MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

CRAFT MASTERY - EXTENSION OF THE ORIGINAL MINECRAFT GAME

LICENSE THESIS

Graduate: **Casian-Cristi CRISTIAN**

Supervisor: **Prof.Dr.Ing. Lia-Anca HANGAN**

Consultant: **Ing. Tudor Florentin COROIAN**

2025

Contents

Chapter 1 Introduction - Project Context **1**
1.1 Problem statement . . . 1
1.2 Proposed solution . . . 1

Chapter 2 Project Objectives **3**
2.1 Objectives . . . 3
2.2 Requirements . . . 3
2.3 Stakeholder and User Descriptions . . . 3

Chapter 3 Bibliographic Research **6**
3.1 Obfuscation and Deobfuscation . . . 6
3.2 Learning through Minecraft . . . 6
3.3 Minecraft as a central focus for research . . . 7
3.4 APIs used for modding . . . 8
3.4.1 Forge API . . . 8
3.4.2 Fabric API . . . 9
3.5 Existing mods . . . 9
3.5.1 Mekanism . . . 9
3.5.2 GregTech . . . 10

Chapter 4 Analysis and Theoretical Foundation **11**
4.1 Minecraft base game . . . 11
4.1.1 Blocks . . . 11
4.1.2 Items . . . 12
4.2 Use cases . . . 12
4.2.1 Crafting and using of Lumber Axe . . . 12
4.2.2 Crafting and using of Mining Hammer . . . 13
4.2.3 Crafting and using of Mining Pickaxe . . . 14
4.2.4 Crafting and using of Magnet . . . 15
4.2.5 Obtain Tool Root . . . 16
4.2.6 Obtain Tool Plate . . . 18
4.3 Data Models . . . 19
4.4 Additions to the game . . . 20
4.4.1 Items . . . 20
4.4.2 Blocks . . . 21
4.5 Algorithms for searching blocks . . . 21
4.6 Algorithms for mining 3x3 blocks . . . 22

Chapter 5 Detailed Design and Implementation **24**
5.1 General project overview . . . 24
5.2 Registries . . . 25
5.2.1 Registering an item . . . 25
5.2.2 Registering a block . . . 27
5.3 Custom items . . . 30
5.3.1 Lumber Axe . . . 30

5.3.2 Mining Hammer . . . 32
5.3.3 Mining Pickaxe . . . 33
5.3.4 Magnet . . . 34
5.4 Custom block entities . . . 35
5.4.1 Rolling Mill . . . 36
5.4.2 Metal Press . . . 36
5.5 Custom user interfaces . . . 37
5.6 Custom recipe types . . . 38

Chapter 6 Testing and Validation **40**
6.1 Crafting and using the Lumber Axe . . . 40
6.2 Crafting and using the Mining Hammer . . . 41
6.3 Crafting and using the Mining Pickaxe . . . 44
6.4 Crafting and using the Magnet . . . 45
6.5 Crafting the Rolling Mill and the Tool Rod . . . 47
6.6 Crafting the Metal Press and Tool Plate . . . 49

Chapter 7 User's Manual **51**
7.1 Resources and installation procedure . . . 51
7.2 How to use Craft Mastery mod . . . 51

Chapter 8 Conclusions **55**
8.1 Contributions and achievements . . . 55
8.2 Results . . . 55
8.3 Further development . . . 56

Bibliography **57**

Appendix A Relevant Code sections **59**

Appendix B Figures **65**

Chapter 1. Introduction - Project Context

1.1. Problem statement

At least one time in the life, everyone heard about the Minecraft game. It is famous for its unique gameplay. The world generated is formed from different types of blocks, inclusively the air is a block. It tests the user's creativity, allowing them to construct, gather resources, and explore like in real life. Most of the processes that can be done in the game are activities that some people also do in reality.

Minecraft started to become a powerful tool for both personal and game development through mods. Mods are extensions added to the base game where new blocks, items, entities can be introduced, basically everything developers want in order to upgrade the game. Some things that are already in the game can be also changed, such as recipes for some crafting tools. These mods are created on the one hand for research, for testing some algorithms, for learning programming concepts, or even for improving personal skills. On the other hand, mods are created because some players want to have something new in the game, they get bored of the things that can be done in the Minecraft base game, and they want to experience new things.

Two processes the players consider that are time consuming and they want an alternative to not waste time are mining and cutting down trees. Those are the main processes that each Minecraft player must do in order to bring in resources. Traditional tools break down only one block at a time. They want to exist in the game an alternative for making these two activities much faster. When we talk about mining, the players desire to explore what is around them rapidly.

Players want to be added different types of tools, which to be crafted with other items. To have other types of tools not only the classical ones: gold, iron, diamond, wood. In this moment in Minecraft, the tools are crafted using sticks and ingots. They desire to use also other materials in the process of crafting tools.

In addition, the furnace is the only entity that is used to smelt ore blocks in order to obtain ingots. Players want to exist other entities for smelting that have a single or multiple recipes, from where to obtain more items from a single item. For example, each entity to smelt ingots, but each one to have different outputs.

1.2. Proposed solution

The solution for the given problem statement presented before is to create a Minecraft mod called *Craft Mastery* which will solve problems from the game. This solution consists in adding new items, new tools, new entities, and new recipes to the game, which on the one hand helps to solve the time consuming problem from the game, but on the other hand, for bringing something new to the Minecraft's players. There will be special tools which are going to be used to cut down tress and for mining efficiently,

these tools will be crafted with new items added to the game. The items will be obtained with the help of new entities that will have special recipes. Each entity will have its own interface and its own recipe.

The motivation behind this project is, first, the fact that I, in turn, have encountered these problems from the game and always I wanted to exist a mod that can procuring resources should not be so time consuming. Second, with the help of APIs, the modification of the game is very easy to realize, allowing to put your unique ideas in the game. Third, from the educational and skill improving point of view, is a good exercise for learning or deepen the programming languages.

The developing process requires the following resources:

- copy of the Fabric API
- IntelliJ
- Blockbench
- Laptop or PC with at least 4GB of RAM
- working copy of the Minecraft game - version 1.20.4.

Chapter 2. Project Objectives

2.1. Objectives

The main objective of *Craft Mastery* mod is to add special tools in Minecraft in order to solve problems from the game. These tools are going to be made from different items, not the classic craft, and will help players to cut down trees faster, and mine efficiently. The items will be made using special entities with custom recipes.

2.2. Requirements

Functional requirements present what the system should do in order to achieve its goals. The Craft Mastery mod will have the following functional requirements:

- **Add new items**. The items should be added in the new mode, and will be used by player for crafting new tools.
- **Introduce new tools**. This is related to the main objective of the project. The scope of the tools are to solve time consuming problem from the game.
- **Insert new entities**. These block entities will have the role of adding new crafting methods into the game.
- **Craft new recipes**. Recipes will be from two types: one custom for entities, and one for crafting.

Non-functional requirements consist on how the system should perform rather than what should do. The Craft Mastery mod will have the following non-functional requirements:

- **Response time**. The mod should have comparable response time with the Minecraft base game.
- **Similarity**. The player should not see big differences when playing the mod with Minecraft.
- **Compliance**. The mod should be created in a manner to be integrate with the existing game.

2.3. Stakeholder and User Descriptions

Stakeholders are persons or group of people who have an interest in your project. In the Table 2.3, are presented the stakeholders of *Craft Mastery* mod.

Table 2.1: Stackeholders

Name	Description	Responsibilities
Minecraft's game players	Persons who plays the mod	They are giving feedback about the mod
Modding Community	People who also develop Minecraft mods	They are making sure that your mod is compliance with other mods
Server Administrators	People who own Minecraft servers	They are testing the mod on multi-player servers and provide feedback

Besides stakeholders, who only have an interest in your project, users interact directly with the project. In Table 2.3, are described the users from *Craft Mastery* mod, together with their responsibilities, description, and from which stakeholder they belong.

Table 2.2: Users

Name	Description	Responsibilities	Stackeholder
Players	Minecraft's players	Testing the features added to the game and provide feedback	Minecraft's game players
Mod developers	People who develop mods	Testing the mod with other mods which already exist for compatibility, testing also with Minecraft base game. Provide feedback	Modding Community
Server Owners	Persons who possesses multiplayer Minecraft's servers	Testing the mod in multiplayer servers. Provide feedback	Server Administrators

Table 2.3 describes which are the challanges encountered by players when playing Minecraft and proposed a solution for their needs.

Table 2.3: User needs

Need	**Priority**	**Concerns**	**Current Solution**	**Proposed Solutions**
Faster tree cutting	0	Cutting down trees block by block is time consuming	Axe made from iron, diamond, gold, wood, stone, or netherite which can break only one block	Add Lumber Axe tool which cuts down the tree by breaking only one wood block
Efficient mining	0	Mining tunnels is very slow when there are a lot of stone blocks near the player	Players use pickaxes for making tunnels	Add Mining Hammer which mines in 3x3 areas by breaking only one block

Need	Priority	Concerns	Current Solution	Proposed Solutions
Faster mining ore blocks	0	Breaking each ore block individually is slow, sometimes there are connected blocks which are seen after breaking a nearby block	Pickaxe made from iron, diamond, gold, wood, stone, or netherite which can break only one block	Add Mining Pickaxe which mines all the connected ore blocks of the same type by mining only one block
New items for crafting	2	Players are using the same items for crafting	Stick and same ingots for crafting are used	Add Tool Rod and Tool Plate for crafting the new tools
New crafting methods for items	1	Players are using the same methods in order to obtain items	Furnace is the only entity used for smelting	Add Rolling Mill and Metal Press which smelt iron for obtaining Tool Rod and Tool Plate
Faster method for pulling nearby objects	1	Players have to go near each dropped thing in order to pick them up	There are no tools which pull nearby dropped objects	Add Magnet which, when used, pulls all the dropped objects from a certain distance

Based on the user needs that were presented before, the mod will have the following features in order to satisfy these needs.

Lumber Axe, will be the solution for the first need. It will be crafting using Tool Rod and Tool Plate, it will have a different durability, and it will cut down all the wood blocks, which create the tree, by only cutting down a single block. Durability should decrease with exactly the number of wood blocks which were cut down.

Mining Hammer, will be the solution for the second need. It will be crafting using Tool Rod and Tool Plate, it will have a different durability, and it will make 3x3 tunnels when mining. The middle block is mined, and all its neighbours will be mined also. Durability should decrease with nine.

Mining Pickaxe, will be the solution for the third need. It will be crafting using Tool Rod and Tool Plate, it will have a different durability, and it will mine all the ore blocks which are connected to the mined one. Durability should decrease with exactly the number of mined ore blocks.

Magnet, will be the solution for the last need. It will be crafting using Tool Rod, Tool Plate and Ender Pearl, it will have a 64 durability, and it pull all the nearby dropped items into its inventory. Durability should decrease with exactly the number of stack items that were pulled.

Tool Rod and Tool Plate, will be the solution for the fourth need. Those items will be obtained by smelting iron ingots in custom block entities.

Rolling Mill and Metal Press, will be the solution for the fifth need. These block entities will be crafted using a pattern well structured, and will have the role of handle custom recipes for obtaining the Tool Rod and Tool Plate.

Chapter 3. Bibliographic Research

3.1. Obfuscation and Deobfuscation

Obfuscation, in software development, is the process of transforming the code that is readable by humans into a computer code (also called machine code) which is quite difficult for humans to read and understand it. The main reason why this operation is done by developers is to prevent attackers from modifying their code. Also, obfuscation is used to create challenges where the developers must make the code human readable again. The operation can be done manually or using obfuscators. When we talk about the methods for obfuscating the code, we remaind next ones: giving meaningless names to the variables, classes, and methods, making a confusion between comment, code, and data in the program, by replacing some code with comments and syntax with data, and having duplicate code [1].

Gregory Wroblewski presents in his paper [2] that obfuscation "is present almost everywhere", basically if someone encrypts data, this process is nothing else than obfuscation. Also, he is presenting a simple obfuscator, machine code obfuscation applying two theorems:

- an obfuscating transformation can only include code that is equivalent to the original, reversible, or changes only unused parts of the program's context.
- if a program which is divided in two obfuscated programs by two different obfuscating transformations, a combined obfuscated program can be constructed.

Deobfuscation is the opposite operation of obfuscation, where the developers make the obfuscate code to be human readable and in an understandable form. The process can be done manually applying revere engineering in order to reconstruct the whole design, or with the help of deobfuscators, or solutions based on artificial intelligence which used machine learning [3].

Collberg, Thomborson and Low present in their paper [4] three deobfuscation techniques:

- Data-Flow Analysis, which consists on removing unnecessary obfuscated code.
- Thorem Proving, which is used when Data-Flow Analysis is insufficient, this approach is uses theorem provers for predicates, for example, proves that a loop always terminates.
- Partial Evaluation removes the unnecessary code and lets only the code needed at runtime by analyzing the static parts of a program.

3.2. Learning through Minecraft

Minecraft can be taken as a good platform for learning for everyone, be they children or students. Children can learn about science, technology, engineering, or mathematics; however, students can understand the basic concepts of programming.

Daisuke, Takebayaski, and Tsuneo present in their paper [5] how important Minecraft can be in the process of educating an entry-level programmers to become productive members of a software development team. Their are highlighting that the basic trainings at the job, in most of the cases, do not really help the beginners to understand the project structure, communication skills, and how to write a document. They are demonstrating that Minecraft can help beginners to bring important skills to a programmer, those being how to communicate with the team members, how to understand a project flow and how to write the document. The results of the study show that the learners understood more than 50% from the project at the beginning, they understood how important communication and shared information is.

Dr. Maram Meccawy from King Abdulaziz University in Jeddah, Saudi Arabia, researched in her paper [6] how children between 5 and 15 years old can learn the basic concepts of computer programming by just playing the Creative Programmer Minecraft mod. She observed that this kind of learning is more enjoyable among children. In order to test that indeed there are processes in kids education, they answered to a pre and post questionnaires. The results show that the kids with zero experience answer almost all the questions correctly, when the kids with some knowledge about programming improved their skills. The skills improved or learned by students are: writing commands from programming, understanding the concepts of variables, repetitions, and algorithms.

The author of [7], the author, presents how he puts in practice the main concept from Minecraft in teaching, namely, the students "customize the class concepts based on their needs", in this way teachers want to improve the students creativity, and interest in programming through Minecraft-inspired activities. The game is used directly in both the theoretical and practical part from the training process.

According to [8], the focus has moved from teachers instruction to how Minecraft can handle both formal and informal learning for students. The study shows that the system based on rewards after an achievement and the rules constructed as in the game, increases the attention and the implication of the students in the process of learning. "Steinbeiß-Ruotsalainen Model" offers a base for constructing a learning scheme in Minecraft. It consists of formal learning, informal learning, and non-formal learning.

3.3. Minecraft as a central focus for research

Since Minecraft is an open and interactive game, in time it becomes a focus point in educational research. Minecraft is not only a platform towards educational research, but it's also the subject of general research in other areas such as: software engineering, artificial intelligence, or human-computer interaction. In the next papers, I will show where Minecraft is used as a means to an end for learning areas such as programming, solving creative problems, and teamwork.

Guivari Dzar Amri in his paper [9], investigates the performance of mods in Minecraft, since there are few researches into this area. In this way, the game is used not only as a learning tool or a creation environment, but also for benchmarking, analysis of software performance, or optimization. The World Generation Minecraft mods used are: Biomes O'Plenty, Naturalist, and The Twilight Forest. The Vanilla game was also tested without any mode, and all the mods were used together. Performance metrics used are: Tick Duration and Lag Spikes. The result shows that running mods individually has greater performance impact than running multiple mods in parallel.

In this paper [10], the author raises the problem of introducing Artificial Intelligence into games. Some algorithms which work in theory ended up having bad outputs for speed or memory. Those algorithms were given to students where they have to implement them in Minecraft, but to take care of consequences which may appear in the game. In this way, we can see how Minecraft is used as a testing concept, being a strong exercise for students to make difference between theoretical concepts and real-world development. The following AI algorithms were tested in Minecraft: Heuristic search, Logical foundations of AI, Fuzzy logic, Neural networks, Planning, Bayesian reasoning, Reinforcement learning, and Agent architectures.

In another paper written by teachers from Flinders University of South Australia, is described how students develop mods in Minecraft in order to improve their skills or learning new programming languages. In general, young generations learning the syntax of a programming language is a tough process, so introducing concepts in Minecraft can create a favorable environment for learning. Introducing the game in this process can motivate students to be more productive and motivate them to apply what they learned [11].

In [12], it is presented how Minecraft plays an important role in simulating scenarios similar to multi-user computer-aided design (CAD) environments. The management of complex design can be done in Minecraft and it can be tested how members of the virtual team communicate between them, coordinate, and manage the project.

3.4. APIs used for modding

In order to change the gamestyle of Minecraft, and to add something new, to change how the game works, those APIs come to help the developers in order to extend the game's functionality. In the world of Minecraft modding, there are multiple APIs, but two of them are the most used in this industry: Forge and Fabric. Even they are similar, in the sense that both intercept the registry initialization before the registers are frozen, they handle code injection differently and approach mod development from a separate perspective, in terms of resulting binary size: Forge exposes all of the features of Minecraft in a monolithic way, while Fabric has multiple modules that can be used, based on the theme of the developed mod. They act as a bridge between the standard game and the additions that are provided to the game.

3.4.1. Forge API

Minecraft Forge and Forge Mod Loader were released in November 2012 and are a modding framework used for creating compatible mods for Minecraft. It suits perfect when developer wants to integrate multiple mods into a single one because it "allows large modpacks to work coherently". That means even if there are duplicated items and blocks, those are not causing any issues. It can also perform basic operations like adding an item, a block, an interface. Since it's a big project, their update takes a long time to be released. In practice, those updates come in a decent time after the new release of Minecraft, but they come with bugs [13].

The book "Minecraft Modding with Forge" [14], presents how entertaining it is to modify the game and add features or to change some logic from entities. In the beginning, is to show what exists and what are the actions which can be done in Minecraft, further on it starts to explain how to make your own mod using Fabric, being a guide for the

developers. It learns how to add new blocks, new items, new entities, how to share the created mod, or how to add new commands to the game.

3.4.2. Fabric API

Fabric API was released in 2018, and its purpose was to handle small projects. It is a better option for developers who want to approach an easier way to produce a mod. Since its architecture is minimal, it provides faster updates on new Minecraft releases, also an advantage being its "independence from Minecraft versions". The game loading time decreases because Fabric contains fewer features[15, 16].

