Westfälische Wilhelms-Universität Münster

Bachelorarbeit

Einführung in die Objektorientierung anhand einer Stellpultsimulation im Kontext Eisenbahn

Introduction to object-orientation based on a control panel simulation in context of railways

vorgelegt von: Niklas Nogossek

Studiengang: 2-Fach Bachelor Informatik/Mathematik

SS 2022

Betreuer: Professor Dr. Marco Thomas

Datum: 09. September 2022

Inhaltsverzeichnis

1 Einleitung

> Das Lernen in Kontexten, die durch Lehrkräfte vor Ort festgelegt werden, ist verbindlich. Lernen in Kontexten bedeutet, dass Fragestellungen aus der Lebenswelt der Schülerinnen und Schüler sowie gesellschaftliche und technische Fragestellungen den Rahmen für Unterricht und Lernprozesse bilden. Dafür geeignete Kontexte beschreiben reale Situationen mit authentischen Problemen, deren Relevanz gleichermaßen für Schülerinnen und Schüler erkennbar ist und die mit den zu erwerbenden Kompetenzen gelöst werden können. (*Kernlehrplan Biologie*, 2019, S. 14)

Diesen Absatz findet man nicht nur im Kernlehrplan für Biologie, sondern ebenfalls im Kernlehrplan für Physik und Chemie. Für den Informatikunterricht sucht man im Kernlehrplan eine so deutliche Forderung nach Kontextorientierung allerdings vergeblich. Das Projekt Informatik im Kontext, das im Jahr 2008 gestartet (Diethelm et al., 2011, S. 97) und auf der INFOS 2009 vorgestellt wurde (Koubek et al., 2009), formulierte erste Unterrichtsentwürfe und Konzepte für einen kontextorientierten Informatikunterricht.
Das Projekt „LoCo - Eisenbahn als informatischer Kontext - locomotive control“ knüpft an diese Konzepte an. Die Eisenbahn soll als strukturreicher Kontext dienen und die Möglichkeit bieten, viele Unterrichtsbausteine dem Kontext hinzuzufügen (Thomas, 2021, S. 6/7). Im Folgenden soll untersucht werden, warum sich das digitale Gleisbildstellpult im Kontext Eisenbahn als Baustein für die Einführung in die Objektorientierung anbietet.

1.1 Kontextorientierter Unterricht - Informatik im Kontext

Bei der Vorstellung des Projekts Informatik im Kontext (IniK) auf der INFOS 2009 wurde die „Kluft zwischen fachlichen Lernzielen, der Strukturierung des Unterrichts und den Interessen, Erwartungen und der Motivation der Schülerinnen und Schüler“ (Koubek et al., 2009, S. 269), in den MINT-Fächern (Mathematik, Informatik, Naturwissenschaften und Technik) beanstandet. Es gehe darum, „die Orientierung an einem sinnstiftenden Kontext nicht aus den Augen zu verlieren“ (Koubek et al., 2009, S. 268). Gerade in Naturwissenschaften und Technik sei zunächst das Interesse groß, da durch naturwissenschaftliche Phänomene und die Technik im Alltag ein Grundinteresse an den Fächern existiere. Nach anfänglicher Euphorie stelle sich aber immer wieder ein Verdruss ein, da die notwendigen Erklärungen unter dem „Deckmantel“ der fachlichen Exaktheit immer komplexer und wissenschaftlicher werden. (Koubek et al., 2009, S. 269) Ältere Schülerinnen und Schüler präferieren meist das Diskutieren und Bewerten und nicht das Berechnen und Formalisieren (Elster, 2006, S. 3).
In den Naturwissenschaften gibt es aus diesem Grund seit längerem drei verschiedene Projekte (Biologie, Physik und Chemie im Kontext (bik, piko und ChiK)), die sich kontext-

orientierten Ansätzen widmen. Einig sind sich diese Projekte darin, dass der Kontext für die Schülerinnen und Schüler aus lebensweltlichen Fragen aus Alltagsbezügen kommen muss und so Interesse und Lernerfolg steigern soll. Obwohl die Kernlehrpläne der Naturwissenschaften zum Thema kontextorientierter Unterricht identisch sind, unterscheidet sich jedoch das Verständnis von Kontextbezügen und ihre Einbettung in den Unterricht in den jeweiligen kontextorientierten Projekten. (Koubek et al., 2009, S. 270)
Basierend darauf formulierte Koubek für IniK (Informatik im Kontext) drei Prinzipien:

1. Orientierung am Kontext
2. Orientierung an Standards für die Informatik in der Schule
3. Methodenvielfalt
   (Koubek et al., 2009, S. 271)

1.2 Eisenbahn als informatischer Kontext - Project LoCo

Das Projekt „LoCo - Eisenbahn als informatischer Kontext“ knüpft an das Konzept von IniK an. Basierend auf den von Koubek formulierten fünf Kriterien Mehrdimensionalität, Breite, Tiefe, Lebensweltbezug und Stabilität (Diethelm et al., 2011, S. 102/103) modifizierte man das Konzept für einen kontextorientierten Unterricht, um es den im Informatikunterricht gegebenen Ressourcen anzupassen (Thomas, 2021, S. 5). Als Ausgangsposition des Projekts LoCo ergaben sich so folgende vier Punkte:

1. Langfristig strukturreicher Kontext aus der Lebenswelt
2. Enge Anbindung an curriculare Vorgaben und Empfehlungen
3. Momente zur potenziell intrinsischen Motivation von Schülerinnen und Schülern
4. Modulares Verknüpfen von flexibel einsetzbaren Bausteinen
   (Thomas, 2021, S. 5)

Der gewählte Kontext soll sich also, wie in ChiK angedacht, als roter Faden durch den Unterricht ziehen (Thomas, 2021, S. 6). Im Gegensatz zu dem von ChiK vorgestellten Konzept beschränkt sich der Kontext aber nicht nur auf eine Unterrichtsreihe, sondern - wenn möglich - auf den gesamten Informatikunterricht. Dies soll durch modulares Verknüpfen von flexibel einsetzbaren Bausteinen gelingen. Dabei können Lehrerinnen und Lehrer nach Belieben Bausteine im Kontext Eisenbahn zu unterschiedlichen Themen des Informatikunterrichts nutzen. Ein solcher Baustein wäre beispielsweise der in dieser Arbeit beschriebene Baustein zur Einführung in die Objektorientierung anhand einer digitalen Gleisbildstellpultsimulation.

Ein geeigneter Kontext muss also besonders die Kriterien der Mehrdimensionalität, Tiefe und Breite erfüllen, um langfristig als sinnstiftender Kontext dienen zu können. Aber auch die von Koubek geforderte Stabilität und der Lebensweltbezug des Kontexts sollen natürlich gegeben sein.

Die Eisenbahn bietet sich in dieser Hinsicht als Kontext an. Mehrdimensionalität wird durch die Möglichkeiten der verschiedenen Fragestellungen, die dieser Kontext aufwirft, charakterisiert (Diethelm et al., 2011, S. 102). Dabei sind unter anderem ökologische Fragen, wie zum Beispiel „Wie sehr kann die Eisenbahn gegen den Klimawandel helfen?“, sowie ethische Fragen wie das „Trolley-Problem“ (Weichenstellerfall) möglich. Aber natürlich können sich auch informatische Fragen beispielsweise zur Steuerung oder Fahrplanerstellung aus der Lebenswelt der Schülerinnen und Schüler ergeben.

Die nötige Breite besitzt ein Kontext, wenn er nicht nur aus informatischer Sicht interessant ist, sondern auch eine gesellschaftliche Relevanz besitzt (Diethelm et al., 2011, S. 102). Die Eisenbahn spielt in der Gesellschaft eine große Rolle, wie man jetzt wieder bei der Einführung des 9€-Tickets feststellen konnte. Im Jahr 2021 verzeichnete die deutsche Bahn AG trotz Corona rund 1,4 Milliarden Reisende im Schienenpersonenverkehr (Deutsche Bahn AG, 2021, S. 4). Allein dieser Teilbereich des Kontexts Eisenbahn zeigt schon die enorme gesellschaftliche Relevanz.

Mit der Tiefe des Kontexts fordert Koubek einen Unterricht im Rahmen eines Kontexts, der möglichst viele Kompetenzen der GI-Bildungsstandards vermitteln soll (Diethelm et al., 2011, S. 102). Eisenbahnen könnten als Kontext bei Sortierverfahren (Thomas, 2021, S. 8) und Datenbanken (Mahnke, 2021) dienen, aber auch bei der Automatentheorie im Zusammenhang mit der Geldrückgabe bei Ticketautomaten und der Programmierung, wie diese Arbeit zeigen soll.

Die Frage nach der Stabilität des Kontexts ist bei der Eisenbahn schnell beantwortet. Seit nun fast 200 Jahren fahren Eisenbahnen in Deutschland und der Bedarf an Eisenbahnen im Personennah- und Güterverkehr ist heute noch groß.

Abschließend gilt es noch zu klären, ob der Kontext „direkten Bezug und Handlungsrahmen in der Lebenswelt der Schülerinnen und Schüler“ (Diethelm et al., 2011, S. 102) aufweist. Dabei gilt es auch, den Genderaspekt zu beachten. Der Kontext soll das Interesse beider Geschlechter regen und gleichermaßen erlebbar sein. Wahrscheinlich haben durchschnittlich Schüler ein höheres Interesse an Eisenbahnen als ihr weiblicher Gegenpart, allerdings kann sich der Kontext durch seine starke Alltagspräsenz dennoch eignen. So nutzen die meisten Schülerinnen und Schüler die Eisenbahn regelmäßig in ihrer Frei-

zeit, bei Reisen und potenziell auch auf ihrem Weg zur Schule.
Resümierend kann festgehalten werden, dass der Kontext Eisenbahn die fünf Kriterien von Koubek erfüllt und sich mit dem Konzept des Projekts LoCo auch als langfristiger Kontext anbietet.

2 Einführung in die objektorientierte Programmierung

Um die Frage „Warum bietet sich das digitale Gleisbildstellpult als Beispiel für die kontextorientierte Einführung in die objektorientierte Informatik an?“ beantworten zu können, gilt es zu betrachten, wie man anhand des digitalen Gleisbildstellpults die vom Kernlehrplan geforderten Kompetenzbereiche und Inhaltsfelder vermitteln kann. Darüber hinaus bietet sich ein Vergleich zum momentan gängigen Konzept zur Einführung in die objektorientierte Programmierung an, dem Rover aus dem Schulbuch „Informatik 1“ des Schöningh Verlags, das ebenso wie das Gleisbildstellpult die Java-Entwicklungsumgebung Greenfoot nutzt. Dabei soll der Fokus vor allem auf der Nutzung des Gleisbildstellpults im Vergleich zum Rover liegen. Für mögliche Unterrichtsabläufe, zum Beispiel zur Einführung in das Thema Eisenbahn und elektronische Stellwerke, verweise ich auf den im Projekt LoCo entstandenen „Baustein 001 - Das Stellwerk“ von Prof. Dr. Thomas (Thomas, 2022).

2.1 Objekte und Klassen

Im Kernlehrplan für Informatik wird beim inhaltlichen Schwerpunkt „Objekte und Klassen“ aus dem Inhaltsfeld „Daten und ihre Strukturierung“ zunächst gefordert, dass die Schülerinnen und Schüler lernen sollen, Eigenschaften, Operationen und Beziehungen von Objekten zu ermitteln sowie Attribute, Methoden und Assoziationsbeziehungen von Klassen zu modellieren. Zusätzlich sollen sie Attributen, Parametern und Rückgaben von Methoden einfache Datentypen, Objekttypen oder lineare Datensammlungen zuordnen können. (*Kernlehrplan Informatik*, 2014, S. 22) Im Baustein wird hierzu ein Objektspiel als Unterrichtsmethode vorgeschlagen (Thomas, 2022, S. 15/16). Hier bietet sich das digitale Gleisbildstellpult als Kontext an, da die Schülerinnen und Schüler die Klassen oder auch Objekte aus ihrem Alltag identifizieren können. So können Züge, Signale, Haltestellen oder Schienen (im Programm: Blöcke) als solche herausgearbeitet werden und ihre Attribute, Methoden und Beziehungen modelliert werden.
Allerdings ist es ebenso wie beim Rover im Buch „Informatik 1“ auch möglich, die Schülerinnen und Schüler durch Beschreibung des Kontexts und der Funktionalität der einzel-

nen Objekte Rückschlüsse auf Klassen- und Objektkarten ziehen zu lassen. Analog zu dem Beispiel aus dem Buch, in dem die Klassen Rover, Huegel, Gestein und Marke betrachtet werden (*Informatik 1*, 2014, S. 23-25), können beim Gleisbildstellpult die Klassen Block, Zug, Signal und Haltestelle untersucht werden. Dabei bietet das Gleisbildstellpult den Schülerinnen und Schülern zusätzliche Möglichkeiten bei der Entwurfsentscheidung, da es viele Wege gibt, die Klassen zu modifizieren und zu erweitern. Exemplarisch hierfür wäre die Methode „fahreZuHaltestelle“ für die Klasse Zug (siehe Abbildung 1). Als weitergedachte Form der Methode „fahre“ könnte man so eine Beziehung zwischen den Klassen Zug und Haltestelle aufbauen. Ebenso ist es den Schülerinnen und Schülern selbst überlassen, wie sie Weichen modellieren möchten. Man kann die Klasse Block so ändern, dass auch Weichen inkludiert sind, oder man kann Weichen eine eigene Klasse geben. Infolgedessen wäre es möglich, den Schülerinnen und Schülern das Prinzip von Unterklassen und Vererbung näherzubringen (siehe Abbildung 1).

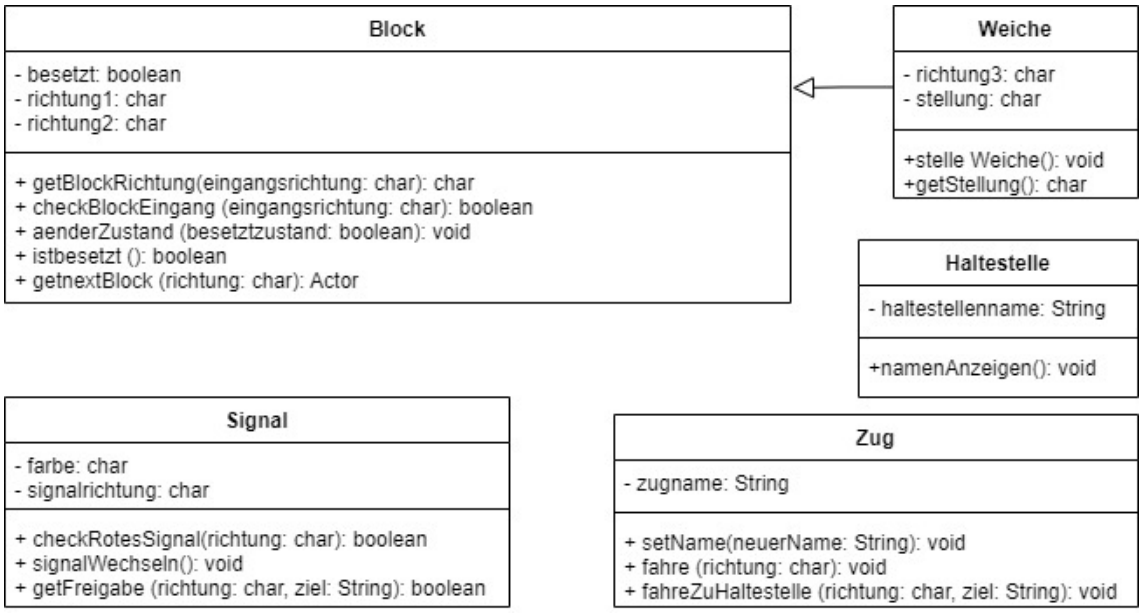

Abbildung 1: Beispiel für mögliche Klassenkarten zum digitalen Gleisbildstellpult

2.2 Methoden

Anschließend sehen sowohl der Baustein (Thomas, 2022, S. 6/7) als auch das Schulbuch (*Informatik 1*, 2014, S. 27-33) vor, den Schülerinnen und Schülern die Java-Entwicklungsplattform Greenfoot vorzustellen sowie ihnen Methodenaufrufe näherzubringen. In Analogie zur Vorgehensweise beim Rover im Schulbuch könnte man den Schülerinnen und Schülern die Methoden „fahre(richtung: char)“, „signalWechseln()“, und „stelleWeiche()“ zur Verfügung stellen. Die Methode „fahre(richtung: char)“ lässt den Zug immer um ein

Feld in die angegebene Richtung fahren. Für die Anwendung dieser drei Methoden bekommen die Schülerinnen und Schüler Beispielszenarien unterschiedlicher Komplexität an die Hand (siehe Abbildung 2).

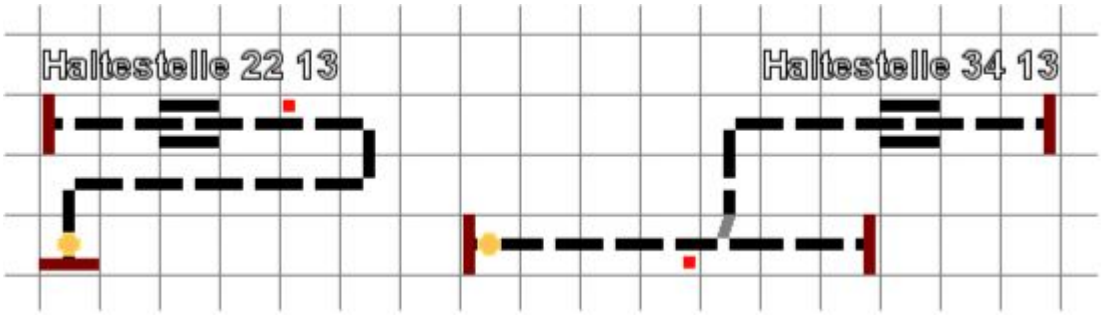

Abbildung 2: Beispielszenarien zum Einüben von Methodenaufrufen

Sie müssen nun überlegen, mithilfe welcher Methodenaufrufe der Zug in die Haltestelle einfahren kann. In Greenfoot kann diese Reihe von Methodenaufrufen in die Methode act() programmiert werden, sodass die Schülerinnen und Schüler ihre Ergebnisse überprüfen können und gleichzeitig lernen, wie Methoden in Greenfoot aufgerufen werden. Dies hilft auch, die im Kernlehrplan vorgesehenen Kompetenzen des Testens und Korrigierens von Programmen sowie das Analysieren und Modifizieren einfacher Algorithmen auszubilden (*Kernlehrplan Informatik*, 2014, S. 21/23).
Je nach Umfang der Beispiele werden die Schülerinnen und Schüler dann bemerken, dass manchmal identische Methodenaufrufe mehrmals hintereinander erfolgen. So muss im schwierigeren Beispielszenario (siehe Abbildung 2 rechtes Szenario) der Zug zweimal je drei Blöcke nach Osten fahren. Dies könnte man nutzen, um den Schülerinnen und Schülern das Programmieren eigener kleiner Methoden näherzubringen. Eine eigene Methode dreiBloeckeNachOsten() könnte selbstständig programmiert und dann in act() aufgerufen werden. Diese Methoden lassen sich in ihrer Komplexität und ihrem Umfang beliebig erweitern.
Somit kann die Einführung in Greenfoot und Methodenaufrufe mithilfe des Gleisbildstellpults sehr ähnlich zu der des Rovers erfolgen. Während man beim Rover Gesteine analysiert, werden beim Gleisbildstellpult Weichen gestellt und Signale auf Grün geschaltet.

2.3 Sequenzdiagramm

Um Abläufe der Methoden und Beziehungen von Objekten, wie im Kernlehrplan gefordert, auch in grafischer Form darstellen zu können, bietet es sich an, den Schülerinnen und Schülern das Sequenzdiagramm vorzustellen. Auch wenn sie im Kernlehrplan nicht

explizit gefordert werden, bieten Sequenzdiagramme eine sehr gute Möglichkeit, das bereits Gelernte mit den kommenden Themen Klassendiagramm und Algorithmen zu verbinden.

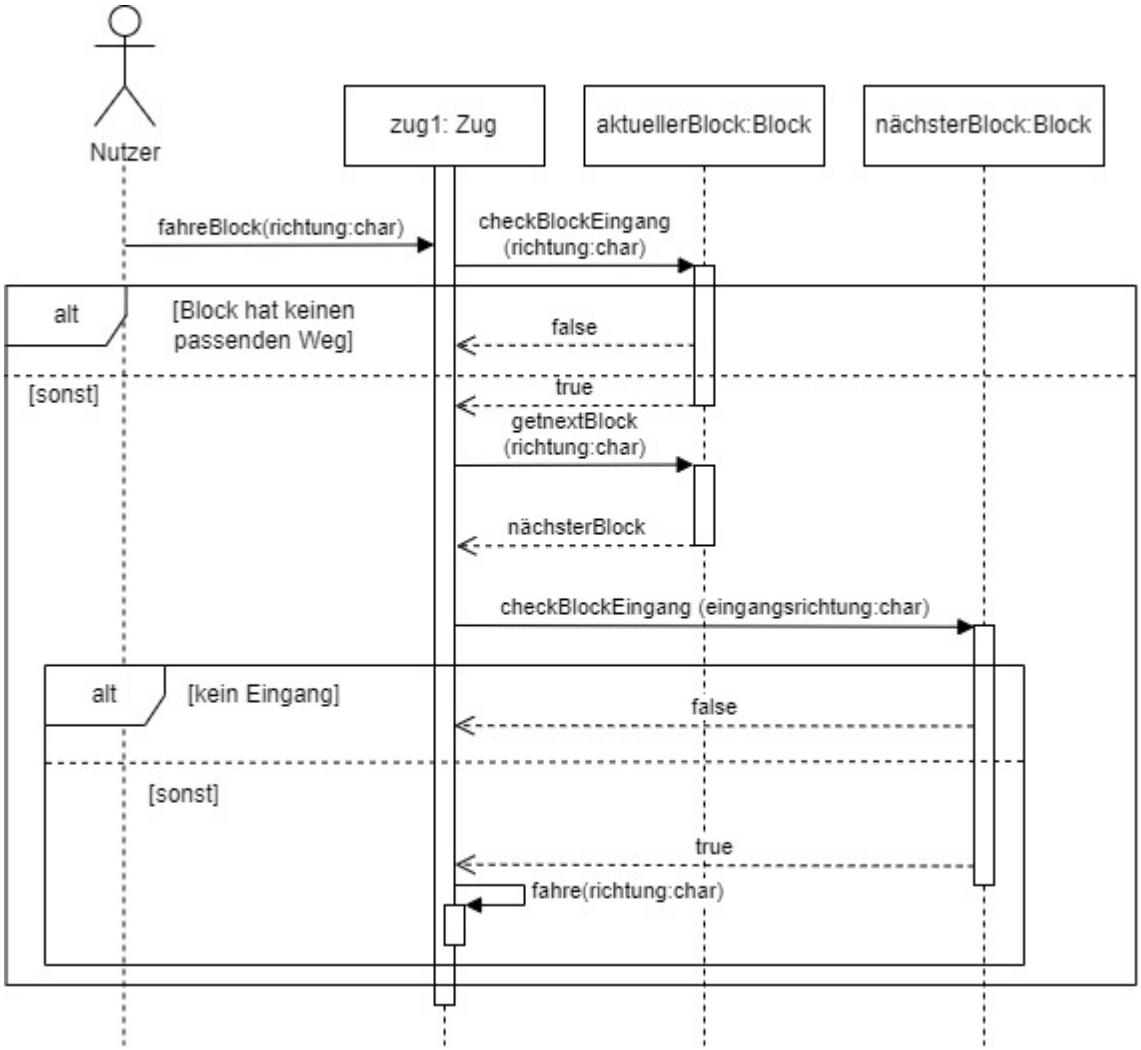

Abbildung 3: Sequenzdiagramm für fahreBlock(richtung: char)

Als Beispiel wird hier die Methode fahreBlock(richtung: char) genauer betrachtet. Anhand des Sequenzdiagramms dieser Methode wird den Schülerinnen und Schülern ein Einblick in das Konzept gegeben, wie der Zug später mit nur einem Methodenaufruf beliebige Schienennetze befahren soll (siehe Abbildung 3). Da sich der Zug, anders als der Rover, nicht dreht und dann in „Blickrichtung“ fährt, sondern von den Schienen geleitet wird und eine Schiene in die Richtung verlässt, die nicht die eingehende Richtung war, muss die fahreBlock-Methode immer die nötigen Informationen sowohl des aktuellen als auch des kommenden Blocks bekommen. Zuerst muss geprüft werden, ob der aktuelle Block einen Ausgang in der geforderten Richtung hat. Danach wird der nächste Block in dieser Richtung benötigt. Von diesem muss die Information eingeholt werden, ob er in

der Richtung, in die der Zug in den Block einfahren würde, einen Eingang hat. Sollte eine der beiden Bedingungen nicht erfüllt werden, kann der Zug nicht fahren.
Möchte man es komplexer gestalten, ist es auch noch möglich, Signale und Weichen in das Sequenzdiagramm mit einzubauen. Die Schülerinnen und Schüler könnten, wenn sie mit der Grundidee des Fahrens des Zugs vertraut sind, selbst überlegen, wie man diese Abläufe in einem Sequenzdiagramm darstellen könnte. Dabei würde der Kompetenzbereich des Darstellens und Interpretierens gefördert. Präziser: Die Schülerinnen und Schüler würden Beziehungen und Abläufe in Form von grafischen Darstellungen interpretieren und selbst darstellen.
Abschließend können die Sequenzdiagramme für einen Ausblick auf kommende Unterrichtsreihen genutzt werden, indem die Schülerinnen und Schüler dem Sequenzdiagramm entnehmen, dass sich die Methode fahreBlock(richtung: char) immer wieder in Schleife aufrufen lässt, bis der Zug wahlweise an einer Haltestelle angekommen ist oder vor einem Signal oder Prellbock steht. Dieser Gedankengang kann dann in einer späteren Unterrichtsreihe zu Algorithmen, insbesondere Schleifen, wieder aufgegriffen werden. Ebenso wird hier schon ein Einblick in die Bool'sche Aussagenlogik gewährt, da die Bedingung des passenden Ausgangs und die des passenden Eingangs erfüllt sein müssen (siehe Tabelle 1, Abschnitt 2.5, S. 12).
Im Schulbuch „Informatik 1“ wird das Sequenzdiagramm erst nach der Einführung von Algorithmen zusammen mit dem Klassendiagramm vorgestellt (*Informatik 1*, 2014, S. 109). Dabei wird dann auch nicht mehr auf den Kontext des Rovers zurückgegriffen, sondern ein neuer Kontext präsentiert. Vermutlich liegt der Grund dafür in der mangelnden Komplexität der Methoden des Rovers, die sich nicht für Sequenzdiagramme anbieten, da die Kommunikation von Objekten auf Methodenebene nur in geringem Maße stattfindet. In Bezug auf die Frage, warum sich das digitale Gleisbildstellpult als Beispiel für die kontextorientierte Einführung in die objektorientierte Informatik anbietet, kann hier ein Vorteil festgestellt werden.
Ich halte es für sinnvoll, das Sequenzdiagramm vor der Unterrichtsreihe zu Algorithmen zu behandeln, da dieses als Planungswerkzeug den Schülerinnen und Schülern einen leichteren Weg zum Verständnis und Implementieren von Methoden geben kann. So wird der Charakter der objektorientierten Programmierung in Sequenzdiagrammen sehr deutlich dargestellt, und den Schülerinnen und Schülern wird eine Möglichkeit an die Hand gegeben, ihre Überlegungen zu den Abläufen einer Methode und der Kommunikation von Objekten grafisch anschaulich darzustellen. Darüber hinaus behandelt

man neben dem bereits erwähnten Kompetenzbereich des Darstellens und Interpretierens auch einen Teilaspekt des Inhaltsschwerpunktes Objekte und Klassen. Auch wenn das Sequenzdiagramm selbst nicht im Kernlehrplan erwähnt wird, so wird erwartet, dass die Schülerinnen und Schüler die Kommunikation zwischen Objekten grafisch darstellen können. Dabei sollen informatische Modelle konstruiert, modifiziert und auch erweitert werden (*Kernlehrplan Informatik*, 2014, S. 20/22). Diese Kompetenzen können in der Unterrichtsreihe „Algorithmen“ dann weiter ausgebildet werden, wenn die Schülerinnen und Schüler selbstständig das Sequenzdiagramm für Entwurfs- und Implementierungsentscheidungen nutzen.

2.4 Klassendiagramm

Das Klassendiagramm bildet den Abschluss in der Unterrichtsreihe zur Einführung in die objektorientierte Programmierung. Der Kernlehrplan sieht vor, dass die Schülerinnen und Schüler Klassen und ihre Assoziations- und Vererbungsbeziehungen in Diagrammen grafisch darstellen sowie Klassen durch Beschreibung der Funktionalität der Methoden dokumentieren können (*Kernlehrplan Informatik*, 2014, S. 22).

Im Gegensatz zum Rover kann man beim digitalen Gleisbildstellpult ein Klassendiagramm sowohl mit Vererbungs- als auch Assoziationsbeziehungen erstellen.[1] Dabei können die Schülerinnen und Schüler anhand ihrer CRC-Karten aus dem Objektspiel oder der entworfenen vorläufigen Klassenkarten (siehe Abbildung 1) und der nun kennengelernten Methoden neue Klassenkarten erstellen. Hilfreich können ebenfalls die zuvor besprochenen Sequenzdiagramme sein, anhand derer Beziehungen zwischen Objekten verfestigt wurden.

Ebenso wie bei den Sequenzdiagrammen verwendet das Schulbuch andere Kontexte als den des Rovers zur Einführung des Klassendiagramms, unter anderem auch wieder den für das Sequenzdiagramm gewählten Kontext (*Informatik 1*, 2014, S. 98-119). Da Unterklassen im Beispiel des Rovers nicht existieren, können Vererbungsbeziehungen hier nicht gezeigt werden. Wie bereits in Abschnitt 2.3 erwähnt, wird das Klassendiagramm erst nach der Einführung von Algorithmen vorgestellt, was in meinen Augen erneut zu hinterfragen ist. So ist das Klassendiagramm eine sinnvolle Hilfe bei der Planung von Klassen und ihren Methoden. Im Kernlehrplan wird gefordert, dass die Schülerinnen und Schüler lernen sollen, Klassen in einer Programmiersprache zu implementieren (*Kernlehrplan Informatik*, 2014, S. 22). Dies wäre meiner Meinung nach eine

[1] Siehe Anhang A.1 Abbildung 5 S. 33

gute Brücke von der Planungsphase zur Programmierphase. So könnte das Kapitel Algorithmen mit der Implementierung einiger Klassen begonnen werden. Eine Möglichkeit, wie dies am Beispiel des digitalen Gleisbildstellpultes umsetzbar wäre, folgt im kommenden Abschnitt. So wäre es mit dem Kontext der Eisenbahn, im Speziellen mit dem digitalen Gleisbildstellpult, möglich, die gesamte Einführung in die objektorientierte Programmierung samt Sequenz- und Klassendiagrammen für die nächste Unterrichtsreihe, die Algorithmen, zu nutzen. Dabei würden die Schülerinnen und Schüler auch lernen, dass vor dem Programmieren der Algorithmen erst Entwurfs- und Implementierungsentscheidungen getroffen werden müssen. Gerade bei der objektorientierten Programmierung sollte es ihnen nähergebracht werden, wie wichtig eine gute Planung ist. Vor dem Programmieren einer Methode sollte bereits klar sein, welche Objekte miteinander kommunizieren müssen (Sequenzdiagramm) und welche Beziehungen die Klassen zueinander haben (Klassendiagramm). Somit würde man den Schülerinnen und Schülern allein durch die Reihenfolge der Phasen innerhalb der Unterrichtsreihe den Weg weisen, wie sie in Zukunft eigene Projekte angehen sollten.

