3/22/2020

# Feature Planning

Project Deliverable 3

Andrew Yue-Keung Leung, Brian Ning Yu Chim, David Fernandes, Patrick Angelo Del Rosario Ocampo, Sameed Sohani

LES PROGRAMMEURS

# Table of Contents

# Issue Screening

## **"New" Issue:** *Revamping Forge's Plant System*

**Relevant Links:**

- https://github.com/MinecraftForge/MinecraftForge/issues/5371
  - Follow-up to our PR - https://github.com/MinecraftForge/MinecraftForge/pull/6573
- https://github.com/MinecraftForge/MinecraftForge/pull/5362

Based on our bug fix for issue 6286, a developer commented that rather than accepting the interim fix for the issue, a better approach would be to instead start work on revamping (some of) the entire plant system in Forge. From the discussions in issue 5371, it is evident that some work has already been done back in version 1.13 for Forge, but in interest of version 1.14 coming out at the time (keeping in mind we are now in version 1.15), the work was halted due to still needing major changes on top of compatibility. As it stands currently, the plant system was not reworked in the newer versions and remains as they were. The primary issue behind Forge's plant system is that it is very limited and does not allow modders to interact extensively with plants. Additionally, there are certain design aspects within the Forge plants which could be updated given the current modder environments.

On the problem of limited ability for modders to use the Plant APIs, one of the examples is that modders are unable to define custom harvesting methods for plants and crops. Modders are also unable to define custom "fertilizer" for crops which can be used to accelerate their growth. With respect to outdated design aspects, one example would be the PlantType enum. Originally created back in 2012, the Forge developers did not anticipate modders adding in new custom plant types. As a result, PlantTypes were created just as enums based on vanilla Minecraft plants. If modders wanted to add in new plant types, they would have to use one of Minecraft's plant types or use Forge code that was eventually updated to use ASM / reflection in order to add it in. Instead of doing this, the Plant Types could be redesigned to follow an interface so that modders can implement them as needed.

Rather than doing a full redesign of the Forge plant system, we anticipate that more realistic changes (given our timeframe) we could attempt on this issue would be to instead add in a few new features (based on the implementation specs provided on issue 6371) which would also require some redesigning of the Forge plant system.

## **Issue #6500:** *[1.15.2] PlaySoundEvent#getVolume NPE*

**Link:** https://github.com/MinecraftForge/MinecraftForge/issues/6500

Forge has many events that modders can make use of in order to trigger specific behaviour for their mods. For example, if a chicken were to spawn on a specific block and a mod wanted to trigger some sort of custom behaviour as a result, a modder could make use of the LivingSpawnEvent on MinecraftForge's EventBus in order to get conditions of the event in which that chicken spawned and override, ignore or pass interactions of the chicken on the block.

In Minecraft, there is an event (PlaySoundEvent) that gets fired when a sound is queued up to play in the game. This event is necessary for developers that wish to make mods that replace game sounds or inject custom sound files. For example, if I as a developer wanted to make a mod that replaced the default sounds of chickens in the game, my mod would need to listen in on the event that gets fired right when a sound is queued up to be played. At this point, I would have to make a check to see if the sound in the queue was being made by a chicken, in which I would then write the necessary code to swap out the default audio with my own.

The current implementation of Forge's PlaySoundEvent is causing issues at runtime – whenever this event is called upon by some mod, null pointer exceptions can be raised by attempting to access certain uninitialized fields (which are initialized immediately after the Event is posted). The issuer of this feature is requesting for some sort of fix in order to have proper behaviour that allows developers to create mods based on sounds that are made in-game. Moving the event below this initialization point would prove incorrect – the sound file is loaded on the same line of code, rendering the event neutered if moved. This hints to us the necessity for a new implementation of the sound event that does not face the same runtime issues as before.

