

How to create a Minecraft Mod

Lilian Gallon - lgallon@ucsc.edu - University of California, Santa Cruz

September 2019

Keywords: Minecraft, Game modding, Tutorial, Forge API

Summary

Game modding is a growing field of game developing which started from the initiative of players. When the game itself is not enough, you can improve it by adding your own functionalities. It is a way to learn coding by having fun. However, it can be difficult to get into game modding because you need to read a lot of documentation (if you find it) and know how the game is coded. This tutorial will guide you through your journey on how to make – your first? – Minecraft mod. You will learn each part of it, from setting up your Integrated Development Environment to adding your mod in your Minecraft game. We will also use the latest technologies available at this time. So the tutorial is taught using Minecraft Forge 1.14.4, GitHub Actions and IntelliJ IDEA 2019!

Intended audience

This tutorial is designed for people with a minimum of knowledge about Computer Science (Java and data structures). It is not needed to know Oriented Object Programming, but it will be easier to follow the tutorial by knowing this concept. Some parts of this tutorial won't be required to create a Minecraft mod, but it will teach you good programming practices. Continuous Integration and Unit testing are some of these non-required parts. It is not necessary to have a Minecraft.net account to program the mod, but it will be mandatory if you want to use it in your own world.

Style guide

We will use a modern style guide to redact this tutorial. The Google Developer Documentation Style Guide [1] was chosen as it is adapted to this kind of tutorial. It is really simple to understand compared to the other ones (Oxford, Chicago and APA for example). Moreover, the theme is clean and the tone has to be conversational, friendly and respectful without being overly colloquial or frivolous.

System requirements

Recommended means that this tutorial has been made using this technology. If a technology is not cited, it means that it is not working (ex: Forge 1.14.4 is incompatible with Java JDK >11) or not tested.

- **Operating System:**
 - Windows 10 (recommended)
 - Linux (part 2.1.1 differ a lot, slight changes for part 2.1.2 and 2.1.3)
 - MacOS (same as Linux)
- **Java JDK:**
 - JDK 1.8 (recommended)
 - JDK 9 (Not maintained anymore, slight changes for part 2.1.2)
 - JDK 10 (Not maintained anymore, slight changes for part 2.1.2)
- **Minecraft Forge:**
 - 1.14.4 (recommended)
 - 1.14.x (Unknown issues may occur)
 - 1.13.x (Slight changes for part 2.1.3, and 3.)
- **Integrated development environment:**
 - IntelliJ IDEA 2019.x (recommended)
 - IntelliJ IDEA 2018.x (Slight changes for part 2.1.3)
 - Eclipse Neon (parts 2.1.2 and 2.1.3 differ a lot)

Table of contents

Summary	1
Intended audience	1
Style guide	2
System requirements	2
Table of contents	3
1. Introduction	5
1.1 Introduction to Game Modding	5
1.2 Introduction to Minecraft Forge API	5
1.3 Introduction to Github	6
2. Setting up your development environment	8
2.1 Java JDK, IntelliJ IDEA and Forge	8
2.1.1 Java	8
2.1.2 IntelliJ IDEA	8
2.1.3 Forge with IntelliJ IDEA	8
2.1.4 How to change mod details	10
2.1.5 Hello World	11
2.1.6 Mod exportation	11
2.1.7 Sending Codebase to GitHub	12
2.2 Continuous Integration using GitHub Actions	12
3. Mod programming	13
3.1 How to create your first item	13
3.2 How to create a custom sword	14
3.3 How to create a custom recipe	16
3.4 How to change the world generation to include a custom ore	17
4. Installation	19
5. Conclusion	19
Resources	19

References	20
Appendices	20
Appendix 1: How to change JAVA_PATH to Java JDK 1.8	20
Appendix 2: Starter code	21
Appendix 3: ToolMaterialList enum	22

1. Introduction

1.1 Introduction to Game Modding

Prerequisites: none

Game modding is a growing field of game development. The term "mod" comes from "modification" or "modify". It is a domain where the participants use their skills to alter a video game. It can be adding new features, fixing issues, or even improving existing functionalities.

Modding became relatively popular with the game Doom (1993) [2]. The company Id Software separated the media files from the main program and made it accessible for users [3].

Before, it was a form of hacking, where the game was altered without the company's consent [4]. Then, over the next few years, game development companies realized that modding was beneficial for their games, and they started releasing tools to help modders. A tool can have different forms. Sometimes, it is an in-game editor that the player can use to create new quests, new missions or new environments. It can also be adapted for experimented developers as an API that they can use to alter the code base.

