and receive the code to and from the C# component.

### Tool editor – tool.js

```
1. let workspace = null;
2.
3. function sendDataToWinForm() {
4.     const state = JSON.stringify(Blockly.serialization.workspaces.save(workspace));
5.     window.chrome.webview.postMessage(state);
6. }
7.
8. function sendTranslatedCode() {
9.     const code = 'public override void SetDefaults() {' +
JSON.stringify(Blockly.JavaScript.workspaceToCode(workspace)).slice(0, -1).slice(1);
10.    window.chrome.webview.postMessage(code);
11. }
12. }
13.
14. function redo() {
15.     workspace.redo();
16. }
17.
18. function clear() {
19.     workspace.clear();
20. }
21.
22. function clear() {
23.     workspace.clear();
24. }
25.
26. function loadData(theData) {
27.     var json = null;
28.     json = JSON.parse(theData);
29.     try {
30.         Blockly.serialization.workspaces.load(json, workspace);
31.     }
32.     catch { }
33. }
34.
35. const toolbox = {
36.     "kind": "categoryToolbox",
37.     "contents": [
38.         {
39.             "kind": "category",
40.             "name": "Basic",
41.             "colour": 230,
42.             "contents": [
43.                 {
44.                     "kind": "block",
45.                     "type": "define_item"
46.                 },
47.                 {
```

```

48.         "kind": "block",
49.         "type": "define_tool"
50.     },
51.     {
52.         "kind": "block",
53.         "type": "define_weapon_essential"
54.     },
55.     {
56.         "kind": "block",
57.         "type": "max_stack"
58.     }
59. ]
60. },
61. {
62.     "kind": "category",
63.     "name": "Tool",
64.     "colour": 60,
65.     "contents": [
66.         {
67.             "kind": "block",
68.             "type": "tool_power"
69.         },
70.     ]
71. },
72. {
73.     "kind": "category",
74.     "name": "Projectile",
75.     "colour": 360,
76.     "contents": [
77.         {
78.             "kind": "block",
79.             "type": "shoot_existing_ammo"
80.         },
81.         {
82.             "kind": "block",
83.             "type": "use_custom_projectile"
84.         }
85.     ]
86. },
87. {
88.     "kind": "category",
89.     "name": "Other",
90.     "colour": 120,
91.     "contents": [
92.         {
93.             "kind": "block",
94.             "type": "use_mana"
95.         },
96.         {
97.             "kind": "block",
98.             "type": "is_consumable"
99.         },
100.        {
101.            "kind": "block",
102.            "type": "no_melee"
103.        },
104.        {
105.            "kind": "block",
106.            "type": "hit_effect"
107.        }
108.    ]
109. },
110. {
111.     "kind": "category",
112.     "name": "Buffs",

```

```

113.         "colour": 330,
114.         "contents": [
115.             {
116.                 "kind": "block",
117.                 "type": "grant_ability"
118.             },
119.             {
120.                 "kind": "block",
121.                 "type": "change_class_stat"
122.             },
123.             {
124.                 "kind": "block",
125.                 "type": "set_class_stat"
126.             },
127.             {
128.                 "kind": "block",
129.                 "type": "change_player_stat"
130.             },
131.             {
132.                 "kind": "block",
133.                 "type": "set_player_stat"
134.             },
135.             {
136.                 "kind": "block",
137.                 "type": "create_wings"
138.             },
139.             {
140.                 "kind": "block",
141.                 "type": "wing_hover"
142.             },
143.             {
144.                 "kind": "block",
145.                 "type": "grant_effect"
146.             }
147.         ]
148.     },
149.     {
150.         "kind": "category",
151.         "name": "Clothes",
152.         "colour": 165,
153.         "contents": [
154.             {
155.                 "kind": "block",
156.                 "type": "equip_slot"
157.             }
158.         ]
159.     },
160.     {
161.         "kind": "category",
162.         "name": "Blocks",
163.         "colour": 360,
164.         "contents": [
165.             {
166.                 "kind": "block",
167.                 "type": "create_tile"
168.             }
169.         ]
170.     },
171.     {
172.         "kind": "category",
173.         "name": "Summoning",
174.         "colour": 80,
175.         "contents": [
176.             {
177.                 "kind": "block",

```

```

178.         "type": "spawn_enemy"
179.     }
180. ]
181. }
182. ]
183. };
184.
185. Blockly.common.defineBlocksWithJsonArray([
186.     {
187.         "type": "define_weapon_essential",
188.         "message0": "Deal %1 damage %2 Deal %3 damage %4 Deal %5 knockback %6 Have a %7 %
chance of a critical hit",
189.         "args0": [
190.             {
191.                 "type": "field_number",
192.                 "name": "damage",
193.                 "value": 0,
194.                 "precision": 1,
195.                 "min": 0,
196.                 "max": 2147483646
197.             },
198.             {
199.                 "type": "input_dummy"
200.             },
201.             {
202.                 "type": "field_dropdown",
203.                 "name": "damageType",
204.                 "options": [
205.                     [
206.                         "melee",
207.                         "Melee"
208.                     ],
209.                     [
210.                         "ranged",
211.                         "Ranged"
212.                     ],
213.                     [
214.                         "magic",
215.                         "Magic"
216.                     ],
217.                     [
218.                         "summon",
219.                         "Summon"
220.                     ],
221.                     [
222.                         "generic",
223.                         "Generic"
224.                     ]
225.                 ]
226.             },
227.             {
228.                 "type": "input_dummy"
229.             },
230.             {
231.                 "type": "field_number",
232.                 "name": "knockback",
233.                 "value": 0,
234.                 "precision": 1,
235.                 "min": 0,
236.                 "max": 2147483646
237.             },
238.             {
239.                 "type": "input_dummy"
240.             },
241.             {

```



```

242.         "type": "field_number",
243.         "name": "crit",
244.         "value": 0,
245.         "precision": 1,
246.         "min": 0,
247.         "max": 2147483646
248.     }
249. ],
250.     "previousStatement": null,
251.     "nextStatement": null,
252.     "colour": 230,
253.     "tooltip": "Defines a basic weapon",
254.     "helpUrl": ""
255. },
256. {
257.     "type": "define_tool",
258.     "message0": "Take %1 ticks to use %2 Use %3 animation when used %4 Use sound ID %5 %6
Automatically reuse %7",
259.     "args0": [
260.         {
261.             "type": "field_number",
262.             "name": "useTime",
263.             "value": 1,
264.             "precision": 1,
265.             "min": 1,
266.             "max": 2147483646
267.         },
268.         {
269.             "type": "input_dummy"
270.         },
271.         {
272.             "type": "field_dropdown",
273.             "name": "useStyle",
274.             "options": [
275.                 [
276.                     "null (item is unusable)",
277.                     "0"
278.                 ],
279.                 [
280.                     "swinging",
281.                     "1"
282.                 ],
283.                 [
284.                     "drinking",
285.                     "2"
286.                 ],
287.                 [
288.                     "thrusting",
289.                     "3"
290.                 ],
291.                 [
292.                     "holding up",
293.                     "4"
294.                 ],
295.                 [
296.                     "shooting",
297.                     "5"
298.                 ],
299.                 [
300.                     "long drinking",
301.                     "6"
302.                 ],
303.                 [
304.                     "eating",
305.                     "7"

```

```

306.         ],
307.         [
308.             "golf swinging",
309.             "8"
310.         ],
311.         [
312.             "drinking liquid",
313.             "9"
314.         ],
315.         [
316.             "hidden",
317.             "10"
318.         ],
319.         [
320.             "mowing the lawn",
321.             "11"
322.         ],
323.         [
324.             "guitar",
325.             "12"
326.         ],
327.         [
328.             "stabbing",
329.             "13"
330.         ],
331.         [
332.             "raising",
333.             "14"
334.         ]
335.     ]
336. },
337. {
338.     "type": "input_dummy"
339. },
340. {
341.     "type": "field_number",
342.     "name": "UseSound",
343.     "value": 1,
344.     "min": 1,
345.     "max": 172,
346.     "precision": 1
347. },
348. {
349.     "type": "input_dummy"
350. },
351. {
352.     "type": "field_checkbox",
353.     "name": "autoReuse",
354.     "checked": true
355. }
356. ],
357. "previousStatement": null,
358. "nextStatement": null,
359. "colour": 230,
360. "tooltip": "Defines how a weapon is used",
361. "helpUrl": "https://terraria.fandom.com/wiki/Sound_IDs"
362. },
363. {
364.     "type": "define_item",
365.     "message0": "Set value to %1 copper %2 Have %3 rarity",
366.     "args0": [
367.         {
368.             "type": "field_number",
369.             "name": "value",
370.             "value": 0,

```

```

371.         "precision": 1,
372.         "min": 0,
373.         "max": 2147483646
374.     },
375.     {
376.         "type": "input_dummy"
377.     },
378.     {
379.         "type": "field_dropdown",
380.         "name": "rare",
381.         "options": [
382.             [
383.                 "Grey",
384.                 "-1"
385.             ],
386.             [
387.                 "White",
388.                 "0"
389.             ],
390.             [
391.                 "Blue",
392.                 "1"
393.             ],
394.             [
395.                 "Green",
396.                 "2"
397.             ],
398.             [
399.                 "Orange",
400.                 "3"
401.             ],
402.             [
403.                 "Light Red",
404.                 "4"
405.             ],
406.             [
407.                 "Pink",
408.                 "5"
409.             ],
410.             [
411.                 "Light Purple",
412.                 "6"
413.             ],
414.             [
415.                 "Lime",
416.                 "7"
417.             ],
418.             [
419.                 "Yellow",
420.                 "8"
421.             ],
422.             [
423.                 "Cyan",
424.                 "9"
425.             ],
426.             [
427.                 "Red",
428.                 "10"
429.             ],
430.             [
431.                 "Purple",
432.                 "11"
433.             ],
434.             [
435.                 "Rainbow",

```

```

436.         "-12"
437.     ],
438.     [
439.         "Fiery Red",
440.         "-13"
441.     ],
442.     [
443.         "Amber",
444.         "-11"
445.     ]
446. ]
447. }
448. ],
449. "nextStatement": null,
450. "colour": 230,
451. "tooltip": "Defines a basic item",
452. "helpUrl": "no"
453. },
454. {
455.     "type": "tool_power",
456.     "message0": "Set %2 power to %1",
457.     "args0": [
458.         {
459.             "type": "field_number",
460.             "name": "power",
461.             "value": 0,
462.             "precision": 1,
463.             "min": 0,
464.             "max": 2147483646
465.         },
466.         {
467.             "type": "field_dropdown",
468.             "name": "tool_type",
469.             "options": [
470.                 [
471.                     "pickaxe",
472.                     "pick"
473.                 ],
474.                 [
475.                     "axe",
476.                     "axe"
477.                 ],
478.                 [
479.                     "hammer",
480.                     "hammer"
481.                 ],
482.                 [
483.                     "fishing",
484.                     "fishing"
485.                 ]
486.             ]
487.         }
488.     ],
489.     "previousStatement": null,
490.     "nextStatement": null,
491.     "colour": 60,
492.     "tooltip": "Defines the item's power as a tool",
493.     "helpUrl": ""
494. },
495. {
496.     "type": "use_mana",
497.     "message0": "Consume %1 mana per use",
498.     "args0": [
499.         {
500.             "type": "field_number",

```

```

501.         "name": "useMana",
502.         "value": 0,
503.         "precision": 1,
504.         "min": 0,
505.         "max": 2147483646
506.     }
507. ],
508. "previousStatement": null,
509. "nextStatement": null,
510. "colour": 120,
511. "tooltip": "Defines if an item uses mana",
512. "helpUrl": ""
513. },
514. {
515.     "type": "no_melee",
516.     "message0": "Deal contact damage %1",
517.     "args0": [
518.         {
519.             "type": "field_checkbox",
520.             "name": "melee",
521.             "checked": true
522.         }
523.     ],
524.     "previousStatement": null,
525.     "nextStatement": null,
526.     "colour": 120,
527.     "tooltip": "Defines if an item deals melee damage",
528.     "helpUrl": ""
529. },
530. {
531.     "type": "shoot_existing_ammo",
532.     "message0": "Fire %1 with %2 velocity",
533.     "args0": [
534.         {
535.             "type": "field_dropdown",
536.             "name": "ammo_type",
537.             "options": [
538.                 [
539.                     "bullets",
540.                     "Bullet"
541.                 ],
542.                 [
543.                     "arrows",
544.                     "Arrow"
545.                 ],
546.                 [
547.                     "rockets",
548.                     "Rocket"
549.                 ],
550.                 [
551.                     "darts",
552.                     "Dart"
553.                 ]
554.             ]
555.         },
556.         {
557.             "type": "field_number",
558.             "name": "shoot_speed",
559.             "value": 0,
560.             "precision": 1,
561.             "min": 0,
562.             "max": 2147483646
563.         }
564.     ],
565.     "previousStatement": null,

```

```

566.     "nextStatement": null,
567.     "colour": 0,
568.     "tooltip": "Allows the item to fire an existing ammo type",
569.     "helpUrl": ""
570. },
571. {
572.     "type": "change_class_stat",
573.     "message0": "Increase %1 for %2 by %3",
574.     "args0": [
575.         {
576.             "type": "field_dropdown",
577.             "name": "stat",
578.             "options": [
579.                 [
580.                     "Crit Chance",
581.                     "GetCritChance"
582.                 ],
583.                 [
584.                     "Damage",
585.                     "GetDamage"
586.                 ],
587.                 [
588.                     "Attack Speed",
589.                     "GetAttackSpeed"
590.                 ],
591.                 [
592.                     "Armor Penetration",
593.                     "GetArmorPenetration"
594.                 ],
595.                 [
596.                     "Knockback",
597.                     "GetKnockback"
598.                 ]
599.             ]
600.         },
601.         {
602.             "type": "field_dropdown",
603.             "name": "class_name",
604.             "options": [
605.                 [
606.                     "Melee",
607.                     "Melee"
608.                 ],
609.                 [
610.                     "Ranged",
611.                     "Ranged"
612.                 ],
613.                 [
614.                     "Magic",
615.                     "Magic"
616.                 ],
617.                 [
618.                     "Summon",
619.                     "Summon"
620.                 ],
621.                 [
622.                     "Generic",
623.                     "Generic"
624.                 ]
625.             ]
626.         },
627.         {
628.             "type": "field_number",
629.             "name": "value",
630.             "value": 0,

```

```

631.         "precision": 1,
632.         "min": 0,
633.         "max": 2147483646
634.     }
635. ],
636. "previousStatement": null,
637. "nextStatement": null,
638. "colour": 330,
639. "tooltip": "Increases a specific class' stat by the given value",
640. "helpUrl": ""
641. },
642. {
643.     "type": "set_class_stat",
644.     "message0": "Set %1 for %2 to %3",
645.     "args0": [
646.         {
647.             "type": "field_dropdown",
648.             "name": "stat",
649.             "options": [
650.                 [
651.                     "Crit Chance",
652.                     "GetCritChance"
653.                 ],
654.                 [
655.                     "Damage",
656.                     "GetDamage"
657.                 ],
658.                 [
659.                     "Attack Speed",
660.                     "GetAttackSpeed"
661.                 ],
662.                 [
663.                     "Armor Penetration",
664.                     "GetArmorPenetration"
665.                 ],
666.                 [
667.                     "Knockback",
668.                     "GetKnockback"
669.                 ]
670.             ]
671.         },
672.         {
673.             "type": "field_dropdown",
674.             "name": "class_name",
675.             "options": [
676.                 [
677.                     "Melee",
678.                     "Melee"
679.                 ],
680.                 [
681.                     "Ranged",
682.                     "Ranged"
683.                 ],
684.                 [
685.                     "Magic",
686.                     "Magic"
687.                 ],
688.                 [
689.                     "Summon",
690.                     "Summon"
691.                 ],
692.                 [
693.                     "Generic",
694.                     "Generic"
695.                 ]

```

```

696.         ]
697.     },
698.     {
699.         "type": "field_number",
700.         "name": "value",
701.         "value": 0,
702.         "precision": 1,
703.         "min": 0,
704.         "max": 2147483646
705.     }
706. ],
707. "previousStatement": null,
708. "nextStatement": null,
709. "colour": 330,
710. "tooltip": "Sets a specific class' stat to the given value",
711. "helpUrl": ""
712. },
713. {
714.     "type": "grant_ability",
715.     "message0": "Grant player %1",
716.     "args0": [
717.         {
718.             "type": "field_dropdown",
719.             "name": "ability",
720.             "options": [
721.                 [
722.                     "damage immunity",
723.                     "immune"
724.                 ],
725.                 [
726.                     "gravity control",
727.                     "gravControl"
728.                 ],
729.                 [
730.                     "fire block immunity",
731.                     "fireWalk"
732.                 ],
733.                 [
734.                     "fall damage immunity",
735.                     "noFallDmg"
736.                 ],
737.                 [
738.                     "jump boost",
739.                     "jumpBoost"
740.                 ],
741.                 [
742.                     "knockback immunity",
743.                     "noKnockback"
744.                 ],
745.                 [
746.                     "water walking",
747.                     "waterWalk"
748.                 ],
749.                 [
750.                     "thorns",
751.                     "thorns"
752.                 ],
753.                 [
754.                     "lava immunity",
755.                     "lavaImmune"
756.                 ],
757.                 [
758.                     "invisibility",
759.                     "invis"
760.                 ],

```



```

761.         [
762.             "reduced potion sickness",
763.             "pStone"
764.         ],
765.         [
766.             "increased invincibility duration",
767.             "longInvince"
768.         ]
769.     ]
770. }
771. ],
772. "previousStatement": null,
773. "nextStatement": null,
774. "colour": 330,
775. "tooltip": "Grant the player various abilities",
776. "helpUrl": ""
777. },
778. {
779.     "type": "change_player_stat",
780.     "message0": "Increase %1 by %2",
781.     "args0": [
782.         {
783.             "type": "field_dropdown",
784.             "name": "stat",
785.             "options": [
786.                 [
787.                     "defence",
788.                     "statDefense"
789.                 ],
790.                 [
791.                     "maximum life",
792.                     "statLifeMax2"
793.                 ],
794.                 [
795.                     "movement speed",
796.                     "moveSpeed"
797.                 ],
798.                 [
799.                     "mana costs",
800.                     "manaCost"
801.                 ],
802.                 [
803.                     "maximum mana",
804.                     "statManaMax2"
805.                 ],
806.                 [
807.                     "mining time",
808.                     "pickSpeed"
809.                 ],
810.                 [
811.                     "rocket boot flight time",
812.                     "rocketTime"
813.                 ],
814.                 [
815.                     "wing flight time",
816.                     "wingTime"
817.                 ],
818.                 [
819.                     "immunity time",
820.                     "immuneTime"
821.                 ]
822.             ]
823.         },
824.         {
825.             "type": "field_number",

```

```

826.         "name": "value",
827.         "value": 0,
828.         "precision": 1,
829.         "min": 0,
830.         "max": 2147483646
831.     }
832. ],
833.     "previousStatement": null,
834.     "nextStatement": null,
835.     "colour": 330,
836.     "tooltip": "Change various stats for the player",
837.     "helpUrl": ""
838. },
839. {
840.     "type": "set_player_stat",
841.     "message0": "Set %1 to %2",
842.     "args0": [
843.         {
844.             "type": "field_dropdown",
845.             "name": "stat",
846.             "options": [
847.                 [
848.                     "maximum life",
849.                     "statLifeMax2"
850.                 ],
851.                 [
852.                     "movement speed",
853.                     "moveSpeed"
854.                 ],
855.                 [
856.                     "mana costs",
857.                     "manaCost"
858.                 ],
859.                 [
860.                     "maximum mana",
861.                     "statManaMax2"
862.                 ],
863.                 [
864.                     "mining time",
865.                     "pickSpeed"
866.                 ],
867.                 [
868.                     "rocket boot flight time",
869.                     "rocketTime"
870.                 ],
871.                 [
872.                     "wing flight time",
873.                     "wingTime"
874.                 ],
875.                 [
876.                     "immunity time",
877.                     "immuneTime"
878.                 ]
879.             ]
880.         },
881.         {
882.             "type": "field_number",
883.             "name": "value",
884.             "value": 0,
885.             "precision": 1,
886.             "min": 0,
887.             "max": 2147483646
888.         }
889.     ],
890.     "previousStatement": null,

```

```

891.     "nextStatement": null,
892.     "colour": 330,
893.     "tooltip": "Sets various stats for the player",
894.     "helpUrl": ""
895. },
896. {
897.     "type": "is_consumable",
898.     "message0": "Consume on use %1",
899.     "args0": [
900.         {
901.             "type": "field_checkbox",
902.             "name": "consumable",
903.             "checked": true
904.         }],
905.     "previousStatement": null,
906.     "nextStatement": null,
907.     "colour": 120,
908.     "tooltip": "Defines if an item is consumed on use",
909.     "helpUrl": ""
910. },
911. {
912.     "type": "create_wings",
913.     "message0": "Create wings with flight time %1 %2 flight speed %3 , %4 and acceleration
914. %5",
915.     "args0": [
916.         {
917.             "type": "field_number",
918.             "name": "flight_time",
919.             "value": 180,
920.             "precision": 1,
921.             "min": 0,
922.             "max": 2147483646
923.         },
924.         {
925.             "type": "input_dummy"
926.         },
927.         {
928.             "type": "field_number",
929.             "name": "flight_speed",
930.             "value": 9,
931.             "precision": 1,
932.             "min": 0,
933.             "max": 2147483646
934.         },
935.         {
936.             "type": "input_dummy"
937.         },
938.         {
939.             "type": "field_number",
940.             "name": "acceleration",
941.             "value": 2.5,
942.             "min": 0,
943.             "max": 2147483646
944.         }],
945.     "previousStatement": null,
946.     "nextStatement": null,
947.     "colour": 330,
948.     "tooltip": "Sets the flight properties of the item when equipped",
949.     "helpUrl": ""
950. },
951. {
952.     "type": "wing_hover",
953.     "message0": "Allow player to hover with hover speed %1 %2 and hover acceleration %3",
954.     "args0": [