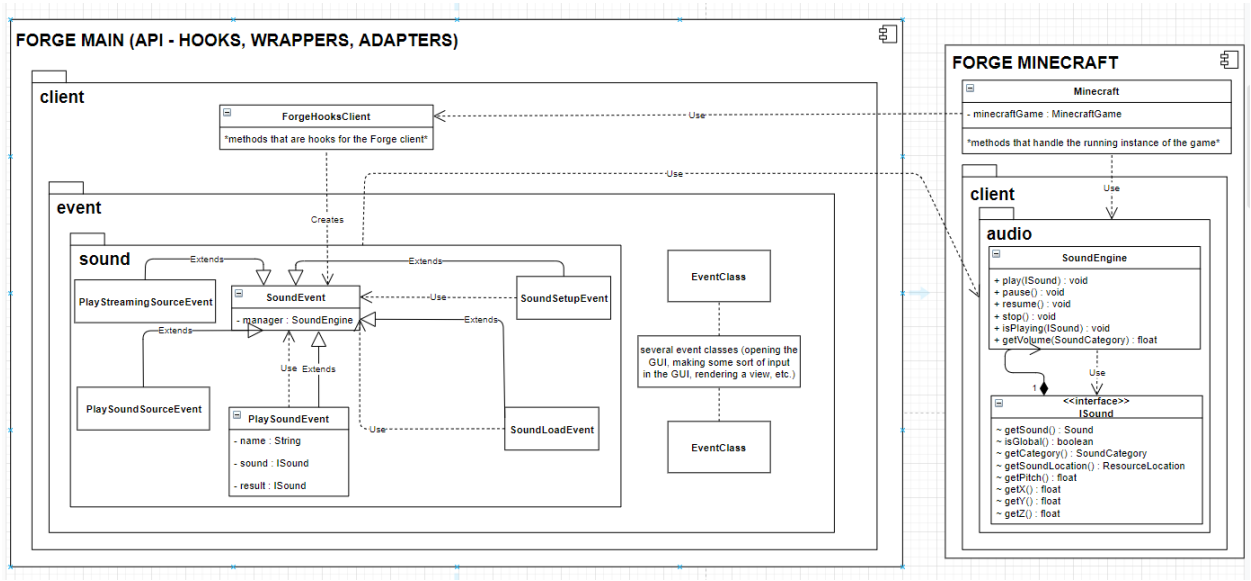
# Issue Design

## **Issue #6500:** *[1.15.2] PlaySoundEvent#getVolume NPE*

**Link:** https://github.com/MinecraftForge/MinecraftForge/issues/6500

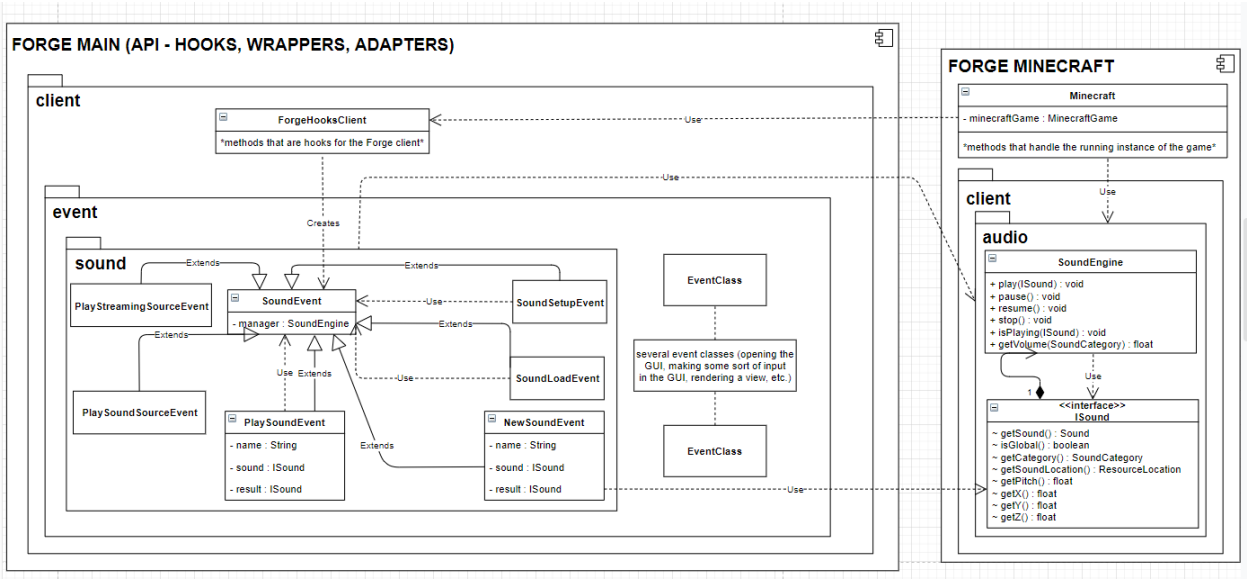
For this particular feature, we would need to make changes to the code that handles the creation and firing of events (namely for the events related to sounds). Under the client.event.sound package, there exist event classes related to sound events (a sound is made, loaded, created by a specific entity, etc.). These events extend a parent called SoundEvent, which consists of methods that allow modders to get the sound manager and specific properties related to the Sound object(s) (volume, source, name, etc.). Ideally, we would need to create a new event class (in addition to the existing sound events) that extends the SoundEvent. This particular event will get fired at the time the Minecraft sounds are instantiated, so as to not yield any null pointer exceptions. For this, it will have to be dependent on the ISound interface (i.e., dependent on the type of Sound, since all sounds implement this interface). Modders would then be able to take the necessary properties of the sound from the given event.

BEFORE *(for full image, please refer to **sound-issue_before.png**)*




*Forge Main (focused on sound and event related components)*

AFTER *(for full image, please refer to **sound-issue_after.png**)*




*Forge Main (focused on sound and event related components)*

## **"New" Issue:** *Revamping Forge's Plant System*

**Relevant Links:**

- https://github.com/MinecraftForge/MinecraftForge/issues/5371
  - Follow-up to our PR - https://github.com/MinecraftForge/MinecraftForge/pull/6573
- https://github.com/MinecraftForge/MinecraftForge/pull/5362

For this particular feature, we would need to make changes and additions to the code that interacts with plants in Minecraft. Under the *common* package in **Forge Main**, there exists the IPlantable interface and the PlantType enum. These are currently the components that modders can interact with in order to add their own custom plant types and plant blocks. In order to extend the functionality and allow modders to do more with their plant mods, we have replaced both the IPlantable interface and the PlantType enum with a PlantType interface. All the enums are now classes that implement this interface and they consist of methods that can extract and declare more information about a plant type. The old implementation did not allow for modders to create their own custom plant types (as per our PR 6573, listed under the relevant links). This new implementation would allow for modders to create their own plant types with ease and not have to worry about any null pointer exceptions. They can also specify more information and behaviour for their custom plant types.

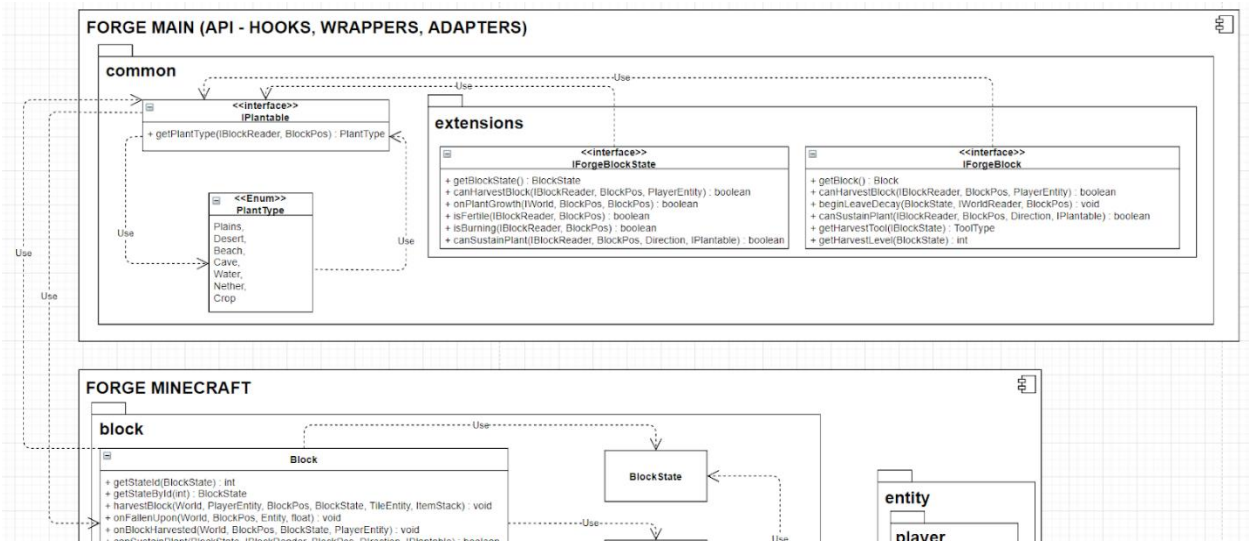
In addition to the **Forge Main** API, there exists the Block object and the IGrowable interface under **Forge Minecraft**. Modders would have to extend and implement these components in order to create their custom plant blocks. For this feature, we have included

more functions that grant the user more access to information they did not have previously (getting a plant's plant type, its age, the seeds that it can drop in addition to the plant block itself, etc.).