1.2 Introduction to Minecraft Forge API

Prerequisites: none

Minecraft Forge is an Open-Source modding API developed and maintained by the community [5]. Forge contains numerous hooks¹ into the Minecraft game engine, allowing us to create mods with a high level of compatibility [6]. The Minecraft Forge API was entirely re-written for the Minecraft 1.13 release [7]. It means that most of the tutorials are now outdated.

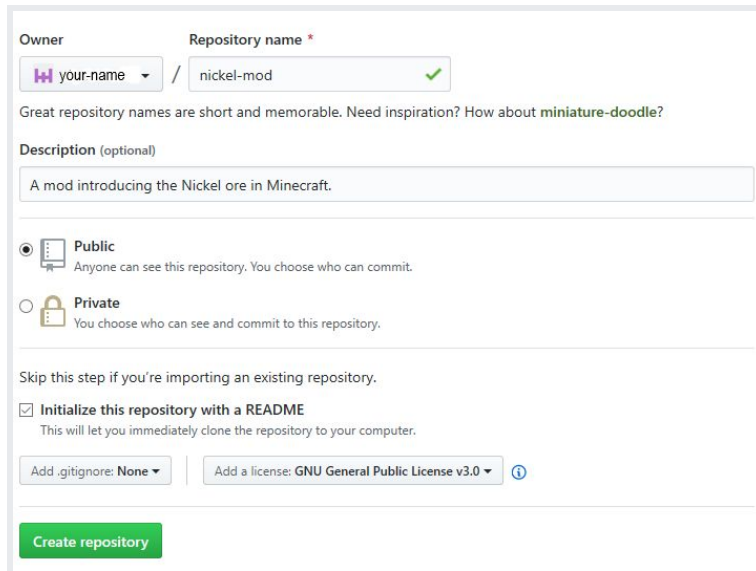
¹ A hook can be seen as a wire between Forge API and Minecraft where the data pass through

1.3 Introduction to Github

Prerequisites: none

Github is a company that provides hosting for software development. It is based on Git, a version control system [8]. We will use this service to keep track of changes in our project. If you plan to use GitHub, you need to follow these steps:

1. Create an **Account** on <https://github.com/>. The **Free** plan is enough.
2. Create a **Repository** on <https://github.com/new>. The **Repository Page** appears.



The screenshot shows the GitHub 'Create repository' form. At the top, there are two input fields: 'Owner' with a dropdown menu showing 'your-name' and 'Repository name' with a text input containing 'nickel-mod' and a green checkmark. Below these is a suggestion: 'Great repository names are short and memorable. Need inspiration? How about [miniature-doodle?](#)'. The 'Description (optional)' field contains the text 'A mod introducing the Nickel ore in Minecraft.'. There are two radio button options for visibility: 'Public' (selected) and 'Private'. Below these is a checkbox for 'Initialize this repository with a README' which is checked. At the bottom, there are two dropdown menus: 'Add .gitignore: None' and 'Add a license: GNU General Public License v3.0'. A green 'Create repository' button is at the bottom.

Figure 1. Creating a GitHub repository

3. Click **Clone or download**, and copy the https link. We will use it later.

★ **Note:** If you are a student, you can have the Pro plan for free (and a bunch of tools with it). Go to https://education.github.com/discount_requests/new and fill the form.

Now that your repository is created, you need to install Git on your computer, so that it will recognize the `git` command in the terminal.

1. Download **Git for windows** on <https://gitforwindows.org/>.
2. Install **Git for windows**.

Your computer knows what `git` means, so you can use it.

1. Create a **Folder** anywhere in your computer. Open it. A **Window** appears.
2. Right click in the **Window**. Select **Git Bash here**. A **Terminal** appears.
3. Type `git init` in the **Terminal**. It will create git files.
4. Type `git remote add origin https-link` by replacing `https-link` by the https link of your repository that you copied earlier. It should look like this:

```
$ git remote add origin https://github.com/YOURNAME/nickel-mod.git
```

5. Type `git pull origin master`. If you ticked *Initialize this repository with a README*, and if you selected a license, you should respectively see a file called *LICENSE* and *README.md*.

