Project Setup

Mod Development Kit (MDK)

The Mod Development Kit (or MDK for short) contains the skeleton for starting the implementation of a new mod. It contains the following files and folders:

- *gradle/wrapper/*
- *src*
- *.gitattributes*
- *.gitignore*
- *build.gradle*
- *changelog.txt*
- *CREDITS.txt*
- *gradle.properties*
- *gradlew*
- *gradlew.bat*
- *LICENSE.txt*
- *README.txt*

The MDK can be opened as a project directly or some key files can be used to start a new project. I chose the second option, as it gives more flexibility for the setup options. The files and folders needed to start another project are:

- *gradle* - The folder containing the Gradle Wrappers
- *src* - The folder containing the source code
- *gradle.properties* - The Gradle properties file, for defining additional variables and options
- *gradlew* - The *nix shell file for executing the Gradle wrapper
- *gradlew.bat* - The Windows batch file for executing the Gradle wrapper
- *build.gradle* - The Gradle buildscript, which defines the project and tasks

The latest version of the MDK can be downloaded from the official Forge website. I chose the latest version available today (20th of April, 2022), namely version 40.1.0 for Minecraft 1.18.2.

> **JDK version**
>
> Starting with Minecraft 1.18, the game uses JDK version 17. While the gradle build can run on JDK version 16, the actual game needs JDK 17 in order to compile and run.

Updating values in `build.gradle`

After opening the new project file and configuring the JDK, certain values in the `build.gradle` file need to be updated, in order for my mod to be uniquely identifiable among the others and to specify the running configurations for the project.

The original values used for mod identification in the `build.gradle` file are presented below.

```
version = '1.0'
group = 'com.example.modid'
archivesBaseName = 'modid'
```

Since I do not own any domain, the updated values are:

```
version = '1.0'
group = 'me.tudorcoroian.armortinkers'
archivesBaseName = 'armortinkers'
```

> **Mappings**
>
> Since Minecraft ships with obfuscated names for parameters and methods, there is a parameter within the `build.gradle` file where I can specify the mappings used to deobfuscate the source code.

Up until Minecraft 1.16, the mod developers used MCP (Mod Coder Pack) mappings, which were put together by the community. With the release of Minecraft 1.16, Mojang also made available the official mappings. One problem with the official mappings is that they do not have mappings for the parameter names. As such, another possible option is Parchment, which is another community-sourced modloader-neutral set of mappings of parameter names and javadocs, that augment the official names released by Mojang. This last option also provides names for the parameters of methods.

In order to use the Parchment mappings, the following depencies must be added to the `build.gradle` file:

- under the `buildscript` directive, inside the `repositories` group

```
maven { url = 'https://maven.parchmentmc.org' }
```

- under the `buldscript` directive, inside the `dependencies` group

```
classpath 'org.parchmentmc:librarian:1.+'
```

- with the `apply plugin` tag

```
apply plugin: 'org.parchmentmc.librarian.forgegradle'
```

- with the `mappings` tag

```
mappings channel: 'parchment', version: '2022.03.13-1.18.2'
```

> **Parchment version**
>
> As of today (20th of April, 2022), the latest version of Parchment was released in the 13th of March, 2022, for Minecraft 1.18.2.

Runs

Minecraft can be ran from within the IDE (in this case, IntelliJ), with the help of the `runs` group, under the `minecraft` directive. There are three running configurations:

1. *client* - Runs Minecraft as a physical client
2. *server* - Runs Minecraft as a physical server
3. *data* - Used in data generation

Extending other mods

When developing a mod, there is the option to have other mods required. This may be done in case the mod builds on top of another or is an extension of another mod. For my project, I am implementing a standalone mod. However, in order to have access to some additional functionalities, I will add in the `repositories` directive the following:

```
maven {url "https://dvs1.progwml6.com/files/maven"} // JEI
maven {url "https://cursemaven.com"} // TOP
```

1. JEI (Just Enough Items) is a utility mod that modifies the existing Inventory GUI, by adding a list of items to the far right of the screen. This list is useful for

checking recipes and uses for items, and can also be used to search for items in opened GUIs. JEI has a lot of addons, generally adding extra modded recipe support like Thaumic JEI.
2. TOP (The One Probe) is another utility mod that adds an item called *The One Probe*. When this item is being held in hand by the player, it gives information on the block or entity the user points it at in an on-screen tooltip in the upper left corner. This includes the name, the mod that adds it, harvestability, health, energy, and many other attributes. The One Probe is inspired by WAILA (What Am I Looking At), and designed to be a more immersive version of it. The option `needsProbe`, toggled in The One Probe's config file, determines whether or not the item is needed to view the information.

Also, under the `dependencies` directive, I added the following lines of code:

```
compileOnly fg.deobf("mezz.jei:jei-${jei_version}:api")
implementation fg.deobf("mezz.jei:jei-${jei_version}")

implementation fg.deobf("curse.maven:the-one-probe-245211:3550084")
```

The `jei_version` variable is defined in the `gradle.properties` file, as follows:

```
jei_version=1.18.2:9.5.0.125
```

Generating IntelliJ Runs

After I made all the configurations above, the last step is to generate the running configurations for my IDE (in this case, IntelliJ). This is done with one of the gradle tasks, under *forgegradle runs*, namely the *genIntelliJRuns* task.

