CAMP MINECRAFT MODS

– Tag 2 –

Kapitel 1 – Minecraft, jetzt aber wirklich!

Einen wunderschönen Guten Morgen am 2. Tag von eurem Camp Minecraft Mods! Heute schauen wir uns aber wirklich mal Minecraft an, und all die Sachen, die ihr dafür so braucht. Das bedeutet aber auch, dass wir erst einmal die wichtigen Funktionen durchgehen müssen – und ein paar **Spielregeln** festlegen. Wir würden euch sehr darum bitten, die **Vorlage** sorgfältig zu behandeln und aufmerksam zu *lesen*, damit ihr keine großen Fehler verursacht – die schaden euch nämlich am meisten, weil ihr dann weniger Zeit zum Arbeiten mit der Mod habt. Ein paar **Hinweise** also vorab. In der Vorlage findet ihr immer wieder so graue Zeilen Code, vor denen dann „//“ diese zwei Striche sind. Diese Zeilen Code sind sogenannte **Kommentare**, die euch zum Verständnis dazugeschrieben sind. Bitte *lest* diese in der jeweiligen Klasse genau durch, damit ihr wisst, wozu die Klasse gedacht ist, und was ihr da machen sollt. Darüber hinaus haltet ihr euch bitte an die Vorgaben dieser Doku – also verändert ihr keine Klassen oder keinen Code in der Vorlage, wenn euch das nicht in dieser Doku ausdrücklich gesagt wird.

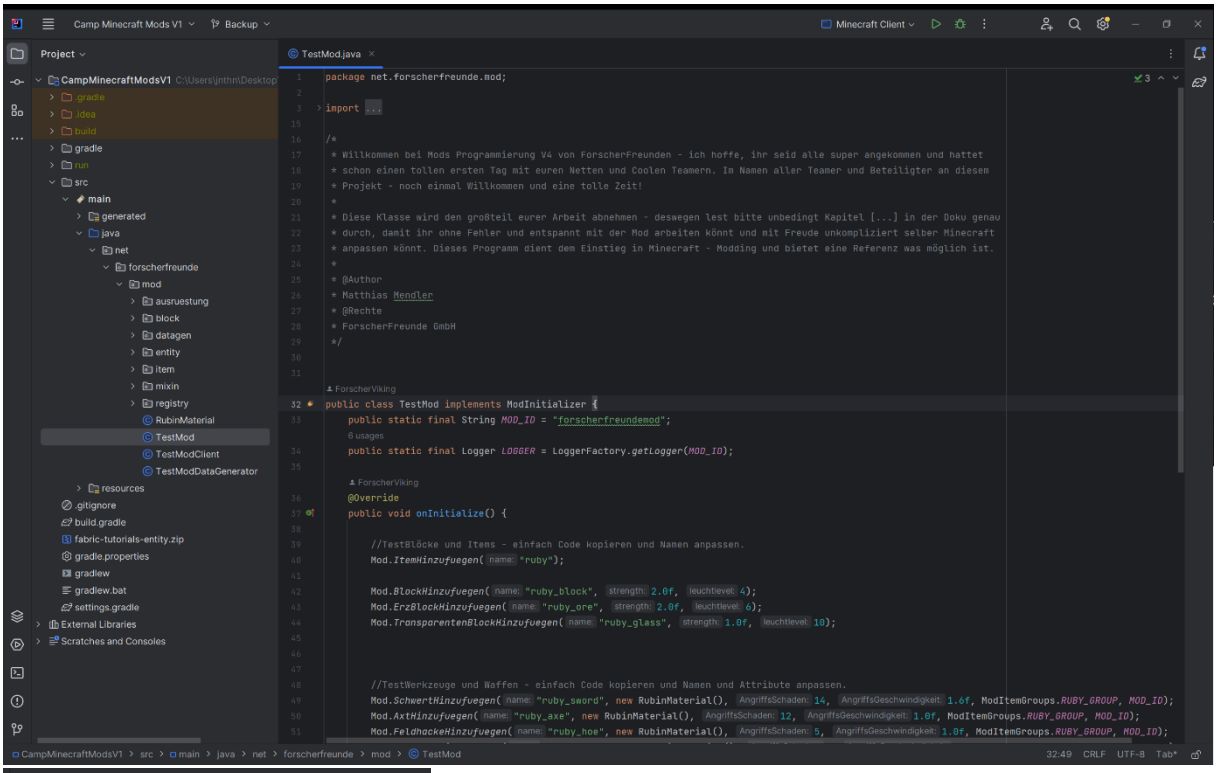
Es gibt eine Menge Klassen in der Mod, in denen ihr nichts zu tun habt – bzw. wodurch ihr das gesamte Programm kaputt machen könnt, wenn ihr da irgendwas dran ändert. Also Vorsicht!

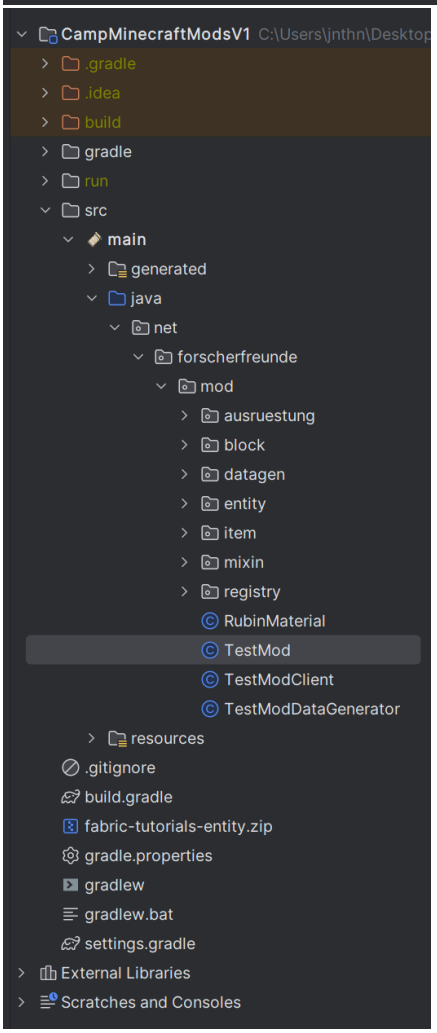
```
/*
* Kommentare können auch als Blockkommentar erstellt sein, vor allem am Anfang jeder Klasse – also
* auch diese Kommentare bitte sorgfältig lesen. In denen Steht meistens eine kurze Einführung in die
* Klasse – oder aber auch drin, ob ihr in der Klasse überhaupt etwas machen sollt oder nicht.
*/
```