You will only use basic commands for this tutorial. Here are the commands that you will use:

- `git add .`: To stage changes of all the files (except the ones in the gitignore file).
- `git commit -m "message"`: The commit message. "added a new block", or "fixed world generation" for instance.
- `git push origin master`: To send the changes to the git repository, on the master branch.

You can also use the git plugin integrated in IntelliJ IDEA, but it is important to know how the commands work.

★ **Note:** If you want to know more about GitHub, there is a good animated guide on how to use it on <https://guides.github.com/activities/hello-world/>

2. Setting up your development environment

2.1 Java JDK, IntelliJ IDEA and Forge

Prerequisites: none

2.1.1 Java

As we use the most recent technologies in each domain, we will use the latest Java JDK version which is 13.

1. Go to <https://www.oracle.com/technetwork/java/javase/downloads/index.html> and download **Java JDK 8**. According to the time when you read this tutorial, you may need to create an Oracle account.
2. Install it.

2.1.2 IntelliJ IDEA

IntelliJ IDEA is the software with which you will write the mod. It is one of the most used in Java development.

1. Go to <https://www.jetbrains.com/idea/download/> and download **IntelliJ IDEA Community**.
2. Install it.

★ **Note:** If you are a student, you can have the Ultimate version for free. It includes almost all of the JetBrains softwares. Go to https://education.github.com/discount_requests/new.

2.1.3 Forge with IntelliJ IDEA

Now that you have everything installed, it is time to download forge and to integrate it with IntelliJ IDEA.

1. Go to <https://files.minecraftforge.net/>. On the side, click **1.14 > 1.14.4**.

2. In the middle of the page, click on **Mdk** in the “Download recommended” panel. You will get a **.zip File**.
3. Open this **.zip File**, and extract everything in your **Project Folder**. If you did not follow 1.3: Introduction to GitHub, then create a new **Project Folder**.
4. Open **IntelliJ IDEA**. Click on **Open**.



Figure 2. IntelliJ IDEA startup window

★ **Note:** You can use the Material Theme, by clicking **Configure > Plugins** on figure 2. Then search for **Material Theme UI**, and click **Install**. Then, follow the instructions.

5. Open **IntelliJ IDEA**. Click on **Open**. Then browse to your **Project Folder**, and select **build.gradle**. Click **OK > Open as a project**.
6. IntelliJ will download files. Wait for it to end.
7. Click **File > Settings**. A window will appear. Click **Build, Execution, Deployment > Build Tools > Gradle**. Make sure that **Gradle JVM** is set to “1.8”.
8. Click **File > Project Structure**. A window will appear. Click **Project**.
 - 8.1. Make sure that **Project JDK** is set to “1.8”.
 - 8.2. Make sure that **Project language level** is set to “8 - Lambdas, type annotations etc.”. Click on **OK**.
9. At the bottom of the IntelliJ window, click **Terminal**. Type `gradlew genIntelliJRuns`.
10. In IntelliJ IDEA, next to the run button, click **Add Configuration**. In the left panel, click **Application > runClient**. Next to “Use classpath of module”, select **nickel-mod.main**.

(!) **Caution:** If you receive an error while running “gradlew genIntelliJRuns” saying “Found java version {x}. Minimum required is 1.8.0_101 [...]”, it means that the **JAVA_PATH** variable is not

set correctly. It can happen if you already had installed Java. Go to Appendix 1 at the end of the tutorial. Then, restart IntelliJ IDEA.

Now you should be able to run Minecraft by clicking on **Run**. In the game, click **Mods** > **Example Mod**. You can see the details of your mod!

Warning: The game sound may be loud. You can change it by clicking Options > Music & Sounds

2.1.4 How to change mod details

You probably want to say that it was **you** that made the mod. Let's see how you can do that.

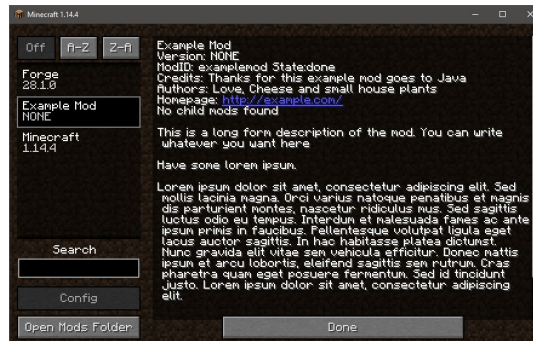


Figure 3. Default mod details

In IntelliJ IDEA, open the file `src/main/resources/META-INF/mods.toml`. Here are the fields that you need to change. We won't cover everything, only what is needed for the Nickel mod.