```

```

955.         {
956.             "type": "field_number",
957.             "name": "hover_speed",
958.             "value": 9,
959.             "precision": 1,
960.             "min": 0,
961.             "max": 2147483646
962.         },
963.         {
964.             "type": "input_dummy"
965.         },
966.         {
967.             "type": "field_number",
968.             "name": "acceleration",
969.             "value": 2.5,
970.             "min": 0,
971.             "max": 2147483646
972.         }
973.     ],
974.     "previousStatement": null,
975.     "nextStatement": null,
976.     "colour": 330,
977.     "tooltip": "Allows the player to hover",
978.     "helpUrl": ""
979. },
980. {
981.     "type": "use_custom_projectile",
982.     "message0": "Fire custom projectile called %1 with %2 velocity",
983.     "args0": [
984.         {
985.             "type": "field_input",
986.             "name": "projectile",
987.             "text": "projectile name"
988.         },
989.         {
990.             "type": "field_number",
991.             "name": "shoot_speed",
992.             "value": 0,
993.             "precision": 1,
994.             "min": 0,
995.             "max": 2147483646
996.         }
997.     ],
998.     "previousStatement": null,
999.     "nextStatement": null,
1000.     "colour": 0,
1001.     "tooltip": "Uses a custom projectile created by the user",
1002.     "helpUrl": ""
1003. },
1004. {
1005.     "type": "equip_slot",
1006.     "message0": "Equip to %1",
1007.     "args0": [
1008.         {
1009.             "type": "field_dropdown",
1010.             "name": "slot",
1011.             "options": [
1012.                 [
1013.                     "head",
1014.                     "Head"
1015.                 ],
1016.                 [
1017.                     "body",
1018.                     "Body"
1019.                 ]

```

```

1020.         [
1021.             "legs",
1022.             "Legs"
1023.         ]
1024.     ]
1025. }
1026. ],
1027. "previousStatement": null,
1028. "nextStatement": null,
1029. "colour": 165,
1030. "tooltip": "Makes the item display on the corresponding slot on the body",
1031. "helpUrl": "",
1032. },
1033. {
1034.     "type": "create_tile",
1035.     "message0": "Create tile called %1",
1036.     "args0": [
1037.         {
1038.             "type": "field_input",
1039.             "name": "tileName",
1040.             "text": "tile name"
1041.         }
1042.     ],
1043.     "previousStatement": null,
1044.     "nextStatement": null,
1045.     "colour": 360,
1046.     "tooltip": "Places an existing tile",
1047.     "helpUrl": "https://terraria.wiki.gg/wiki/Tile_IDs/Part1"
1048. },
1049. {
1050.     "type": "grant_effect",
1051.     "message0": "Grant player %1 effect for %2 seconds",
1052.     "args0": [
1053.         {
1054.             "type": "field_input",
1055.             "name": "effect",
1056.             "text": "buff"
1057.         },
1058.         {
1059.             "type": "field_number",
1060.             "name": "time",
1061.             "value": 0,
1062.             "precision": 1,
1063.             "min": 0,
1064.             "max": 2147483646
1065.         }
1066.     ],
1067.     "previousStatement": null,
1068.     "nextStatement": null,
1069.     "colour": 330,
1070.     "tooltip": "Grants the player a status effect",
1071.     "helpUrl": "https://terraria.wiki.gg/wiki/Buff_IDs"
1072. },
1073. {
1074.     "type": "hit_effect",
1075.     "message0": "Inflict %1 effect on hit for %2 seconds",
1076.     "args0": [
1077.         {
1078.             "type": "field_input",
1079.             "name": "effect",
1080.             "text": "buff"
1081.         },
1082.         {
1083.             "type": "field_number",
1084.             "name": "time",

```

```

1085.         "value": 0,
1086.         "precision": 1,
1087.         "min": 0,
1088.         "max": 2147483646
1089.     }
1090. ],
1091. "previousStatement": null,
1092. "nextStatement": null,
1093. "colour": 120,
1094. "tooltip": "Inflicts a buff or debuff on hit",
1095. "helpUrl": "https://terraria.wiki.gg/wiki/Buff_IDs"
1096. },
1097. {
1098.     "type": "spawn_enemy",
1099.     "message0": "Spawn %1 on player",
1100.     "args0": [
1101.         {
1102.             "type": "field_input",
1103.             "name": "enemy_name",
1104.             "text": "enemy name"
1105.         }
1106.     ],
1107.     "previousStatement": null,
1108.     "nextStatement": null,
1109.     "colour": 80,
1110.     "tooltip": "Spawns an enemy when used",
1111.     "helpUrl": "https://terraria.wiki.gg/wiki/Data_IDs"
1112. },
1113. {
1114.     "type": "max_stack",
1115.     "message0": "Set maximum stack size to %1",
1116.     "args0": [
1117.         {
1118.             "type": "field_number",
1119.             "name": "max",
1120.             "value": 0,
1121.             "precision": 1,
1122.             "min": 1,
1123.             "max": 2147483646
1124.         }
1125.     ],
1126.     "previousStatement": null,
1127.     "nextStatement": null,
1128.     "colour": 230,
1129.     "tooltip": "Sets the number of items that will fit in one inventory slot",
1130.     "helpUrl": ""
1131. }
1132. ]);
1133.
1134. workspace = Blockly.inject('blocklyDiv', {
1135.     toolbox: toolbox,
1136.     scrollbars: true,
1137.     horizontalLayout: false,
1138.     toolboxPosition: "left",
1139. });
1140.

```

## NPC editor – npc.js

```

1. let workspace = null;
2.
3. function sendDataToWinForm() {
4.     const state = JSON.stringify(Blockly.serialization.workspaces.save(workspace));

```

```

5.     window.chrome.webview.postMessage(state);
6. }
7.
8. function sendTranslatedCode() {
9.     const code = 'public override void SetDefaults() {' +
JSON.stringify(Blockly.JavaScript.workspaceToCode(workspace)).slice(0, -1).slice(1);
10.     window.chrome.webview.postMessage(code);
11.
12. }
13.
14. function undo() {
15.     workspace.undo();
16. }
17.
18. function redo() {
19.     workspace.redo();
20. }
21.
22. function clear() {
23.     workspace.clear();
24. }
25.
26. function loadData(theData) {
27.     var json = null;
28.     json = JSON.parse(theData);
29.     try {
30.         Blockly.serialization.workspaces.load(json, workspace);
31.     }
32.     catch { }
33. }
34.
35.
36.
37. const toolbox = {
38.     kind: 'categoryToolbox',
39.     contents: [
40.         {
41.             "kind": "category",
42.             "name": "Basic",
43.             "colour": 300,
44.             "contents": [
45.                 {
46.                     "kind": "block",
47.                     "type": "npc_basic"
48.                 },
49.                 {
50.                     "kind": "block",
51.                     "type": "use_npc_ai"
52.                 },
53.                 {
54.                     "kind": "block",
55.                     "type": "set_spawn_rate"
56.                 },
57.                 {
58.                     "kind": "block",
59.                     "type": "set_spawn_condition"
60.                 },
61.                 {
62.                     "kind": "block",
63.                     "type": "add_loot_drop"
64.                 },
65.                 {
66.                     "kind": "block",
67.                     "type": "set_npc_property"
68.                 },

```

```

69.         {
70.             "kind": "block",
71.             "type": "set_flavour_text"
72.         }
73.     ],
74. },
75. {
76.     "kind": "category",
77.     "name": "Friendly",
78.     "colour": 250,
79.     "contents": [
80.         {
81.             "kind": "block",
82.             "type": "chat_option"
83.         },
84.         {
85.             "kind": "block",
86.             "type": "npc_friendly"
87.         },
88.         {
89.             "kind": "block",
90.             "type": "add_shop_item"
91.         }
92.     ]
93. },
94. {
95.     "kind": "category",
96.     "name": "Enemy",
97.     "colour": 170,
98.     "contents": [
99.         {
100.             "kind": "block",
101.             "type": "is_boss"
102.         },
103.         {
104.             "kind": "block",
105.             "type": "set_boss_value"
106.         },
107.         {
108.             "kind": "block",
109.             "type": "drop_potion"
110.         }
111.     ]
112. }
113. ]
114. }
115.
116. Blockly.common.defineBlocksWithJsonArray([
117.     {
118.         "type": "npc_basic",
119.         "message0": "Deal %1 damage on hit %2 Set defence to %3 %4 Set maximum life to %5 %6
Take %7 knockback",
120.         "args0": [
121.             {
122.                 "type": "field_number",
123.                 "name": "damage",
124.                 "value": 0,
125.                 "precision": 1,
126.                 "min": 0,
127.                 "max": 2147483646
128.             },
129.             {
130.                 "type": "input_dummy"
131.             },
132.             {

```



```

133.         "type": "field_number",
134.         "name": "defense",
135.         "value": 1,
136.         "precision": 1,
137.         "min": 0,
138.         "max": 2147483646
139.     },
140.     {
141.         "type": "input_dummy"
142.     },
143.     {
144.         "type": "field_number",
145.         "name": "life",
146.         "value": 0,
147.         "precision": 1,
148.         "min": 0,
149.         "max": 2147483646
150.     },
151.     {
152.         "type": "input_dummy"
153.     },
154.     {
155.         "type": "field_number",
156.         "name": "knockResist",
157.         "value": 0,
158.         "min": 0,
159.         "max": 2147483646
160.     }
161. ],
162. "nextStatement": null,
163. "colour": 300,
164. "tooltip": "Sets the basic properties of an NPC",
165. "helpUrl": ""
166. },
167. {
168.     "type": "use_npc_ai",
169.     "message0": "Use %1 AI style",
170.     "args0": [
171.         {
172.             "type": "field_dropdown",
173.             "name": "style",
174.             "options": [
175.                 [
176.                     "Null",
177.                     "0"
178.                 ],
179.                 [
180.                     "Slime",
181.                     "1"
182.                 ],
183.                 [
184.                     "Simple Flier",
185.                     "2"
186.                 ],
187.                 [
188.                     "Simple Fighter",
189.                     "3"
190.                 ],
191.                 [
192.                     "Friendly NPC",
193.                     "7"
194.                 ],
195.                 [
196.                     "Teleporting Spellcaster",
197.                     "8"

```

```

198.         ],
199.         [
200.             "General Flier",
201.             "14"
202.         ],
203.         [
204.             "King Slime",
205.             "15"
206.         ],
207.         [
208.             "Fish",
209.             "16"
210.         ],
211.         [
212.             "Vulture",
213.             "17"
214.         ],
215.         [
216.             "Jellyfish",
217.             "18"
218.         ],
219.         [
220.             "Antlion",
221.             "19"
222.         ],
223.         [
224.             "Animated Weapons",
225.             "23"
226.         ],
227.         [
228.             "Birds",
229.             "24"
230.         ],
231.         [
232.             "Mimic",
233.             "25"
234.         ],
235.         [
236.             "Unicorn",
237.             "26"
238.         ],
239.         [
240.             "Snowman",
241.             "38"
242.         ],
243.     ],
244.     ]
245. }
246. ],
247. "nextStatement": null,
248. "previousStatement": null,
249. "colour": 300,
250. "tooltip": "Defines the way in which the NPC moves",
251. "helpUrl": ""
252. },
253. {
254.     "type": "chat_option",
255.     "message0": "Add chat option: %1",
256.     "args0": [
257.         {
258.             "type": "field_input",
259.             "name": "chat",
260.             "text": "Hello World!"
261.         }
262.     ],

```

```

263.     "previousStatement": null,
264.     "nextStatement": null,
265.     "colour": 250,
266.     "tooltip": "Adds a phrase to the list of things the NPC can say",
267.     "helpUrl": ""
268. },
269. {
270.     "type": "fire_projectile",
271.     "message0": "Fire projectile called %1",
272.     "args0": [
273.         {
274.             "type": "field_input",
275.             "name": "projectile",
276.             "text": "default"
277.         }
278.     ],
279.     "previousStatement": null,
280.     "nextStatement": null,
281.     "colour": 230,
282.     "tooltip": "",
283.     "helpUrl": ""
284. },
285. {
286.     "type": "set_spawn_rate",
287.     "message0": "Set spawn rate to %1",
288.     "args0": [
289.         {
290.             "type": "field_number",
291.             "name": "rate",
292.             "value": 0,
293.             "min": 0,
294.             "max": 2147483646
295.         }
296.     ],
297.     "previousStatement": null,
298.     "nextStatement": null,
299.     "colour": 300,
300.     "tooltip": "Sets a flat rate at which the NPC will spawn",
301.     "helpUrl": ""
302. },
303. {
304.     "type": "set_spawn_condition",
305.     "message0": "Use %1 spawn condition with a %2 times multiplier",
306.     "args0": [
307.         {
308.             "type": "field_dropdown",
309.             "name": "condition",
310.             "options": [
311.                 [
312.                     "BoundCaveNPC",
313.                     "BoundCaveNPC"
314.                 ],
315.                 [
316.                     "Cavern",
317.                     "Cavern"
318.                 ],
319.                 [
320.                     "Corruption",
321.                     "Corruption"
322.                 ],
323.                 [
324.                     "Crimson",
325.                     "Crimson"
326.                 ],
327.                 [

```

```

328.         "Default Water Critter",
329.         "DefaultWaterCriticter"
330.     ],
331.     [
332.         "Desert Cave",
333.         "DesertCave"
334.     ],
335.     [
336.         "Dungeon",
337.         "Dungeon"
338.     ],
339.     [
340.         "Jungle Temple",
341.         "JungleTemple"
342.     ],
343.     [
344.         "Ocean",
345.         "Ocean"
346.     ],
347.     [
348.         "Overworld",
349.         "Overworld"
350.     ],
351.     [
352.         "Overworld Day",
353.         "OverworldDay"
354.     ],
355.     [
356.         "Overworld Hallow",
357.         "OverworldHallow"
358.     ],
359.     [
360.         "Overworld Night",
361.         "OverworldNight"
362.     ],
363.     [
364.         "Overworld Night Monster",
365.         "OverworldNightMonster"
366.     ],
367.     [
368.         "Sky",
369.         "Sky"
370.     ],
371.     [
372.         "Solar Eclipse",
373.         "SolarEclipse"
374.     ],
375.     [
376.         "Underground",
377.         "Underground"
378.     ],
379.     [
380.         "Underworld",
381.         "Underworld"
382.     ]
383. ],
384. },
385. {
386.     "type": "field_number",
387.     "name": "multiplier",
388.     "value": 0,
389.     "min": 0,
390.     "max": 2147483646
391. },
392. ],

```

```

393.     "previousStatement": null,
394.     "nextStatement": null,
395.     "colour": 300,
396.     "tooltip": "Sets the spawn rate to a given condition",
397.     "helpUrl": ""
398. },
399. {
400.     "type": "npc_friendly",
401.     "message0": "Make NPC friendly %1",
402.     "args0": [
403.         {
404.             "type": "field_checkbox",
405.             "name": "friendly",
406.             "checked": true
407.         }
408.     ],
409.     "previousStatement": null,
410.     "nextStatement": null,
411.     "colour": 250,
412.     "tooltip": "Stops the NPC from damaging the player",
413.     "helpUrl": ""
414. },
415. {
416.     "type": "add_shop_item",
417.     "message0": "Add %1 to shop",
418.     "args0": [
419.         {
420.             "type": "field_input",
421.             "name": "item",
422.             "text": "item name"
423.         }
424.     ],
425.     "previousStatement": null,
426.     "nextStatement": null,
427.     "colour": 250,
428.     "tooltip": "Adds an item to the NPC's shop",
429.     "helpUrl": "https://terraria.fandom.com/wiki/Item_IDs"
430. },
431. {
432.     "type": "add_loot_drop",
433.     "message0": "Drop %1 to %2 %3 with a %4 in %5 chance",
434.     "args0": [
435.         {
436.             "type": "field_number",
437.             "name": "min",
438.             "value": 0,
439.             "precision": 1,
440.             "min": 0,
441.             "max": 2147483646
442.         },
443.         {
444.             "type": "field_number",
445.             "name": "max",
446.             "value": 0,
447.             "precision": 1,
448.             "min": 0,
449.             "max": 2147483646
450.         },
451.         {
452.             "type": "field_input",
453.             "name": "item",
454.             "text": "item"
455.         },
456.         {
457.             "type": "field_number",

```

```

458.         "name": "numerator",
459.         "value": 0,
460.         "precision": 1,
461.         "min": 0,
462.         "max": 2147483646
463.     },
464.     {
465.         "type": "field_number",
466.         "name": "denominator",
467.         "value": 0,
468.         "precision": 1,
469.         "min": 0,
470.         "max": 2147483646
471.     }
472. ],
473. "previousStatement": null,
474. "nextStatement": null,
475. "colour": 300,
476. "tooltip": "Gives the NPC a chance to drop an item on death",
477. "helpUrl": "https://terraria.fandom.com/wiki/Item_IDs"
478. },
479. {
480.     "type": "is_boss",
481.     "message0": "Is boss %1",
482.     "args0": [
483.         {
484.             "type": "field_checkbox",
485.             "name": "boss",
486.             "checked": true
487.         }
488.     ],
489.     "previousStatement": null,
490.     "nextStatement": null,
491.     "colour": 170,
492.     "tooltip": "Makes the NPC a boss",
493.     "helpUrl": "https://terraria.fandom.com/wiki/Item_IDs"
494. },
495. {
496.     "type": "set_boss_value",
497.     "message0": "Drop %1 copper on death",
498.     "args0": [
499.         {
500.             "type": "field_number",
501.             "name": "value",
502.             "value": 0,
503.             "precision": 1,
504.             "min": 0,
505.             "max": 2147483646
506.         }
507.     ],
508.     "previousStatement": null,
509.     "nextStatement": null,
510.     "colour": 170,
511.     "tooltip": "Sets the amount of money the NPC will drop on death",
512.     "helpUrl": ""
513. },
514. {
515.     "type": "set_npc_property",
516.     "message0": "Make NPC %1",
517.     "args0": [
518.         {
519.             "type": "field_dropdown",
520.             "name": "property",
521.             "options": [
522.                 [

```

```

523.             "immune to lava",
524.             "lavaImmune"
525.         ],
526.         [
527.             "unaffected by gravity",
528.             "noGravity"
529.         ]
530.     ]
531. }
532. ],
533. "previousStatement": null,
534. "nextStatement": null,
535. "colour": 300,
536. "tooltip": "Sets a property of the NPC",
537. "helpUrl": ""
538. },
539. {
540.     "type": "set_flavour_text",
541.     "message0": "Set bestiary flavour text to %1",
542.     "args0": [
543.         {
544.             "type": "field_input",
545.             "name": "text",
546.             "text": "text"
547.         }
548.     ],
549.     "previousStatement": null,
550.     "nextStatement": null,
551.     "colour": 300,
552.     "tooltip": "Sets flavour test for the NPC in the bestiary",
553.     "helpUrl": ""
554. },
555. {
556.     "type": "drop_potion",
557.     "message0": "Drop %1 potion on death",
558.     "args0": [
559.         {
560.             "type": "field_dropdown",
561.             "name": "potion",
562.             "options": [
563.                 [
564.                     "lesser healing",
565.                     "LesserHealingPotion"
566.                 ],
567.                 [
568.                     "healing",
569.                     "HealingPotion"
570.                 ],
571.                 [
572.                     "greater healing",
573.                     "GreaterHealingPotion"
574.                 ],
575.                 [
576.                     "super healing",
577.                     "SuperHealingPotion"
578.                 ]
579.             ]
580.         }
581.     ],
582.     "previousStatement": null,
583.     "nextStatement": null,
584.     "colour": 170,
585.     "tooltip": "Sets what potion is dropped when killed",
586.     "helpUrl": ""
587. }

```

```

588. });
589.
590.
591. workspace = Blockly.inject('blocklyDiv', {
592.   toolbox: toolbox,
593.   scrollbars: true,
594.   horizontalLayout: false,
595.   toolboxPosition: "left",
596. });
597.

