Westfälischen Wilhelms-Universität Münster

BACHELOR ARBEIT:

# ARDUINO MIT JMRI UND GREENFOOT FÜR DEN INFORMATIKUNTERRICHT

USING ARDUINO WITH JMRI AND GREENFOOT FOR CSE


vorgelegt von: Lars Bockstette

[redacted]

Studiengang: 2-Fach Bachelor Informatik/Mathematik

Matrikelnummer: [redacted]

WS 2021/22

Betreuer Professor Dr. Marco Thomas


Datum 27. Dezember 2021

# EIDESSTATTLICHE ERKLÄRUNG

**Plagiatserklärung der / des Studierenden**

Hiermit versichere ich, dass die vorliegende Arbeit über Arduino mit JMRI und Greenfoot für den Informatikunterricht / Using Arduino with JMRI and Greenfoot for CSE selbstständig 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.

27.12.2021 [signature]

(Datum, Unterschrift)

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 eine Datenbank einverstanden.

27.12.2021 [signature]

(Datum, Unterschrift)

# VORWORT

Diese Bachelorarbeit behandelt die Fragestellung, wie der *Arduino Uno* zur Steuerung von Modelleisenbahnen im Informatikunterricht eingesetzt werden kann. Verfasst wurde diese Arbeit als Abschlussarbeit meines 2-Fach-Bachelor Informatik und Mathematik an der Westfälischen Wilhelms-Universität Münster. Ziel dieser Arbeit ist es neue Methoden für den kontextorientierten Informatikunterricht zu schaffen.
Zusammen mit meinem Betreuer Professor Marco Thomas, wurde die Fragestellung für diese Bachelorarbeit, im Hinblick auf das Projekt ”LoCo-Eisenbahn als informatischer Kontext” des Arbeitsbereichs Didaktik der Informatik, entwickelt.
Ich wünsche Ihnen viel Freude beim Lesen meiner Bachelorarbeit
Lars Bockstette
Rheine, 27. Dezember 2021

INHALTSVERZEICHNIS

WESTFÄLISCHE WILHELMS-UNIVERSITÄT MÜNSTER

# Bachelorarbeit

Lars Bockstette

27. Dezember 2021

## 1. EINLEITUNG

Die Naturwissenschaften sind von fachwissenschaftlichen Inhalten dominierte Fächer. Schülerinnen und Schülern fällt es leichter solch fachwissenschaftliche Inhalte zu verstehen und zu verarbeiten, wenn diese in Kontexten mit Bezug zur Lebenswelt dargestellt werden.[1] Kontextorientierter Unterricht in den naturwissenschaftlichen Fächern Physik, Biologie und Chemie wird durch Projekte wie ”Physik im Kontext” (piko), ”Chemie im Kontext ”(ChiK) und ”Biologie im Kontext” (bik) vom BMBF[2] gefördert und wurde in die Kerncurricula aufgenommen und ist somit verpflichtend.[3] Für den Informatikunterricht ist die Kontextorientierung in den Kerncurricula der Länder nicht erwähnt. Das Projekt Informatik im Kontext, welches auf der INFOS 2009 vorgestellt wurde, hat einige Unterrichtsentwürfe und Konzepte auf ihrer Website[4] veröffentlicht und 2011 in der LOG IN die Entwicklungen in dem Projekt zusammengefasst.[5]

[1] Nawrath, „Kontextorientierung Rekonstruktion einer fachdidaktischen Konzeption für den Physikunterricht“.
[2] Bundesministerium für Bildung und Forschung, *Bundesministerium für Bildung und Forschung.*
[3] Ministerium für Schule und Bildung des Landes Nordrhein-Westfalen, „Kernlehrplan für die Sekundarstufe I Gymnasium in Nordrhein-Westfalen Biologie“.
[4] Koubek *u. a., Informatik im Kontext(IniK).*
[5] Diethelm, Koubek & Witten, „IniK - Informatik im Kontext - Entwicklungen, Merkmale und Perspektiven“.

Das Projekt LoCo[6] greift diese Konzepte auf und in dieser Arbeit soll untersucht werden, welche Möglichkeiten der Arduino Uno zur Steuerung einer Modellbahn bietet und wie der Arduino Uno in Kombination mit Greenfoot und JMRI im Informatikunterricht zur Umsetzung von Standards der informatischen Bildung eingesetzt werden kann.