Table 3.1 briefly presents which are the main differences between Forge and Fabric based on principle features.

Table 3.1: Comparison between Forge and Fabric modding platforms.

Feature	**Forge**	**Fabric**
Loading time	longer	faster
Performance	more resources,	less resources
Update Speed	slower	quick
Compatibility	compatible with many mods	limited
Usage	complex	easy

3.5. Existing mods

In the world of Minecraft, there exist mods which have the special tools for mining, automated crafting which simulate as machines, and custom recipes. In this section, I will talk about some of them.

3.5.1. Mekanism

Mekanism is a mod in Minecraft, which its purpose is to create different types of machines. It does not have a special goal, but the difficulty of the mod increases step by step. At the beginning, the player creates easy machines in order to get familiar with their creation, gradually the complexity of machines being increased. Player can update a machine to the next level by updating it with some materials, also can update using an installer but this one cannot update, for example a level one machine to a level three, it will directly install only that version. Each machine has its own GUI and its own custom recipe, making it a complex mod.

Also, Mekanism introduces new armors and tools with special functionalities. For example, Jetpack which allows the players to fly, Atomic Dissasembler which acts as an Axe, Pickaxe, Shovel having different levels of speed, this speed being controllable by the player.

In Mekanism we can find a robot which acts as a pet, it follows the player and collets the dropped items for him. In addition to this functionality, the robot also has the role of a chest, furnace, anvil, and workspace. The mod also adds new materials [17].

3.5.2. GregTech

GregTech is a mod in Minecraft which comes with a lot of upgrades to the original game, such as adding new machines, new materials, and changing the classic recipes for the tools. Machines are also in tier levels, each one unlocks a more efficient process, like obtaining more resources from an ore block. Also, each process works only if it is powered with energy, there can be the risk to explode the machine if the voltage is higher than they can handle. Each machine comes with a custom graphical interface with different numbers of slots, and voltage indicators.

In the world generation, this mod comes with new ores like: Lithium, Nickel, Sapphire, and so on, as well as Asteroids, which can have different types of ores, and at each 25 spawns, the player can have a considerable amount of ores.

The mod is constructed in an industrial manner having the process names, materials names, real-world meanings for example, instead of smelt ore blocks directly into ingots, in GregTech are a lot of intermediate steps from ore block to ingot: crushing, washing, centrifuging, and chemical processing. Another example can be the one mentioned before with an higher voltage on machines which causes explosions [18].

Table 3.2 shows which new things are added to the Mekanism and GregTech modes.

Table 3.2: Comparison of features between Mekanism and GregTech

Feature	**Mekanism**	**GregTech**
Tool rods from iron	NO	YES
Tool plates from iron	NO	YES
Metal Press	YES	YES
Rolling Mill	NO	NO
Lumber Axe	NO	NO
Mining Hammer	YES	NO
Mining Pickaxe	NO	NO
Magnet	YES	NO

Chapter 4. Analysis and Theoretical Foundation

4.1. Minecraft base game

In this section, I will present fundamental concepts about the two most widely used elements by the players in Minecraft: items and blocks.

4.1.1. Blocks

Block is the most used concept in Minecraft because with the help of those we made up the entire game world. It has a direction and position in a 3-dimensional grid (x,y,z coordinates)[19]. Based on its properties, there are several types of blocks:

1. Solid blocks: They occupy an entire cube space and the player cannot walk through them. Example: **stone**, **wood blocks**.
2. Non-solid blocks: On top of them, the player cannot place any other block. Example: **flowers**.
3. Non-opaque blocks: The light can pass through it. Example: **glass**, **leaves**.
4. Ore blocks: From those the player obtains resources. Example: **coal ore**, **iron ore**, **diamond ore**.
5. Fluids: They have the property of flowing over surfaces. Example: **water**, **lava**.
6. Block entities: They have functionalities. Example: **furnace**, it smelts blocks and items.
7. Blocks which are affected by gravity: if there is no other block under them, they will fall. Examples: **sand**, **gravel**.
8. Decorative blocks: most of them, they are just placed in the world.
9. Blocks with which player interacts passive: Those blocks affects the stats of the player. Examples: **soul sand**, **ice**, **magma block**.
10. Blocks with which player interacts active: Those blocks have also an interface. Examples: **chest**, **crafting table**, **furnace**
11. **Air block**: Is an invisible block, presented everywhere other block can exist.

In Craft Mastery mod, there will be two blocks which will be added:

- Rolling Mill, used to melt iron ingot for obtaining Tool Rod.
- Metal Press, used to melt iron ingot for obtaining Tool Plate.

Block entity, according to the Minecraft Wiki "is an additional object associated with certain blocks"[20]. In other words, it is a type of block which can store information, allowing us to make blocks with functionalities. Examples of block entities: **chest** used to store items, **furnace** used to smelt ore blocks or to cook food.

Because the Rolling Mill and the Metal Press need to interact active with the player, a block entity will be added for each block added to the game. Active interaction refers to an external input in order to obtain something, in our case for obtaining a Tool Rod or a Tool Plate we need to put iron ingot in the input slot of the block entities. On

the other hand, passive interaction has effects but without direct interplay of the player.

4.1.2. Items

Item is an object which can't be places in the world, like a block, it only exists in the player's inventory, hands or can be dropped[21]. Based on its properties, there are several types of items:

1. Tools: Used in breaking blocks faster. Example: **pickaxe**, **axe**, **shovel**.
2. Consumables: After using, these can affect the player's state in a positive or negative way. Example: **bread**, **apple**, **potion of poison**.
3. Armor: Offer protection when the player wears them. Example: **chestplate**, **boots**, **helmet**.
4. Weapons: Cause damage to other players when it is used. Example: **bow**, **trident**.
5. Crafting materials: With the help of them, the player can create other items or blocks. Example: **stick**, **iron ingot**.
6. Item block: These are placeable blocks in the world. Example: **cobblestone**, **oak wood**.
7. Entity block: When used place an entity in the world. Example: **boat**, **minecart**.

In Craft Mastery mod, there are several items which are introduced, some of them are tools, others are crafting materials:

- Lumber Axe, tool used for cutting down the whole tree by breaking a single block.
- Mining Hammer, tool used for mining 3x3.
- Mining Pickaxe, tool used for mining resources efficiently.
- Magnet, tool used for collecting nearby dropped items and blocks.
- Tool Rod, crafting material used for making the new tools.
- Tool Plate, crafting material used for making the new tools.

4.2. Use cases

Use case is a diagram that describes the interaction between the user and system in order to obtain a goal. In the following sub-chapters, I will present the use cases for Craft Mastery mod. In the next diagrams, the color yellow represents the happy flow, which means that the use case works without problems. The other colors which are there: blue, gray, pink, and green, are the alternative flows which is a deviation from the main flow, but the goals is going to be achieved but in an alternative way.

4.2.1. Crafting and using of Lumber Axe

This use case describes the process of crafting and using a specialized tool for cutting down trees efficiently. The primary actor in the figure is the **Player (P)**, who interacts with the game environment. The following precondition must be met to start the use case:

- The player must have access to a crafting table.

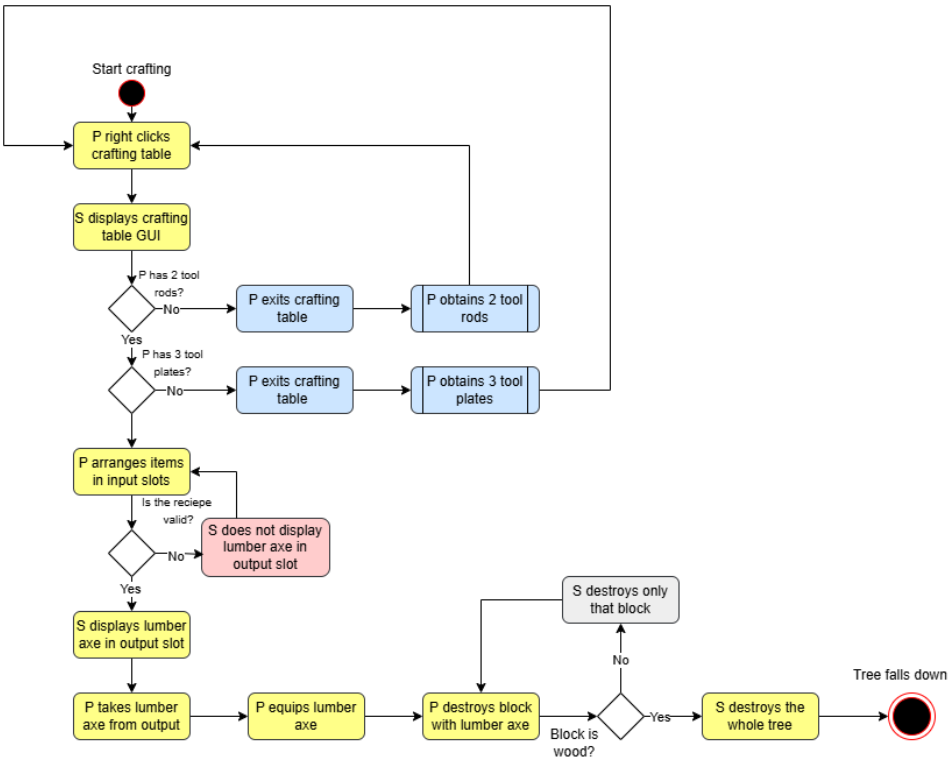

Figure 4.1: Use case for crafting and using Lumber Axe

The main (happy) flow is colored in yellow and starts when a player decides to craft and use a Lumber Axe. Steps from the basic flow:

- The player right-clicks the crafting table and the system (S) displays the crafting table GUI.
- P arranges items in the input slot (2 tool rods, 3 tool plates), S validates the recipe and displays the tool in the output slot.
- P takes the Lumber Axe from the output slot, equips it, destroys a block with it, and S causes the entire tree to fall down.

As shown in the figure, there are 4 alternative flows, colored in blue, pink, and gray:

- The first occurs when the player does not have 2 tool rods. The player exits the crafting table and obtains the missing items(see Figure 4.5).
- The second occurs when the player has 2 tool rods but not 3 tool plates. The player exits the crafting table and obtains 3 tool plates(see Figure 4.6).
- The third occurs when the player has the necessary items but the recipe is incorrect. In this case, S does not display the Lumber Axe in the output slot.
- The fourth occurs when the player has necessary items, provides a valid recipe and destroys a block that is not wood. The system destroys only that single block.

4.2.2. Crafting and using of Mining Hammer

This use case describes the process of crafting and using a specialized tool for mining efficiently. The primary actor in the figure is the **Player (P)**, who interacts with the game environment. The following precondition must be met to start the use case:

- The player must have access to a crafting table.

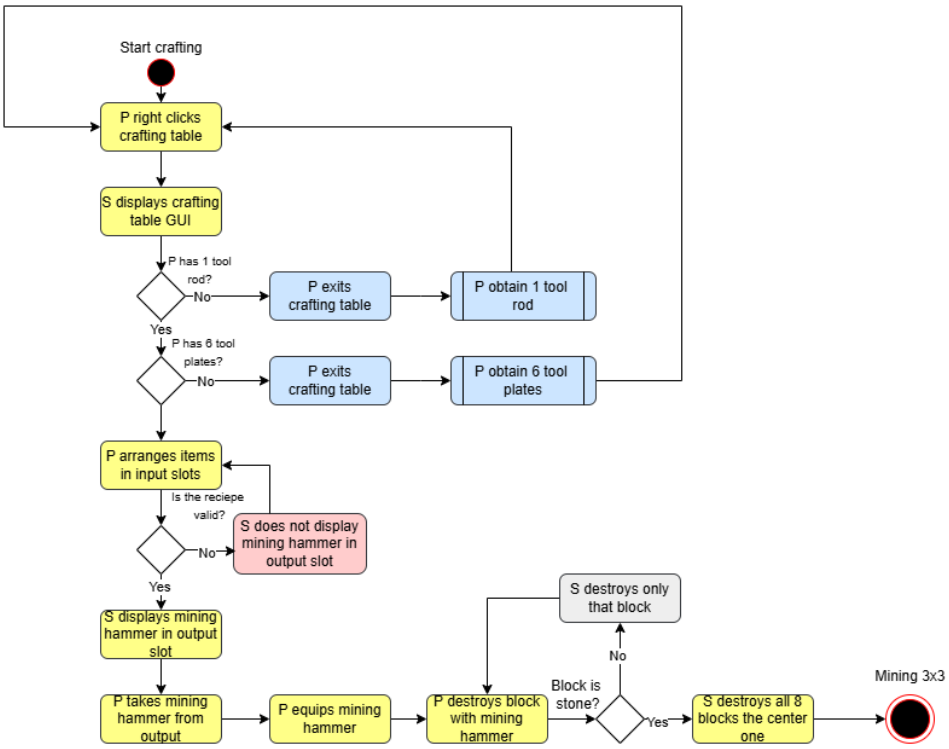

Figure 4.2: Use case for crafting and using Mining Hammer

The main (happy) flow is colored in yellow and starts when a player decides to craft and use a Mining Hammer. Steps from the basic flow:

- The player right-clicks the crafting table and the system (S) displays the crafting table GUI.
- P arranges items in the input slot (1 tool rod, 6 tool plates), S validates the recipe and displays the tool in the output slot.
- P takes the Mining Hammer from the output slot, equips it, destroys a block with it, and S destroys all 8 blocks around the center one.

As shown in the figure, there are 4 alternative flows, colored in blue, pink, and gray:

- The first occurs when the player does not have 1 tool rod. The player exits the crafting table and obtains 1 tool rod(see Figure 4.5).
- The second appears when the player has 1 tool rod but not 6 tool plates. The player exits and obtains the missing plates(see Figure 4.6).
- The third occurs when the player has the correct items but the wrong recipe. S does not display the Mining Hammer.
- The fourth appears when the player provides a valid recipe and destroys a non-stone block. Only that block is destroyed.

4.2.3. Crafting and using of Mining Pickaxe

This use case describes the process of crafting and using a specialized tool for vein mining. The primary actor in the figure is the **Player (P)**, who interacts with the game environment. The following precondition must be met to start the use case:

- The player must have access to a crafting table.

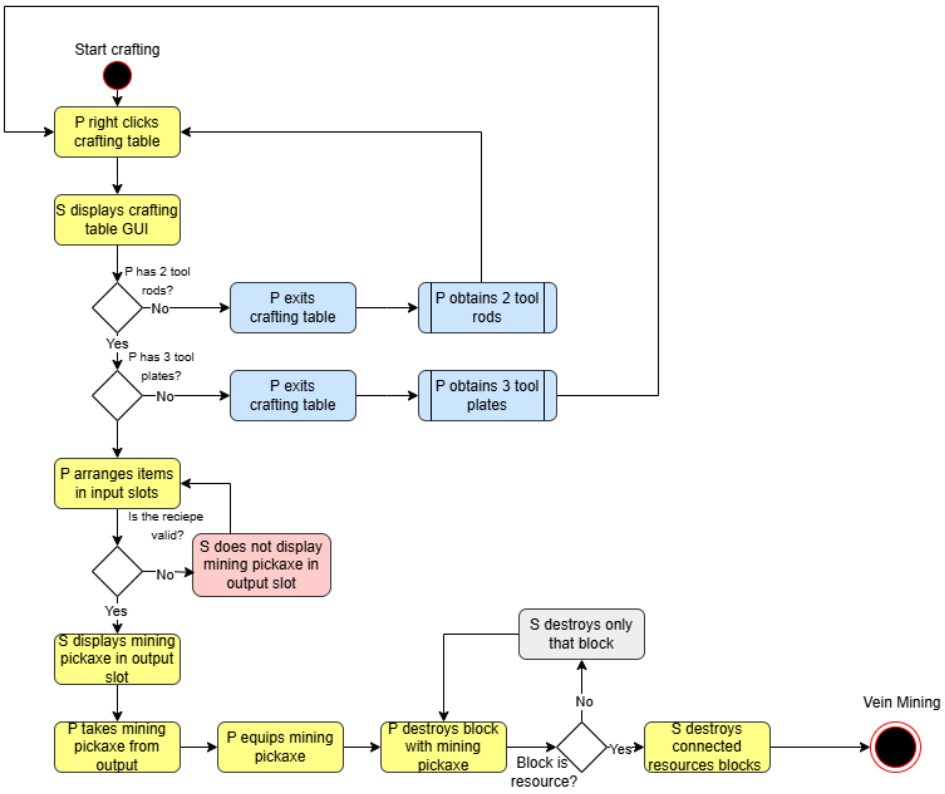

Figure 4.3: Use case for crafting and using Mining Pickaxe

The main (happy) flow is colored in yellow and starts when a player decides to craft and use a Mining Pickaxe. Steps from the basic flow:

- The player right-clicks the crafting table and the system (S) displays the crafting table GUI.
- P arranges items in the input slot (2 tool rods, 3 tool plates), S validates the recipe and displays the tool in the output slot.
- P takes the Mining Hammer from the output slot, equips it, destroys a block with it, and S destroys all resources which are connected.

As shown in the figure, there are 4 alternative flows, colored in blue, pink, and gray:

- The first occurs when the player does not have 2 tool rods. The player exits the crafting table and obtains 2 tool rods(see Figure 4.5).
- The second appears when the player has 2 tool rods but not 3 tool plates. The player exits and obtains the missing plates(see Figure 4.6).
- The third occurs when the player has the correct items but the wrong recipe. S does not display the Mining Pickaxe.
- The fourth appears when the player has necessary items, provides a valid recipe and destroys a block which is not a resource. Only that block is destroyed.

4.2.4. Crafting and using of Magnet

This use case describes the process of crafting and using a specialized tool for collecting all nearby dropped items efficiently. The primary actor in the figure is the **Player (P)**, who interacts with the game environment. The following precondition must be met to start the use case:

- The player must have access to a crafting table.