```

## Projectile editor – projectile.js

```

1. let workspace = null;
2.
3. function sendDataToWinForm() {
4.   const state = JSON.stringify(Blockly.serialization.workspaces.save(workspace));
5.   window.chrome.webview.postMessage(state);
6. }
7.
8. function sendTranslatedCode() {
9.   const code = 'public override void SetDefaults() {' +
JSON.stringify(Blockly.JavaScript.workspaceToCode(workspace)).slice(0, -1).slice(1);
10.   window.chrome.webview.postMessage(code);
11. }
12. }
13.
14. function redo() {
15.   workspace.redo();
16. }
17.
18. function clear() {
19.   workspace.clear();
20. }
21.
22. function clear() {
23.   workspace.clear();
24. }
25.
26. function loadData(theData) {
27.   var json = null;
28.   json = JSON.parse(theData);
29.   try {
30.     Blockly.serialization.workspaces.load(json, workspace);
31.   }
32.   catch { }
33. }
34.
35.
36.
37. const toolbox = {
38.   kind: 'categoryToolbox',
39.   contents: [
40.     {
41.       "kind": "category",
42.       "name": "Projectile",
43.       "colour": 250,
44.       contents: [
45.         {
46.           "kind": "block",
47.           "type": "projectile_basic"
48.         },
49.         {

```



```

50.         "kind": "block",
51.         "type": "use_ai"
52.     },
53.     {
54.         "kind": "block",
55.         "type": "set_value"
56.     },
57.     {
58.         "kind": "block",
59.         "type": "declare_friendly"
60.     },
61.     {
62.         "kind": "block",
63.         "type": "declare_hostile"
64.     },
65.     {
66.         "kind": "block",
67.         "type": "hide_projectile"
68.     },
69.     {
70.         "kind": "block",
71.         "type": "collide_with_tiles"
72.     },
73.     {
74.         "kind": "block",
75.         "type": "emit_light"
76.     }
77. ]
78. }
79. ],
80. };
81.
82.
83.
84.
85. Blockly.common.defineBlocksWithJsonArray([
86.     {
87.         "type": "projectile_basic",
88.         "message0": "Last for %1 seconds",
89.         "args0": [
90.             {
91.                 "type": "field_number",
92.                 "name": "time_left",
93.                 "value": 0,
94.                 "min": 0,
95.                 "max": 2147483646
96.             }
97.         ],
98.         "nextStatement": null,
99.         "colour": 230,
100.        "tooltip": "Sets the amount of time the projectile will stay in the world for",
101.        "helpUrl": ""
102.    },
103.    {
104.        "type": "use_ai",
105.        "message0": "Use %1 AI style",
106.        "args0": [
107.            {
108.                "type": "field_dropdown",
109.                "name": "style",
110.                "options": [
111.                    [
112.                        "bullet",
113.                        ""
114.                    ],

```

```

115.      [
116.          "arrow",
117.          "1"
118.      ],
119.      [
120.          "thrown",
121.          "2"
122.      ],
123.      [
124.          "boomerang",
125.          "3"
126.      ],
127.      [
128.          "progressive fading",
129.          "4"
130.      ],
131.      [
132.          "falling star",
133.          "5"
134.      ],
135.      [
136.          "powder",
137.          "6"
138.      ],
139.      [
140.          "grapple",
141.          "7"
142.      ],
143.      [
144.          "magic bounce",
145.          "8"
146.      ],
147.      [
148.          "controlled bolt",
149.          "9"
150.      ],
151.      [
152.          "falling block",
153.          "10"
154.      ],
155.      [
156.          "follow player",
157.          "11"
158.      ],
159.      [
160.          "water stream",
161.          "12"
162.      ],
163.      [
164.          "harpoon",
165.          "13"
166.      ],
167.      [
168.          "bounce",
169.          "14"
170.      ],
171.      [
172.          "flail",
173.          "15"
174.      ],
175.      [
176.          "bounce & explode",
177.          "16"
178.      ],
179.      [

```

```

180.         "stationary",
181.         "17"
182.     ],
183.     [
184.         "exponential acceleration (with sparkles)",
185.         "18"
186.     ],
187.     [
188.         "spear",
189.         "19"
190.     ],
191.     [
192.         "motorised tool",
193.         "20"
194.     ],
195.     [
196.         "no gravity",
197.         "21"
198.     ],
199.     [
200.         "blue trail",
201.         "22"
202.     ],
203.     [
204.         "fiery",
205.         "23"
206.     ],
207.     [
208.         "shard",
209.         "24"
210.     ],
211.     [
212.         "boulder",
213.         "25"
214.     ],
215.     [
216.         "fly to player",
217.         "26"
218.     ],
219.     [
220.         "demon trident",
221.         "27"
222.     ],
223.     [
224.         "ice bolt",
225.         "28"
226.     ],
227.     [
228.         "amethyst bolt",
229.         "29"
230.     ],
231.     [
232.         "mushroom",
233.         "30"
234.     ],
235.     [
236.         "solution spray",
237.         "31"
238.     ],
239.     [
240.         "beach ball",
241.         "32"
242.     ],
243.     [
244.         "flare",

```

```

245.         "33"
246.     ],
247.     [
248.         "rocket",
249.         "34"
250.     ],
251.     [
252.         "rope idk",
253.         "35"
254.     ],
255.     [
256.         "bee",
257.         "36"
258.     ],
259.     [
260.         "spear",
261.         "37"
262.     ],
263.     [
264.         "stationary flame turret",
265.         "38"
266.     ],
267.     [
268.         "mechanical pirahna",
269.         "39"
270.     ],
271.     [
272.         "leaf",
273.         "40"
274.     ],
275.     [
276.         "flower petal",
277.         "41"
278.     ],
279.     [
280.         "crystal leaf A",
281.         "42"
282.     ],
283.     [
284.         "crystal leaf B",
285.         "43"
286.     ],
287.     [
288.         "spore cloud",
289.         "44"
290.     ],
291.     [
292.         "rain cloud",
293.         "45"
294.     ],
295.     [
296.         "rain cloud",
297.         "46"
298.     ],
299.     [
300.         "magnet sphere",
301.         "47"
302.     ],
303.     [
304.         "heat ray",
305.         "48"
306.     ],
307.     [
308.         "explosive bunny",
309.         "49"

```

```

310.      ],
311.      [
312.          "inferno",
313.          "50"
314.      ],
315.      [
316.          "lost soul",
317.          "51"
318.      ],
319.      [
320.          "spirit heal",
321.          "52"
322.      ],
323.      [
324.          "frost hydra",
325.          "53"
326.      ],
327.      [
328.          "raven",
329.          "54"
330.      ],
331.      [
332.          "flaming jack",
333.          "55"
334.      ],
335.      [
336.          "flaming scythe",
337.          "56"
338.      ],
339.      [
340.          "north pole",
341.          "57"
342.      ],
343.      [
344.          "present",
345.          "58"
346.      ],
347.      [
348.          "spectre wraith",
349.          "59"
350.      ],
351.      [
352.          "water gun",
353.          "60"
354.      ],
355.      [
356.          "bobber",
357.          "61"
358.      ],
359.      [
360.          "loop around player",
361.          "62"
362.      ],
363.      [
364.          "baby spider",
365.          "63"
366.      ],
367.      [
368.          "sharknado A",
369.          "64"
370.      ],
371.      [
372.          "sharknado B",
373.          "65"
374.      ],

```

```

375.         [
376.             "big loop",
377.             "66"
378.         ],
379.         [
380.             "return to player",
381.             "67"
382.         ],
383.         [
384.             "molotov cocktail",
385.             "68"
386.         ],
387.         [
388.             "flairon",
389.             "69"
390.         ],
391.         [
392.             "flairon bubble",
393.             "70"
394.         ],
395.         [
396.             "typhoon",
397.             "71"
398.         ],
399.         [
400.             "bubble",
401.             "72"
402.         ],
403.         [
404.             "lightning orb",
405.             "88"
406.         ]
407.     ]
408. }
409. ],
410.     },
411.     ],
412.     "previousStatement": null,
413.     "nextStatement": null,
414.     "colour": 230,
415.     "tooltip": "Sets the way the projectile acts when fired",
416.     "helpUrl": ""
417. },
418. {
419.     "type": "set_value",
420.     "message0": "Set %1 to %2",
421.     "args0": [
422.         {
423.             "type": "field_dropdown",
424.             "name": "property",
425.             "options": [
426.                 [
427.                     "damage",
428.                     "damage"
429.                 ],
430.                 [
431.                     "knockback",
432.                     "knockBack"
433.                 ],
434.                 [
435.                     "penetration",
436.                     "penetrate"
437.                 ],
438.                 [
439.                     "alpha",

```

```

440.         "alpha"
441.     ]
442. ]
443. },
444. {
445.     "type": "field_number",
446.     "name": "value",
447.     "value": 0,
448.     "min": 0,
449.     "max": 2147483646,
450.     "precision": 0
451. }
452. ],
453. "previousStatement": null,
454. "nextStatement": null,
455. "colour": 230,
456. "tooltip": "Sets a property of the projectile",
457. "helpUrl": ""
458. },
459. {
460.     "type": "declare_friendly",
461.     "message0": "Make projectile friendly %1",
462.     "args0": [
463.         {
464.             "type": "field_checkbox",
465.             "name": "friendly",
466.             "checked": true
467.         }
468.     ],
469.     "previousStatement": null,
470.     "nextStatement": null,
471.     "colour": 230,
472.     "tooltip": "Stops the projectile from damaging the player",
473.     "helpUrl": ""
474. },
475. {
476.     "type": "declare_hostile",
477.     "message0": "Make projectile hostile %1",
478.     "args0": [
479.         {
480.             "type": "field_checkbox",
481.             "name": "hostile",
482.             "checked": true
483.         }
484.     ],
485.     "previousStatement": null,
486.     "nextStatement": null,
487.     "colour": 230,
488.     "tooltip": "Makes the projectile target and damage the player",
489.     "helpUrl": ""
490. },
491. {
492.     "type": "hide_projectile",
493.     "message0": "Hide projectile %1",
494.     "args0": [
495.         {
496.             "type": "field_checkbox",
497.             "name": "hide",
498.             "checked": true
499.         }
500.     ],
501.     "previousStatement": null,
502.     "nextStatement": null,
503.     "colour": 230,
504.     "tooltip": "Makes the projectile invisible",

```

```

505.     "helpUrl": ""
506.   },
507.   {
508.     "type": "collide_with_tiles",
509.     "message0": "Collide with tiles %1",
510.     "args0": [
511.       {
512.         "type": "field_checkbox",
513.         "name": "collide",
514.         "checked": true
515.       }
516.     ],
517.     "previousStatement": null,
518.     "nextStatement": null,
519.     "colour": 230,
520.     "tooltip": "Stops the projectile from hitting blocks",
521.     "helpUrl": ""
522.   },
523.   {
524.     "type": "emit_light",
525.     "message0": "Emit %1 light while active",
526.     "args0": [
527.       {
528.         "type": "field_number",
529.         "name": "light",
530.         "value": 0,
531.         "min": 0,
532.         "max": 2147483646
533.       }
534.     ],
535.     "previousStatement": null,
536.     "nextStatement": null,
537.     "colour": 230,
538.     "tooltip": "Makes the projectile emit light",
539.     "helpUrl": ""
540.   }
541. ])
542.
543. workspace = Blockly.inject('blocklyDiv', {
544.   toolbox: toolbox,
545.   scrollbars: true,
546.   horizontalLayout: false,
547.   toolboxPosition: "left",
548. });
549.

```

## Tile editor – tile.js

```

1. let workspace = null;
2.
3. function sendDataToWinForm() {
4.   const state = JSON.stringify(Blockly.serialization.workspaces.save(workspace));
5.   window.chrome.webview.postMessage(state);
6. }
7.
8. function sendTranslatedCode() {
9.   const code = 'public override void SetDefaults() {' +
JSON.stringify(Blockly.JavaScript.workspaceToCode(workspace)).slice(0, -1).slice(1);
10.   window.chrome.webview.postMessage(code);
11. }
12. }
13.
14. function redo() {

```



```

15.     workspace.redo();
16. }
17.
18. function clear() {
19.     workspace.clear();
20. }
21.
22. function clear() {
23.     workspace.clear();
24. }
25.
26. function loadData(theData) {
27.     var json = null;
28.     json = JSON.parse(theData);
29.     try {
30.         Blockly.serialization.workspaces.load(json, workspace);
31.     }
32.     catch { }
33. }
34.
35.
36.
37. const toolbox = {
38.     kind: 'categoryToolbox',
39.     contents: [
40.         {
41.             "kind": "category",
42.             "name": "Tile",
43.             "colour": 15,
44.             contents: [
45.                 {
46.                     "kind": "block",
47.                     "type": "define_tile"
48.                 },
49.                 {
50.                     "kind": "block",
51.                     "type": "min_pickaxe"
52.                 },
53.                 {
54.                     "kind": "block",
55.                     "type": "mine_resist"
56.                 },
57.                 {
58.                     "kind": "block",
59.                     "type": "have_solid_top"
60.                 },
61.                 {
62.                     "kind": "block",
63.                     "type": "frame_important"
64.                 }
65.             ]
66.         },
67.     ],
68. };
69.
70.
71.
72.
73. Blockly.common.defineBlocksWithJsonArray([
74.     {
75.         "type": "define_tile",
76.         "message0": "Make tile solid %1 %2 Make tile merge with dirt %3 %4 Make tile block light %5 %6 Set map colour to %7",
77.         "args0": [
78.             {
79.                 "type": "field_checkbox",

```

```

79.         "name": "solid",
80.         "checked": true
81.     },
82.     {
83.         "type": "input_dummy"
84.     },
85.     {
86.         "type": "field_checkbox",
87.         "name": "mergeDirt",
88.         "checked": true
89.     },
90.     {
91.         "type": "input_dummy"
92.     },
93.     {
94.         "type": "field_checkbox",
95.         "name": "blockLight",
96.         "checked": true
97.     },
98.     {
99.         "type": "input_dummy"
100.    },
101.    {
102.        "type": "field_colour",
103.        "name": "colour",
104.        "colour": "#ff0000"
105.    }
106. ],
107. "colour": 15,
108. "tooltip": "Sets the basic properties of a block",
109. "helpUrl": "",
110. "nextStatement": null
111. },
112. {
113.     "type": "min_pickaxe",
114.     "message0": "Set minimum pickaxe power to mine to %1",
115.     "args0": [
116.         {
117.             "type": "field_number",
118.             "name": "power",
119.             "value": 0,
120.             "min": 0,
121.             "max": 2147483646,
122.             "precision": 1
123.         }
124.     ],
125.     "previousStatement": null,
126.     "nextStatement": null,
127.     "colour": 15,
128.     "tooltip": "Makes the tile only breakable with certain pickaxes",
129.     "helpUrl": ""
130. },
131. {
132.     "type": "mine_resist",
133.     "message0": "Set block mining time to %1 times the default",
134.     "args0": [
135.         {
136.             "type": "field_number",
137.             "name": "resist",
138.             "value": 0,
139.             "min": 0,
140.             "max": 2147483646
141.         }
142.     ],
143.     "previousStatement": null,

```

```

144.     "nextStatement": null,
145.     "colour": 15,
146.     "tooltip": "Sets the amount of time it takes to mine the tile",
147.     "helpUrl": ""
148.   },
149.   {
150.     "type": "have_solid_top",
151.     "message0": "Make only the top of item solid %1",
152.     "args0": [
153.       {
154.         "type": "field_checkbox",
155.         "name": "solid",
156.         "checked": true
157.       }
158.     ],
159.     "previousStatement": null,
160.     "nextStatement": null,
161.     "colour": 15,
162.     "tooltip": "Make tile behave like a platform",
163.     "helpUrl": ""
164.   },
165.   {
166.     "type": "frame_important",
167.     "message0": "Do not connect to tiles %1",
168.     "args0": [
169.       {
170.         "type": "field_checkbox",
171.         "name": "connect",
172.         "checked": true
173.       }
174.     ],
175.     "previousStatement": null,
176.     "nextStatement": null,
177.     "colour": 15,
178.     "tooltip": "Stops the tile from connecting to anything",
179.     "helpUrl": ""
180.   }
181. ])
182.
183. workspace = Blockly.inject('blocklyDiv', {
184.   toolbox: toolbox,
185.   scrollbars: true,
186.   horizontalLayout: false,
187.   toolboxPosition: "left",
188. });
189.

```

## HTML code for editors

The HTML here is very similar here, only calling different scripts.

### Tool editor – tool\_editor.html

```

1. <main>
2.   <div class="blockly-editor">
3.     <div id="blocklyDiv" style="height: 100%; width: 100%;"></div>
4.   </div>
5. </main>
6.
7. <script src="scripts/blockly.min.js"></script>
8. <script src="scripts/tool.js"></script>
9. <script src="scripts/tool_blocks.js"></script>

```

10.

## NPC editor – npc\_editor.html

```
1. <main>
2.     <div class="blockly-editor">
3.         <div id="blocklyDiv" style="height: 100%; width: 100%;"></div>
4.     </div>
5. </main>
6.
7. <script src="scripts/blockly.min.js"></script>
8. <script src="scripts/npc.js"></script>
9. <script src="scripts/tool_blocks.js"></script>
10.
```

## Projectile editor – projectile\_editor.html

```
1. <main>
2.     <div class="blockly-editor">
3.         <div id="blocklyDiv" style="height: 100%; width: 100%;"></div>
4.     </div>
5. </main>
6.
7. <script src="scripts/blockly.min.js"></script>
8. <script src="scripts/projectile.js"></script>
9. <script src="scripts/tool_blocks.js"></script>
10.
```

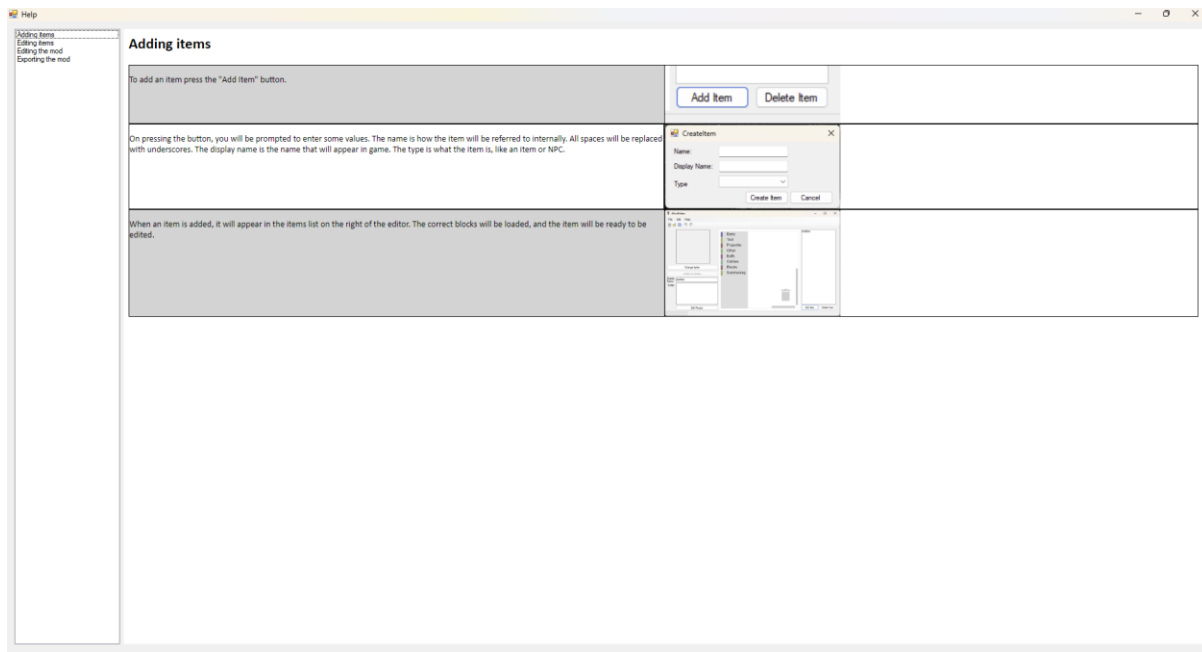
## Tile editor – tile\_editor.html

```
1. <main>
2.     <div class="blockly-editor">
3.         <div id="blocklyDiv" style="height: 100%; width: 100%;"></div>
4.     </div>
5. </main>
6.
7. <script src="scripts/blockly.min.js"></script>
8. <script src="scripts/tile.js"></script>
9. <script src="scripts/tool_blocks.js"></script>
10.
```

## Help Pages

These are various HTML pages for the help menu.