## 1.1. Kontextorientierter Unterricht

Die Ausrichtung an den akademischen Fachdisziplinen innerhalb der naturwissenschaftlichen Fächer in den Schulen hat dafür gesorgt, dass die Fachprinzipien und -logiken stärker im Fokus sind als die Anwendungen eben dieser Prinzipien und Logiken im Alltag. Besonders in den MINT-Fächern, Mathematik, Informatik, Naturwissenschaften und Technik, sorgt diese Orientierung an den Fachwissenschaften dafür, dass die Interessen und Erwartungen von den Schülerinnen und Schülern oft von den fachlichen Lernzielen abweichen. Das Interesse vieler Schülerinnen und Schüler liegt in der Oberstufe beim Diskutieren und Bewerten, während Rechnen, Symbolisieren und Formalisieren weitgehend abgelehnt wird.[7]
Die naturwissenschaftlichen Fächer haben aus diesen Gründen kontextorientierte Ansätze, die sich auch bereits in den Kernlernplänen der einzelnen Bundesländer widerspiegeln. Die einzelnen Naturwissenschaften haben ein unterschiedliches Verständnis von Kontextbezügen. Biologie im Kontext orientiert sich an fünf Kriterien, anhand derer Aufgaben entwickelt werden. Nachfolgende Kriterien sind ausschlaggebend:

1. Der Kontext der Aufgabe sollte der Lebenswelt der Schülerinnen und Schüler entstammen.
2. Welche fachlichen Kompetenzen/Basiskonzepte werden benötigt?
3. Welche Kompetenzen nach den Bildungsstandards sind erforderlich?
4. Die Aufgabe sollte interessant oder motivierend sein (Affektiven Dimension).
5. Die Unterrichtsphase für die Aufgabe sollte klar festgelegt sein.[8]

Hingegen haben Mikelskis-Seifert und Duit den Begriff des Kontextes für Physik anders definiert. Zum einen in der Form, dass mit dessen Hilfe ein fachlicher Inhalt vermittelt wird, um sinnstiftend zu sein, zum anderen als den Kontext, in dem gelernt wird.[9] Es

[6] Thomas, *Eisenbahn im kontextorientierten Informatikunterricht*.
[7] Koubek *u. a.,* „Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht“, S. 268–269.
[8] Bayrhuber *u. a.,* „Biologie im Kontext“.
[9] Koubek *u. a.,* „Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht“, S. 268–269.

sollen das naturwissenschaftliche Denken gefördert und moderne Technologien in den Unterricht einbezogen werden. Lehrkörper sollen eine gewisse Methodenvielfalt anbieten um den Unterricht interessant zu gestalten.[10]

In der Chemie wird der Kontext als roter Faden der Unterrichtseinheit gesehen und der Unterricht wird anhand des Kontextes in vier Phasen aufgeteilt. In der Begegnungsphase wird der Kontext den Schülerinnen und Schülern vorgestellt und sie können sich mit diesem vertraut machen. In der zweiten Phase werden zu dem Kontext Fragen formuliert. Das Projekt ChiK spricht hier von der Neugier- und Planungsphase. Die Erarbeitungsphase dient zur Beantwortung der vorher formulierten Fragen zum Kontext. In der Vertiefungs- und Vernetzungsphase werden die fachlichen Basiskonzepte in den Mittelpunkt gestellt und mit anderen Kontexten in Verbindung gebracht.[11]

Auch wenn die verschiedenen Projekte Kontext unterschiedlich definieren, so können jedoch in allen drei Fächern gesteigertes Interesse durch die Kontextualisierung nachgewiesen werden.[12,13,14] Das Projekt Informatik im Kontext greift diese Konzepte auf und probiert eine Grundlage für die Kontextualisierung des Informatikunterrichts zu schaffen. Koubek formulierte 2009 drei Prinzipien: die Orientierung an Kontexten, die Orientierung an den Standards für die Informatik in der Schule und die Methodenvielfalt.[15] So soll ein Kontext das Interesse der Schülerinnen und Schüler gewinnen. Es soll sich an ihm ein möglichst breites Spektrum der Standards für die Informatik unterrichten lassen und die Möglichkeit bieten von den Schülerinnen und Schülern selbst erkundet zu werden. Es sollen nicht, wie oft in der Schule aufgrund von ”Zeitnot und Bequemlichkeit”, alle Inhalte im Frontalunterricht übermittelt werden.[16]