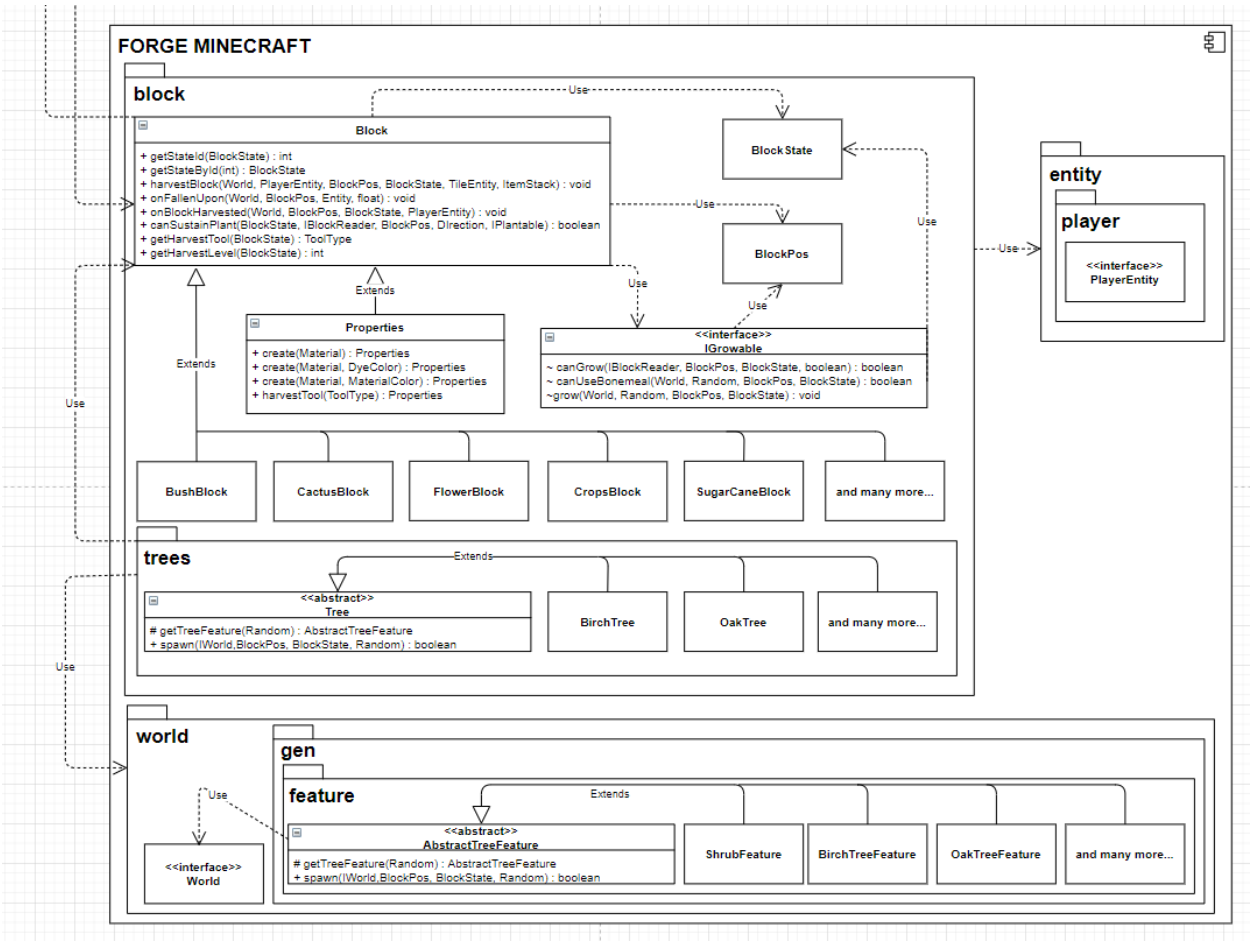
Beyond this, we also want to allow modders to add different types of fertilizers so that they have more control over their plants. Currently, many growable Minecraft plants implement the IGrowable interface. Within this interface, they must specify whether or not the bone meal item (fertilizer) can be used on the plant in question. Rather than making bone meal the only type of fertilizer, we want to instead generalize bone meal to become an implementation of a fertilizer so that Minecraft modders are able to specify exactly what kind of item they would like to use as fertilizer and not be limited to just bone meal. Furthermore, we also plan to add the ability to specify how potent these fertilizers are. By adding this, we allow the modders more customizability to how their plants interact with their desired fertilizers.

Note: The UMLs below focus on the structure of the Plant system and any (relevant) components they may interact with in the codebase.