## Adding items page – AddItemHelp.html



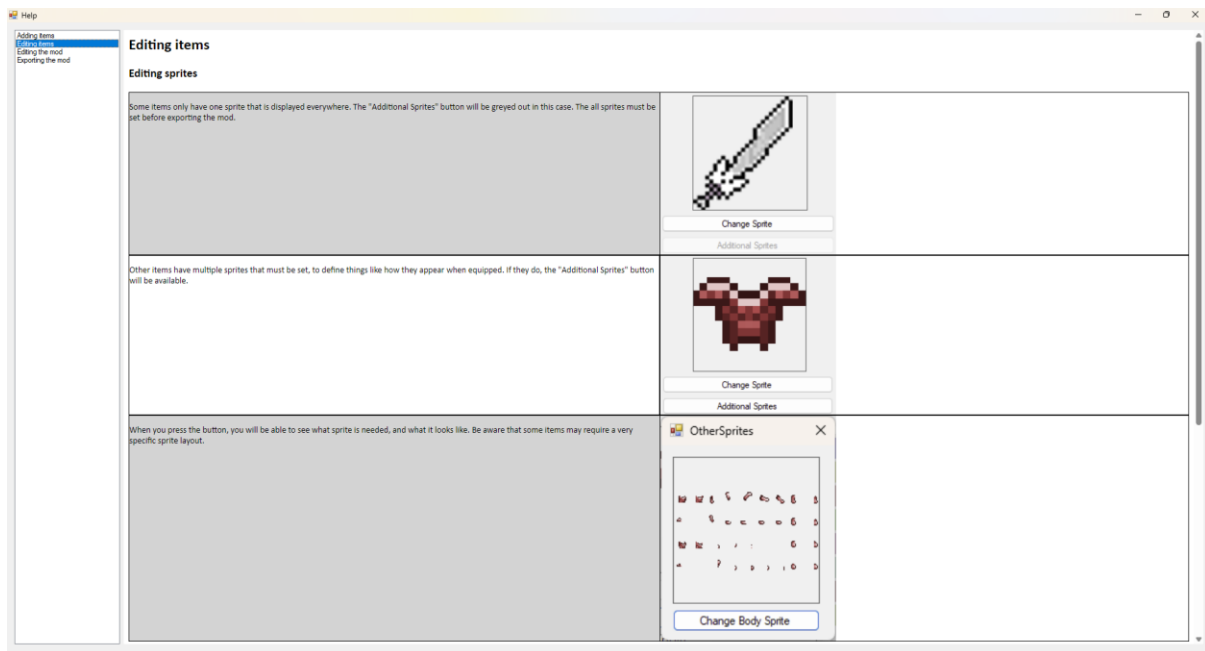
```
1. <html>
2. <head>
3.   <style>
4.     body {
5.       font-family: Calibri
6.     }
7.   }
8.   p {font-size :14px}
9. </style>
10. </head>
11. <body>
12.   <h2>Adding items</h2>
13.   <div style="width: 100%;">
14.     <div style="width: 50%; height: 92px; float: left; border: 1px solid black; background-
15.       color:lightgray">
16.       <p>To add an item press the "Add Item" button.</p>
17.     </div>
18.     <div style="margin-left: 50%; height: 92px; border: 1px solid black; ">
19.       
20.     </div>
21.   <div style="width: 100%;">
22.     <div style="width: 50%; height: 134.5px; float: left; border: 1px solid black;">
23.       <p>
24.         On pressing the button, you will be prompted to enter some values.
25.         The name is how the item will be referred to internally. All spaces will be
26.         replaced with underscores.
27.         The display name is the name that will appear in game.
28.         The type is what the item is, like an item or NPC.
29.       </p>
30.     </div>
31.     <div style="margin-left: 50%; height: 134.5px; border: 1px solid black;">
32.       
33.     </div>
34.   </div>
35. </div>
```

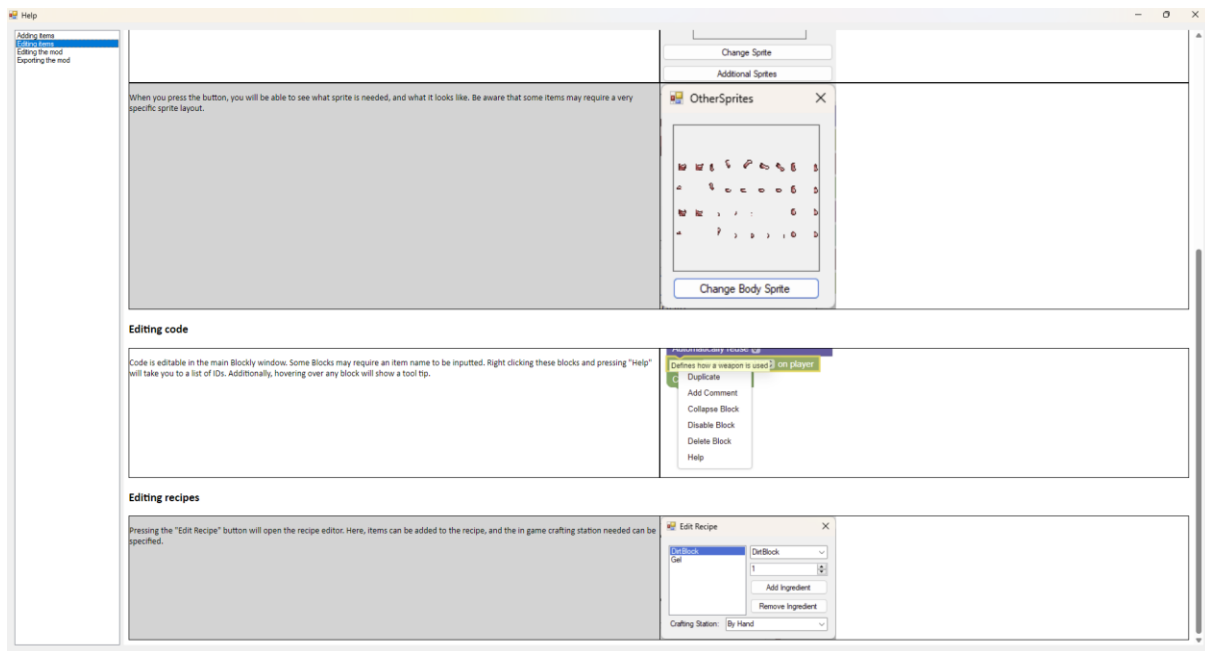
```

35.         <div style="width: 50%; height: 168.3px; float: left; border: 1px solid black;
background-color: lightgray ">
36.             <p>
37.                 When an item is added, it will appear in the items list on the right of the
editor.
38.                 The correct blocks will be loaded, and the item will be ready to be edited.
39.             </p>
40.         </div>
41.         <div style="margin-left: 50%; height: 168.3px; border: 1px solid black;">
42.             
43.         </div>
44.     </div>
45. </body>
46. </html>
47.

```

## Editing items page – EditItemHelp.html





```

1. <html>
2. <head>
3.   <style>
4.     body {
5.       font-family: Calibri
6.     }
7.
8.     p {
9.       font-size: 14px
10.    }
11.  </style>
12. </head>
13. <body>
14.   <h2>Editing items</h2>
15.   <h3>Editing sprites</h3>
16.   <div style="width: 100%;">
17.     <div style="width: 50%; height: 258.6px; float: left; background-color: lightgray;
border: 1px solid black; ">
18.       <p>
19.         Some items only have one sprite that is displayed everywhere.
20.         The "Additional Sprites" button will be greyed out in this case.
21.         The all sprites must be set before exporting the mod.
22.       </p>
23.     </div>
24.     <div style="margin-left: 50%; height: 258.6px; border: 1px solid black; ">
25.       
26.     </div>
27.     <div style="width: 50%; height: 253px; float: left; border: 1px solid black; ">
28.       <p>
29.         Other items have multiple sprites that must be set, to define things like how
they appear when equipped.
30.         If they do, the "Additional Sprites" button will be available.
31.       </p>
32.     </div>
33.     <div style="margin-left: 50%; height: 253px; border: 1px solid black; ">
34.       
35.     </div>
36.     <div style="width: 50%; height: 359.2px; float: left; background-color: lightgray;
border: 1px solid black; ">

```

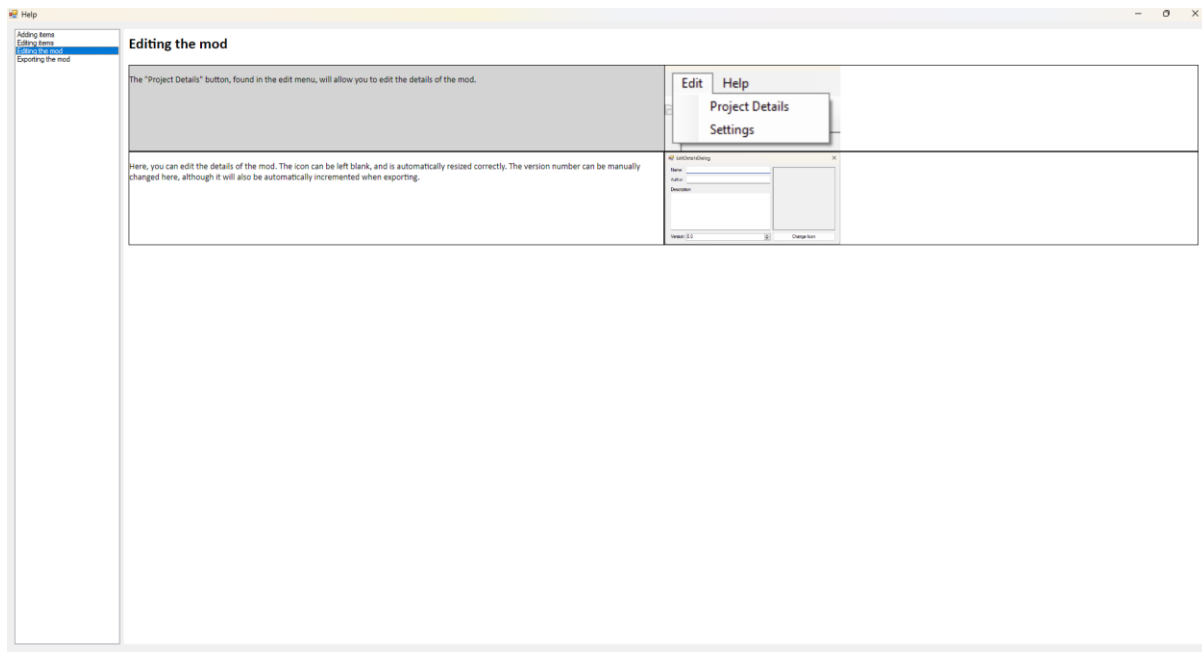
```

37.         <p>
38.             When you press the button, you will be able to see what sprite is needed, and
what it looks like.
39.             Be aware that some items may require a very specific sprite layout.
40.         </p>
41.     </div>
42.     <div style="margin-left: 50%; height: 359.2px; border: 1px solid black; ">
43.         
44.     </div>
45. </div>
46. <h3>Editing code</h3>
47. <div style="width: 100%; ">
48.     <div style="width: 50%; height: 205px; float: left; border: 1px solid black; ">
49.         <p>
50.             Code is editable in the main Blockly window.
51.             Some Blocks may require an item name to be inputted. Right clicking these blocks
and pressing "Help" will take you to a list of IDs.
52.             Additionally, hovering over any block will show a tool tip.
53.         </p>
54.     </div>
55.     <div style="margin-left: 50%; height: 205px; border: 1px solid black; ">
56.         
57.     </div>
58. </div>
59. <h3>Editing recipes</h3>
60. <div style="width: 100%; ">
61.     <div style="width: 50%; height: 195.6px; float: left; background-color: lightgray;
border: 1px solid black; ">
62.         <p>
63.             Pressing the "Edit Recipe" button will open the recipe editor.
64.             Here, items can be added to the recipe, and the in game crafting station needed
can be specified.
65.         </p>
66.     </div>
67.     <div style="margin-left: 50%; height: 195.6px; border: 1px solid black; ">
68.         
69.     </div>
70. </div>
71. </body>
72. </html>
73.

```



## Editing mod page – EditModHelp.html



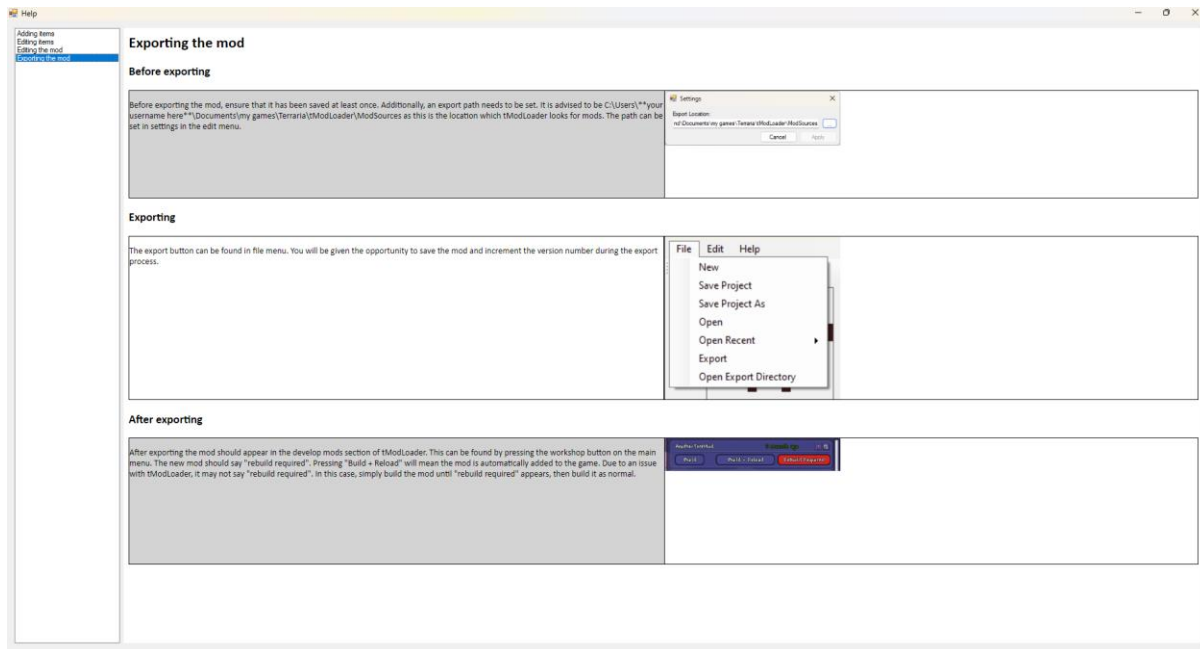
```
1. <html>
2. <head>
3.   <style>
4.     body {
5.       font-family: Calibri
6.     }
7.
8.     p {
9.       font-size: 14px
10.    }
11.  </style>
12. </head>
13. <body>
14.   <h2>Editing the mod</h2>
15.   <div style="width: 100%;">
16.     <div style="width: 50%; height: 135.8px; float: left; background-color: lightgray;
border: 1px solid black;">
17.       <p>
18.         The "Project Details" button, found in the edit menu, will allow you to edit the
details of the mod.
19.       </p>
20.     </div>
21.     <div style="margin-left: 50%; height: 135.8px; border: 1px solid black;">
22.       
23.     </div>
24.   </div>
25.   <div style="width: 100%;">
26.     <div style="width: 50%; height: 147.5px; float: left; border: 1px solid black;">
27.       <p>
28.         Here, you can edit the details of the mod. The icon can be left blank, and is
automatically resized correctly.
29.         The version number can be manually changed here, although it will also be
automatically incremented when exporting.
30.       </p>
31.     </div>
32.     <div style="margin-left: 50%; height: 147.5px; border: 1px solid black;">
33.       
```

```

34.         </div>
35.     </div>
36. </body>
37. </html>
38.

```

## Exporting mod page – ExportModHelp.html



```

1. <html>
2. <head>
3.     <style>
4.         body {
5.             font-family: Calibri
6.         }
7.     }
8.     p {
9.         font-size: 14px
10.    }
11. </style>
12. </head>
13. <body>
14.     <h2>Exporting the mod</h2>
15.     <h3>Before exporting</h3>
16.     <div style="width: 100%;">
17.         <div style="width: 50%; height: 170px; float: left; background-color: lightgray; border:
18.         1px solid black;">
19.             <p>
20.                 Before exporting the mod, ensure that it has been saved at least once.
21.                 Additionally, an export path needs to be set. It is advised to be
22.                 C:\Users\**your username here*\Documents\my games\Terraria\ModLoader\ModSources
23.                 as this is the location which tModLoader looks for mods. The path can be set in
24.                 settings in the edit menu.
25.             </p>
26.         </div>
27.         <div style="margin-left: 50%; height: 170px; border: 1px solid black;">
28.             
29.         </div>
30.     </div>
31. </body>
32. </html>

```

```

27.     </div>
28.     <h3>Exporting</h3>
29.     <div style="width: 100%;">
30.         <div style="width: 50%; height: 258.9px; float: left; border: 1px solid black;">
31.             <p>
32.                 The export button can be found in file menu. You will be given the opportunity
33.                 to save the mod and increment the version number during the export process.
34.             </p>
35.         </div>
36.         <div style="margin-left: 50%; height: 258.9px; border: 1px solid black;">
37.             
38.         </div>
39.     <h3>After exporting</h3>
40.     <div style="width: 100%;">
41.         <div style="width: 50%; height: 200px; float: left; background-color: lightgray; border:
42.         1px solid black;">
43.             <p>
44.                 After exporting the mod should appear in the develop mods section of tModLoader.
45.                 This can be found by pressing the workshop button on the main menu.
46.                 The new mod should say "rebuild required". Pressing "Build + Reload" will mean
47.                 the mod is automatically added to the game.
48.                 Due to an issue with tModLoader, it may not say "rebuild required". In this
49.                 case, simply build the mod until "rebuild required" appears, then build it as normal.
50.             </p>
51.         </div>
52.         <div style="margin-left: 50%; height: 200px; border: 1px solid black;">
53.             
54.         </div>
55.     </div>
56. </body>
57. </html>

```

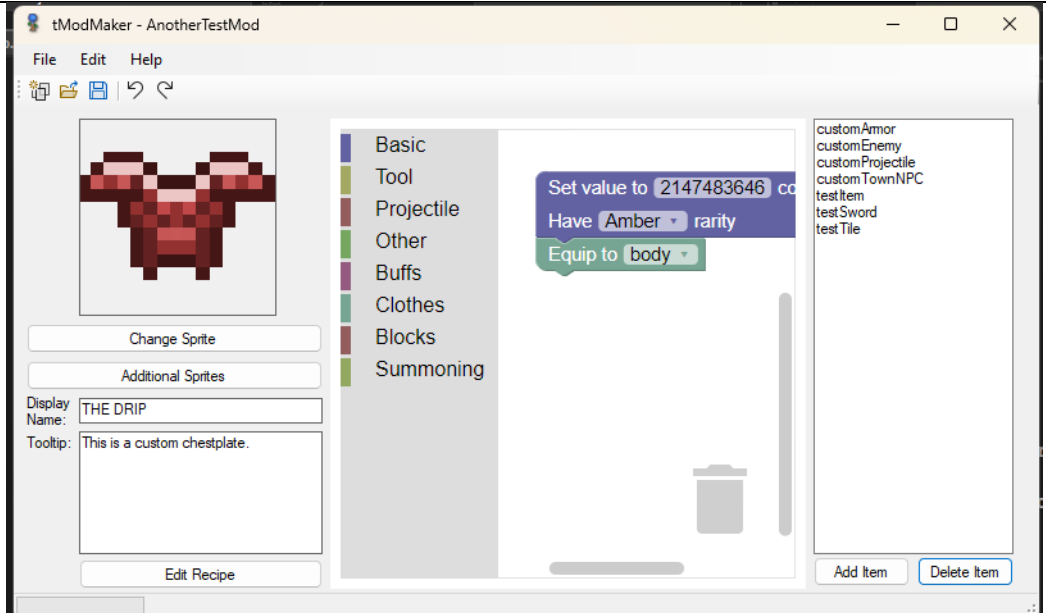
## Testing

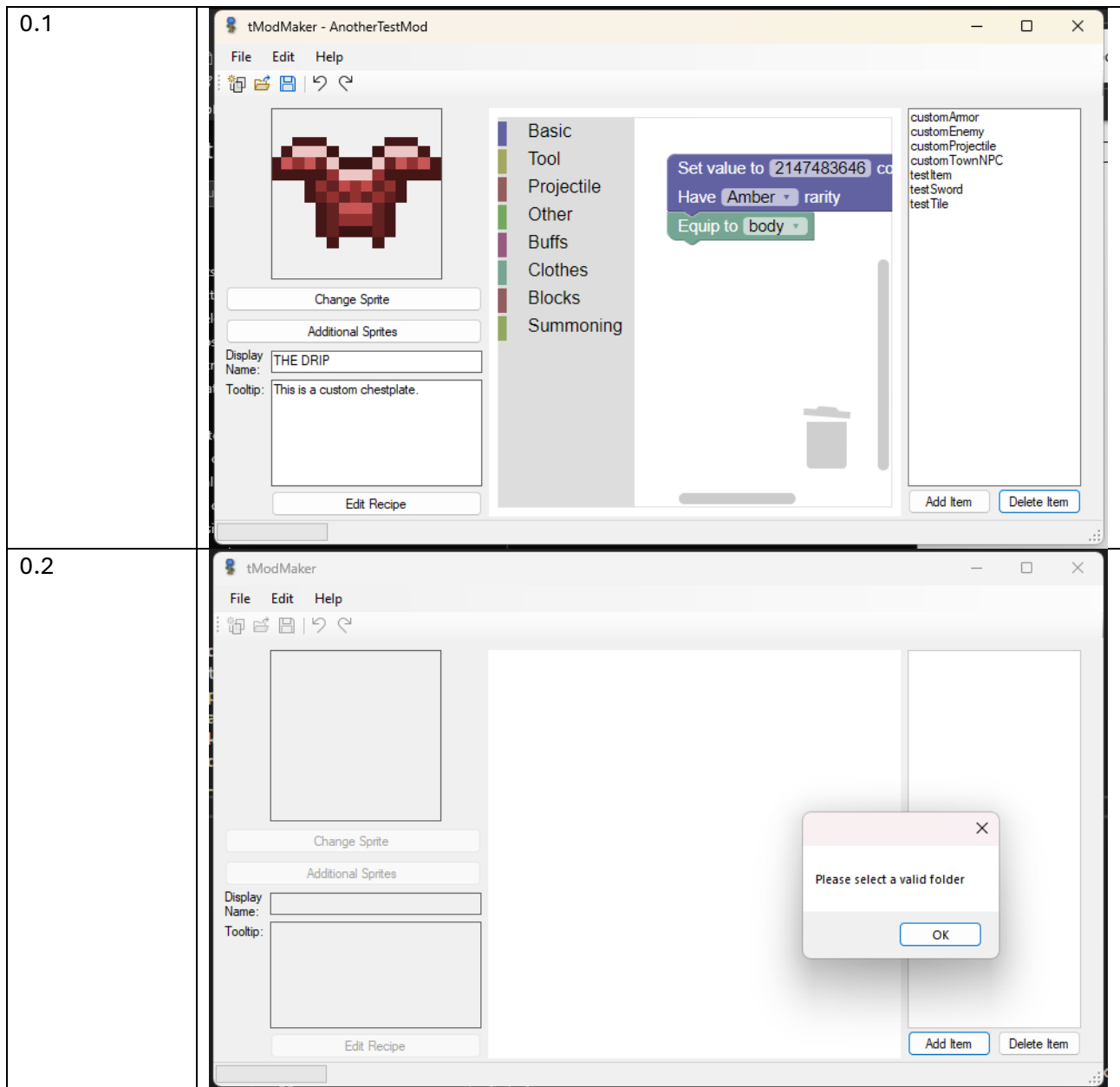
### 0 Saving, loading and exporting

Test Number	Test Description	Expected Outcome	Actual Outcome	Test Passed?	Comments
0.0	Attempting to open a valid project via the open button.	A folder selection dialogue is opened. When the mod is selected, the items list is populated by the mod's items.	As expected	Y	It would be nice if the first item in the mod was loaded into the editor.