## 1.2. Eisenbahnen als Kontext

Jede Information die durch Computer verarbeitet wird, kann als Kontext gesehen werden.[17] Dieses beschränkt sich nicht nur auf Informationen außerhalb des Systems, sondern

[10] Mikelskis-Seifert & Duit, „Physik im Kontext“, S. 266–268.
[11] Di Fuccia, Schellenbach-Zell & Ralle, „Chemie im Kontext“.
[12] Smith & Matthews, „Science, technology and society in transition year: A pilot study“.
[13] Parchmann *u. a.,* „“Chemie im Kontext”: A symbiotic implementation of a context-based teaching and learning approach“.
[14] Höttecke, *Naturwissenschaftlicher Unterricht im internationalen Vergleich.*
[15] Koubek *u. a.,* „Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht“, S. 271.
[16] Koubek *u. a.,* „Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht“, S. 275.
[17] Hirschfeld, Costanza & Nierstrasz, „Context-oriented programming“.

meint auch Informationen die innerhalb eines Computersystems entstehen und verarbeitet werden.[18] Im Rahmen des IniK Projekts hat Koubek fünf Kriterien für die Auswahl von geeigneten Kontexten aufgestellt. Die fünf Kriterien sind *Mehrdimensionalität*, *Breite*, *Tiefe*, *Lebenswelt* und *Stabilität*.[19] Anhand dieser Kriterien wird der Kontext Eisenbahn auf die Tauglichkeit für den Informatikunterricht untersucht.

Das erste Kriterium die *Mehrdimensionalität* meint, dass der gewählte Kontext unter verschiedenen Aspekten Fragestellungen bieten muss. So bietet der Kontext Eisenbahn Fragestellungen aus Bereichen wie Ökologie: ”Reisen mit Auto oder Eisenbahn, was hinterlässt den größeren ökologischen Fußabdruck?”, Ökonomie: ”ÖPNV, günstigere Alternative zum Auto?” aber eben auch informatische Fragestellungen zur Steuerung, Überwachung oder Ticketverkauf sind möglich.

Der Kontext soll *Breite*, also gesellschaftliche Relevanz haben und nicht ausschließlich auf informatische Inhalte bezogen sein. Eisenbahnen haben einen Anteil im gesamten Verkehrswesen, also Personenverkehr und Güterverkehr, von 19%.[20] Sie sind damit auf dem zweiten Platz in der deutschen Gesamtverkehrsleistung und haben so höchste Relevanz in der Gesellschaft.

Neben der Breite sollen Kontexte auch *Tiefe* haben. Gemeint ist damit die informatische Relevanz des Kontextes, welche möglichst viele Kompetenzen der Bildungsstandards abdecken sollte. Eisenbahnen als Kontext können beispielsweise Datenbanken[21] oder Sortierverfahren[22] abdecken. Allerdings auch andere Kompetenzen wie zum Beispiel Automaten für Rückgeld beim Ticketverkauf, verschiedenste Programmieraufgaben, Arbeit mit Bibliotheken oder Binärdarstellung für die Steuerung der DCC Modelleisenbahnen.

*Lebenswelt* bedeutet, dass der Kontext einen Bezug zu der Lebenswelt der Schülerinnen und Schüler haben soll. Er sollte erlebbar sein und möglichst auch für Gespräche außerhalb des Unterrichts sorgen. Hier ist auch auf den Genderaspekt zu achten, so das sich Schülerinnen ebenso wie Schüler von dem Aspekt angesprochen fühlen. Erlebbar sind Eisenbahnen für alle Schülerinnen und Schüler in der Freizeit, bei Reisen oder Klassenfahrten und für viele jeden Tag auf dem Schulweg. Keine Einigkeit gibt es bei dem Thema

[18] Salvaneschi, Ghezzi & Pradella, „Context-oriented programming: A software engineering perspective“.

[19] Koubek, *Kriterien für die Auswahl von Kontexten.*

[20] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, *Marktuntersuchung Eisenbahnen 2021.*

[21] Mahnke, „Konzeptionierung einer Design-Based Research Studie zur Motivationsförderung durch den Kontext Eisenbahn im Rahmen der Behandlung von Datenbanken im Informatikunterricht der Oberstufe“.