BEFORE *(for full image, please refer to **plant-issue_before.png**)*

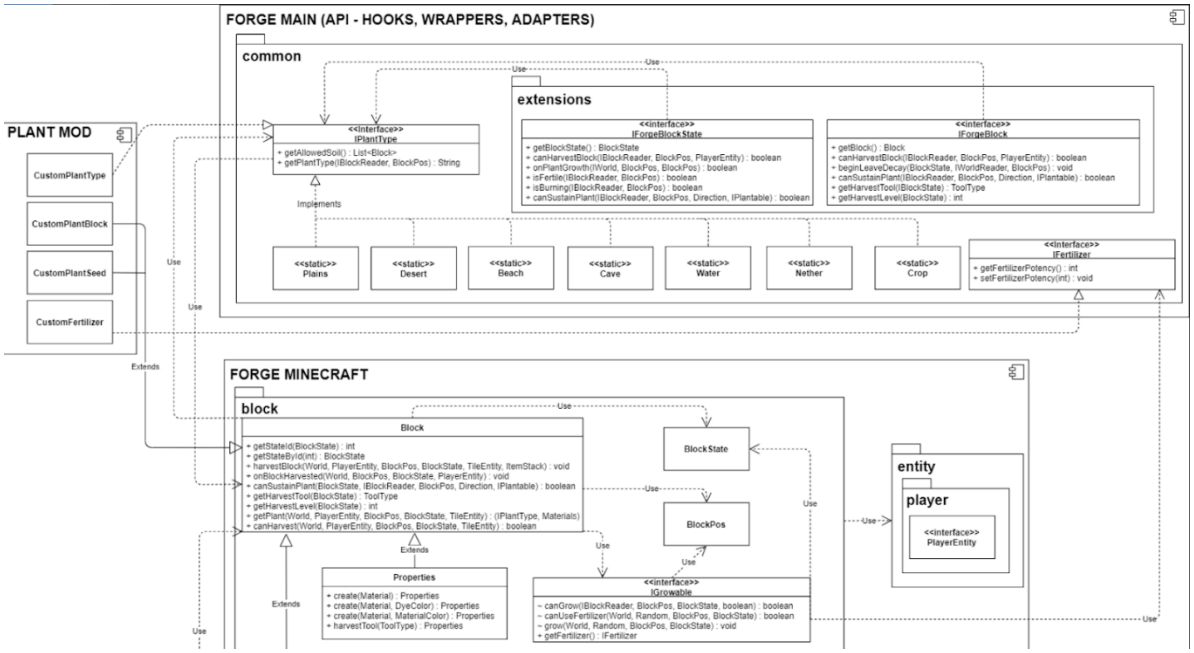



*Forge Main (focused on plant-related components)*

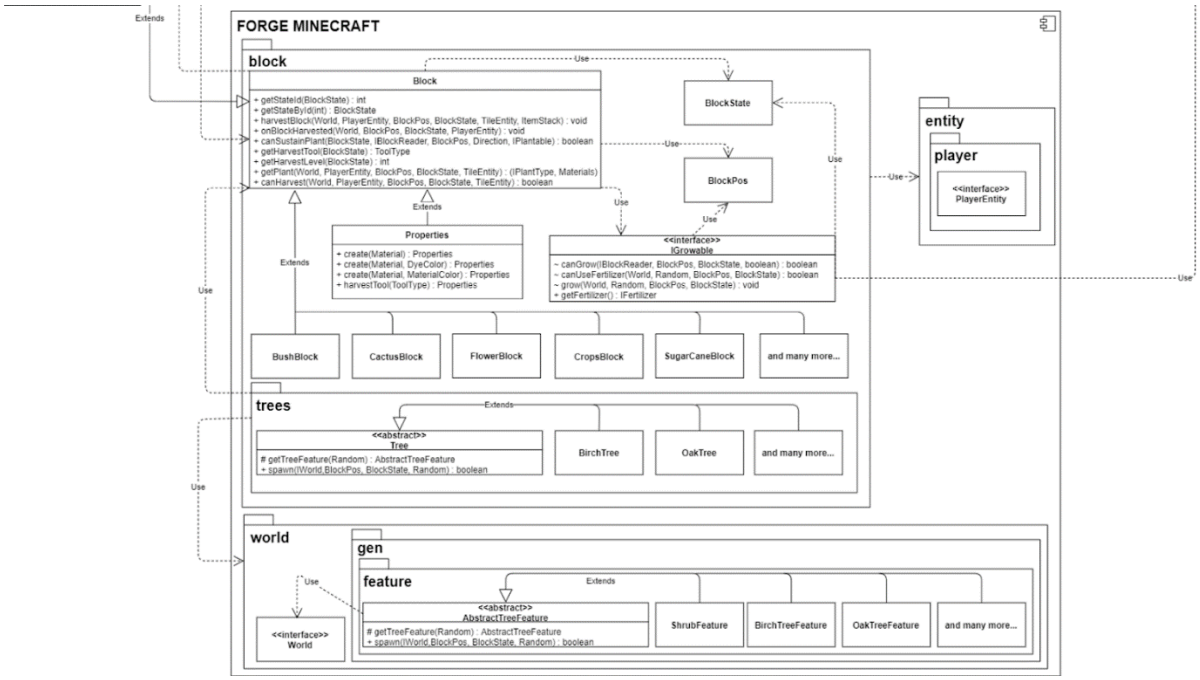



*Forge Minecraft (focused on plant-related components)*

AFTER *(for full image, please refer to **plant-issue_after.png**)*




*Forge Main (focused on plant-related components)*




*Forge Minecraft (focused on plant-related components)*

Note: As part of our redesign of some of the plant systems, we will be removing (now) unnecessary classes. However, in the actual code, these will instead simply become deprecated because we only have access to Forge code and cannot realistically assume that existing modders and their respective mods will be unaffected by these new changes.

## Issue Selection

Of the two issues we've investigated, we decided that we would go forward with some of the revamp and reimplementation of Forge's Plant System. This choice comes as a result from looking at the current system concerning Forge plants and having mutual thoughts with the issuer of the PR. Unlike the other issue, going forward with a revamp of the plant system would open up some new endpoints for developers to make use of when they're creating plant-based mods. Indeed, the inclusion of a new event provides flexibility for modders that want to change the behaviour of in-game sounds, but this pales in comparison to the importance and cruciality of updating the abilities of plants in the game. In other words, addressing this issue would have more of an impact on Forge compared to the sound event issue (improving a core system vs. the inclusion of a new sound event that prevents NPEs). Also, we believe that this issue fits the given timeline appropriately in the sense that we do not have too much nor too little to work with. From what we've estimated for both issues, the sound event issue can be addressed very quickly and would not require a lot of time to conduct the appropriate development as well as testing.

Another interesting reason is that one of our team members, David, is an avid Minecraft modder. One of his mods, ExtraFood, relies heavily on extending minecraft's code then overriding most of the base functions to implement custom plants as well as reflection in certain areas. By reworking the current Forge plant code, reflection would no longer be necessary for David's mods – changes needed for a majority of plant-based Minecraft mods.

## Acceptance Testing

With regards to acceptance testing, there are two sets of tests we can split them into – regression and feature-based testing. Regression testing will be done to ensure that a rework of the plant system did not break existing Forge code, and feature testing will test the new Plant API endpoints exposed to modders that will be implemented with the rework.

The following are regression (acceptance) tests that a user can run through in order to check that the program works as expected:

- Launch the game and enter “Creative Mode”, i.e., sandbox mode
    - For each of the Minecraft trees, check that they are able to be planted and can grow with bone meal.
        - The user adds a sapling for each tree type into their inventory (Birch, Oak, Spruce, Acacia, Jungle, and Dark Oak) from the Creative inventory
        - The user places the sapling on the appropriate land (i.e. on a grass/dirt block in an open plain)
        - The user adds bone meal into their inventory from the Creative inventory and equips it
        - The user proceeds to right click each of the planted saplings
        - At this point, the user should expect to see fully grown trees of each type
    - For each of the Minecraft flowers, check that they can be planted on the respective correct soil
        - The user adds each type of flower into their inventory from the Creative inventory
        - The user places the appropriate flower on the appropriate land (i.e. on a grass/dirt block in an open plain)
        - The flower should plant with no problems
    - For each crop plant in the game (carrots, wheat, potatoes, pumpkins, melons, and sugarcane), check that they can be grown appropriately
        - The user adds a hoe, shovel, and a bucket of water into their inventory from the Creative inventory
        - The user finds a plot of land with grass and dirt, and digs out a 1 x 10 row of dirt
        - The user fills up the hole with water
        - The user fertilizes the dirt blocks adjacent to the water by equipping the hoe and right clicking said blocks
        - The user plants each of the seeds into the fertile land
            - Sugar cane would be an exception - this must be planted on a sand block adjacent to a water block
        - At this point, the user should expect to see fully grown crops.
    - For each of the other Minecraft plants, (Cactus, Seagrass, Sea Pickle, Coral), check that they can be planted on the respective correct soil and that they grow properly
        - The user adds a cactus, seagrass, and sea pickle block into their inventory from the Creative inventory
            - The user will need to place the Cactus on some sand block in the open

        - Seagrass will need to be placed on either dirt or sand underwater
        - Sea Pickles can be placed anywhere
        - Coral blocks can be placed anywhere
    - At this point, these plant blocks should be present on the blocks they were placed in
        - For coral blocks → if they were placed on dry land, they should die in a few seconds and turn grey in colour; if they were placed underwater, they should retain their colour/vibrance
        - For sea pickles → an attempt should also be made to place multiple sea pickles on a single block → the light levels (i.e. brightness) should increase