Field	Value
issueTrackerURL	<code>https://github.com/YOURNAME/nickel-mod/issues</code>
modId	<code>nickelmod</code>
version	<code>0.1</code>
displayName	<code>Nickel Mod</code>
displayURL	<code>https://github.com/YOURNAME/nickel-mod</code>
credits	<code>Tutorial of Lilian Gallon on game modding</code>
authors	<code>YOU</code>

description	A mod introducing the Nickel ore in Minecraft.
-------------	--

Make sure to change `dependencies.examplemod` to `dependencies.nickelmod`. Also if you are not using `updateJsonURL` and `logoFile`, comment them with “#” or delete them if you prefer.

2.1.5 Hello World

You are almost done! The last task to do is to create your own source files.

1. Delete the **example mod**. To do so, delete the folder named `com` in `src/main/java`.
2. Right click on **Java**, then click **New > Package**. Type `yourname.nickelmod`. Click **OK**.
3. Right click on **nickelmod**, then click **New > Java Class**. Type `NickelMod`. Press the **Enter** key.
4. Copy the code of [Appendix 2](#) into the `NickelMod` class.

You should be able to run the game. You will see the new mod details and some messages in the terminal coming from your mod. You are finally done! You can start programming your own Minecraft 1.14.4 mod!

2.1.6 Mod exportation

You will need to update a file with your mod information.

1. In IntelliJ IDEA, open **build.gradle**. Then, update those fields:

Field	Value	Description
<code>version</code>	0.1	The version of your mod (can be different than the one in <code>mods.toml</code>)
<code>group</code>	<code>yourname.nickelmod</code>	The package of your mod.
<code>archiveBaseName</code>	<code>nickelmod</code>	The name of your build file.

2. Type `gradlew build` in a terminal.

The file created will be called `{archiveBaseName}-{version}.jar` and is available in the `build/libs/` folder.

2.1.7 Sending Codebase to GitHub

Now that everything is ready, you can send your changes to GitHub. To do so, you need to type:

- `git add .` - to stage all the changes.
- `git add commit -m "added codebase"` - to commit them,
- `git push origin master` - to send them to GitHub. If you used the https link, you will need to enter your GitHub credentials. Otherwise you can learn how to use an SSH key [9].

2.2 Continuous Integration using GitHub Actions

Prerequisites: 1.3 Introduction to GitHub and 2.1 Java JDK, IntelliJ IDEA and Forge

You can make sure that your master branch contains a “stable” version by building it every time new code is pushed.

On your GitHub page, click on the **Actions** tab. Search for “Gradle” and click **Set up this workflow**. It should use Java JDK 1.8 by default. You need to change the “run” line by `chmod +x gradlew ; ./gradlew build`. You can customize the build name, for example, let’s use “Build MC1.14.4”. Once that you are done, click **Start commit** with “created gradle.yml”. You can add a shield in your README.md by writing:

```
![build status] (https://github.com/YOUR-NAME/nickel-mod/workflows/Build%20MC1.14.4/badge.svg)
```

It will show this badge in your README: 

3. Mod programming

3.1 How to create your first item

Prerequisites: 2.1 Java JDK, IntelliJ IDEA and Forge

Let's create the nickel ore. First, you will need to create few resource files. Go to `src/main/resources/assets/nickelmod`. Here, you need to create a few files.

Path	Value
<code>lang/en_us.json</code>	<pre>{ "item.nickelmod.nickel": "Nickel" }</pre>
<code>models/item/nickel.json</code>	<pre>{ "parent": "item/generated", "textures": { "layer0": "nickelmod:item/nickel" } }</pre>
<code>textures/item/nickel.png</code>	Go to the GitHub repository to find the nickel file.

The first file tells to the game that for the English (US) language, the item name should be "Nickel Ore". The second one tells to the game that the item model is located in `item/nickel_ore`. The last one is just the item texture.