[22] Bienbeck, „Konzeptionierung einer Design-Based Research Studie zum Einsatz von Computern im Informatikunterricht der Einführungsphase zum Thema Sortierverfahren im Kontext Eisenbahn“.

der zeitlichen Nähe der Erlebbarkeit, so könnte ein Kontext auch erst nach der Schulzeit für die Schülerinnen und Schüler eine Relevanz bekommen. Bei dem Genderaspekt lässt sich kritisieren, dass Schüler wahrscheinlich eine stärkere Faszination für das Thema aufbringen als Schülerinnen, jedoch kann der Kontext durch seine Präsenz in der Lebenswelt vieler Schülerinnen und Schüler trotzdem geeignet sein.
Das letzte Kriterium von Koubek ist die *Stabilität* des Kontextes. Ein Kontext ist *stabil*, wenn er über einen längeren Zeitraum relevant ist. Die ersten Eisenbahnen in Deutschland fuhren bereits ab 1835 und jährlich werden in Deutschland circa 6 Milliarden Euro in den Ausbau des Schienenverkehrs investiert, Tendenz steigend.[23] Dies lässt die Schlussfolgerung zu, dass Eisenbahnen als Kontext eine starke Stabilität besitzen.
Zusammenfassend lässt sich somit sagen, der Kontext Eisenbahn erfüllt die fünf geforderten Kriterien von Koubek und kann als Kontext innerhalb des Informatikunterrichts genutzt werden.

## 1.3. JMRI

JMRI- Java Model Railroad Interface ist ein Open-Source Projekt um Modelleisenbahnen per Computer zu steuern.[24] JMRI verwaltet dabei das gesamte Schienennetz, mit Weichen, Abstellgleisen, Lichtern, Sensoren, Signalen und Rückmeldern, aber auch die gesamte Flotte an Modelleisenbahnen die auf diesem Schienennetz fahren. Es können Gleisbilder, Kontrolltafeln und Schaltpulte angelegt werden. Der Quellcode kann heruntergeladen und angeschaut werden (https://github.com/JMRI/JMRI) und hat über zwei Millionen Zeilen Code. Ein anderes ähnliches Programm wäre Rocrail.[25] In dieser Bachelorarbeit wurde sich jedoch aufgrund der besseren Kompatibilität zum Arduino für JMRI entschieden. JMRI ist in fünf Programme unterteilt. Mit dem *DecoderPro* können DCC Zentralen über den Computer gesteuert und programmiert werden. Mit *DispatcherPro*, welches über das Hauptprogramm *PanelPro* zu öffnen ist, können die Modelleisenbahnen automatisiert werden. Ebenfalls über das Hauptprogramm zu öffnen ist *OperationsPro* mit dessen Hilfe die Umgebung der Lok gesteuert werden kann. Das sind zum Beispiel Autos, Schranken oder anderes Zubehör, welches bei der Fahrt der Lokomotiven bestimmte Aktionen durchführen soll. *SoundPro*, welches ebenfalls Bestandteil des Hauptprogramms

[23] Allianz pro Schiene, *Trend: Investitionen in die Schieneninfrastruktur in Deutschland von 2009-2020.*
[24] Java Model Railroad Interface, *About JMRI.*
[25] Rocrail, *Rocrail - Innovative Software zur Steuerung von Modelleisenbahnen.*

ist, steuert die Audioausgaben der Lokomotiven, falls diese Audio Decoder besitzen.[26] PanelPro ist das Hauptprogramm von JMRI und das Programm, welches in dieser Bachelorarbeit hauptsächlich genutzt wird. Da dieses Programm sehr umfangreich ist, lohnt es sich die dazugehörige Dokumentation zu lesen, um in das Programm einzusteigen. Für alle Handlungen die innerhalb der Bachelorarbeit in PanelPro bzw. JMRI durchgeführt werden, sind Anleitungen im Anhang zu finden.