- The player must have an ender pearl in inventory

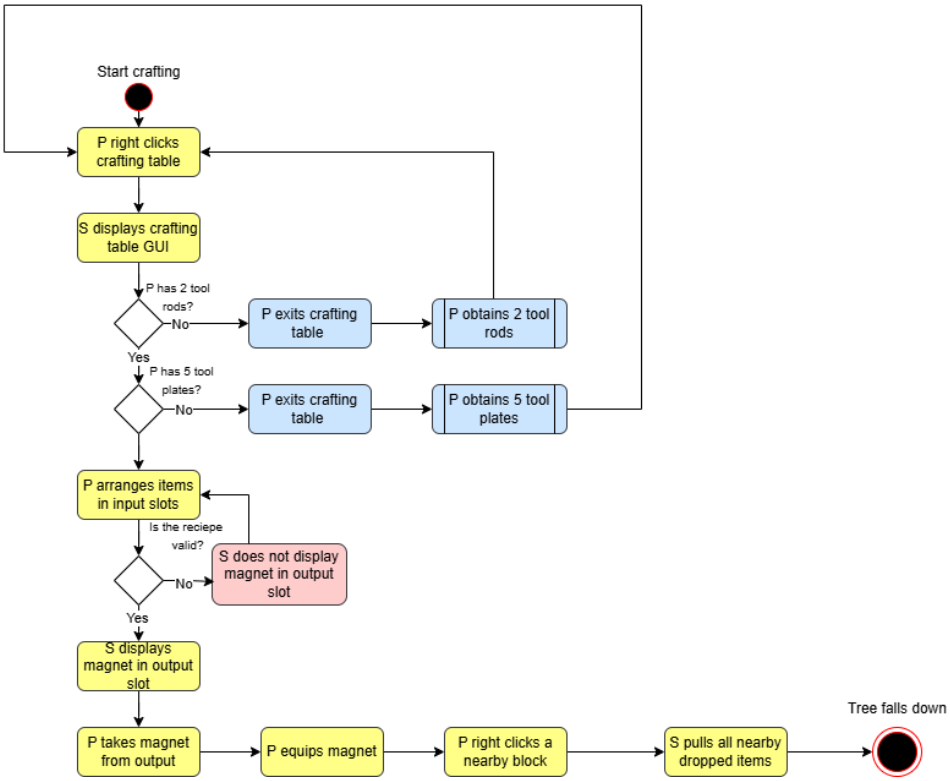

Figure 4.4: Use case for crafting and using magnet

The main (happy) flow is colored in yellow and starts when a player decides to craft and use a magnet. Steps from the basic flow:

- The player right-clicks the crafting table and the system (S) displays the crafting table GUI.
- P arranges items in the input slot (2 tool rods, 5 tool plates and one ender pearl), S validates the recipe and displays magnet in the output slot.
- P takes the magnet from the output slot, equips it, right click a nearby block and S pulls all nearby dropped items.

As shown in the figure, there are 2 alternative flows, colored in blue, and pink:

- The first occurs when the player does not have 2 tool rods. The player exits the crafting table and obtains 2 tool rods(see Figure 4.5).
- The second appears when the player has 2 tool rods but not 5 tool plates. The player exits and obtains 5 tool plates(see Figure 4.6).

4.2.5. Obtain Tool Root

This use case describes the process of crafting tool rod. The primary actor in the figure is the **Player (P)**, who interacts with the game environment. The following precondition must be met to start the use case:

- The player must have access to a crafting table.

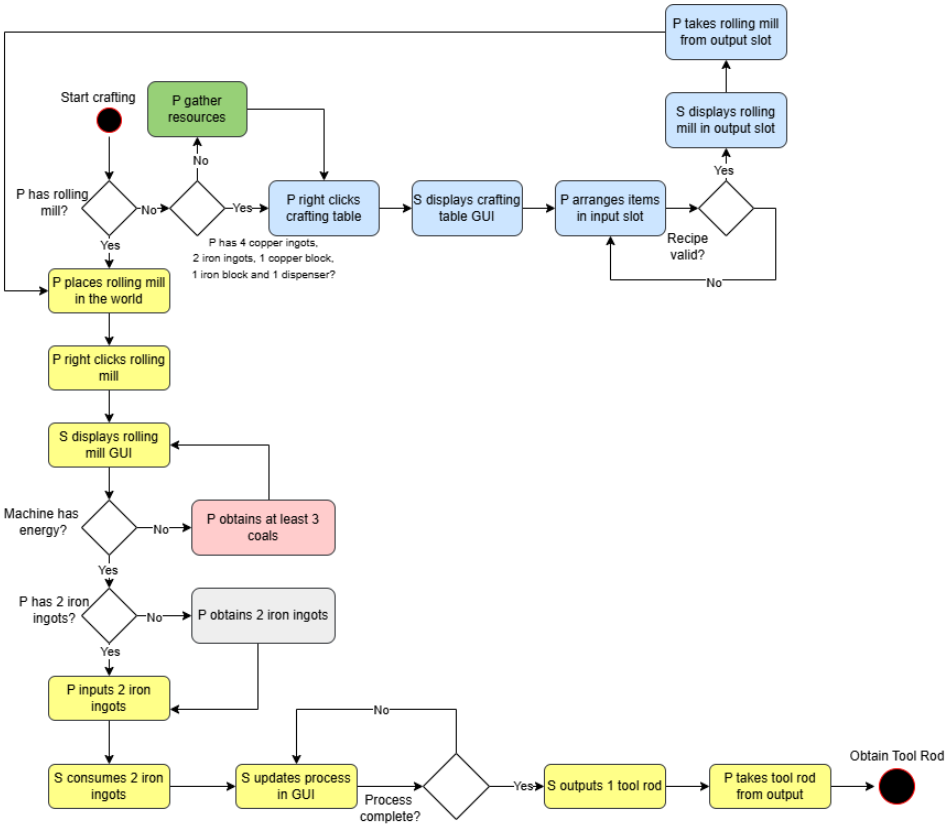

Figure 4.5: Use case for obtaining Tool Rod

The main (happy) flow is colored in yellow and starts when a player decides to craft tool plate. Steps from the basic flow:

- The player places the rolling mill into the world, clicks it and the system displayes the GUI.
- P inputs 2 iron ingots, S consumes it and updates the process in GUI
- S displays the tool rod in output slot and the P takes it from there

As shown in the figure, there are 6 alternative flows, colored in green, blue, gray and pink:

- The first occurs when the player does not have rolling mill and the necessary resources for crafting it. The player gather resources.
- The second appears when the player does not have rolling mill, it has the necessary resources but the recipe is not valid. In this case, the player rearranges items in the input slot.
- The third appears when the player does not have rolling mill, it has necessary resources, and also the recipe is valid. In this case, the rolling mill can be crafted.
- The fourth occurs when the player has rolling mill but the machinary does not have energy. The player has to obtain at least 3 coals.
- The fifth occurs when the player has rolling mill, has coal but does not have 2 iron ingots. The player has to obtain 2 iron ingots.
- The sixth occurs when P has rolling mill, has coal, has 2 iron ingots but the process of producing tool rod is not completed. In this case, the player has to wait until the system finished it.

4.2.6. Obtain Tool Plate

This use case describes the process of crafting tool plate. The primary actor in the figure is the **Player (P)**, who interacts with the game environment. The following precondition must be met to start the use case:

- The player must have access to a crafting table.

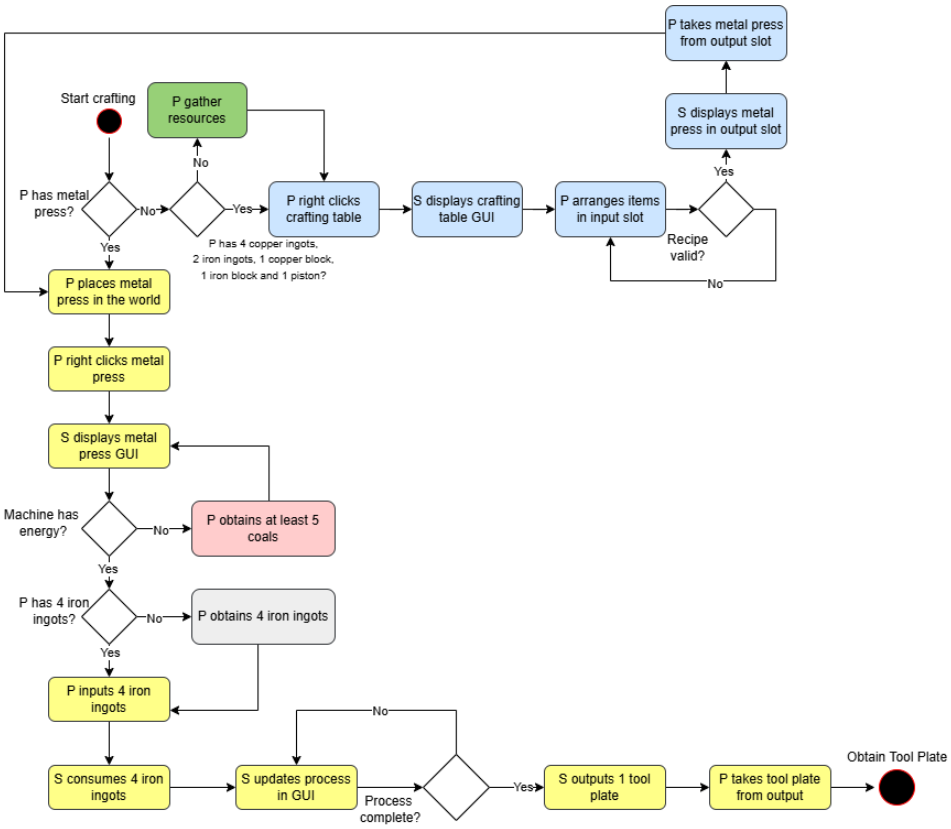

Figure 4.6: Use case for obtaining Tool Plate

The main (happy) flow is colored in yellow and starts when a player decides to craft tool plate. Steps from the basic flow:

- The player places the metal press into the world, clicks it and the system displayes the GUI.
- P inputs 4 iron ingots, S consumes it and updates the process in GUI
- S displays the tool plate in output slot and the P takes it from there

As shown in the figure, there are 6 alternative flows, colored in green, blue, gray and pink:

- The first occurs when the player does not have metal press and the necessary resources for crafting it. The player gather resources.
- The second appears when the player does not have metal press, it has the necessary resources but the recipe is not valid. In this case, the player rearranges items in the input slot.
- The third appears when the player does not have metal press, it has necessary resources, and also the recipe is valid. In this case, the metal press can be crafted.
- The fourth occurs when the player has metal press but the machinary does not have energy. The player has to obtain at least 5 coals.

- The fifth occurs when the player has metal press, has coal but does not have 4 iron ingots. The player has to obtain 4 iron ingots.
- The sixth occurs when P has metal press, has coal, has 4 iron ingots but the process of producing tool plate is not completed. In this case, the player has to wait until the system finished it.

4.3. Data Models

In this chapter, I will describe how the relations between data models are done. In Minecraft moding, here the developer defines, register and connect to the game the custom items, blocks, recipes, textures, etc.

In the *src/main/resources* directory are stored all materials that the developer needs to implement the mod. In this directory, assets are organized under *assets/modId* path, also the data such as recipes and tags are organized under *data/modId* directory. Let's take them one by one.

Assets refers to resources used by mod, which are not codable such as: textures, models, sounds, and block states. In the *assets/modId* path is defined the language file under *lang* folder, which is used to make readable text for players, for example, naming the new items or blocks added to the game, or to add tooltips which act as a description of the item or block. Some other assets type: texture, where are placed 2 dimentional models which will be mapped on 3 dimentional models in the game. Data Model (model folder) represents a connection between the model and in game representation, with other words, here are determined how the item/block will render in the game in terms of shape and texture. Block state describe all the possible states of a single block, such as: powered, growth stage. Figure 4.7 shows the structure needed for item/block to be respected for having the right representation in the game.

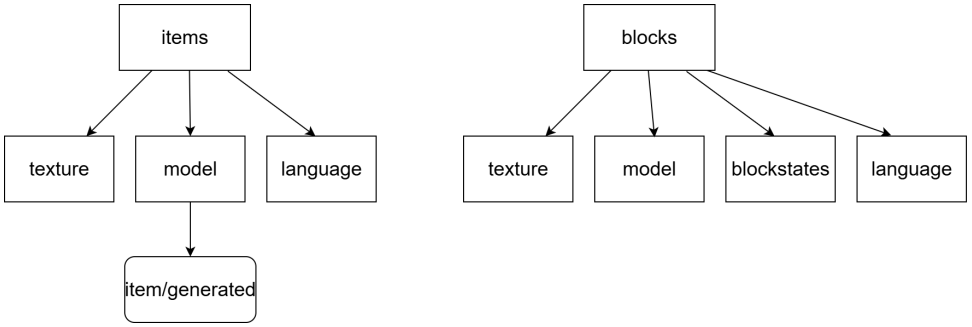

Figure 4.7: Items and blocks structured under assets

Data folder is the place where are stored some logic regarding recipes, tags. In the recipe folder from *data/modId/recipe* is defined the logic of crafting new items or blocks. Is specified from which material is obtained together with the pattern. Tags are used for define a group of items or blocks. Those are used when to address to multiple things from the game and instead of calling one by one, just call the tag which contains all of them.

When it comes to the recipe for the Rolling Mill and Metal Press block entities, the structure of making a recipe was respected, but there was added a field called ***"count"*** which tells how many input to be consumed in order to obtain the tool rod in case of Rolling Mill and the tool plate in case of Metal Press. Figure 4.8 indicates this process.

Fields that are also included are: type(which block entity is responsible for this recipe), ingredients(which ingredients can be used) and output(which is the output).

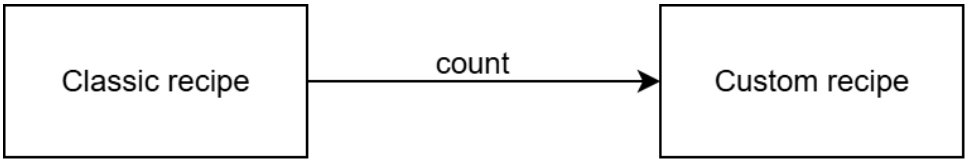

Figure 4.8: Field count added to model

Next, I will present how items and blocks are modeled in Minecraft not just from the point of view of visualization, but also from it's behavior. When we talk about the items, we have to think about what they should do when the player uses them. After cutting down a wood block from a tree with Lumber Axe, all the wood blocks from that tree must be cut down. After mining with Mining Hammer, all the blocks around that block should be mined (3x3 mining). After mining an ore block with Mining Pickaxe, all the connected ore blocks of the same time must be mined. When using magnet, all the nearby dropped blocks and items should pulled in the inventory. Having this information we can decide which click (left or right) to be used, for Lumber Axe, Mining Hammer and Mining Pickaxe will be used left click, and for Magnet will be used right click. For having this actions when the player presses the click, I have to override the function which executes this event.

Rolling Mill and Metal Press block entities are constructed in order to smelt iron ingot to obtain Tool Rod and Tool Plates. In order to make this action, I have to override also the function which is executed when the input from the machines is valid.

4.4. Additions to the game

For a better gameplay experience of Minecraft and based on the use cases presented in Section 4.2, we can start describing which are the items and blocks introduced in the game.

4.4.1. Items

Table 4.1: Items

No.	Name	Item Description
1	Tool Rod	Used for crafting the tools
2	Tool Plate	Used for crafting the tools
3	Lumber Axe	Used for cutting down the whole tree by breaking a single block
4	Mining Hammer	Used for mining 3x3
5	Mining Pickaxe	Used for mining resources efficiently
6	Magnet	Used for collecting nearby dropped items and blocks

The items that are added to the game are of 2 types: simple items, without any functionality, which are used for crafting other items. The second type of items introduced are those tools offering powerful abilities in the process of cutting down trees, mining,

and collect nearby dropped object.The next table presents a summary of the items in Craft Mastery mod.

4.4.2. Blocks

The blocks added to the game are functional ones, used in crafting some of the items presented in Table 4.1. These blocks have the same dimensions as a normal Minecraft block, which is 16x16x16 pixels. Since these blocks have specific functionalities, each will have its own block entity to support the implementation of the new crafting methods. In the table below, the name of this block is shown together with his functionality.

Table 4.2: Blocks

No.	Name	Block Description
1	Rolling Mill	Used for crafting Tool Rod
2	Metal Press	Used for crafting Tool Plate

4.5. Algorithms for searching blocks

Search algorithms are used to navigate, find, or process connected elements. Those are used for graphs and trees and are categorized by the type of the graph. Here are some examples of algorithms:

- Undirected graphs: Breadth-First Search(BFS) Algorithm, Depth-First Search (DFS), Prim's Algorithm finds a minimum spanning tree. Kruskal's Algorithm find minimum spanning tree using edge sorting.
- Directed graphs: Dijkstra's Algorithm finds the shortest path from a source node to all nodes from the graph. It works on non-negative weights. A* (A-Star) Algorithm finds the shortest path by combining the actual cost with an estimated one (heuristic).
- Weighted graphs: Bellman-Ford Algorithm finds the shortest path with negative weights in a graph. Floyd-Warshall Algorithm finds the shortest path between all combinational nodes in a graph.
- Bipartite graphs: Hopcroft-Karp Algorithm finds the maximum matching, Uniform Cost Search Algorithm (UCS) finds optimal paths between all nodes in a graph with non-negative edge weights, by expending the least-cost node first.

When we talk about Minecraft and its blocks, we can consider them as nodes from graphs. In the implementation of Lumber Axe and Mining Pickaxe, we have to apply a search algorithm in order to find all the blocks connected to the one that is cut or mined. Here from the algorithms mentioned before, the ones which fits the best are BFS and DFS, because conceptually the blocks are connected in all direction: up, down, north, south, east, west, and diagonals. A second reason why is considered as undirected graph is that there is no direction restriction from moving one block to another.

Breadth-First Search (BFS) uses queue as data structure where to store all the nodes. The approach of this algorithm is to visit all neighbors of a node before moving to the next one (children). This algorithm is used when the developer wants to find the shortest path to a destination, or the developer wants to have the minimum steps possible to a target if it exists.

Depth-First Search (DFS) uses stack as data structure, and its approach is to go as deeply as possible on one branch until it can not goes anywhere, after that it backtracks. Also, DFS can be implemented using recursion. From the point of view of memory, DFS requires less memory because it needs to store only the current node, on the other hand BFS must store all the neighbors of a node.

Table 4.3: Differences between BFS and DFS

Aspect	BFS	DFS
Memory	Can be high	Less memory
Data structure	Queue	Stack
Traversal Order	Explore node level by level	Explore as deeply as possible

Based on the comparison between these two, the best algorithm for implementing the tools is DFS. The reasons are:

- Trees and connected resource blocks are tended to be in a row, without many ramifications. For this one, the Last-In-First-One (LIFO) approach, requires an implementation using stack.
- DFS is very efficient in this case, when we want to visit all connected nodes.
- DFS uses less memory since there needs to store only the current path there.