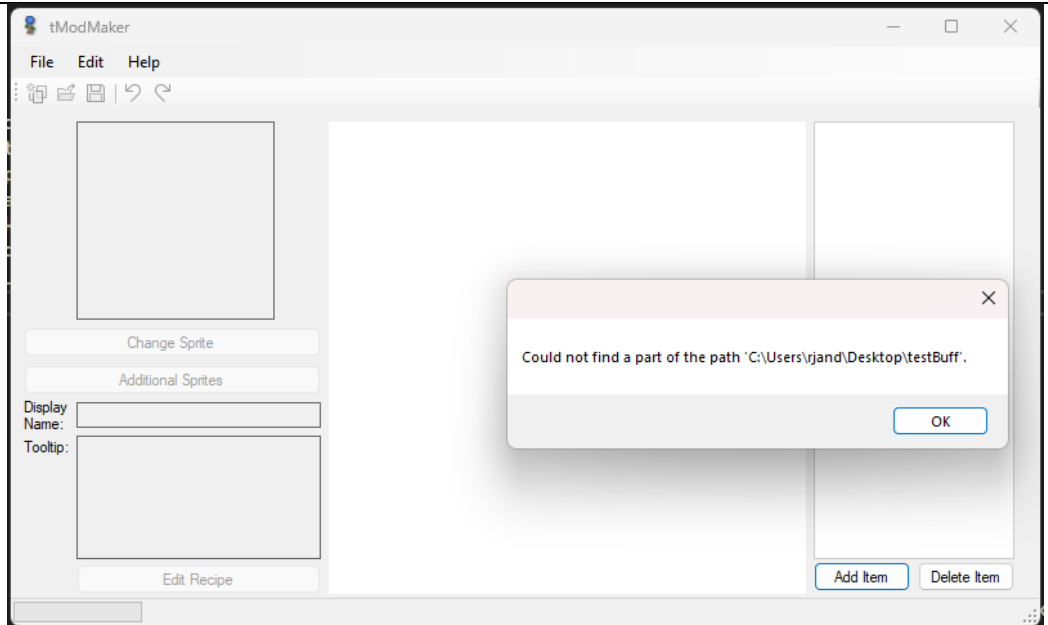
0.1	Attempting to open a valid project via the “recent projects” menu.	The mod is loaded as before.	As expected	Y	
0.2	Attempting to open an invalid project.	The user is informed the project is not valid.	As expected	Y	
0.3	Attempting to open a project at a location that does not exist.	The user is informed the location does not exist.	As expected	Y	
0.4	Saving a project that has been saved before.	Save progress should be indicated by the progress bar. During saving, all controls are locked.	As expected	Y	
0.5a	Saving a project that has not been saved before.	The user should be prompted to enter a name and path before saving takes place as normal.	As expected	N	There are a number of minor problems. Firstly, the name of the form is not updated to reflect the name of the project. Secondly, and more importantly, a project with no items can be saved, but is not considered valid when the user attempts to load it.
0.5b	-	-	As expected	Y	The previous issues are resolved.
0.6	Exporting a valid project.	The files should be written to the target directory, and a pop-up should appear say	As expected	Y	

		"Export Complete".			
0.7a	Exporting an invalid project.	The export should be called, and the user informed of which items are invalid.	The correct message is given, but the files at the export location are deleted.	N	
0.7b	-	-	As expected.	Y	The issue is now resolved.
0.8	Pressing "Save Project As".	User is prompted to enter new name and path.	As expected.	Y	

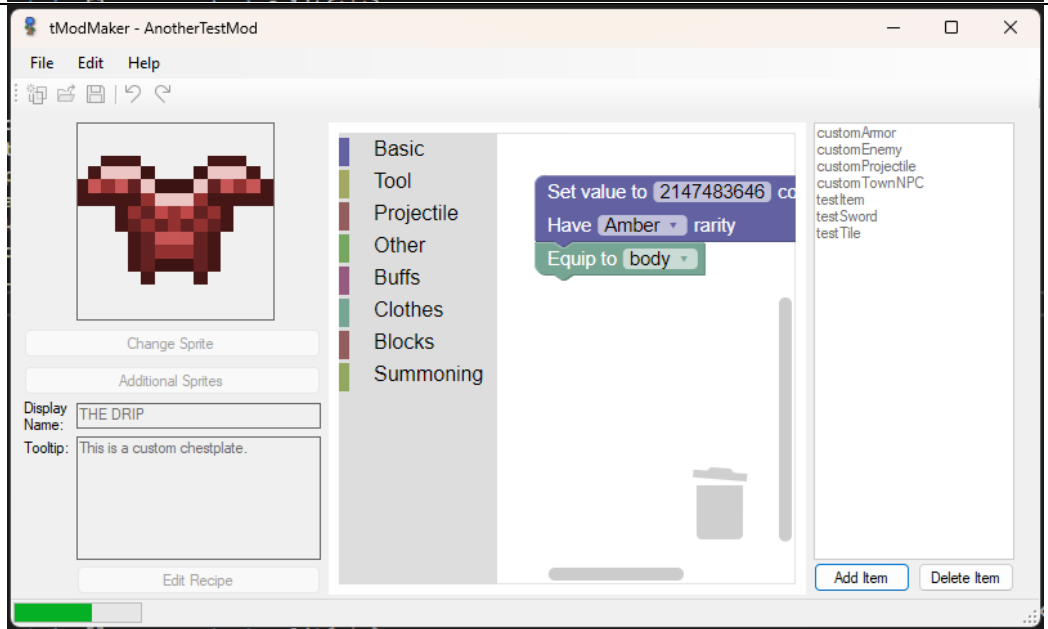
Test Number	Proof of test
0.0	



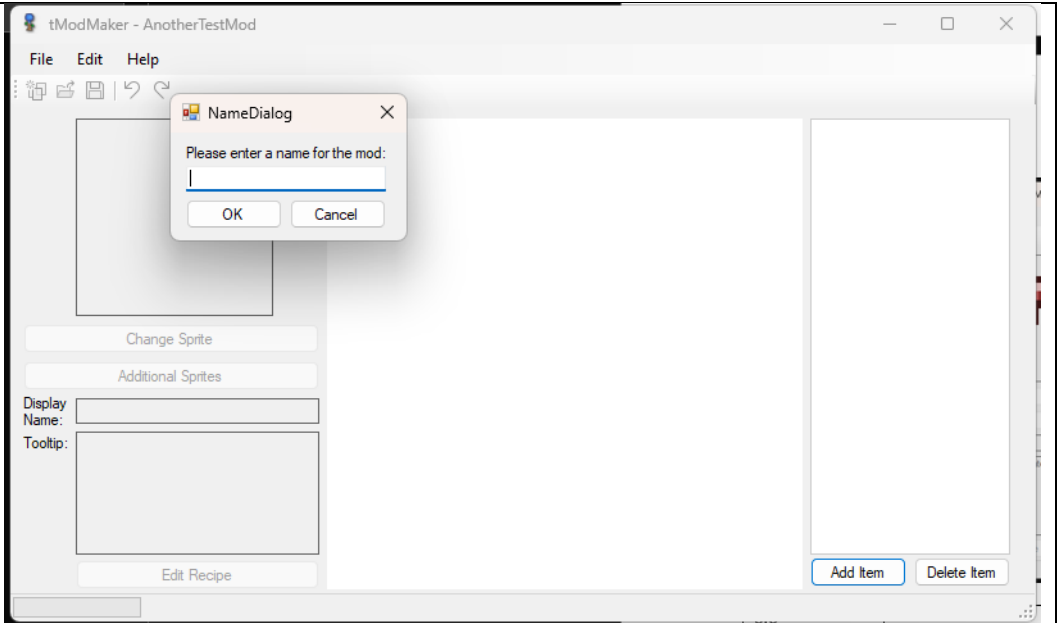
0.3



0.4

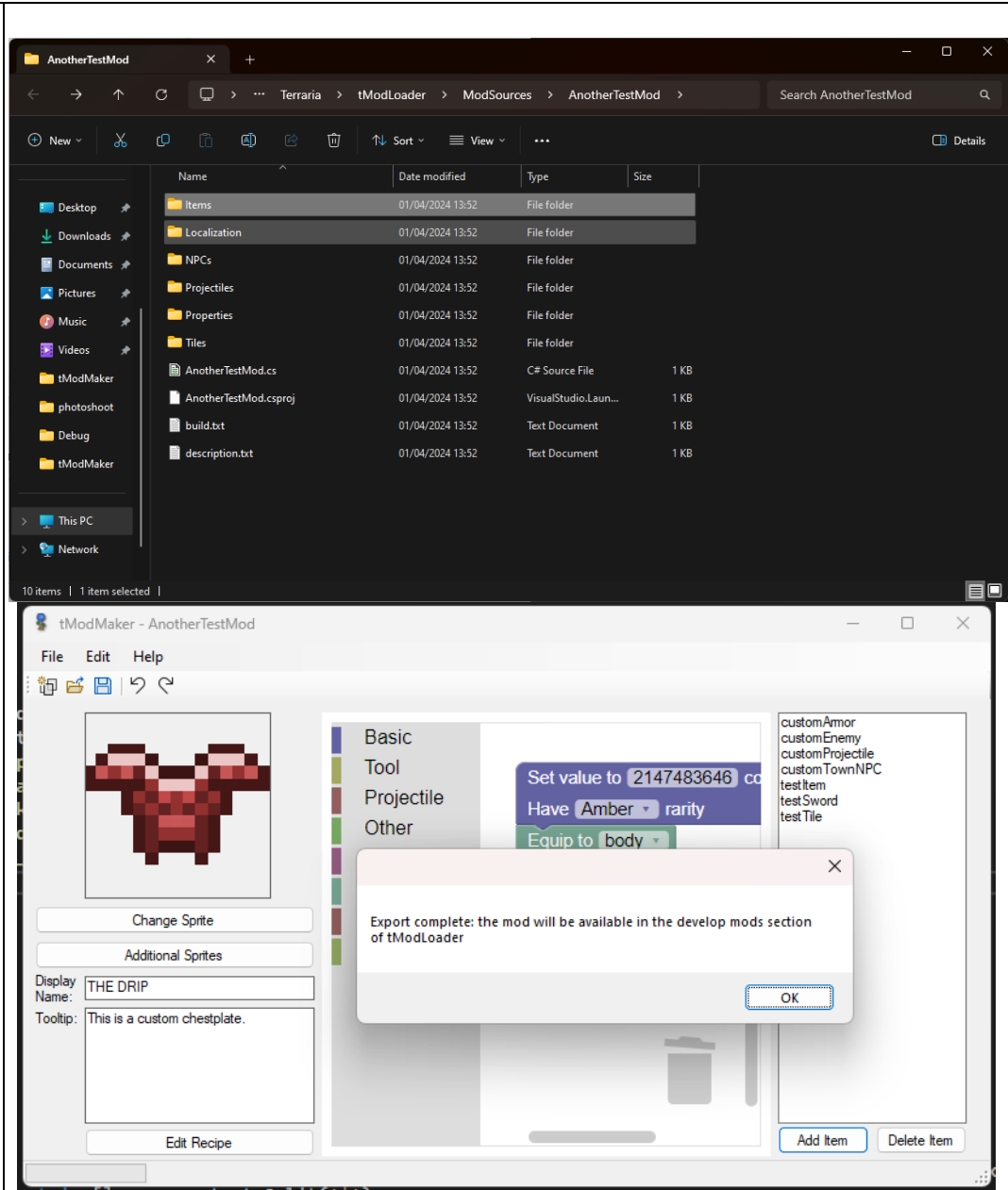


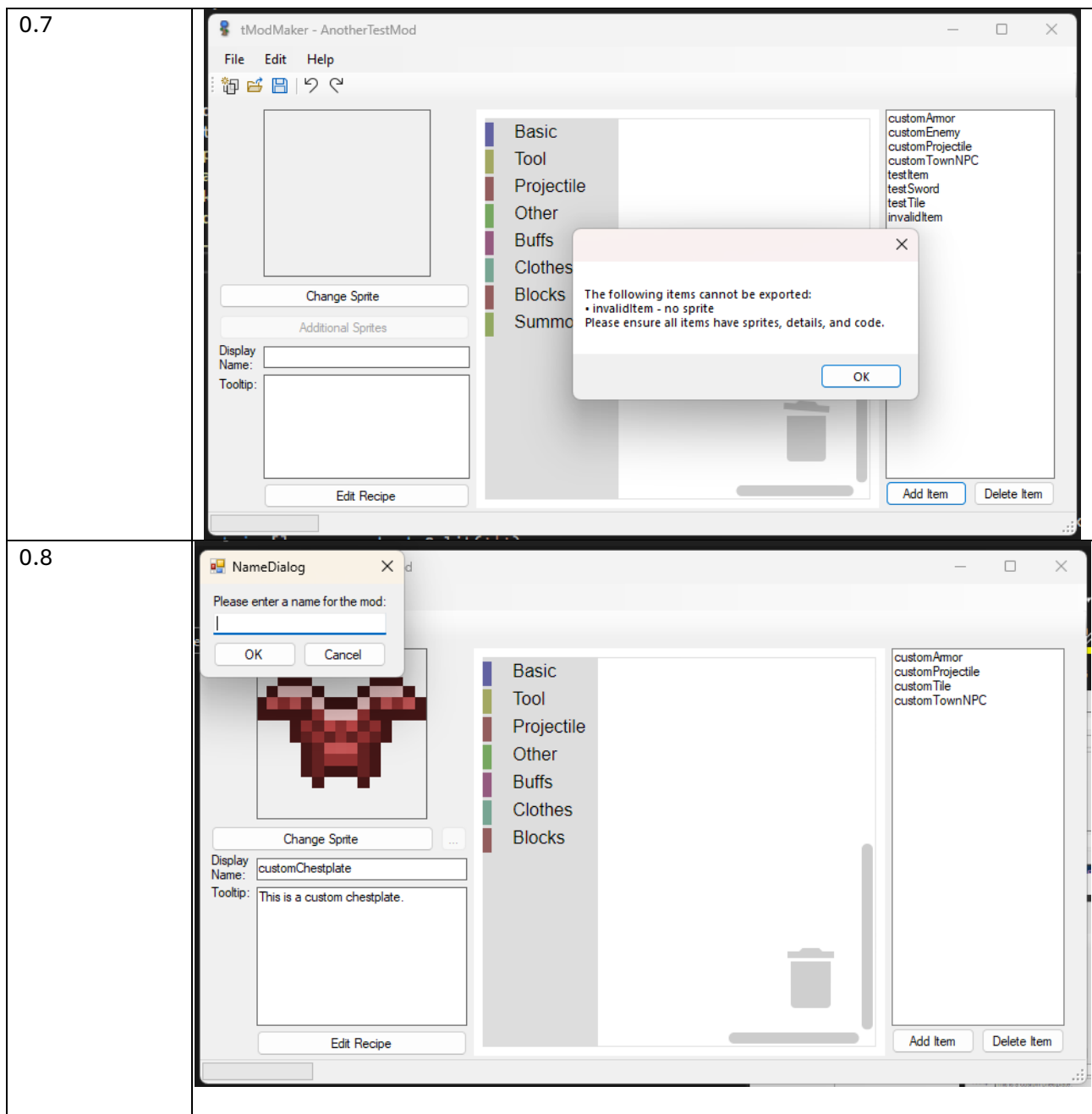
0.5





0.6

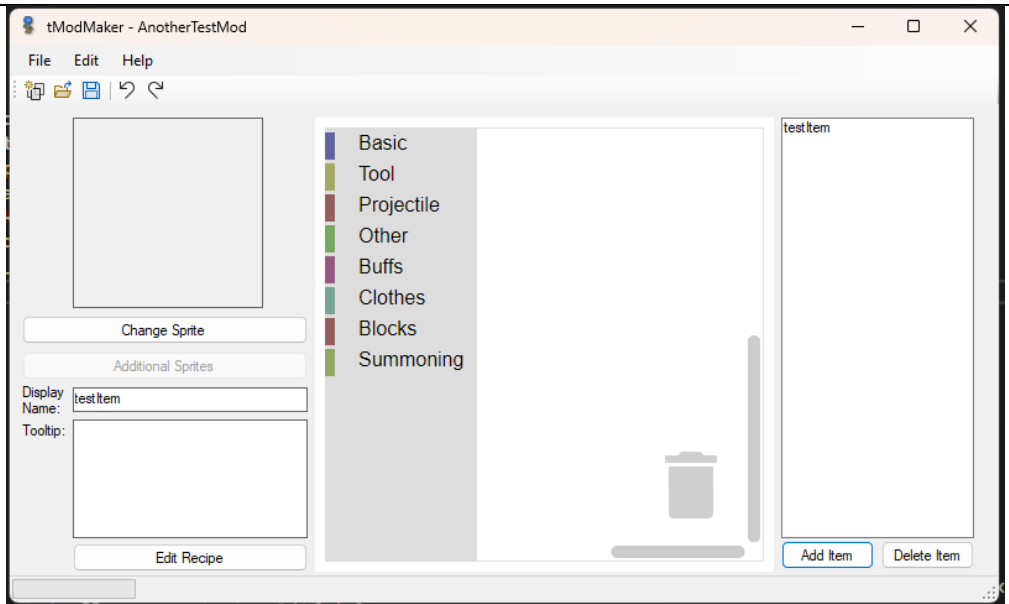




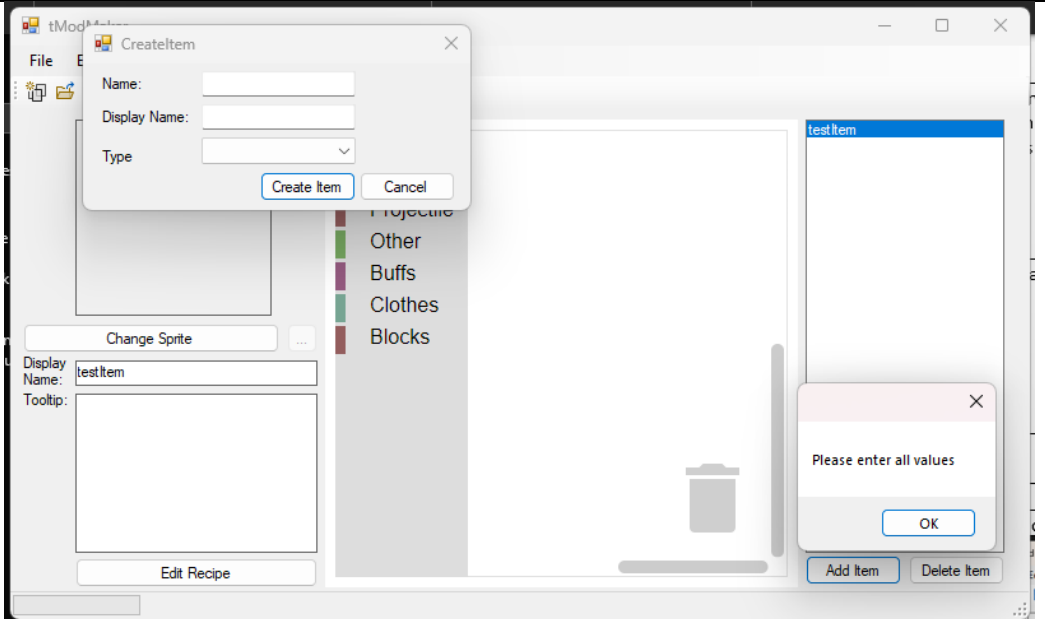
## 1 Adding and deleting items

Test Number	Test Description	Expected Outcome	Actual Outcome	Test Passed?	Comments
1.1a	Adding a new item with all values	The new item appears in the list.	As expected	Y	It would be nice if the new item were loaded into the editor.