Kapitel 1.1 – Die TestMod Klasse

Um unseren Code – und damit auch am Ende die Mod in Minecraft zu bekommen, benutzen wir ein Programm, welches und genau diese Arbeit abnimmt. Dieses Programm nennt sich *Fabric.* Wir brauchen aber natürlich auch sonst eine Menge weiterer Strukturen und Funktionalitäten, damit unser Programm überhaupt als Mod für Minecraft erkannt wird. Schauen wir uns also einmal die wohl wichtigste Klasse für die Mod an – die Klasse *TestMod.*

Wenn ihr die Datei *Camp Minecraft Mods V1* geöffnet habt, solltet ihr folgendes Fenster vorfinden:





Schauen wir uns die einzelnen interessanten Punkte, die wir hier finden doch einmal gemeinsam an. Zuallererst finden wir auf der linken Seite, wie auch zuvor in den Programmierbasics, unsere Ordnerstruktur, mit allen wichtigen Dateien, Klassen und externen Bibliotheken. Hier finden wir vor allem die Klassen *TestMod* und *TestModClient.* Diese beiden Klassen werden uns den Großteil der Arbeit abnehmen in Bezug auf die Handhabung der Mod. Wie bereits erwähnt wollen wir uns einmal zuerst die Klasse *TestMod* anschauen. Diese sollte auch bereits offen sein, wenn ihr das Programm startet. Falls nicht – einfach Doppelklick auf die Klasse in der Ordnerstruktur machen.

Die Klasse begrüßt euch dann zuerst mit einem *Blockkommentar* zu Beginn, lest diesen bitte auch einmal gründlich durch. In dem Kommentar stehen zwar keine wichtigen Infos zur Verwendung der Klasse, trotzdem ist es wie gesagt wichtig, dass ihr die Kommentare lest, und euch an mögliche Anweisungen in den Kommentaren haltet.

Schauen wir uns also einmal an, was die einzelnen Zeilen in der *TestMod*-Klasse so machen.

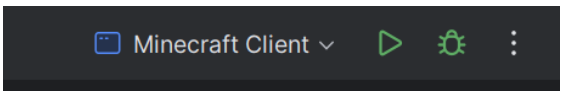
```java
/*
 * Willkommen bei Mods Programmierung V4 von ForscherFreunden - ich hoffe, ihr seid alle super angekommen und hattet
 * schon einen tollen ersten Tag mit euren Netten und Coolen Teamern. Im Namen aller Teamer und Beteiligter an diesem
 * Projekt - noch einmal Willkommen und eine tolle Zeit!
 *
 * Diese Klasse wird den großteil eurer Arbeit abnehmen - deswegen lest bitte unbedingt Kapitel [...] in der Doku genau
 * durch, damit ihr ohne Fehler und entspannt mit der Mod arbeiten könnt und mit Freude unkompliziert selber Minecraft
 * anpassen könnt. Dieses Programm dient dem Einstieg in Minecraft - Modding und bietet eine Referenz was möglich ist.
 *
 * @Author
 * Matthias Mendler
 * @Rechte
 * ForscherFreunde GmbH
 */


ForscherViking
public class TestMod implements ModInitializer {
    public static final String MOD_ID = "forscherfreundemod";
    6 usages
    public static final Logger LOGGER = LoggerFactory.getLogger(MOD_ID);

    ForscherViking
    @Override
    public void onInitialize() {

        //TestBlöcke und Items - einfach Code kopieren und Namen anpassen.
        Mod.ItemHinzufuegen( name: "ruby");

        Mod.BlockHinzufuegen( name: "ruby_block", strength: 2.0f, leuchtlevel: 4);
        Mod.ErzBlockHinzufuegen( name: "ruby_ore", strength: 2.0f, leuchtlevel: 6);
        Mod.TransparentenBlockHinzufuegen( name: "ruby_glass", strength: 1.0f, leuchtlevel: 10);

```

Wir finden zuerst den Beginn der Klasse in Zeile 32 – dort finden wir *public class* ***TestMod*** – also unser Kennzeichen für eine Klasse. Das *implements* dahinter müssen wir nicht beachten, alles, was das macht, ist, dass diese Klasse automatisch ausgeführt wird, wenn wir den grünen Starknopf oben rechts drücken – bzw. sobald sich Minecraft öffnet.

Wir sehen nun – wie wir gelernt haben – zwei Attribute der Klasse. Zuerst ein *String* Attribut, mit dem Namen **MOD_ID,** und dann ein Attribut des Typen *Logger.* Was das ist – müssen wir erneut nicht wissen. Das Attribut **MOD_ID** ist dazu da, damit alle unsere Items, Blöcke und Kreaturen, die wir erschaffen, auch tatsächlich in der richtigen Datenbank in Minecraft zu finden sind. Dieses Attribut **verändert ihr bitte nicht**!!