2.5 Algorithmen

Um wie in den Abschnitten zuvor angedeutet zu zeigen, dass sich das digitale Gleisbildstellpult auch nach der Einführung in die Grundzüge der objektorientierten Programmierung für das Erlernen erster Algorithmen eignet, möchte ich Beispiele geben, wie die im Kernlehrplan vorgegebenen Ziele des Inhaltsfelds Algorithmen erlernt werden können. Diese sind das Analysieren, Erläutern, Modifizieren und Entwerfen einfacher Algorithmen und Programme. Unter Verwendung von Variablen und Wertzuweisungen, Kontrollstrukturen, sowie Methodenaufrufen sollen diese Algorithmen und Programme implementiert und an Beispielen getestet werden können (*Kernlehrplan Informatik*, 2014, S. 23). Das Schulbuch „Informatik 1“ betrachtet im Kontext des Rovers dabei Wiederholungen, Zählschleifen, bedingte Anweisungen und logische Operatoren (*Informatik 1*, 2014, S. 38-59).

Wiederholungen, die an eine Bedingung geknüpft sind, können den Schülerinnen und Schülern analog zum Beispiel des Rovers, der, solange er rechts von sich einen Berg hat, vorwärts fahren soll (*Informatik 1*, 2014, S. 38/39), auch anhand des Gleisbildstellpultes beigebracht werden. Dafür soll überlegt werden, wie man den Zug so lange weiterfahren lassen kann, bis er eine Haltestelle erreicht hat. Hierbei kann, ähnlich wie beim Rover, der beim Umfahren eines Berges Gestein untersuchen soll (*Informatik 1*, 2014, S. 41-44),

die Aufgabe erschwert werden, wenn Signale an den Schienen stehen und diese vor der Weiterfahrt auf grün gestellt werden sollen.
Um Zählschleifen einzuführen, kann den Schülerinnen und Schülern ein Schienennetz mit mehreren Haltestellen gegeben werden (siehe Abbildung 4). Sie sollen eine Methode schreiben, die den Zug bis beispielsweise zur vierten Haltestelle fahren lässt. Ähnlich könnte man Aufgaben stellen, bei denen der Zug die Freigabe der nächsten zwei Signale hat, aber beim dritten stehen bleiben soll.

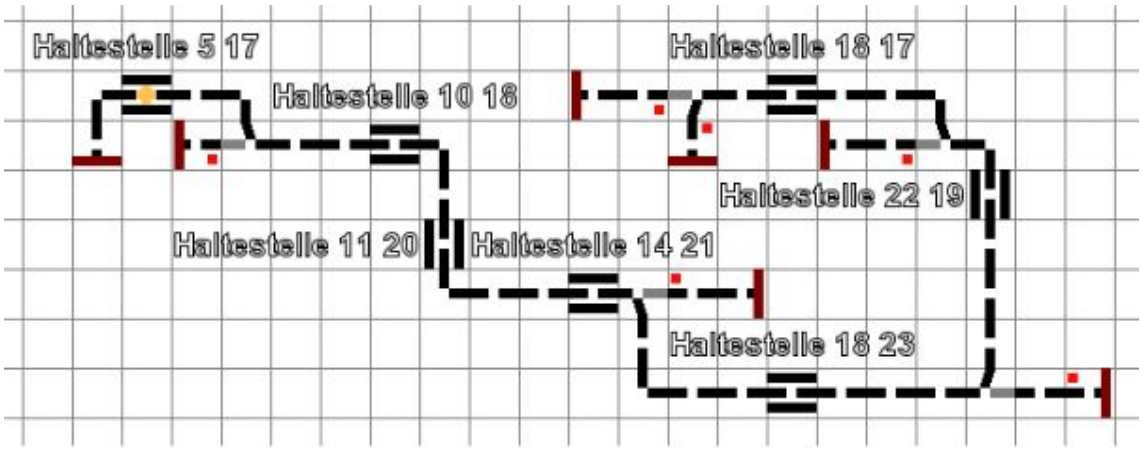

Abbildung 4: Beispiel eines Schienennetzes für eine Aufgabe zur Einführung von Zählschleifen

Gleichermaßen funktioniert die passende Aufgabenstellung im Schulbuch, die eine Untersuchung von genau vier Gesteinsproben fordert (*Informatik 1*, 2014, S. 44-47). In beiden Aufgaben soll eine Tätigkeit also in einer bestimmten Anzahl von Wiederholungen durchgeführt werden, weshalb sich eine Zählschleife als Lösung anbietet.
Als nächstes sollen bedingte Anweisungen behandelt werden. Hier kann erneut die Methode fahre(richtung:char) im Zusammenhang mit einem Signal genutzt werden. Falls der Zug vor einem Signal steht, soll er um Freigabe bitten, ansonsten soll er weiterfahren. Eine andere Möglichkeit wäre die Kontrolle, ob der Zug vor einem Prellbock steht.
Abschließend gilt es noch zu prüfen, ob sich das digitale Gleisbildstellpult auch für die Bool'sche Aussagenlogik und die damit verbundenen logischen Operatoren anbietet. Im Schulbuch „Informatik I“ nutzt man dafür ein Szenario, in dem der Rover nur fahren darf, wenn sich rechts und/oder links von ihm Hügel befinden. Im Kontext Gleisbildstellpult lässt sich die und-Verknüpfung leicht anhand der Methode fahre(richtung:char) erklären. Ein Zug kann nur in eine Richtung fahren, wenn der aktuelle Block dort einen Ausgang hat und wenn der nächste dort einen Eingang hat. Dabei könnte man sich auf das zuvor behandelte Sequenzdiagramm beziehen (siehe Abbildung 3, Abschnitt 2.3). Eine Wahrheitstabelle ließe sich entsprechend herleiten (siehe Tabelle 1).

Ausgang aktueller Block	Eingang kommender Block	Ausgang **und** Eingang
wahr	wahr	wahr
wahr	falsch	falsch
falsch	wahr	falsch
falsch	falsch	falsch

Tabelle 1: Wahrheitstabelle für die und-Verknüpfung.

Die oder-Verknüpfung ließe sich durch Szenarien erklären, in denen der Zug beispielsweise nur auf geraden Schienen fahren darf oder auf Schienen, die ein grünes Signal anzeigen. Anhand von Beispielszenarien könnten die Schülerinnen und Schüler dann überlegen, wie weit der Zug fahren dürfte.
Analog können auch für den nicht-Operator Beispielszenarien gefunden werden: Solange der Zug nicht vor einem roten Signal steht, soll er fahren.
Den Schülerinnen und Schülern könnte man abschließend die Aufgabe geben, sich eigene Szenarien zu überlegen und dazu sinnvolle Schienennetze zu konstruieren, die dann in Partnerarbeit getauscht und gelöst werden können. Dabei können Kombinationen der Operatoren genutzt werden, um die Komplexität der Szenarien zu erhöhen.
Zusammenfassend lässt sich feststellen, dass die Einführung in die Algorithmen anhand des digitalen Gleisbildstellpults ähnlich zu der des Schulbuchs im Kontext des Rovers geschehen kann. Auch das Entwickeln der Algorithmen, das nun in der Unterrichtsreihe folgen würde, ist mit den oben gezeigten Beispielen möglich, da diese sich auch gut zu größeren und komplexeren Methoden kombinieren lassen.

3 Konzept und Implementierung des Programms

Im Folgenden wird die Umsetzung eines digitalen Gleisbildstellpults in der Java-Entwicklungsumgebung Greenfoot vorgestellt. Dabei werden sowohl das Konzept und die damit verbundenen Entscheidungen als auch die Implementierung ausgewählter Methoden näher beleuchtet.

3.1 Konzept

Konzeptionelle Entscheidungen galt es vor allem in zwei Bereichen zu treffen. Zunächst musste ein optisches Konzept aufgestellt werden, das auch in Greenfoot umsetzbar ist. Zusätzlich musste ein System festgelegt werden, wie Züge auf den Schienen fahren sollen.

3.1.1 Optisches Konzept

Optisch ist das digitale Gleisbildstellpult stark an Gleisbildstellpulte für Modelleisenbahnen angelehnt. Schaut man sich beispielsweise das Gleisbildstellpult „Track-Control“ der Firma Uhlenbrock an (Uhlenbrock Elektronik, 2020), so erkennt man die Einteilung in viele kleine Vierecke, mithilfe derer beliebige Schienennetze dargestellt werden können. Dabei werden die Schienenteile schlicht schwarz gehalten, mit Aussparungen in der Mitte für Lampen, die leuchten, wenn der Zug die Schiene befährt.
Diese Grundzüge eines Gleisbildstellpults bieten sich für Greenfoot geradezu an. Die „Greenfoot-Welt“ besteht aus quadratischen Zellen, deren Größe und Anzahl in x- und y-Richtung im Konstruktor der Welt bestimmt wird. Zusätzlich kann man der Zelle ein Bild zuweisen. Wählt man dieses Bild so, dass zwei nebeneinanderliegende Seiten grau sind und das restliche Bild weiß, so entsteht durch das Aneinanderreihen derselben Bilder ein Raster aus weißen Quadraten mit grauem Rand.[2]
Für das digitale Gleisbildstellpult habe ich mich für eine Größe von 50 Zellen in der Breite und 25 Zellen in der Höhe entschieden. Die quadratischen Zellen haben dabei eine Seitenlänge von 30 Pixeln. Die Zellengröße von 30 Pixeln wurde gewählt, da man alle Schienenblöcke sowie Signale und Haltestellen gut erkennen kann. Dennoch ist sie klein genug, dass man auch auf kleineren Bildschirmen ein ausreichend großes Schienennetz bauen kann. Hierzu sei gesagt, dass die Größe von 50x25 Zellen für Full HD Bildschirme (1920x1080 Pixel) optimiert ist und Greenfoot so den Bildschirm gut ausfüllt. Für kleinere Bildschirme kann die Anzahl der Zellen in Breite und Höhe beliebig angepasst werden.
Um die Schienen optisch passend zum Vorbild zu gestalten, wurde für jede Schienenart ein passendes Bild erstellt, welches eine Zelle im Raster ausfüllt.[2] Dabei wurde immer die Mitte der Schiene weiß gelassen, was die Aussparung für Lampen darstellen soll. Der Zug fährt als gelber Kreis in der Mitte einer Zelle auf den Schienen, sodass der Effekt von an- und ausgehenden Lampen entsteht. Weichen haben im Gegensatz zu den anderen Schienenarten zwei verschiedene Bilder. Für jede der beiden Weichenstellungen gibt es jeweils ein eigenes Bild, bei dem der zu befahrende Weg schwarz und der abgetrennte Weg grau gekennzeichnet ist.[2]
Auf weitere Schienenarten, wie zum Beispiel Drei-Wege-Weichen oder schräge Schienen mit entsprechenden Kurven, wurde aus Gründen der Übersichtlichkeit verzichtet. Da man mit den bestehenden Blöcken ausreichend komplexe Schienennetze für die Einführung in die Objektorientierung bauen kann und sonst die Actorliste noch länger

[2] Siehe Anhang A.2.1 Abbildung 6 S. 34

und unübersichtlicher wäre, habe ich mich auf diese Schienenarten beschränkt.
Die lange Actorliste könnte man zwar mithilfe von enumerations (enums) umgehen,[3] aber dadurch erschwert man das Erstellen eines Schienennetzes enorm. Die Umsetzung mit enums fordert für jeden neuen Block die Eingabe eines Strings, um die passende Schienenart zu wählen. Dies ist besonders für Schülerinnen und Schüler, die mit diesem Programm an die objektorientierte Programmierung herangeführt werden sollen, schwierig und umständlich. Befinden sich aber alle Schienenarten in der Actorliste, genügen dafür zwei Klicks. Des Weiteren kann durch das Halten der „shift-Taste“ die zuletzt platzierte Schienenart ohne weitere Umwege erneut platziert werden, was in der Umsetzung mit enums nicht möglich wäre.
Aus diesen Gründen habe ich mich für eine längere Actorliste und gegen die Umsetzung mit enums entschieden. So gewinnt das digitale Gleisbildstellpult auch an Ähnlichkeit zu käuflich erwerbbaren Programmen, wie zum Beispiel dem „TrainController“ von Freiwald Software (?, ?). Dort lassen sich die Schienen per „drag and drop“ an die passenden Stellen ziehen und man benötigt ebenfalls keine Eingabe eines Strings, sondern kann per Klick aus einer Vielfalt von Streckenarten wählen.
Abgerundet wird das optische Konzept durch die Darstellung der Haltestellen, Prellböcke und Signale.[4] Diese habe ich, passend zu den Schienen, sehr schlicht gehalten. Haltestellen werden mit zwei parallel zu den Schienen laufenden schwarzen Geraden dargestellt. Diese können als Andeutung von Bahnsteigen verstanden werden. Prellböcke werden, wie auch im „TrainController“, als roter senkrecht zur Schiene stehender Block dargestellt. Signale stehen jeweils in Fahrtrichtung rechts der Schienen und werden als rotes beziehungsweise grünes Quadrat dargestellt.
Im Konstruktor der „Greenfoot-Welt“ wurde zusätzlich noch die Reihenfolge der Ebenen, in denen die einzelnen Klassen dargestellt werden, festgelegt.[5] Analog zu den gängigen Bildbearbeitungsprogrammen wird, falls die Bilder zweier Actor überlappen, das Bild des Actors in der höheren Ebene über dem anderen dargestellt. So fährt beispielsweise der Zug immer auf und nicht unter den Schienen. Standardmäßig wird in Greenfoot der zuletzt hinzugefügte Actor immer über den vorigen dargestellt. Sollte also ein Schienennetz erweitert werden, würden alle bereits existierenden Züge unter diesen neuen Schienen dargestellt.

[3] Siehe Anhang A.2.7 S. 110f.
[4] Siehe Anhang A.2.1 Abbildung 6 S. 34
[5] Siehe Anhang A.2.2 S. 34

3.1.2 Konzept der Blöcke

Da bei der Zugfahrt nicht der Zug lenkt, sondern die Schienen die Richtung vorgeben, muss auch bei der Umsetzung des digitalen Gleisbildstellpults die Richtung von den Schienen vorgegeben werden. Um dies zu realisieren, besitzt grundsätzlich jeder Schienenblock zwei Richtungen. Der Zug muss sich dann die Richtung merken, in der er in den Block eingefahren ist, um vom Schienenblock die jeweils andere Richtung, in die er fahren soll, zu bekommen. Dies wird dann beim nächsten Block wiederholt. So kann jede beliebige Strecke befahren werden. Ausnahmen bilden Weichen und Prellböcke. Während Prellböcke nur eine Richtung für Ein- und Ausfahrt besitzen, haben Weichen eine zusätzliche dritte Richtung und können gestellt werden.
Die Umsetzung von Prellböcken als Unterklasse der Klasse Block, die Kurven und Geraden beinhaltet, gelingt, indem beide Richtungen identisch sind, sodass die Richtung, in die der Zug in den Block einfährt, auch gleichzeitig die Richtung ist, in die der Zug den Block wieder verlassen kann. Jeder Prellbock (Nord, Ost, Süd und West) hat dementsprechend nur eine Richtung. Wird ein Prellbock in die „Greenfoot-Welt“ gesetzt, wird automatisch die zweite Richtung gleich der ersten gesetzt.[6]
Weichen in das Konzept zu integrieren, ist eine etwas umfangreichere Aufgabe. Die Unterklasse der Weichen besitzt zusätzlich zu den zwei Richtungen der Oberklasse Block eine dritte Richtung sowie die Information, ob sie für die Kurve oder Gerade gestellt ist. Um die Richtungen für alle Weichentypen konsistent zu definieren, habe ich folgendes Schema aufgestellt: r1 ist die Richtung, die, sollte ein Zug von dort in den Block einfahren, dem Zug ermöglicht, in beide anderen Richtungen weiterzufahren abhängig von der Stellung der Weiche. Sollte die Weiche für eine gerade Weiterfahrt gestellt sein, würde der Zug den Block in Richtung r2 verlassen. r3 gibt somit die Richtung der Weiche an, in die der Zug fährt, wenn die Weiche für eine Kurve gestellt ist. Aufgrund dieser eindeutigen Definition für alle Weichen ist es möglich, in der Unterklasse Weiche eine Methode zu schreiben, die für alle Weichentypen die passende Fahrtrichtung des Zuges abhängig von der Eingangsrichtung des Zuges und der Stellung der Weiche zurückgeben kann.[7]

3.2 Implementierung

In den kommenden Abschnitten wird die Implementierung einiger ausgewählter Methoden genauer betrachtet. Zunächst werden die einfachen Fahre-Methoden, die zu Beginn

[6] Siehe Anhang A.2.3 S. 59
[7] Siehe Anhang A.2.3 S. 63

von den Schülerinnen und Schülern genutzt werden, vorgestellt. Außerdem soll es um die Freigabe zur Abfahrt beziehungsweise Weiterfahrt der Züge gehen. Sowohl Signale als auch Haltestellen und Prellböcke lassen die Züge erst bei gegebener Freigabe fahren. Ebenso wird die Implementierung der Methode fahreZuHaltestelle(char startRichtung, String nameZielhaltestelle) vorgestellt. Dafür wird auch auf die Speicherung der Wege zu den einzelnen Haltestellen eingegangen. Abschließend wird die Möglichkeit erläutert, mehrere Züge gleichzeitig fahren zu lassen.

3.2.1 Ein Feld fahren

Um in Greenfoot einen Actor zu bewegen, muss man ihm eine neue Position mit dem Befehl setLocation(integer xPos, integer yPos) zuweisen. Die Methode fahre(char richtung)[8] soll den Zug passend zu der gegebenen Richtung bewegen. Dies geschieht mit einer Aneinanderreihung von if-Abfragen, die alle vier Himmelsrichtungen durchgehen und dann entsprechend den Zug um eine Einheit versetzen. Da es sich um eine Methode handelt, die die Schülerinnen und Schüler selbst aktiv nutzen und sehen, habe ich mich für eine Folge von if-Abfragen und gegen eine Switch-Case-Verzweigung entschieden. Die if-Abfrage ist für Anfängerinnen und Anfänger meiner Meinung nach elementarer und wichtiger zu lernen.
Um den Zug abhängig von den Blöcken, auf denen er steht, fahren zu lassen, gibt es die Methode fahreBlock(char richtung).[9] Da ein Zug, wie bereits beim Sequenzdiagramm in Abschnitt 2.3. beschrieben, von den Blöcken gelenkt wird, muss immer überprüft werden, ob der aktuelle Block einen Ausgang in der geforderten Richtung besitzt und der kommende Block einen passenden Eingang hat.
Die Methode checkBlockEingang(Actor block, char Richtung, boolean beachteWeichenstellung)[10] überprüft, ob der übergebene Block in der geforderten Richtung einen Eingang, beziehungsweise in diesem Fall einen Ausgang hat. Der übergebene Boolean sagt aus, ob die Weichenstellung dabei beachtet werden soll oder nicht. Die Methode gibt dann einen Boolean zurück, der besagt, ob ein passender Eingang oder Ausgang vorhanden ist.
Für fahreBlock(...) wird zunächst die Weichenstellung in der Abfrage berücksichtigt, sodass bei einem wahren Rückgabewert, die Frage des passenden Ausgangs zweifelsfrei geklärt ist. Sollte der aktuelle Block keinen passenden Ausgang haben, wird mit check-

[8] Siehe Anhang A.2.6 S. 81f.
[9] Siehe Anhang A.2.6 S. 82ff.
[10] Siehe Anhang A.2.3 S. 40

BlockEingang(...) erneut getestet, ob kein passender Ausgang vorhanden ist. Dieses Mal wird die Weichenstellung jedoch nicht berücksichtigt. Bei einem wahren Rückgabewert besteht nur die Möglichkeit, dass der Zug an einer Weiche steht, die noch gestellt werden muss. Ansonsten kann der Zug den Block nicht in der gewünschten Richtung verlassen. Nachdem nun geklärt ist, ob der aktuelle Block einen passenden Ausgang hat, muss überprüft werden, ob der kommende Block einen Eingang in der gewünschten Richtung hat. Dies funktioniert analog zur Überprüfung des Ausgangs des aktuellen Blocks. Sollte der nächste Block einen passenden Eingang haben, so wird der Zug auf diesen Block gesetzt. Sollte eine der nötigen Bedingungen nicht erfüllt werden, wird die passende Fehlermeldung ausgegeben.

3.2.2 Wege zu Haltestellen festlegen

Um eine Möglichkeit zu schaffen, Züge zu ihren Zielhaltestellen fahren zu lassen, müssen die Wege zu diesen Haltestellen festgelegt und gespeichert werden. Da ein Zug nur an Weichen die Möglichkeit hat, auf einen anderen Weg zu wechseln, ergibt es Sinn, dass die Weichen die nötigen Informationen für die Wege speichern. Dabei sollte neben der Richtung auch noch die Länge des Weges zu jeder Haltestelle gespeichert werden, sodass man gegebenenfalls die Weiche für kürzere Wege stellen kann. Ich habe mich dafür entschieden, dass jede Weiche einen String-Array wege[] bekommt. Jeder Eintrag in diesem Array hat die folgende Struktur “Richtung Haltestellenname; Schritte bis zur Haltestelle” (Beispiel: “N Haltestelle 15 13; 27”). Um dies umsetzen zu können wird beim Platzieren einer Haltestelle in der „Greenfoot-Welt“ die Methode wegeFestlegen() aufgerufen. Hierfür gibt es in Greenfoot die addedToWorld(World w)-Methode,[11] die aufgerufen wird, wenn ein Objekt dieser Klasse in der „Greenfoot-Welt“ platziert wird. Sollten nach dem Platzieren der Haltestelle weitere Weichen und Wege dem Schienennetz hinzugefügt werden, so muss die wegeFestlegen()-Methode erneut aufgerufen werden.
Die Methode wegeFestlegen() überprüft zunächst, ob die Haltestelle sich an einem Schienenblock befindet, da sonst keine Wege gefunden werden können. Dann wird je nach Haltestelle (senkrecht oder waagerecht) die Methode hinzufuegenWegeWeichen(Actor startBlock, int xPosStart, int yPosStart, char ausgangsRichtung, String haltestellenname, int schrittzahl, String[] bekannteWeichen)[12] in der Klasse Block für beide Ausgangsrichtungen der Haltestelle aufgerufen. Dabei wird die Schrittzahl auf 0 gesetzt und der String-Array mit bekannten Weichen ist leer, da, selbst wenn zuvor schon einmal die Wege für

[11] Siehe Anhang A.2.4 S. 72
[12] Siehe Anhang A.2.3 S. 46f.

eine Haltestelle festgelegt wurden, wieder alle Wege erneut überprüft werden müssen, da sich Teilstrecken geändert haben könnten.
Die Methode hinzufuegenWegeWeichen(...) soll, wie der Name schon sagt, den Weichen die Wege hinzufügen. Abhängig von der gegebenen Richtung wird dafür die nächste Weiche in dieser Richtung gesucht und in einem Object-Array gespeichert.
Dafür gibt es die Hilfsmethode findeNaechsteWeiche(Actor g, char eingangsRichtung, int anzahlSchritte).[13] Diese bekommt einen Block als Actor sowie die Eingangsrichtung und die Anzahl der bisherigen Schritte, die von der Haltestelle bis zum übergebenen Block gegangen werden mussten. Zurück gibt die Methode einen Array, der die Eingangsrichtung in die gefundene Weiche sowie ihre x- und y-Koordinaten und die Anzahl der nötigen Schritte beinhaltet (Beispiel: 'N', 12, 23, 10 für die Weiche an Stelle (12|23), die man in 10 Schritten von Norden anfahren kann). Um die nächste Weiche zu finden, wird das Schienennetz Block für Block überprüft. Dabei geht man analog zur fahreBlock(char richtung)-Methode vor. Es wird die Eingangsrichtung mit dem Blockeingang abgeglichen und überprüft, ob der Block eine Weiche ist. Sollte es sich um keine Weiche handeln und der Eingang passend sein, so wird der kommende Block in Richtung des aktuellen Blockausgangs gesucht. Ist dieser vorhanden, wird die Methode findeNaechsteWeiche(...) für den kommenden Block mit um eins erhöhter Schrittzahl erneut aufgerufen. Im Falle einer Sackgasse ohne Prellbock wird eine Fehlermeldung ausgegeben und ein leerer Object-Array zurückgegeben. Sollte es sich beim aktuellen Block tatsächlich um eine Weiche handeln, die auch einen passenden Eingang unabhängig von der Weichenstellung besitzt, so wird der passende Object-Array mit Richtung, Koordinaten und Schrittzahl zurückgegeben.
Sollte nun mithilfe der findeNaechsteWeiche(...)-Methode eine Weiche gefunden worden sein, wird diese in hinzufuegenWegeWeichen(...) auf Bekanntheit und Schrittlänge überprüft. Die Bekanntheit bezieht sich erneut nur auf diesen Durchlauf der Wegsuche. Sollte aus vorhergegangenen Durchläufen die Weiche bereits bekannt sein, wird dies hier nicht berücksichtigt, da durch ein neues Schienennetz mögliche neue Wege entstanden sein könnten.
Dazu wird die Hilfsmethode weicheSchonBekannt(String[] schonBekannteWeichen, int zahlSchritte, char weichenEingangsRichtung, int xPosNaechste, int yPosNaechste)[14] aufgerufen. Die Methode durchläuft den übergebenen String-Array und überprüft, ob die Weiche in Kombination mit der Richtung bereits bekannt ist. Die im Array gespeicherten

[13] Siehe Anhang A.2.3 S. 45
[14] Siehe Anhang A.2.3 S. 44f.

Strings haben die Form “Schrittzahl,Himmelsrichtung.x-Koordinate;y-Koordinate” (zum Beispiel “12,W.45;14” für 12 Schritte, Westrichtung an der Stelle (45|14)). Um herauszufinden, ob die Weiche in Kombination mit der Richtung bekannt ist, wird jeder Eintrag im Array daraufhin kontrolliert, ob er die Kombination “Richtung.x-Koordinate;y-Koordinate” enthält . Sollte das der Fall sein, so wird überprüft, ob der bekannte Weg kürzer als der mögliche neue Weg ist oder gleich lang. Dazu wird aus dem String der Teil bis zum „ , “ als Schrittzahl des bekannten Weges ausgelesen und mit der neuen übergebenen Schrittzahl verglichen. Ist der bekannte Weg kürzer oder gleich lang, gibt die Methode den boolean true zurück. In allen anderen Fällen ist der Rückgabewert false.
Wurde nun festgestellt, dass der gefundene Weg neu oder kürzer als der bislang bekannte Weg ist, so soll er im String-Array der bekannten Weichen dieses Durchlaufs gespeichert werden. Dafür existiert die Hilfsmethode weicheZuBekanntHinzufuegen(String[] schonBekannteWeichen, int zahlSchritte, char weichenEingangsRichtung, int xPosN, int yPosN).[15] Der übergebene String-Array wird durchlaufen um zu prüfen, ob es sich um einen unbekannten oder einen kürzeren Weg handelt. Handelt es sich um einen unbekannten Weg, wird der String-Array um eins vergrößert und der Weg an der neu geschaffenen Stelle in der oben beschriebenen Form gespeichert. Ist der Weg bereits bekannt, ersetzt der neue kürzere Weg den schon bekannten Weg im Array.
Nachdem man nun die Kombination aus Weiche und Richtung für diesen Durchlauf gespeichert hat, gilt es zu überprüfen, ob der gefundene Weg im Wege-Array der Weiche noch zu speichern ist. Dazu muss herausgefunden werden, ob die Weiche in dieser Richtung bereits einen Weg zur Haltestelle gespeichert hat und, falls ja, ob dieser kürzer ist als der mögliche neue Weg. Deshalb gibt es in der Klasse Weiche die Methode checkKuerzesterWeg(char ausgangsRichtung, String nameHaltestelle, int neueSchrittzahl).[16] Diese bekommt Richtung, Haltestellenname und Schrittzahl übergeben und durchläuft den Wege-Array auf der Suche nach Übereinstimmungen. Sollte die Kombination aus Richtung und Haltestellenname bekannt sein, wird die Schrittlänge verglichen. Ist der neue Weg kürzer als der bekannte Weg oder noch unbekannt, gibt die Methode den Booleanwert true zurück.
Gilt es den neuen Weg zu speichern, wird dies mit der Methode neuerWeg(char ausgangsRichtung, String nameDerHaltestelle, int anzahlDerSchritte)[17] in der Klasse Weiche gemacht. Dafür wird mit der Hilfsfunktion wegBekannt(char bekanntAusgangsRich-

[15] Siehe Anhang A.2.3 S. 43f.
[16] Siehe Anhang A.2.3 S. 66
[17] Siehe Anhang A.2.3 S. 65

tung, String bekanntNameDerHaltestelle)[18] überprüft, ob der Wege-Array für einen neuen Weg vergrößert werden muss oder ein bestehender Weg ersetzt wird.[19]
Nachdem nun der Weg zur Weiche umfassend untersucht und gespeichert wurde, werden jetzt die potenziell bis zu zwei weiteren Wege, die durch die Weiche entstanden sind, untersucht. Um zu überprüfen, ob es sich um einen oder zwei zu überprüfende Wege handelt, gibt die Methode mglAusgaengeWeiche(Actor weiche, char eingangsRichtung)[20] einen String mit allen möglichen Ausgängen der Weiche bei gegebener Eingangsrichtung an. Dieser hat entweder eine oder zwei Stellen, je nach möglichen Ausgängen.
Für jede der möglichen Richtungen wird nun hinzufuegenWegeWeichen(...) neu aufgerufen. Dabei ist die aktuelle Weiche der neue Startblock und die aktuelle Schrittzahl sowie der String-Array mit den bekannten Weichen werden übergeben.

3.2.3 Freigabe erteilen

Bevor Signale auf Grün gestellt werden können und Züge Haltestellen oder Prellböcke verlassen dürfen, muss es dafür eine Freigabe geben. Diese muss in allen Fällen das Gleiche überprüfen: Ist das Schienennetz bis zum nächsten roten Signal, bis zur nächsten Haltestelle oder bis zum nächsten Prellbock korrekt gebaut und sind alle Weichen richtig gestellt? Zusätzlich gilt es zu überprüfen, ob kein Zug auf der kommenden Strecke steht. Die Methode getFreigabe(Actor start, char ausgangsrichtung, String zielhaltestelle)[21] gibt für jeden möglichen Fall einen anderen String zurück. Sollte es ein Problem beim Freigabeprozess geben, wird der Nutzer entsprechend darauf aufmerksam gemacht und kann dieses möglicherweise beheben. Wenn der Weg bis zum nächsten roten Signal, der nächsten Haltestelle oder dem nächsten Prellbock frei ist und alle Weichen bis dahin richtig gestellt sind, wird der String “0” zurückgegeben. Sollte eine Weiche gestellt werden müssen, gibt der String “1, x-Position; y-Position der Weiche” an, welche Weiche gestellt werden muss. Analog gibt der String “2, x-Position; y-Position des Zugs” an, welcher Block besetzt ist und weshalb der Zug keine Freigabe bekommt. Sollte die Strecke fehlerhaft sein, also zum Beispiel eine Sackgasse ohne Prellbock die Strecke beenden, wird der String “3, x-Position; y-Position des letzten Blocks” zurückgegeben.
Um den Weg zu überprüfen, wird zunächst der nächste Block in der geforderten Richtung gesucht und die Eingangsrichtung in diesen gespeichert. Sollte dieser Block existieren und einen passenden Eingang haben, wird die Methode checkFreigabe(Actor start, char

[18] Siehe Anhang A.2.3 S. 65
[19] Siehe Anhang A.2.3 S. 66f.
[20] Siehe Anhang A.2.3 S. 64f.
[21] Siehe Anhang A.2.5 S. 77f.

eingangsRichtung, String zielhaltestelle, String[] merker)[22] in der Klasse Block aufgerufen. Diese Methode soll alle Blöcke bis zum nächsten Signal, bis zur nächsten Haltestelle oder bis zum nächsten Prellbock überprüfen. Dabei geht es zum einen darum, ob die Blöcke besetzt sind, und zum anderen um die richtige Weichenstellung sowie eine korrekte Aneinanderreihung der Blöcke beim Streckenbau. Die Methode bekommt den aktuell zu überprüfenden Block, die Eingangsrichtung in diesen sowie den Namen der Zielhaltestelle und einen String-Array merker[] mit den bereits bekannten Weichen, ähnlich zur wegeFestlegen(...)-Methode, übergeben.