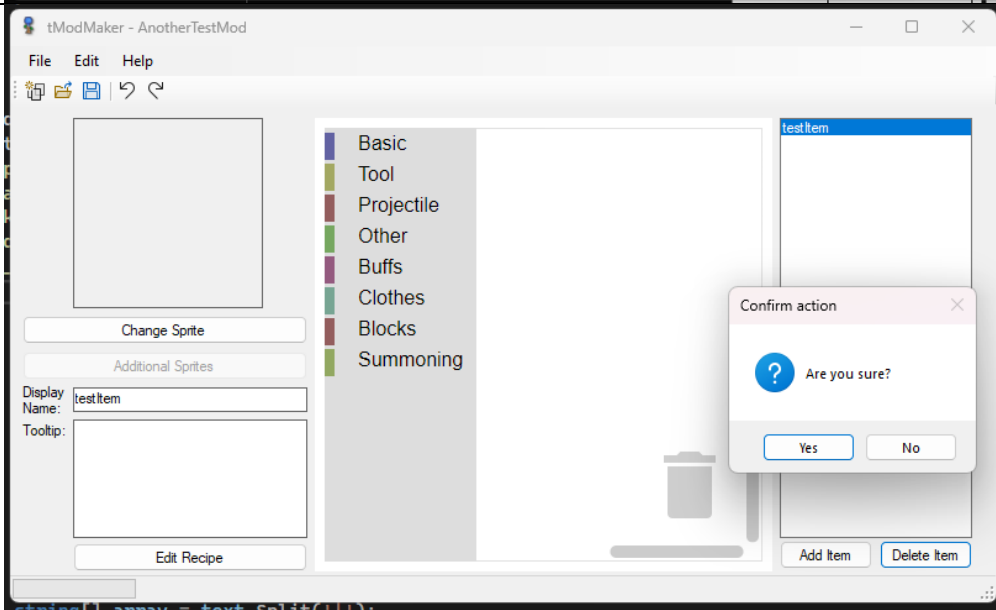
	entered correctly.				
1.1b	-	-	As expected	Y	
1.2	Adding an item with no values entered	The user should be prompted to enter the remaining values.			
1.3a	Deleting an item.	The item is removed from the list, after the user is asked to confirm.	Errors under some circumstances.	N	The program will crash if the mod has not been saved before. It tries to delete a path that does not exist.
1.3b	-	-	As expected	Y	The errors are resolved.

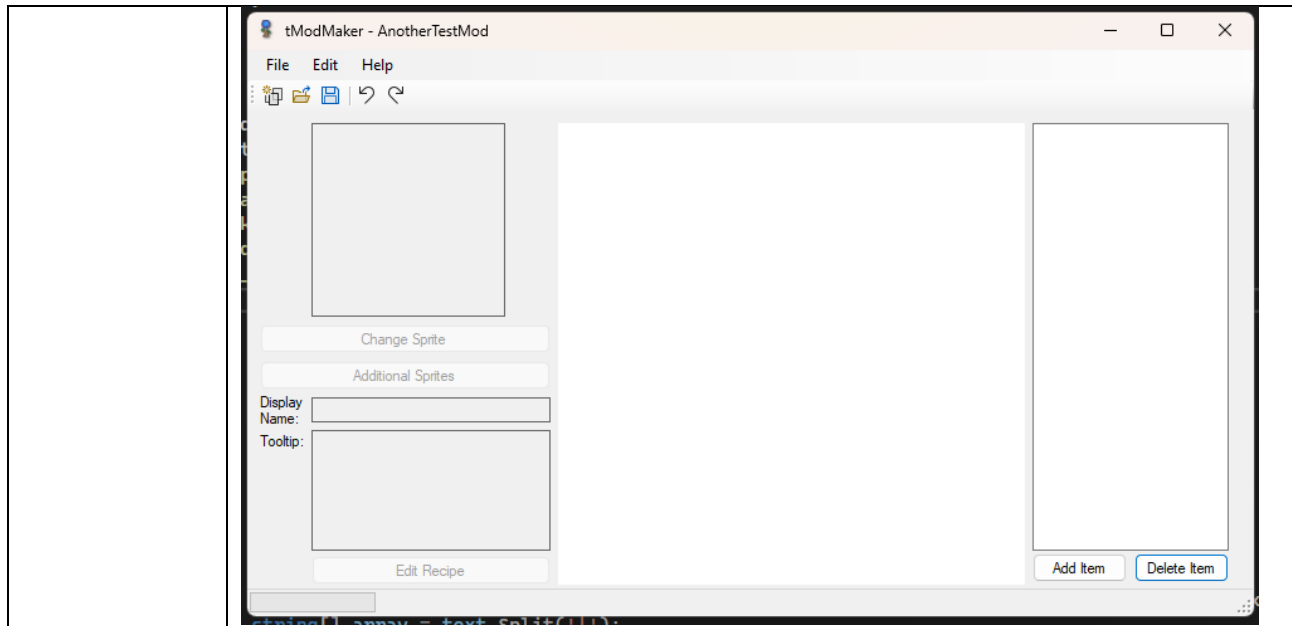
Test Number	Proof of test
1.1b	

1.2



1.3b

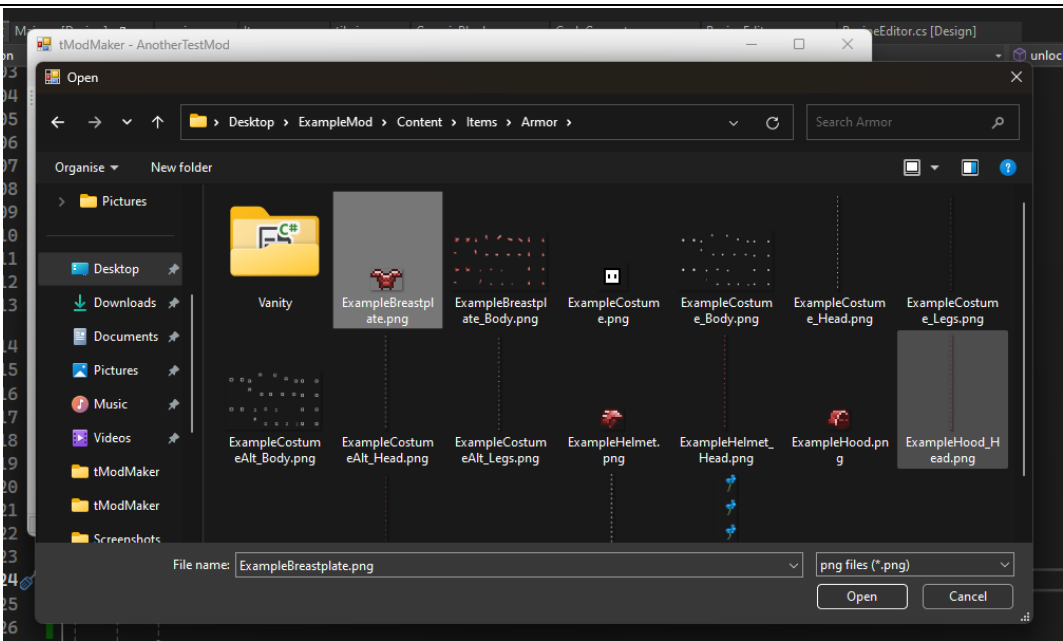


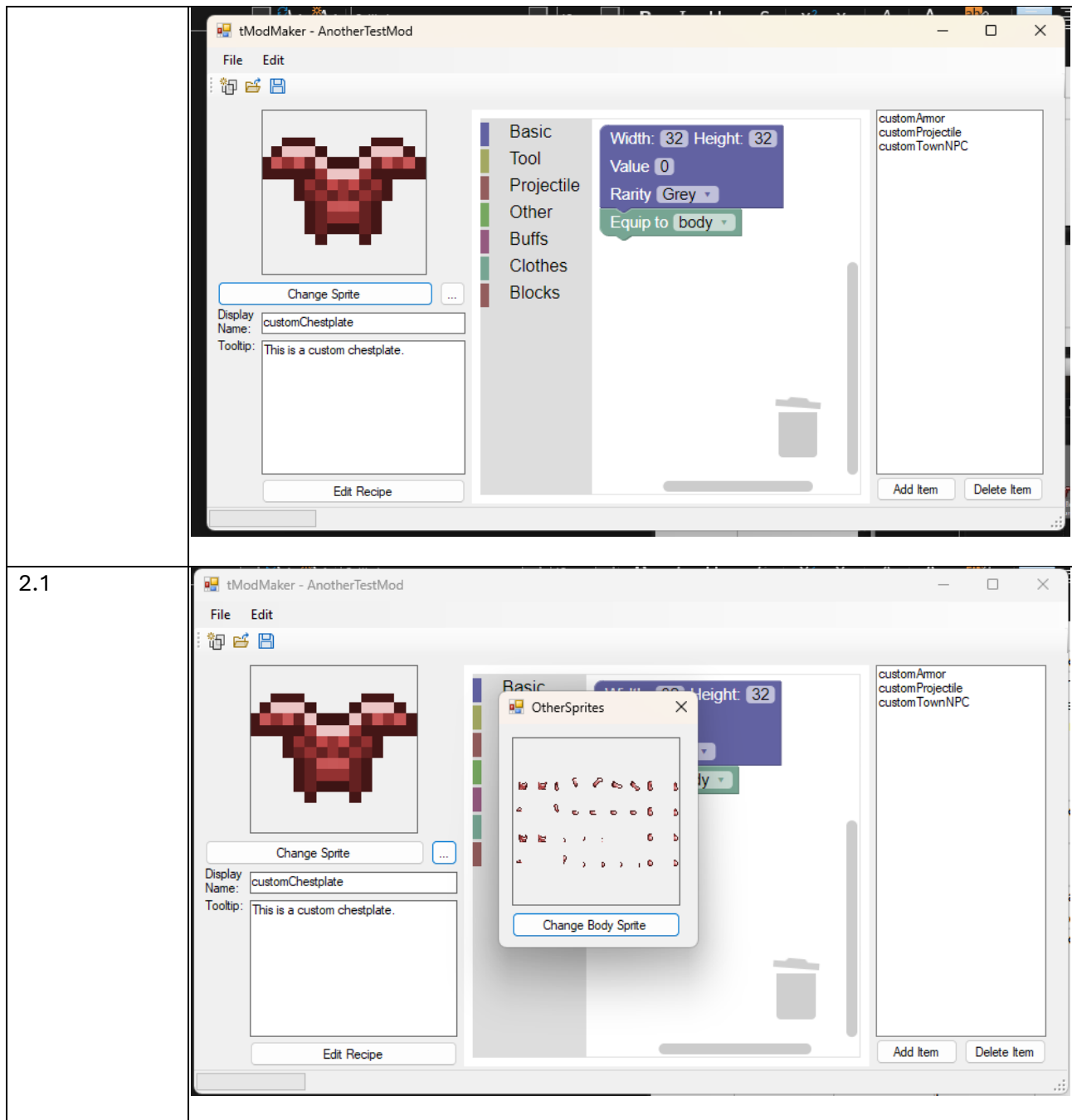


## 2 Editing items

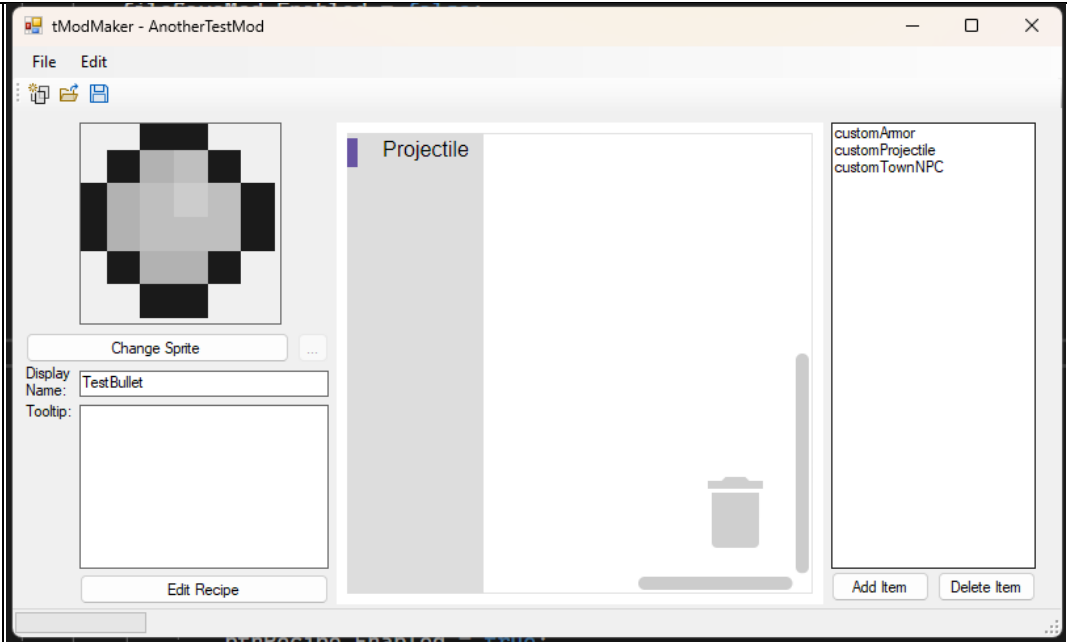
Test Number	Test Description	Expected Outcome	Actual Outcome	Test Passed?	Comments
2.0	Changing the sprite of an item.	A dialogue to select an .png file should open. The new sprite should appear in the picture box.	As expected	Y	
2.1	Changing the additional sprite of an item that requires a second sprite.	The user can see the second sprite and has the option to change it.	As expected	Y	
2.2	Changing the additional sprite of an item that does not require a	The button should be disabled.	As expected	Y	

	second sprite.				
2.3	Editing the recipe of an item that can have a recipe.	The recipe editing menu is opened.	As expected	Y	
2.4	Editing the recipe of an item that cannot have a recipe.	The button is disabled.	As expected	Y	

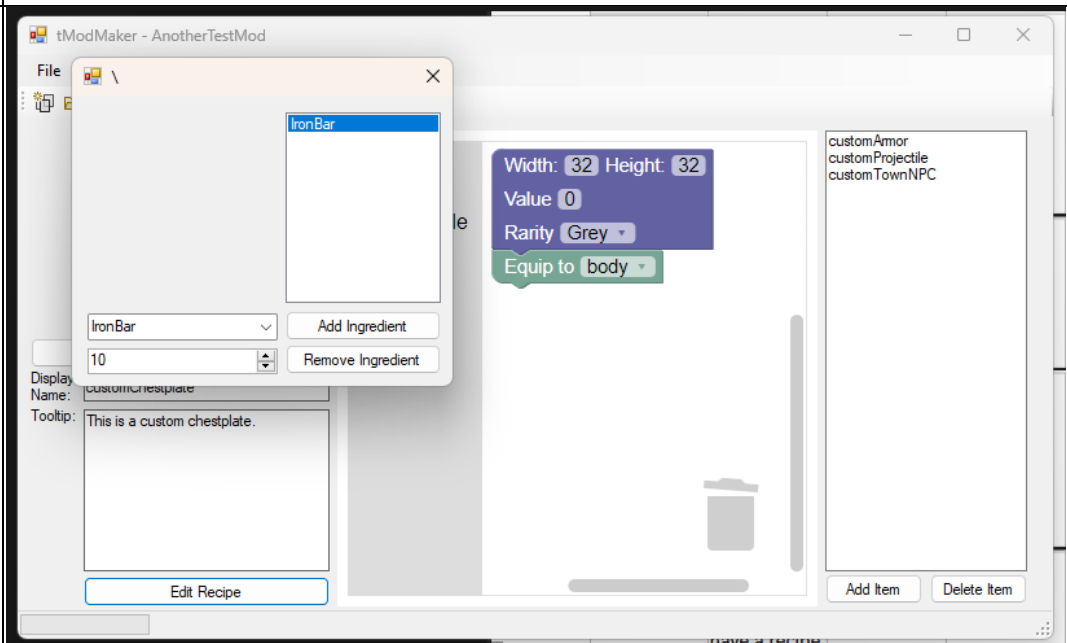
Test Number	Proof of test
2.0	 <p>The screenshot shows a Windows File Explorer window titled 'tModMaker - AnotherTestMod'. The address bar indicates the path: Desktop &gt; ExampleMod &gt; Content &gt; Items &gt; Armor. The left sidebar shows the 'Desktop' location selected. The main pane displays a grid of files and folders. The files include 'Vanity', 'ExampleBreastplate.png', 'ExampleBreastplate_Body.png', 'ExampleCostume.png', 'ExampleCostume_Body.png', 'ExampleCostume_Head.png', 'ExampleCostume_Legs.png', 'ExampleCostumeAlt_Body.png', 'ExampleCostumeAlt_Head.png', 'ExampleCostumeAlt_Legs.png', 'ExampleHelmet.png', 'ExampleHelmet_Head.png', 'ExampleHood.png', and 'ExampleHood_Head.png'. The 'File name' field at the bottom shows 'ExampleBreastplate.png' and the file type is set to 'png files (*.png)'. The 'Open' button is visible.</p>



2.2

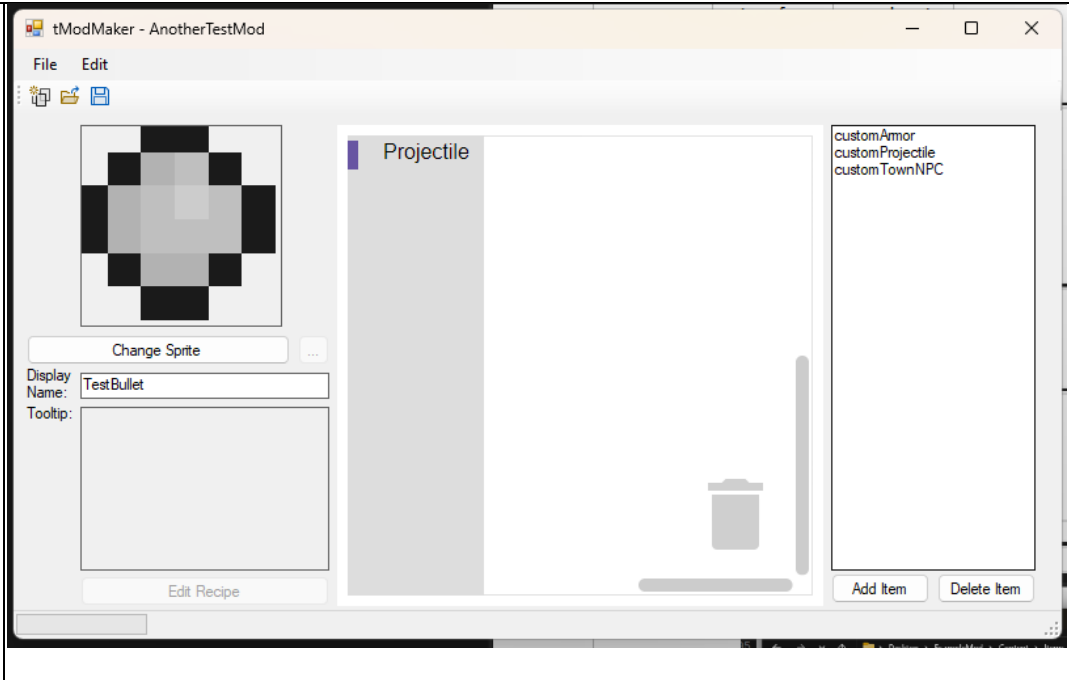


2.3





2.4

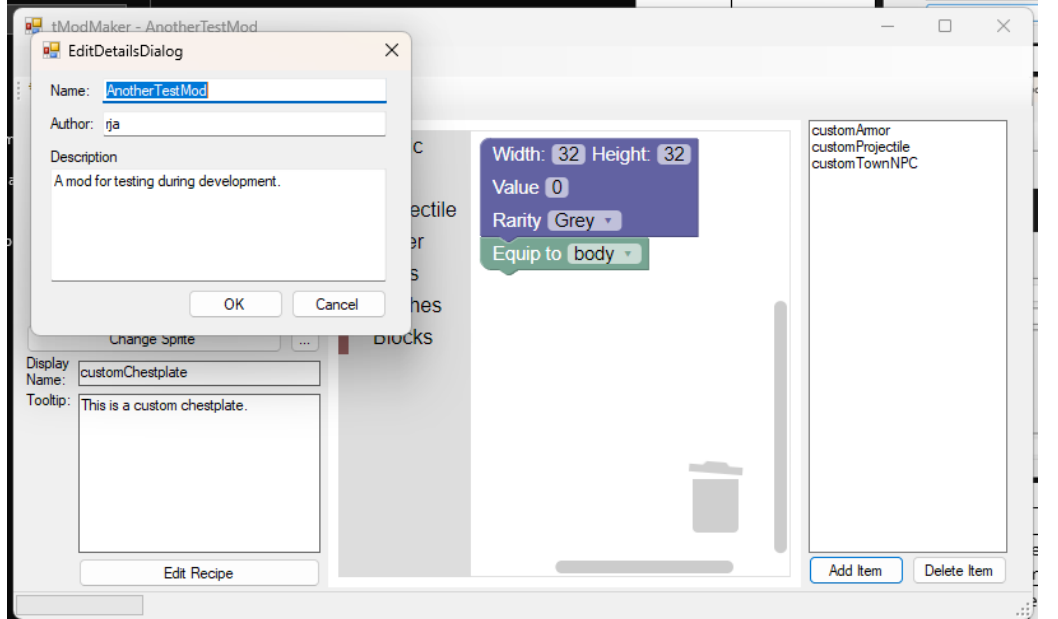


### 3 Editing the project

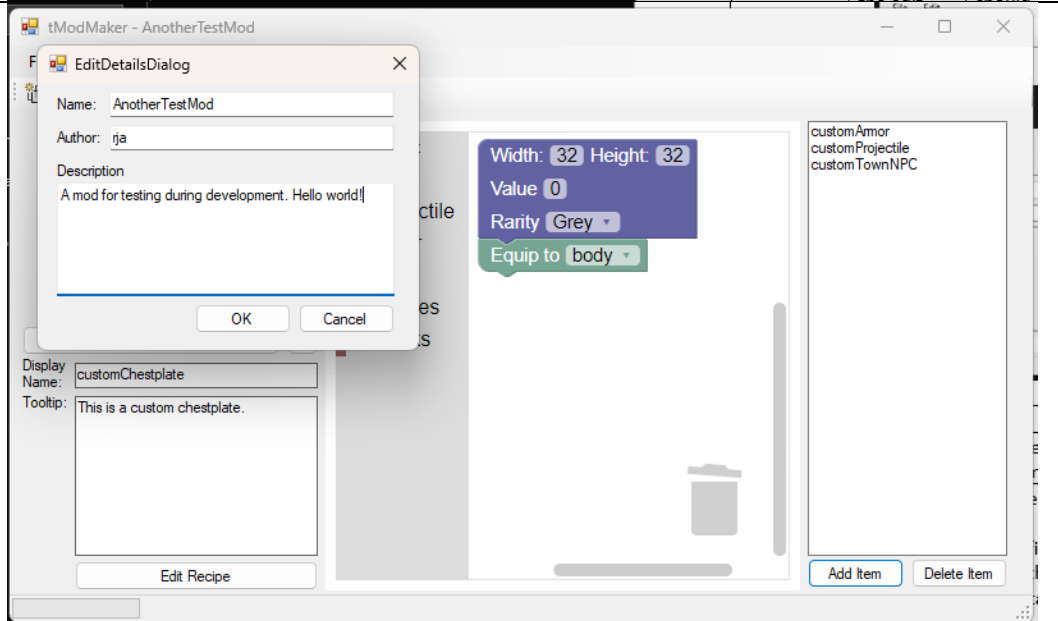
Test Number	Test Description	Expected Outcome	Actual Outcome	Test Passed?	Comments
3.0	Opening the edit project details menu.	The menu should open, filled out with the details of the project.	As expected	Y	
3.1	Editing the details of the project.	The details will be saved on closing.	As expected	Y	