Danach finden wir unsere erste – und wohlbemerkt einzige – Funktion der Klasse vor. Nämlich die Funktion *onInitialize().* Im Englischen bedeutet das so viel wie *beimStarten().* Wie bereits weiter oben erwähnt ist das Ziel der Klasse, dass unser Code in Minecraft bei dem Start des Spiels auch umgesetzt wird. Diese Funktion macht genau das. Hier finden wir auch bereits eine Menge Zeilen Code, die uns erst mal noch nicht zu interessieren haben. Diese Zeilen sind für die Beispielitems und Blöcke da, die ihr in game bereits findet – und dienen euch als Vorlage. Also diese Zeilen Code bitte nicht verändern, sondern immer nur kopieren und neue Zeilen im Code hinzufügen, sonst verursacht ihr eine Menge Fehler im Code.

Schauen wir uns vorab doch einmal die bisherige Mod Vorlage im Spiel an. Drückt dafür, wie zuvor auch in den Programmierbasics, einfach den grünen *Play* Knopf. Zuvor müssen wir allerdings einmal schauen, dass der Knopf genauso aussieht:



Falls bei euch daneben nicht *Minecraft Client* steht, sagt einmal eurem Teamer Bescheid, damit dieser sich das einmal anschauen kann. Alternativ schaut einmal, ob, wenn ihr auf den kleinen Pfeil nach unten daneben drückt, ihr in dieser Liste die Konfiguration *Minecraft Client* findet. Dann schaut euch einmal im Singleplayer die Welt an.

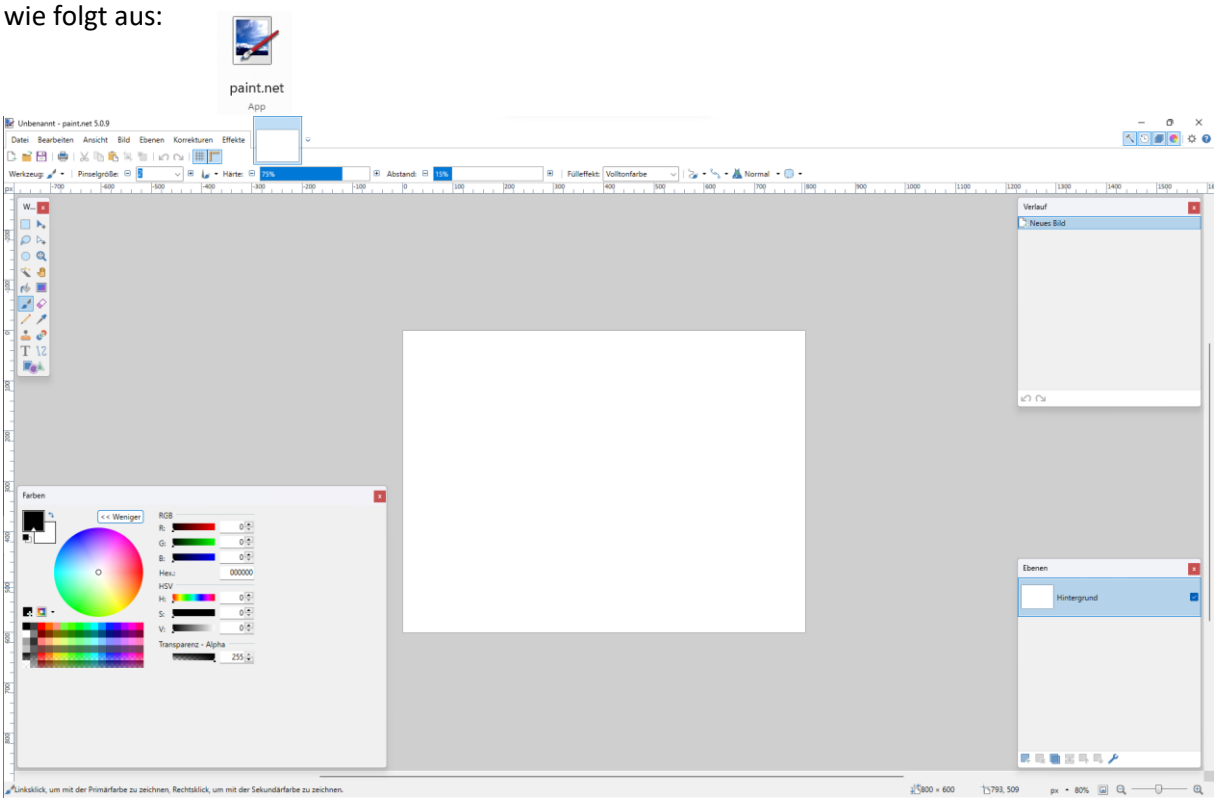
Kapitel 2 – Die ersten eigenen Schritte

Fangen wir also mit den ersten eigenen Sachen für unsere Mod an. Zuerst schauen wir uns einfache Items an, und wie diese hinzugefügt werden können. Aber natürlich brauchen wir dafür zuerst mal etwas ganz Entscheidendes. Eine Sache darf nämlich auf keinen Fall in einem Spiel fehlen – die Texturen, also das Aussehen im Spiel selbst. Denn alles, was wir im Code machen, ist Text schreiben. Aber woher soll das Spiel wissen, wie unser Text bitte als Objekt in Minecraft aussehen soll?

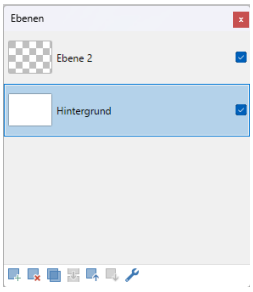
Starten wir nun mit unseren ersten eigenen Items. Dafür brauchen wir…

Kapitel 2.1 – …eine eigene Textur.