Zu Beginn der Methode wird überprüft, ob der aktuelle Block unabhängig von der Weichenstellung einen passenden Eingang besitzt. Sollte dies nicht der Fall sein, wird die passende Fehlermeldung für eine fehlerhafte Strecke zurückgegeben. Ansonsten wird untersucht, ob der Block besetzt ist. Jeder Block besitzt einen Boolean „besetzt“, der, wenn ein Zug auf ihm steht, true und ansonsten false ist. Wie dieser Wert beim Fahren des Zugs auf jedem Block geändert wird, wird im folgenden Abschnitt erklärt, wenn auf das Fahren zu angegebenen Haltestellen eingegangen wird. Hier genügt aber eine einfache Abfrage nach dem Zustand des Blocks. Sollte der Block besetzt sein, wird die Fehlermeldung 2 für besetzte Schienen zurückgegeben.

Im Folgenden wird zunächst überprüft, ob der Zug vor einem roten Signal steht. Dazu wird die Hilfsmethode istRotesSignal(Actor mglsignal, char eingangsRichtung)[23] aufgerufen. Da mehrere Signale in verschiedenen Richtungen an einem Block platziert werden können, werden mit dem Befehl getObjectsAtOffset(int dx, int dy, Class cls) alle Signale, die sich an diesem Block befinden, in einer Liste gespeichert. Sollte diese Liste leer sein, wird direkt der Boolean false zurückgegeben, da sich kein Signal an diesem Block befindet. Ansonsten wird die Liste mit einer for-Schleife durchlaufen und jeder Eintrag wird überprüft. Dafür wird der Methode checkRotesSignal(Actor signal, char richtung)[24] aus der Klasse Signal das zu überprüfende Signal samt Blockausgangsrichtung übergeben. Sollte es sich um ein für die Richtung relevantes Signal halten, gibt die Methode checkRotesSignal(...) und damit auch die Methode istRotesSignal(...) den Boolean true zurück. Sollte man bei der Streckenüberprüfung vor einem roten Signal ankommen, so wird die Freigabe erteilt, da der Zug zumindest bis zu diesem Signal freie Fahrt hat. Für eine Weiterfahrt von diesem Signal muss erneut um Freigabe gebeten werden.

Ist kein Signal vorhanden, gilt es nun zu überprüfen, ob der Block ein Prellbock ist oder

[22] Siehe Anhang A.2.3 S. 47ff.
[23] Siehe Anhang A.2.3 S. 43
[24] Siehe Anhang A.2.5 S. 76

ob eine Haltestelle an diesem Block vorhanden ist. In beiden Fällen würde die Freigabe erteilt werden, da der Weg bis zu diesem Block kontrolliert und für den Zug als befahrbar befunden wurde. Für eine Weiterfahrt muss wie beim roten Signal dann wieder die Freigabe erteilt werden.

Sollte bei der Überprüfung weder eine Haltestelle noch ein Prellbock erreicht worden sein, wird nun die Blockausgangsrichtung ermittelt. Die Hilfsmethode getBlockRichtung(Actor block, char eingangsR)[25] gibt die Ausgangsrichtung des Blocks bei gegebener Eingangsrichtung zurück. Sollte der Block eine Weiche sein, die nicht für ein Anfahren aus der Eingangsrichtung gestellt ist, so gibt die Methode den char 'H' zurück. Dann gilt es zu kontrollieren, ob die Weiche bereits im Freigabeprozess überprüft wurde. Dafür wird der der Methode übergebene String-Array merker[] per for-Schleife durchlaufen und überprüft, ob die Weiche darin schon gespeichert ist. Dabei ist die Richtung irrelevant, da es nicht möglich ist, eine Weiche zweimal von der gleichen Richtung anzufahren, ohne zuvor eine Weiche von zwei verschiedenen Richtungen angefahren zu haben oder wieder über den Startblock gefahren zu sein. Somit würde dann die zweite Weiche gar nicht mehr getestet werden, da die Methode schon bei der ersten doppelt angefahrenen Weiche einen Fehler melden würde. Dieser Fehler muss gemeldet werden, da, um das Entgleisen eines Zugs zu vermeiden, eine Weiche nicht während der Fahrt gestellt werden soll, sondern immer vor der Abfahrt beziehungsweise vor der Weiterfahrt. Im Normalfall haben Schienennetze daher in einem gewissen Abstand vor Weichen ein Signal stehen, sodass der Zug dort auf richtig gestellte Weichen warten kann. Wenn kein Eintrag im String-Array merker[] die Koordinaten der zu überprüfenden Weiche enthält, wird mit der Methode der Programmnutzer gefragt, ob er die Weiche stellen möchte. Dies geschieht in der Hilfsmethode userFrage(String fragetyp)[26], die für alle Fragen an den Nutzer während der Fahrt des Zugs genutzt wird. Wird dies bejaht, wird die Weiche mit der Methode stelleDieWeiche(Actor g)[27] aus der Klasse Weiche gestellt. Zum einen wird dabei der char „stellung", den jede Weiche besitzt, von 'k' wie Kurve auf 'g' wie Gerade oder andersherum gestellt, zum anderen wird das Bild, das Greenfoot für diese Weiche nutzt, geändert. Jeder Weichentyp hat jeweils ein Bild für jede Weichenstellung. Die Bildnamen („Weichentyp_stellung ") wurden dabei so gewählt, dass man nicht für jeden Weichentyp eine eigene Methode benötigt.

Ist die Weiche gestellt, kann der Freigabeprozess nun fortgesetzt werden. Die Methode

[25] Siehe Anhang A.2.3 S. 39f.
[26] Siehe Anhang A.2.3 S. 38f.
[27] Siehe Anhang A.2.3 S. 62f.

wird dafür neu aufgerufen und dabei die nun gestellte Weiche als Startblock übergeben. Stellt der Nutzer die Weiche jedoch nicht, wird keine Freigabe erteilt und die entsprechende Fehlermeldung zurückgegeben. Damit wäre der Fall einer für die Eingangsrichtung falsch gestellten Weiche abgehandelt.
Anschließend wird der aktuelle Block daraufhin überprüft, ob er eine Weiche ist. Ist er es nicht, wird mit der getnextBlock(int aktuellesX, int aktuellesY, char richtung)-Methode[28] der nächste Block in der Ausgangsrichtung des Blocks gesucht. Existiert dieser, wird die Methode checkFreigabe(...) mit dem nächsten Block als Startblock neu aufgerufen, ansonsten wird die Fehlermeldung für eine Sackgasse ohne Prellbock zurückgegeben.
Sollte es sich beim aktuellen Block aber um eine Weiche handeln, gilt es nun mehrere Dinge zu berücksichtigen. Dabei ist zu unterscheiden, ob die checkFreigabe(...)-Methode eine Zielhaltestelle übergeben bekommen hat oder ob stattdessen „null“ übergeben wurde. Zunächst betrachten wir, was getan wird, wenn keine Zielhaltestelle angegeben wurde. Abhängig von der aktuellen Weichenstellung wird der nächste Block in Blockausgangsrichtung gesucht. Dazu wird wieder die getnextBlock(...)-Methode genutzt. Sollte dieser Block existieren, wird die aktuelle Weichenposition samt Eingangsrichtnug im Array-String merker[] gespeichert und die checkFreigabe()-Methode mit dem nächsten Block als Startblock wieder aufgerufen. Ansonsten wird die Fehlermeldung für Sackgassen ohne Prellbock zurückgegeben.
Ist eine Zielhaltestelle angegeben, gilt es zu überprüfen, ob die Weiche so gestellt ist, dass mit dem eingestellten Weg die Zielhaltestelle zu erreichen ist. Zusätzlich soll festgestellt werden, ob, sollte es die Möglichkeit geben, eine Weichenstellung einen kürzeren Weg zur Zielhaltestelle zur Folge hätte.
Von der Methode mglAusgaengeWeiche(Actor weiche, char eingangsRichtung)[29] aus der Klasse Weiche erhält man abhängig vom Weichentyp und der Eingangsrichtung einen String mit einer oder zwei Stellen. Der String gibt die möglichen Ausgangsrichtungen an, die die Weiche je nach Stellung besitzt. Mit der Methode getWeicheRichtung(Actor a, char s),[30] ebenfalls aus der Klasse Weiche, erhält man einen char, der die aktuelle Ausgangsrichtung abhängig von der Weichenstellung angibt. Nun gilt es erneut, zwei Fälle zu unterscheiden. Entweder gibt es zwei mögliche Wege, die jeweils zum Zielbahnhof führen könnten, oder nur einen potenziellen Weg.
Sollte die Weiche nur eine Ausgangsrichtung zulassen, wird zunächst überprüft, ob man

[28] Siehe Anhang A.2.3 S. 40ff.
[29] Siehe Anhang A.2.3 S. 64f.
[30] Siehe Anhang A.2.3 S. 63

in dieser Richtung zur Zielhaltestelle kommen kann. Dazu wird mit der sich in der Klasse Weiche befindenden Methode getWeg(char ausgangsRichtung, String nameDerHaltestelle)[31] abgeglichen, ob in der Weiche ein passender Weg zur Zielhaltestelle gespeichert ist. Die Methode gibt dann diesen Weg als String in der aus der Methode wegeFestlegen(...) bekannten Form ”Richtung in die man die Weiche verlassen muss Haltestellenname; Anzahl der Schritte” zurück. Gibt es in dieser Richtung keinen Weg, wird „null“ zurückgegeben. Gibt die Methode einen passenden Weg zurück, so wird die Weichenposition samt Richtung im String-Array merker[] gespeichert. Nun wird die Methode checkFreigabe(...) mit dem nächsten Block in Ausgangsrichtung als Startblock aufgerufen. Sollte der Weichenausgang nicht zur Zielhaltestelle führen, so wird die Fehlermeldung eines unvollständigen Schienennetzes zurückgegeben.

Im Falle von zwei möglichen Ausgängen der Weiche wird für beide Richtungen mit der getWeg(...)-Methode überprüft, ob diese Wege zur Zielhaltestelle führen. Sollten nicht beide Wege zum Ziel führen, wird kontrolliert, ob es einer der beiden Wege tut oder keiner. Führt keiner der beiden Wege zum Ziel, wird analog zum eben beschriebenen Fall die Fehlermeldung eines unvollständigen Schienennetzes zurückgegeben. Sollte eine der beiden Richtungen aber einem Weg zum Ziel folgen, wird abgeglichen, ob die Weiche in diese Richtung gestellt ist. Bei korrekter Weichenstellung wird die Weichenposition im String-Array merker[] gespeichert und die Methode checkFreigabe(...) wird mit dem passenden nächsten Block aufgerufen. Ansonsten wird der Programmnutzer um Weichenumstellung gebeten. Sollte diese erfolgen, wird die Methode mit nun korrekt gestellter Weiche erneut aufgerufen. Andernfalls wird die passende Fehlermeldung für falsch gestellte Weichen zurückgegeben. Somit sind alle Möglichkeiten für Weichen mit genau einem passenden Weg zur Zielhaltestelle abgehandelt.

Der letzte übrige Fall ist eine Weiche mit zwei möglichen Ausgangsrichtungen, die beide zur Zielhaltestelle führen. In diesem Fall wird die jeweilige Länge der Wege als Substring der von der Methode getWeg(...) bekommenen Wege ausgelesen und verglichen. Für den jeweils kürzeren Weg wird dann überprüft, ob die Weiche entsprechend gestellt ist. Ist dies der Fall, wird die Weichenposition im String-Array merker[] gespeichert und die Methode checkFreigabe(...) mit dem kommenden Block in Ausgangsrichtung aufgerufen. Bei Weichen, die für den längeren Weg gestellt sind, wird zunächst überprüft, ob sie bereits in diesem Freigabeprozess geprüft wurden. Beim kürzeren Weg war dies eben nicht nötig, da eine Weiche auf kürzestem Wege nicht zweimal angefahren werden kann,

[31] Siehe Anhang A.2.3 S. 66

ohne dass dazwischen ein Prellbock liegt, der für das Rangieren des Zuges nötig ist. Dieser würde aber den Freigabeprozess beenden. Sollte die Weiche noch unbekannt sein, wird ihre Position im String-Array merker[] gespeichert. Anschließend wird mit Greenfoot.ask(...) gefragt, ob die Weiche für einen kürzeren Weg gestellt werden soll. Je nach Antwort wird die Weiche gestellt oder nicht und die Methode checkFreigabe(...) mit dem entsprechenden nächsten Block je nach Ausgangsrichtung aufgerufen.
Sollte die Weiche jedoch bekannt sein, wird die Fehlermeldung einer fehlerhaften Strecke zurückgegeben. Eine Weiche darf nicht zweimal in einem Freigabeprozess überprüft werden. Dies ist nämlich nur dann möglich, wenn die Weiche von zwei verschiedenen Richtungen angefahren wird. Da aber während der Fahrt eine Weiche nicht gestellt werden soll, kann auch keine Freigabe erteilt werden. Aus derselben Richtung kann eine Weiche nur angefahren werden, wenn zuvor eine Weiche aus zwei verschiedenen Richtungen angefahren wurde oder der Zug wieder über den Startblock fuhr. In beiden Fällen wäre der Freigabeprozess also schon vorher abgebrochen worden, sodass diese Fälle nicht beachtet werden müssen.
Damit sind nun alle Möglichkeiten, die beim Freigabeprozess untersucht werden müssen, überprüft. Die Methode checkFreigabe(...) ruft sich solange rekursiv auf, bis entweder die Freigabe erteilt werden kann oder eine Fehlermeldung zurückgegeben wird. Die Methode getFreigabe(...) gibt den von der Methode checkFreigabe(...) erhaltenen String an die aufrufenden Methoden zurück. Einzig der Fall der besetzten Schiene wird noch einmal überprüft. Soll ein Zug ohne Zielhaltestelle im Kreis fahren, so wird der Block, an dem der Zug um Freigabe bittet, als besetzt angezeigt. Um in diesem speziellen Fall dennoch die Freigabe zu gewähren, wird bei der Fehlermeldung eines besetzten Blocks überprüft, ob es sich beim besetzten Block um den Block handelt, auf dem der Zug gerade steht. Sollte dies der Fall sein, wird ebenfalls Freigabe gewährt.

3.2.4 Zug zu einer Zielhaltestelle fahren lassen

Mit Hilfe der beiden zuvor beschriebenen Methoden soll es nun gelingen, einen Zug zu einer bestimmten Haltestelle fahren zu lassen. Die Methode fahreZuHaltestelle(char startRichtung, String nameZielhaltestelle)[32] benötigt dazu eine Richtung, in die der Zug zu Beginn fahren soll sowie den Namen der Zielhaltestelle. Den Namen jeder Haltestelle kann man sich in der „Greenfoot-Welt“ mit der Methode toggleNamenAnzeigen()[33] aus der Klasse Haltestelle anzeigen lassen. Um die Korrektheit der Eingabe zu überprüfen,

[32] Siehe Anhang A.2.6 S. 86ff.
[33] Siehe Anhang A.2.4 S. 72

wird mit der Hilfsmethode getHaltestelle(String name)[34] nach dem passenden Objekt in der „Greenfoot-Welt“ gesucht. Der Befehl getObjects(Class class) gibt eine Liste aller Objekte der gewünschten Klasse in der Welt zurück. Sollte in dieser Liste eine Haltestelle mit dem passenden Namen vorhanden sein, wird diese zurückgegeben. Ansonsten ist der Returnwert „null“. In diesem Fall wird der Nutzer mit einer Nachricht auf seine fehlerhafte Eingabe hingewiesen. Bei korrekter Namensangabe wird für die angestrebte Haltestelle die Methode wegeFestlegen() aufgerufen, sodass alle gespeicherten Wege in den Weichen auf dem Stand des aktuellen Streckennetzes sind.
Nachdem nun die Existenz der Haltestelle geklärt ist und die Weichen die aktuellen Wege zu der Haltestelle gespeichert haben, muss der Zug noch eine Freigabe zur Abfahrt bekommen, bevor er dann losfahren darf. Bevor die checkFreigabe(...)-Methode aber aufgerufen werden kann, wird zuerst überprüft, ob der aktuelle Block, auf dem der Zug steht, und der nächste Block in gegebener Richtung eine zusammenhängende Schiene bilden. Sollte dabei der nächste Block eine falsch gestellte Weiche sein, wird um das Stellen der Weiche gebeten und entsprechend der Fahrtprozess fortgeführt oder abgebrochen. Sollte der Zug vor einem roten Signal stehen, wird mit der getFreigabe(...)-Methode aus der Klasse Signal um Freigabe gebeten, sonst geschieht dies mit der checkFreigabe(...)-Methode aus der Klasse Block. Der von der Methode checkFreigabe(...) zurückgegebene String wird danach mithilfe einer Switch-Case-Verzweigung für alle Fehlercodes durchgegangen. Je nach Fehlercode wird dabei dann die passende Meldung ausgegeben. Einzig der Fehlercode “2,...” für besetzte Schienen wird erneut überprüft um anzugeben, ob der Zug möglicherweise im Kreis fahren würde - in diesem Fall ohne die Möglichkeit ihn anzuhalten - oder ob ein anderer Schienenblock besetzt ist.
Mit erhaltener Freigabe wird gegebenenfalls das Signal auf Grün gestellt und die Methode fahreHaltestelle(char richtung, String nameZielhaltestelle) aufgerufen. Mit einer Switch-Case-Verzweigung werden dort alle vier Himmelsrichtungen durchgegangen. Der Zug wird dann auf den zur Richtung passenden nächsten Block gesetzt. Dabei wird auch der besetzt-Zustand der Blöcke beachtet. Vor dem Verlassen des aktuellen Blocks wird besetzt auf false gesetzt und entsprechend wird beim nächsten Block besetzt auf true gestellt. Nach einem einsekündigen Verzögern mithilfe der Grennfoot.delay(int i)-Methode, das dafür sorgt, dass man den Zug während seiner Fahrt auf allen Blöcken, die er befährt, auch sehen kann, wird die Methode fahreSchleifeZurHaltestelle(char eintrittsRichtung, String nameZielhaltestelle)[35] aufgerufen.

[34] Siehe Anhang A.2.6 S. 85
[35] Siehe Anhang A.2.6 S. 89ff.

Bevor diese Methode sich dem aktuellen Block samt seinen Problemen widmet, wird zunächst überprüft, ob im Block zuvor ein Signal auf Grün gestellt wurde. In einem solchen Fall wird dieses Signal wieder auf Rot gestellt, da nachfolgende Züge erneut um Freigabe an diesem Signal bitten sollen. Da mehrere Signale an einem Block stehen können, gilt es wieder, alle möglichen Signale an diesem Block auf ihre Richtung zu kontrollieren, um nicht versehentlich ein zuvor irrelevantes Signal von Rot auf Grün zu stellen. Die Methode fahreSchleifeZurHaltestelle(...) wird auch von der Methode fahreEwigRichtung(char fahrtRichtung),[36] die analog zur Methode fahreZuHaltestelle(...) funktioniert, aufgerufen. Einziger Unterschied ist die nicht vorhandene Zielhaltestelle. Daher prüft die Methode fahreSchleifeZurHaltestelle(...) nun zunächst, ob eine Zielhaltestelle angegeben wurde oder nicht. In diesem Abschnitt wird nur der Fall einer gegebenen Zielhaltestelle betrachtet.
Sollte sich der Zug an einer Haltestelle befinden, wird ihr Name mit dem der Zielhaltestelle verglichen. Wurde das Ziel erreicht, wird eine entsprechende Nachricht ausgegeben, ansonsten wird, falls der Zug an einer anderen Haltestelle steht, um Freigabe gebeten. Sollte diese nicht erteilt werden, wird eine entsprechende Fehlermeldung ausgegeben. Befindet sich der Zug an keiner Haltestelle oder hat er die entsprechende Freigabe bekommen, wird überprüft, ob der Zug an einem roten Signal steht, welches, sollte die entsprechende Freigabe erteilt werden, auf Grün gestellt wird. Ebenfalls wird untersucht, ob der Zug in einer Sackgasse steht oder an einem Prellbock. Im Fall einer Sackgasse wird eine Fehlermeldung zurückgegeben. Steht der Zug an einem Prellbock, wird gefragt, ob er wieder in entgegengesetzter Richtung zurückfahren soll. Sind all diese Fälle ausgeschlossen, so wird erneut fahreHaltestelle(...) aufgerufen, der Zug darf auf den nächsten Block fahren und die Überprüfung beginnt erneut. Weichen werden in der Methode fahreSchleifeZurHaltestelle(...) nicht extra betrachtet, da diese im Freigabeprozess bereits gestellt werden und somit hier wie „normale“ Blöcke mit zwei Richtungen angesehen werden können.

3.2.5 Zug ohne Zielhaltestelle fahren lassen

Im Abschnitt zur Methode fahreZuHaltestelle(...) wurde bereits kurz die Methode fahreEwigRichtung(char fahrtRichtung)[36] angesprochen. Beide Methoden fuktionieren sehr ähnlich. Bis zum Aufruf der Methode fahreSchleifeZurHaltestelle(...) unterscheiden sich die Methoden lediglich darin, dass in der fahreEwigRichtung(...)-Methode die Zielhalte-

[36] Siehe Anhang A.2.6 S. 95ff.

stelle nicht überprüft wird und auch die Wege in den Weichen nicht neu gespeichert werden.
Die Methode fahreSchleifeZurHaltestelle(...) unterscheidet sich im Falle einer nicht vorhandenen Zielhaltestelle in folgenden Dingen: Zum einen wird bei jeder erreichten Haltestelle gefragt, ob der Zug hier stehen bleiben soll, zum anderen wird im Freigabeprozess ebenfalls keine Zielhaltestelle angegeben und somit keine Weiche gestellt, die nicht für die Weiterfahrt notwendigerweise gestellt werden muss.

3.2.6 Mehrere Züge gleichzeitig fahren lassen

Mit den bislang gezeigten Methoden (fahreZuHaltestelle(...) und fahreEwigRichtung(...)) ist es leider nicht möglich, mehrere Züge gleichzeitig fahren zu lassen. Dies ist nur umsetzbar, wenn die Methoden in der act-Methode der Klasse Zug aufgerufen werden können. Die beiden gezeigten Methoden rufen sich jedoch immer wieder rekursiv auf, sodass immer ein Zug seine Fahrt beenden muss, bevor der nächste fahren kann. Mit der fahreBlock(...)-Methode ließe sich der Zug immer um einen Schienenblock weiterbewegen, allerdings müsste für jeden Schritt die Richtung angegeben werden. Außerdem kann die Methode keine Zielhaltestelle verarbeiten.
Um diese Probleme zu lösen, bekommt die Klasse Zug zwei weitere Attribute. Einen char, der immer die Eingangsrichtung des Zuges in den aktuellen Block speichert, sowie einen String, der die Zielhaltestelle des Zugs beinhaltet. Somit kann nach einmaliger Eingabe der Startrichtung und der Zielhaltestelle die Methode fahreBlock(...) ohne erneute Richtungsangabe aufgerufen werden. Damit auch weiterhin die Möglichkeit besteht, zu einer Zielhaltestelle zu fahren, wird die Vorgehensweise der Methode fahreZuHaltestelle(...) für die neue Methode actfahre()[37] genutzt. Einzig die Rekursivität wird nicht übernommen. Statt die fahreSchleifeZurHaltestelle(...)-Methode aufzurufen, wird lediglich fahreBlock(...) genutzt und das Attribut Eingangsrichtung des Zuges angepasst. Analog zu fahreEwigRichtung(...) können Züge aber auch ohne Zielhaltestelle fahren. Dafür muss lediglich die Frage der Zielhaltestelle mit „0“ beantwortet werden. Ist gewünscht, dass der Zug an einer Haltestelle, einem Prellbock oder vor einem Signal halten und nicht weiterfahren soll, so wird der char, der die Eingangsrichtung angibt, auf ‘A’ für „Anhalten“ gesetzt und der Zug fährt nicht mehr weiter. Mit der Methode resetZug()[38] ist es allerdings möglich, die beiden neuen Attribute zurückzusetzen, sodass sie bei erneutem Aufrufen der act-Methode neu festgelegt werden können.

[37] Siehe Anhang A.2.6 S. 99ff.
[38] Siehe Anhang A.2.6 S. 81

4 Befragung: Kontext Eisenbahn vs. Planetenerkundung

Um herauszufinden, ob sich der Kontext Eisenbahn und insbesondere das Gleisbildstellpult zur Einführung anbietet, wurden im Rahmen dieser Arbeit drei Lehrerinnen und Lehrer befragt, die in ihren Schulen mit dem Schulbuch „Informatik 1“ und dem darin enthaltenen Szenario der Planetenerkundung mit dem Marsrover arbeiten.[39]
Dabei wurde zum einen der Kontext Eisenbahn und das Konzept Gleisbildstellpult hinterfragt und dem Konzept des Marsrovers gegenübergestellt, zum anderen wurde aber auch gefragt, ob das im Konzept Gleisbildstellpult vorgesehene Vorziehen des Klassen- und Sequenzdiagramms Zustimmung findet und ob die Befragten sich mit der Idee eines langfristig tragbaren Kontexts anfreunden können oder diese eher ablehnen.

4.1 Auswertung

Auch wenn klar ist, dass eine Befragung in diesem Rahmen kein wirklich repräsentatives Bild ergibt, so lassen sich dennoch einige Rückschlüsse für das Projekt ziehen.
Zunächst lässt sich festhalten, dass der Kontext Eisenbahn aufgrund seiner Nähe zum Alltag der Schülerinnen und Schüler als durchaus motivierend angesehen wurde, das Konzept Gleisbildstellpult allerdings mit ein paar Abstrichen. Einig waren sich alle drei Befragten darin, dass sich das Gleisbildstellpult zur Einführung von Objekten und Klassen, Methoden und Algorithmen eignet. Die Komplexität des Klassendiagramms wurde von zwei Befragten als möglicherweise zu hoch erachtet. Das Thema Sequenzdiagramm wurde sehr unterschiedlich bewertet. Einmal wurde es konsequent abgelehnt, da es im Kernlehrplan nicht gefordert wird. Ein anderes Mal hielt man es im Konzept Gleisbildstellpult für geeignet, während in der dritten Rückmeldung die Eignung aufgrund zu hoher Komplexität infrage gestellt wurde.
Im Vergleich zum Kontext Planetenerkundung wird der Kontext Eisenbahn in den oben genannten fünf Unterthemen nur selten bevorzugt. Hauptkritikpunkte sind dabei die Vielzahl an Klassen, schlechtere Grafik und die höhere Einstiegshürde. Einzig die Einführung von Methoden wird im Kontext Eisenbahn zweimal ebenso gut bewertet wie im Kontext Planetenerkundung. Die zusätzlichen Möglichkeiten zur kreativen Entfaltung im Kontext Eisenbahn werden insbesondere für starke Schülerinnen und Schüler als Vorteil gesehen. Zwei der drei Befragten sehen allerdings den Grundlagenfokus in der reduzierten Welt des Kontexts Planetenerkundung für weniger starke Schülerinnen und Schüler als einen Vorteil an.

[39] Siehe Anhang A.3 S.111ff.

Auf die Frage, ob man das Sequenz- und Klassendiagramm vor den Algorithmen behandeln soll (vgl. Reihenfolge in Abschnitt 2 dieser Arbeit), gab es unterschiedliche Antworten. Mehrheitlich hielt man es für sinnvoll, das Klassendiagramm vorzuziehen. Eine/r der Befragten sah zusätzlich auch das Vorziehen des Sequenzdiagramms als Vorteil an, da so den Schülerinnen und Schülern der Kerngedanke der Objektorientierung frühzeitig beigebracht wird. Die anderen beiden Befragten lehnen das Vorziehen des Sequenzdiagramms aufgrund von möglichen Verständnisproblemen der Schülerinnen und Schüler bzw. Nichtvorhandenseins im Kernlehrplan ab.
Die Frage, wie vorteilhaft das Arbeiten mit einem Kontext über unterschiedlich lange Zeiträume gesehen wird, ergab übereinstimmende Zustimmung für eine Unterrichtsreihe und etwas abgeschwächt auch noch für ein Quartal. Für längere Zeiträume nimmt die Begeisterung aber immer weiter ab.
Zuammenfassend kann man sagen, dass die befragten Lehrerinnen und Lehrer sehr gern mit dem Kontext der Planetenerkundung arbeiten. Dennoch konnte auch das Konzept des Gleisbildstellpults im Kontext Eisenbahn in einigen Aspekten überzeugen.

5 Schlussbetrachtung

Das Projekt LoCo möchte zu allen Themenbereichen des Informatikunterrichts Bausteine im Kontext Eisenbahn anbieten. In dieser Arbeit wurde ein Teil des Bausteins zur Einführung in die Objektorientierung beleuchtet. Dabei ließ sich feststellen, dass das Konzept Gleisbildstellpult sich durchaus eignet, alle für die Einführung in die Objektorientierung benötigten Themen im Kontext Eisenbahn zu behandeln. Vor allem durch die zusätzlichen Möglichkeiten der Behandlung von Klassen- und Sequenzdiagrammen im selben Kontext und der kreativen Entfaltung setzt es sich vom aktuell verbreiteten Konzept des Marsrovers ab, wobei in der Resonanz der Lehrerinnen und Lehrer die Vermutung geäußert wurde, das dies eher nur für stärkere Schülerinnen und Schüler von Vorteil wäre. Hier gilt es noch zu prüfen, ob man dem mit abgeschwächten Varianten entgegenwirken kann. Insgesamt halte ich das Konzept der Einführung in die Objektorientierung anhand des Gleisbildstellpults für eine sinnvolle Alternative, von der bei richtiger Ausführung alle Schülerinnen und Schüler profitieren können, da ich die Alltagsnähe und die Möglichkeiten zur kreativen Entfaltung als große Stärken ansehe. Zusätzlich könnte die im Projekt LoCo geplante Verwirklichung einer Verbindung des Programms mit einer Modelleisenbahn die Begeisterung bei Schülerinnen und Schülern deutlich erhöhen.

Literatur

Deutsche Bahn AG. (2021). Integrierter Bericht 2021. *(abgerufen am 30.08.2022)* *https://ibir.deutschebahn.com/2021/fileadmin/pdf/DB_IB21_web_01.pdf*.

Diethelm, I., Koubek, J. & Witten, H. (2011). IniK - Informatik im Kontext - Entwicklungen, Merkmale und Perspektiven. *LOG IN (169/170), S. 97-105*.

Elster, D. (2006). *Kontexte als Strukturelemente des Unterrichts. Handreichungen für die Unterrichtspraxis im Projekt Biologie im Kontext. (abgerufen am 30.08.2022)* *https://www.researchgate.net/profile/Doris-Elster/publication/271845192_Handreichung_fur_die_Praxis_Kontexte_als_Strukturelemente_des_Unterrichts/links/54d487a80cf246475805ff46/Handreichung-fuer-die-Praxis-Kontexte-als-Strukturelemente-des-Unterrichts.pdf*.

Informatik 1. (2014). Schoeningh Verlag.

Kernlehrplan Biologie. (2019). Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen Biologie. Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen (Hrsg.) (abgerufen am 30.08.2022) https://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/bi/KLP_GOSt_Biologie.pdf.

Kernlehrplan Informatik. (2014). Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen Informatik. Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen (Hrsg.) (abgerufen am 30.08.2022) https://www.schulentwicklung.nrw.de/lehrplaene/upload/klp_SII/if/KLP_GOSt_Informatik.pdf.

Koubek, J., Schulte, C., Schulze, P. & Witten, H. (2009). *Informatik im Kontext (IniK), (abgerufen am 30.08.2022)* *http://www.informatik-im-kontext.de/*.

Mahnke, F. (2021). *Konzeptionierung einer Design-Based Research Studie zur Motivationsförderung durch den Kontext Eisenbahn im Rahmen der Behandlung von Datenbanken im Informatikunterricht der Oberstufe.* Masterarbeit (Westfälische Wilhelms Universität Münster, Institut für Didaktik der Mathematik und der Informatik, Arbeitsbereich Didaktik der Informatik) (abgerufen am 30.08.2022) https://www.uni-muenster.de/imperia/md/content/idmi/ag-thomas/publikationen/2021_mahnke_eisenbahn_datenbank_masterarbeit.pdf.

Thomas, M. (2021). Eisenbahn im kontextorientierten Informatikunterricht. *(abgerufen am 30.08.2022)* *https://www.uni-muenster.de/imperia/md/content/*

idmi/ag-thomas/publikationen/2021_eisenbahn_im_kontextorientierten_informatik_unterricht_-_juni_2021_-_marco_thomas.pdf.

Thomas, M. (2022). *Baustein 001 - Das Stellwerk.* Projekt LoCo - Eisenbahn als informatischer Kontext - locomotive control.

