acm @osu

Minecraft Modding 101

Starting with Java

- Written in the Java programming language
- Java is an **compiled and interpreted** language run on the Java Virtual Machine (JVM)
- Compiles into Java Bytecode (.class)
  - Assembly like instructions for the virtual CPU
  - Can be decompiled back into source code (.java)
- Packaged as a normal zip archive (.jar)
  - Can be opened and inspected like normal zips
- Classes loaded at runtime into the JVM


```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello Minecraft");
    }
}
```

Java Bytecode Example

```
// access flags 0x1
public b(Lgzc;)Z
 L0
  LINENUMBER 189 L0
  ALOAD 0
  ICONST_0
  PUTFIELD guz.T : Z
 L1
  LINENUMBER 190 L1
  ALOAD 0
  ALOAD 1
  INVOKESPECIAL gti.b (Lgzc;)Z
  IRETURN
 L2
  LOCALVARIABLE this Lguz; L0 L2 0
  LOCALVARIABLE $$0 Lgzc; L0 L2 1
  MAXSTACK = 2
  MAXLOCALS = 2
```

Bytecode

```
public boolean b(gzc $$0) {
    this.T = false;
    return super.b($$0);
}
```

Decompiled Source Code

Obfuscation

- Problem: The Minecraft code is **obfuscated**
  - Very difficult to read/understand
  - Unstable and changes every version
  - Protects the Mojang codebase
  - Reduces the size of the game

- Solution: Mappings
  - Community reverse engineered the codebase
  - Created stable, readable names for every class and function
  - Examples:
    - Minecraft Coder Pack (MCP) - 2010
    - Yarn Mappings - 2018
    - MojMap - 2020 Official Mappings provided by Mojang
- Mojang de-obfuscates the game in 26.1
  - Q1 2026

Mapping Example

```
public boolean b(gzc $$0) {
    this.T = false;
    return super.b($$0);
}
```

```
public class StonecutterScreen extends AbstractContainerScreen<StonecutterMenu> {
    @Override
    public boolean mouseReleased(MouseButtonEvent mouseButtonEvent) {
        this.scrolling = false;
        return super.mouseReleased(mouseButtonEvent);
    }
```

Yarn Mappings

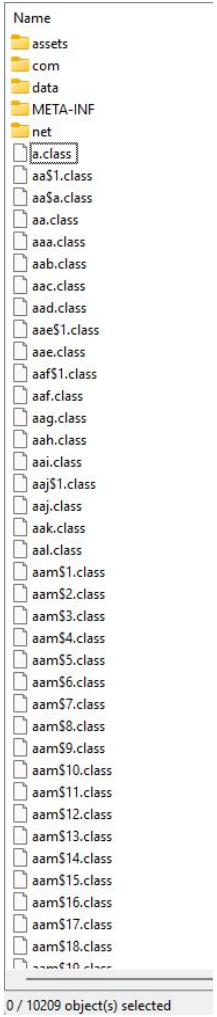
Modding the game

Actually pretty straight forward

1. Open the minecraft.jar
2. Decompile the .class bytecode into .java source code
3. Add your new logic
4. Recompile back into bytecode
5. Drag it back into the jar and replace the old class
6. Run the game
7. It doesn’t work
8. Delete META-INF
9. Success!

Issues with “Jar Modding”

- Jar Modding - directly modifying the Minecraft jar file
- Compatibility
  - Any mods the needed to modify the same .class files were incompatible
  - Most mods were incompatible. Rarely more than 2 or 3 at once
    - Ex: Only one mod could edit Blocks.java
- It was annoying to do