Dafür öffnen wir einmal das Programm *Paint.net*, welches ihr auf dem Desktop finden könnt. Es sieht wie folgt aus:



Nun müssen wir ein paar Änderungen vornehmen, um anfangen zu können. Zuerst wollen wir einmal eine neue Ebene erzeugen, auf der wir arbeiten können. Dafür schauen wir uns unten rechts einmal das Fenster an.



Hier klicken wir einmal auf die Funktion ganz links unten – und erstellen somit eine neue Ebene. Wir wählen dann *Ebene1* aus und klicken auf die zweite Funktion von links unten, und löschen die Ebene damit. Nun sollten wir nur eine transparente Zeichenebene haben, auf der wir zeichnen können.

Alle unsere Texturen in Minecraft haben eine bestimmte Größe, die wir natürlich auch berücksichtigen müssen. Dafür gehen wir einmal auf oben unter *Bild -> Größe* ändern (alternativ drückt ihr *ctrl + R*), und wählt als Größe 16 x 16 Pixel aus. Also für Höhe 16 Pixel und für Breite 16 Pixel. Nun müssen wir vermutlich etwas ran zoomen,

da die Zeichenfläche jetzt ziemlich klein sein sollte. Dafür einfach *ctrl* gedrückt halten, und das Mausrad drehen (alternativ könnt ihr unter *Ansicht -> Vergrößern* die Größe ändern).

Auf der linken Seite seht ihr nun eine Auswahl an Werkzeugen, mit denen ihr Arbeiten könnt.

Wir empfehlen das hier ausgewählte Werkzeug zum Pixeln der Textur zu verwenden, da ihr so einzelne Pixel auswählen und zeichnen könnt. Alternativ könnt ihr aber auch den Pinsel verwenden. Daneben das Werkzeug ist die Pipette – mit dieser könnt ihr eine Farbe auswählen auf dem Bildschirm, um genau diese Farbe in der Palette zu bekommen. Ganz schön nützlich, wenn man eine bestimmte Farbe verwenden möchte, die man anderswo bereits verwendet hat.

Abgesehen davon könnt ihr nun eurer Fantasie freien Lauf lassen – hier kommt eure Aufgabe:

entwerft eine Textur für ein Item, aus dem wir nachher einen Block machen können.

Die Idee ist, dass das Item – ähnlich wie die Vorlage (der Rubin),  ein Item sein sollte, welches in Minecraft also ähnlich wie ein Diamant ist.

Es sollte also ein Item sein, welches fallen gelassen werden kann, wenn man das zugehörige Erz abbaut, und es sollte einen festen Block aus diesem Item geben können – also auch wie der Diamantblock, der komplett aus Diamanten ist.

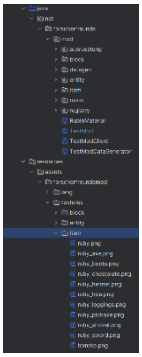
Die Blöcke – und auch die Erze – schauen wir uns in den kommenden Kapiteln an, also konzentriert euch erst mal auf ein schönes Item. Dieses Item ist der Grundstein für eure eigene Mod, also gebt euch beim Textur zeichnen viel Mühe und nehmt euch die Zeit, die ihr braucht, um etwas großartiges zu erstellen.

Wenn ihr damit fertig seid, wollen wir dieses Item nun….

Kapitel 2.2 – … auch in Minecraft sehen.

Dazu müsst ihr eure eben erstellte Textur in einem bestimmten Ordner abspeichern. Dafür geht ihr unter *Datei -> Speichern unter…* und sucht dann auf dem Desktop nach dem Ordner CampMinecraftModsV1, geht in diesem dann in den Ordner src -> main -> resources -> assets -> forscherfreundemod -> textures -> item und speichert dort euer Item als *name.png* ab. Wichtig hierbei: der Name eures Items sollte bitte kleingeschrieben sein, ohne Leerzeichen, und am Ende muss die Endung *.png* dahinterstehen. Den Namen werden wir immer wieder benötigen, also merkt euch die Schreibweise des Item namens ganz genau – sehr Wichtig!

Nun wechseln wir zurück zu **IntelliJ,** und schauen uns einmal alle wichtigen Schritte an, um das Item auch im Spiel sehen zu können. Dafür gehen wir zuallererst einmal in der Ordnerstruktur auf resources -> assets -> forscherfreundemod -> textures -> item:



Hier müssten wir jetzt unsere eben erstellte Textur Datei sehen können, ihr könnt diese einmal doppelklicken und euch anschauen

Nun gehen wir wieder in die Klasse *TestMod* zurück, und schauen uns die Zeile 40 im Code an. Hier haben wir die Vorlage, wie das Item *Ruby* integriert ist. Diese Zeile Code könnt ihr jetzt einfach **KOPIEREN**, nicht **VERÄNDERN**, indem ihr die Zeile mit der Maus auswählt, wichtig: die ganze Zeile Code, also auch das Semikolon (;) am Ende, und dann *ctrl + C* drückt, und dann hinter das Semikolon der Zeile einmal Enter drückt, und in der neuen Zeile dann *ctrl + V* drückt. Nun sollten dort zwei Mal die gleiche Zeile Code stehen.