Uhlenbrock Elektronik. (2020). *Track-Control, (abgerufen am 30.08.2022)* *https://www.uhlenbrock.de/de_DE/produkte/trackcon/index.htm.*

A Anhang

A.1 Beispiel für ein mögliches Klassendiagramm

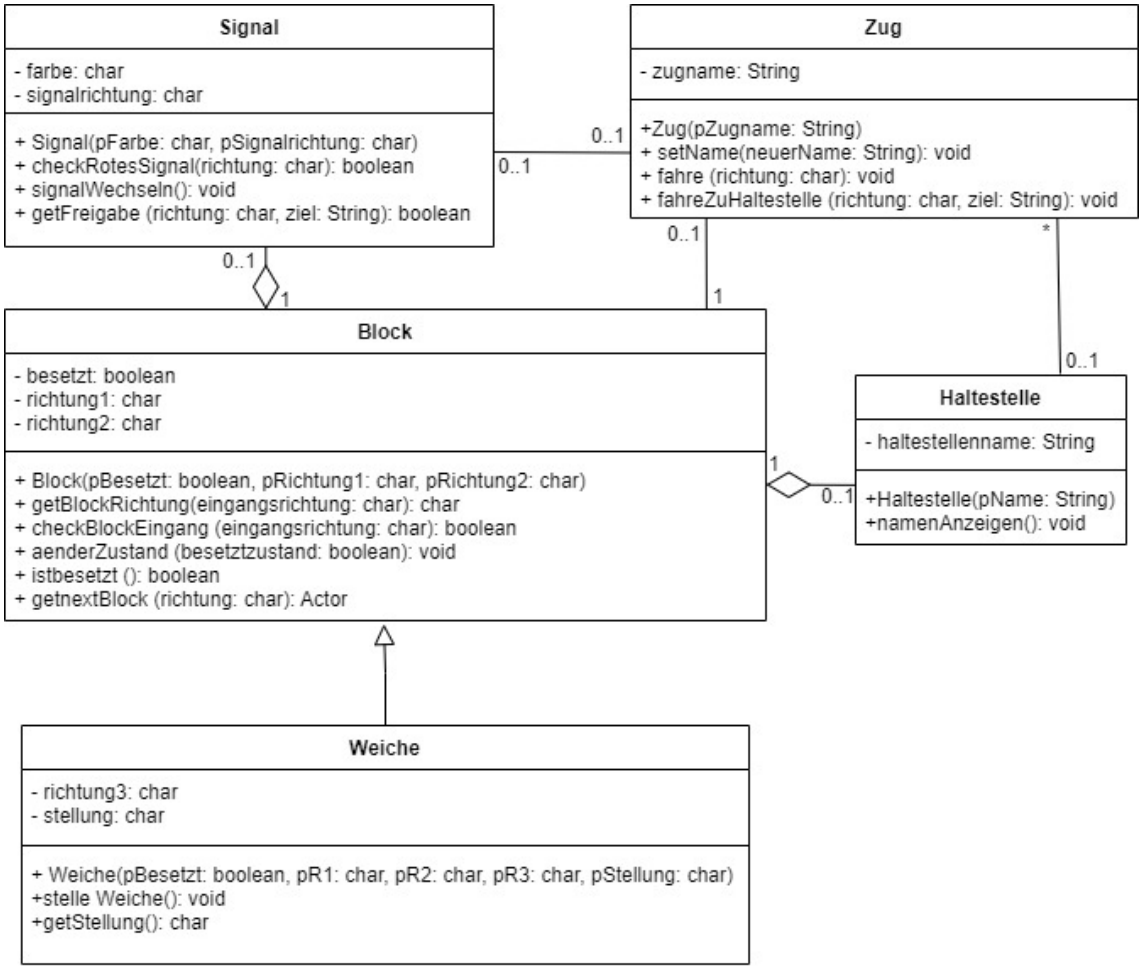

Abbildung 5: Beispiel für ein mögliches Klassendiagramm

A.2 Programm digitales Gleisbildstellpult

A.2.1 Greenfoot-Welt

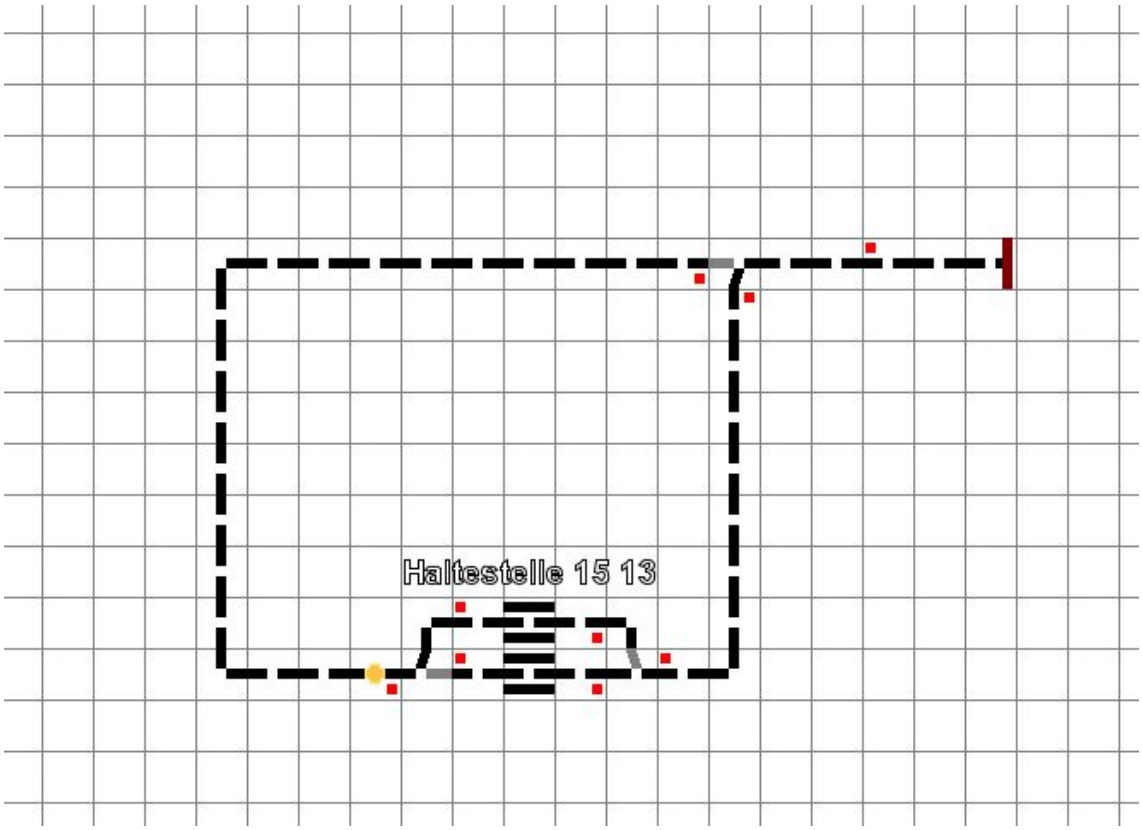

Abbildung 6: Ausschnitt aus der Greenfoot-Welt inklusive Beispielschienennetz

A.2.2 MyWorld

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   * Write a description of class MyWorld here.
5   *
6   * @author Niklas Nogossek
7   *
8   */
9  public class MyWorld extends World
10 {
11     /**
12      * Create a new world with 50x25cells and.
13      * with a cellsize of 30x30 px
14      */
15     public MyWorld()
16     {
17         super(50, 25, 30);
18         setBackground("cell_weiss30.jpg");
19         setPaintOrder(Zug.class, Block.class, Signal.class, Haltestelle
               .class);  //Darstellungsreihenfolge der Klassen
```

```
20      }
21
22      public void beispielSchienennetz()  //erzeugen eines
            Beispielschienennetzes
23      {
24          Gerade_waagerecht g1 = new Gerade_waagerecht();
25          addObject(g1, 10, 6);
26          Gerade_waagerecht g2 = new Gerade_waagerecht();
27          addObject(g2, 11, 6);
28          Gerade_waagerecht g3 = new Gerade_waagerecht();
29          addObject(g3, 12, 6);
30          Gerade_waagerecht g4 = new Gerade_waagerecht();
31          addObject(g4, 13, 6);
32          Gerade_waagerecht g5 = new Gerade_waagerecht();
33          addObject(g5, 14, 6);
34          Gerade_waagerecht g6 = new Gerade_waagerecht();
35          addObject(g6, 15, 6);
36          Gerade_waagerecht g7 = new Gerade_waagerecht();
37          addObject(g7, 16, 6);
38          Gerade_waagerecht g8 = new Gerade_waagerecht();
39          addObject(g8, 17, 6);
40          Gerade_waagerecht g9 = new Gerade_waagerecht();
41          addObject(g9, 18, 6);
42          Weiche_Ost_Sued w3 = new Weiche_Ost_Sued();
43          addObject(w3, 19, 6);
44          Signal_Ost s7 = new Signal_Ost();
45          addObject(s7, 18, 6);
46          Gerade_waagerecht g34 = new Gerade_waagerecht();
47          addObject(g34, 20, 6);
48          Gerade_waagerecht g35 = new Gerade_waagerecht();
49          addObject(g35, 21, 6);
50          Gerade_waagerecht g36 = new Gerade_waagerecht();
51          addObject(g36, 22, 6);
52          Gerade_waagerecht g37 = new Gerade_waagerecht();
53          addObject(g37, 23, 6);
54          Signal_West s8 = new Signal_West();
55          addObject(s8, 22, 6);
56          Signal_Nord s9 = new Signal_Nord();
57          addObject(s9, 19, 7);
58          Prellbock_Ost p1 = new Prellbock_Ost();
59          addObject(p1, 24, 6);
60
61          Gerade_senkrecht g10 = new Gerade_senkrecht();
62          addObject(g10, 19, 7);
63          Gerade_senkrecht g11 = new Gerade_senkrecht();
64          addObject(g11, 19, 8);
65          Gerade_senkrecht g12 = new Gerade_senkrecht();
66          addObject(g12, 19, 9);
67          Gerade_senkrecht g13 = new Gerade_senkrecht();
68          addObject(g13, 19, 10);
69          Gerade_senkrecht g14 = new Gerade_senkrecht();
70          addObject(g14, 19, 11);
71          Gerade_senkrecht g15 = new Gerade_senkrecht();
72          addObject(g15, 19, 12);
73          Gerade_senkrecht g16 = new Gerade_senkrecht();
74          addObject(g16, 19, 13);
75          Kurve_Nord_West k2 = new Kurve_Nord_West();
76          addObject(k2, 19, 14);
```

```
77          Gerade_waagerecht g17 = new Gerade_waagerecht();
78          addObject(g17, 18, 14);
79          Weiche_Ost_Nord w1 = new Weiche_Ost_Nord();
80          addObject(w1, 17, 14);
81          Weiche w=(Weiche) w1;
82          w.stelleDieWeiche(w1);
83          Kurve_Sued_West k3 = new Kurve_Sued_West();
84          addObject(k3, 17, 13);
85          Gerade_waagerecht g18 = new Gerade_waagerecht();
86          addObject(g18, 16, 13);
87          Signal_Ost s1 = new Signal_Ost();
88          addObject(s1, 16, 13);
89          Gerade_waagerecht g19 = new Gerade_waagerecht();
90          addObject(g19, 15, 13);
91          Gerade_waagerecht g20 = new Gerade_waagerecht();
92          addObject(g20, 14, 13);
93          Signal_West s3 = new Signal_West();
94          addObject(s3, 14, 13);
95          Gerade_waagerecht g21 = new Gerade_waagerecht();
96          addObject(g21, 16, 14);
97          Signal_Ost s2 = new Signal_Ost();
98          addObject(s2, 16, 14);
99          Gerade_waagerecht g22 = new Gerade_waagerecht();
100         addObject(g22, 15, 14);
101         Gerade_waagerecht g23 = new Gerade_waagerecht();
102         addObject(g23, 14, 14);
103         Signal_West s4 = new Signal_West();
104         addObject(s4, 14, 14);
105         Kurve_Sued_Ost k4 = new Kurve_Sued_Ost();
106         addObject(k4, 13, 13);
107         Weiche_West_Nord w2 = new Weiche_West_Nord();
108         addObject(w2, 13, 14);
109         Gerade_waagerecht g24 = new Gerade_waagerecht();
110         addObject(g24, 12, 14);
111         Gerade_waagerecht g25 = new Gerade_waagerecht();
112         addObject(g25, 11, 14);
113         Gerade_waagerecht g26 = new Gerade_waagerecht();
114         addObject(g26, 10, 14);
115         Kurve_Nord_Ost k5 = new Kurve_Nord_Ost();
116         addObject(k5, 9, 14);
117         Gerade_senkrecht g27 = new Gerade_senkrecht();
118         addObject(g27, 9, 7);
119         Gerade_senkrecht g28 = new Gerade_senkrecht();
120         addObject(g28, 9, 8);
121         Gerade_senkrecht g29 = new Gerade_senkrecht();
122         addObject(g29, 9, 9);
123         Gerade_senkrecht g30 = new Gerade_senkrecht();
124         addObject(g30, 9, 10);
125         Gerade_senkrecht g31 = new Gerade_senkrecht();
126         addObject(g31, 9, 11);
127         Gerade_senkrecht g32 = new Gerade_senkrecht();
128         addObject(g32, 9, 12);
129         Gerade_senkrecht g33 = new Gerade_senkrecht();
130         addObject(g33, 9, 13);
131         Kurve_Sued_Ost k6 = new Kurve_Sued_Ost();
132         addObject(k6, 9, 6);
133         Haltestelle_waagerecht h1 = new Haltestelle_waagerecht();
134         addObject(h1,15,13);
```

```
135         h1.wegeFestlegen();
136         h1.toggleNamenAnzeigen();
137         Haltestelle_waagerecht h2 = new Haltestelle_waagerecht();
138         addObject(h2,15,14);
139         h2.wegeFestlegen();
140         Signal_Ost s5 = new Signal_Ost();
141         addObject(s5, 12, 14);
142         Signal_West s6 = new Signal_West();
143         addObject(s6, 18, 14);
144
145         Zug z1 = new Zug();
146         addObject(z1, 12, 14);
147     }
148 }
```

A.2.3 Block

```
1  public class Block extends Actor
2  {
3      public boolean besetzt;
4      char r1, r2;
5      public Block()
6      {
7          besetzt = false;
8      }
```

Besetzt-Methoden

```
1  private boolean istbesetzt(Actor istbesetztBlock)  //bekommt Block und
       gibt zurück ob Zug auf dem Block steht
2      {
3          Actor zug = getOneObjectAtOffset(istbesetztBlock.getX()-getX(),
               istbesetztBlock.getY()-getY(), Zug.class);
4          if(zug != null) {
5              besetzt=true;
6              return true;
7          }
8          else {
9              besetzt=false;
10             return false;
11         }
12     }
13     public boolean getbesetzt()  //gibt zurück ob Zug auf dem Block
           steht
14     {
15         Actor zug = getOneObjectAtOffset(0, 0, Zug.class);
16         if(zug != null) {
17             besetzt=true;
18             return besetzt;
19         }
20         else {
21             besetzt=false;
22             return besetzt;
23         }
24     }
```

```
25      public void setbesetzt(Actor istbesetztBlock)  //bekommt Block und
            passt besetzt an
26      {
27          Actor zug = getOneObjectAtOffset(istbesetztBlock.getX()-getX(),
                istbesetztBlock.getY()-getY(), Zug.class);
28          if(zug != null)
29          {
30              besetzt=true;
31          }
32          else
33          {
34              besetzt=false;
35          }
36      }
```

userFrage

```
1       public boolean userFrage(String fragetyp)
2       {
3           String input="";
4           switch(fragetyp)
5           {
6               case "Haltestelle":
7                   input = Greenfoot.ask("Haltestelle erreicht. Stehen
                        bleiben? (y/n)");
8                   break;
9               case "Signal":
10                  input = Greenfoot.ask("Zug hat Freigabe. Signal auf grü
                        n stellen? (y/n)");
11                  break;
12              case "Weiche falsch":
13                  input = Greenfoot.ask("Weiche steht falsch. Weiche
                        stellen? (y/n)");
14                  break;
15              case "Weiche kürzer":
16                  input = Greenfoot.ask("Weiche umstellen für kürzeren
                        Weg möglich. Weiche stellen? (y/n)");
17                  break;
18              case "Prellbock":
19                  input = Greenfoot.ask("Zug steht an einem Prellbock. In
                         anderer Richtung weiterfahren? (y/n)");
20                  break;
21              case "Ziel":
22                  input = Greenfoot.ask("Zug hat Ziel erreicht. Stehen
                        bleiben? (y/n)");
23                  break;
24          }
25          if (input.equals("y"))
26          {
27              return true;
28          }
29          else
30          {
31              if(input.equals("n"))
32              {
33                  return false;
34              }
35              else
```

```
36              {
37                  System.out.println("Ungültige Eingabe. Bitte mit y oder
                        n antworten.");
38                  return userFrage(fragetyp);
39              }
40          }
41      }
```

richtungskonverter

```
1       public char richtungskonverter(char ausgangsRichtung)  //bekommt
            Ausgangsrichtung aus altem Block, gibt Eingangsrichtung in
            neuen Block
2       {
3           switch(ausgangsRichtung){
4               case 'N':
5                   return 'S';
6               case 'O':
7                   return 'W';
8               case 'S':
9                   return 'N';
10              case 'W':
11                  return 'O';
12              default:
13                  return 'H';
14          }
15      }
```

Blockrichtung/Blockeingang

```
1       public char getBlockRichtung(Actor block, char eingangsR)  //nimmt
            Block und Eingangsrichtung, gibt Ausgangsrichtung
2       {
3           String schiene = block.getClass().getName();
4           if (schiene.contains("Weiche"))
5               {
6                   Weiche w=(Weiche) block;
7                   return w.getWeicheRichtung(block, eingangsR);
8               }
9               else
10              {
11                 return blockgetRichtung(block, eingangsR);
12              }
13      }
14      private char blockgetRichtung(Actor a, char s)  //bekommt Actor und
            Eingangsrichtung, gibt Ausgangsrichtung
15      {
16          Block b=(Block) a;
17          if (s == b.r1)
18          {
19              return b.r2;
20          }
21          else
22          {
23              return b.r1;
24          }
25      }
```

```
26      public boolean checkBlockEingang(Actor blockEingang, char
            eingangsRichtung, boolean beachteWeichenstellung)  //bekommt
            Block, Eingangsrichtung und ob Weichenstellung beachtet werden
            soll, gibt zurück ob Block da Eingang hat
27      {
28          String schiene = blockEingang.getClass().getName();
29          if (schiene.contains("Weiche"))
30          {
31              Weiche w=(Weiche) blockEingang;
32              if (beachteWeichenstellung == true)
33              {
34                  return w.checkWeichenEingang(blockEingang,
                        eingangsRichtung);
35              }
36              else
37              {
38                  return w.checkTheoretischenWeichenEingang(blockEingang,
                        eingangsRichtung);
39              }
40          }
41          else
42          {
43              return blockcheckEingang(blockEingang, eingangsRichtung);
44          }
45      }
46      private boolean blockcheckEingang(Actor a,char eingangsRichtung) //
            bekommt Actor und Eingangsrichtung, gibt zurück ob valider
            Eingang
47      {
48          Block b=(Block) a;
49          if (eingangsRichtung == b.r1 || eingangsRichtung == b.r2)
50          {
51              return true;
52          }
53          else
54          {
55              return false;
56          }
57      }
```

getnextBlock

```
1       public Actor getnextBlock(int aktuellesX, int aktuellesY,char
            richtung)  //bekommt x,y-Koordinaten von Actor und
            Ausgangsrichtung, gibt den Nachbaractor in der Ausgangsrichtung
            zurück
2       {
3           if (richtung=='N')
4           {
5               Actor nextBlock = getOneObjectAtOffset(aktuellesX-getX(),
                    aktuellesY-1-getY(), Block.class);  //gibt Block ein
                    Feld über (aktuellesX|aktuellesY)
6               if (nextBlock != null)
7               {
8                   return nextBlock;
9               }
10              else
11              {
```

```
12                  if((aktuellesX-getX()==0) && (aktuellesY-1-getY()==0))
                        //falls gesuchte Block er aktuelle Block ist.
                       getOneObject gibt dann null zurück
13                  {
14                      return this;
15                  }
16                  else
17                  {
18                      return null;
19                  }
20              }
21          }
22          else
23          {
24          if (richtung=='O')
25          {
26              Actor nextBlock = getOneObjectAtOffset(aktuellesX+1-getX(),
                    aktuellesY-getY(), Block.class);  //gibt Block ein Feld
                    rechts von (aktuellesX|aktuellesY)
27              if (nextBlock != null)
28              {
29              return nextBlock;
30              }
31              else
32              {
33                  if((aktuellesX+1-getX()==0) && (aktuellesY-getY()==0))
                        //falls gesuchte Block er aktuelle Block ist.
                        getOneObject gibt dann null zurück
34                  {
35                      return this;
36                  }
37                  else
38                  {
39                      return null;
40                  }
41              }
42          }
43          else
44          {
45               if (richtung=='S')
46          {
47              Actor nextBlock = getOneObjectAtOffset(aktuellesX-getX(),
                    aktuellesY+1-getY(), Block.class);  //gibt Block ein
                    Feld unter (aktuellesX|aktuellesY)
48              if (nextBlock != null)
49              {
50              return nextBlock;
51              }
52              else
53              {
54              if((aktuellesX-getX()==0) && (aktuellesY+1-getY()==0)) //
                    falls gesuchte Block er aktuelle Block ist. getOneObject
                     gibt dann null zurück
55                   {
56                      return this;
57                   }
58                   else
59                   {
```

```
60                      return null;
61                  }
62              }
63          }
64          else
65          {
66          if (richtung=='W')
67          {
68              Actor nextBlock = getOneObjectAtOffset(aktuellesX-1-getX(),
                    aktuellesY-getY(), Block.class);  //gibt Block ein Feld
                    links von (aktuellesX|aktuellesY)
69              if (nextBlock != null)
70              {
71              return nextBlock;
72              }
73              else
74              {
75                  if((aktuellesX-1-getX()==0) && (aktuellesY-getY()==0))
                        //falls gesuchte Block er aktuelle Block ist.
                        getOneObject gibt dann null zurück
76                  {
77                      return this;
78                  }
79                  else
80                  {
81                      return null;
82                  }
83              }
84          }
85          else
86          {
87              System.out.println("Fehler (unbekannte Richtung in
                    getnextBlock)");
88              return null;
89          }
90          }
91          }
92          }
93      }
```

istWeiche

```
1       public boolean istWeiche(Actor mglweiche)   //checkt ob Actor eine
            Weiche ist
2       {
3           if (mglweiche != null)
4           {
5               String weiche = mglweiche.getClass().getName();
6               return (weiche.contains("Weiche"));
7           }
8           else
9           {
10              return false;
11          }
12      }
```

istRotesSignal

```
1   public boolean istRotesSignal(Actor mglsignal, char
        eingangsRichtung)  //bekommt möglichen Signalblock und
        Eingangsrichtung. Checkt ob relevantes rotes Signal
2   {
3       List<Signal> listSignal;
4       listSignal = getObjectsAtOffset(mglsignal.getX()-getX(),
            mglsignal.getY()-getY(), Signal.class);  //bekommt Liste
            aller Signale an diesem Block
5       boolean rotesSignal=false;
6       if (listSignal.isEmpty() == false)
7       {
8           for (int i = 0; i<listSignal.size(); i++)   //durchlaufe
                Liste
9           {
10              if (rotesSignal == false)    //falls noch kein
                    passendes Signal gefunden
11              {
12                  Actor msignal = listSignal.get(i);
13                  Signal checkMglSignal = (Signal) msignal;
14                  rotesSignal = checkMglSignal.checkRotesSignal(
                        msignal, blockgetRichtung(mglsignal,
                        eingangsRichtung));
15              }
16          }
17      }
18      return rotesSignal;
19  }
```

weicheZuBekanntHinzufuegen

```
1   private String[] weicheZuBekanntHinzufuegen(String[]
        schonBekannteWeichen,int zahlSchritte,char
        weichenEingangsRichtung,int xPosN,int yPosN) // speichert Weg
        in der Form (zB. "12,W.45;14" für 12 Schritte und Westrichtung
        an der Stelle 45,14)
2   {
3       boolean schonbekannt = false;
4       if (schonBekannteWeichen !=null)
5       {
6           for(int lauf=0; lauf<schonBekannteWeichen.length; lauf++)
                //durchläuft den Array
7           {
8               if (schonBekannteWeichen[lauf].contains(
                    weichenEingangsRichtung+xPosN+";"+yPosN))   //falls
                    Richtung inklusive Ort bekannt
9               {
10                  int bekannteZahlSchritte = Integer.parseInt(
                        schonBekannteWeichen[lauf].substring(0,
                        schonBekannteWeichen[lauf].indexOf(",")));//
                        Schrittzahl des bekannten Wegs
11                  if (Integer.valueOf(bekannteZahlSchritte).compareTo
                        (Integer.valueOf(zahlSchritte))>0) //falls neue
                         Weg kürzer
12                  {
13                      schonbekannt=true;
```

```
14                          schonBekannteWeichen[lauf]= (zahlSchritte+","+
                                weichenEingangsRichtung+"."+xPosN+";"+yPosN
                                );
15                      }
16                  }
17              }
18              if (schonbekannt == false)  //weg unbekannt
19              {
20                  String temporaererArray[]= new String[
                        schonBekannteWeichen.length+1];
21                  System.arraycopy(schonBekannteWeichen, 0,
                        temporaererArray, 0, schonBekannteWeichen.length);
22                  schonBekannteWeichen=temporaererArray;
23                  schonBekannteWeichen[schonBekannteWeichen.length-1] = (
                        zahlSchritte+","+weichenEingangsRichtung+"."+xPosN+
                        ";"+yPosN);
24              }
25          }
26          else{
27              schonBekannteWeichen = new String [1];
28              schonBekannteWeichen[0]=(zahlSchritte+","+
                    weichenEingangsRichtung+"."+xPosN+";"+yPosN);
29          }
30          return schonBekannteWeichen;
31      }
```

weicheSchonBekannt

```
1       private boolean weicheSchonBekannt(String[] schonBekannteWeichen,
            int zahlSchritte,char weichenEingangsRichtung,int xPosNaechste,
            int yPosNaechste)
2       //bekommt String mit schon abgefahrenen Weichen samt Schrittzahl
            und Eingangsrichtung in der Form (zB. "12,W.45;14" für 12
            Schritte und Westrichtung an der Stelle 45,14), gibt zurück ob
            bekannt
3       {
4           boolean wegSchonBekannt = false;
5           if (schonBekannteWeichen != null)
6           {
7               for(int lauf=0; lauf<schonBekannteWeichen.length; lauf++)
                    //durchläuft den Array
8               {
9                   if (schonBekannteWeichen[lauf].contains(
                        weichenEingangsRichtung+"."+xPosNaechste+";"+
                        yPosNaechste))  //falls Richtung inklusive Ort
                        bekannt
10                  {
11                      int bekannteZahlSchritte = Integer.parseInt(
                            schonBekannteWeichen[lauf].substring(0,
                            schonBekannteWeichen[lauf].indexOf(",")));//
                            Schrittzahl des bekannten Wegs
12                      if ((Integer.valueOf(bekannteZahlSchritte).
                            compareTo(Integer.valueOf(zahlSchritte))<0)||(
                            Integer.valueOf(bekannteZahlSchritte).compareTo
                            (Integer.valueOf(zahlSchritte))==0))
13                      //falls bekannte Weg kürzer oder gleich lang
14                      {
15                          wegSchonBekannt = true;
```

```
16                      }
17                  }
18              }
19          }
20          return wegSchonBekannt;
21      }
```

findeNaechsteWeiche

```
1  public Object[] findeNaechsteWeiche(Actor g, char eingangsRichtung, int
       anzahlSchritte)
2  //Suche nächste Weiche Bekommt: Actor und Eingangsrichtung. Gibt: Array
       {char Eingangsrichtung,int x-Koordinate,int y-Koordinate, Anzahl
      Schritte}
3  {
4      Object[] weicheEinXY = new Object[4];
5      weicheEinXY[0] = null;
6      weicheEinXY[1] = null;
7      weicheEinXY[2] = null;
8      weicheEinXY[3] = null;
9      if (checkBlockEingang(g,eingangsRichtung, false) && (istWeiche(g)
           != true))
10     {
11         char ausgang = getBlockRichtung(g, eingangsRichtung);
12         Actor n = getnextBlock(g.getX(), g.getY(), ausgang);  //gibt
              den nächsten Block in Richtung ausgang. Bei Sackgasse n =
              block der Starthaltestelle.
13         Block nB = (Block) n;
14         if((n!=null) && ((getX() != n.getX()) || (getY() != n.getY())))
                //wenn keine Sackgasse
15         {
16             return findeNaechsteWeiche(n, richtungskonverter(ausgang),
                  anzahlSchritte+1);
17         }
18         else
19         {
20             return weicheEinXY;
21         }
22     }
23     else
24     {
25        if (istWeiche(g) &&  (checkBlockEingang(g,eingangsRichtung,
              false)))
26        {
27            weicheEinXY[0] = String.valueOf(eingangsRichtung);
28            weicheEinXY[1] = g.getX();
29            weicheEinXY[2] = g.getY();
30            weicheEinXY[3] = anzahlSchritte;
31            return weicheEinXY;
32         }
33         else  //Sackgasse
34         {
35             System.out.println("Fehlerhafte Strecke. Konnte keine
                  Weiche finden!");
36             return weicheEinXY;
37         }
38     }
39 }
```

hinzufuegenWegeWeichen

```
1  public void hinzufuegenWegeWeichen(Actor startBlock, int xPosStart, int
       yPosStart, char ausgangsRichtung,String haltestellenname, int
       schrittzahl, String[] bekannteWeichen)
2  {
3      Object[] naechsteWeicheTypPos = new Object[4];
4      switch(ausgangsRichtung){
5          case 'N':
6              Actor nord = getOneObjectAtOffset(xPosStart-getX()+0,
                   yPosStart-getY()-1, Block.class);
7              if (nord !=null){
8              naechsteWeicheTypPos=findeNaechsteWeiche(nord,
                   richtungskonverter(ausgangsRichtung), schrittzahl+1);
                   //finde nächste Weiche
9              }
10         break;
11         case 'O':
12             Actor ost = getOneObjectAtOffset(xPosStart-getX()+1,
                   yPosStart-getY()+0, Block.class);
13             if (ost != null){
14             naechsteWeicheTypPos=findeNaechsteWeiche(ost,
                   richtungskonverter(ausgangsRichtung), schrittzahl+1);
                   //finde nächste Weiche
15             }
16         break;
17         case 'S':
18             Actor sued = getOneObjectAtOffset(xPosStart-getX()+0,
                   yPosStart-getY()+1, Block.class);
19             if (sued != null){
20                 naechsteWeicheTypPos=findeNaechsteWeiche(sued,
                       richtungskonverter(ausgangsRichtung), schrittzahl
                       +1); //finde nächste Weiche
21             }
22         break;
23         case 'W':
24             Actor west = getOneObjectAtOffset(xPosStart-getX()-1,
                   yPosStart-getY()+0, Block.class);
25             if (west != null){
26                 naechsteWeicheTypPos=findeNaechsteWeiche(west,
                       richtungskonverter(ausgangsRichtung), schrittzahl
                       +1); //finde nächste Weiche
27             }
28         break;
29         default:
30             System.out.println("Unbekannte Richtung beim hinzufügen der
                   Wege zu den Weichen.");
31     }
32     if (naechsteWeicheTypPos[1] != null)
33     {
34         String weiche0 = (String) naechsteWeicheTypPos[0];
35         char weichenEingangsRichtung = weiche0.charAt(0);
36         int xPosNaechste = (int) naechsteWeicheTypPos[1];
37         int yPosNaechste = (int) naechsteWeicheTypPos[2];
38         int zahlSchritte= (int) naechsteWeicheTypPos[3];
39         Actor naechsteW = getOneObjectAtOffset(xPosNaechste-getX(),
```