Figure 4.9 illustrates the pseudocode for searching blocks in Minecraft using a DFS approach.

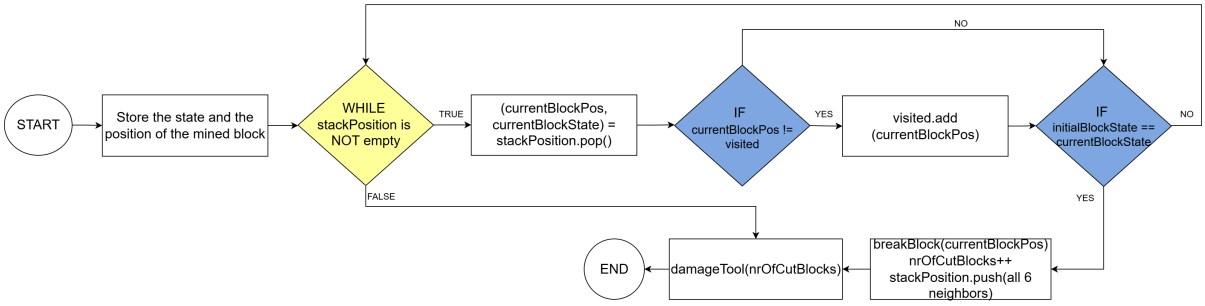

Figure 4.9: Algorithm for searching blocks pseudocode

4.6. Algorithms for mining 3x3 blocks

This algorithm is used for efficient mining. It works by finding the target position of the player. Based on its placement, the coordinates of the mined block are stored and with the help of them, we can find the neighbors of the targeted block, inclusively the diagonal ones. Based on each coordinate found for the neighbors, the state of the block must be found, and only if the state are the same with the mined one, then that block is mined also. Figure 4.10 illustrates the pseudocode for mining 3x3 blocks in Minecraft.

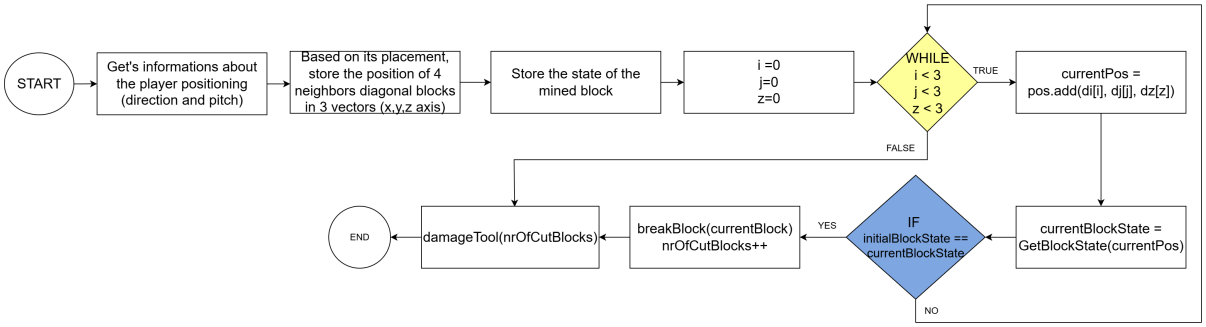

Figure 4.10: Algorithm for mining 3x3 blocks pseudocode

Chapter 5. Detailed Design and Implementation

5.1. General project overview

In this section, I will present the general overview of the CraftMastery mod. The block diagram from Figure 5.1 shows a high-level architecture of the project, illustrating the main classes and packages. The class colored green represents the main class, the components colored yellow handle the block, and the block entity registration, the components colored blue handle the GUI registration of the blocks, the components colored pink handle the items registration, the class colored cyan is responsible for creating the group, the components purple handle the recipe registration, and the package in orange contains the tags from the game.

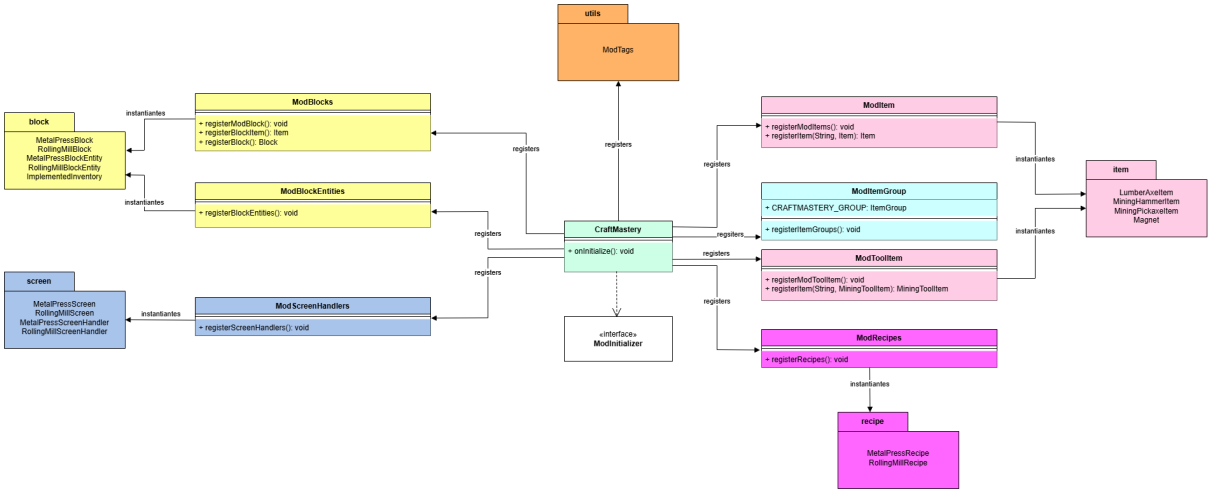

Figure 5.1: CraftMastery block diagram

Since, the name of the project is *CraftMastery*, the main class where we register all the components into the game must have the same name, and must implement the ModInitializer interface. In the *onInitialize()* method, all the registering methods are called from all mod elements: blocks, items, block entities, screens, tags, recipes. *ModBlocks* and *ModBlockEntities* are responsible for registering the respective block and block entities. *ModScreenHandlers* is responsible for registering the GUI components, *ModRecipes* for registering the custom recipes, and *ModItem* together with *ModToolItem* registry the items into the game(simple items or tools). Inside the *ModItemGroup* class colored in cyan, a new tab is created in the game which can be visualized when the player is in the creative mode in the game and where all the components added to the game are stored. Figure B.1 shows how this looks.

The project is organized in packages for having a better structure by grouping the related classes together. This ensures security, readability, and maintainability.

- **block** package is the place where the blocks are defined. Here are created also 2 other packages: **custom** where are defined the physical block from the game(*RollingMillBlock* and *MetalPressBlock*), and **entity** where are created the block entities attached to the blocks(*RollingMillBlockEntity*, *MetalPressBlockEntity*).
- **item** package is the place where the items are created, here also being an extra package **custom** where *LumberAxeItem*, *MiningHammerItem*, *MiningPickaxe*, *Magnet* are created.
- **recipe** package stores the custom recipes created: *MetalPressRecipe*, *RollingMillRecipe*.
- **screen** package is responsible for storing the *Menu* and *Screen* classes: *MetalPressScreen*, *MetalPressScreenHandler*, *RollingMillScreen*, *RollingMillScreenHandler*.
- **utils** package store the *ModTags* class where all the tags from the game are created.

Information regarding how the *.json* files and other files are stored in packages can be found at Section 4.3.

5.2. Registries

Registry is the most important operation that is performed in Minecraft modding, because any addition to the game has to be registered in order to be recognized correctly in the game.

5.2.1. Registering an item

In this section, I will show how a simple item is registered in the game.

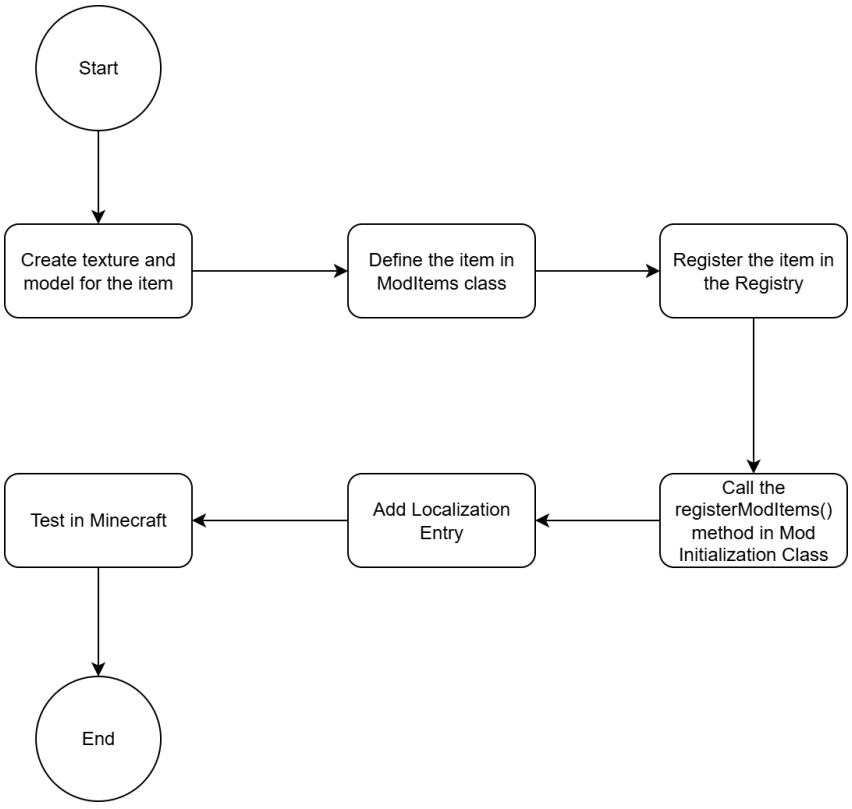

Figure 5.2: Steps for registering an item into the game

In order to be much easier to talk about the steps from Figure 5.2, I will take as example Tool Rod. First step is to create the texture for the desired item. This can be done in a tool such as Blockbench or Asprite. The *.png* file which is created, is placed under *resources/assets/craftmastery/textures/item* with a meaningful name.

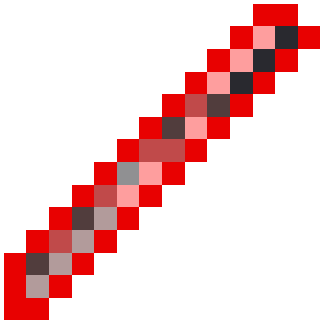

Figure 5.3: Tool Rod texture

Figure 5.3 represents the texture which is drawn as 16x16 pixels image for Tool Rod. All the item textures added in the game have the same size.

Second step is to define the model for this texture. The next code snipped represents the model for Tool Rod, placed in *resources/assets/craftmastery/models/item* directory:

```
{
  "parent": "item/generated",
  "textures": {
    "layer0": "craftmastery:item/tool_rod"
  }
}
```

Listing 5.1: Tool Rod model

The item will have the default (basic) properties, since the parent model inherits the commonly model for 2D flat items (item/generated). The texture for Tool Rod, represented by *layer0*, is taken from *textures/item/tool_rod.png*, and links to the actual imported model.

```
1 private static Item registerItem(String name, Item item) {
2         return Registry.register(Registries.ITEM, new Identifier(
              CraftMastery.MOD_ID, name),item);
3     }
4
5 public static final Item TOOL_ROD = registerItem("tool_rod", new
    Item(new FabricItemSettings()));
```

Listing 5.2: Defining and registry the item in the ModItem class

Third and fourth steps are to define the item in the ModItem class and to register the item in the Registry. The *registerItem(String string, Item item)* helper method is used to register the Tool Rod item into the Minecraft game. Creates a special ID, unique, with which the game knows that it corresponds to a single item, making it visible in Minecraft.

The last 2 lines create a basic Item, but with the help of the above method, we registered it.

```
public static void registerModItems() {
        CraftMastery.LOGGER.info("Registering Mod Items for " +
           CraftMastery.MOD_ID);

}

@Override
public void onInitialize() {
    LOGGER.info("Hello Fabric world!");
    ModItems.registerModItems();

}
```

Listing 5.3: registerModItems method and it's call in the CraftMastery class

Fifth step is to make sure that the ModItem class is loaded correctly in the game, and this is done by creating a method in ModItem class and to be called in the main class of the project, ***CraftMastery***, into *onInitialize()* method.

In order to have a readable name for the item and to not have displayed the ID which was created during registration, we have to define in *en_us.json* file from *resources/assets/craftmastery/lang* directory for translating. Additional, can be added tooltip for it in order to have a description about the item. The next listing shows how translation for Tool Rod is done.

```
{
  "item.craftmastery.tool_rod": "Tool Rod",
}
```

Listing 5.4: Localization entry

The last step called test in Minecraft, more informations can be found in the Section 6.5. Informations regarting its recipe can be found in Section 5.13.

5.2.2. Registering a block

Same as for registering an item, first of all I will provide a diagram where all the steps are described 5.4.

The steps are almost the same as registering an item, but here are added extra steps regarding the item dropped when the block is destroyed and the block state. First, we have to create the texture for the block. It can be multiple textures based on our preferences, the block can have the same texture on each face, but also different textures on every face. Further on, in the model's directory for each block we map the texture to the model created.

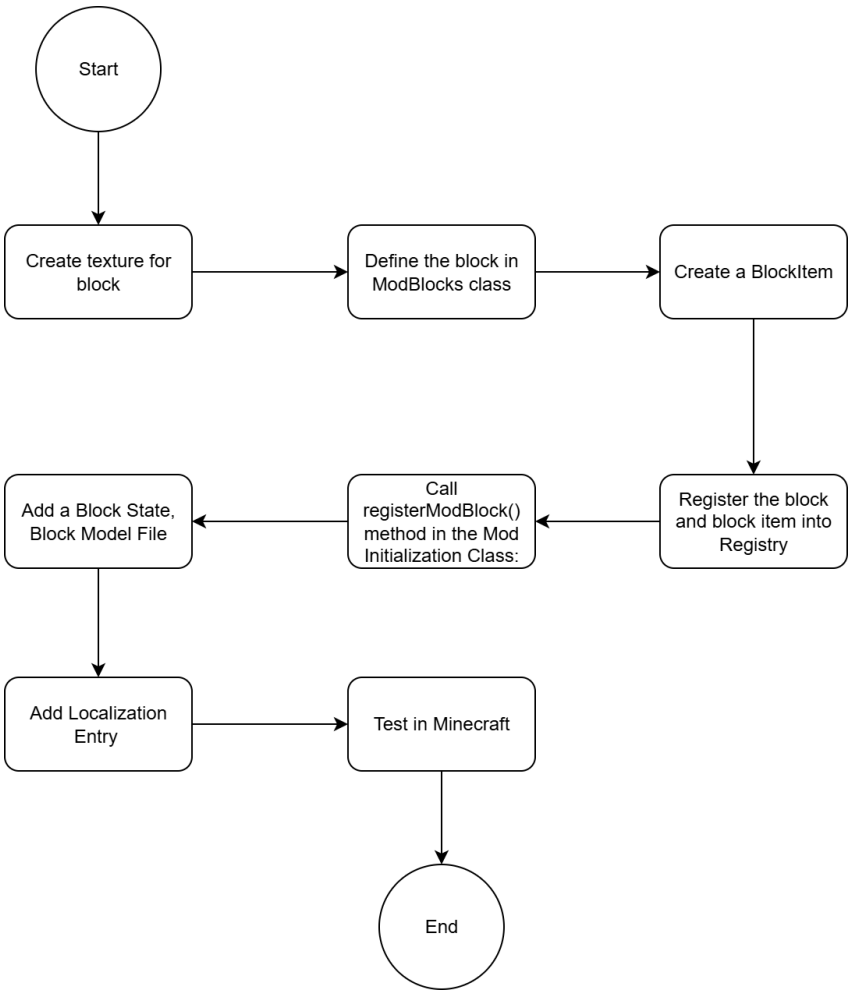

Figure 5.4: Steps for registering a block into the game

In my case, on metal press block, I decided to have the same texture from Figure 5.5 on all faces of the cube. The size of the block is 16x16 pixels.

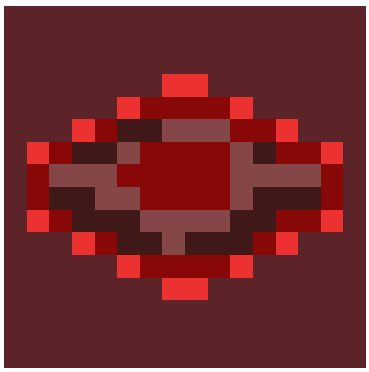

Figure 5.5: Metal Press texture

In the Listing A.1 are presented the steps 2,3 and 4 all being done in the ModBlocks class. Defining the Metal Press block is done at lines 4 and 5. The new block, will have the same properties as an iron block from the game (material, sound, resistance), and also to be non-opaque (transparent). BlockItem is created and registered at line 12, and with this, it lets to take the dropped block to your inventory. In method registerBlock()

from line 17 is called the previous function and is the place where the actual register for BlockItem and Block takes part.

In the same manner as for registering an item, we call the registerBlockMod() method in the CraftMastery class, in the onInitialize() method (main function) in order to make sure that the ModBlocks class is loaded.

```
1
2 public class CraftMastery implements ModInitializer {
3   public static final String MOD_ID = "craftmastery";
4   public static final Logger LOGGER = LoggerFactory.getLogger(
      MOD_ID);
5
6   @Override
7   public void onInitialize() {
8     LOGGER.info("Hello Fabric world!");
9     ModBlocks.registerModBlock();
10  }
11 }
```

Listing 5.5: CraftMastery class

Listing 5.6 shows the block state file for Metal Press situated under *resources/assets/craftmastery/blockstate* and where we decide how our block will look in the game. In our case, it has default values, only the model is imported.

```
{
  "variants": {
    "": {
      "model": "craftmastery:block/metal_press"
    }
  }
}
```

Listing 5.6: Metal Press blockstate

Listing 5.7, show the resources/assets/craftmastery/models/item/metalpress.json file, and here is the placed where we define how the dropped block will look like. In this case, it simply points to the block model.

```
{
  "parent": "craftmastery:block/metal_press"
}
```

Listing 5.7: Metal Press blockItem

Add localization entry step is for giving a name and tooltip (description) for each block. We need to edit the en_us.json file and to specify there which name will have the block in the game. Next snipped code takes part from language file located at *resources/assets/craftmastery/lang*.

```
{
  "block.craftmastery.metal_press": "Metal Press",
  "tooltip.craftmastery.metal_press.tooltip": "Metal Press
     machine which makes the Tool Plate",
}
```

Listing 5.8: Add localization entry

The last step called test in Minecraft, more informations can be found in the Figure 6.7. Informations regarting its recipe can be found in Section 5.6.