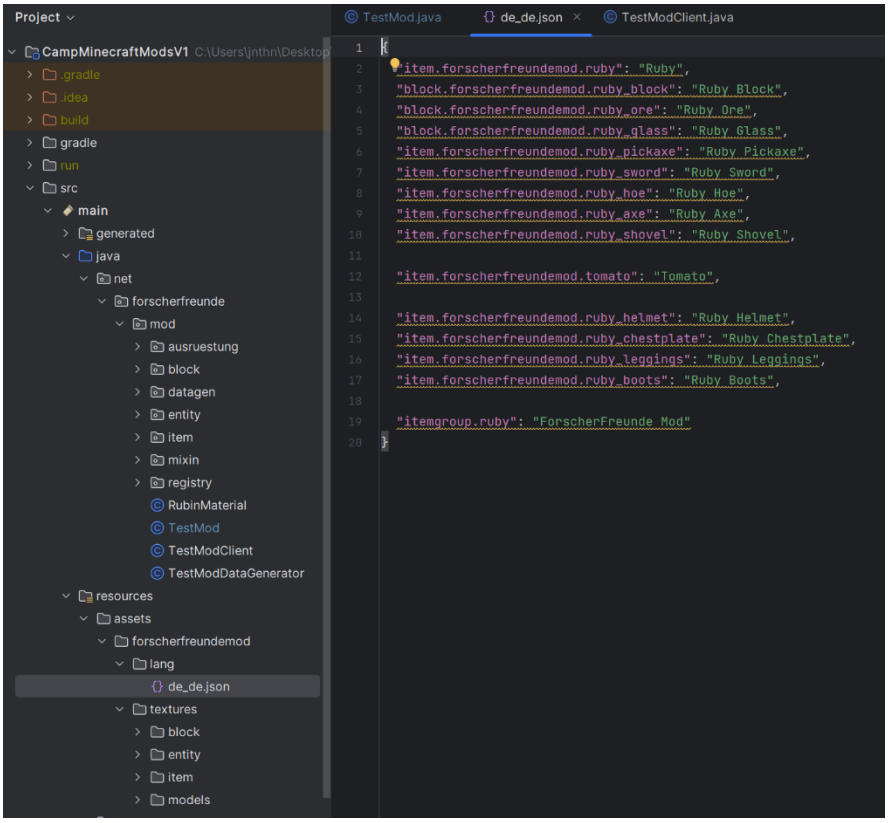
Nun müsst ihr nur noch den Namen des Items anpassen: also statt „ruby“ innerhalb der Klammern, euren Namen der Textur Datei – **WICHTIG: GENAU GLEICH GESCHRIEBEN, ALSO ALLES KLEINBUCHSTABEN UND KEINE LEERZEICHEN.**

Die Endung *.png* muss nicht mit übernommen werden – gehört auch nicht zum Namen, sondern gibt nur den Typen der Datei an, also eine Bilddatei.

Als nächsten Schritt brauchen wir noch eine Übersetzung in die deutsche Sprache, damit euer Item in Minecraft auch einen ordentlichen Namen hat, und nicht mit seiner Speicheradresse dargestellt wird. Dafür gehen wir in den Ordner resources -> assets -> forscherfreundemod -> lang, und finden dort die Datei *de_de.json.*

Kapitel 2.3 – .JSON und die Sprache

Wenn ihr die Datei öffnet, seht ihr viele komische Zeilen Code, die nicht so aussehen wie wir es bis jetzt kennengelernt haben. Diese Datei ist eine *JSON*-Datei, ein bestimmter Dateityp, welcher für wichtige Anpassungen in Programmstrukturen wichtig ist.



Dieser Dateityp beinhaltet keine Funktionen, Attribute, Methoden oder Klassen, sondern ist für die Sprache im Spiel zuständig. Die Datei sagt also Minecraft, wie es bestimmte Namen von unseren Items und neuen Blöcken darstellen soll – allerdings nur in der deutschen Sprache. Also solltet ihr in Minecraft in den Einstellungen die Sprache auf Deutsch einstellen.

Um jetzt den Namen von eurem neu erstellten Item in Minecraft anzugeben, müsst ihr erneut einfach die Zeile Code finden, in der der Rubin steht. Hier also Zeile 2. Diese Zeile kopiert ihr, und verändert sie an zwei stellen. Zum einen in dem Lila Teil des Codes – nämlich statt:

„item.forscherfreundemod.ruby“ -> „item.forscherfreundemod.itemname“.

Außerdem im grünen Bereich, wo ihr statt:

„Ruby“ -> „ItemName“ schreibt. Hier könnt ihr zum einzigen Mal auch Großschreibung und Leerzeichen verwenden. Der Grüne Teil ist der Name, wie das Item im Spiel dann benannt ist, und welcher angezeigt wird. Als nächstes bringen wir die Texturen ins Spiel.

Kapitel 2.4 – Models und eure Helfer, die Datagen-Klasse

Wir haben jetzt unsere Texturen im Ordner, jedoch muss das Spiel auch noch wissen, welche Textur zu welchem Item gehört. Dafür müssen wir eigentlich selber eine neue *.json* Datei erzeugen, und mit vielen Zeilen Code angeben, wie die Textur auszusehen hat im Spiel – also wie das sogenannte *Model* aussieht. Das haben wir euch etwas vereinfacht, da das nur Zeit in Anspruch nimmt, die wir lieber mit dem erstellen neuer cooler Items verbringen könnten. Also gehen wir einmal in den Ordner *mod -> datagen* und schauen uns die Klasse *ModModelsProvider* an. Diese Klasse nimmt euch die ganze Arbeit ab, alles was ihr tun müsst, ist innerhalb der Klasse angeben, welches Item welche Textur bekommen soll. Wir schreiben also in der Methode *generateItemModels(…)* in Zeile 28 hinter die Zeile vom Rubin eine Neue Zeile:

itemModelGenerator.register(ModItems.GetItem(„name_vom_item“), Models.GENERATED);

und sind dann auch fertig. Alternativ könnt ihr natürlich auch die Zeile vom Rubin kopieren und einfach den Namen anpassen, das geht deutlich schneller. Jetzt sollte im Spiel auch die passende Textur angezeigt werden, schauen wir uns das doch einmal an!

Jetzt könnt ihr Minecraft öffnen, und euer erstes eigenes Item im Spiel anschauen. Das Item findet ihr in dem Custom Inventar, wenn ihr auf den Pfeil nach rechts klickt, neben den ganzen Vorlagen Items. Sollte die Textur nicht ordentlich dargestellt sein, müsst ihr die *TestMod* Klasse noch einmal prüfen, dann habt ihr nicht genau den Namen der Textur Datei verwendet.