```
            yPosNaechste-getY(), Weiche.class); //Actor naechsteW ist
            die nächste Weiche in Nordrichtung
40          Weiche naechsteWeiche;
41          naechsteWeiche=(Weiche) naechsteW;
42          String ausgaengeWeiche = naechsteWeiche.mglAusgaengeWeiche(
                naechsteW, weichenEingangsRichtung);
43          if((bekannteWeichen ==null)||(weicheSchonBekannt(
                bekannteWeichen, zahlSchritte, weichenEingangsRichtung,
                xPosNaechste, yPosNaechste) !=true))//check ob Weiche in
                diesem Durchlauf schon bekannt (bzw. kürzester)
44          {
45              bekannteWeichen = weicheZuBekanntHinzufuegen(
                    bekannteWeichen, zahlSchritte, weichenEingangsRichtung,
                     xPosNaechste, yPosNaechste);
46              if (naechsteWeiche.checkKuerzesterWeg(
                    weichenEingangsRichtung, haltestellenname, zahlSchritte
                    ))//überprüfe ob Weg schon bekannt, bzw. es falls
                    bekannt kürzerer Weg ist
47              {
48                  naechsteWeiche.neuerWeg(weichenEingangsRichtung,
                        haltestellenname, zahlSchritte);
49              }
50              for(int nummerAusgang=0; nummerAusgang<ausgaengeWeiche.
                    length(); nummerAusgang++)
51              {
52                  hinzufuegenWegeWeichen(naechsteW, xPosNaechste,
                        yPosNaechste, ausgaengeWeiche.charAt(nummerAusgang)
                        , haltestellenname, zahlSchritte, bekannteWeichen);
53              }
54          }
55      }
56  }
```

checkFreigabe

```
1   public String checkFreigabe(Actor start,char eingangsRichtung,String
        zielhaltestelle,String[] merker)
2   /*
3    * gibt zurück: "0" - wenn Weg bis zum nächsten roten Signal, der nä
        chsten Haltestelle/Prellbock frei ist und alle Weichen bis dahin
        richtig gestellt sind
4    *              "1,xPos der Weiche;yPos der Weiche" - wenn Weiche
        gestellt werden muss
5    *              "2,xPos;yPos des Zugs" - wenn keine Freigabe (besetzte
        Schiene)
6    *              "3,xPos;yPos des letzten Blocks" - Fehlerhafte Strecke
7    */
8   {
9   if (checkBlockEingang(start,eingangsRichtung, false))
10  {
11  if (istbesetzt(start) != true)   //check ob zu prüfender Block besetzt
        ist
12  {
13      if  (istRotesSignal(start, eingangsRichtung) != true)  //falls kein
            Rotes Signal
14      {
15      Actor aktuellerPrellbock =getOneObjectAtOffset(start.getX()-getX(),
            start.getY()-getY(), Prellbock.class);
```

```
16      Prellbock aktPrellbock = (Prellbock) aktuellerPrellbock;
17      if (aktuellerPrellbock == null)
18      {
19          Actor haltestelle = getOneObjectAtOffset(start.getX()-getX(),
                start.getY()-getY(), Haltestelle.class);
20          Haltestelle hstelle = (Haltestelle) haltestelle;
21          if ((haltestelle==null))      //falls keine Haltestelle
                erreicht
22          {
23              char ausgang = getBlockRichtung(start, eingangsRichtung);
                    //get Blockausgang
24              if (ausgang == 'H')  //wenn eingangsRichtung bei Weiche zur
                     Stellung nicht passt, gibt getBlockRichtung 'H' zurück
25              {
26                  if (merker!= null)  //überprüfe ob Weiche bekannt
27                  {
28                      for (int j=0; j<merker.length; j++)
29                      {
30                          if(merker[j].substring(2, merker[j].length()).
                                equals(start.getX()+";"+start.getY())) //
                                falls man schonmal an dieser Weiche war
31                          {
32                              return "3, "+start.getX()+";"+start.getY();
                                    //Weiche kann nicht von anderer
                                   Richtung als bisher angefahren werden
                                   ohne dass die Weiche während der Fahrt
                                   gestellt werden müsste
33                          }
34                      }
35                  }
36                  if(userFrage("Weiche falsch"))
37                  {
38                      Greenfoot.delay(3);
39                      Weiche kommeWeiche = (Weiche) start;
40                      kommeWeiche.stelleDieWeiche(start);
41                      return checkFreigabe(start, eingangsRichtung,
                            zielhaltestelle, merker);//neu aufrufen mit
                            jetzt gestellter Weiche
42                  }
43                  else
44                  {
45                      return "1,"+start.getX()+";"+start.getY();
46                  }
47              }
48              else  //keine falsch gestellte Weiche
49              {
50                  if (istWeiche(start)!=true)  //keine Weiche
51                  {
52                      Actor n = getnextBlock(start.getX(), start.getY(),
                            ausgang);  //gibt den nächsten Block in
                            Richtung ausgang. Bei Sackgasse n = block der
                            Starthaltestelle.
53                      if(n!=null)  //wenn keine Sackgasse
54                      {
55                          return checkFreigabe(n, richtungskonverter(
                                ausgang),zielhaltestelle, merker);
56                      }
57                      else //Sackgasse
```

```
58                      {
59                          return "3,"+start.getX()+";"+start.getY();
60                      }
61                  }
62                  else  //ist Weiche
63                  {
64                  if (zielhaltestelle != null)  //falls zielhaltestelle
                        angegeben
65                  {
66                      Weiche kommeWeiche = (Weiche) start;
67                      String mglWegeDerWeiche = kommeWeiche.
                            mglAusgaengeWeiche(start, eingangsRichtung);
                            //return String (max Länge 2) mit den möglichen
                             Ausgängen
68                      char weichenstellung = kommeWeiche.
                            getWeicheRichtung(start, eingangsRichtung);
                                  //bekomme aktuellen Ausgang abhängig der
                            Stellung der Weiche
69                      char richtungWeg1 = mglWegeDerWeiche.charAt(0);
70                      String richtung1Weg = kommeWeiche.getWeg(
                            richtungWeg1, zielhaltestelle);    //bekomme
                            String der Form "richtungWeg1 zielhaltestelle;
                            AnzahlderSchritte", falls Weg zum Ziel möglich,
                             sonst null
71                      if (mglWegeDerWeiche.length()==2)            //falls
                             es 2 mgl Ausgänge gibt
72                      {
73                          char richtungWeg2 = mglWegeDerWeiche.charAt(1);
74                          String richtung2Weg = kommeWeiche.getWeg(
                                richtungWeg2, zielhaltestelle);  //bekomme
                                String der Form "richtungWeg2
                                zielhaltestelle; AnzahlderSchritte", falls
                                Weg zum Ziel möglich, sonst null
75                          if ((richtung1Weg!=null) && (richtung2Weg !=
                                null))  //wenn beide Wege zur Haltestelle f
                                ühren
76                          {
77                              int schrittzahlWeg1 = Integer.parseInt(
                                    richtung1Weg.substring(richtung1Weg.
                                    indexOf(';')+2, richtung1Weg.length()))
                                    ; //Schrittlänge zum Ziel mit Weg 1
78                              int schrittzahlWeg2= Integer.parseInt(
                                    richtung2Weg.substring(richtung2Weg.
                                    indexOf(';')+2, richtung2Weg.length()))
                                    ; //Schrittlänge zum Ziel mit Weg 2
79                              if((Integer.valueOf(schrittzahlWeg1).
                                    compareTo(Integer.valueOf(
                                    schrittzahlWeg2)) < 0) || ((
                                    schrittzahlWeg1==schrittzahlWeg2)&&(
                                    richtungWeg1 == weichenstellung))) //if
                                     Weg1 kürzer als Weg 2, bei gleicher Lä
                                    nge wähle Weg1, wenn Weiche dafür steht
80                              {
81                                  if(richtungWeg1 == weichenstellung)  //
                                        Weiche steht für kürzesten Weg
82                                  {
83                                      String temporaererArray[]= new
                                            String[merker.length+1];
```

```
                                                    //Weiche
                                        wird in merker mit
                                        Eingangsrichtung gespeichert
84                                  System.arraycopy(merker, 0,
                                        temporaererArray, 0, merker.
                                        length);
85                                  merker=temporaererArray;
86                                  merker[merker.length-1] = (
                                        eingangsRichtung+","+start.getX
                                        ()+";"+start.getY());
87
88                                  Actor n = getnextBlock(start.getX()
                                        , start.getY(), ausgang);
89                                  return checkFreigabe(n,
                                        richtungskonverter(ausgang),
                                        zielhaltestelle, merker);//
                                        Weiche steht richtig, also
                                        weitergucken
90                              }
91                              else  //Weiche steht auf längerem Weg
92                              {
93                                  if (merker!= null)
94                                  {
95                                      for (int j=0; j<merker.length;
                                            j++)
96                                      {
97                                          if(merker[j].substring(2,
                                                merker[j].length()).
                                                equals(start.getX()+";"
                                                +start.getY())) //falls
                                                 man schonmal an dieser
                                                 Weiche war
98                                          {
99                                              //bekannt = true;
100                                             return "3, "+start.getX
                                                    ()+";"+start.getY()
                                                    ;  //Weiche kann
                                                    nicht von anderer
                                                    Richtung als bisher
                                                     angefahren werden
                                                    ohne dass die
                                                    Weiche während der
                                                    Fahrt gestellt
                                                    werden müsste
101                                         }
102                                     }
103                                     //unbekannte Weiche
104                                     String temporaererArray[]= new
                                            String[merker.length+1];
                                            //Weiche und Richtung in
                                            Merker speichern
105                                     System.arraycopy(merker, 0,
                                            temporaererArray, 0, merker
                                            .length);
106                                     merker=temporaererArray;
107                                     merker[merker.length-1] = (
                                            eingangsRichtung+","+start.
                                            getX()+";"+start.getY());
```

```
108                                     if(userFrage("Weiche kürzer"))
109                                     {
110                                         Greenfoot.delay(3);
111                                         kommeWeiche.stelleDieWeiche
                                                (start);
112                                         ausgang = getBlockRichtung(
                                                start, eingangsRichtung
                                                );
113                                         Actor n = getnextBlock(
                                                start.getX(), start.
                                                getY(), ausgang);
114                                         return checkFreigabe(n,
                                                richtungskonverter(
                                                ausgang),
                                                zielhaltestelle, merker
                                                );
115                                     }
116                                     else  //weiche bleibt für
                                            langen Weg gestellt,
                                            speichern in merker
117                                     {
118                                         Actor n = getnextBlock(
                                                start.getX(), start.
                                                getY(), ausgang);
119                                         return checkFreigabe(n,
                                                richtungskonverter(
                                                ausgang),
                                                zielhaltestelle, merker
                                                ); //zum ersten Mal bei
                                                 der Weiche also
                                                weitergucken
120                                     }
121                                 }
122                                 else  //merker == null
123                                 {
124                                     String temporaererArray[]= new
                                            String[merker.length+1];
125                                     merker=temporaererArray;
126                                     merker[merker.length-1] = (
                                            eingangsRichtung+","+start.
                                            getX()+";"+start.getY());
127                                     Actor n = getnextBlock(start.
                                            getX(), start.getY(),
                                            ausgang);
128                                     return checkFreigabe(n,
                                            richtungskonverter(ausgang),
                                            zielhaltestelle, merker); //
                                            zum ersten Mal bei der
                                            Weiche also weitergucken
129                                 }
130                             }
131                         }
132                         else    //Weg2 kürzer
133                         {
134                             if(richtungWeg2 == weichenstellung) //
                                    Weiche steht für kürzesten Weg
135                             {
136                                 String temporaererArray[]= new
```

```
                                String[merker.length+1];
                                                    //Weiche
                                wird in merker mit
                                Eingangsrichtung gespeichert
137                         System.arraycopy(merker, 0,
                                temporaererArray, 0, merker.
                                length);
138                         merker=temporaererArray;
139                         merker[merker.length-1] = (
                                eingangsRichtung+","+start.getX
                                ()+";"+start.getY());
140
141                         Actor n = getnextBlock(start.getX()
                                , start.getY(), ausgang);
142                         return checkFreigabe(n,
                                richtungskonverter(ausgang),
                                zielhaltestelle, merker);//
                                Weiche steht richtig, also
                                weitergucken
143                     }
144                     else    //Weiche steht auf längerem Weg
145                     {
146                         if (merker!= null)
147                         {
148                             for (int j=0; j<merker.length;
                                    j++)
149                             {
150                                 if(merker[j].substring(2,
                                        merker[j].length()).
                                        equals(start.getX()+";"
                                        +start.getY())) //falls
                                         man schonmal an dieser
                                         Weiche war
151                                 {
152                                     return "3, "+start.getX
                                            ()+";"+start.getY()
                                            ;  //Weiche kann
                                            nicht von anderer
                                            Richtung als bisher
                                             angefahren werden
                                            ohne dass die
                                            Weiche während der
                                            Fahrt gestellt
                                            werden müsste
153                                 }
154                             }
155                             //Weiche unbekannt
156                             String temporaererArray[]= new
                                    String[merker.length+1];
                                    //Weiche und Richtung in
                                    Merker speichern
157                             System.arraycopy(merker, 0,
                                    temporaererArray, 0, merker
                                    .length);
158                             merker=temporaererArray;
159                             merker[merker.length-1] = (
                                    eingangsRichtung+","+start.
                                    getX()+";"+start.getY());
```

```
160                                     if(userFrage("Weiche kürzer"))
161                                     {
162                                         Greenfoot.delay(3);
163                                         kommeWeiche.stelleDieWeiche
                                                (start);
164                                         ausgang = getBlockRichtung(
                                                start, eingangsRichtung
                                                );
165                                         Actor n = getnextBlock(
                                                start.getX(), start.
                                                getY(), ausgang);
166                                         return checkFreigabe(n,
                                                richtungskonverter(
                                                ausgang),
                                                zielhaltestelle, merker
                                                );
167                                     }
168                                     else
169                                     {
170                                         Actor n = getnextBlock(
                                                start.getX(), start.
                                                getY(), ausgang);
171                                         return checkFreigabe(n,
                                                richtungskonverter(
                                                ausgang),
                                                zielhaltestelle, merker
                                                ); //zum ersten Mal bei
                                                 der Weiche also
                                                weitergucken
172                                     }
173                                 }
174                                 else  //merker == null
175                                 {
176                                     String temporaererArray[]= new
                                            String[merker.length+1];  //
                                            Weiche und Richtung in
                                            Merker speichern
177                                     merker=temporaererArray;
178                                     merker[merker.length-1] = (
                                            eingangsRichtung+","+start.
                                            getX()+";"+start.getY());
179                                     Actor n = getnextBlock(start.
                                            getX(), start.getY(),
                                            ausgang);
180                                     return checkFreigabe(n,
                                            richtungskonverter(ausgang),
                                            zielhaltestelle, merker); //
                                            zum ersten Mal bei der
                                            Weiche also weitergucken
181                                 }
182                             }
183                         }
184                     }
185                     else  //es führen nicht beide Wege zum Ziel
186                     {
187                         if ((richtung1Weg != null)&&(richtung2Weg==
                                null))   //wenn nur Weg1 zur
                                Haltestelle führt
```

```
188                         {
189                             if(richtungWeg1 == weichenstellung)  //
                                    Weiche steht richtig
190                             {
191                                 String temporaererArray[]= new
                                        String[merker.length+1];
                                                        //Weiche
                                        wird in merker mit
                                        Eingangsrichtung gespeichert
192                                 System.arraycopy(merker, 0,
                                        temporaererArray, 0, merker.
                                        length);
193                                 merker=temporaererArray;
194                                 merker[merker.length-1] = (
                                        eingangsRichtung+","+start.getX
                                        ()+";"+start.getY());
195
196                                 Actor n = getnextBlock(start.getX()
                                        , start.getY(), ausgang);
197                                 return checkFreigabe(n,
                                        richtungskonverter(ausgang),
                                        zielhaltestelle, merker);//
                                        Weiche steht richtig, also
                                        weitergucken
198                             }
199                             else  //Weiche steht falsch
200                             {
201                                 if(userFrage("Weiche falsch"))
202                                 {
203                                     Greenfoot.delay(3);
204                                     kommeWeiche.stelleDieWeiche(
                                            start);
205                                     return checkFreigabe(start,
                                            eingangsRichtung,
                                            zielhaltestelle, merker);//
                                            neu aufrufen mit jetzt
                                            gestellter Weiche
206                                 }
207                                 else
208                                 {
209                                     return "1,"+start.getX()+";"+
                                            start.getY();
210                                 }
211                             }
212                         }
213                         else  //Weg1 führt nicht nzur
                                Zielhaltestelle
214                         {
215                             if ((richtung1Weg == null)&&(
                                    richtung2Weg!=null))    //wenn nur
                                    Weg2 zur Haltestelle führt
216                             {
217                                 if(richtungWeg2 == weichenstellung)
                                        //Weiche steht richtig
218                                 {
219                                     String temporaererArray[]= new
                                            String[merker.length+1];
                                                            //
```

```
                                    Weiche wird in merker mit
                                    Eingangsrichtung
                                    gespeichert
220                                 System.arraycopy(merker, 0,
                                    temporaererArray, 0, merker
                                    .length);
221                                 merker=temporaererArray;
222                                 merker[merker.length-1] = (
                                    eingangsRichtung+","+start.
                                    getX()+";"+start.getY());
223
224                                 Actor n = getnextBlock(start.
                                    getX(), start.getY(),
                                    ausgang);
225                                 return checkFreigabe(n,
                                    richtungskonverter(ausgang)
                                    ,zielhaltestelle, merker);
                                    //Weiche steht richtig,
                                    also weitergucken
226                             }
227                             else  // Weiche steht falsch
228                             {
229                                 if(userFrage("Weiche falsch"))
230                                 {
231                                     Greenfoot.delay(3);
232                                     kommeWeiche.stelleDieWeiche
                                        (start);
233                                     return checkFreigabe(start,
                                         eingangsRichtung,
                                        zielhaltestelle, merker
                                        );//neu aufrufen mit
                                        jetzt gestellter Weiche
234                                 }
235                                 else
236                                 {
237                                     return "1,"+start.getX()+";
                                        "+start.getY();
238                                 }
239                             }
240                         }
241                         else
242                         {
243                             if ((richtung1Weg==null) && (
                                richtung2Weg == null)) //wenn
                                weder Weg1 noch Weg2 zur
                                Haltestelle führen
244                             {
245                                 System.out.println("Kein Weg
                                    zur Haltestelle: " +
                                    zielhaltestelle+ "
                                    vorhanden");
246                                 return "3,"+start.getX()+";"+
                                    start.getY();
247                             }
248                         }
249                     }
250                 }
251             }
```

```
252                     else  // nur eine Weichenrichtung
253                     {
254                         if(richtung1Weg != null)  //Weg führt zur
                                Haltestelle
255                         {
256                             String temporaererArray[]= new String[
                                    merker.length+1];
                                    //Weiche wird in merker mit
                                    Eingangsrichtung gespeichert
257                             System.arraycopy(merker, 0,
                                    temporaererArray, 0, merker.length);
258                             merker=temporaererArray;
259                             merker[merker.length-1] = (eingangsRichtung
                                    +","+start.getX()+";"+start.getY());
260
261                             Actor n = getnextBlock(start.getX(), start.
                                    getY(), ausgang);
262                             return checkFreigabe(n, richtungskonverter(
                                    ausgang),zielhaltestelle, merker);//
                                    Weiche steht richtig, also weitergucken
263                         }
264                         else  //kein Weg führt zur Zielhaltestelle
265                         {
266                             System.out.println("Kein Weg zur
                                    Haltestelle: " + zielhaltestelle+ "
                                    vorhanden");
267                             return "3,"+start.getX()+";"+start.getY();
268                         }
269                     }
270                 }
271                 else // Zielhaltestelle = null
272                 {
273                     Actor n = getnextBlock(start.getX(), start.getY(),
                            ausgang);  //gibt den nächsten Block in
                            Richtung ausgang. Bei Sackgasse n = block der
                            Starthaltestelle.
274                     if((n!=null) && ((getX() != n.getX()) || (getY() !=
                            n.getY())))  //wenn keine Sackgasse
275                     {
276                         String temporaererArray[]= new String[merker.
                                length+1];                    //Weiche
                                wird in merker mit Eingangsrichtung
                                gespeichert
277                         System.arraycopy(merker, 0, temporaererArray,
                                0, merker.length);
278                         merker=temporaererArray;
279                         merker[merker.length-1] = (eingangsRichtung+","
                                +start.getX()+";"+start.getY());
280
281                         return checkFreigabe(n, richtungskonverter(
                                ausgang),zielhaltestelle, merker);
282                     }
283                     else //Sackgasse
284                     {
285                         return "3,"+start.getX()+";"+start.getY();
286                     }
287                 }
288             }
```

```
289             }
290         }
291         else //Haltestelle erreicht
292             {
293                 return "0";
294             }
295         }
296         else //Prellbock
297         {
298             return "0";
299         }
300         }
301         else    //rotes Signal
302         {
303             return "0";
304         }
305 }
306 else  //Block besetzt
307 {
308     return "2,"+start.getX()+";"+start.getY();
309 }
310 }
311 else  //Blockeingang falsch
312 {
313     return "3,"+start.getX()+";"+start.getY();
314 }
315 return "4";  //4 wird als Fehlercode zurückgegeben
316 }
```

Gerade_senkrecht

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Gerade_senkrecht
5   *
6   * @author Niklas Nogossek
7   */
8  public class Gerade_senkrecht extends Block
9  {
10     public Gerade_senkrecht()
11     {
12         r1='N';
13         r2='S';
14     }
15 }
```

Gerade_waagerecht

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Gerade_waagerecht
5   *
6   * @author Niklas Nogossek
7   */
8  public class Gerade_waagerecht extends Block
```

```
9  {
10     public Gerade_waagerecht()
11     {
12         r1='W';
13         r2='O';
14     }
15 }
```

Kurve_Nord_Ost

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Kurve_Nord_Ost
5   *
6   * @author Niklas Nogossek
7   */
8  public class Kurve_Nord_Ost extends Block
9  {
10     public Kurve_Nord_Ost()
11     {
12         r1='N';
13         r2='O';
14     }
15 }
```

Kurve_Nord_West

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Kurve_Nord_West
5   *
6   * @author Niklas Nogossek
7   */
8  public class Kurve_Nord_West extends Block
9  {
10     public Kurve_Nord_West()
11     {
12         r1='N';
13         r2='W';
14     }
15 }
```

Kurve_Sued_Ost

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Kurve_Sued_Ost
5   *
6   * @author Niklas Nogossek
7   */
8  public class Kurve_Sued_Ost extends Block
9  {
10     public Kurve_Sued_Ost()
11     {
```

```
12          r1='S';
13          r2='O';
14      }
15  }
```

Kurve_Sued_West

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Kurve_Sued_West
5   *
6   * @author Niklas Nogossek
7   */
8  public class Kurve_Sued_West extends Block
9  {
10     public Kurve_Sued_West()
11     {
12         r1='S';
13         r2='W';
14     }
15 }
```

Prellbock

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   * Prellbock
5   *
6   * @author Niklas Nogossek
7   */
8  public class Prellbock extends Block
9  {
10 public void addedToWorld(World w)
11 {
12     r2=r1;
13 }
14 public String getFreigabe(Actor start,char ausgangsrichtung, String
       zielhaltestelle)
15 /*
16  * gibt zurück: "0" - wenn Weg bis zum nächsten roten Signal, der nä
       chsten Haltestelle/Prellbock frei ist und alle Weichen bis dahin
       richtig gestellt sind
17  *              "1,xPos der Weiche;yPos der Weiche" - wenn Weiche
       gestellt werden muss
18  *              "2,xPos;yPos des Zugs" - wenn keine Freigabe (besetzte
       Schiene)
19  *              "3,xPos;yPos des letzten Blocks" - Fehlerhafte Strecke
20  */
21 {
22     Actor g;
23     char eRichtung; //eingangsrichtung in kommenden Block. zum Start F
           als Kontrollwert
24     switch(ausgangsrichtung){
25     case 'N':
```

```
26          g = getOneObjectAtOffset(0, -1, Block.class);
27          eRichtung='S';
28      break;
29      case 'O':
30          g = getOneObjectAtOffset(1, 0, Block.class);
31          eRichtung = 'W';
32      break;
33      case 'S':
34          g = getOneObjectAtOffset(0, 1, Block.class);
35          eRichtung = 'N';
36      break;
37      case 'W':
38          g = getOneObjectAtOffset(-1, 0, Block.class);
39          eRichtung = 'O';
40      break;
41      default:
42          g = null;
43          eRichtung ='F';
44          System.out.println("Unbekannte Richtung in getFreigabe!");
45      }
46      if (g != null)
47      {
48          Block nachbar= (Block) g;
49          if (nachbar.checkBlockEingang(g, eRichtung, false)!=true)  //ü
                berprüde ob nachbarblock validen Eingang hat
50          {
51              System.out.println("Fehlerhafte Strecke!");
52              return "3,"+g.getX()+";"+g.getY();
53          }
54          else
55          {
56              String[] merker= new String[0];
57              String freigabe = nachbar.checkFreigabe(g, eRichtung,
                    zielhaltestelle, merker);
58              if (freigabe.substring(0,1).equals("2"))  //falls besetzte
                    Schiene, überprüfe ob es der eigene Block ist
59              {
60                  int xPosBesetzt = Integer.parseInt(freigabe.substring(
                        freigabe.indexOf(',')+1, freigabe.indexOf(';')));
61                  int yPosBesetzt = Integer.parseInt(freigabe.substring(
                        freigabe.indexOf(';')+1, freigabe.length()));
62                  if ((xPosBesetzt == start.getX())&&(yPosBesetzt ==
                        start.getY()))  //falles startBlock= besetzter
                        Block
63                  {
64                      return "0";
65                  }
66                  else
67                  {
68                      System.out.println("besetzt: "+ xPosBesetzt+" / "+
                            yPosBesetzt);
69                      return freigabe;
70                  }
71              }
72              else
73              {
74              return freigabe;
75              }
```

```
76          }
77      }
78      else
79      {
80          System.out.println("Fehlerhafte Strecke!");
81          return "3,"+g.getX()+";"+g.getY();
82      }
83  }
84  }
```

Prellbock_Nord

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   * Prellbock_Nord
5   *
6   * @author Niklas Nogossek
7   */
8  public class Prellbock_Nord extends Prellbock
9  {
10     public Prellbock_Nord()
11     {
12         r1='S';
13     }
14 }
```

Prellbock_Ost

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   * Prellbock_Ost
5   *
6   * @author Niklas Nogossek
7   */
8  public class Prellbock_Ost extends Prellbock
9  {
10     public Prellbock_Ost()
11     {
12         r1='W';
13     }
14 }
```

Prellbock_Sued

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   * Prellbock_Sued
5   *
6   * @author Niklas Nogossek
7   */
8  public class Prellbock_Sued extends Prellbock
```

```
9  {
10     public Prellbock_Sued()
11     {
12         r1='N';
13     }
14 }
```

Prellbock_West

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   * Prellbock_West
5   *
6   * @author Niklas Nogossek
7   */
8  public class Prellbock_West extends Prellbock
9  {
10     public Prellbock_West()
11     {
12         r1='O';
13     }
14 }
```

Weiche

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Weiche
5   *
6   * @author Niklas Nogossek
7   */
8  public class Weiche extends Block
9  {
10 char stellung, r3;    //Weichen haben zusätzliche chars Stellung und 3.
       Richtung (r1 Eingangsrichtung für beide mgl Ausgänge, r2 andere
       Geradeneingang, r3 Kurveneingang)
11 String[] wege;
12 public Weiche()
13 {
14     wege = new String[0];
15     stellung = 'k';
16 }
```

stelleDieWeiche

```
1  public void stelleDieWeiche(Actor g)  //bekommt Weiche und stellt
       Weiche
2  {
3      String schiene = g.getClass().getName();
4      Weiche w=(Weiche) g;
5      if (w.stellung == 'k')
6      {
7          w.setImage(schiene+"_g.gif");
8          w.stellung = 'g';
```

```
9           Greenfoot.delay(50);
10      }
11      else
12      {
13          w.setImage(schiene+"_k.gif");
14          w.stellung = 'k';
15          Greenfoot.delay(50);
16      }
17  }
```

getWeicheRichtung

```
1   public char getWeicheRichtung(Actor a, char s)  //nimmt Weiche und
        Eingangsrichtung, gibt Ausgangsrichtung
2   {
3       Weiche w=(Weiche) a;
4       if (s == w.r3)
5       {
6           if (w.stellung == 'k')
7           {
8               return w.r1;
9           }
10          else
11          {
12              return 'H';
13          }
14      }
15      else
16      {
17          if (s == w.r1)
18          {
19              if (w.stellung == 'k')
20              {
21                  return w.r3;
22              }
23              else
24              {
25                  return w.r2;
26              }
27          }
28          else  // s== w.r2
29          {
30              if (w.stellung == 'k')
31              {
32                  return 'H';
33              }
34              else
35              {
36                  return w.r1;
37              }
38          }
39      }
40  }
```

checkEingang

```
1  public static boolean checkTheoretischenWeichenEingang(Actor g, char
       eingangsRichtung) //bekommt Weiche und Eingangsrichtung, gibt zurü
       ck ob Weiche da Eingang hat
2  {
3      Weiche w = (Weiche) g;
4      if (eingangsRichtung == w.r1 || eingangsRichtung == w.r2 ||
           eingangsRichtung == w.r3)
5      {
6          return true;
7      }
8      else
9      {
10         return false;
11     }
12 }
13 public boolean checkWeichenEingang(Actor a, char eingangsRichtung) //
       bekommt Weiche und Eingangsrichtung, gibt zurück ob Block da
       Eingang hat abhängig der Weichenstellung
14 {
15     Weiche w=(Weiche) a;
16     if ((eingangsRichtung == w.r1)||(stellung=='k'&&eingangsRichtung ==
           r3)||(stellung=='g'&&eingangsRichtung == r2))
17     {
18         return true;
19     }
20     else
21     {
22         return false;
23     }
24 }
```

mglAusgaengeWeiche

```
1  public static String mglAusgaengeWeiche(Actor weiche,char
       eingangsRichtung)    //bekommt Actor Weiche und char
       Eingangsrichtung und gibt als String (1 oder 2 Stellen) mögliche
       Ausgänge
2  {
3      Weiche w = (Weiche) weiche;
4      String mglAusgaenge;
5      if ((eingangsRichtung == w.r2)||(eingangsRichtung == w.r3))
6      {
7          mglAusgaenge=""+w.r1;
8          return mglAusgaenge;
9      }
10     else
11     {
12         if (eingangsRichtung == w.r1)
13         {
14             mglAusgaenge = ""+String.valueOf(w.r3)+String.valueOf(w.r2)
                   ;
15             return mglAusgaenge;
16         }
17         else
18         {
```

```
19              return "";
20          }
21      }
22  }
```

neuerWeg

```
1  public void neuerWeg(char ausgangsRichtung, String nameDerHaltestelle,
       int anzahlDerSchritte)  //Bekommt Weg (Richtung, Haltestelle und
       Schrittzahl) und soll spreichern, wenn neu oder kürzester Weh
2  {
3      String ausRichtung = Character.toString(ausgangsRichtung);
4      int i=0;
5      if (wegBekannt(ausgangsRichtung, nameDerHaltestelle)!=true) //wenn
           Weg unbekannt
6      {
7          String tempArray[]= new String[wege.length+1];
8          System.arraycopy(wege, 0, tempArray, 0, wege.length);
9          wege=tempArray;
10         wege[wege.length-1] = (ausRichtung + " " + nameDerHaltestelle +
               "; "+ anzahlDerSchritte);//speichere Weg zu Haltestelle in
               Form:"Richtung in die man die Weiche verlassen muss" "
               Haltestellenname"; AnzahlderSchritte
11     }
12     else
13     {
14         if(checkKuerzesterWeg(ausgangsRichtung, nameDerHaltestelle,
               anzahlDerSchritte))  //falls neuer Weg kürzer
15         {
16             ersetzeWeg(ausgangsRichtung, nameDerHaltestelle,
                   anzahlDerSchritte);
17         }
18     }
19 }
```

wegBekannt

```
1  private boolean wegBekannt(char bekanntAusgangsRichtung, String
       bekanntNameDerHaltestelle)  //Bekommt Weg (Richtung und Haltestelle
       ) und gibt zurück ob bekannt
2  {
3      String bekanntAusRichtung = Character.toString(
           bekanntAusgangsRichtung);
4      for (int j=0; j<wege.length; j++)
5      {
6          if ((wege[j] !=null)&&(wege[j].contains(bekanntAusRichtung + "
               " + bekanntNameDerHaltestelle)))
7          {
8              return true;
9          }
10     }
11     return false;
12 }
```

getWeg