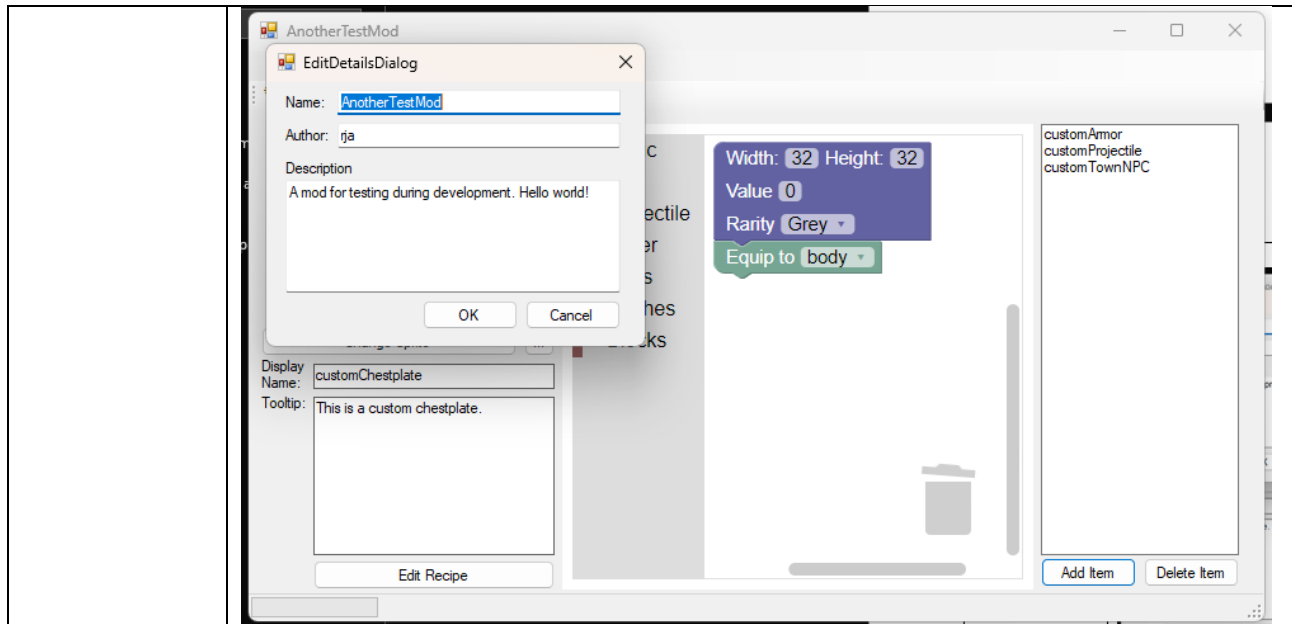
Test Number	Proof of test
-------------	---------------

3.0



3.1

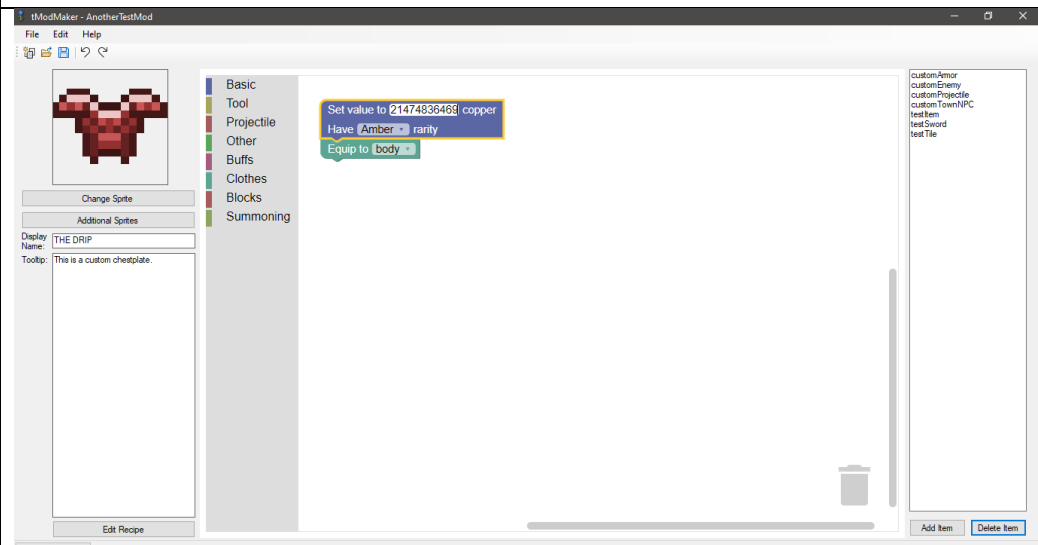


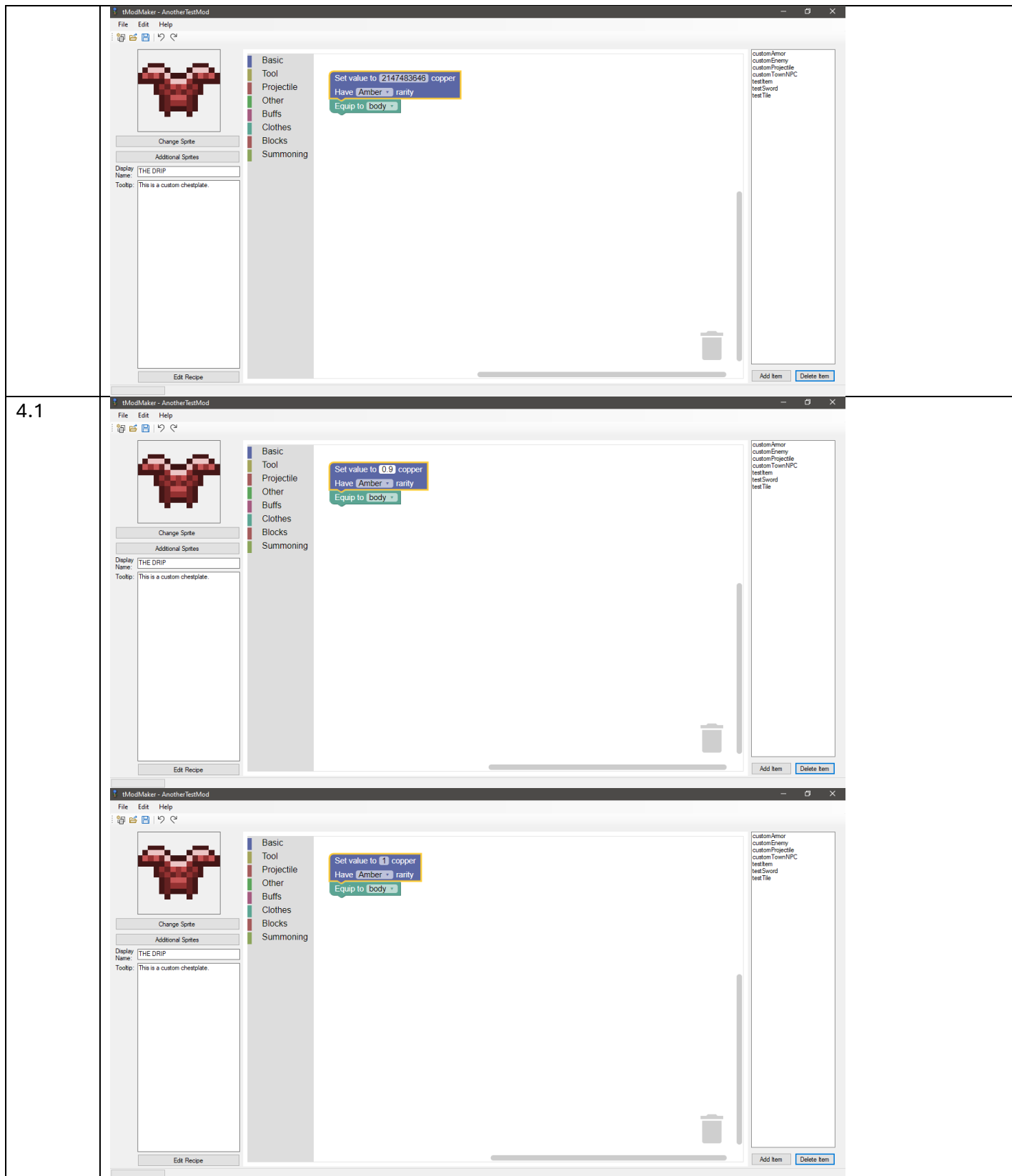


## 4 Validation in Blockly

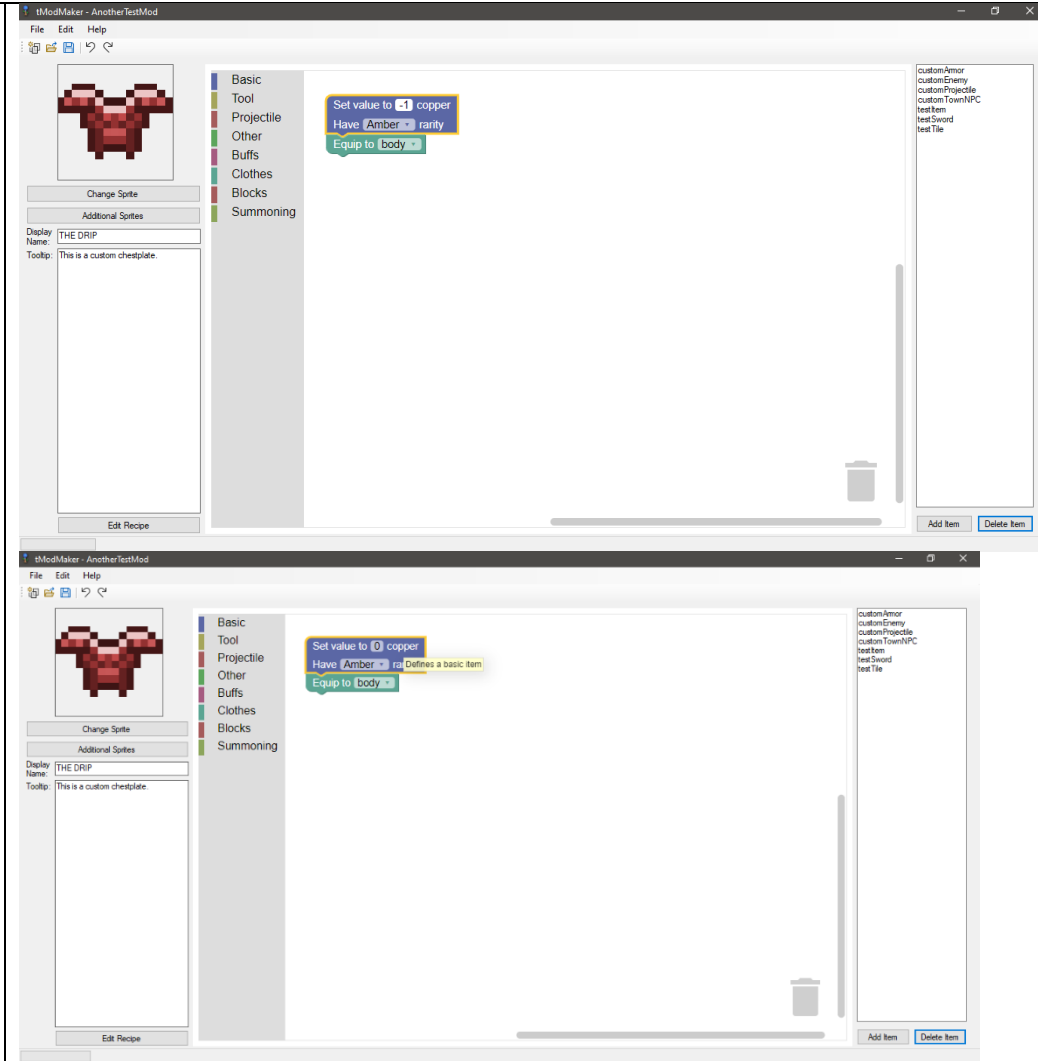
Test Number	Test Description	Expected Outcome	Actual Outcome	Test Passed?	Comments
4.0	Entering a value larger than the signed 32-bit integer limiting into an integer field.	The value will be set to 2147483646 instead, one less than the integer limit.	As expected	Y	
4.1	Entering a non-integer value into an integer field.	The value is set to the nearest whole number	As expected	Y	
4.2	Entering a negative number into a field for a positive integer.	The value is set to 0.	As expected	Y	
4.3	Entering a single quotation mark into a text field.	The program continues to operate as normal.	The code becomes unreadable by Blockly.	N	The code sees the quote as the beginning or end of a string.

4.4	Entering an open curly bracket into a text field.	The program continues to operate as normal.	As expected	Y	
4.5	Entering a closed curly bracket into a text field.	The program continues to operate as normal.	The program crashes.	N	The program attempts to handle invalid Json and crashes.
4.6	Entering a random in range value into an integer field	The program continues to operate as normal.	As expected	Y	

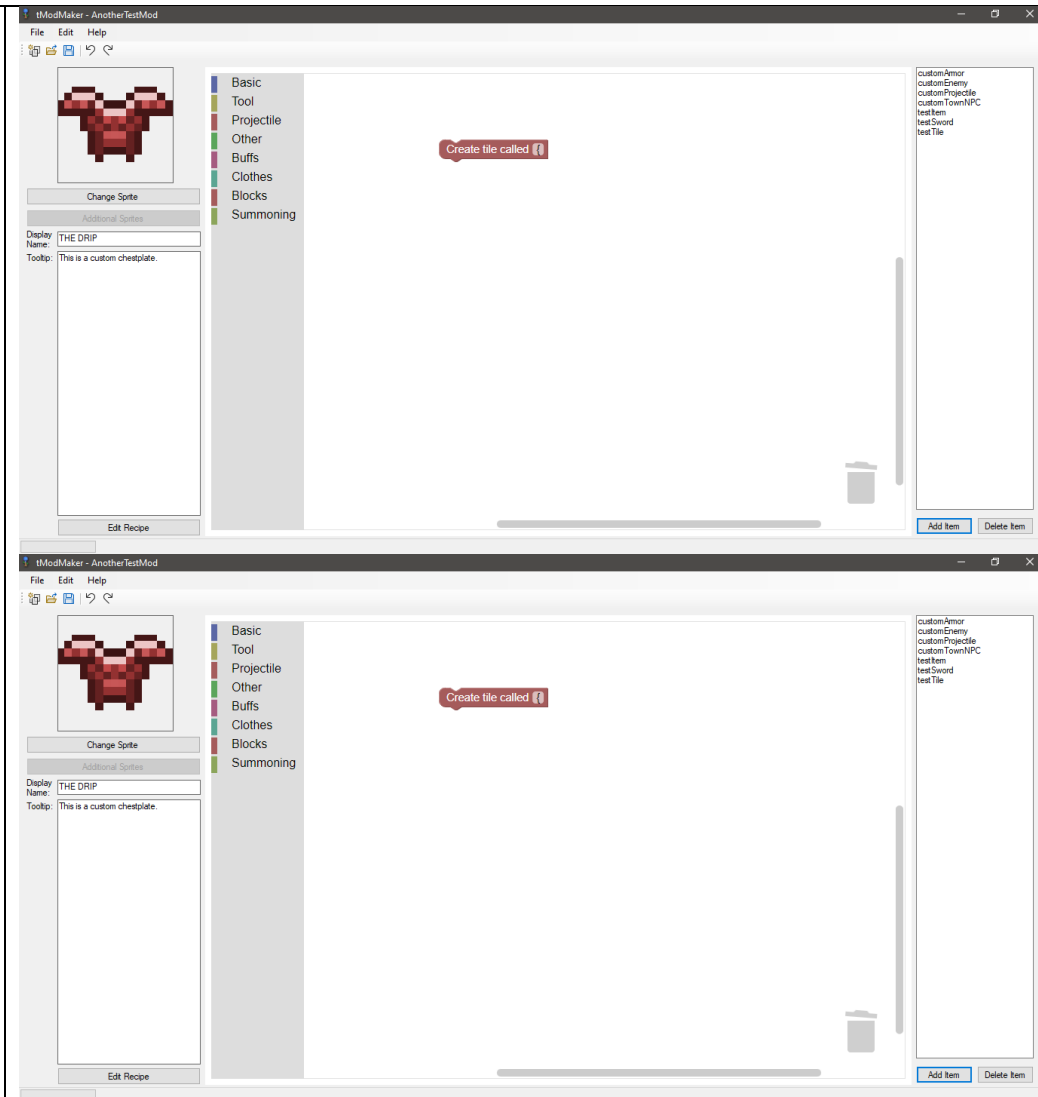
Test Number	Proof of test
4.0	 <p>The screenshot shows the tModMaker - AnotherTestMod interface. On the left, there is a list of categories: Basic, Tool, Projectile, Other, Buffs, Clothes, Blocks, and Summoning. The 'Basic' category is selected. In the center, there is a recipe editor for a chestplate. The recipe includes a copper ore, an amber rarity, and a body slot. The display name is 'THE DRIP' and the tooltip is 'This is a custom chestplate.' On the right, there is a list of items: customArmor, customEnemy, customProjectile, customTownNPC, testItem, testSword, and testTile. The 'Add Item' button is highlighted.</p>