Now that all the resources are ready, we can start coding. In `nickelmod`, create a new package called "lists", and then create a class called `ItemList` inside. This is a simple class where we will list all the custom items. Here is the code.

```
import net.minecraft.item.Item;

public class ItemList {
    // The item name must be lower case!
    public static Item nickel;
}
```

Then, we need to say to Forge that we created a new item. Go in your main class (NickelMod), and write this function:

```
@Mod.EventBusSubscriber(bus=Mod.EventBusSubscriber.Bus.MOD)
public static class RegistryEvents {
    // Function called during the items' registration
    @SubscribeEvent
    public static void registerItems(final RegistryEvent.Register<Item> event) {
        event.getRegistry().registerAll(
            ItemList.nickel_ore = new Item(
                // put all your custom Items here (separate them with a comma)
                new Item.Properties().group(ItemGroup.MATERIALS)).setRegistryName(location("nickel"))
            );
    }

    private static ResourceLocation location(String name) {
        // It tells where the resource is located
        return new ResourceLocation(MODID, name);
    }
}
```

During the Forge registration, your function “registerItems” will be called. Then you need to use registerAll(...) with the new item to add it in the game. The location(...) function will be used multiple time in the future.

Now you can run your game, and you should be able to see the new item in your creative inventory in the miscellaneous tab. Note that you can change the item group by changing ItemGroup.MATERIALS. Finally, you can push your changes to GitHub.

3.2 How to create a custom sword

Prerequisites: 2.1 Java JDK, IntelliJ IDEA and Forge, 3.1 How to create your first item

Our new sword has to be useful in the game. Let’s say that it will have a big attack damage, but a slow attack speed. The only difference with the 3.1 part is that you need to specify the item’s properties.

You will create a ToolMaterialList. It will contain the properties of all the tools using Nickel as material. In this tutorial, we will only learn how to create a Sword, but it is the same steps to create other tools (pickaxe, hoe, shovel, ...).

Here are all the properties of the nickel material.

Field	Value	Description
attackDamage	5	The damage of the nickel tools.
efficiency	9	The efficiency of nickel tools
durability	800	The durability of your tools.
harvestLevel	3	It means that you can harvest blocks that usually require diamond tools. It will only be useful if you decide to create a nickel pickaxe.
enchantability	25	You have more chances to have good enchantments if this value is low, the biggest value being 41, and the lowest 0.

And here are the properties of swords:

Field	Value	Description
attackDamageIn	4	The sword attack damage
attackSpeedIn	-3	The sword attack speed

Here are how the sword speed and the sword damage are computed:

- Attack speed: $4 + \{tool\ attack\ speed\}$
- Attack damage: $1 + \{tool\ attack\ damage\} + \{sword\ attack\ damage\}$

Now that you know all the properties, you can create the `ToolMaterialList` enum in the `lists` package. The code is available in [Appendix 3](#). Then, you can create a new `Item` in `ItemList` called `nickel_sword`. Finally, you need to register the new sword:

```
ItemList.nickel_sword = new SwordItem(ToolMaterialList.nickel_sword, 4, - 3, new  
Item.Properties().group(ItemGroup.COMBAT)).setRegistryName(location("nickel_sword"))
```

Make sure to separate the items with a comma.

You are not done yet! You need to add some resources about the sword. It is the same thing as for the custom item. Here are the values:

Path	Value
lang/en_us.json	<pre>{ "item.nickelmod.nickel_sword": "Nickel Sword" }</pre>
models/item/nickel_sword.json	<pre>{ "parent": "item/handled", "textures": { "layer0": "nickelmod:item/nickel_sword" } }</pre>
textures/item/nickel_sword.png	Go to the GitHub repository to find the ore file.

3.3 How to create a custom recipe

Prerequisites: 3.1 How to create your first item, 3.2 How to create a custom sword

This is easiest part of the tutorial. You just need to create one file inside a new directory called recipes. Here is the path: `src/resources/data/modid/recipes`

Then create a file name `nickel_sword.json`. The content should be like this:

```
{
  "type": "minecraft:crafting_shaped",
  "pattern":
    [
      "I",
      "I",
      "S"
    ],
  "key": {
    "I": { "item": "nickelmod:nickel"},
    "S": { "item": "minecraft:stick"}
  },
  "result": { "item": "nickelmod:nickel_sword"},
  "count": 1
}
```


But what are these fields?

- type: specifies that this is a crafting recipe.
- pattern: specifies the crafting pattern
- key: specifies to what "I" and "S" refer to
- results: specifies what this recipe creates
- count: specifies how many "result" we have, in this case, only one sword.

It should be working without changing anything in the code.