```
1  public String getWeg(char ausgangsRichtung, String nameDerHaltestelle)
       //gibt Weg zur Haltestelle in Form:"Richtung in die man die Weiche
       verlassen muss" "Haltestellenname"; AnzahlderSchritte
2  {
3      String ausRichtung = Character.toString(ausgangsRichtung);
4      for (int j=0; j<wege.length; j++)
5      {
6          if ((this.wege[j] !=null)&&(this.wege[j].contains(ausRichtung +
               " " + nameDerHaltestelle)))
7          {
8              return this.wege[j];
9          }
10     }
11     return null;
12 }
```

checkKuerzesterWeg

```
1  public boolean checkKuerzesterWeg(char ausgangsRichtung, String
       nameHaltestelle, int neueSchrittzahl)  //bekommt Weg und
       Schrittzahl, gibt zurück ob kürzer als bekannter Weg
2  {
3      String bekannteARichtung = Character.toString(ausgangsRichtung);
4      for (int j=0; j<wege.length; j++)
5      {
6          if ((wege[j] !=null)&&(wege[j].contains(bekannteARichtung + " "
               + nameHaltestelle)))
7          {
8              int bekannteSchrittzahl = Integer.parseInt(wege[j].
                   substring(wege[j].indexOf(';')+2, wege[j].length()));
9              if(Integer.valueOf(neueSchrittzahl).compareTo(Integer.
                   valueOf(bekannteSchrittzahl)) < 0) //falls neue Weg kü
                   rzer als bekannter Weg
10             {
11                 return true;
12             }
13             else
14             {
15                 return false;
16             }
17         }
18     }
19     return true;
20 }
```

ersetzeWeg

```
1  private void ersetzeWeg(char gleicheAusgangsRichtung, String
       gleicheNameDerHaltestelle, int neueKurzeSchrittzahl)  //Bekommt Weg
       (Richtung, Haltestelle und Schrittzahl) und ersetzt bekannten lä
       ngeren Weg
2  {
3      String gleicheARichtung = Character.toString(
           gleicheAusgangsRichtung);
4      for (int e=0; e<wege.length; e++)
```

```
5       {
6           if ((wege[e] != null)&&(wege[e].contains(gleicheARichtung + " "
                + gleicheNameDerHaltestelle)))
7           {
8               wege[e] = (gleicheARichtung + " " +
                    gleicheNameDerHaltestelle +"; "+ neueKurzeSchrittzahl);
9           }
10      }
11  }
```

Weiche_Nord_Ost

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3   /**
4    * Weiche_Nord_Ost
5    *
6    * @author Niklas Nogossek
7    */
8   public class Weiche_Nord_Ost extends Weiche
9   {
10      public Weiche_Nord_Ost()
11      {
12          r1='N';
13          r2='S';
14          r3='O';
15      }
16      public void stelleWeiche()  //In Greenfoot-Welt Weiche vom Nutzer
            stellen lassen
17      {
18          String className = this.getClass().getSimpleName();
19          if (stellung == 'k')
20          {
21          setImage(className+"_g.gif");
22          stellung = 'g';
23          }
24          else{
25             setImage(className+"_k.gif");
26             stellung ='k';
27          }
28      }
29  }
```

Weiche_Nord_West

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3   /**
4    * Weiche_Nord_West
5    *
6    * @author Niklas Nogossek
7    */
8   public class Weiche_Nord_West extends Weiche
9   {
10      public Weiche_Nord_West()
11      {
12          r1='N';
```

```
13          r2='S';
14          r3='W';
15      }
16      public void stelleWeiche()//In Greenfoot-Welt Weiche vom Nutzer
            stellen lassen
17      {
18          String className = this.getClass().getSimpleName();
19          if (stellung == 'k')
20          {
21          setImage(className+"_g.gif");
22          stellung = 'g';
23          }
24          else{
25              setImage(className+"_k.gif");
26              stellung ='k';
27          }
28      }
29  }
```

Weiche_Ost_Nord

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3   /**
4    * Weiche_Ost_Nord
5    *
6    * @author Niklas Nogossek
7    */
8   public class Weiche_Ost_Nord extends Weiche
9   {
10      public Weiche_Ost_Nord()
11      {
12          r1='O';
13          r2='W';
14          r3='N';
15      }
16      public void stelleWeiche()//In Greenfoot-Welt Weiche vom Nutzer
            stellen lassen
17      {
18          String className = this.getClass().getSimpleName();
19          if (stellung == 'k')
20          {
21          setImage(className+"_g.gif");
22          stellung = 'g';
23          }
24          else{
25              setImage(className+"_k.gif");
26              stellung ='k';
27          }
28      }
29  }
```

Weiche_Ost_Sued

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3   /**
```

```
4  * Weiche_Ost_Sued
5  *
6  * @author Niklas Nogossek
7  */
8  public class Weiche_Ost_Sued extends Weiche
9  {
10     public Weiche_Ost_Sued()
11     {
12         r1='O';
13         r2='W';
14         r3='S';
15     }
16     public void stelleWeiche()//In Greenfoot-Welt Weiche vom Nutzer
           stellen lassen
17     {
18         String className = this.getClass().getSimpleName();
19         if (stellung == 'k')
20         {
21         setImage(className+"_g.gif");
22         stellung = 'g';
23         }
24         else{
25            setImage(className+"_k.gif");
26            stellung ='k';
27         }
28     }
29 }
```

Weiche_Sued_Ost

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4  * Weiche_Sued_Ost
5  *
6  * @author Niklas Nogossek
7  */
8  public class Weiche_Sued_Ost extends Weiche
9  {
10     public Weiche_Sued_Ost()
11     {
12         r1='S';
13         r2='N';
14         r3='O';
15     }
16     public void stelleWeiche()//In Greenfoot-Welt Weiche vom Nutzer
           stellen lassen
17     {
18         String className = this.getClass().getSimpleName();
19         if (stellung == 'k')
20         {
21         setImage(className+"_g.gif");
22         stellung = 'g';
23         }
24         else{
25            setImage(className+"_k.gif");
26            stellung ='k';
27         }
```

```
28      }
29  }
```

Weiche_Sued_West

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Weiche_Sued_West
5   *
6   * @author Niklas Nogossek
7   */
8  public class Weiche_Sued_West extends Weiche
9  {
10     public Weiche_Sued_West()
11     {
12         r1='S';
13         r2='N';
14         r3='W';
15     }
16     public void stelleWeiche()//In Greenfoot-Welt Weiche vom Nutzer
           stellen lassen
17     {
18         String className = this.getClass().getSimpleName();
19         if (stellung == 'k')
20         {
21         setImage(className+"_g.gif");
22         stellung = 'g';
23         }
24         else{
25            setImage(className+"_k.gif");
26            stellung ='k';
27         }
28     }
29 }
```

Weiche_West_Nord

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Weiche_West_Nord
5   *
6   * @author Niklas Nogossek
7   */
8  public class Weiche_West_Nord extends Weiche
9  {
10     public Weiche_West_Nord()
11     {
12         r1='W';
13         r2='O';
14         r3='N';
15     }
16     public void stelleWeiche()//In Greenfoot-Welt Weiche vom Nutzer
           stellen lassen
17     {
18         String className = this.getClass().getSimpleName();
```

```
19          if (stellung == 'k')
20          {
21          setImage(className+"_g.gif");
22          stellung = 'g';
23          }
24          else{
25             setImage(className+"_k.gif");
26             stellung ='k';
27          }
28      }
29  }
```

Weiche_West_Sued

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3   /**
4    * Weiche_West_Sued
5    *
6    * @author Niklas Nogossek
7    */
8   public class Weiche_West_Sued extends Weiche
9   {
10      public Weiche_West_Sued()
11      {
12          r1='W';
13          r2='O';
14          r3='S';
15      }
16      public void stelleWeiche()//In Greenfoot-Welt Weiche vom Nutzer
            stellen lassen
17      {
18          String className = this.getClass().getSimpleName();
19          if (stellung == 'k')
20          {
21          setImage(className+"_g.gif");
22          stellung = 'g';
23          }
24          else{
25             setImage(className+"_k.gif");
26             stellung ='k';
27          }
28      }
29  }
```

A.2.4 Haltestelle

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3   /**
4    * Haltestelle
5    *
6    * @author Niklas Nogossek
7    */
8   public class Haltestelle extends Actor
9   {
```

```
10      String name_Haltestelle;
11      boolean zeigeNamen;
12      public Haltestelle()
13      {
14          name_Haltestelle=getClass().getName();
15      }
```

addedToWorld

```
1       public void addedToWorld(World w)
2       {
3           haltestelleBenennen();
4           wegeFestlegen();
5           zeigeNamen=false;
6       }
```

Haltestellenname

```
1       private void haltestelleBenennen()
2       {
3           name_Haltestelle = "Haltestelle "+getX()+ " "+getY();
4       }
5       public String getname_Haltestelle()
6       {
7           return name_Haltestelle;
8       }
9       public void toggleNamenAnzeigen()
10      {
11          if (getClass().getName().contains("senkrecht"))
12          {
13              if (zeigeNamen)
14              {
15                  getWorld().showText(null, getX()-3, getY());
16                  zeigeNamen=false;
17              }
18              else
19              {
20                  getWorld().showText(name_Haltestelle, getX()-3, getY())
                        ;
21                  zeigeNamen=true;
22              }
23          }
24          else  //Haltestelle_waagerecht
25          {
26              if (zeigeNamen)
27              {
28                  getWorld().showText(null, getX(), getY()-1);
29                  zeigeNamen=false;
30              }
31              else
32              {
33                  getWorld().showText(name_Haltestelle, getX(), getY()-1)
                        ;
34                  zeigeNamen=true;
35              }
36          }
37      }
```

wegeFestlegen

```
1  public void wegeFestlegen()    //ruft hinzufuegenWegeWeichen in den
       entsprechenden Richtungen auf
2  {
3      Actor schiene = getOneObjectAtOffset(0, 0, Block.class);
4      String[] schonbekannteWeichen=null;
5      if (schiene != null)
6      {
7          Block gleis;
8          gleis = (Block) schiene;
9          if (getClass().getName().contains("senkrecht"))    //
               Haltestelle_senkrecht
10         {
11             gleis.hinzufuegenWegeWeichen(schiene, getX(), getY(), 'N',
                   name_Haltestelle, 0, schonbekannteWeichen);
12             gleis.hinzufuegenWegeWeichen(schiene, getX(), getY(), 'S',
                   name_Haltestelle, 0, schonbekannteWeichen);
13         }
14         else  //Haltestelle_waagerecht
15         {
16             gleis.hinzufuegenWegeWeichen(schiene, getX(), getY(), 'W',
                   name_Haltestelle, 0, schonbekannteWeichen);
17             gleis.hinzufuegenWegeWeichen(schiene, getX(), getY(), 'O',
                   name_Haltestelle, 0, schonbekannteWeichen);
18         }
19     }
20     else
21     {
22         System.out.println("Haltestelle befindet sich nicht an Schienen
               ");
23     }
24 }
```

getFreigabe

```
1  public String getFreigabe(Actor start,char ausgangsrichtung, String
       zielhaltestelle)
2  /*
3   * gibt zurück: "0" - wenn Weg bis zum nächsten roten Signal, der nä
       chsten Haltestelle/Prellbock frei ist und alle Weichen bis dahin
       richtig gestellt sind
4   *              "1,xPos der Weiche;yPos der Weiche" - wenn Weiche
       gestellt werden muss
5   *              "2,xPos;yPos des Zugs" - wenn keine Freigabe (besetzte
       Schiene)
6   *              "3,xPos;yPos des letzten Blocks" - Fehlerhafte Strecke
7   */
8  {
9      Actor g;
10     char eRichtung; //eingangsrichtung in kommenden Block. zum Start F
           als Kontrollwert
11     switch(ausgangsrichtung){
12     case 'N':
13         g = getOneObjectAtOffset(0, -1, Block.class);
14         eRichtung='S';
15     break;
```

```
16      case 'O':
17          g = getOneObjectAtOffset(1, 0, Block.class);
18          eRichtung = 'W';
19      break;
20      case 'S':
21          g = getOneObjectAtOffset(0, 1, Block.class);
22          eRichtung = 'N';
23      break;
24      case 'W':
25          g = getOneObjectAtOffset(-1, 0, Block.class);
26          eRichtung = 'O';
27      break;
28      default:
29          g = null;
30          eRichtung ='F';
31          System.out.println("Unbekannte Richtung in getFreigabe!");
32      }
33      if (g != null)
34      {
35          Block nachbar= (Block) g;
36          if (nachbar.checkBlockEingang(g, eRichtung, false)!=true)  //ü
                berprüde ob nachbarblock validen Eingang hat
37          {
38              System.out.println("Fehlerhafte Strecke!");
39              return "3,"+g.getX()+";"+g.getY();
40          }
41          else
42          {
43              String[] merker= new String[0];
44              String freigabe = nachbar.checkFreigabe(g, eRichtung,
                    zielhaltestelle, merker);
45              if (freigabe.substring(0,1).equals("2"))  //falls besetzte
                    Schiene, überprüfe ob es der eigene Block ist
46              {
47                  int xPosBesetzt = Integer.parseInt(freigabe.substring(
                        freigabe.indexOf(',')+1, freigabe.indexOf(';')));
48                  int yPosBesetzt = Integer.parseInt(freigabe.substring(
                        freigabe.indexOf(';')+1, freigabe.length()));
49                  if ((xPosBesetzt == start.getX())&&(yPosBesetzt ==
                        start.getY()))  //falles startBlock= besetzter
                        Block
50                  {
51                      return "0";
52                  }
53                  else
54                  {
55                      System.out.println("besetzt: "+ xPosBesetzt+" / "+
                            yPosBesetzt);
56                      return freigabe;
57                  }
58              }
59              else
60              {
61              return freigabe;
62              }
63          }
64      }
65      else
```

```
66      {
67          System.out.println("Fehlerhafte Strecke!");
68          return "3,"+g.getX()+";"+g.getY();
69      }
70  }
```

Haltestelle_senkrecht/ Haltestelle_waagerecht

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2   /**
3    * Haltestelle_senkrecht
4    *
5    * @author Niklas Nogossek
6    */
7   public class Haltestelle_senkrecht extends Haltestelle
8   {
9       public Haltestelle_senkrecht()
10      {
11
12      }
13  }
14
15  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
16  /**
17   * Haltestelle_waagerecht
18   *
19   * @author Niklas Nogossek
20   */
21  public class Haltestelle_waagerecht extends Haltestelle
22  {
23
24      public Haltestelle_waagerecht()
25      {
26
27      }
28  }
```

A.2.5 Signal

```
1   import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3   /**
4    * Signal
5    *
6    * @author Niklas Nogossek
7    */
8   public class Signal extends Actor
9   {
10      char farbe, signalrichtung;  //Signalfarbe kann r oder g sein,
            signalrichtung ist die Fahrtrichtung für die das Signal zustä
            ndig ist
11      public Signal()
12      {
13          farbe = 'r';
14      }
```

signalWechseln

```
1  public void signalWechseln(Actor signal)   //soll Signal von rot auf gr
   ün/ grün auf rot stellen
2  {
3      Signal s =(Signal) signal;
4      String signalart = signal.getClass().getName();
5      if (s.farbe == 'r')
6      {
7          s.setImage(signalart+"_g.gif");
8          s.farbe = 'g';
9      }
10     else
11     {
12         s.setImage(signalart+"_r.gif");
13         s.farbe ='r';
14         }
15 }
```

checkSignal

```
1      public static boolean checkSignal(Actor signal, char richtung)  //
       bekommt Actor Signal und Ausgangsrichtung gibt zurück ob
       relevantes Signal
2      {
3          Signal s = (Signal) signal;
4          if (richtung == s.signalrichtung)
5          {
6             return true;
7          }
8          else
9          {
10             return false;
11         }
12     }
```

checkRotesSignal

```
1      public static boolean checkRotesSignal(Actor signal, char richtung)
           //bekommt Actor Signal und Ausgangsrichtung gibt zurück ob
       relevantes rotes Signal
2      {
3          Signal s = (Signal) signal;
4          if (richtung == s.signalrichtung && s.farbe=='r')
5          {
6             return true;
7          }
8          else
9          {
10             return false;
11         }
12     }
```

getFreigabe

```
1  public String getFreigabe(Actor start,char ausgangsrichtung, String
       zielhaltestelle)
2  /*
3   * gibt zurück: "0" - wenn Weg bis zum nächsten roten Signal, der nä
       chsten Haltestelle/Prellbock frei ist und alle Weichen bis dahin
       richtig gestellt sind
4   *             "1,xPos der Weiche;yPos der Weiche" - wenn Weiche
       gestellt werden muss
5   *             "2,xPos;yPos des Zugs" - wenn keine Freigabe (besetzte
       Schiene)
6   *             "3,xPos;yPos des letzten Blocks" - Fehlerhafte Strecke
7   */
8  {
9      Actor g;
10     char eRichtung; //eingangsrichtung in kommenden Block. zum Start F
           als Kontrollwert
11     switch(ausgangsrichtung){
12     case 'N':
13         g = getOneObjectAtOffset(0, -1, Block.class);
14         eRichtung='S';
15     break;
16     case 'O':
17         g = getOneObjectAtOffset(1, 0, Block.class);
18         eRichtung = 'W';
19     break;
20     case 'S':
21         g = getOneObjectAtOffset(0, 1, Block.class);
22         eRichtung = 'N';
23     break;
24     case 'W':
25         g = getOneObjectAtOffset(-1, 0, Block.class);
26         eRichtung = 'O';
27     break;
28     default:
29         g = null;
30         eRichtung ='F';
31         System.out.println("Unbekannte Richtung in getFreigabe!");
32     }
33     if (g != null)
34     {
35         Block nachbar= (Block) g;
36         if (nachbar.checkBlockEingang(g, eRichtung, false)!=true)  //ü
               berprüde ob nachbarblock validen Eingang hat
37         {
38             System.out.println("Fehlerhafte Strecke!");
39             return "3,"+g.getX()+";"+g.getY();
40         }
41         else
42         {
43             String[] merker= new String[0];
44             String freigabe = nachbar.checkFreigabe(g, eRichtung,
                   zielhaltestelle, merker);
45             if (freigabe.substring(0,1).equals("2"))  //falls besetzte
                   Schiene, überprüfe ob es der eigene Block ist
46             {
47                 int xPosBesetzt = Integer.parseInt(freigabe.substring(
```

```
                    freigabe.indexOf(',')+1, freigabe.indexOf(';')));
48                  int yPosBesetzt = Integer.parseInt(freigabe.substring(
                    freigabe.indexOf(';')+1, freigabe.length()));
49                  if ((xPosBesetzt == start.getX())&&(yPosBesetzt ==
                    start.getY()))  //falles startBlock= besetzter
                    Block
50                  {
51                      return "0";
52                  }
53                  else
54                  {
55                      System.out.println("besetzt: "+ xPosBesetzt+" / "+
                        yPosBesetzt);
56                      return freigabe;
57                  }
58              }
59              else
60              {
61                  return freigabe;
62              }
63          }
64      }
65      else
66      {
67          System.out.println("Fehlerhafte Strecke!");
68          return "3,"+g.getX()+";"+g.getY();
```

Signal_Nord

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   *
5   * @author Niklas Nogossek
6   * @
7   */
8  public class Signal_Nord extends Signal
9  {
10     public Signal_Nord()
11     {
12         signalrichtung = 'N';
13     }
14     public void aendereFarbe()   //Signalfarbe vom Nutzer in Greenfoot
           ändern lassen
15     {
16         String className = this.getClass().getSimpleName();
17         if (farbe == 'r'){
18         setImage(className+"_g.gif");
19         farbe = 'g';
20     }
21         else{
22            setImage(className+"_r.gif");
23            farbe ='r';
24         }
25     }
26 }
```

Signal_Ost

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   *
5   * @author Niklas Nogossek
6   * @
7   */
8  public class Signal_Ost extends Signal
9  {
10     public Signal_Ost()
11     {
12         signalrichtung = 'O';
13     }
14     public void aendereFarbe()    //Signalfarbe vom Nutzer in Greenfoot
           ändern lassen
15     {
16         String className = this.getClass().getSimpleName();
17         if (farbe == 'r'){
18         setImage(className+"_g.gif");
19         farbe = 'g';
20     }
21         else{
22            setImage(className+"_r.gif");
23            farbe ='r';
24         }
25     }
26 }
```

Signal_Sued

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   *
5   * @author Niklas Nogossek
6   * @
7   */
8  public class Signal_Sued extends Signal
9  {
10     public Signal_Sued()
11     {
12         signalrichtung = 'S';
13     }
14     public void aendereFarbe()    //Signalfarbe vom Nutzer in Greenfoot
           ändern lassen
15     {
16         String className = this.getClass().getSimpleName();
17         if (farbe == 'r'){
18         setImage(className+"_g.gif");
19         farbe = 'g';
20     }
21         else{
22            setImage(className+"_r.gif");
```

```
23              farbe ='r';
24          }
25      }
26  }
```

Signal_West

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and
       MouseInfo)
2
3  /**
4   *
5   * @author Niklas Nogossek
6   * @
7   */
8  public class Signal_West extends Signal
9  {
10     public Signal_West()
11     {
12         signalrichtung = 'W';
13     }
14     public void aendereFarbe()     //Signalfarbe vom Nutzer in Greenfoot
           ändern lassen
15     {
16         String className = this.getClass().getSimpleName();
17         if (farbe == 'r'){
18         setImage(className+"_g.gif");
19         farbe = 'g';
20     }
21         else{
22            setImage(className+"_r.gif");
23            farbe ='r';
24         }
25     }
26 }
```

A.2.6 Zug

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2  import java.util.List;
3  /**
4   * Zug
5   *
6   * @author Niklas Nogossek
7   */
8  public class Zug extends Actor
9  {
10     String zugname;
11     char er;
12     String ziel;
13     public Zug()
14     {
15         zugname=getClass().getName();
16     }
17     public void setName()
18     {
```

```
19          zugname=Greenfoot.ask("Neuer Zugname:");
20      }
21      public void resetZug()
22      {
23          er = 'Z';
24          ziel = "0";
25      }
```

addedToWorld

```
1       public void addedToWorld(World w)
2       {
3           zugname=getClass().getName()+" "+getX()+ " "+getY();
4           er = 'Z';
5           ziel = "0";
6           Actor b = getOneObjectAtOffset(0, 0, Block.class);
7           Block block = (Block) b;
8           block.aendereZustand(true);
9       }
```

fahre

```
1   /*
2    * Richtung bitte mit 'N', 'O', 'S' oder 'W' angeben.
3    */
4   public void fahre(char richtung)  //Methode fahre zur Einführung in die
        Objektorientierung - Zug fährt immer ein Feld in angegebene
        Richtung
5   {
6       if (richtung == 'N')
7       {
8           setLocation(getX(), getY()-1);
9       }
10      else
11      {
12          if (richtung == 'O')
13          {
14              setLocation(getX()+1, getY());
15          }
16          else
17          {
18              if (richtung == 'S')
19              {
20                  setLocation(getX(), getY()+1);
21              }
22              else
23              {
24                  if (richtung == 'W')
25                  {
26                      setLocation(getX()-1, getY());
27                  }
28                  else
29                  {
30                      System.out.println("Ungültige Fahrtrichtung");
31                  }
32              }
33          }
```

```
34      }
35  }
```

fahreBlock

```
1   /*
2    * Richtung bitte mit 'N', 'O', 'S' oder 'W' angeben.
3    */
4   public void fahreBlock (char richtung)  ///Methode fahre zur Einführung
        in die Objektorientierung - Zug fährt immer ein Feld in angegebene
        Richtung wenn Blöcke es zulassen
5   {
6       if (richtung != 'N'&& richtung != 'O'&& richtung != 'S'&& richtung
            != 'W')
7       {
8           System.out.println("Ungültige Fahrtrichtung");
9       }
10      else{
11      Actor aktuellerBlock = getOneObjectAtOffset(0, 0, Block.class);  //
            aktueller Block
12      if (aktuellerBlock != null)  //Zug steht auf Schiene
13      {
14          Block aktBlock = (Block) aktuellerBlock;
15          if (aktBlock.checkBlockEingang(aktuellerBlock, richtung, true))
                //check ob Block einen Ausgang in geforderter Richtung
               hat
16          {
17              Actor naechsteBlock = aktBlock.getnextBlock(aktuellerBlock.
                    getX(), aktuellerBlock.getY(), richtung);  //bekomme nä
                    chsten Block
18              if (naechsteBlock != null)
19              {
20                  Block nBlock = (Block) naechsteBlock;
21                  if (nBlock.checkBlockEingang(naechsteBlock, nBlock.
                        richtungskonverter(richtung), true))  //check ob nä
                        chster Block Eingang in geforderter Richt ung hat
22                  {
23                      if (richtung == 'N')
24                      {
25                          setLocation(getX(), getY()-1);
26                      }
27                      else{
28                          if (richtung == 'O')
29                          {
30                              setLocation(getX()+1, getY());
31                          }
32                          else{
33                              if (richtung == 'S')
34                              {
35                                  setLocation(getX(), getY()+1);
36                              }
37                              else{
38                                  if (richtung == 'W')
39                                  {
40                                  setLocation(getX()-1, getY());
41                                  }
42                                  else{
43                                      System.out.println("Ungültige
```

```
                                        Fahrtrichtung");
44                                  }
45                              }
46                          }
47                      }
48                  }
49                  else //nächster Block kein Eingang in gefragter
                        Richtung
50                  {
51                      if (nBlock.istWeiche(naechsteBlock))  //check ob
                            naechster Block eine Weiche ist
52                      {
53                          if (nBlock.checkBlockEingang(nBlock, nBlock.
                                richtungskonverter(richtung), false))  //
                                check ob bei anderer Weichenstellung Block
                                in der Richtung Eingang hätte
54                          {
55                              System.out.println("Weiche falsch gestellt!
                                    ");
56                          }
57                          else
58                          {
59                              System.out.println("Zug steh in Richtung "+
                                    richtung+" in einer Sackgasse.");
60                          }
61                      }
62                      else  //keine Weiche
63                      {
64                          System.out.println("Zug steh in Richtung "+
                                richtung+" in einer Sackgasse.");
65                      }
66                  }
67              }
68              else  //kein nächster Block gefunden
69              {
70                  System.out.println("Zug steh in Richtung "+richtung+"
                        in einer Sackgasse.");
71              }
72          }
73          else //Block keinen Ausgang in dieser Richtung
74          {
75              if (aktBlock.istWeiche(aktuellerBlock))  //check ob
                    aktueller Block eine Weiche ist
76              {
77                  if (aktBlock.checkBlockEingang(aktuellerBlock, richtung
                        , false))  //check ob bei anderer Weichenstellung
                        Block in der Richtung Ausgang hätte
78                  {
79                      System.out.println("Weiche falsch gestellt!");
80                  }
81                  else
82                  {
83                      System.out.println("Zug kann den Block nicht in
                            Richtung " +richtung+" verlassen.");
84                  }
85              }
86              else  //keine Weiche
87              {
```

```
88                  System.out.println("Zug kann den Block nicht in
                        Richtung " +richtung+" verlassen.");
89              }
90          }
91      }
92      else
93      {
94         System.out.println("Zug steht nicht auf Schienen");
95      }
96      }
97  }
```

blockzustandaendern

```
1  private void blockzustandaendern(boolean zustand)
2  {
3      Actor g = getOneObjectAtOffset(0, 0, Block.class);
4      if (g != null)
5      {
6              Block b = (Block) g;
7              b.aendereZustand(zustand);
8      }
9  }
```

checkRotesSignal

```
1  private boolean checkRotesSignal(char richtung)  //bekommt
       Ausgangsrichtung checkt ob rotes Signal
2  {
3      List<Signal> listSignal;
4      listSignal = getObjectsAtOffset(0, 0, Signal.class);  //bekommt
           Liste aller Signale an diesem Block
5      boolean rotesSignal=false;
6      if (listSignal.isEmpty() == false)
7      {
8          for (int i = 0; i<listSignal.size(); i++)   //durchlaufe Liste
9          {
10             if (rotesSignal == false)    //falls noch kein passendes
                   Signal gefunden
11             {
12                 Actor msignal = listSignal.get(i);
13                 Signal checkMglSignal = (Signal) msignal;
14                 rotesSignal = checkMglSignal.checkRotesSignal(msignal,
                       richtung);
15             }
16         }
17     }
18     return rotesSignal;
19 }
```

getRotesSignal

```
1  private Actor getRotesSignal (char richtung)
2  {
3      List<Signal> listSignal;
4      listSignal = getObjectsAtOffset(0, 0, Signal.class);  //bekommt
           Liste aller Signale an diesem Block
```

```
5       if (listSignal.isEmpty() == false)
6       {
7           for (int i = 0; i<listSignal.size(); i++)   //durchlaufe Liste
8           {
9               Actor msignal = listSignal.get(i);
10              Signal checkMglSignal = (Signal) msignal;
11              if(checkMglSignal.checkRotesSignal(msignal, richtung))
12              {
13                  return msignal;
14              }
15          }
16      }
17      return null;
18  }
```

getHaltestelle

```
1   private Haltestelle getHaltestelle(String name)  //bekommt
        Haltestellenname, gibt Haltestelle
2   {
3       List<Haltestelle> listHaltestelle;
4       listHaltestelle = getWorld().getObjects(Haltestelle.class);  //
            bekommt Liste aller Signale an diesem Block
5       if (listHaltestelle.isEmpty() == false)
6       {
7           for (int i = 0; i<listHaltestelle.size(); i++)   //durchlaufe
                Liste
8           {
9               Haltestelle mglhstelle = listHaltestelle.get(i);
10              if (mglhstelle.getname_Haltestelle().equals(name))
11              {
12                  return mglhstelle;
13              }
14          }
15      }
16      return null;
17  }
```

fahreHaltestelle

```
1   private void fahreHaltestelle(char richtung, String nameZielhaltestelle
        )  //bekommt Richtung und Ziel, fährt um 1 in geforderte Richtung
2   {
3       switch(richtung){
4           case 'N':
5               blockzustandaendern(false);   //aktueller Block nicht mehr
                    besetzt
6               setLocation(getX(), getY()-1); //bewege Zug um 1 in gewü
                    nschte Richtung
7               blockzustandaendern(true);    //neuer Block jetzt besetzt
8               Greenfoot.delay(10);
9               fahreSchleifeZurHaltestelle('S', nameZielhaltestelle);  //
                    Schleife aufrufen
10              break;
11          case 'O':
12              blockzustandaendern(false);
13              setLocation(getX()+1, getY());
```

```
14              blockzustandaendern(true);
15              Greenfoot.delay(10);
16              fahreSchleifeZurHaltestelle('W', nameZielhaltestelle);
17          break;
18          case 'S':
19             blockzustandaendern(false);
20             setLocation(getX(), getY()+1);
21             blockzustandaendern(true);
22             Greenfoot.delay(10);
23             fahreSchleifeZurHaltestelle('N', nameZielhaltestelle);
24          break;
25          case 'W':
26              blockzustandaendern(false);
27              setLocation(getX()-1, getY());
28              blockzustandaendern(true);
29              Greenfoot.delay(10);
30              fahreSchleifeZurHaltestelle('O', nameZielhaltestelle);
31          break;
32          default:
33              System.out.println("Falsche Richtung in fahreHaltestelle.")
                    ;
34      }
35  }
```