5.3. Custom items

Custom items or tools are items that are used by the players in order to make and action for obtaining resources faster[22]. Each tool introduces new functionality from the basic ones that can already be found in the game. Lumber Axe, Mining Hammer, Mining Pickaxe has it's own class which extend the *MiningToolItem* class because it comes with built-in functionalities of the item, for example, mining speed. The magnet is extending the Item class because it does not need to have all the properties that a mining tool has. All the items override a method which specifies when the action to take, for example: after mining a block, while clicking right. Also, all the classes extends the *appendTooltip()* method for displaying an extra information about the item when the player hovers over it. Let's talk about the implementation of each one.

5.3.1. Lumber Axe

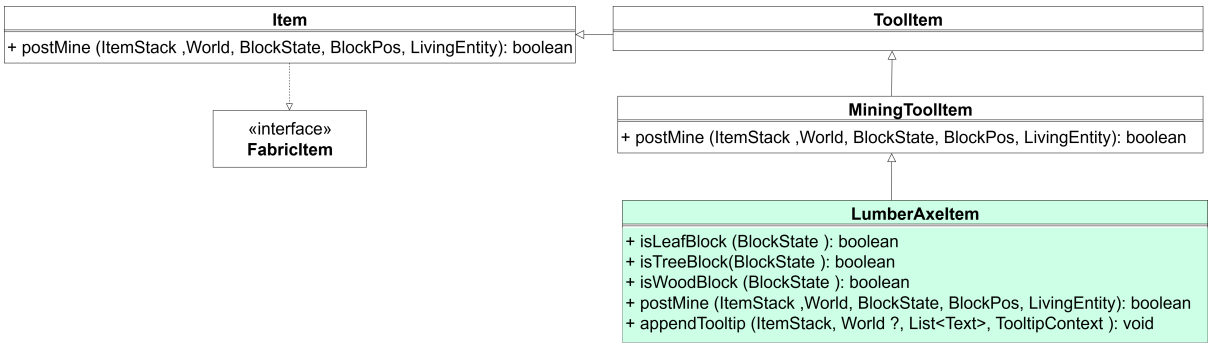

Figure 5.6: LumberAxeItem class diagram

Figure 5.6 illustrates the diagram of UML classes. *LumberAxeItem* class extends the *MiningToolItem* from where the *postMine()* method is overridden in order to perform an action after cutting down a wood block. Inside this method, is the entire logic of the item. There is implemented a custom Depth-first search algorithm for finding all the wood blocks from the tree. This is done by using a *Stack* to store the positions and states of neighboring blocks , and a *Set* for storing the visited blocks. When we talk about the neighbor of a block, we consider those that are situated up, down, north, south, west, east, and the diagonals. Listing A.2 shows how the neighbors are stored in the stack.

Here, are implemented 2 different tags, one which contains all the types of wood, and the other one which stores all types of leaves from the game. Those are used in the implementation of Lumber Axe for a faster comparison of the current block with each type of leaves and woods. First we store in the stack all the leaves and woods of the tree, and at the end we cut down the block only if it is a wood, and if the state of the current block is the same as the initial one which was cut down, with other words, if the blocks are of the same type of wood.

```
1
2 if(isWoodBlock(currentBlockState) && initialBlockState.getBlock()
      == currentBlockState.getBlock()) {
3                     world.breakBlock(currentBlockPos, true);
4                     nrOfBlocksCut++;
5
6                 }
```

Listing 5.9: Cutting down a wood block

The durability of the Lumber Axe is calculated using the simple rule of 3. First, we calculate the durability of one iron ingot, which can be easily done by dividing the durability of the iron tool to the number of iron ingots that are needed to craft it. After that, the durability of Lumber Axe is calculated by multiplying the durability of one iron ingot with the number of iron ingots used in crafting the new tool. The next forumas shows how the durability for Lumber Axe was obtained.

$$\text{DurabilityOfOneIronIngot} = \frac{\text{DurabilityOfIronAxe}}{\text{NumberOfIronIngotsUsedInCraftingTheAxe}} = \frac{250}{6} = 83.33 \tag{5.1}$$

$$\begin{aligned}\text{Durability} &= (\text{DurabilityOfOneIronIngot}) \times (\text{IronIngotsUsedInCraftingTheLumberAxe}) \\ &= 83.33 \times 16 \text{ (12 for tool plate + 4 for tool rod)} \approx 1328\end{aligned} \tag{5.2}$$

The recipe for obtaining Lumber Axe is stored in a *.json* file under *resources/-data/craftmastery/recipes/lumber_axe_recipe.json*. The structure of the file looks like in the Listing A.3.

Tool Rod and *Tool Plate* are the items necessary for crafting the *Lumber Axe*, and there must be respect a defined pattern in order to have as result Lumber Axe: 2 *Tool Plates* on the first two postions from first row, one *Tool Plate* on the second row, first position and one *Tool Rod* on the second position, and in the end, one *Tool Rod* on the third row, second position.

Figure 5.7: Obtaining Lumber Axe from crafting table

In the Figure 5.7 is presented the correct items with the correct pattern from the crafting table in order to obtain the *Lumber Axe*.

5.3.2. Mining Hammer

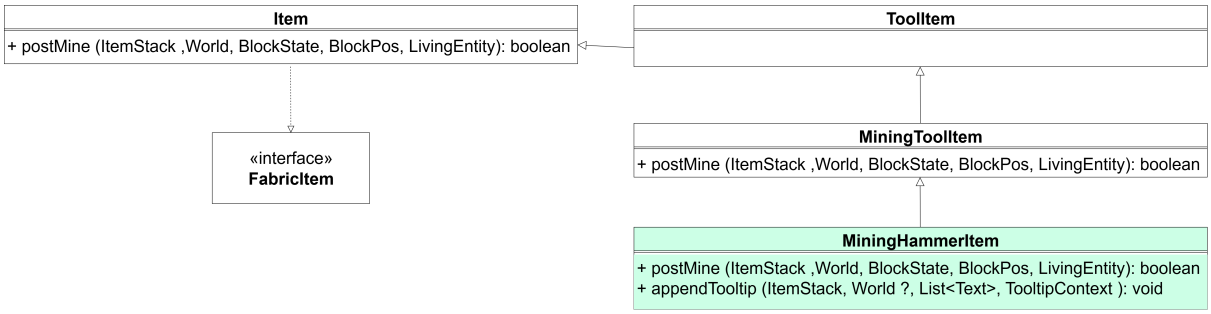

Figure 5.8: MiningHammerItem class diagram

Figure 5.8 illustrates the diagram of UML classes *MiningHammerItem* class extends the *MiningToolItem* class, and with the help of *postMine()* method from the extended class, we can easily provide actions after breaking a block with the Mining Hammer tool. The scope of this tool is that after mining a block, all the same blocks are also mined from a 3x3 area. The key of the implementation is to find all the 8 neighbors correctly based on which direction the player is mining. The player can mine in 6 directions: north, south, east, west, up and down. For the first 4 directions, we can easily find which one is correct, by comparing the direction of the player with each one. From here we can determine that only two coordinates are changed, for example, mining in north or south direction, z coordinate for all the neighbor blocks remains the same, while only x and y coordinates are changing. To determine if the player is mining up or down, we used the pitch, which tells how up or down the player is looking. If the value of the pitch is under -45 degrees, the player's direction is down, while if the value is higher than 45 degrees, the player's direction is up. In the Listing A.4 we can see how this is translating in Java.

After storing the positions of each neighbor, we find if the state of the blocks is the same as the mined one, if this is true, we break all the blocks. At the end of the implementation, we damage the block with the number of blocks which were mined. The total durability of the Mining Hammer is 2166, and the reasons why is this number can be find in next formulas.

$$\text{DurabilityOfOneIronIngot} = \frac{\text{DurabilityOfIronHammer}}{\text{NumberOfIronIngotsUsedInCraftingTheHammer}} = \frac{500}{6} = 83.33 \tag{5.3}$$

$$\begin{aligned}\text{Durability} &= (\text{DurabilityOfOneIronIngot}) \times (\text{IronIngotsUsedInCraftingTheMiningHammer}) \\ &= 83.33 \times 26 \text{ (24 for tool plate + 2 for tool rod)} \approx 2166\end{aligned} \tag{5.4}$$

The recipe for obtaining Mining Hammer is stored in a *.json* file under *resources/-data/craftmastery/recipes/mining_hammer_recipe.json*. The structure of the file looks like in the Listing A.5.

Tool Rod and Tool Plate are the items necessary for crafting the Mining Hammer, and there must be respect a defined pattern in order to have as result Mining Hammer: 6 Tool Plates on the first 2 rows, and a Tool Rod on the second position from the third row.

Tool Rod and *Tool Plate* are the items necessary for crafting the *Mining Hammer*, and there must be respect a defined pattern in order to have as result Mining Hammer: 3 *Tool Plates* on the first and second row, one *Tool Rod* on the second position from third row.

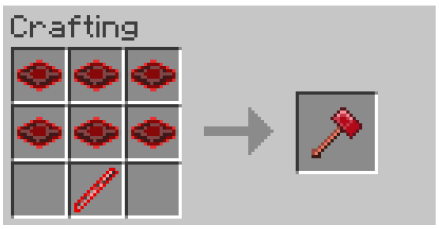

Figure 5.9: Obtaining Mining Hammer from crafting table

In the Figure 5.9 is presented the correct items with the correct pattern from the crafting table in order to obtain the *Mining Hammer*.

5.3.3. Mining Pickaxe

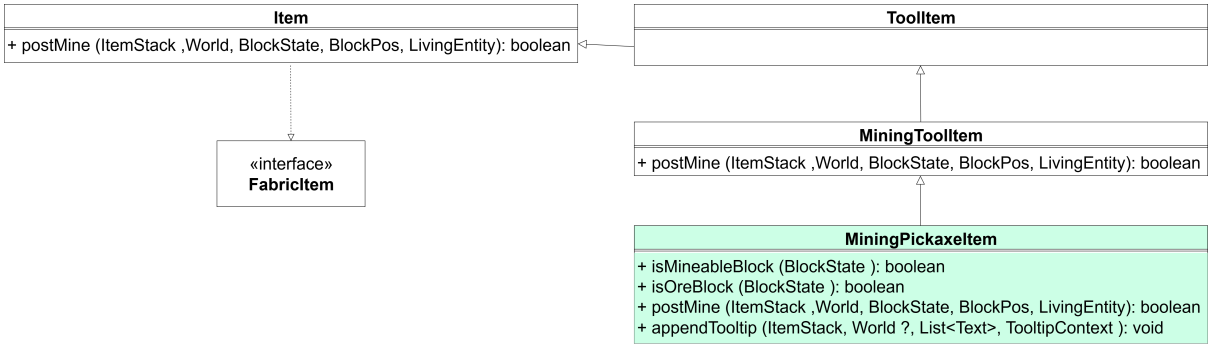

Figure 5.10: MiningPickaxeItem class diagram

Figure 5.10 illustrates the diagram of UML classes. *MiningPickaxeItem* class extends the *MiningToolItem* class because we want to implement an action which is done after a block is mined, and we can do this by overriding the function *postMine()*. The scope of Mining Pickaxe is to mine all the ore blocks connected to the destroyed one, also the state of the blocks must be the same (to be the same type of ore block). The implementation is almost the same with the one for *Lumber Axe*, Listing A.2 showing how the neighbors of a block are found. Mining Pickaxe also follows a DFS algorithm, but instead of searching for woods, it searches for ore blocks. Here are another two tags that were made: one containing all the ore blocks in the game (iron, gold, diamond, etc.) and another one containing the elements from previous tag and all the blocks that can be mined using pickaxe (stone, cobblestone, etc.).

```
1 private boolean isMineableBlock(BlockState blockState) {
2     return blockState.isIn(ModTags.Blocks.MINING_PICKAXE);
3 }
4
5 private boolean isOreBlock(BlockState blockState) {
6     return blockState.isIn(ModTags.Blocks.ORE_BLOCKS);
```

```
}
```

Listing 5.10: Tags used for Mining Pickaxe

The next formulas show how the durability of Mining Pickaxe was computed.

$$\text{DurabilityOfOneIronIngot} = \frac{\text{DurabilityOfIronPickaxe}}{\text{NumberOfIronIngotsUsedInCraftingThePickaxe}} = \frac{250}{6} = 83.33 \tag{5.5}$$

$$\begin{aligned}\text{Durability} &= (\text{DurabilityOfOneIronIngot}) \times (\text{IronIngotsUsedInCraftingTheMiningPickaxe}) \\ &= 83.33 \times 26 \text{ (12 for tool plate + 2 for tool rod)} \approx 1328\end{aligned} \tag{5.6}$$

The recipe for obtaining Mining Pickaxe is stored in a *.json* file under *resources/-data/craftmastery/recipes/mining_pickaxe_recipe.json*. The structure of the file looks like in the Listing A.6.

Tool Rod and *Tool Plate* are the items necessary for crafting the *Mining Pickaxe*, and there must be respect a defined pattern in order to have as result Mining Pickaxe: 3 *Tool Plates* on the first row, one *Tool Rod* on the second position from second row, and in the end, one *Tool Rod* on the third row, second position.

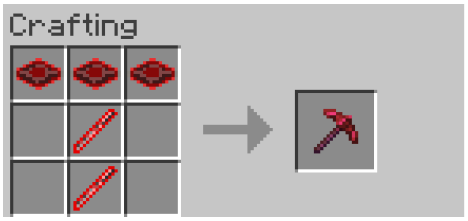

Figure 5.11: Obtaining Mining Pickaxe from crafting table

In the Figure 5.11 is presented the correct items with the correct pattern from the crafting table in order to obtain the *Mining Pickaxe*.

5.3.4. Magnet

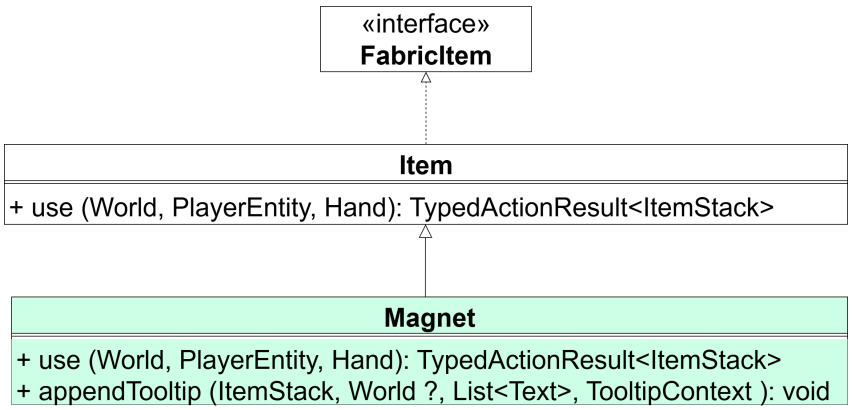

Figure 5.12: Magnet class diagram

Figure 5.12 illustrates the diagram of UML classes. *Magnet* extends the *Item* class because we don't need to do some extra actions after cutting a block. We want just to attract the dropped item entities dropped nearby. For performing this feature, is enough to override the *use()* method from the *Item* class, which is called when the player presses right-click from your mouse. In this method, first of all we define a range around the player from where the dropped item entities are to be picked up (in our case, the range is 5 blocks in each direction). Then, with the help of *getEntitiesByClass()* method we store in a stack all the dropped items from that range. In the end, we go through the stack and put all the items in the inventory of the player. Listing 5.11 shows how insertion is done.

```
for(ItemEntity item: items) {
    ItemStack itemStack = item.getStack();

    user.getInventory().insertStack(itemStack);
    count++;
}
```

Listing 5.11: Inserting in player's inventory

Durability of the *Magnet* is 64, which means that 64 different dropped items can be inserted into the player's inventory.

Listing A.7 shows the recipe for obtaining the *Magnet* in the game, we need 2 Tool Plates, 4 Tool Rod, and one ender pearl. In the first row on positions one and three must be placed Tool Plates, in the second row, first and third positions must be placed Tool Rods, and in the middle (second position) must be placed the ender pearl, and in the last row, are only Tool Rods.

In the Figure 5.13 is presented the correct items with the correct pattern from the crafting table in order to obtain the *Magnet*.

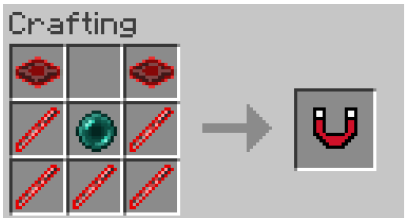

Figure 5.13: Obtaining Magnet from crafting table

5.4. Custom block entities

In this section, I will present how these two blocks added to the game (Rolling Mill and Metal Press) are implemented and attached to them a block entity, in order to provide an action. Both works as a furnace, the differences being the custom recipe presented in the Section 5.6.