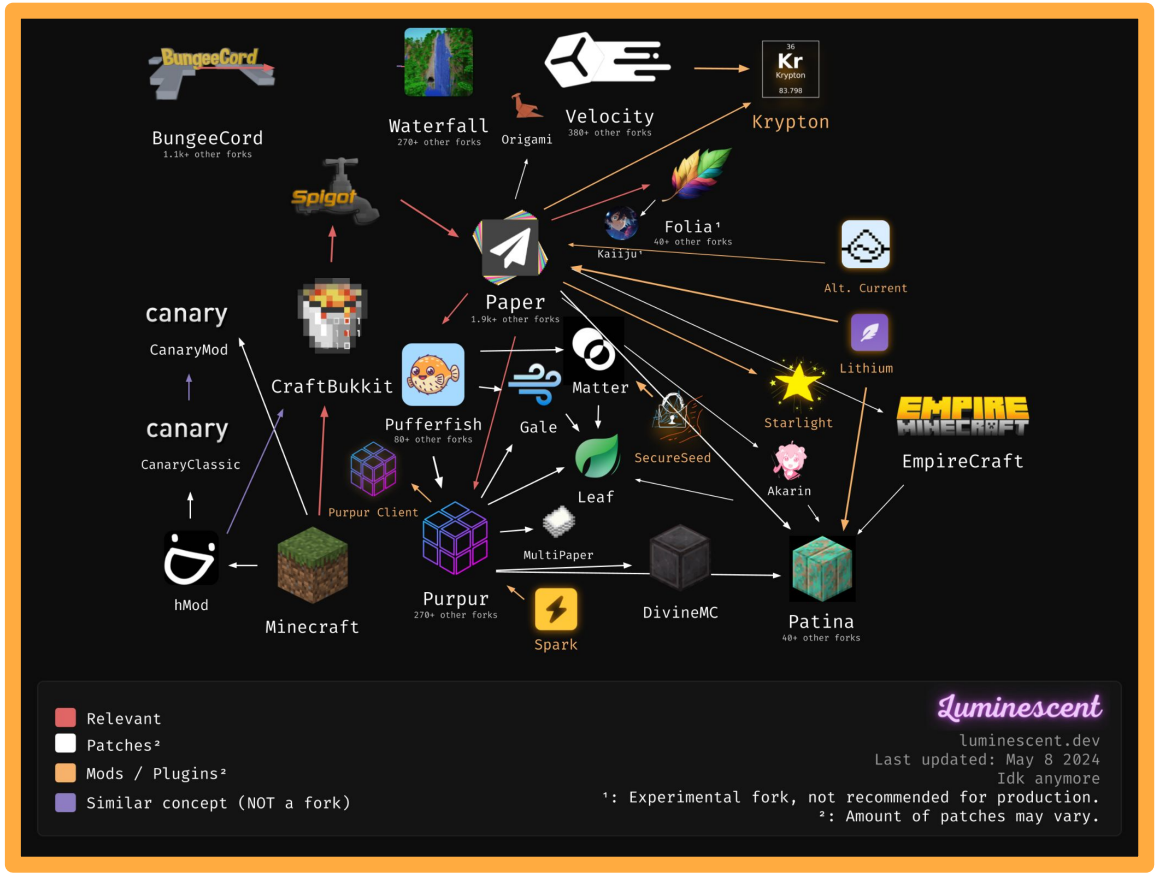
Early Mod Loaders

- Risugami's ModLoader
  - Created a base mod for other mods to talk to
  - Patched common functionality classes
  - Added API methods for other mods to use
  - Ex:
    - Don't modify Blocks.class to add a block
    - Call ModLoader.registerBlock(...)
    - ModLoader kept a list of all registered blocks and injected them for you
  - Allowed compatibility between between mods that added blocks, items, recipes, entities, etc.
  - Loaded mod files from /mods folder
  - Became the standard mod loader for a long time
  - Problems:
    - Client-side only. Server codebase was separate and annoying
    - Anything beyond what the basic hooks supported still required overwriting classes **CONFLICTS**

Server Modding

- Minecraft server was a separate codebase
- hMod
  - Added command system
  - Added support for teleporting, banning, etc.
  - Created a basic plugin system
- Bukkit
  - Created a large API that could be used by plugins without directly interfacing with Minecraft at all
  - Developed by “Dinnerbone” and “Grumm” who were then hired by Mojang
    - Their usernames are easter eggs in game
  - Only allowed simple modifications that would still allow non-modded clients to join. No new features. Limited by the existing protocol
- **Lots and lots of forks**

Server Mod Fork Graph



Forge Mod Loader

- Fixed block texture limit
  - Minecraft stored all block textures in a 256x256 atlas (16x16)
  - Forge dynamically stitched block textures together into an atlas at runtime
- Added tons of new API features
  - Fluid system
  - Ore Dictionary (10 different mods that add copper are compatible)
  - Event bus (mods can react to different hooks)
  - Methods for modifying most of the game safely
  - Provides access to ASM library for advanced bytecode manipulation
- Unified Client and Server mods
- Became the new standard

1.13 - The Flattening

- Pre-flattening
  - Everything had hard coded magic number IDs
    - Stone = 1, Grass = 2
  - Only 4096 slots
  - Forge *mostly* handled conflicts and compat, but it wasn't pretty
- The Flattening
  - In 1.13 Mojang rewrote the whole engine
  - Got rid of all numeric IDs and metadata
  - Created new BlockState and named registry system
  - 35:14 -> "minecraft:red_wool"
- Forge is very broken and must be practically rewritten
  - Takes **6 months** to update

Modern Era - Fabric

- Waiting for Forge to update
- Do we actually need a massive API or just a way to load classes?
- Provides a minimal mod loader
- Supports the new Mixin framework for easy compatible bytecode manipulation
  - Developers can now easily modify Minecraft code anywhere without replacing whole classes
- Updates to new Minecraft versions instantly because it doesn't need to wrap the whole game
- Provides an optional Fabric API mod with some common hooks

- (Forge also forked and is now NeoForged)

Demo/Workshop!

- Creating a fabric mod
- Change the splash text
- Add an item
- Other stuff?

Resources

- https://fabricmc.net/develop
- https://docs.fabricmc.net/develop
- https://wiki.fabricmc.net/tutorial:start