fahreZuHaltestelle

```
1   /*
2    * Richtung bitte mit 'N', 'O', 'S' oder 'W' angeben. Ziel mit "
       Haltestellenname"
3    */
4   public void fahreZuHaltestelle(char startRichtung, String
       nameZielhaltestelle)
5   {
6   Haltestelle hstelle = getHaltestelle(nameZielhaltestelle);
7   if (hstelle==null)
8   {
9       System.out.println("Haltestelle: " +nameZielhaltestelle+ " ist
           nicht bekannt.");
10  }
11  else
12  {
13      hstelle.wegeFestlegen();
14      System.out.println("Fahre in Richtung: " + startRichtung + " zur
           Haltestelle " + nameZielhaltestelle);
15      Actor startBlock = getOneObjectAtOffset(0, 0, Block.class);  //ü
           berprüfe ob Zug auf Schiene steht
16      if (startBlock != null)              //steht auf Schiene
17          {
18              Block stBlock = (Block) startBlock;
19              if (stBlock.checkBlockEingang(startBlock, startRichtung,
                   true))    //Überprüfe ob die Schiene einen Eingang/
                   Ausgang in die gewünschte Richtung hat
20              {
21                  if (checkRotesSignal(startRichtung) != true)  //Überprü
                       fe ob Zug vor rotem Signal steht
22                  {
23                      Actor nextBlock = stBlock.getnextBlock(startBlock.
                           getX(), startBlock.getY(), startRichtung);  //
```

```
                finde den nächsten Block und speicher ihn in
                nextBlock
24          if(nextBlock != startBlock)       //getnextBlock
                gibt aktuellen Block zurück, wenn in Sackgasse
25          {
26              Block nxtBlock = (Block) nextBlock;
27              if (nxtBlock.checkBlockEingang(nextBlock,
                    nxtBlock.richtungskonverter(startRichtung),
                     true)) //checkt ob nächste Block Eingang
                    in der Richtung hat
28              {
29                  String[] merker= new String[0];
30                  String freigabe = stBlock.checkFreigabe(
                        nextBlock, nxtBlock.richtungskonverter(
                        startRichtung), nameZielhaltestelle,
                        merker);
31                  char freigabecode = freigabe.charAt(0);
32                  switch(freigabecode){
33                      case '0':
34                          fahreHaltestelle(startRichtung,
                                nameZielhaltestelle);
35                          break;
36                      case '1':
37                          int xPosWeiche = Integer.parseInt(
                                freigabe.substring(freigabe.
                                indexOf(',')+1, freigabe.
                                indexOf(';')));
38                          int yPosWeiche = Integer.parseInt(
                                freigabe.substring(freigabe.
                                indexOf(';')+1, freigabe.length
                                ()));
39                          System.out.println("Keine Freigabe.
                                 Weiche an der Stelle("+
                                xPosWeiche+"|"+yPosWeiche+")
                                muss gestellt werden.");
40                          break;
41                      case '2':
42                          int xPosBesetzt = Integer.parseInt(
                                freigabe.substring(freigabe.
                                indexOf(',')+1, freigabe.
                                indexOf(';')));
43                          int yPosBesetzt = Integer.parseInt(
                                freigabe.substring(freigabe.
                                indexOf(';')+1, freigabe.length
                                ()));
44                          if ((xPosBesetzt == startBlock.getX
                                ())&&(yPosBesetzt == startBlock
                                .getY()))  //falles startBlock=
                                 besetzter Block
45                          {
46                              System.out.println("Keine
                                    Freigabe. Zug fährt im
                                    Kreis ohne Möglichkeit ihn
                                    zu stoppen.");
47                          }
48                          else
49                          {
50                              System.out.println("Keine
```

```
                                        Freigabe: besetzt: "+
                                        xPosBesetzt+" / "+
                                        yPosBesetzt);
51                                  }
52                                  break;
53                              case '3':
54                                  int xPosletzteBlock = Integer.
                                        parseInt(freigabe.substring(
                                        freigabe.indexOf(',')+1,
                                        freigabe.indexOf(';')));
55                                  int yPosletzteBlock = Integer.
                                        parseInt(freigabe.substring(
                                        freigabe.indexOf(';')+1,
                                        freigabe.length()));
56                                  System.out.println("Keine Freigabe.
                                         Strecke nach ("+
                                        xPosletzteBlock+"|"+
                                        yPosletzteBlock+") unvollstä
                                        ndig.");
57                                  break;
58                              default:
59                                  System.out.println("Keine Freigabe.
                                        ");
60                          }
61                      }
62                      else  //Sackgasse oder Weiche falsch gestellt
63                      {
64                          if (nxtBlock.checkBlockEingang(nextBlock,
                                nxtBlock.richtungskonverter(
                                startRichtung), false)) //steht vor
                                Weiche die falsch gestellt ist
65                          {
66                              if(nxtBlock.userFrage("Weiche falsch"))
67                              {
68                                  Greenfoot.delay(20);
69                                  Weiche kommeWeiche = (Weiche)
                                        nextBlock;
70                                  kommeWeiche.stelleDieWeiche(
                                        nextBlock);
71                                  fahreZuHaltestelle(startRichtung,
                                        nameZielhaltestelle);//neu
                                        aufrufen mit jetzt gestellter
                                        Weiche
72                              }
73                              else
74                              {
75                                  System.out.println("Zug bekommt
                                        keine Freigabe zur Abfahrt.");
76                              }
77                          }
78                          else  //Sackgasse
79                          {
80                              System.out.println("Zug steht in einer
                                    Sackgasse");
81                          }
82                      }
83                  }
84                  else  //Sackgasse
```

```
85                      {
86                          System.out.println("Zug steht in einer
                               Sackgasse");
87                      }
88                  }
89                  else
90                  {
91                      Actor rotessignal = getRotesSignal(startRichtung);
92                      Signal rs = (Signal) rotessignal;
93                      if (rs.getFreigabe(rotessignal, startRichtung,
                           nameZielhaltestelle).equals("0"))
94                      {
95                          if(stBlock.userFrage("Signal"))
96                          {
97                              Greenfoot.delay(20);
98                              rs.signalWechseln(rotessignal);
99                              fahreHaltestelle(startRichtung,
                                   nameZielhaltestelle);
100                         }
101                     }
102                     else{
103                     System.out.println("Zug hat keine Freigabe");
104                     }
105                 }
106             }
107             else  //Startblockausgang passt nicht
108             {
109                 if (stBlock.checkBlockEingang(startBlock, startRichtung
                        , false)) //Zug steht auf Weiche
110                 {
111                     System.out.println("Zug steht auf falschgestellter
                           Weiche um in Richtung "+startRichtung+" zu
                           fahren.");
112                 }
113                 else
114                 {
115                     System.out.println("Kann von hier nicht in Richtung
                          " +startRichtung+ " fahren.");
116                 }
117             }
118         }
119         else
120         {
121             System.out.println("Zug steht auf keiner Schiene.");
122         }
123 }
124 }
```

fahreSchleifeZurHaltestelle

```
1  private void fahreSchleifeZurHaltestelle(char eintrittsRichtung, String
       nameZielhaltestelle)
2  {
3      Actor aktuellerBlock = getOneObjectAtOffset(0, 0, Block.class);  //
           überprüfe ob Zug auf Schiene steht
4      if (aktuellerBlock!=null)//steht auf Schiene
5      {
6          Block aktBlock = (Block) aktuellerBlock;
```

```
7           Actor prevBlock = aktBlock.getnextBlock(aktuellerBlock.getX(),
                aktuellerBlock.getY(), eintrittsRichtung);
8           Block pBlock= (Block) prevBlock;
9           List<Signal> listSignal;
10          listSignal = getObjectsAtOffset(prevBlock.getX()-getX(),
                prevBlock.getY()-getY(), Signal.class);  //bekommt Liste
                aller Signale an diesem Block
11          if (listSignal.isEmpty() == false)
12          {
13              for (int i = 0; i<listSignal.size(); i++)   //durchlaufe
                    Liste
14              {
15                  Actor prevsignal = listSignal.get(i);
16                  Signal prevMglSignal = (Signal) prevsignal;
17                  if(prevMglSignal.checkSignal(prevsignal, pBlock.
                        richtungskonverter(eintrittsRichtung)))  //falls
                        vorheriges Signal
18                  {
19                      prevMglSignal.signalWechseln(prevsignal);
                                //Wieder auf rot stellen
20                  }
21              }
22          }
23          Actor aktuelleHaltestelle = getOneObjectAtOffset(0, 0,
                Haltestelle.class);  //überprüfe ob Zug an Haltestelle
                steht
24          Haltestelle aktHaltestelle = null;
25          if (aktuelleHaltestelle != null)
26          {
27              aktHaltestelle = (Haltestelle) aktuelleHaltestelle;
28          }
29          if (nameZielhaltestelle != null)
30          {
31              if ((aktHaltestelle==null)||(nameZielhaltestelle.equals(
                    aktHaltestelle.getname_Haltestelle()) != true)) //falls
                     Ziel nicht erreicht
32              {
33                  char aktAusgangsRichtung = aktBlock.getBlockRichtung(
                        aktuellerBlock, eintrittsRichtung);     //bekomme
                        Ausgangsrichtung des aktuellen Blocks
34                  if ((aktHaltestelle == null)||(aktHaltestelle.
                        getFreigabe(aktuellerBlock, aktAusgangsRichtung,
                        null).equals("0")))  //falls keine Haltestelle oder
                         Zug Freigabe hat
35                  {
36                      if (checkRotesSignal(aktAusgangsRichtung) != true)
                            //Überprüfe ob Zug vor rotem Signal steht
37                      {
38                          Actor nextBlock = aktBlock.getnextBlock(
                                aktuellerBlock.getX(), aktuellerBlock.getY
                                (), aktAusgangsRichtung);  //finde den nä
                                chsten Block und speicher ihn in nextBlock
39                          if(nextBlock != aktuellerBlock)  //getnextBlock
                                gibt aktuellen Block zurück, wenn in
                                Sackgasse
40                          {
41                              Block nxtBlock = (Block) nextBlock;
42                              if (nxtBlock.checkBlockEingang(nextBlock,
```

```
                    nxtBlock.richtungskonverter(
                    aktAusgangsRichtung), true)) //checkt
                    ob nächste Block Eingang in der
                    Richtung hat
43                  {
44                      Actor aktuellerPrellbock =
                            getOneObjectAtOffset(aktuellerBlock
                            .getX()-getX(), aktuellerBlock.getY
                            ()-getY(), Prellbock.class);
45                      if (aktuellerPrellbock != null)  //ü
                            berprüfe ob Zug an einem Prellbock
                            steht
46                      {
47                          if(nxtBlock.userFrage("Prellbock"))
48                          {
49                              Prellbock aktPrellbock = (
                                    Prellbock)
                                    aktuellerPrellbock;
50                              if(aktPrellbock.getFreigabe(
                                    aktuellerPrellbock,
                                    aktAusgangsRichtung,
                                    nameZielhaltestelle).equals
                                    ("0"))
51                              {
52                                  fahreHaltestelle(
                                        aktAusgangsRichtung,
                                        nameZielhaltestelle);
53                              }
54                              else
55                              {
56                                  System.out.println("Zug hat
                                        keine Freigabe den
                                        Prellbock zu verlassen"
                                        );
57                              }
58                          } //Zug soll an Prellbock stehen
                                bleiben
59                          else
60                          {
61                              System.out.println("Zug an
                                    Prellbock stehen geblieben"
                                    );
62                          }
63                      } //kein Prellbock
64                      else
65                      {
66                          fahreHaltestelle(
                                aktAusgangsRichtung,
                                nameZielhaltestelle);
67                      }
68                  }
69                  else  //nächste Block keinen passenden
                        Eingang
70                  {
71                      System.out.println("Zug steht in einer
                            Sackgasse");
72                  }
73              }
```

```
74                          else  //kein nächster Block
75                          {
76                              System.out.println("Zug steht in einer
                                    Sackgasse");
77                          }
78                      }
79                      else  //falls Zug vor rotem Signal steht
80                      {
81                          Actor rotessignal = getRotesSignal(
                                aktAusgangsRichtung);
82                          Signal rs = (Signal) rotessignal;
83                          if (rs.getFreigabe(rotessignal,
                                aktAusgangsRichtung, nameZielhaltestelle).
                                equals("0"))
84                          {
85                              if(pBlock.userFrage("Signal"))
86                              {
87                                  Greenfoot.delay(20);
88                                  rs.signalWechseln(rotessignal);
89                                  fahreSchleifeZurHaltestelle(
                                        eintrittsRichtung,
                                        nameZielhaltestelle);
90                              }
91                          }
92                          else
93                          {
94                              System.out.println("Zug hat keine Freigabe"
                                    );
95                          }
96                      }
97                  }
98                  else   //Haltestelle ohne Freigabe
99                  {
100                     System.out.println("Zug hat keine Freigabe um die
                            Haltestelle zu verlassen");
101                 }
102             }
103             else
104             {
105                 System.out.println("Zug hat Ziel erreicht");
106             }
107         }
108         else  //keine Zielhaltestelle
109         {
110             if (aktuelleHaltestelle != null)
111             {
112                 if(aktBlock.userFrage("Haltestelle"))
113                 {
114                     System.out.println("Zug hat gehalten");
115                 }
116                 else   //soll weiterfahren
117                 {
118                     char aktAusgangsRichtung = aktBlock.
                            getBlockRichtung(aktuellerBlock,
                            eintrittsRichtung);     //bekomme
                            Ausgangsrichtung des aktuellen Blocks
119                     if(aktHaltestelle.getFreigabe(aktuellerBlock,
                            aktAusgangsRichtung, null).equals("0"))
```

```
120                     {
121                         Actor nextBlock = aktBlock.getnextBlock(
                               aktuellerBlock.getX(), aktuellerBlock.getY
                               (), aktAusgangsRichtung);  //finde den nä
                               chsten Block und speicher ihn in nextBlock
122                         if(nextBlock != aktuellerBlock)  //getnextBlock
                               gibt aktuellen Block zurück, wenn in
                               Sackgasse
123                         {
124                             Block nxtBlock = (Block) nextBlock;
125                             if (nxtBlock.checkBlockEingang(nextBlock,
                                   nxtBlock.richtungskonverter(
                                   aktAusgangsRichtung), true)) //checkt
                                   ob nächste Block Eingang in der
                                   Richtung hat
126                             {
127                                 fahreHaltestelle(aktAusgangsRichtung,
                                       nameZielhaltestelle);
128                             }
129                             else
130                             {
131                                 System.out.println("Zug steht in einer
                                       Sackgasse");
132                             }
133                         }
134                         else
135                         {
136                             System.out.println("Zug steht in einer
                                   Sackgasse");
137                         }
138                     }
139                     else
140                     {
141                         System.out.println("Zug hat keine Freigabe um
                               die Haltestelle zu verlassen");
142                     }
143                 }
144             }
145             else //keine Haltestelle
146             {
147                 char aktAusgangsRichtung = aktBlock.getBlockRichtung(
                       aktuellerBlock, eintrittsRichtung);    //bekomme
                       Ausgangsrichtung des aktuellen Blocks
148             if (checkRotesSignal(aktAusgangsRichtung) != true)  //Ü
                   berprüfe ob Zug vor rotem Signal steht
149                 {
150                     Actor nextBlock = aktBlock.getnextBlock(
                           aktuellerBlock.getX(), aktuellerBlock.getY(),
                           aktAusgangsRichtung);  //finde den nächsten
                           Block und speicher ihn in nextBlock
151                     if(nextBlock != aktuellerBlock)  //getnextBlock
                           gibt aktuellen Block zurück, wenn in Sackgasse
152                     {
153                         Block nxtBlock = (Block) nextBlock;
154                         if (nxtBlock.checkBlockEingang(nextBlock,
                               nxtBlock.richtungskonverter(
                               aktAusgangsRichtung), true)) //checkt ob nä
                               chste Block Eingang in der Richtung hat
```

```
155                         {
156                             Actor aktuellerPrellbock =
                                    getOneObjectAtOffset(aktuellerBlock
                                    .getX()-getX(), aktuellerBlock.getY
                                    ()-getY(), Prellbock.class);
157                             if (aktuellerPrellbock != null)  //ü
                                    berprüfe ob Zug an einem Prellbock
                                    steht
158                             {
159                                 if(aktBlock.userFrage("Prellbock"))
160                                 {
161                                     Prellbock aktPrellbock = (
                                            Prellbock)
                                            aktuellerPrellbock;
162                                     if(aktPrellbock.getFreigabe(
                                            aktuellerPrellbock,
                                            aktAusgangsRichtung,
                                            nameZielhaltestelle).equals
                                            ("0"))
163                                     {
164                                         fahreHaltestelle(
                                                aktAusgangsRichtung,
                                                nameZielhaltestelle);
165                                     }
166                                     else
167                                     {
168                                         System.out.println("Zug hat
                                                keine Freigabe den
                                                Prellbock zu verlassen"
                                                );
169                                     }
170                                 }
171                                 else
172                                 {
173                                     System.out.println("Zug an
                                            Prellbock stehen geblieben"
                                            );
174                                 }
175                             }
176                             else
177                             {
178                                 fahreHaltestelle(
                                        aktAusgangsRichtung,
                                        nameZielhaltestelle);
179                             }
180                         }
181                         else
182                         {
183                             System.out.println("Zug steht in einer
                                    Sackgasse");
184                         }
185                     }
186                     else
187                     {
188                         System.out.println("Zug steht in einer
                                Sackgasse");
189                     }
190                 }
```

```
191                 else   //falls Zug vor rotem Signal steht
192                 {
193                     Actor rotessignal = getRotesSignal(
                            aktAusgangsRichtung);
194                     Signal rs = (Signal) rotessignal;
195                     if (rs.getFreigabe(rotessignal, aktAusgangsRichtung
                            , nameZielhaltestelle).equals("0"))
196                     {
197                         if(aktBlock.userFrage("Signal"))
198                         {
199                             Greenfoot.delay(20);
200                             rs.signalWechseln(rotessignal);
201                             fahreSchleifeZurHaltestelle(
                                    eintrittsRichtung, nameZielhaltestelle)
                                    ;
202                         }
203                     }
204                     else
205                     {
206                         System.out.println("Zug hat keine Freigabe");
207                     }
208                 }
209             }
210         }
211     }
212     else //falls aktueller Block == null
213     {
214        System.out.println("Zug Steht nicht auf keiner Schiene.");
215     }
216 }
```

fahreEwigRichtung

```
1  /*
2   * Richtung bitte mit 'N', 'O', 'S' oder 'W' angeben.
3   */
4  public void fahreEwigRichtung(char fahrtRichtung)
5  {
6      Actor startBlock = getOneObjectAtOffset(0, 0, Block.class);  //ü
           berprüfe ob Zug auf Schiene steht
7      if (startBlock != null)             //steht auf Schiene
8          {
9              Block stBlock = (Block) startBlock;
10             if (stBlock.checkBlockEingang(startBlock, fahrtRichtung,
                   true))    //Überprüfe ob die Schiene einen Eingang/
                   Ausgang in die gewünschte Richtung hat
11             {
12                 if (checkRotesSignal(fahrtRichtung) != true)  //Überprü
                       fe ob Zug vor rotem Signal steht
13                 {
14                     Actor nextBlock = stBlock.getnextBlock(startBlock.
                           getX(), startBlock.getY(), fahrtRichtung);  //
                           finde den nächsten Block und speicher ihn in
                           nextBlock
15                     if(nextBlock != startBlock){
                                                                  //
                           getnextBlock gibt aktuellen Block zurück, wenn
                           in Sackgasse
```

```
16                          Block nxtBlock = (Block) nextBlock;
17                          if (nxtBlock.checkBlockEingang(nextBlock,
                                nxtBlock.richtungskonverter(fahrtRichtung),
                                 true)) //checkt ob nächste Block Eingang
                                in der Richtung hat
18                          {
19                                Actor startHaltestelle =
                                    getOneObjectAtOffset(0, 0,
                                    Haltestelle.class);
20                                if (startHaltestelle == null)   //ü
                                    berprüfe ob Zug in Haltestelle
21                                {
22                                    Actor startPrellbock =
                                        getOneObjectAtOffset(0, 0,
                                        Prellbock.class);
23                                    if (startPrellbock == null) //ü
                                        berprüfe ob Zug an Prellbock
24                                    {
25                                        String[] merker= new String[0];
26                                        String freigabe = stBlock.
                                            checkFreigabe(nextBlock,
                                            nxtBlock.richtungskonverter
                                            (fahrtRichtung), null,
                                            merker);
27                                        char freigabecode = freigabe.
                                            charAt(0);
28                                        switch(freigabecode){
29                                        case '0':
30                                            fahreHaltestelle(
                                                fahrtRichtung, null);
31                                            break;
32                                        case '1':
33                                            int xPosWeiche = Integer.
                                                parseInt(freigabe.
                                                substring(freigabe.
                                                indexOf(',')+1,
                                                freigabe.indexOf(';')))
                                                ;
34                                            int yPosWeiche = Integer.
                                                parseInt(freigabe.
                                                substring(freigabe.
                                                indexOf(';')+1,
                                                freigabe.length()));
35                                            System.out.println("Keine
                                                Freigabe. Weiche an der
                                                 Stelle("+xPosWeiche+"|
                                                "+yPosWeiche+") muss
                                                gestellt werden.");
36                                            break;
37                                        case '2':
38                                            int xPosBesetzt = Integer.
                                                parseInt(freigabe.
                                                substring(freigabe.
                                                indexOf(',')+1,
                                                freigabe.indexOf(';')))
                                                ;
39                                            int yPosBesetzt = Integer.
                                                parseInt(freigabe.
```

```
                                    substring(freigabe.
                                    indexOf(';')+1,
                                    freigabe.length()));
40                                  if ((xPosBesetzt ==
                                    startBlock.getX())&&(
                                    yPosBesetzt ==
                                    startBlock.getY())) //
                                    falles startBlock=
                                    besetzter Block
41                                  {
42                                      System.out.println("
                                        Keine Freigabe. Zug
                                         fährt im Kreis
                                        ohne Möglichkeit
                                        ihn zu stoppen.");
43                                  }
44                                  else
45                                  {
46                                      System.out.println("
                                        Keine Freigabe:
                                        besetzt: "+
                                        xPosBesetzt+" / "+
                                        yPosBesetzt);
47                                  }
48                                  break;
49                              case '3':
50                                  int xPosletzteBlock =
                                    Integer.parseInt(
                                    freigabe.substring(
                                    freigabe.indexOf(',')
                                    +1, freigabe.indexOf(';
                                    ')));
51                                  int yPosletzteBlock =
                                    Integer.parseInt(
                                    freigabe.substring(
                                    freigabe.indexOf(';')
                                    +1, freigabe.length()))
                                    ;
52                                  System.out.println("Keine
                                    Freigabe. Strecke nach
                                    ("+xPosletzteBlock+"|"+
                                    yPosletzteBlock+")
                                    unvollständig.");
53                                  break;
54                              default:
55                                  System.out.println("Keine
                                    Freigabe.");
56                              }
57                          }
58                          else //Zug steht an Prellbock
59                          {
60                              Prellbock stPrellbock = (
                                Prellbock) startPrellbock;
61                              if (stPrellbock.getFreigabe(
                                startBlock, fahrtRichtung,
                                null).equals("0"))
62                              {
63                                  fahreHaltestelle(
```

```
                                        fahrtRichtung, null);
64                                  }
65                                  else
66                                  {
67                                      System.out.println("Zug hat
                                            keine Freigabe um
                                           Prellbock zu verlassen"
                                           );
68                                  }
69                              }
70                          }
71                          else //steht in Haltestelle
72                          {
73                              Haltestelle stHaltestelle= (
                                    Haltestelle) startHaltestelle;
74                              if (stHaltestelle.getFreigabe(
                                    startBlock, fahrtRichtung, null
                                    ).equals("0"))
75                              {
76                                  fahreHaltestelle(fahrtRichtung,
                                        null);
77                              }
78                              else
79                              {
80                                  System.out.println("Zug hat
                                        keine Freigabe um die
                                        Haltestelle zu verlassen");
81                              }
82                          }
83                      }
84                      else  //Sackgasse oder Weiche falsch gestellt
85                      {
86                          if (nxtBlock.checkBlockEingang(nextBlock,
                                nxtBlock.richtungskonverter(
                                fahrtRichtung), false)) //steht vor
                                Weiche die falsch gestellt ist
87                          {
88                              if(nxtBlock.userFrage("Weiche falsch"))
89                              {
90                                  Greenfoot.delay(20);
91                                  Weiche kommeWeiche = (Weiche)
                                        nextBlock;
92                                  kommeWeiche.stelleDieWeiche(
                                        nextBlock);
93                                  fahreEwigRichtung(fahrtRichtung);//
                                        neu aufrufen mit jetzt
                                        gestellter Weiche
94                              }
95                              else
96                              {
97                                  System.out.println("Zug bekommt
                                        keine Freigabe zur Abfahrt.");
98                              }
99                          }
100                         else  //Sackgasse
101                         {
102                             System.out.println("Zug steht in einer
                                    Sackgasse");
```

```
103                             }
104                         }
105                     }
106                     else
107                     {
108                         System.out.println("Zug steht in einer
                               Sackgasse");
109                     }
110                 }
111                 else  //vor rotem Signal
112                 {
113                     Actor rotessignal = getRotesSignal(fahrtRichtung);
114                     Signal rs = (Signal) rotessignal;
115                     if (rs.getFreigabe(rotessignal, fahrtRichtung, null
                           ).equals("0"))
116                     {
117                         if(stBlock.userFrage("Signal"))
118                         {
119                             Greenfoot.delay(20);
120                             rs.signalWechseln(rotessignal);
121                             fahreHaltestelle(fahrtRichtung, null);
122                         }
123                     }
124                     else
125                     {
126                         System.out.println("Zug hat keine Freigabe");
127                     }
128                 }
129             }
130             else
131             {
132                 System.out.println("Kann von hier nicht in Richtung " +
                       fahrtRichtung+ " fahren.");
133             }
134         }
135         else
136         {
137             System.out.println("Zug steht auf keiner Schiene.");
138         }
139 }
```