3.4 How to change the world generation to include a custom ore

Prerequisites: 2.1 Java JDK, IntelliJ IDEA and Forge, 3.1 How to create your first item

As an exercise, you have to create your own Nickel Ore. You can find the Nickel Ore image on the tutorial's Github (see [Resources](#)). Here are a few tips:

- 1) You will need to use a function called `registerBlocks` instead of `registerItems`

The block registration should look like this:

```
BlockList.nickel_ore = new Block(Block.Properties.create(Material.ROCK).hardnessAndResistance(50f, 3.0f).harvestLevel(3).sound(SoundType.METAL)).setRegistryName(location("nickel_ore"))
```

With:

- Material.ROCK referring to the material type, so we need a pickaxe to harvest it.
- Hardness and resistance being 50 (the same as an Obsidian) and 3 (the same as a regular Ore).
- Harvest level specifying the tool needed to mine it. Here, we need a diamond tool (or better).

- 2) You will need to use a class called `BlockList` instead of `ItemList`

- 3) The `nickel_ore.json` should be located in `models/block`, and should contain:

```
{
  "parent": "block/cube_all",
  "textures": {
    "all": "nickelmod:block/nickel_ore"
  }
}
```

4) Then, for a block, you need a new .json located in the "blockstates" folder which is next to the lang folder. Then write this:

```
{
  "variants": {
    "": { "model": "nickelmod:block/nickel_ore" }
  }
}
```

It says to use the model nickel_ore for the default blockstate.

⚠ Warning: Make sure to create a new Item called nickel_ore, otherwise we won't be able to use The block in the game! Follow 3.1 but with the nickel_ore. If you are struggling, you can find the code on the github project page.

Now that you have the new Nickel Ore, we can generate it in the world generation. You need to edit the function `commonSetup` that you created in the beginning of this tutorial. In the code below, we specify that we want the nickel ore to generate between layers 50 and 100. Also, we want 10 of them maximum per chunk, and they can appear at most in veins of 4.

```
for(Biome biome : ForgeRegistries.BIOMES) {
    biome.addFeature(
        GenerationStage.Decoration.UNDERGROUND_ORES,
        Biome.createDecoratedFeature(
            Feature.ORE,
            new OreFeatureConfig(
                OreFeatureConfig.FillerBlockType.NATURAL_STONE,
                BlockList.nickel_ore.getDefaultState(),
                4 // vein size (how many ores are attached at most?)
            ),
            Placement.COUNT_RANGE,
            new CountRangeConfig(
                10, // max number of block in this layer -> probability
                50, // minimum height
                0, // height base
                100 // max height
            )
        )
    );
}
```

4. Installation

Prerequisites: having a minecraft.net account (~\$27) 2. Setting up your development environment and 5.1 Forge installation

First, create a minecraft.net account and install Minecraft. Now, download **Forge 1.14.4 Installer** on <https://files.minecraftforge.net/> and install it. In your Minecraft launcher, change the profile, and select the one with forge.

Now, press windows key and R key in the same time. Type "%appdata%". Go to .minecraft and create a folder named "mods". You can drag your Minecraft mod in this folder.

5. Conclusion

Game modding sounds difficult until you start working on it. You learn to use a new language, to understand how a game code works, and you discover all the possibilities that it offers you. Personally, game modding helped me to learn a lot of concepts in Computer Science before having classes on it. It also pushes you to work on your own, to fix your issues, to find some hacks to make your mod working. All of these skills are really important in Computer Science, and it will help you in your studies.

Resources

- Tutorial code:
 - If you get stuck, the code used in the tutorial is right here
 - <https://github.com/shorebre4k/nickel-mod>
- Minecraft Gamepedia :
 - You can find the properties of game objects (i.e. pickaxe, world generation, ...)
 - <https://minecraft.gamepedia.com/>
- Forge Forums:
 - If you need help, this is the official forum, and there is a lot of activity
 - <https://www.minecraftforge.net/forum/>