5.4.1. Rolling Mill

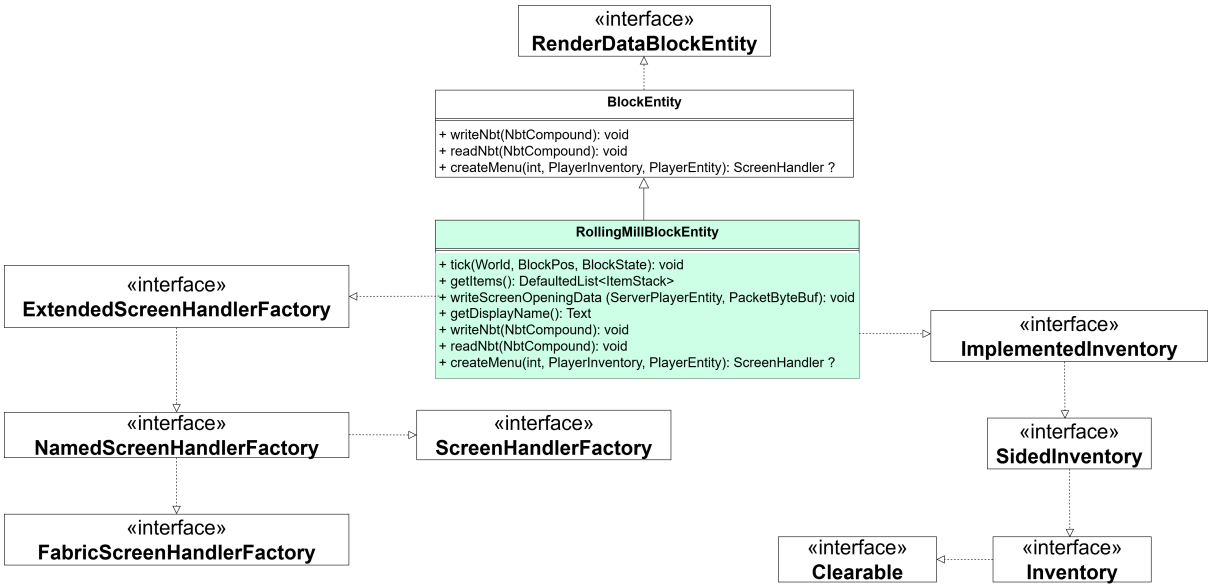

Figure 5.14: RollingMillBlockEntity class diagram

Rolling Mill block entity transforms 2 iron ingots and 3 coal in one *Tool Rod*. It extends the *BlockEntity* class in order to allow ticking which simulates the *Rolling Mill* to be as a machinery. A single tick occurs every 0.05 seconds, or in another words, 20 ticks are in one second.

Figure 5.14 illustrates the diagram of UML classes. In the Listing A.8 is the main logic of *RollingMillBlockEntity* class. At each tick, the system checks if the output slot is empty or if the output slot has space to receive the output. It also checks if the item from input slot is the same with one from recipe and has the minimum amount, and it also checks if the fuel is present and is enough for starting the process. If all the conditions are true, then the system updates the arrow from GUI which tells that the process continues, otherwise, the process resets to the beginning.

5.4.2. Metal Press

Figure 5.15 illustrates the diagram of UML classes. Metal Press is a machinery which transforms 4 iron ingots in one Tool Plate, consuming 5 coals. The implementation of *tick()* method is the same as the one from Listing A.8.

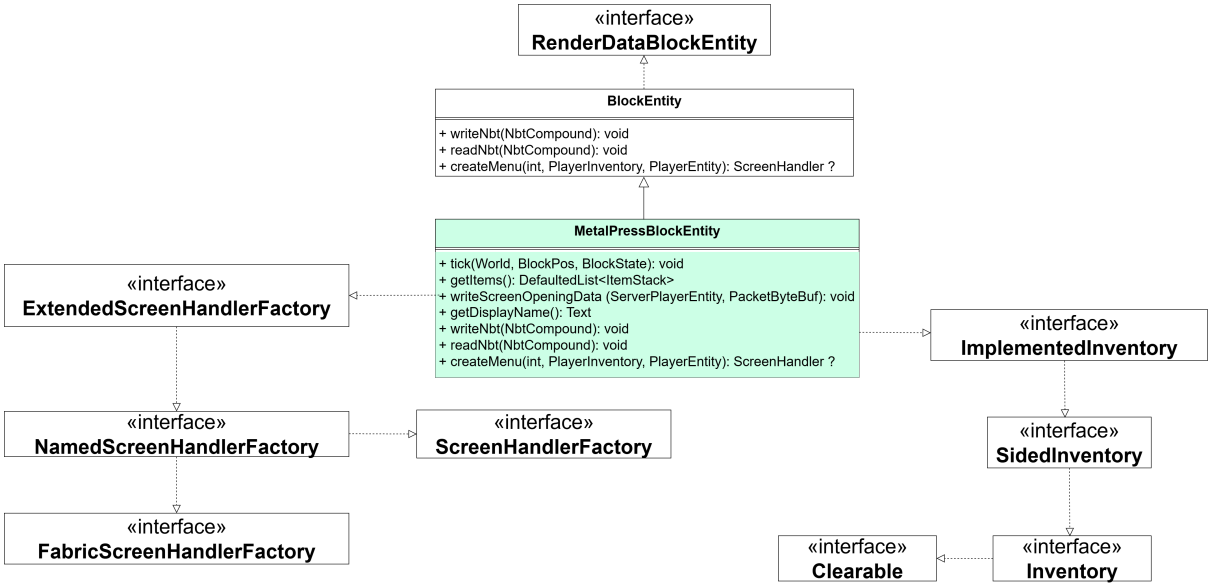

Figure 5.15: MetalPressBlockEntity class diagram

5.5. Custom user interfaces

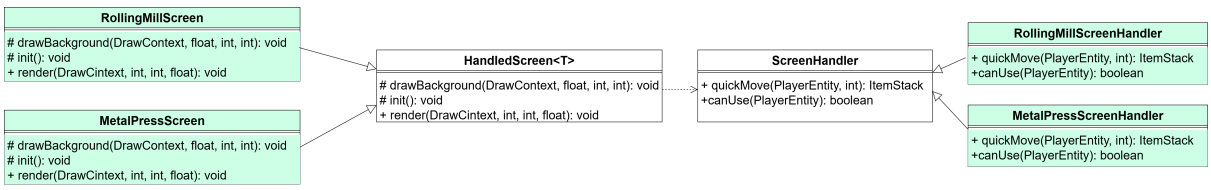

Figure 5.16: Custom user interfaces class diagram

Figure 5.16 illustrates the diagram of UML classes. Custom interfaces define the interaction between the player and the block entities. They show the GUI, and let the player to put items in the input slot and fuel slot, they also simulate the process of smelting the iron ingots, and let the player to collect the output. Here, I have to mention that for *Rolling Mill* and *Metal Press*, implementation for custom interfaces are similar. For realizing this, we need to create two classes: *Menu* and *Screen*. The Menu class is responsible for the logic and acts as a back-end, server side. There is managing the inventory slots, crafting process, and everything which is behind the custom interfaces. On the other hand, Screen is responsible for the visualization part, for the rendering part, tooltips, in other words, it is responsible for what the client should see (client side).

MetalPressScreen and *RollingMillScreen* classes extends the abstract class *HandledScreen* and are responsible for drawing and rendering. It draws the menu of the block entity which is saved under *resources/assets/craftmastery/textures/gui*. In the method drawBackground() from Listing 5.12, shows the steps of drawing the GUI, in the end being called the *renderProgressArrow()* method which is responsible for showing that the process of smelting is going on.

```
1 @Override
2 protected void drawBackground(DrawContext context, float delta,
    int mouseX, int mouseY) {
```

```
3       RenderSystem.setShader(GameRenderer::getPositionTexProgram);
4       RenderSystem.setShaderColor(1f, 1f, 1f, 1f);
5       RenderSystem.setShaderTexture(0, TEXTURE);
6       int x = (width - backgroundWidth) / 2;
7       int y = (height - backgroundHeight) / 2;
8
9       context.drawTexture(TEXTURE, x, y, 0, 0, backgroundWidth,
            backgroundHeight);
10
11      renderProgressArrow(context, x, y);
12  }
```

Listing 5.12: drawBackground() method

In the *MetalPressScreenHandler* and *RollingMillScreenHandler* classes, called also *Menu* class, are responsible for handling the inventory and the logic for the block entities logic. In the constructor of the classes are declared the inventory slots and hotbar. Also here, is created the method *quickMove()* which has multiple functionalities. Is responsible for shift-clicking the items for moving fast the item from inventory to the slots and the other way around, here is also made the logic for stack splitting, and only if the inventory slot is empty to put the item there. isCrafting() and getScaledProgress() are used to calculate and to upgrade the progress of the arrow, and canUse() checks the distance from where the player can access the block entity. Listing A.9 shows the methods described above.

5.6. Custom recipe types

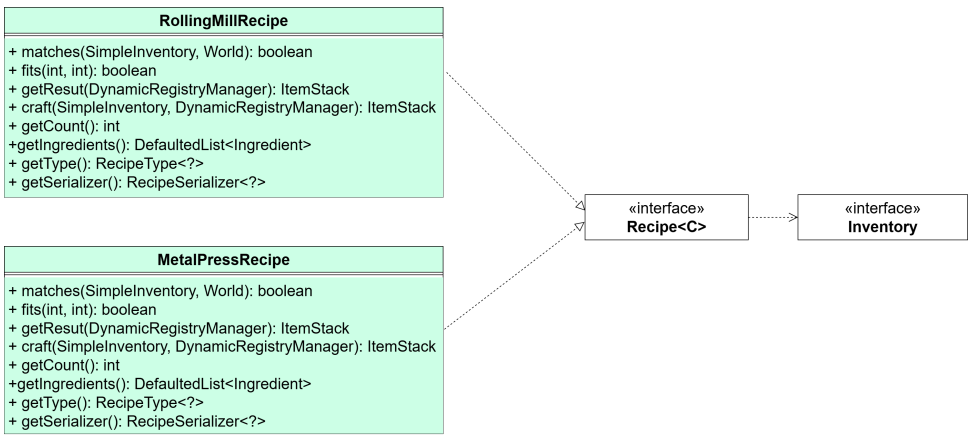

Figure 5.17: Custom recipe types class diagram

Figure 5.17 illustrates the diagram of UML classes. Both block entities, the Rolling Mill and the Metal Press, work as a furnace, they smelt iron ingot in order to obtain the Tool Rod and Tool Plate. The recipes that were created for the tools from Section 5.3, are the simplest one and are done in Crafting Table. For having the smelting effect of the machineries, we need to define custom recipes for them. In the Listing 5.13 we can see an example of the *.json* file for obtaining the Tool Plate. We need to smelt 4 iron ingots in order to obtain one item. The *"count"* field, is not by default in the structure of the file, it was added by me, an in the *Serializer* class it was proceed.

```
{
  "type": "craftmastery:metal_press",
  "count": 4,
  "ingredients": [
    {
      "item": "minecraft:iron_ingot"
    }
  ],
  "output": {
    "item": "craftmastery:tool_plate"
  }
}
```

Listing 5.13: Metal Press custom recipe

In the *Recipe* class, we define a list of items that must be in the input slot of the machinery, a count for the minimum number of items in the input slot, and the output. The method *matches()* checks if the items and the number of them from the input slots match what is written in the *.json* file. The *Type* class is created for making the connection between the *.json* file and the logic of the machinery. The *Serializer* class is used for read and write the recipe from the *.json* file.

Chapter 6. Testing and Validation

This section is about the validation of the use cases described in Section 4.2 by testing their context in the game. We have to specify that everything was tested in survival mode, since in this way we can meet all the possible scenarios during testing.

6.1. Crafting and using the Lumber Axe

To validate the functionality of Lumber Axe, we have to follow the steps presented in Figure 4.1. As a precondition, we must have access to a crafting table, and the output must be the action of cutting down a tree. If the materials used for crafting are the correct ones (Tool Rod and Tool Plate), and also the pattern was respected (see Figure 5.7), the system displays the Lumber Axe.

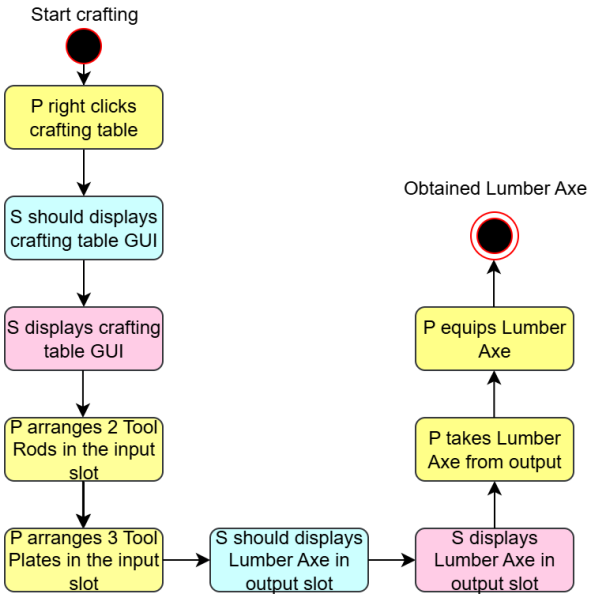

Figure 6.1: Crafting the Lumber Axe test

In Figure 6.1, are described the steps that were performed when we tested the crafting of the Lumber Axe. Rectangles which are colored in yellow describe the action which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output.

After crafting the tool, it is ready to be used in the world. In order to test its functionality, we consider the following scenarios. In the first scenario, we looked for a tree in the world, numbered how many wood blocks are in order to test the durability, and cut the one from the base. The expected output is that the durability of Lumber

Axe decreases with exactly the number of wood blocks that the tree had, and also the tool is performing a depth-first search algorithm on the wood blocks and cuts them down. The actual output is that all wood blocks are cut down in a DFS manner and durability decreases with exactly the number of blocks that were in the tree. In the second scenario, we break a block with Lumber Axe which was not a wood block(dirt). The expected output is to break only that block, drop it in the world, and the durability to decrease with one. The actual output is that the durability decreases with one, and only that block is broken and dropped. The third scenario is to break a block which an ax, in real life, would take a long period of time to destroy it, such as stone. The expected output is that it takes a long time to break it, durability decreases with one, and the destroyed block is not dropped in the world. The actual output is that the durability reduces with one, the process takes a long period of time to break down the stone block, and it is not dropped in the world.

Table 6.1: Lumber Axe testing scenarios with expected and actual results.

Test	**Expected Output**	**Actual Output**
Player right-clicks crafting table.	System should display the crafting table GUI.	System displays the crafting table GUI.
Player arranges 2 Tool Rods and 3 Tool Plates in the input slot.	System should display Lumber Axe in the output slot.	System displays Lumber Axe in the output slot.
Locate a tree in the world, numbered how many wood blocks are, and cut the block from the base of the tree.	Durability of the tool decreases with exactly the number of wood blocks that the tree had, perform DFS and cut all wood blocks down.	Durability of the tool decreases with exactly the number of wood blocks that the tree had, perform DFS and cut all wood blocks down.
A dirt block is broken using the Lumber Axe.	The dirt block is broken and dropped, and durability decreases by one.	The dirt block is broken and dropped, and durability decreases by one.
A stone block is broken using the Lumber Axe.	The process takes a long time, durability decreases by one, and the block is not dropped.	The process takes a long time, durability decreases by one, and the block is not dropped.

In the Table 6.1 is a summary of the scenarios described in this sub-chapter.

6.2. Crafting and using the Mining Hammer

To validate the functionality of Mining Hammer, we have to follow the steps presented in Figure 4.2. As a precondition, we must have access to a crafting table, and the output must be the action of mining 3x3. If the materials used for crafting are the correct ones (Tool Rod and Tool Plate), and also the pattern was respected (see Figure 5.9), the system displays the Mining Hammer.

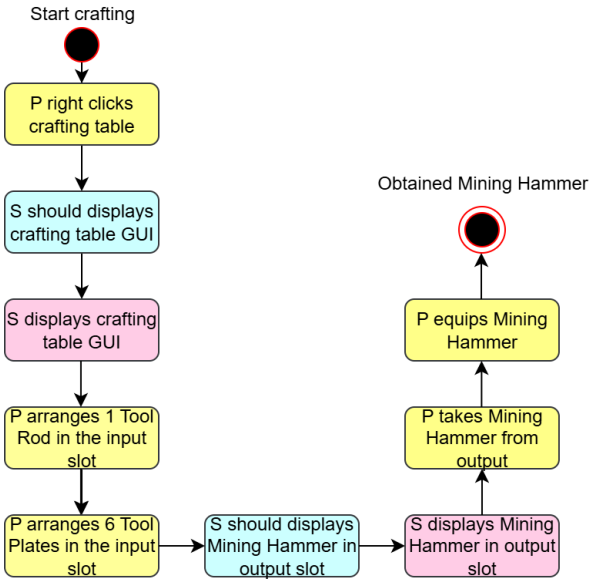

Figure 6.2: Crafting the Mining Hammer test

In Figure 6.2, are described the steps that were performed when we tested the crafting of the Mining Hammer. Rectangles which are colored in yellow describe the action which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output.

After crafting the tool, it is ready to be used in the world. In order to test its functionality, we consider the following scenarios. In the first scenario, we aimed down towards a dirt block. The expected output is that Mining Hammer only mines the targeted dirt block. The actual output is that Mining Hammer only mines the targeted dirt block. In the second scenario, we aimed down toward a stone block. The exptected output is that Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane. The actual output is that Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane. In the third scenario, we aimed up towards a dirt block. The expected output is that the Mining Hammer only mines the targeted dirt block. The actual output is that the Mining Hammer only mines the targeted dirt block. In the fourth scenario, we aimed up toward a stone block. The expected output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane. The actual output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane. In the fifth scenario, we aimed north towards a dirt block. The expected output is that the Mining Hammer only mines the targeted dirt block. The actual output is that the Mining Hammer only mines the targeted dirt block. In the sixth scenario, we aimed north toward a stone block. The expected output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane. The actual output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane. In the seventh scenario, we aimed south towards a dirt block. The expected output is that the Mining Hammer only mines the targeted dirt block. The actual output is that the Mining Hammer only mines the targeted dirt block. In the eighth scenario, we aimed south toward a stone block. The expected output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.

The actual output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane. In the ninth scenario, we aimed east towards a dirt block. The expected output is that the Mining Hammer only mines the targeted dirt block. The actual output is that the Mining Hammer only mines the targeted dirt block. In the tenth scenario, we aimed east toward a stone block. The expected output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane. The actual output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane. In the eleventh scenario, we aimed west towards a dirt block. The expected output is that the Mining Hammer only mines the targeted dirt block. The actual output is that the Mining Hammer only mines the targeted dirt block. In the twelfth scenario, we aimed west toward a stone block. The expected output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane. The actual output is that the Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.

Table 6.2: Mining Hammer testing scenarios in different directions and block types.

Test	**Expected Output**	**Actual Output**
Player right-clicks crafting table.	System should display the crafting table GUI.	System displays the crafting table GUI.
Player arranges 1 Tool Rod and 6 Tool Plates in the input slot.	System should display Mining Hammer in the output slot.	System displays Mining Hammer in the output slot.
Aimed down towards a dirt block.	Mining Hammer only mines the targeted dirt block.	Mining Hammer only mines the targeted dirt block.
Aimed down toward a stone block.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane.
Aimed up towards a dirt block.	Mining Hammer only mines the targeted dirt block.	Mining Hammer only mines the targeted dirt block.
Aimed up toward a stone block.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the horizontal plane.
Aimed north towards a dirt block.	Mining Hammer only mines the targeted dirt block.	Mining Hammer only mines the targeted dirt block.
Aimed north toward a stone block.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.
Aimed south towards a dirt block.	Mining Hammer only mines the targeted dirt block.	Mining Hammer only mines the targeted dirt block.