act und actfahre

```
1  public void act()
2  {
3      actfahre();
4  }
5  public void actfahre()
6  {
7      char fahrtrichtung;
8      Actor b = getOneObjectAtOffset(0, 0, Block.class);
9      if(b!=null)
10     {
11     Block block = (Block) b;
12     block.setbesetzt(b);
13     if (er == 'Z')   //check ob Zug in Startkonfiguration
14     {
15         String gibfahrtrichtung = Greenfoot.ask("Fahrtrichtung für "+
```

```
            zugname+" ? (N/O/S/W) (Halt zum Anhalten)");
16          if ((gibfahrtrichtung.equals("N"))||(gibfahrtrichtung.equals("O
            "))||(gibfahrtrichtung.equals("S"))||(gibfahrtrichtung.
            equals("W")))
17          {
18              fahrtrichtung= gibfahrtrichtung.charAt(0);
19              if(block.checkBlockEingang(b, fahrtrichtung, true))
20              {
21                  er=block.getBlockRichtung(b, fahrtrichtung);
22                  String gibZiel = Greenfoot.ask("Zielhaltestelle für "+
                        zugname+"? Im Falle keiner Zielhaltestelle bitte 0
                        antworten.");
23                  if (gibZiel.equals("0"))
24                  {
25                      ziel = null;
26                  }
27                  else
28                  {
29                      Haltestelle hstelle = getHaltestelle(gibZiel);
30                      if (hstelle==null)
31                      {
32                          System.out.println("Haltestelle: " +gibZiel+ "
                                ist nicht bekannt.");
33                          er='Z';
34                          actfahre();
35                      }
36                      else
37                      {
38                          ziel = gibZiel;
39                          hstelle.wegeFestlegen();
40                      }
41                  }
42                  if (checkRotesSignal(fahrtrichtung) != true)  //Überprü
                        fe ob Zug vor rotem Signal steht
43                  {
44                      Actor nextBlock = block.getnextBlock(block.getX(),
                            block.getY(), fahrtrichtung);  //finde den nä
                            chsten Block und speicher ihn in nextBlock
45                      if(nextBlock != block)       //getnextBlock gibt
                            aktuellen Block zurück, wenn in Sackgasse
46                      {
47                          Block nxtBlock = (Block) nextBlock;
48                          if (nxtBlock.checkBlockEingang(nextBlock,
                                nxtBlock.richtungskonverter(fahrtrichtung),
                                 true)) //checkt ob nächste Block Eingang
                                in der Richtung hat
49                          {
50                              String[] merker= new String[0];
51                              String freigabe = block.checkFreigabe(
                                    nextBlock, nxtBlock.richtungskonverter(
                                    fahrtrichtung), ziel, merker);
52                              char freigabecode = freigabe.charAt(0);
53                              switch(freigabecode){
54                                  case '0':
55                                      fahreBlock(fahrtrichtung);
56                                      Greenfoot.delay(10);
57                                      er=block.richtungskonverter(
                                            fahrtrichtung);
```

```
58                                  break;
59                              case '1':
60                                  int xPosWeiche = Integer.parseInt(
                                        freigabe.substring(freigabe.
                                        indexOf(',')+1, freigabe.
                                        indexOf(';')));
61                                  int yPosWeiche = Integer.parseInt(
                                        freigabe.substring(freigabe.
                                        indexOf(';')+1, freigabe.length
                                        ()));
62                                  System.out.println("Keine Freigabe.
                                         Weiche an der Stelle("+
                                        xPosWeiche+"|"+yPosWeiche+")
                                        muss gestellt werden.");
63                                  break;
64                              case '2':
65                                  int xPosBesetzt = Integer.parseInt(
                                        freigabe.substring(freigabe.
                                        indexOf(',')+1, freigabe.
                                        indexOf(';')));
66                                  int yPosBesetzt = Integer.parseInt(
                                        freigabe.substring(freigabe.
                                        indexOf(';')+1, freigabe.length
                                        ()));
67                                  if ((xPosBesetzt == block.getX())
                                        &&(yPosBesetzt == block.getY())
                                        ) //falles startBlock=
                                        besetzter Block
68                                  {
69                                      System.out.println("Keine
                                            Freigabe. Zug fährt im
                                            Kreis ohne Möglichkeit ihn
                                            zu stoppen.");
70                                  }
71                                  else
72                                  {
73                                      System.out.println("Keine
                                            Freigabe: besetzt: "+
                                            xPosBesetzt+" / "+
                                            yPosBesetzt);
74                                  }
75                                  break;
76                              case '3':
77                                  int xPosletzteBlock = Integer.
                                        parseInt(freigabe.substring(
                                        freigabe.indexOf(',')+1,
                                        freigabe.indexOf(';')));
78                                  int yPosletzteBlock = Integer.
                                        parseInt(freigabe.substring(
                                        freigabe.indexOf(';')+1,
                                        freigabe.length()));
79                                  System.out.println("Keine Freigabe.
                                         Strecke nach ("+
                                        xPosletzteBlock+"|"+
                                        yPosletzteBlock+") unvollstä
                                        ndig.");
80                                  break;
81                              default:
```

```
82                                  System.out.println("Keine Freigabe.
                                        ");
83                              }
84                          }
85                          else  //Sackgasse oder Weiche falsch gestellt
86                          {
87                              if (nxtBlock.checkBlockEingang(nextBlock,
                                    nxtBlock.richtungskonverter(
                                    fahrtrichtung), false)) //steht vor
                                    Weiche die falsch gestellt ist
88                              {
89                                  if(nxtBlock.userFrage("Weiche falsch"))
90                                  {
91                                      Greenfoot.delay(20);
92                                      Weiche kommeWeiche = (Weiche)
                                            nextBlock;
93                                      kommeWeiche.stelleDieWeiche(
                                            nextBlock);
94                                      Greenfoot.delay(20);
95                                      fahreBlock(fahrtrichtung);  //darf
                                            fahren
96                                      Greenfoot.delay(10);
97                                      er=block.richtungskonverter(
                                            fahrtrichtung);
98                                  }
99                                  else
100                                 {
101                                     System.out.println("Zug bekommt
                                            keine Freigabe zur Abfahrt.");
102                                 }
103                             }
104                             else  //Sackgasse
105                             {
106                                 System.out.println("Zug steht in einer
                                        Sackgasse");
107                             }
108                         }
109                     }
110                     else  //Sackgasse
111                     {
112                         System.out.println("Zug steht in einer
                                Sackgasse");
113                     }
114                 }
115                 else  //vor rotem Signal
116                 {
117                     Actor rotessignal = getRotesSignal(fahrtrichtung);
118                     Signal rs = (Signal) rotessignal;
119                     String signalfreigabe = rs.getFreigabe(rotessignal,
                            fahrtrichtung, ziel);
120                     char signalfreigabecode = signalfreigabe.charAt(0);
121                     switch(signalfreigabecode)
122                     {
123                         case '0':  //freigabe
124                             if(block.userFrage("Signal"))
125                             {
126                                 Greenfoot.delay(20);
127                                 rs.signalWechseln(rotessignal);
```

```
128                                 Greenfoot.delay(20);
129                                 fahreBlock(fahrtrichtung);
130                                 Greenfoot.delay(10);
131                                 er=block.richtungskonverter(
                                        fahrtrichtung);
132                             }
133                             else
134                             {
135                                 er='A';
136                             }
137                             break;
138                         case '1':  //Weiche falsch gestellt
139                             er='A';
140                             System.out.println("Weiche falsch gestellt"
                                    );
141                             break;
142                         case '2':
143                             System.out.println("Strecke besetzt. Warte
                                    dass Strecke frei wird.");
144                             break;
145                         case '3':
146                             er='A';
147                             System.out.println("Fehlerhafte Strecke");
148                             break;
149                     }
150                 }
151             }
152             else
153             {
154                 if(block.checkBlockEingang(b, fahrtrichtung, false))
155                 {
156                    System.out.println("Weiche für diese Richtung falsch
                        gestellt!");
157                 }
158                 else
159                 {
160                     System.out.println("Zug kann von hier nicht in
                        Richtung "+gibfahrtrichtung+ " fahren.");
161                 }
162             }
163         }
164         else
165         {
166             if ((gibfahrtrichtung.equals("halt")||(gibfahrtrichtung.
                    equals("Halt"))))
167             {
168                 er='A';
169             }
170             else
171             {
172                 System.out.println("Ungültige Fahrtrichtung");
173             }
174         }
175     }
176     else  //er != Z, somit nicht erste act-durchlauf, er nun
            Eingangsrichtung in aktuellen Block
177     {
178     if(er!='A')
```

```
179     {
180         fahrtrichtung=block.getBlockRichtung(b, er);
181         Actor prevBlock = block.getnextBlock(block.getX(), block.getY()
                , er);
182         Block pBlock= (Block) prevBlock;
183         pBlock.setbesetzt(prevBlock);
184         List<Signal> listSignal;
185         listSignal = getObjectsAtOffset(prevBlock.getX()-getX(),
                prevBlock.getY()-getY(), Signal.class);  //bekommt Liste
                aller Signale an diesem Block
186         if (listSignal.isEmpty() == false)
187         {
188             for (int i = 0; i<listSignal.size(); i++)   //durchlaufe
                    Liste
189             {
190                 Actor prevsignal = listSignal.get(i);
191                 Signal prevMglSignal = (Signal) prevsignal;
192                 if(prevMglSignal.checkSignal(prevsignal, pBlock.
                        richtungskonverter(er)))  //falls vorheriges Signal
193                 {
194                     prevMglSignal.signalWechseln(prevsignal);
                                //Wieder auf rot stellen
195                 }
196             }
197         }
198         Actor aktuelleHaltestelle = getOneObjectAtOffset(0, 0,
                Haltestelle.class);  //überprüfe ob Zug an Haltestelle
                steht
199         Haltestelle aktHaltestelle = null;
200         if (aktuelleHaltestelle != null)
201         {
202             aktHaltestelle = (Haltestelle) aktuelleHaltestelle;
203         }
204         if (ziel != null)
205         {
206             if ((aktHaltestelle==null)||(ziel.equals(aktHaltestelle.
                    getname_Haltestelle()) != true)) //falls Ziel nicht
                    erreicht
207             {
208                 char aktAusgangsRichtung = block.getBlockRichtung(b, er
                        );    //bekomme Ausgangsrichtung des aktuellen
                        Blocks
209                 if ((aktHaltestelle == null)||(aktHaltestelle.
                        getFreigabe(b, aktAusgangsRichtung, null).equals("0
                        ")))  //falls keine Haltestelle oder Zug Freigabe
                        hat
210                 {
211                     if (checkRotesSignal(aktAusgangsRichtung) != true)
                            //Überprüfe ob Zug vor rotem Signal steht
212                     {
213                         Actor nextBlock = block.getnextBlock(block.getX
                                (), block.getY(), aktAusgangsRichtung);  //
                                finde den nächsten Block und speicher ihn
                                in nextBlock
214                         if(nextBlock != b)  //getnextBlock gibt
                                aktuellen Block zurück, wenn in Sackgasse
215                         {
216                             Block nxtBlock = (Block) nextBlock;
```

```
217                         if (nxtBlock.checkBlockEingang(nextBlock,
                                nxtBlock.richtungskonverter(
                                aktAusgangsRichtung), true)) //checkt
                                ob nächste Block Eingang in der
                                Richtung hat
218                         {
219                             Actor aktuellerPrellbock =
                                    getOneObjectAtOffset(block.getX()-
                                    getX(), block.getY()-getY(),
                                    Prellbock.class);
220                             if (aktuellerPrellbock != null)  //ü
                                    berprüfe ob Zug an einem Prellbock
                                    steht
221                             {
222                                 if(nxtBlock.userFrage("Prellbock"))
223                                 {
224                                     Prellbock aktPrellbock = (
                                            Prellbock)
                                            aktuellerPrellbock;
225                                     if(aktPrellbock.getFreigabe(
                                            aktuellerPrellbock,
                                            aktAusgangsRichtung, ziel).
                                            equals("0"))
226                                     {
227                                         fahreBlock(
                                                aktAusgangsRichtung);
228                                         Greenfoot.delay(10);
229                                         er=block.richtungskonverter
                                                (aktAusgangsRichtung);
230                                     }
231                                     else
232                                     {
233                                         System.out.println("Zug hat
                                                keine Freigabe den
                                                Prellbock zu verlassen"
                                                );
234                                     }
235                                 }  //Zug soll an Prellbock stehen
                                        bleiben
236                                 else
237                                 {
238                                     System.out.println("Zug an
                                            Prellbock stehen geblieben"
                                            );
239                                     resetZug();
240                                     er='A';
241                                 }
242                             }  //kein Prellbock
243                             else
244                             {
245                                 fahreBlock(aktAusgangsRichtung);
246                                 Greenfoot.delay(10);
247                                 er=block.richtungskonverter(
                                        aktAusgangsRichtung);
248                             }
249                         }
250                         else  //nächste Block keinen passenden
                                Eingang
```

```
251                             {
252                                 System.out.println("Zug steht in einer
                                        Sackgasse");
253                             }
254                         }
255                         else  //kein nächster Block
256                         {
257                             System.out.println("Zug steht in einer
                                    Sackgasse");
258                         }
259                     }
260                     else  //falls Zug vor rotem Signal steht
261                     {
262                         Actor rotessignal = getRotesSignal(
                                aktAusgangsRichtung);
263                         Signal rs = (Signal) rotessignal;
264                         String signalfreigabe = rs.getFreigabe(
                                rotessignal, fahrtrichtung, ziel);
265                         char signalfreigabecode = signalfreigabe.charAt
                                (0);
266                         switch(signalfreigabecode)
267                         {
268                         case '0':  //freigabe
269                             if(pBlock.userFrage("Signal"))
270                             {
271                                 Greenfoot.delay(20);
272                                 rs.signalWechseln(rotessignal);
273                                 Greenfoot.delay(20);
274                                 fahreBlock(fahrtrichtung);
275                                 Greenfoot.delay(10);
276                                 er=block.richtungskonverter(
                                        fahrtrichtung);
277                             }
278                             else
279                             {
280                                 er='A';
281                             }
282                             break;
283                         case '1':  //Weiche falsch gestellt
284                             er='A';
285                             System.out.println("Weiche falsch gestellt"
                                    );
286                             break;
287                         case '2':
288                             System.out.println("Strecke besetzt. Warte
                                    dass Strecke frei wird.");
289                             break;
290                         case '3':
291                             er='A';
292                             System.out.println("Fehlerhafte Strecke");
293                             break;
294                         }
295                     }
296                 }
297                 else   //Haltestelle ohne Freigabe
298                 {
299                     System.out.println("Zug hat keine Freigabe um die
                            Haltestelle zu verlassen");
```

```
300                 }
301             }
302             else  //Zielhaltestelle erreicht
303             {
304                 resetZug();
305                 if(pBlock.userFrage("Ziel"))
306                 {
307                     er='A';
308                 }
309             }
310         }
311         else  //keine Zielhaltestelle
312         {
313             if (aktuelleHaltestelle != null)
314             {
315                 if(block.userFrage("Haltestelle"))
316                 {
317                     System.out.println("Zug hat gehalten");
318                     resetZug();
319                     er='A';
320                 }
321                 else   //soll weiterfahren
322                 {
323                     char aktAusgangsRichtung = block.getBlockRichtung(b
                        , er);     //bekomme Ausgangsrichtung des
                        aktuellen Blocks
324                     if(aktHaltestelle.getFreigabe(b,
                        aktAusgangsRichtung, null).equals("0"))
325                     {
326                         Actor nextBlock = block.getnextBlock(block.getX
                            (), block.getY(), aktAusgangsRichtung);  //
                            finde den nächsten Block und speicher ihn
                            in nextBlock
327                         if(nextBlock != b)  //getnextBlock gibt
                            aktuellen Block zurück, wenn in Sackgasse
328                         {
329                             Block nxtBlock = (Block) nextBlock;
330                             if (nxtBlock.checkBlockEingang(nextBlock,
                                nxtBlock.richtungskonverter(
                                aktAusgangsRichtung), true)) //checkt
                                ob nächste Block Eingang in der
                                Richtung hat
331                             {
332                                 fahreBlock(aktAusgangsRichtung);
333                                 Greenfoot.delay(10);
334                                 er=block.richtungskonverter(
                                    aktAusgangsRichtung);
335                             }
336                             else
337                             {
338                                 System.out.println("Zug steht in einer
                                    Sackgasse");
339                             }
340                         }
341                         else
342                         {
343                             System.out.println("Zug steht in einer
                                Sackgasse");
```

```
344                         }
345                     }
346                     else
347                     {
348                         System.out.println("Zug hat keine Freigabe um
                               die Haltestelle zu verlassen");
349                     }
350                 }
351             }
352             else //keine Haltestelle
353             {
354             char aktAusgangsRichtung = block.getBlockRichtung(b, er);
                    //bekomme Ausgangsrichtung des aktuellen Blocks
355             if (checkRotesSignal(aktAusgangsRichtung) != true)  //Ü
                berprüfe ob Zug vor rotem Signal steht
356                 {
357                     Actor nextBlock = block.getnextBlock(block.getX(),
                        block.getY(), aktAusgangsRichtung);  //finde
                        den nächsten Block und speicher ihn in
                        nextBlock
358                     if(nextBlock != b)  //getnextBlock gibt aktuellen
                        Block zurück, wenn in Sackgasse
359                     {
360                         Block nxtBlock = (Block) nextBlock;
361                         if (nxtBlock.checkBlockEingang(nextBlock,
                            nxtBlock.richtungskonverter(
                            aktAusgangsRichtung), true)) //checkt ob nä
                            chste Block Eingang in der Richtung hat
362                         {
363                             Actor aktuellerPrellbock =
                                getOneObjectAtOffset(block.getX()-
                                getX(), block.getY()-getY(),
                                Prellbock.class);
364                             if (aktuellerPrellbock != null)  //ü
                                berprüfe ob Zug an einem Prellbock
                                steht
365                             {
366                                 if(nxtBlock.userFrage("Prellbock"))
367                                 {
368                                     Prellbock aktPrellbock = (
                                        Prellbock)
                                        aktuellerPrellbock;
369                                     if(aktPrellbock.getFreigabe(
                                        aktuellerPrellbock,
                                        aktAusgangsRichtung, ziel).
                                        equals("0"))
370                                     {
371                                         fahreBlock(
                                            aktAusgangsRichtung);
372                                         Greenfoot.delay(10);
373                                         er=block.richtungskonverter
                                            (aktAusgangsRichtung);
374                                     }
375                                     else
376                                     {
377                                         System.out.println("Zug hat
                                            keine Freigabe den
                                            Prellbock zu verlassen"
```

```
                                                    );
378                                             }
379                                         }
380                                         else
381                                         {
382                                             System.out.println("Zug an
                                                    Prellbock stehen geblieben"
                                                    );
383                                             resetZug();
384                                             er='A';
385                                         }
386                                     }
387                                     else
388                                     {
389                                         fahreBlock(aktAusgangsRichtung);
390                                         Greenfoot.delay(10);
391                                         er=block.richtungskonverter(
                                                aktAusgangsRichtung);
392                                     }
393                                 }
394                                 else
395                                 {
396                                     System.out.println("Zug steht in einer
                                            Sackgasse");
397                                 }
398                             }
399                             else
400                             {
401                                 System.out.println("Zug steht in einer
                                        Sackgasse");
402                             }
403                         }
404                         else    //falls Zug vor rotem Signal steht
405                         {
406                             Actor rotessignal = getRotesSignal(
                                    aktAusgangsRichtung);
407                             Signal rs = (Signal) rotessignal;
408                             String signalfreigabe = rs.getFreigabe(rotessignal,
                                    fahrtrichtung, ziel);
409                             char signalfreigabecode = signalfreigabe.charAt(0);
410                             switch(signalfreigabecode)
411                             {
412                                 case '0':  //freigabe
413                                     if(block.userFrage("Signal"))
414                                     {
415                                         Greenfoot.delay(20);
416                                         rs.signalWechseln(rotessignal);
417                                         Greenfoot.delay(20);
418                                         fahreBlock(fahrtrichtung);
419                                         Greenfoot.delay(10);
420                                         er=block.richtungskonverter(
                                                fahrtrichtung);
421                                     }
422                                     else
423                                     {
424                                         er='A';
425                                     }
426                                     break;
```

```
427                         case '1':  //Weiche falsch gestellt
428                             er='A';
429                             System.out.println("Weiche falsch gestellt"
                                    );
430                             break;
431                         case '2':
432                             System.out.println("Strecke besetzt. Warte
                                    dass Strecke frei wird.");
433                             break;
434                         case '3':
435                             er='A';
436                             System.out.println("Fehlerhafte Strecke");
437                             break;
438                     }
439                 }
440             }
441         }
442     }
443     }
444     }
445     else
446     {
447         System.out.println("Zug steht auf keiner Schiene.");
448     }
449 }
```

A.2.7 Lösung mit enums

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, and Greenfoot)
2
3  /**
4   * Schiene per enums
5   *
6   * @author Niklas Nogossek
7   */
8  public class Schiene extends Actor
9  {
10     enum Schienentyp{
11         GS("gerade_senkrecht_weiß.gif"),
12         GW("gerade_waage_weiß.gif"),
13         KUL("Kurve_unten_links_weiß.gif"),
14         KOL("Kurve_oben_links_weiß.gif");
15
16         String bild;
17
18         Schienentyp(String i){
19             bild = i;
20         }
21     }
22     String schienenart;
23     public Schiene(String typ)
24     {
25     switch(typ){
26         case "GS":
27             schienenart = "gerade_senkrecht";
28             this.setImage(Schienentyp.GS.bild);
29             break;
```

```
30          case "GW":
31              schienenart = "gerade_senkrecht";
32              this.setImage(Schienentyp.GW.bild);
33              break;
34          case "KUL":
35              schienenart = "Kurve_unten_links_weiß";
36              this.setImage(Schienentyp.KUL.bild);
37              break;
38          case "KOL":
39              schienenart = "Kurve_oben_links_weiß";
40              this.setImage(Schienentyp.KOL.bild);
41              break;
42          //usw. für weitere Schienentypen
43      }
44  }
45  }
```

A.3 Fragebogen: Kontext Eisenbahn vs. Planetenerkundung

A.3.1 Antwort 1

Fragebogen: Einführung in die Objektorientierung anhand einer Stellpultsimulation im Kontext Eisenbahn

1. Wie sehr kann der Kontext Eisenbahn und insbesondere das Konzept des digitalen Gleisbildstellpults die Schülerinnen und Schüler für die Einführung in die Objektorientierung motivieren?

	sehr stark	stark	neutral	wenig	sehr wenig
Kontext Eisenbahn	○	◉	○	○	○
Konzept Gleisbildstellpult	○	◉	○	○	○

Bemerkung:

Es kommt sehr auf die Ausgestaltung des Kontextes an. Die Bewertung orientiert sich an der durch "kann" in der Fragestellung eröffneten Möglichkeit. Für eine tatsächliche Einschätzung der Motivation müsste das Szenario deutlich konkreter ausgearbeitet sein, evtl. auch mit (Planungen als) Beispiele(n) für Unterrichtsstunden.

2. Für welche Bereiche der Einführung in die Objektorientierung halten sie das digitale Gleisbildstellpult für geeignet?

	geeignet	nicht geeignet
Objekte und Klassen	◉	○
Methoden	◉	○
Sequenzdiagramm	◉	○
Klassendiagramm	◉	○
Algorithmen	◉	○

Bemerkung:

Die Idee eines durchgängigen Beispiels ist zu begrüßen. Der gewählte Kontext kann sicherlich so ausgestaltet werden, dass alle genannten Bereiche mindestens grundlegend abgebildet werden. Eine Vertiefung kann ggf. auch außerhalb des Kontextes gefunden werden.

3. Welchen der beiden Kontexte würde Sie in den folgenden Bereichen favorisieren?

	Eisenbahn	Planetenerkundung	beide gleichermaßen
Objekte und Klassen	○	◉	○
Methoden	○	◉	○
Sequenzdiagramm	○	◉	○
Klassendiagramm	○	◉	○
Algorithmen	○	◉	○

Bemerkung:

Die Bewertung bezieht sich auf die vorliegende Rohfassung des Eisenbahn-Szenarios. Nach einer noch Analyse und Verbesserung dieses Szenarios können beide Szenrien sicherlich in allen Bereichen gleich gut geeignet sein. Ideen für eine Überarbeitung gehen gesondert per E-Mail zu.

4. Verglichen mit dem digitalen Gleisbildstellpult ist das Szenario Planetenerkundung begrenzter in den Möglichkeiten, in denen die Schülerinnen und Schüler kreativ werden können, zum Beispiel in der Planungsphase von Klassen oder bei der Überlegung, wie die Methode fahre(richtung:char) umgesetzt werden soll.
Halten Sie diese zusätzlichen Möglichkeiten zur kreativen Entfaltung für einen Vorteil im Vergleich zu dem begrenzten Kontext Planetenerkundung, bei dem der Fokus auf den Grundlagen bleibt?

	großer Vorteil	Vorteil	egal	Nachteil	großer Nachteil
kreative Entfaltung	○	○	◉	○	○

Bemerkung:

Die Frage zeigt schon auf, dass die Planetenerkundung die Grundlagen in den Vordergrund stellt. Dieses Szenario kann durch ein weiteres Szenario oder die Arbeit mit BlueJ ergänzt werden. Im Lehrwerk selbst wird beispielsweise auf Greeps verwiesen.
Wenn das Eisenbahn-Szenario in sich mehr Möglichkeiten bietet, ist das begrüßenswert, bietet aber auch Einschränkungen. Beispielsweise stellt die Erfordernis, einen Parameter für "fahre" angeben zu müssen, eine Hürde beim Einstieg ins Szenario dar, wenn noch keine Methoden geschrieben, sondern über Rechtsklick mit der Maus ausgeführt werden.

5. Für wie sinnvoll halten Sie es Sequenz- und Klassendiagramm vor algortihmischen Konstrukten (wie z.B. Zählschleifen) einzuführen? (siehe auch Reihenfolge der Themen in der Handreichung)

	1	2	3	4	5	
sehr sinnvoll	○	○	◉	○	○	gar nicht sinnvoll

Bemerkung

Für Klassendiagramme ist dieses sinnvoll, da diese nah an der Darstellung von Klassen / Methodenübersichten in Greenfoot sind. Zudem lässt sich so schnell ein Überblick über das Szenario darstellen. Auch helfen Klassendiagramme bei der Einführung des Fachvokabulars der Objektorientierung. Für Klassendiagramme daher eine 2.

Für Sequenzdiagramme ist dieses gar nicht sinnvoll, da sie die SuS erfahrungsgemäß schnell überfordern. Hier eignen sich zu Beginn Pseudo-Code oder Programmablaufpläne deutlich besser. Zudem muss beachtet werden, dass laut KLP auch Sequenzdiagramme verwendet werden sollen.

6. Sehen Sie einen Vorteil darin, einen Kontext zu haben, der für einen längeren Zeitraum tragfähig ist?

	großer Vorteil	Vorteil	egal	Nachteil	großer Nachteil
für eine Unterrichtsreihe	◉	○	○	○	○
für ein Quartal	○	◉	○	○	○
für ein Jahr	○	◉	○	○	○
Jahrgangsstufenübergreifend	○	◉	○	○	○

Bemerkung:

Die Betonung liegt auf "tragfähig". Damit ist er auch motivierend und bietet stets neue Blickwinkel auf aktuellen Unterrichtsstoff.

A.3.2 Antwort 2

1. Wie sehr kann der Kontext Eisenbahn und insbesondere das Konzept des digitalen Gleisbildstellpults die Schülerinnen und Schüler für die Einführung in die Objektorientierung motivieren?

	sehr stark	stark	neutral	wenig	sehr wenig
Kontext Eisenbahn	○	◉	○	○	○
Konzept Gleisbildstellpult	○	○	◉	○	○

Bemerkung:

"Eisenbahn" ist ein Bereich, den SuS kennen. "Gleisbildstellpult" ist eher ein Kontext für Modelleisenbahnexperten.

2. Für welche Bereiche der Einführung in die Objektorientierung halten sie das digitale Gleisbildstellpult für geeignet?

	geeignet	nicht geeignet
Objekte und Klassen	◉	○
Methoden	◉	○
Sequenzdiagramm	○	◉
Klassendiagramm	○	○
Algorithmen	◉	○

Bemerkung:

Wir schränken ein, das Gleisbildstellpult käme in der EF, im ersten Lernjahr zum Einsatz. Sequenzdiagramme spielen da keine Rolle (im gesamten KLP nicht). Für die erste Einführung von Objekten und Klassen ist das Konzept schon zu weit, es ist schwer echte Objekte zu betrachten und dann Klassen abzuleiten. Für den zweiten Schritt ist es geeignet, sofern ich neue Unterklassen (Gleisteile oder Züge) einbinde. Vielleicht wäre es gut, mit weniger Gleisteilen in der Vorlage zu beginnen. Methoden wird man am Anfang aufzurufen lernen mit Adressierung der Objekte. Hinsichtlich der Klassendiagramm erscheint das ganze sehr komplex und die Schüler werden insbesondere lesen müssen. Ich schätze Editoren, in denen die SuS Klassendiagramme erstellen und daraus der Code entsteht, um den Sinn dieser wichtigen Planungsphase zu verstehen und nicht nur passiv zu lesen. Algorithmen sind die Stärke, man wird Kontrollstrukturen, etc. einführen und üben können. (Dabei ist ein allgemeines Greenfoot-Problem, dass über die act-Methode eine Wiederholung ohne sichtbare Schleife generiert wird.)

3. Welchen der beiden Kontexte würde Sie in den folgenden Bereichen favorisieren?

	Eisenbahn	Planetenerkundung	beide gleichermaßen
Objekte und Klassen	◉	○	○
Methoden	○	○	◉
Sequenzdiagramm	○	○	◉
Klassendiagramm	○	○	◉
Algorithmen	○	◉	○

Bemerkung:

Neue Klassen sind im Planeten schwerer einzupflegen. Die Stärke des Planeten liegt bei den Algorithmen. Lästig ist die immer nur kurze Ausgabe in das sinnfreie Infofeld statt in die Konsole, die die Klasse Rover unübersichtlich macht. Man kann aber nach der Einführung in Algorithmen exemplarisch den Pledge-Algorithmus (oder Teilse davon) erarbeiten und implementieren. Vielleicht gibt es für das Eisenbahn-Szenario eine ähnliche Möglichkeit? Dykstra o.ä. wird vermutlich schwieriger.

4. Verglichen mit dem digitalen Gleisbildstellpult ist das Szenario Planetenerkundung begrenzter in den Möglichkeiten, in denen die Schülerinnen und Schüler kreativ werden können, zum Beispiel in der Planungsphase von Klassen oder bei der Überlegung, wie die Methode fahre(richtung:char) umgesetzt werden soll.
Halten Sie diese zusätzlichen Möglichkeiten zur kreativen Entfaltung für einen Vorteil im Vergleich zu dem begrenzten Kontext Planetenerkundung, bei dem der Fokus auf den Grundlagen bleibt?

	großer Vorteil	Vorteil	egal	Nachteil	großer Nachteil
kreative Entfaltung	○	○	◉	○	○

Bemerkung:

Die Abwägung ist schwierig, weil das auch abhängig vom Kurs ist. Starke SuS sind mit den Planetenaufgaben schnell durch und brauchen zusätzliche Möglichkeiten, für weniger starke SuS ist die reduzierte Welt der Marsrovers vorteilhaft, weil sie sich fokussieren können. Letztlich hängt es dann davon ab, welche in welchem Maß reduzierte oder erweiterbare Vorlagen der Eisenbahn zur Verfügung stehen bzw. der Lehrer erstellt hat.

5. Für wie sinnvoll halten Sie es Sequenz- und Klassendiagramm vor algortihmischen Konstrukten (wie z.B. Zählschleifen) einzuführen? (siehe auch Reihenfolge der Themen in der Handreichung)

	1	2	3	4	5	
sehr sinnvoll	○	○	○	◉	○	gar nicht sinnvoll

Bemerkung

s.o. Sequenzdiagramme finden sich nicht im KLP. Klassendiagramme wird man im Kontext des Szenarios lesen können müssen, wobei der Fokus auf der einzelnen Klasse insbesondere mit den Methoden gelegt ist. Klassen ohne ihre Beziehungen wird man auf jeden Fall vorher machen. Für die erste Einführung in Ist-Beziehungen ist die Darstellung in Greenfoot gut geeignet, schwerer wird das mit Assoziationen. Wenn man dann aktiv eine Unterklasse erstellt, will man Verhalten ändern, dafür braucht es irgendwelche Kontrollstrukturen.

6. Sehen Sie einen Vorteil darin, einen Kontext zu haben, der für einen längeren Zeitraum tragfähig ist?

	großer Vorteil	Vorteil	egal	Nachteil	großer Nachteil
für eine Unterrichtsreihe	◉	○	○	○	○
für ein Quartal	○	◉	○	○	○
für ein Jahr	○	○	◉	○	○
Jahrgangsstufenübergreifend	○	○	○	◉	○

Bemerkung:

Greenfoot will ich nicht in der Q1 und Q2 nutzen, bzw. nur in ausgewählten Kontexten. Der eher spielerische/grafiklastiger Einstieg in der EF ist OK, irgendwann braucht man aber eine Benutzeroberfläche. Aber insgesamt wäre mir auch ein Jahr Eisenbahn oder Marsrover zu lang. Für einen eigenen Kontext ohne große Vorlagen sollte am Ende der EF Zeit sein.

(Anmerkung: In der abgegebenen Arbeit standen zu den Fragen 4-6 fälschlicherweise die Antworten der 3. befragten Person.)

A.3.3 Antwort 3

1. Wie sehr kann der Kontext Eisenbahn und insbesondere das Konzept des digitalen Gleisbildstellpults die Schülerinnen und Schüler für die Einführung in die Objektorientierung motivieren?

	sehr stark	stark	neutral	wenig	sehr wenig
Kontext Eisenbahn	○	○	◉	○	○
Konzept Gleisbildstellpult	○	○	○	◉	○

Bemerkung:

Nach Rücksprache mit meinem Q1 Kurs, die im letzten Jahr das Mars-Rover-Projekt durchgeführt haben, habe ich die Motiationspunkte jeweils einen Punkt reduziert.
Begründung: Der Kontext Eisenbahn ist generell ein guter Kontext, weil es sich auf die Lebenswelt der Kinder bezieht (Eisenbahnfahrten / Spielzeugbahnen). Allerdings sieht für die Schüler*innen der Rover besser aus und es ist zunächst sehr "erschlagend", wenn man die Vielzahl an Bausteinen betrachtet.
Beim Gleisbildstellpult ist die Motivation weniger hoch, da trotz der Vielzahl an Bauteilen die Bewegung auf Schienen als zu eingeschränkt erachtet wird. Außerdem wurde die Grafik vom Rover bevorzugt.

2. Für welche Bereiche der Einführung in die Objektorientierung halten sie das digitale Gleisbildstellpult für geeignet?

	geeignet	nicht geeignet
Objekte und Klassen	◉	○
Methoden	◉	○
Sequenzdiagramm	○	◉
Klassendiagramm	◉	○
Algorithmen	◉	○

Bemerkung:

Generell ist das Szenario für viele Bereiche der Objektorientierung geeignet. Allerdings halte ich das Sequenzdiagramm aufgrund der Vielzahl der beteiligten Objekte weniger geeignet. In den Bereichen Klassen und Objekte / Klassendiagramm ist die Komplexität aufgrund der Vererbung (Weiche_Nord_Ost IST Weiche IST Block IST Actor) sehr hoch und eventuell für die Einführung in die Objektorientierung eher bedingt geeignet. Den Zug fahren zu lassen (Methoden) und Kontrollstukturen (Algorithmen) anzuwenden ist sehr gut geeignet.

3. Welchen der beiden Kontexte würde Sie in den folgenden Bereichen favorisieren?

	Eisenbahn	Planetenerkundung	beide gleichermaßen
Objekte und Klassen	○	◉	○
Methoden	○	○	◉
Sequenzdiagramm	○	◉	○
Klassendiagramm	○	◉	○
Algorithmen	○	○	◉

Bemerkung:

Ingesamt wurde die Planetenerkundung aufgrund der wenigeren Klassen und der besseren Grafik bevorzugt oder beide als gleich gut befunden.

4. Verglichen mit dem digitalen Gleisbildstellpult ist das Szenario Planetenerkundung begrenzter in den Möglichkeiten, in denen die Schülerinnen und Schüler kreativ werden können, zum Beispiel in der Planungsphase von Klassen oder bei der Überlegung, wie die Methode fahre(richtung:char) umgesetzt werden soll.
Halten Sie diese zusätzlichen Möglichkeiten zur kreativen Entfaltung für einen Vorteil im Vergleich zu dem begrenzten Kontext Planetenerkundung, bei dem der Fokus auf den Grundlagen bleibt?

	großer Vorteil	Vorteil	egal	Nachteil	großer Nachteil
kreative Entfaltung	○	◉	○	○	○

Bemerkung:

Grundsätzlich wurden die Vielzahl an Möglichkeiten positiv bewertet. Allerdings ist auch die Bahn in ihren Bewegungen eingeschränkt, weil sie eben nur auf Schienen fahren kann.

5. Für wie sinnvoll halten Sie es Sequenz- und Klassendiagramm vor algortihmischen Konstrukten (wie z.B. Zählschleifen) einzuführen? (siehe auch Reihenfolge der Themen in der Handreichung)

	1	2	3	4	5	
sehr sinnvoll	○	◉	○	○	○	gar nicht sinnvoll

Bemerkung

Der Ansatz, Objekte zunächst kommunizieren zu lassen und Attribute und Methoden zu identifizieren halte ich für sehr sinnvoll - auch weil sich zum Beispiel das Verfahren von Abbott als sinnvoll erwiesen hat.

6. Sehen Sie einen Vorteil darin, einen Kontext zu haben, der für einen längeren Zeitraum tragfähig ist?

	großer Vorteil	Vorteil	egal	Nachteil	großer Nachteil
für eine Unterrichtsreihe	◉	○	○	○	○
für ein Quartal	○	◉	○	○	○
für ein Jahr	○	○	○	◉	○
Jahrgangsstufenübergreifend	○	○	○	○	◉

Bemerkung:

Sich längere Zeit mit einem Kontext zu beschäftigen wird grundsätzlich sehr positiv bewertet. Allerdings sollte dies in einem begrenzten Zeitrahmen geschehen und spätestens nach einem Quartal wird es zunehmend negativ bewertet - auch weil es sonst zu einseitig wird.

Eigenständigkeitserklärung

Hiermit versichere ich, dass die vorliegende Arbeit über *Einführung in die Objektorientierung anhand einer Stellpultsimulation im Kontext Eisenbahn/ Introduction to object-orientation based on a control panel simulation in context of railways* selbstständig von mir und ohne fremde Hilfe verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind. Mir ist bekannt, dass es sich bei einem Plagiat um eine Täuschung handelt, die gemäß der Prüfungsordnung sanktioniert werden kann.

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in einer Datenbank einverstanden.

Ich versichere, dass ich die vorliegende Arbeit oder Teile daraus nicht anderweitig als Prüfungsarbeit eingereicht habe.

_____

(Datum, Unterschrift)