## 1.4. Greenfoot

Greenfoot ist eine 2D interaktive Java-Entwicklungsumgebung, welche von der University of Kent entwickelt wurde.[27] Die Zielgruppe von Greenfoot sind Programmiereinsteiger ab 15 Jahren. Greenfoot Programme werden in Java, einer plattformunabhängigen Programmiersprache geschrieben und bieten so den Vorteil auf fast jedem Gerät ausführbar zu sein.[28] So ist das Programm für Schulen, unabhängig von ihrer Ausstattung einsetzbar. Viele Schulen[29,30] und Schulbücher[31] setzen bereits auf Greenfoot als Programmierumgebung, sodass diese Schülerinnen und Schülern gegebenenfalls bereits bekannt sein könnte. Im Rahmen einer Masterarbeit im Projekt LoCo wurde eine Schnittstelle zwischen Greenfoot und JMRI programmiert, die es erlaubt, über den JMRI Webserver an einen Computer angeschlossene Modelleisenbahnen zu steuern und Weichen zu schalten.[32] Diese Schnittstelle soll nun genutzt werden, um mit ihr über einen Arduino Uno Modelleisenbahnen zu steuern. Bisher geschieht dies bei dem Projekt klassisch, über eine DCC-Zentrale. Die Vorteile der Steuerung mithilfe des Arduino Unos gegenüber der klassischen Steuerung per DCC-Zentrale werden im nächsten Abschnitt erläutert.

## 1.5. Arduino Uno-DCC++

Arduino ist eine Physical-Computing-Plattform bestehend aus Software und Hardware. Die Hardware ist ein Mikrocontroller, in diesem Fall der Arduino Uno mit zugehöriger Software, der *Arduino IDE*, mit welcher der Programmcode zur Ausführung auf den

[26] Java Model Railroad Interface, *About JMRI*.
[27] King's College London, *About Greenfoot*.
[28] Kölling, „Greenfoot: a highly graphical ide for learning object-oriented programming."
[29] *Lernen am Emsland Informaitk*.
[30] *Gesamtschule Bad Lippspringe Informatikunterricht*.
[31] Grimm *u. a., Informatik - Lehrwerk für die gymnasiale Oberstufe*.
[32] Bienbeck, „Konzeptionierung einer Design-Based Research Studie zum Einsatz von Computern im Informatikunterricht der Einführungsphase zum Thema Sortierverfahren im Kontext Eisenbahn".

Mikrocontrollern programmiert und auf diese aufgespielt werden kann. Die Programmierung in der Arduino IDE erfolgt in einer auf C / C++ aufbauenden Programmiersprache, während die IDE selbst in Java programmiert ist, um die Zugänglichkeit zu verschiedenen Systemen zu vereinfachen. Die Programmiersprache wurde gegenüber C/C++ stark vereinfacht und es sind bereits standardmäßig viele Bibliotheken eingebunden. Die Programmierung erfolgt durch die Definitionen von zwei Funktionen: setup() und loop(). Die setup Funktion wird beim Start des Arduinos einmalig ausgeführt und legt beispielsweise Pins des Arduinos als Eingang oder Ausgang fest. Sobald die Funktion beendet ist, wird der *loop* solange durchlaufen, bis der Mikrocontroller abgeschaltet wird. Der Arduino Uno selbst besteht aus einem "ATMEGA 328P" Mikrocontroller, auf dem sich die Arduino-Bootloader-Software, die das Ausführen der Arduino Programme ermöglicht, befindet, einem "ATmega8U2" Mikrocontroller, der die Kommunikation mit dem über USB-Kabel verbundenen Computer übernimmt sowie vierzehn digitale In/Output Pins und 6 analogen Input Pins.[33]

DCC++ ist ein weiteres Open-Source Projekt welches den Arduino Uno zu einer DCC-Zentrale macht.[34] Für die Umfunktionierung des Arduino Unos zu einer DCC-Zentrale sind Veränderungen nötig, welche in Kapitel 3.1 genauer erläutert werden.

Die Vorteile des Arduino Unos gegenüber handelsüblichen DCC-Zentralen zur Nutzung in der Schule sind zum einen der Preis, der Arduino Uno[35] und das nötige Motor Shield[36] kosten jeweils 20€. Ein zusätzlich benötigtes 12V Netzteil gibt es ab 3€ womit der Gesamtpreis von 43€ für die Arduino Uno-Zentrale weniger als die Hälfte der handelsüblichen DCC-Zentralen ist, welche je nach Ausführung zwischen 80€ und mehreren hundert Euro liegen. Zum anderen kann über den Arduino Uno mithilfe von DCC++ die Steuerung von Modelleisenbahnen besser erläutert werden und es Schülerinnen und Schülern ermöglichen in das *Innere* einer DCC Zentrale zu schauen.