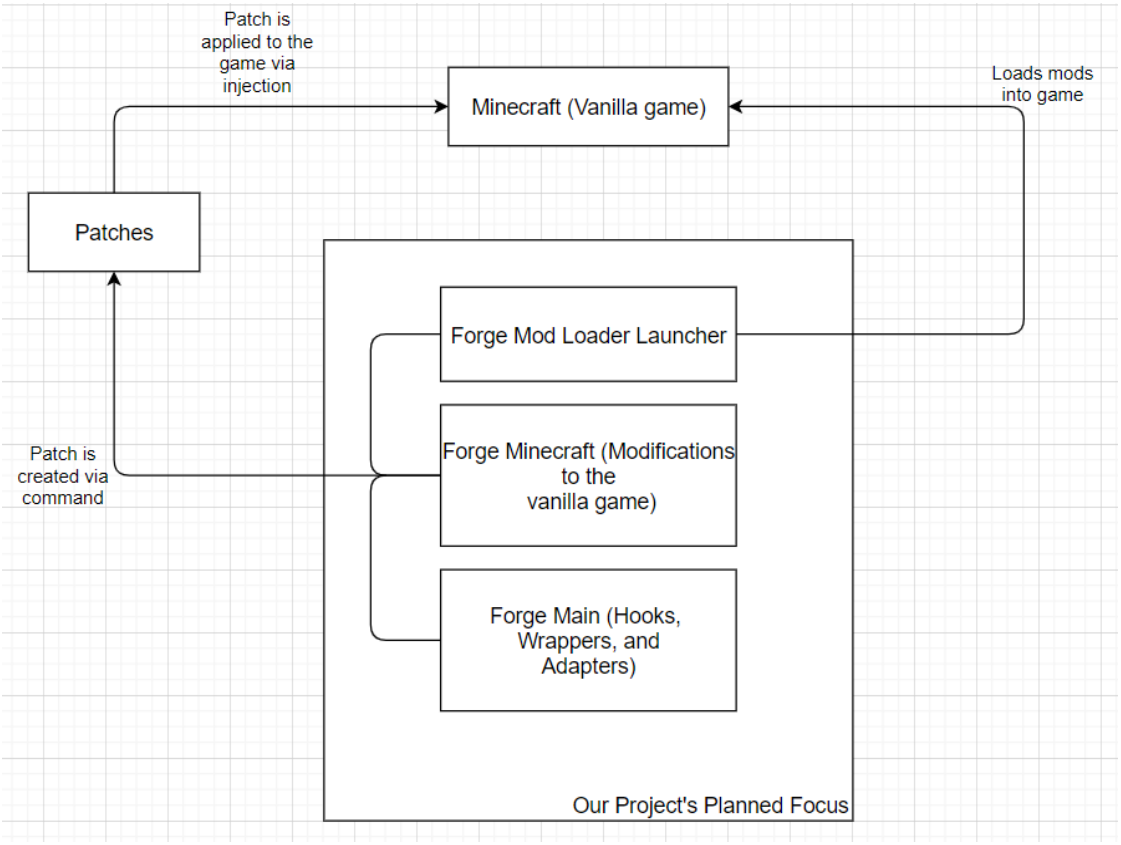
The following are feature (acceptance) tests that a user can run through in order to check that the new features are working as expected:

- Create a mod that makes a custom plant of a custom plant type which can be planted on one vanilla block and one custom block.
    - Launch the game and enter "Creative Mode", i.e., sandbox mode
        - The user adds the custom plant, the vanilla block, and the custom block into their inventory.
        - The user attempts to place their custom plant on the vanilla block and the custom block.
        - At this point, the plant should place correctly.
- Create a mod that makes a custom fertilizer with two times potency (using one of this fertilizer should be equivalent of using two of a regular fertilizer like bone meal)
    - Launch the game and enter "Creative Mode", i.e., sandbox mode
        - The user adds wheat seeds, the custom fertilizer, dirt, and a hoe to their inventory.
        - The user places the dirt on the ground and then right clicks it with the hoe, tilling it.
        - The user plants the seeds and then right clicks on it with the fertilizer
        - The process should be repeated until the user notices an instance where the fertilizer has caused the wheat to grow two stages rather than just a single stage
- Create a mod that makes a custom plant block which can be planted on vanilla blocks and harvested only with a custom block in hand.
    - Launch the game and enter "Creative Mode", i.e., sandbox mode
        - The user adds the custom plant block and the custom block to their inventory.
        - The user plants the custom plant block on a vanilla block

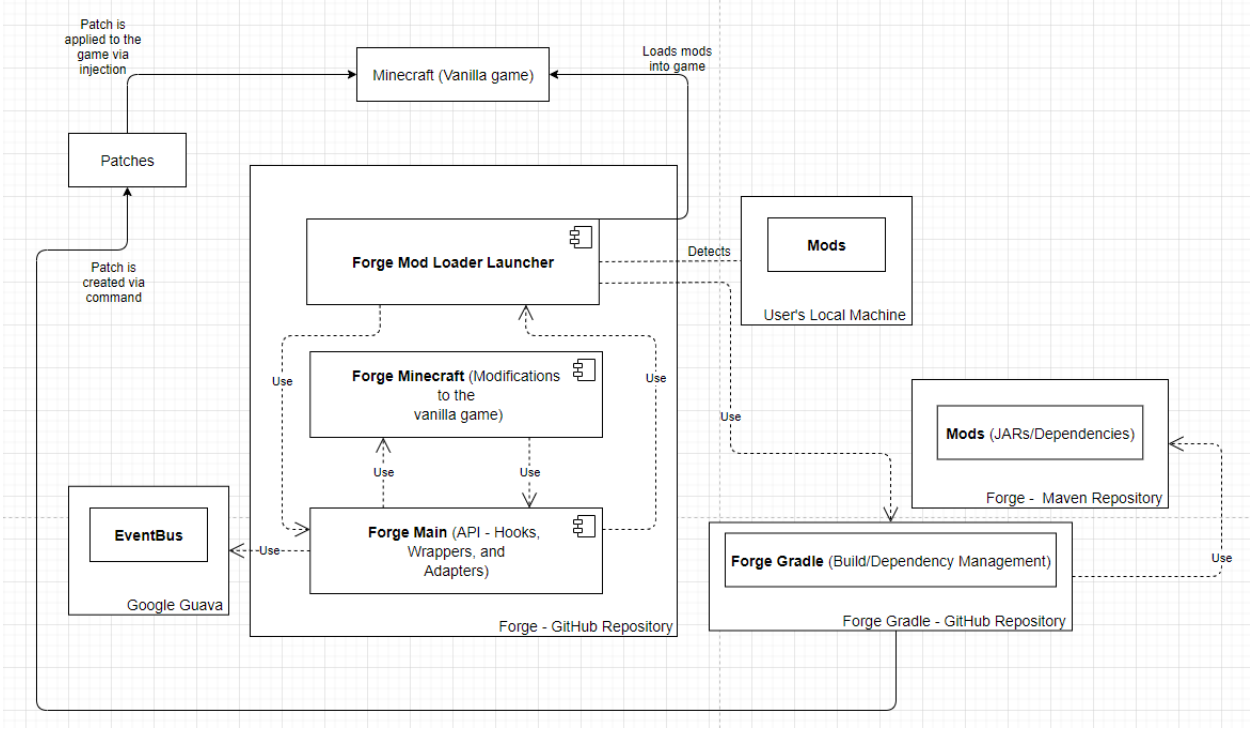
- The user equips the custom block and attempts to harvest the custom plant block.
- At this point, the custom plant block should be harvested and found in the user's inventory
- The user should then plant the custom plant block on a vanilla block again
- The user should not equip anything and attempt to harvest the plant with just their hand.
- At this point, the custom plant block should not be harvested and should still appear on the ground.