Sollte direkt ein Fehler das Spiel am Starten hindern – oder steht in der Kommandozeile:

Dann habt ihr eine Zeile Code doppelt in *TestMod* stehen… schaut also, dass ihr nicht zweimal die Zeile

Mod.ItemHinzufuegen(„ruby“); stehen habt.

Super! Wir haben unser erstes eigenes Item in Minecraft hinzugefügt.

Aufgabe:

Nehmt euch doch jetzt einmal ein bisschen Zeit und geht noch einmal die einzelnen wichtigen Punkte anhand eines neuen Items durch – das kann gerne wieder als Grundbaustein für einen Block später dienen. Die wichtigen Schritte sind:

1. Textur erstellen in Paint.net
2. Textur im Ordner: Desktop -> CampMinecraftModsV1 -> src -> main -> resources -> assets -> forscherfreundemod -> textures -> item abspeichern mit dem Namen in ***Kleinbuchstaben*** und ***ohne Leerzeichen***!
3. Die Zeile vom Rubin duplizieren (also kopieren und neu einfügen) in der TestMod Klasse, und den Namen anpassen -> wieder aufpassen, dass der Name richtig geschrieben ist.
4. In der Klasse *de_de.json* die Zeile Code vom Rubin kopieren, und sowohl die ***Adresse***, als auch den ***Namen*** ersetzen. Wieder auf die *richtige Schreibweise achten*! Den Namen könnt ihr dann in der de_de.json Klasse auch Groß und mit Leerzeichen schreiben.
5. Die Modeldatei erstellen lassen – also in der Klasse *ModModelsProvider* in dem Ordner *datagen* euer Item hinzufügen.
6. Starten über den grünen Pfeil (**Wichtig**, dass der Minecraft Client ausgewählt ist!), und im Inventar nachschauen!

Nun haben wir schon zwei eigene Items im Spiel! Super!

Als nächstes wollen wir uns einmal das wohl wichtigste in Minecraft anschauen – nämlich die Blöcke, für die wir die Items ja gemacht haben.

Also wird es Zeit für das nächste Kapitel, nämlich…

Kapitel 2.5 – … die Blöcke und das, was Minecraft ausmacht!

Nun haben wir uns schonmal **Items** angeschaut und das so weit verstanden, wie wir die ins Spiel kriegen. Zwischen den *Texturen* und den Zeilen *Code* ist es anscheinend wichtig genau zu arbeiten und auf die *Schreibweise* zu achten.

Mit den **Blöcken** verhält es sich sehr ähnlich – deshalb schauen wir uns das jetzt einmal an!

Zuerst schauen wir uns die **einfache** Variante für die Blöcke an – also ganz einfache Blöcke, die auf allen Seiten gleich aussehen.

Dafür öffnen wir nochmal *Paint.net* und jetzt zeichnet ihr bitte eine **Textur** für einen Block – wichtig, wieder in einem 16x16 pxl Fenster. Diesmal sollte die Textur auch die vollen 16x16 pxl ausfüllen (aber ihr könnt damit auch gerne etwas herumprobieren was euch so gefällt!), und dann speichern wir die Datei wieder mit *Speichern unter* ab … diesmal aber in dem Ordner … -> *textures* -> *block* als .png Datei. Bei dem **Namen**, genau wie bei dem **Item**, achtet ihr wieder auf **kleine** Buchstaben und *keine* Leerzeichen, und nennt den Block am besten zur besseren Kennzeichnung *name*_block.png (wobei *name* natürlich der Name, den ihr dem Block gebt, ist).

Wenn wir die **Textur** haben, mit der wir zufrieden sind, dann wollen wir einmal in die *TestMod* Klasse gehen, hier schauen wir uns die Zeile Code unter unserem eigenen neuen Item an, nämlich den **Rubin Block**.

Die Zeile sieht zwar ähnlich zu der Zeile Code vom Rubin aus, jedoch ist sie nicht ganz gleich. Wir sehen, dass der Rubin Block z.B. nicht nur einen *Namen* bekommen hat, sondern auch zwei weitere **Parameter**. Diese Parameter sind einmal die **Stärke** (engl.: **strength**, hier als *float* Wert angegeben, gekennzeichnet durch das *f* hinter der Zahl), und das **Leuchtlevel** (hier als ganze Zahl, also *int* angegeben. Diese beiden Parameter bestimmen, wie schnell der Block abgebaut werden kann, und wie sehr er Licht abgibt. Glowstone z.B. hat ein Leuchtlevel von 15, eine Fackel nur 14, Magmablöcke ein Leuchtlevel von 3 usw… Jeder Block in Minecraft hat ein Leuchtlevel, normale Blöcke wie Erde und Stein haben standardmäßig ein Leuchtlevel von 0.

Bei der **Stärke** sieht es etwas anders aus. Der Wert setzt sich aus zwei verschiedenen Parametern zusammen: *hardness* (engl. Für *Härte*) und *resistence* (engl. Für *Widerstand*). Diese Beiden Werte werden bei uns durch nur einen angegeben, nämlich Stärke (bedeutet bei uns hardness = resistence).

Ein Diamantblock z.B. hat eine Härte von 5f, ein Ofen eine Härte von 3.5f, ein Goldblock nur eine Härte von 3f, usw. Der Wert gibt an, wie schnell ein Block abgebaut, und wie resistent er gegenüber Explosionen ist. Cobblestone hat einen Wert von 2f, Stone nur einen von 1.5f.