[33] Arduino, *Arduino Uno Rev3.*
[34] Berman, *DCC++ Basestation.*
[35] Arduino, *Arduino Uno Rev3.*
[36] Arduino, *Arduino Motor Shield Rev3.*

# 2. Technische Analyse

## 2.1. Wie funktionieren DCC Eisenbahnen?

Digital Command Control oder kurz DCC, ist ein 1994 von der National Model Railroad Association eingeführter Standard zur Steuerung von Modelleisenbahnen.[37] 2007 wurde dieser Standard von dem Verband der Modelleisenbahner und Eisenbahnfreunde Europas (MOROP) übernommen.[38]

Die Übertragung der Daten an die Modelleisenbahn erfolgt durch eine Übermittlung von Bits. Diese Bits werden durch den Spannungsverlauf auf dem Gleis übermittelt und bestehen immer aus Übergängen zwischen zwei gleichen Spannungsniveaus gegensätzlicher Polarität, auch Nulldurchgänge genannt. Ein Nulldurchgang in entgegengesetzter Richtung teilt das Bit in erste und zweite Hälfte, folgen zwei Nulldurchgänge gleicher Richtung aufeinander, folgt ein neues Bit. Das Einsbit wird durch zwei aufeinanderfolgende 58 µs lange Spannungsverläufe dargestellt, benötigt also 116 µs. Das Nullbit wird durch zwei 100 µs Spannungsverläufe dargestellt, wobei die Dauer der Nullbits variieren darf. So beträgt die Mindestdauer jeder Hälfte 95 µs und die Maximaldauer 9900 µs, wenn analoge Loks gesteuert werden sollen. Aus Gründen der Kompatibilität mit anderen Protokollen wird meist 116 µs pro Hälfte als Maximaldauer benutzt. Trotzdem müssen die Decoder Nullbits erkennen können, wo beide Hälften zwischen 90 µs und 10000 µs haben. Die Gesamtdauer eines Nullbits darf dabei jedoch 12000 µs nicht überschreiten.[39]

Durchschnittlich werden pro Sekunde 8000 Bits übertragen oder 141 DCC Datenpakete bestehend aus mindestens 16 Einsbits als Synchronbits, 3 Nullbits, welche die 3 Bytes einleiten und einem abschließenden Einsbit. Im Betriebsmodus sind dies ein Adressbyte, ein Befehlsbyte und ein Prüfbyte, welches per Exklusiv-Oder (EXOR) Verknüpfung aller vorangegangen Bytes gebildet wird. Im Programmiermodus sind das ein Befehlsbyte, ein Datenbyte und das Prüfbyte. Für die Zubehörsteuerung dürfen DCC Datenpakete auch länger sein, die maximale Länge eines Paketes ist nicht festgelegt.[40]

Das Adressbyte enthält die zugewiesenen Adressen des vorgesehenen Empfängers des

[37] National Model Railroad Association, *Communications Standards For Digital Command Control, All Scales.*

[38] Verband der Modelleisenbahner und Eisenbahnfreunde Europas, *Digitales Steuersignal DCC Bitdarstellung.*

[39] Verband der Hersteller Digitaler Modellbahnprodukte e.V, *RCN-211 DCC-Protokoll Paketstruktur und Adressbereiche.*

[40] Verband der Modelleisenbahner und Eisenbahnfreunde Europas, *Digitales Steuersignal DCC Basis-Datenpakete.*

| 1111111111111111 | 0 | xxxxxxxx | 0 | xxxxxxxx<br>76543210 | 0 | PPPPPPPP | 1 |
|---|---|---|---|---|---|---|---|
| Synchronbits | Startbit | Adressbyte | Startbit | Befehlsbyte | Startbit | Prüfbyte | Endbit |

Abbildung 2.1: *Darstellung eines DCC Datenpakets im Betriebsmodus*

Datenpaketes. Dieses ist in feste Blöcke unterteilt, um die Benutzung verschiedener Decoder zu ermöglichen (Abbildung 2.2[41]).