References

- [1]: Google, "Google Developer Documentation Style Guide.", Online, accessed 13-November-2019
<https://developers.google.com/style>
- [2]: Sotamaa, Olli. "On modder labour, commodification of play, and mod competitions." First Monday. 12 (2007): n. pag. Print <https://journals.uic.edu/ojs/index.php/fm/article/view/2006>
- [3]: Sotamaa, Olli. "When the Game Is Not Enough: Motivations and Practices Among Computer Game Modding Culture.", Games and Culture 5.3 (2010): 239–255.
<https://journals.sagepub.com/doi/abs/10.1177/1555412009359765>
- [4]: Hong, Renyi. "Game Modding, Prosumerism and Neoliberal Labor Practices." International journal of communication (Online) (2013): n. pag. Print. <https://ijoc.org/index.php/ijoc>
- [5]: Gamepedia contributors, "Mods/Forge", Online, accessed 27-November-2019
<https://minecraft.gamepedia.com/Mods/Forge>
- [6]: Gamepedia contributors, "Minecraft Forge", Online, accessed 27-November-2019
https://ftb.gamepedia.com/Minecraft_Forge
- [7]: LexManos, "1.13 Announcement.", Online, accessed 27-November-2019
<https://gist.github.com/LexManos/76765455e6938892aed59544a9061321>
- [8]: Wikipedia contributors, "GitHub", Online, accessed 27-November-2019
<https://en.wikipedia.org/w/index.php?title=GitHub&oldid=928056962>
- [9]: Github help, "Connecting to GitHub with SSH", Online, accessed 03-December-2019
<https://help.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh>

Appendices

Appendix 1: How to change JAVA_PATH to Java JDK 1.8

It will be used by gradle to run its scripts.

1. Type **Environment** in the windows search bar. Click on **Edit the system environment variables**.
2. Under **System variables**, click **New**.
 - a. Variable name: `JAVA_PATH`
 - b. Variable value: `C:\Program Files\Java\jdk1.8.0_231` - It can be different if you modified the installation folder, or if you are using another JDK 1.8 version (here it says 231).

Appendix 2: Starter code

Make sure to change the first line with your package name. Also, do not forget to change the modid if you use another one.

```
package yourname.nickelmod;

import net.minecraftforge.common.MinecraftForge;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.fml.event.lifecycle.FMLClientSetupEvent;
import net.minecraftforge.fml.event.lifecycle.FMLCommonSetupEvent;
import net.minecraftforge.fml.javafmlmod.FMLJavaModLoadingContext;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

// This is the mod id. It must be the same as the one in mods.toml.
@Mod("nickelmod")
public class NickelMod {

    public static NickelMod INSTANCE;
    public static final String MODID = "nickelmod";
    private static final Logger LOGGER = LogManager.getLogger(MODID);

    public NickelMod() {
        INSTANCE = this;

        // We need to call these two functions to make sure that setup and clientRegistries are called
        FMLJavaModLoadingContext.get().getModEventBus().addListener(this::commonSetup);
        FMLJavaModLoadingContext.get().getModEventBus().addListener(this::clientSetup);

        // It tells to Forge that this mod exists!
        MinecraftForge.EVENT_BUS.register(this);
    }

    private void commonSetup(final FMLCommonSetupEvent event) {
        LOGGER.info("Hello from commonSetup!");
    }

    private void clientSetup(final FMLClientSetupEvent event) {
        LOGGER.info("Hello from clientSetup!");
    }
}
```

Appendix 3: ToolMaterialList enum

Make sure to change the first line with your package name.

```
package yourname.nickelmod.lists;

import net.minecraft.item.IItemTier;
import net.minecraft.item.Item;
import net.minecraft.item.crafting.Ingredient;

// An enum that set the properties for a material (in this case the Nickel one)
public enum ToolMaterialList implements IItemTier {

    // We create the nickel sword properties here (the basic ones)
    // Some properties are useless for a sword (ex: efficiency)
    nickel_sword(5.0f, 9.0f, 800, 3, 25, ItemList.nickel);

    private float attackDamage, efficiency;
    private int durability, harvestLevel, enchantability;
    private Item repairMaterial;

    // We init all the attributes in the constructor
    private ToolMaterialList(float attackDamage, float efficiency, int durability, int harvestLevel, int
enchantability, Item repairMaterial) {
        this.attackDamage = attackDamage;
        this.efficiency = efficiency;
        this.durability = durability;
        this.harvestLevel = harvestLevel;
        this.enchantability = enchantability;
        this.repairMaterial = repairMaterial;
    }

    @Override
    public float getAttackDamage() { return this.attackDamage; }

    @Override
    public float getEfficiency() { return this.efficiency; }

    @Override
    public int getEnchantability() { return this.enchantability; }

    @Override
    public int getHarvestLevel() { return this.harvestLevel;}

    @Override
    public int getMaxUses(){ return this.durability; }

    @Override
    public Ingredient getRepairMaterial() { return Ingredient.fromItems(this.repairMaterial); }
}
```