Wenn wir unseren Block nun im Code einfügen wollen, machen wir hinter dem **Rubinblock** einfach eine neue Zeile, und schreiben:

Mod.BlockHinzufuegen(„name“, strengthvalue, leuchtlevel);

Dabei ersetzt ihr natürlich den **Namen** mit eurem Blocknamen, wieder kleingeschrieben und ohne Leerzeichen, so wie er abgespeichert ist, und die beiden Parameter für strength und das Leuchtlevel. Hier verwenden wir nun eine andere **Funktion** als bei dem Item. Wie ihr vielleicht festgestellt habt, war die Funktion beim Item:

Mod.ItemHinzufuegen(….);

Wir haben also verschiedene Funktionen, die wir uns in den kommenden Kapiteln noch genauer anschauen werden. Für das Item also *.ItemHinzufuegen,* für den Block eben *.BlockHinzufuegen.*

Ziemlich selbsterklärend würde ich sagen. Sonst könnt ihr euch aber auch einfach immer die Zeile der passenden Vorlage kopieren – also, wenn ihr einen Block machen wollt, den **Rubinblock**, bei einem Item den **Rubin** usw.…

Uns Fehlt nun noch die **Textur** – also das **Model** des Blocks im Spiel, dafür gehen wir wie zuvor beim Item in die Klasse *ModModelsProvider* und schreiben diesmal in der Funktion *generateBlockStateModels()* unter die Zeile vom *ruby_block* eine neue Zeile, kopieren die Zeile vom *ruby_block* und ändern dann den Namen zu eurem neuen Block ab. Das sollte dann in etwa so aussehen:

```
13      public ModModelsProvider(FabricDataOutput output) { super(output); }
16
        2 usages  ForscherViking *
17      @Override
18      public void generateBlockStateModels(BlockStateModelGenerator blockStateModelGenerator) {
19          //Blockstates und Models - siehe Kapitel 2.5 an Tag 2
20          blockStateModelGenerator.registerSimpleCubeAll(ModBlocks.GetBlock( name: "ruby_block"));
21          blockStateModelGenerator.registerSimpleCubeAll(ModBlocks.GetBlock( name: "euer_block"));
22          blockStateModelGenerator.registerSimpleCubeAll(ModBlocks.GetBlock( name: "ruby_ore"));
23          blockStateModelGenerator.registerSimpleCubeAll(ModBlocks.GetBlock( name: "ruby_glass"));
24
25      }
```

Nun haben wir unseren Block fertig im Spiel, brauchen wir nur noch den Namen in der de_de.json Datei anzupassen, das kennen wir ja bereits (und müssen wir auch bei allen unseren selbsterstellten Objekten machen!), und dann fehlt nur noch der letzte Schliff:

Unser Block kann aktuell von allem abgebaut werden – aber das wollen wir ja vielleicht gar nicht. Schauen wir uns also zuerst mal die Sache mit dem Abbauen genauer an. Die **Lösung** dafür finden wir in der Klasse *ModBlockTagProvider* in dem Ordner *datagen*. Hier schreiben wir in die Methode *configure(…)* in den jeweils passenden **Aufruf** noch unseren **Block** dazu. Also soll unser Block mit einer **Spitzhacke** abbaubar sein: in den Aufruf *getOrCreateTagBuilder(BlockTags.PICKAXE_MINEABLE)*

Und machen dann hinter die Klammer zu einfach eine neue Zeile und schreiben dort dann:

.add(ModBlocks.GetBlock(„name_eures_Blocks“))

Wichtig, danach mal kein Semikolon am Ende der Zeile, da der Aufruf dahinter noch weitergeht. Am Ende, bzw. ganz unten in dem Aufruf steht das **Semikolon** für euch schon.

Wollt ihr stattdessen, dass euer Block mit einer *Axt*, *Schaufel* oder *Hacke* abgebaut werden kann, sucht euch den dementsprechenden Aufruf raus und schreibt dort die Zeile Code dazu.

Weiter unten seht ihr dann die Aufrufe für das Abbau Level. Dort fügt ihr die Zeile Code von oben einfach genauso dort ein, wo euer Block hinsoll.

Die Abbaulevel sehen wie folgt aus:

NEEDS_STONE_TOOL: Stein Werkzeug oder höher.

NEEDS_IRON_TOOL: Eisen Werkzeuge oder höher.

NEEDS_DIAMOND_TOOL: Diamant Werkzeuge oder höher.

needs_tool_level_4: Netherrite Werkzeuge oder höher.

Diese Angaben beziehen sich immer auf die Drops, abgebaut kann der Block (auch wenn es lange dauert) mit allem.

Apropos Drops – da war doch noch was. Unsere Blöcke lassen aktuell gar nichts fallen, das können wir aber noch ändern. Normalerweise verwendet Minecraft dafür wieder .json Dateien, die sehr kompliziert aussehen – und auch sind. Wir haben das für euch vereinfacht:

Ihr geht in der Klasse *ModLootTableProvider* auf den Methodenaufruf *generate()* und schreibt euren Block einfach in die Liste dazu – also Kopiert ihr die Zeile vom *ruby_block* einfach, und ersetzt den Namen.

Kapitel 2.6 – Die letzte Aufgabe für heute…

…wäre jetzt, dass ihr einmal noch 2 weitere schöne Blöcke euch überlegt. Die von euch, die den Code etwas genauer betrachtet haben, haben eventuell festgestellt, dass wir noch zwei weitere Arten an Blöcken für euch vorbereitet haben:

Die **Erzblöcke** und **transparente** Blöcke.