Aimed south toward a stone block.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.
Aimed east towards a dirt block.	Mining Hammer only mines the targeted dirt block.	Mining Hammer only mines the targeted dirt block.
Aimed east toward a stone block.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.
Aimed west towards a dirt block.	Mining Hammer only mines the targeted dirt block.	Mining Hammer only mines the targeted dirt block.
Aimed west toward a stone block.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.	Mining Hammer mines all stone-like blocks in a 3x3 area around the targeted stone block in the vertical plane.

In the Table 6.2 is a summary of the scenarios described in this sub-chapter.

6.3. Crafting and using the Mining Pickaxe

To validate the functionality of Mining Pickaxe, we have to follow the steps presented in Figure 4.3. As a precondition, we must have access to a crafting table, and the output must be the action of mining all ore blocks connected to the mined one. If the materials used for crafting are the correct ones (Tool Rod and Tool Plate), and also the pattern was respected (see Figure 5.11), the system displays the Mining Pickaxe.

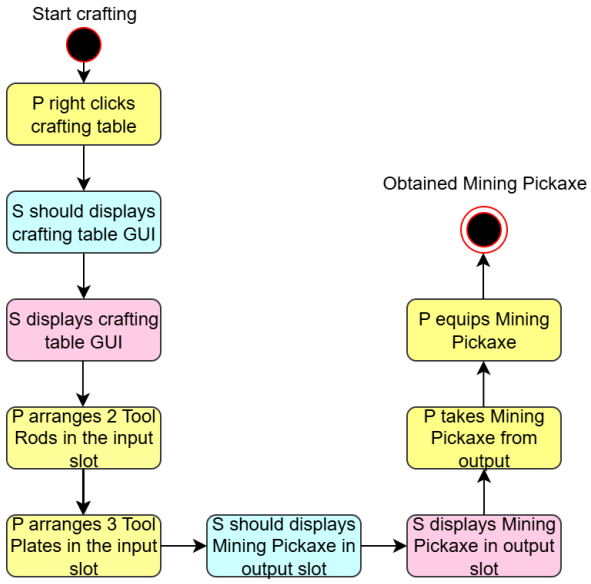

Figure 6.3: Crafting the Mining Pickaxe test

In Figure 6.3, are described the steps that were performed when we tested the crafting of the Mining Pickaxe. Rectangles which are colored in yellow describe the action which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output.

After crafting the tool, it is ready to be used in the world. In order to test its functionality, we consider the following scenarios. In the first scenario, we looked for a ore block in the world, numbered how many connected blocks of the same type are in order to test the durability, and mined the first one. The expected output is that the durability of Mining Pickaxe decreases with exactly the number of connected ore blocks, and also the tool is performing a depth-first search algorithm on the blocks and mine them. The actual output is that all connected ore blocks are mined in a DFS manner, and durability decreases with exactly the number of ore blocks that are connected. In the second scenario, we break a block with Mining Pickaxe which was not an ore block(stone). The expected output is to break only that block, drop it in the world, and the durability to decrease with one. The actual output is that the durability decreases with one, and only that block is broken and dropped.

Table 6.3: Mining Pickaxe testing scenarios with ore and non-ore blocks.

Test	**Expected Output**	**Actual Output**
Player right-clicks crafting table.	System should display the crafting table GUI.	System displays the crafting table GUI.
Player arranges 2 Tool Rods and 3 Tool Plates in the input slot.	System should display Mining Pickaxe in the output slot.	System displays Mining Pickaxe in the output slot.
Mined a connected ore block with the Mining Pickaxe after counting how many connected blocks of the same type are.	Durability of Mining Pickaxe decreases with exactly the number of connected ore blocks, and also the tool is performing a DFS on the blocks and mine them. them.	Durability of Mining Pickaxe decreases with exactly the number of connected ore blocks, and also the tool is performing a DFS on the blocks and mine them.
Broke a stone block using the Mining Pickaxe.	Only the stone block is broken and dropped. Durability decreases by one.	Only the stone block is broken and dropped. Durability decreases by one.

In the Table 6.3 is a summary of the scenarios described in this sub-chapter.

6.4. Crafting and using the Magnet

To validate the functionality of Magnet, we have to follow the steps presented in Figure 4.4. As a precondition, we must have access to a crafting table, and the output must be the action of put in the inventory all the dropped items and blocks from a 5 blocks distance in each direction. If the materials used for crafting are the correct ones (Tool Rod, Tool Plate, and Ender Pearl), and also the pattern was respected (see Figure 5.13), the system displays the Magnet.

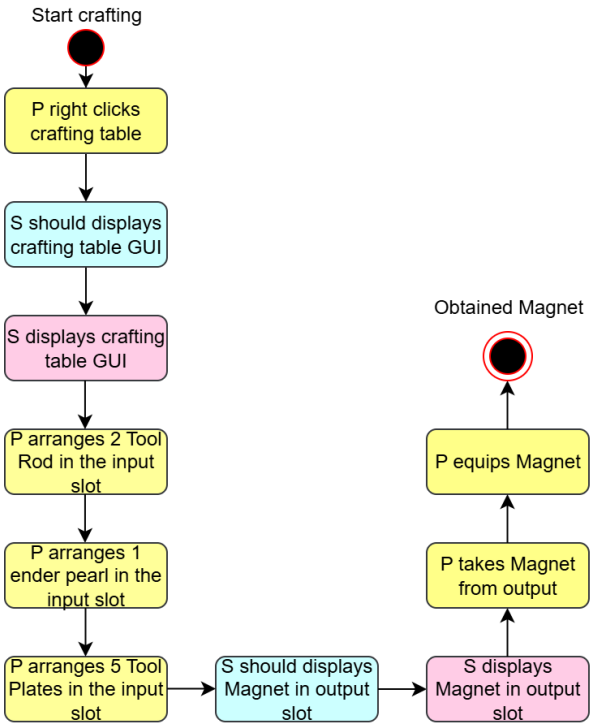

Figure 6.4: Crafting the Magnet test

In Figure 6.4, are described the steps that were performed when we tested the crafting of the Magnet. Rectangles which are colored in yellow describe the action which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output.

After crafting the tool, it is ready to be used in the world. In order to test its functionality, we consider the following scenarios. In the first scenario, we dropped items and blocks at 5 blocks distance in each direction, count them, equipped the Magnet and right click. The expected output is to pulls all nearby dropped items and the durability decreases with the exact number of distinct items pulled. The actual output is that Magnet pulls all nearby dropped items and the durability decreases with the exact number of distinct items pulled. In the second scenario, we dropped items and blocks at more than 5 blocks distance in each direction, equipped the Magnet and right click. The expected output is not to pulls the dropped items and the durability not to decrease. The actual output is that Magnet does not pulls all dropped items and the durability does not decreases.

Table 6.4: Magnet tool testing scenarios with varying item distances.

Test	**Expected Output**	**Actual Output**
Player right-clicks crafting table.	System should display the crafting table GUI.	System displays the crafting table GUI.

Player arranges 2 Tool Rods, 1 Ender Pearl and 5 Tool Plates in the input slot.	System should display Magnet in the output slot.	System displays Magnet in the output slot.
Dropped items and blocks within 5 blocks in each direction, then equipped and right-clicked the Magnet.	The Magnet pulls all nearby dropped items, and durability decreases by the exact number of distinct items pulled.	The Magnet pulls all nearby dropped items, and durability decreases by the exact number of distinct items pulled.
Dropped items and blocks at more than 5 blocks distance in each direction, then equipped and right-clicked the Magnet.	The Magnet does not pull the dropped items, and durability does not decrease.	The Magnet does not pull the dropped items, and durability does not decrease.

In the Table 6.4 is a summary of the scenarios described in this sub-chapter.

6.5. Crafting the Rolling Mill and the Tool Rod

To validate the the Rolling Mill and the Tool Rod test case, we have to follow the steps presented in Figure 4.5. As a precondition, we must have access to a crafting table, and the output must obtain the Tool Rod.

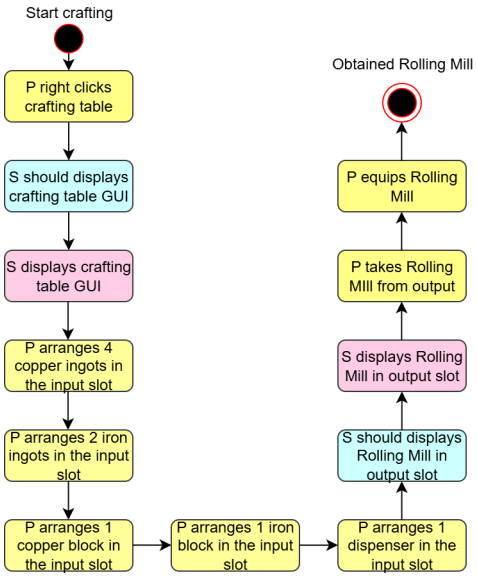

Figure 6.5: Crafting the Rolling Mill test

In Figure 6.5, are described the steps that were performed when we tested the crafting of the Rolling Mill. Rectangles which are colored in yellow describe the action which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output.

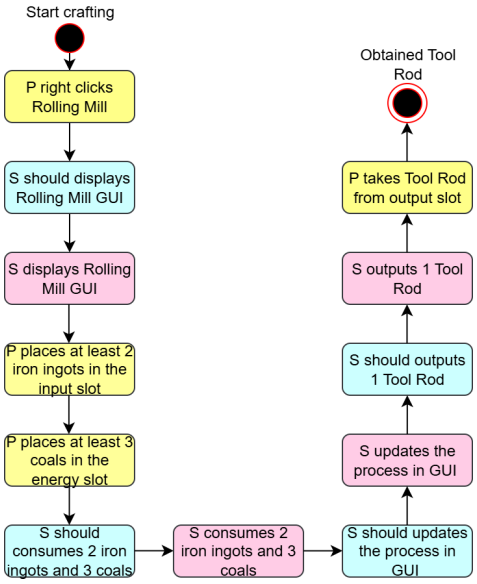

Figure 6.6: Crafting the Tool Rod test

In Figure 6.6, are described the steps that were performed when we tested the crafting of the Tool Rod. Rectangles which are colored in yellow describe the action which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output. If the player puts less items and energy in the input slots than the number required, the process does not start. On the other hand, if he puts more than the threshold items required, the machinery consumes them until they are under the limit.

Table 6.5: Rolling Mill and Tool Rod testing scenarios

Test	**Expected Output**	**Actual Output**
Player right-clicks crafting table.	System should display the crafting table GUI.	System displays the crafting table GUI.
Player arranges 4 copper ingots, 2 iron ingots, 1 copper block, 1 iron block, and 1 dispenser in the input slot.	System should display Rolling Mill in the output slot.	System displays Rolling Mill in the output slot.
Player right-clicks Rolling Mill.	System should display the Rolling Mill GUI.	System displays the Rolling Mill GUI.
Player places at least 2 iron ingots in the input slot and at least 3 coals in the energy slot.	System should consumes 2 iron ingots and 3 coals, should updates the process in GUI, and should outputs 1 Tool Rod.	System consumes 2 iron ingots and 3 coals, updates the process in GUI, and outputs 1 Tool Rod.

In the Table 6.5 is a summary of the scenarios described in this sub-chapter.

6.6. Crafting the Metal Press and Tool Plate

To validate the the Metal Press and the Tool Plate test case, we have to follow the steps presented in Figure 4.6. As a precondition, we must have access to a crafting table, and the output must obtain the Tool Rod.

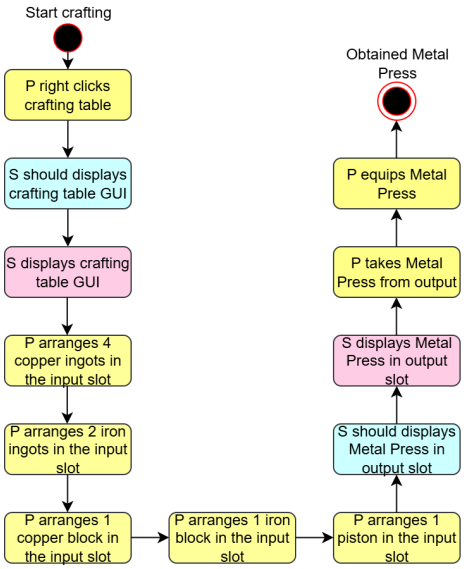

Figure 6.7: Crafting the Metal Press test

In Figure 6.7, are described the steps that were performed when we tested the crafting of the Metal Press. Rectangles which are colored in yellow describe the action which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output.

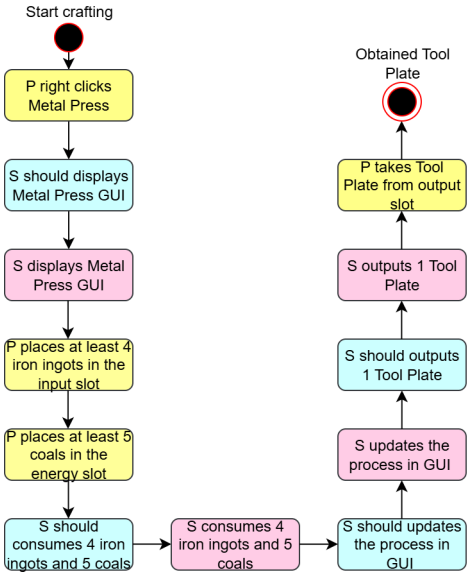

Figure 6.8: Crafting the Tool Plate test

In Figure 6.8, are described the steps that were performed when we tested the crafting of the Tool Plate. Rectangles which are colored in yellow describe the action

which are performed by the player, those colored in cyan illustrate which is the expected output, and those in pink are the actual output. If the player puts less items and energy in the input slots than the number required, the process does not start. On the other hand, if he puts more than the threshold items required, the machinery consumes them until they are under the limit.

Table 6.6: Metal Press and Tool Plate testing scenarios

Test	**Expected Output**	**Actual Output**
Player right-clicks crafting table.	System should display the crafting table GUI.	System displays the crafting table GUI.
Player arranges 4 copper ingots, 2 iron ingots, 1 copper block, 1 iron block, and 1 piston in the input slot.	System should display Metal Press in the output slot.	System displays Metal Press in the output slot.
Player right-clicks Metal Press.	System should display the Metal Press GUI.	System displays the Metal Press GUI.
Player places at least 4 iron ingots in the input slot and at least 5 coals in the energy slot.	System should consumes 4 iron ingots and 5 coals, should updates the process in GUI, and should outputs 1 Tool Plate.	System consumes 4 iron ingots and 5 coals, updates the process in GUI, and outputs 1 Tool Plate.

In the Table 6.6 is a summary of the scenarios described in this sub-chapter.

Chapter 7. User's Manual

In this section, the installation procedure is described together with a step-by-step description of how the application can be deployed. Also, is outlined, from user point of view, how the application can be used.

7.1. Resources and installation procedure

The following resources are needed to be installed in order to use the mod developed using Fabric API:

- Minecraft Java Edition: version 1.20.4
- Fabric Loader: this is needed for letting Fabric mods to run. Version for Minecraft 1.20.4
- Laptop of PC with at least 4GB RAM and Solid State Driven(SSD) with at least 4GB storage
- Connection to the internet

After installing all the resources and dependency needed from the previous sub-chapter, a ***.jar*** file which contains the new mod. This file must be placed in the *mods* folder from Minecraft. To navigate to that folder in Windows operating system, press *Win+R*, then type *%appdata%* and press *Enter*. After that look for *.minecraft/mods* where to place the ***.jar*** file specific for the mod, together with the Fabric API ***.jar*** file. If the *mods* directory is not there, just create it.

7.2. How to use Craft Mastery mod

Craft Mastery is a Minecraft mod created to add a new game experience to the original. It comes with new tools and tools, new blocks, new machineries, new crafting recipes, all this in order to give a more challenging and rewarding progression system for players.

First of all, was added two new items, *Tool Rod*(see Figure 5.3) and *Tool Plate*(see Figure 7.1), which their role is to be used in crafting the new tools. You can obtain the items by smelting iron ingot in a *Rolling Mill* and *Metal Press*. To obtain a *Tool Rod* you have to smelt 2 iron ingots and consume 3 coals in *Rolling Mill* machinery. To obtain a *Tool Plate* you have to smelt 4 iron ingots and consume 5 coals in *Metal Press* machinery.

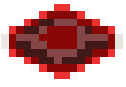

Figure 7.1: Tool Plate texture

Rolling Mill and *Metal Press* machineries are used in order to obtain the *Tool Rod* and *Tool Plate*. For crafting *Rolling Mill* you need to have the following items: 2 iron ingots, 4 copper ingots, 1 copper block, 1 iron block, and 1 dispenser. The pattern that must be followed is illustrated in the Figure 7.2. For crafting *Metal Press* you need to have the following items: 2 iron ingots, 4 copper ingots, 1 copper block, 1 iron block, and 1 piston. The pattern that must be followed is illustrated in the Figure 7.3. After crafting them, you have to place the block in the world and right click it to access its interface. Their Graphical User Interface consists of 3 slots: 2 for inputs(one where to place the iron ingots, and one where to place the coal), and the last slot where the items will be placed after the process is done. In the Figure 7.4 you can see the how the GUI looks like, and where each input items must be placed. You have to respect this pattern from the slots, otherwise the machinery will not start to process. In order to make the *Rolling Mill* to start, you have to place at least 2 iron ingots in the first slot, and at least 3 coals in the right slot. In the same way for *Metal Press* but here you have to place at least 4 iron ingots and at least 5 coals. After the process is done the item will appear in the third slot, from where you can take it and put in your inventory.