## Architecture Revisited

Back in deliverable 1, we had the following high-level diagram for the overall architecture:



In addition, we had several sub diagrams that encapsulated each of the three main components of Forge. With the knowledge we've gained from working on the project and sifting through the codebase, the overall architecture of the project can be described as the following *(for full image, please refer to **overall-arch.png**)*:




Minecraft Forge follows a simple layered architecture amongst its three main components, with reliance on several external packages/modules.

Initially, we have **Forge Minecraft** – the component that consists of all the modifications that Forge developers have made (and can continue to make) to the base game of Minecraft (i.e., the vanilla version). These modifications are general behavioural changes that allow for easy interfacing between Forge's API and a modder's code.

Moving on, we have **Forge Main** – the component that consists of all the code for the wrappers/adapters and hooks. This can also be regarded as the API that allows modders to interface with the code in Minecraft. The notable design feature about Forge as a whole is the fact that this component makes use of many wrappers that aid with the exposure of the Minecraft code to modders/developers (more on this will be discussed later on). It also makes use of Google Guava's **EventBus** module by creating events that mods can subscribe to (i.e., listen in on). This allows for modders to easily create Minecraft mods since it allows for them to

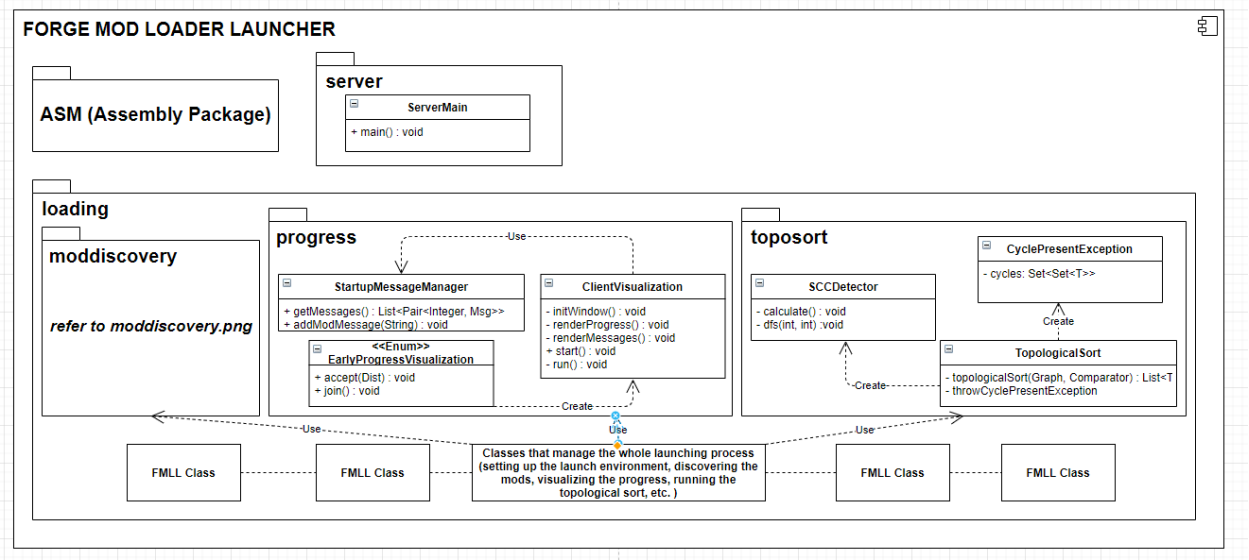
listen in on in-game events and write code accordingly to the behaviour they wouldn't see otherwise without the API.

Lastly, we have the **Forge Mod Loader Launcher (FMLL)** – the component that brings every component together by running the necessary tasks to have **Forge Main** wrap over the vanilla game and inject the changes introduced in **Forge Minecraft**, load the mods, and run the game. **FMLL** first begins by detecting the mods on a user's local machine, in which it then loads them accordingly into the game. At this time, it also has to make sure that all the dependencies and JARs are included – the **Forge Gradle (aka the build/dependency manager) framework** is used in this process and pulls the necessary JARs/dependencies from **Forge's Maven Repository**. With the changes that have been made to the vanilla game, **FMLL** then proceeds to create the necessary patches to be applied to the game (via **Forge Gradle**). Upon building all the code and applying all the patches (via injection/reflection), the game is then ready to be played with the mods that have been included by the user.

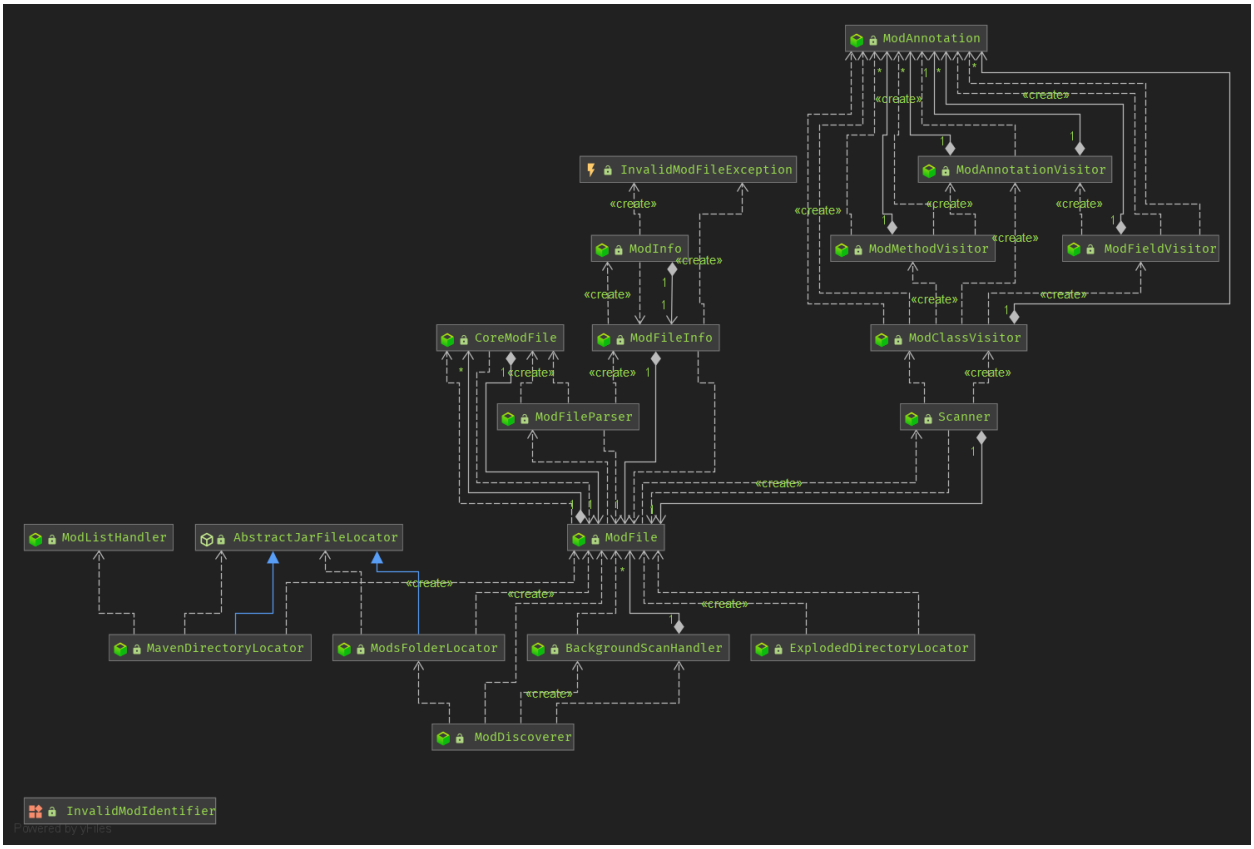
Forge Mod Loader Launcer – src/fmllauncher