Erzblöcke sind im Prinzip genau gleich zu den normalen Blöcken, der einzige Unterschied ist, dass die Erzblöcke auch noch exp. Punkte beim Abbauen fallen lassen. Die transparenten Blöcke – wie der Name schon sagt – sind transparent, eignen sich also super, um Glasblöcke zu machen. Dafür müsst ihr in Paint.net bei der Textur, einfach die transparenten Stellen transparent lassen. Sonst verhalten sich die Blöcke auch im Code sehr ähnlich – beim Erzeugen in der TestMod Klasse, müsst ihr nur die jeweilige Vorlage kopieren, also *ruby_ore* oder *ruby_glass.* Das gleiche gilt dann auch in der *ModModelsProvider* und in der *ModBlockTagProvider* Klasse. In der *ModLootTableProvider* Klasse hingegen, müsst ihr für die Erzblöcke einmal die Zeile vom *ruby_ore* kopieren, und dann zuerst den **Namen** eures Blockes ersetzen, dann noch einmal den **Namen** eures Blockes, und den **Dritten** grünen Namen ersetzt ihr mit dem **Item**, welches aus dem Erz *fallengelassen* werden soll – also das Item, welches ihr selbst zuvor erstellt habt.

Das Glas verhält sich dabei deutlich mehr, wie der normale *ruby_block,* alles, was ihr da extra machen müsst, ist den richtigen Aufruf in der *TestMod* vorzunehmen, also

Mod.TransparentenBlockHinzufuegen(), mit dem **Namen** des Blocks, der **Stärke** und dem **Leuchtlevel**.

Der Block wird dann automatisch vom Programm als **transparent** eingestuft, und dementsprechend vom Spiel gewertet.

Wenn ihr zwei weitere schöne Blöcke zusammen habt, sollte der zweite Tag doch gut gefüllt sein!

Morgen wollen wir den Bauwettbewerb vorbereiten.

Ihr werdet zu Beginn von Tag 3 noch einmal Zeit haben, eure Blöcke final fertig zu bekommen, für alle die noch schnell einen neuen Block machen wollen, oder die nicht fertig geworden sind gestern.

Dann wollen wir einen Bauwettbewerb machen. Also baut ihr alle innerhalb von ca. 30 min +-, eine schöne Struktur, ähnlich zu dem Turm in der Testwelt. Nutzt dabei vor allem eure eigenerschaffenen Blöcke. Am Ende der Zeit zeigt ihr euch gegenseitig eure Bauwerke, und stimmt ab, wer das kreativste Bauwerk gebaut hat, und wer die coolsten Blöcke gemacht hat.

FÜR DIE SCHNELLEN - LOOTTABLES

Wir schauen uns jetzt einmal Loottables etwas genauer an. Ein Loottable sieht als .json Datei so aus:

```json
{
  "type": "minecraft:block",
  "pools": [
    {
      "bonus_rolls": 0.0,
      "conditions": [
        {
          "condition": "minecraft:survives_explosion"
        }
      ],
      "entries": [
        {
          "type": "minecraft:item",
          "name": "forscherfreundemod:ruby_block"
        }
      ],
      "rolls": 1.0
    }
  ]
}
```

Hier sehen wir also eine Menge Klammern, viele komisch eingerückte Zeilen, und sonst eine typische .json Datei. Diese startet mit dem Type vom Loottable – hier also für einen Minecraft Block.

Die nächste Zeile gibt an, aus welchen „pools“, also aus welchen Auswahlmöglichkeiten, ausgewählt werden kann. Diese werden dann durch die eckigen Klammern definiert. Innerhalb dieser Auswahlmöglichkeiten haben wir erst mal angegeben, dass es keine Bonus Rolls geben wird, also wird nur so oft ausgewählt, wie unten unter *rolls* angegeben ist (hier: 1.0).

Die *condition* gibt an, ob es besondere Zustände der Blöcke gibt, die beachtet werden sollen. Hier steht bei unserem Block drin, dass er nur etwas fallen lassen soll, wenn er nicht durch eine Explosion zerstört wurde. Ansonsten lässt er nichts fallen. Darunter, in dem Feld *entries* (wieder durch die Eckigen Klammern markiert), finden wir nun die Items, die fallengelassen werden können (hier: Rubin Block).

Darunter, wie bereits erwähnt die Anzahl, wie oft diese Liste durchgegangen wird, bzw. wie oft eines dieser Items ausgewählt wird zum fallen lassen (hier: 1.0).

Das ist also der Standardaufbau eines Loottables, er sagt uns welche Bedingungen zu erfüllen sind, ob bestimmte Verzauberung gebraucht werden, und was genau der Block überhaupt fallen lassen kann. Außerdem bestimmt er auch, wie viele Items ein Block fallen lässt. Das hier ist ein wirklich simpler Loottable, mit wenigen Bedingungen und wenigen Items zum droppen. Glaubt ihr mir nicht? Schauen wir uns doch einmal den Loottable von Diamantenerz an:

```
{
  "type": "minecraft:block",
  "pools": [
    {
      "bonus_rolls": 0.0,
      "entries": [
        {
          "type": "minecraft:alternatives",
          "children": [
            {
              "type": "minecraft:item",
              "conditions": [
                {
                  "condition": "minecraft:match_tool",
                  "predicate": {
                    "enchantments": [
                      {
                        "enchantment": "minecraft:silk_touch",
                        "levels": {
                          "min": 1
                        }
                      }
                    ]
                  }
                }
              ],
              "name": "minecraft:diamond_ore"
            },
            {
              "type": "minecraft:item",
              "functions": [
                {
                  "enchantment": "minecraft:fortune",
                  "formula": "minecraft:ore_drops",
                  "function": "minecraft:apply_bonus"
                },
                {
                  "function": "minecraft:explosion_decay"
                }
              ],
              "name": "minecraft:diamond"
            }
          ]
        }
      ],
      "rolls": 1.0
    }
  ],
  "random_sequence": "minecraft:blocks/diamond_ore"
}
```

Das sieht schon direkt viel komplexer aus – oder nicht? Versucht den Loottable doch einmal nachzuvollziehen. Was machen die verschiedenen Zeilen? – und damit bis Morgen!