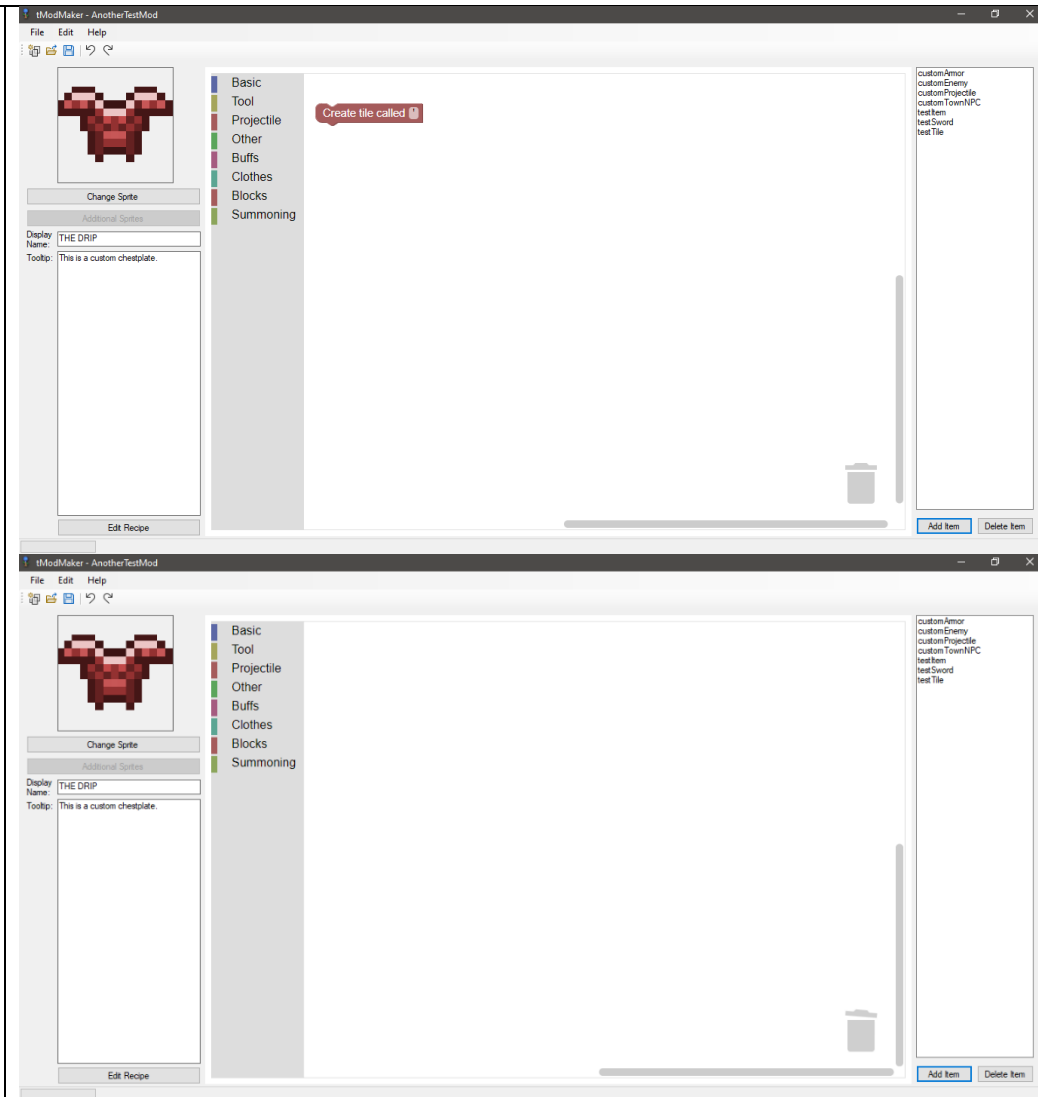
4.2



4.3

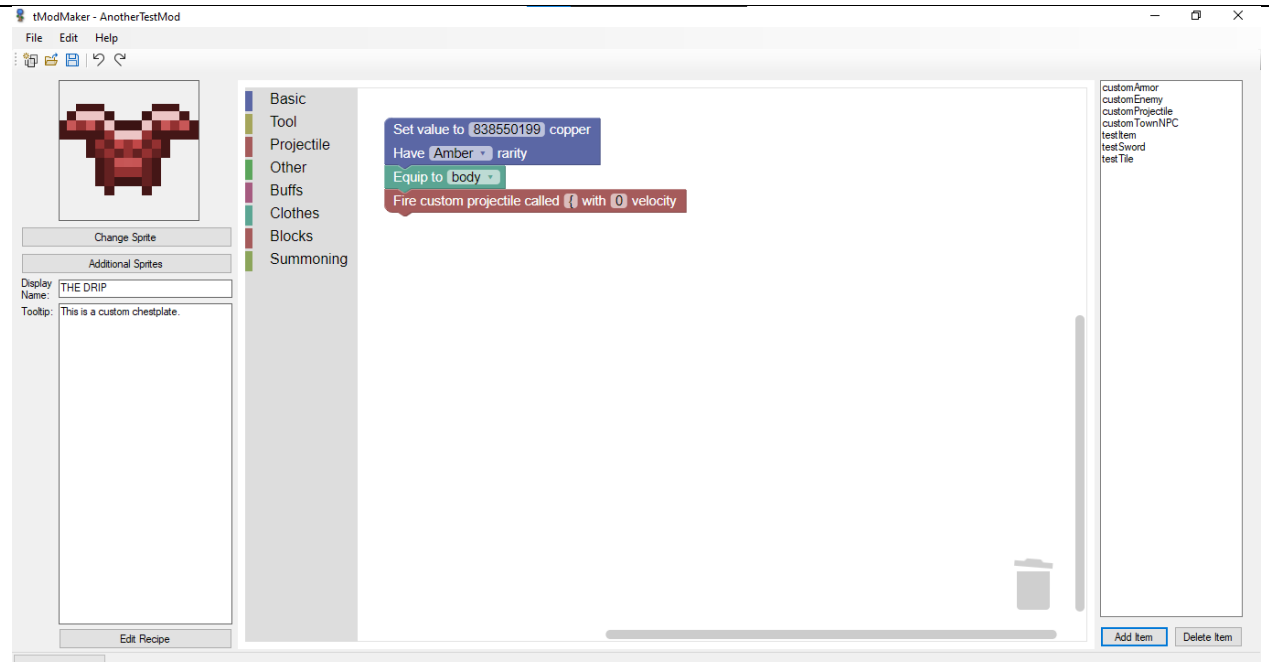


4.3

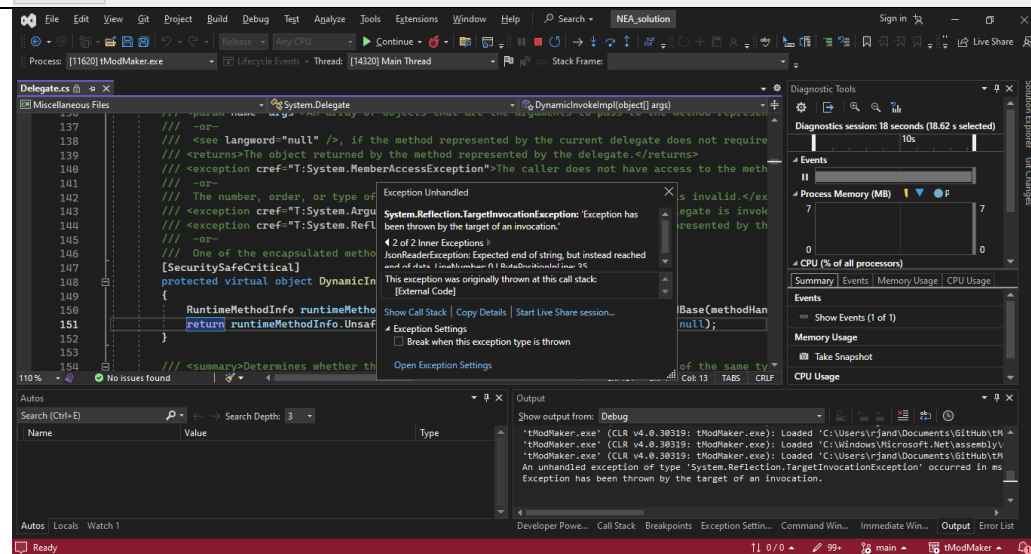




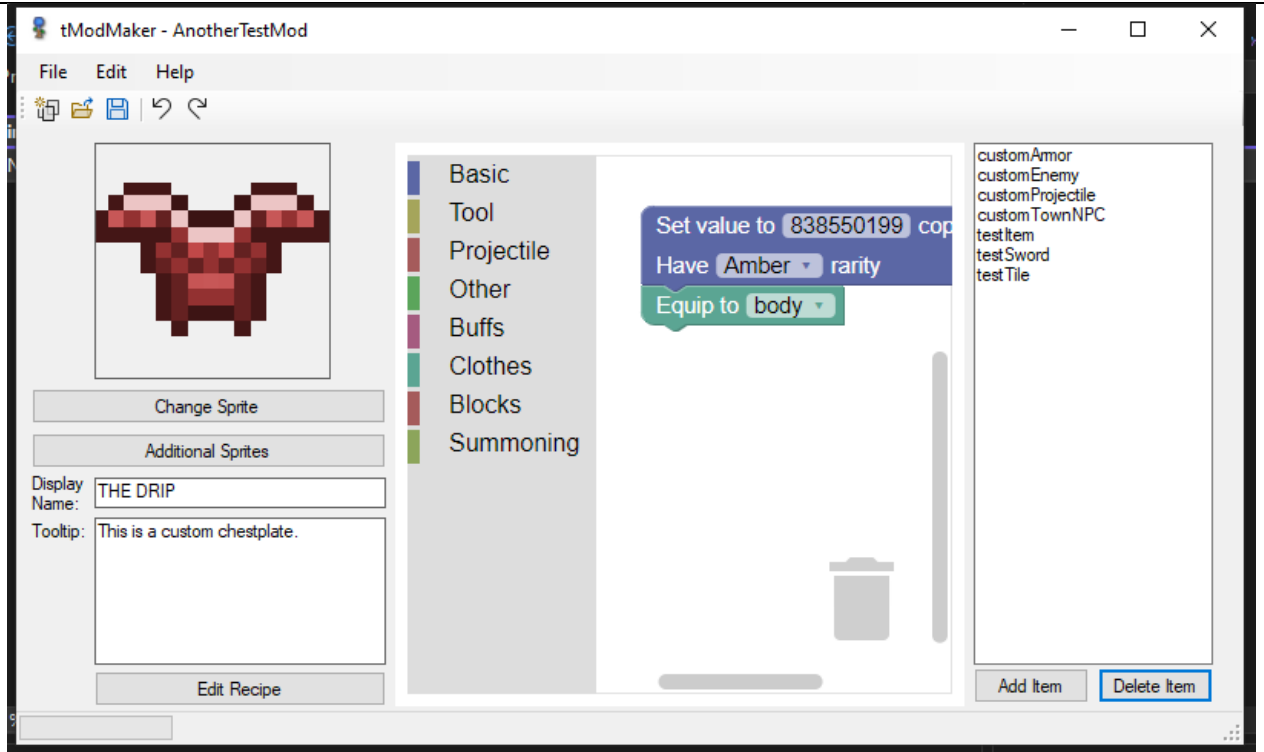
4.4



4.5



4.6



## Blockly input issues

An issue that became evident while using the program was that certain characters cause severe issues with Blockly.

These appear to be characters that have syntactical meaning in Json. For example, single or double quotation marks as well as backslashes render the code unreadable by Blockly. The issue with quotation marks specifically is particularly frustrating, as it is very plausible that the user would try to enter them. A more severe but less prevalent issue exists when entering closing curly bracket. This causes the program to crash entirely, as invalid Json is created. In order for the user to salvage their work, they would have to manually edit the code for their items, something I would not encourage.

However, as I do not have full control over Blockly, this is not an issue I can easily address. The obvious thing to do in theory is to add a backslash to create an escape character. However, by the time the C# component has access to the Json, it is already invalid. It is for this reason that I have been unable to correct this issue.

## Evaluation

### Changes in user interface design

A part of my finished project that changed drastically from the original design was my user interface. I believe that when designing the project, I did not appreciate the conventions of user

interface design and took an approach involving far too many forms. Additionally, many of these forms contained far too much information, which meant that more specialised forms were needed simply to fit what was displayed. For example, the user does not need to be constantly reminded of the type of item they are working with, given they created and named the item.

Furthermore, I overlooked many features that typically exist in software. For example, I did not consider tool and menu strips to contain basic functions like saving and undoing. It was only as I worked on the project that I realised it was genuinely hard to use and began to make changes to the interface accordingly. Some concepts from the original UI remain, but it is near unrecognisable.

## Feedback

In order to evaluate my project, I collected both written and verbal feedback. The written feedback came in the form of a survey, which both evaluated to what extent I had met certain objectives and allowed users to highlight any bugs or issues with the program.

## Survey Results

My first set of questions were regarding the user interface. Overall feedback was positive, with the average score out of ten for ease of use being a seven. The feedback suggests that the biggest issue with the UI was that some buttons did not immediately stand out. Additionally, some blocks were not clear as to what they did, though I subsequently addressed this issue. One user suggested that it was not clear what a sensible value for some fields was, specifically knockback, which should be a very small value, and confused some users.

Modding experience amongst respondents was split 50-50. However, this did not have any real impact on the experience they had using the tool. I believe this is down to the level of abstraction in the program. Interestingly, the people with no experience modding expected more from the program. One person wanted events to be triggered when an NPC died, which I went on to try to add, but had issues with implementing the relatively advanced code as blocks. Another user wanted to be able to modify existing game items, but this was not within the scope of the project, as overriding existing functionality is far more complicated than generating entirely new items.

One user had trouble with the export process. They did not know the process for building a mod in tModLoader and expected it to work without building. While this is not directly an issue with my program, I implemented a help page which explains how to use tModLoader, as well as my program, to the user.

## Verbal Feedback

As my project neared completion, I allowed a small number of people to use it, in order to ensure the quality of the user experience. Two of these people were A-Level classmates, which the third was not a computer science student, though had some programming experience. Working with the computer science students gave me an idea of the best-case scenario: people who understand fully what they are doing and are able to give feedback based on a strong understanding of the project. It was for this reason that they were able to pinpoint certain bugs, as well as giving suggestions on how the user interface could be overhauled to substantially improve the user experience. Working with the third person gave me an idea about how an “average” user would

interact with the program. Things were not as immediately clear for them as they were for the more experienced programmers. I found working with them very valuable for designing blocks, as it allowed me to see where the functionality of a block needed clarification for an unexperienced user. I suspect that given my familiarity with tModLoader, some block designs were obvious only to those who already understood the modding process, and so had to be reworded.

Overall, the verbal feedback was hugely important in the progression of my project. It gave me new perspectives on my project and forced me to redesign some aspects based on how users would actually interact with it.

## Objectives

In order to evaluate the success of my project, I will hold it against my original objectives. These should give a clear frame of reference for the extent to which my project was successful.

### Core Objectives

- 1. The program should allow an inexperienced user to create a mod without using outside documentation.**

I was mostly successful in completing this objective. Through the use of abstraction of much of the mod creating process, as well as descriptive tooltips, using the tool should require no outside documentation under most circumstances. That said, a handful of blocks include a help button that opens the Terraria wiki. Both of these blocks require the name of the tile or item to be entered as text, as Blockly's dropdown menus are not searchable. I made the decision to do this as I think it would be a better user experience. So, I think the one place where I failed to meet this objective is justifiable.

- 2. Under most circumstances, it should be faster to use the program to create a mod than programming in C#.**

I worked with a classmate to test the efficiency of creating a mod with my program. He had no prior experience modding Terraria but is an experienced programmer. He successfully created a mod using my program in under 5 minutes. To write the same mod by hand took him over half an hour, despite his usage of online documentation and GitHub Copilot. Additionally, a notable proportion of the time was spent troubleshooting the file structure. I think it is fair to conclude that my program significantly streamlines the process, especially for inexperienced modders.

- 3. It should be impossible to make a non-functional mod using the tool. A player could make a mod that works badly, but the game should be able to compile and run it.**

I can foresee one circumstance under which a mod could be exported but not work. This is if the user enters a name of an item that does not exist into either of the blocks previously mentioned with regard to objective one. However, despite the problems, I still believe this is the better approach, as it makes the software much more usable, despite the potential for a non-functional mod to be generated. The other situation where a non-functional mod could be generated is if not all details of a mod are filled in, but there is validation in place to stop this.

**4. The program must allow the user to create the following things in Terraria; items (including equipment and armor), projectiles, NPCs, and tiles.**

Unfortunately, I have fallen short of this objective, specifically with regards to NPCs. The main issue my research during analysis for the project revealed was the lack of documentation. It is for this reason that I have failed to implement some parts of NPCs. NPCs cannot have custom AIs, but can make use of existing ones, so they are more than usable as most NPCs use one of a few preset AIs. While I did fail to meet this objective in its entirety, I do not feel that the experience is hindered too much, as it is a relatively minor omission.

**5. The program should hide much of the file management for the mod to make the process simpler for inexperienced users.**

I was totally successful in meeting this objective. The user should never have to interact with the file system, except when saving the file and selecting sprites. When exporting, the files are written to the correct directory in the correct file structure. So, the user has no need to manually manage the files.

**6. The program should produce fully readable C# code, with correct formatting and indentation, to make editing it directly as easy as possible.**

The code output is nicely formatted, making use of the correct levels of indentation. After discussing this with one user, I discovered that they had created a mod with my program, then gone on to further edit the code manually. They specifically mentioned that the formatting of the code had made this process a lot easier.

**7. The program should include an example mod, which highlights how each type of item can be used, and how the code is written.**

I ultimately decided against using an example mod, opting instead to offer support for the user in various other ways. I felt an example mod would not have a huge amount of value, as the blocks are already fairly self-explanatory. I implemented a help page following user feedback, which describes the progress of making and exporting a mod. Additionally, each block has a tooltip to further explain its functionality. I think, despite not implementing the example mod, I met this objective in spirit, and provided the necessary level of support to the user.

**8. The output of the program should be ready to compile and in the correct directory, so no further input is needed from the user.**

While I mostly met this objective, there are two existing issues that mean some further user input is required. Sometimes, tModLoader will not detect new code, so the mod must be built multiple times until the changes to the code are detected. The other issue happens infrequently, and I have not been able to locate the issue. tModLoader will error under unknown circumstances, citing an error that refers to a file not generated by the program. The mod can be exported again, fixing the issue, so I do not consider this to be too severe a problem.

**9. The blocks should be clear in what they do, by using syntax very close to English.**

It is clear to me from both written and verbal feedback that the blocks were clear in how they were meant to be used. No-one I received feedback from suggested that clarity was an issue, so I think I was successful in this objective.

## Extension Objectives

- 10. The tool should contain some form of walkthrough for new users. It should highlight the basic properties and how they apply to each type of object, as well as explain their in-game effect.**

While there is no specific walkthrough in the program, there are a number of resources to improve the ease of using it. Firstly, each block has a tooltip which describes its functionality. This covers the objective of explaining what values mean in-game. In addition, blocks that require an item name link to a list of taken from the game's wiki. There is also a general help tab that explains how to use the program. So, despite not precisely meeting this objective, I feel I have succeeded in sufficiently supporting the user.

- 11. The program should offer some interaction with other mods.**

Ultimately, this objective was not entirely realistic to integrate in a "nice" way. The code would have to refer another mod, and correctly generating the reference is what is challenging. It would most likely have ended up with a block that prompted the user to enter the name of the item and mod as text. I don't feel like this is a good solution, especially as the internal names of items for other mods are not easily available. So, I did not meet this objective for the sake of user experience.

## Improvements

In the previous sections, I have made it clear where I think my project falls short. Here, I will suggest a number of improvements that could be made, and the likelihood and challenges of them being implemented. I have also considered some features people felt were missing in the feedback.

- **Allowing the creation of custom AIs for NPCs and projectiles:** In theory, allowing the user to create a simple AI would not be difficult. However, as most modders make use of the numerous existing AI styles, and the source code for the larger mods with custom AIs tends not to be available, I could not find up-to-date information on how to create an AI. I have no doubt that with the correct information, implementing this feature would be fairly easy, at least for a simple AI. When it comes to more complex AIs, I fear that an environment where they could be created would end up being just as complex as writing the code by hand and require a similar level of programming knowledge. So, while this feature would be a nice technical achievement, I am not convinced that it would be of a huge amount of value in most circumstances.
- **Sprite editor:** In my analysis, I discussed a sprite editor, though it ultimately never made it into the design. I do not think that a basic sprite editor would have had much value, as tools like Photoshop provide much better facilities that I could. That said, there would be a significant benefit to an editor that ensures that the user creates the more complicated sprites in the correct layout. Sprites for equipped armor, for example, have a specific layout so the game can load the right part of the sprite at the right time. A sprite editor that

provided templates like this would be a very useful feature. I would most likely have the user draw each sprite separately, then have the program combine them into the right layout for the game. I feel like this would be a valuable, but very time consuming, addition to the program.

- **Overriding existing items:** While this was not originally something I planned for my project, having spoken to users, it is clearly something people want. While this could be done, the code translation would have been complicated, as it would have been overriding existing items as opposed to creating new ones, which creates another layer of complication. I considered integrating it, but the amount of effort required is, in my opinion, not worth it for a relatively niche use case. So, while possible, I do not consider it to be worthwhile, especially given the effort required.
- **Other content for tModLoader:** When designing my program, I focused only on the most important parts of the game. However, there are a number of types of item I did not include simply due to time constraints. That said, I believe it would be entirely plausible to add this functionality. It would be a case of adding several new item types and editors, which is easily done. The biggest challenges, as with all types of item, would be finding and understanding the documentation in order to create the code generator. Given more time, this would have been part of my program, but they were not part of my objectives, so I spent my time elsewhere.

## Closing comments

In conclusion, I consider this project a success. Having spoken to users and watched them interact with the program, I am very much contented with my outcome. I think I was successful in meeting the most important aspects of the objectives that I set myself, and the program serves a genuinely useful purpose. I am satisfied that the program provides an easy to use and validated environment for people to begin making mods for Terraria. The completion of this project has been very rewarding, but also a valuable learning experience. I am very happy with what I have achieved.