Recapping from deliverable 1, the Forge Mod Loader Launcher is Forge's launcher that is in charge of detecting all the mods present in a user's directory and loading them into the game. The packages can be described as follows:

- **moddiscovery** → detects all the mods to be loaded into the game. It traverses the directory that the users are supposed to place their mods in and parses / validates each JAR file to check if they can be loaded into the game.
- **progress** → gives the client a visualization of the mod loading progress.
- **toposort** → sorts the mods being loaded into the game so that everything is loaded in properly without any errors. This is because if mods are dependent on one another, it may break if one mod is loaded prior to a mod it is dependent on.



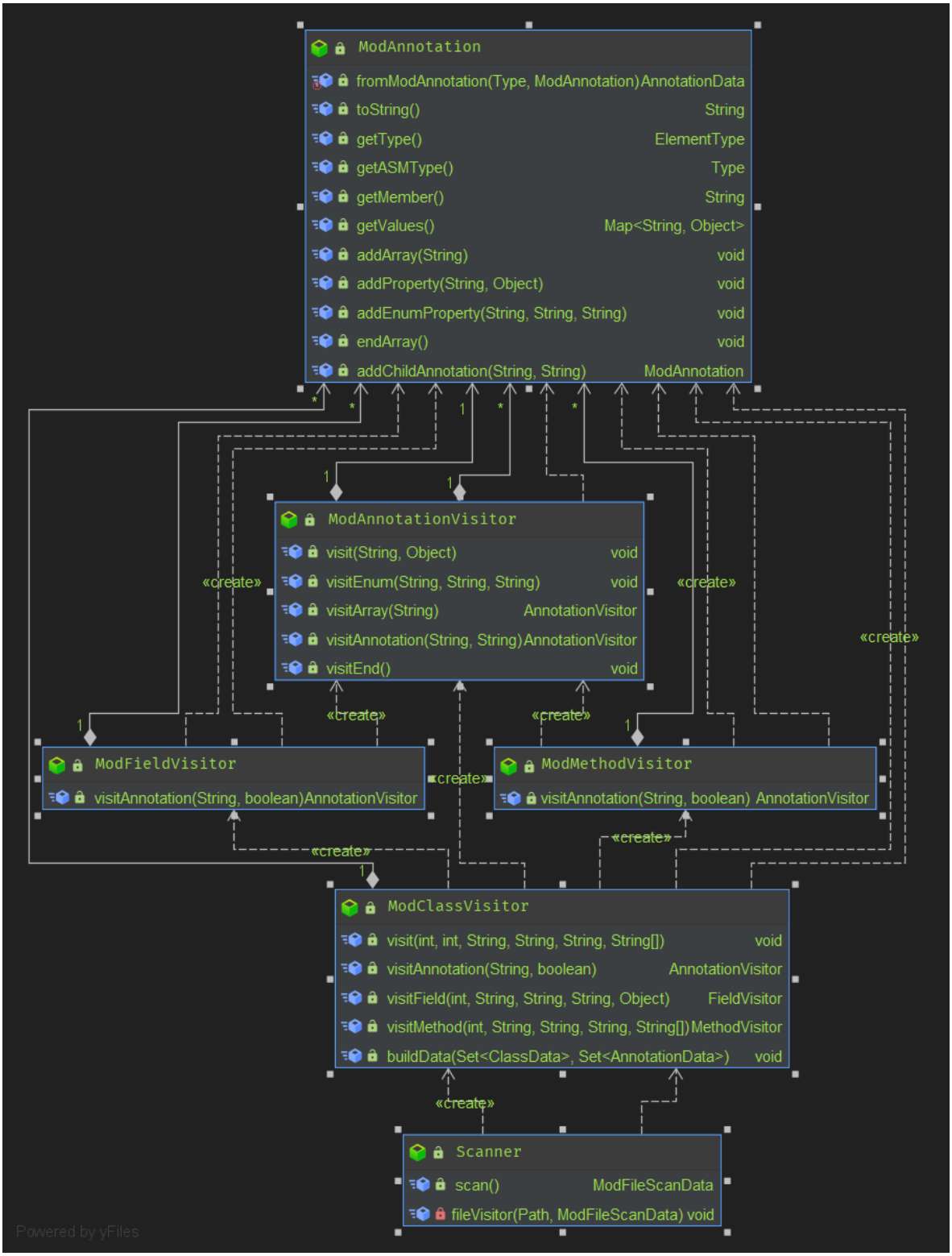

*High Level View of the FMLL component structure*



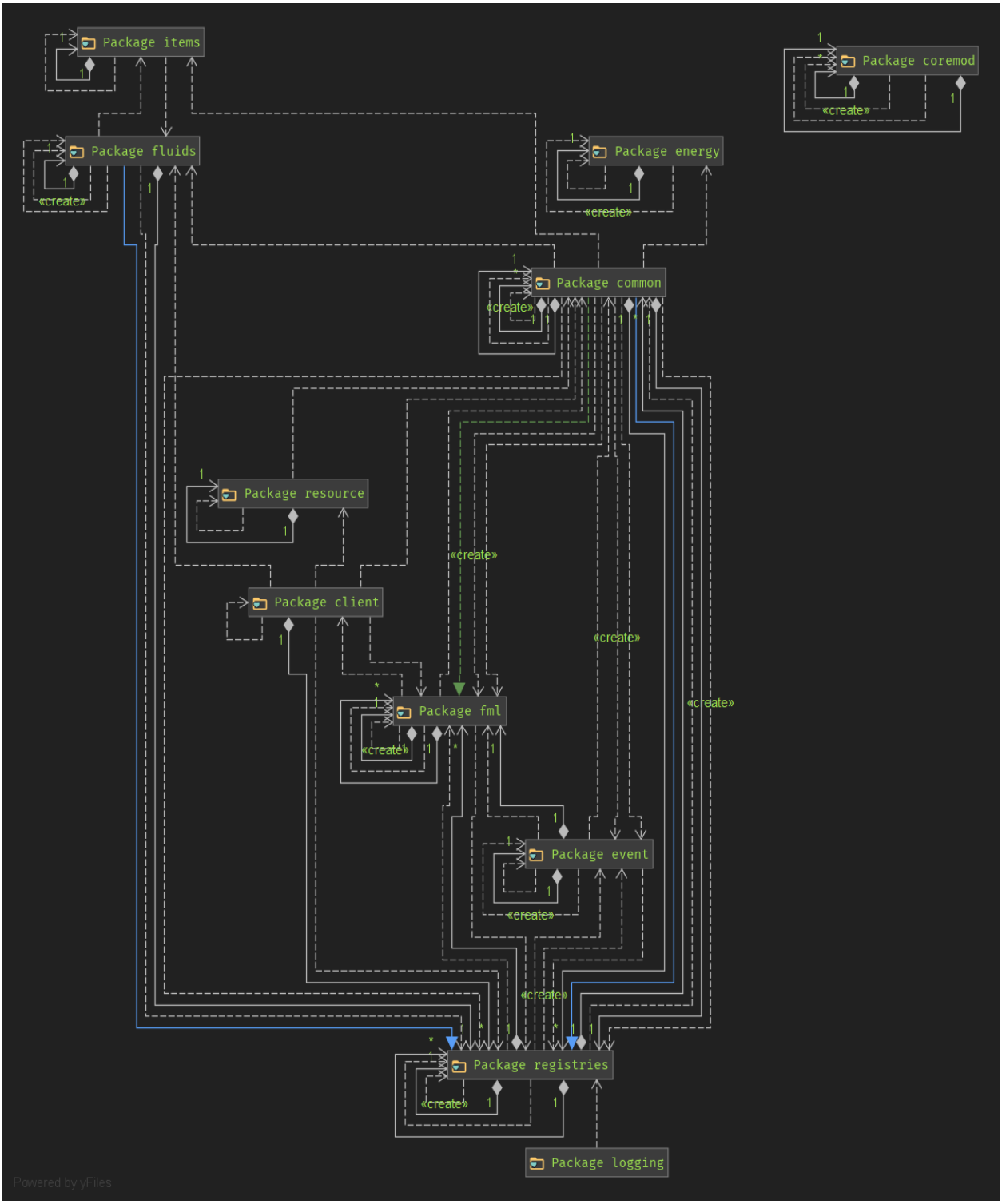

*moddiscovery*

In regards to patterns that were followed and recalling from deliverable 1, the observer/observable and visitor patterns have been implemented with the mod annotation

object. This allows for mods to be given the appropriate annotations during runtime and also keeping track of every mod being loaded.



Forge Main (wrappers, adpaters, and hooks) – src/main




*Root-level overview of the Forge Main component*

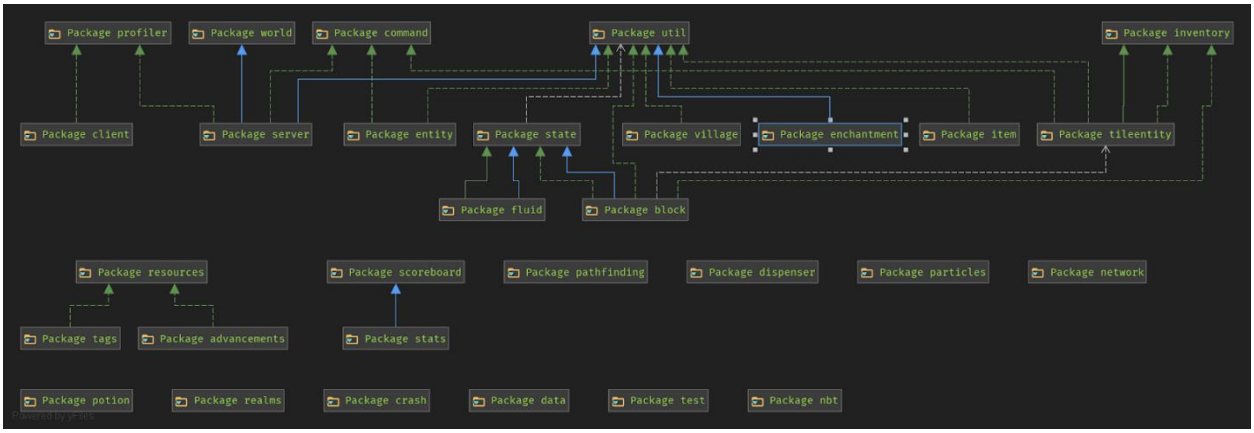
As per deliverable 1, Forge Main is essentially where all of Forge's main functionality resides. It contains all the code that allows for it to wrap around Minecraft and also inject the necessary changes to the game's code. The packages can be described as follows:

- **client** → contains all the event handlers and GUI rendering for the client
- **common** → contains interfaces and other in-game configurations for Minecraft
- **energy** → an API that enables modders to create their own energy implementation in the game
- **event** → contains events that modders often access from Minecraft (entity/block interactions, enchanting items, boss encounters, etc.)
- **fluids** → contains all the logic/configurations related to how custom fluids should work in the game (Minecraft itself doesn't implement fluids generically, it hardcodes most values)