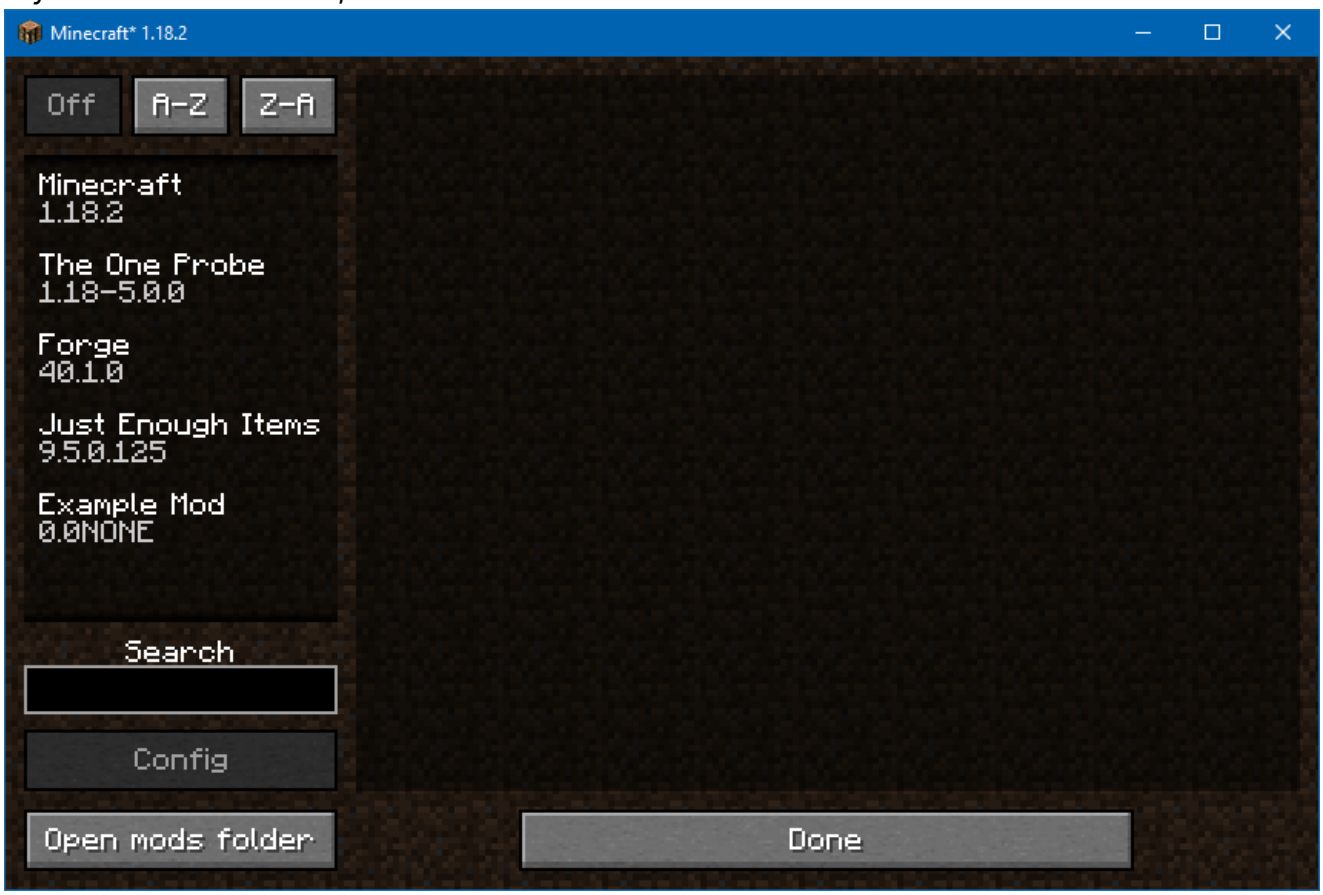
> **IntelliJ IDE bug**
>
> Due to unkown reasons, the IDE (IntelliJ) reports the error `Invalid value: -1`, with no additional information, even if the build is executed successfully.
> As such, I will run all *gradle* commands from the terminal. This way, the IDE does not raise any errors and the game is launched.

The commands used to build and generate the running configurations are:

```
./gradlew build --warning-mode all              # used to build the .jar
./gradlew genIntellijRuns                               # used to generate
```

```
the running config
./gradlew runClient --warning-mode all  # used to launch the application
```

After running `./gradlew build`, a `.jar` of the project will be generated in `build/libs/` with the name `armortinkers-1.0.jar`. Also, after launching the game, I can already see my mod loaded within, under the **Mods** menu.



As it can be seen in the image, there are 5 mods loaded in the game:

1. *Minecraft 1.18.2* - Vanilla Game
2. *Forge 40.1.0* - The Forge framework used to load all the other mods
3. *The One Probe 1.18-5.0.0* - Mod added by me as a dependecy
4. *Just Enough Items 9.5.0.125* - Mod added by me as a dependecy
5. *Example Mod 0.0NONE* - The mod I am developing

The reason my project appears like this is that I have not modified the mannifest file yet.