Figure 7.2: Rolling Mill crafting table

Figure 7.3: Metal Press crafting table

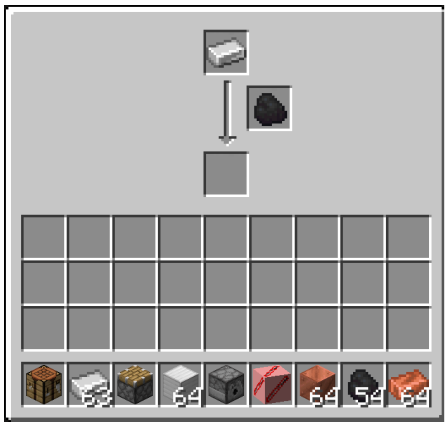

Figure 7.4: Machineries GUI

Lumber Axe(see Figure 7.5), is a tool used for cutting down trees in an efficient way. You can craft it using 2 *Tool Rods* and 3 *Tool Plates* and following the recipe from Figure 5.7. After crafting, the tool is ready to be used in the world. Find a tree and cut a wood block using the *Lumber Axe* and the entire wood blocks from that tree must be automatically cut. If you try to use it on other blocks which are not wood, only that block will be broken. Its durability decreases with the number of cut blocks. The initial durability is 1328.

Figure 7.5: Lumber Axe texture

Mining Hammer(see Figure 7.6), is a tool used to mine the stone and ore blocks in a 3x3 area. You can craft it using 1 *Tool Rod* and 6 *Tool Plates* and following the recipe from Figure 5.9. After crafting, the tool is ready to be used in the world. Start mining a stone block, after the block is broken, automatically all 8 blocks which are around that one(up, down, left, right, and 4 diagonals) are broken. If you try to mine a block which is not from the list mentioned above, only that block will be broken. The durability decreases with exactly the number of mined blocks. The initial durability is 2166.

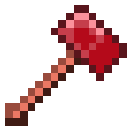

Figure 7.6: Mining Hammer texture

Mining Pickaxe(see Figure 7.7), is a tool used to mine all connected ore blocks of the same type(e.g. iron, diamond, gold, etc.) You can craft it using 2 *Tool Rods* and 3 *Tool Plates* and following the recipe from Figure 5.11. After crafting, the tool is ready to be used in the world. Whenever an ore block is targeted, it is mined, and automatically all the same blocks that are connected to the mined one are broken. If you try to mine a block which is not ore, only that block will be broken. Its durability decreases with exactly the number of mined blocks. The initial durability is 1328.

Figure 7.7: Mining Pickaxe texture

Magnet(see Figure 7.8), is a tool used to pulls dropped items and blocks from a distance of 5 blocks in each direction. You can craft it using 5 *Tool Rods*, 2 *Tool Plates*, 1 *Ender Pearl* and following the recipe from Figure 5.13. After crafting, the tool is ready to be used in the world. Equip the *Magnet* and spot dropped items and blocks which are at a distance less than or equal 5 blocks, right-click on it and all items will be pulled into your inventory. Materials which are at a distance larger than 5 blocks will remain

there. Its durability decreases with the number of different items and blocks collected. The initial durability is 64, that means you can collect 64 different items.

Figure 7.8: Magnet texture

Chapter 8. Conclusions

8.1. Contributions and achievements

The achievements of this project are that all the features and objectives were implemented.

Lumber Axe was added in the mod. It will be used for cutting down trees by breaking only one wood block.

Mining Hammer was added in *Craft Mastery* and will be used for mining 3x3.

Mining Pickaxe was added in the mod. It will be used for mining all the ore blocks of the same type connected to the mined one.

Magnet tool was added, and it will be used to pull all the nearby dropped items and blocks.

Tool Rod and Tool Plate items were added in order to craft the previous tools.

Rolling Mill and Metal Press block entities were added in order to have a special crafting mechanism (machines), which will handle **custom recipes** from obtaining the items mentioned before.

8.2. Results

Table 8.1 shows the feature results between the existing mods which were compared at Section 3.5 and the Craft Mastery mod, developed by me. GregTech is a mod which is specialized in adding machines and materials. Mekanism is specialized in having special tools. Craft Mastery is taking all the good things from these two mods, combine their ideas in order to create a better mod.

Table 8.1: Comparison of features between existing solutions and Craft Mastery mod

Feature	**Mekanism**	**GregTech**	**Craft Mastery**
Tool rods from iron	NO	YES	YES
Tool plates from iron	NO	YES	YES
Metal Press	YES	YES	YES
Rolling Mill	NO	NO	YES
Lumber Axe	NO	NO	YES
Mining Hammer	YES	NO	YES
Mining Pickaxe	NO	NO	YES
Magnet	YES	NO	YES

Functional requirements were solved in Craft Mastery mod by developing a system which consists of custom tools, items, custom block entities, and crafting mechanism.

The new items introduced for crafting the new tools were registered in the game using the registries, and in the resource package is added the texture and configure the language, tooltip and model.

The new tools are introduced by having a class which extends the MiningToolItem class from the Minecraft classes. Each tool override a method where all the logic is implemented. In the resources package the texture is placed, and language and model are created.

Each block entity is created by two class which extends the appropiate classes from Minecraft, one for Menu and one for Screen.

Custom recipes were created for obtaining the new items. A field called "count" were added in the recipe file which is processed in the recipe class. This class implements the Recipe interface.

Faster tree cutting user need is solved by introducing the Lumber Axe which cut down all the wood blocks from the tree.

Efficient mining is implemented in the game with the help of Mining Hammer which mines 3x3 the stone block in all directions (all neighbors of a mined stone block are mined).

Faster mining ore blocks are done with the help of Mining Pickaxe which mines all the connected ore blocks of the same type by mining one block.

Tool Rod and Tool Plate are the new items for crafting the new tools.

New crafting methods for items user need is solved by introducing the Rolling Mill and Metal Press which smelt iron for obtaining the new items added for crafting.

Faster method for pulling nearby objects is held by the Magnet tool which when is used, pulls all the dropped objects from a certain distance.

8.3. Further development

As for further improvements to the project, there are several. I will present them in the next segments.

Integration with other mods. One improvement for the player will be that the mod to be from the beginning integrated with *The One Probe(TOP)* and *Just Enough Items(JEI)* mods. This will help the players because the JEI mod provides all the recipes from the game and is easy for the player to find them, and TOP mod tells the player which is the name of the block or item where he is looking at.

Generating new resource. Another improvement is to generate a new resource in the world like petrol, and to be used as energy for the new block entities added in the game. In the current solution *Rolling Mill* and *Metal Press* works only with coal.

Making new machines. Last further improvement will be to simulate the process of energy captured from the sun and store it. To have solar power which makes the energy and transfer it to the *Rolling Mill* and *Metal Press* in order to make them functional. The surplus energy to be stored into a generator and when is night and the player wants to power up the machines to have energy from the generator.

Bibliography

[1] "How obfuscation works in software development." [Online]. Available: https://medium.com/@hendurhance/how-obfuscation-works-in-software-development-7f52edfff520

[2] G. Wroblewski, "General method of program code obfuscation," 2002. [Online]. Available: https://logic.pdmi.ras.ru/~yura/of/w2002.pdf

[3] "Deobfuscation." [Online]. Available: https://promon.io/resources/security-software-glossary/deobfuscation

[4] D. L. Christian Collberg, Clark Thomborson, "A taxonomy of obfuscating transformations," 1997. [Online]. Available: https://www.researchgate.net/publication/37987523_A_Taxonomy_of_Obfuscating_Transformations

[5] D. Saito, A. Takebayashi, and T. Yamaura, "Minecraft-based preparatory training for software development project," in *2014 IEEE International Professional Communication Conference (IPCC)*, 2014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7020393

[6] M. Meccawy, "Learning while playing: Introducing programming concepts to children in minecraft," 2022. [Online]. Available: https://www.academia.edu/99300265/Learning_While_Playing_Introducing_Programming_Concepts_to_Children_in_Minecraft

[7] M. Lee, "The development of instruction model for sw education using the minecraft platform," in *Journal of Korea Society of Digital Industry and Information Management 15, no. 3*, 2019. [Online]. Available: https://koreascience.kr/article/JAKO201928463079633.page

[8] G.-J. Steinbeiss, "Minecraft as a learning and teaching tool : designing integrated game experiences for formal and informal learning activities," 2017. [Online]. Available: https://oulurepo.oulu.fi/handle/10024/8221

[9] G. D. Amri, "Benchmarking the performance impact of mods on minecraft-like games," 2024. [Online]. Available: https://jdonkervliet.com/assets/pdf/students/202408-bsc-thesis-guivari-amri.pdf

[10] J. D. Bayliss, "Teaching game ai through minecraft mods," in *2012 IEEE International Games Innovation Conference*, 2012, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6329841

[11] P. A. Brett Wilkinson, Neville Williams, "Improving student understanding, application and synthesis of computer programming concepts with minecraft," 2013. [Online]. Available: https://papers.iafor.org/wp-content/uploads/papers/ectc2013/ECTC2013_0477.pdf

[12] *Collaborative Design Principles From Minecraft With Applications to Multi-User CAD*, ser. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, vol. Volume 1B: 34th Computers and Information in Engineering Conference, 08 2014. [Online]. Available: https://doi.org/10.1115/DETC2014-35279

[13] "Forge, neoforge, and fabric. which should you use?" [Online]. Available: https://madelinemiller.dev/blog/forge-vs-fabric/

[14] A. G. . A. Grupta, "Minecraft modding with forge," 2015. [Online]. Available: https://books.google.ro/books?hl=en&lr=&id=2e3gBwAAQBAJ&oi=fnd&pg=PP1&dq=minecraft+mod+development&ots=K1FlTfCbJc&sig=4JGLc-7ZRpz3q8OW9P-PF7fRV6Y&redir_esc=y#v=onepage&q=minecraft%20mod%20development&f=false

[15] "Minecraft fabric vs forge: Why choose fabric." [Online]. Available: https://sudapeople.tv/1006

[16] "Why is fabric mod development faster than forge?" [Online]. Available: https://jangro.com/2024/06/21/why-is-fabric-mod-development-faster-than-forge/

[17] "Mekanism." [Online]. Available: https://wiki.aidancbrady.com/wiki/Mekanism

[18] "Gregtech." [Online]. Available: https://mods-of-minecraft.fandom.com/wiki/GregTech#Recipe_Overrides

[19] "Blocks." [Online]. Available: https://minecraft.wiki/Block

[20] "Block entities." [Online]. Available: https://minecraft.wiki/Block_entity

[21] "Items." [Online]. Available: https://minecraft.wiki/Item

[22] "Tools." [Online]. Available: https://minecraft.fandom.com/wiki/Tool

Appendix A. Relevant Code sections

```
1
2 public class ModBlocks {
3
4     public static final Block METAL_PRESS_BLOCK = registerBlock("
      metal_press",
5          new MetalPressBlock(FabricBlockSettings.copyOf(Blocks
           .IRON_BLOCK).nonOpaque()));
6
7
8     public static void registerModBlock() {
9         CraftMastery.LOGGER.info("Registering ModBlocks for " +
           CraftMastery.MOD_ID);
10    }
11
12    private static Item registerBlockItem(String name, Block
      block) {
13        return Registry.register(Registries.ITEM, new Identifier(
           CraftMastery.MOD_ID, name),
14             new BlockItem(block, new FabricItemSettings()));
15    }
16
17    private static Block registerBlock(String name, Block block)
      {
18        registerBlockItem(name, block);
19        return Registry.register(Registries.BLOCK, new Identifier
           (CraftMastery.MOD_ID, name),block);
20    }
21 }
```

Listing A.1: ModBlock class

```
1
2 if(isTreeBlock(currentBlockState)) {
3     stackPosition.push(new Pair<>(currentBlockPos.up(), world.
      getBlockState(currentBlockPos.up())));
4     stackPosition.push(new Pair<>(currentBlockPos.down(), world.
      getBlockState(currentBlockPos.down())));
5     stackPosition.push(new Pair<>(currentBlockPos.north(), world.
      getBlockState(currentBlockPos.north())));
6     stackPosition.push(new Pair<>(currentBlockPos.south(), world.
      getBlockState(currentBlockPos.south())));
7     stackPosition.push(new Pair<>(currentBlockPos.west(), world.
```

```
        getBlockState(currentBlockPos.west()));
    stackPosition.push(new Pair<>(currentBlockPos.east(), world.
        getBlockState(currentBlockPos.east())));


    for(int i=0;i<2;i++) {
        for(int z=0;z<2;z++) {
            stackPosition.push(new Pair<>(currentBlockPos.add(di[
                i],0,dz[z]), world.getBlockState(currentBlockPos.
                add(di[i],0,dz[z]))));
        }
    }
}
```

Listing A.2: Neighbours of a block

```
{
  "type": "minecraft:crafting_shaped",
  "category": "misc",
  "pattern": [
    "## ",
    "#$ ",
    " $ "
  ],
  "key": {
    "#": {
      "item": "craftmastery:tool_plate"
    },
    "$": {
      "item": "craftmastery:tool_rod"
    }
  },
  "result": {
    "item": "craftmastery:lumber_axe"
  }
}
```

Listing A.3: Lumber Axe recipe

```
if(pitch < -45 || pitch > 45){
    di = new int[] {-1,0,1};
    dj = new int[] {0,0,0};
    dz = new int[] {-1,0,1};
} else if(direction == Direction.NORTH || direction == Direction.
    SOUTH) {
    di = new int[] {-1,0,1};
    dj = new int[] {-1,0,1};
    dz = new int[] {0,0,0};
} else if(direction == Direction.EAST || direction == Direction.
    WEST){
    di = new int[] {0,0,0};
    dj = new int[] {-1,0,1};
```

```
12      dz = new int[] {-1,0,1};
13  } else {
14      di = new int[] {0,0,0};
15      dj = new int[] {0,0,0};
16      dz = new int[] {0,0,0};
17      }
```

Listing A.4: Neighbours of a block based on its direction

```
{
  "type": "minecraft:crafting_shaped",
  "category": "misc",
  "pattern": [
    "###",
    "###",
    " $ "
  ],
  "key": {
    "#": {
      "item": "craftmastery:tool_plate"
    },
    "$": {
      "item": "craftmastery:tool_rod"
    }
  },
  "result": {
    "item": "craftmastery:mining_hammer"
  }
}
```

Listing A.5: Mining Hammer recipe

```
{
  "type": "minecraft:crafting_shaped",
  "category": "misc",
  "pattern": [
    "###",
    " $ ",
    " $ "
  ],
  "key": {
    "#": {
      "item": "craftmastery:tool_plate"
    },
    "$": {
      "item": "craftmastery:tool_rod"
    }
  },
  "result": {
    "item": "craftmastery:mining_pickaxe"
  }
}
```

Listing A.6: Mining Pickaxe recipe

```
{
  "type": "minecraft:crafting_shaped",
  "category": "misc",
  "pattern": [
    "# #",
    "$!$",
    "$$$"
  ],
  "key": {
    "#": {
      "item": "craftmastery:tool_plate"
    },
    "$": {
      "item": "craftmastery:tool_rod"
    },
    "!": {
      "item": "minecraft:ender_pearl"
    }
  },
  "result": {
    "item": "craftmastery:magnet"
  }
}
```

Listing A.7: Magnet recipe

```
public void tick(World world, BlockPos pos, BlockState state) {
        if(!world.isClient()) {
            if(isOutputSlotEmptyOrReceivable()) {
                if(this.hasRecipe()) {
                    if(this.hasEnoughFuel()) {
                        this.increaseCraftProgress();
                        markDirty(world, pos, state);

                        if(hasCraftingFinished()) {
                            this.craftItem();
                            this.consumeFuel();
                            this.resetProgress();
                        }
                    } else {
                        this.resetProgress();
                    }
                } else {
                    this.resetProgress();
                }
            } else {
                this.resetProgress();
                markDirty(world, pos, state);
            }
```

```
        }
    }
```

Listing A.8: tick() method

```
public MetalPressScreenHandler(int syncId, PlayerInventory
   playerInventory,
                                   BlockEntity blockEntity,
                                      PropertyDelegate
                                      arrayPropertyDelegate) {
        super(ModScreenHandlers.METAL_PRESS_SCREEN_HANDLER,
           syncId);
        checkSize(((Inventory) blockEntity), 2);
        this.inventory = ((Inventory) blockEntity);
        inventory.onOpen(playerInventory.player);
        this.propertyDelegate = arrayPropertyDelegate;
        this.blockEntity = ((MetalPressBlockEntity) blockEntity);

        this.addSlot(new Slot(inventory, 0, 80, 11));
        this.addSlot(new Slot(inventory, 1, 97, 33));
        this.addSlot(new Slot(inventory, 2, 80, 59));


        addPlayerInventory(playerInventory);
        addPlayerHotbar(playerInventory);

        addProperties(arrayPropertyDelegate);
    }

    public boolean isCrafting() {
        return propertyDelegate.get(0) > 0;
    }

    public int getScaledProgress() {
        int progress = this.propertyDelegate.get(0);
        int maxProgress = this.propertyDelegate.get(1);
        int progressArrowSize = 26;

        return maxProgress != 0 && progress != 0 ? progress *
           progressArrowSize / maxProgress : 0;
    }

    @Override
    public ItemStack quickMove(PlayerEntity player, int invSlot)
       {
        ItemStack newStack = ItemStack.EMPTY;
        Slot slot = this.slots.get(invSlot);
        if (slot != null && slot.hasStack()) {
            ItemStack originalStack = slot.getStack();
            newStack = originalStack.copy();
            if (invSlot < this.inventory.size()) {
                if (!this.insertItem(originalStack, this.
```

```
                    inventory.size(), this.slots.size(), true)) {
42                      return ItemStack.EMPTY;
43                  }
44              } else if (!this.insertItem(originalStack, 0, this.
                    inventory.size(), false)) {
45                  return ItemStack.EMPTY;
46              }
47
48              if (originalStack.isEmpty()) {
49                  slot.setStack(ItemStack.EMPTY);
50              } else {
51                  slot.markDirty();
52              }
53          }
54
55          return newStack;
56      }
57
58      @Override
59      public boolean canUse(PlayerEntity player) {
60          return this.inventory.canPlayerUse(player);
61      }
62
63      private void addPlayerInventory(PlayerInventory
            playerInventory) {
64          for (int i = 0; i < 3; ++i) {
65              for (int l = 0; l < 9; ++l) {
66                  this.addSlot(new Slot(playerInventory, l + i * 9
                        + 9, 8 + l * 18, 84 + i * 18));
67              }
68          }
69      }
70
71      private void addPlayerHotbar(PlayerInventory playerInventory)
            {
72          for (int i = 0; i < 9; ++i) {
73              this.addSlot(new Slot(playerInventory, i, 8 + i * 18,
                    142));
74          }
75      }
```

Listing A.9: MetalPressScreenHandler class

Appendix B. Figures

Figure B.1: CraftMastery menu from creative mode