- **fml** → contains all of the mod loader's sided event handlers (server vs. client), hook management, and wrapper management

The rest of the packages follow a similar structure in regards to containing wrapper code, hooks, registry managers, and event handlers.

Recalling from deliverable 1, we have said that the most notable aspect in Forge Main's architecture is the interdependency of the packages mentioned in addition to how each subcomponent is a wrapper for some component in the vanilla Minecraft. Through this, Forge is capable of exposing a decent chunk of the vanilla game's code, allowing for modders/developers to manipulate behaviour accordingly. As an example, one can see that every wrapper under *main.java.net.minecraftforge.client.items.wrapper* exposes bits of the vanilla game, allowing developers to change the behaviour of item handling in the game (storing items in your inventory, holding an item in your hand, wearing clothing/armor, etc.).

Forge Minecraft (modifications to the Minecraft codebase) – projects/forge




Forge Minecraft is the subproject of Forge which deals directly with modifications to the base code of Minecraft. Due to the amount of classes present, the above diagram only displays a diagram of how the packages are structured. From a lower level, many of the classes interact with one another in between packages.

For some of the classes within this subproject, there are little to no changes between Forge Minecraft and vanilla Minecraft. On the other hand, Forge Minecraft may redesign classes from the original implementation in order to add new features. An example of redesigning a base Minecraft class is Forge's reimplementation of rendering. The Forge team created a custom rendering engine on top of Minecraft's 1.10 engine, allowing more adaptability for modders to load custom models (such as OBJ files). One of the interesting aspects of the design in Forge is that the project chooses to keep the same file / package structure as the base Minecraft game (located in projects/clean). The only area that differs is the connection to wrappers that are found in Forge Main.