| 0000-0000 | Nachricht an alle Fahrzeugdecoder |
|---|---|
| 0000-0001 bis 0111-1111 | Fahrzeugdecoder mit 7 Bit Adressen 0AAA-AAAA |
| 1000-0000 bis 1011-1111 | Zubehördecoder mit 11 Bit Adressen 10AA-AAAA 1AAA-DAAR/0AA1 |
| 1100-0000 bis 1110-0111 | Fahrzeugdecoder mit 14 Bit Adressen 11AA-AAAA AAAA-AAAA |
| 1110-1000 bis 1111-1110 | Reserviert für zukünftige Anwendungen |
| 1111-1111 | Leerlauf oder auch Idle-Paket |

Abbildung 2.2: *Adressbyte Aufteilung(A=Adressbit,D=Datenbit)*

Das Befehlsbyte enthält die Informationen zur Steuerung des Empfängers. Die Bits 7 und 6 kennzeichnen als Nullbit und Einsbit das Datenbyte als Befehlsbyte. Bit 5 zeigt die Fahrtrichtung an. Ein Einsbit bedeutet in Richtung des Endes und ein Nullbit in die entgegengesetzte Richtung. Die Bits 4(C) und 3(S0)-0(S3) Steuern die Geschwindigkeit der adressierten Lok (siehe Abbildung 2.3)[42]. Das Prüfbyte ermöglicht den Decodern fehlerhaft übertragene Datenpakete zu erkennen. Sollte ein Datenpaket Fehler enthalten wird es ignoriert.[43]

| $S_3S_2S_1S_0C$ | Fahrstufe | $S_3S_2S_1S_0C$ | Fahrstufe | $S_3S_2S_1S_0C$ | Fahrstufe | $S_3S_2S_1S_0C$ | Fahrstufe |
|---|---|---|---|---|---|---|---|
| 0 0 0 0 0 | Stop | 0 1 0 0 0 | 5 | 1 0 0 0 0 | 13 | 1 1 0 0 0 | 21 |
| 0 0 0 0 1 | Stop** | 0 1 0 0 1 | 6 | 1 0 0 0 1 | 14 | 1 1 0 0 1 | 22 |
| 0 0 0 1 0 | EStop* | 0 1 0 1 0 | 7 | 1 0 0 1 0 | 15 | 1 1 0 1 0 | 23 |
| 0 0 0 1 1 | EStop** | 0 1 0 1 1 | 8 | 1 0 0 1 1 | 16 | 1 1 0 1 1 | 24 |
| 0 0 1 0 0 | 1 | 0 1 1 0 0 | 9 | 1 0 1 0 0 | 17 | 1 1 1 0 0 | 25 |
| 0 0 1 0 1 | 2 | 0 1 1 0 1 | 10 | 1 0 1 0 1 | 18 | 1 1 1 0 1 | 26 |
| 0 0 1 1 0 | 3 | 0 1 1 1 0 | 11 | 1 0 1 1 0 | 19 | 1 1 1 1 0 | 27 |
| 0 0 1 1 1 | 4 | 0 1 1 1 1 | 12 | 1 0 1 1 1 | 20 | 1 1 1 1 1 | 28 |

Abbildung 2.3: *Zusammenhang zwischen Kodierung der 4-0 und den Fahrstufen*
* Nothalt (emergency stop), Triebfahrzeuge stoppen so schnell wie möglich!
** Optional darf das Bit 5 für die Fahrtrichtungsinformation ignoriert werden.

Das DCC-Gleissignal wird auch zur allgemeinen Energieversorgung der Lokomotiven und des Zubehörs verwendet. Es muss daher durchgängig mit einem Abstand von 5 ms gesendet werden.

[41] Verband der Hersteller Digitaler Modellbahnprodukte e.V, *RCN-210 DCC-Protokoll Bit-Übertragung.*
[42] Verband der Modelleisenbahner und Eisenbahnfreunde Europas, *Digitales Steuersignal DCC Basis-Datenpakete.*
[43] Verband der Modelleisenbahner und Eisenbahnfreunde Europas, *Digitales Steuersignal DCC Basis-Datenpakete.*

## 2.2. Welche Voraussetzungen muss der Arduino erfüllen um eine DCC-Modelleisenbahn zu steuern?

Die Modelleisenbahn die im Projekt LoCo verwendet wird ist mit einem PIKO Smart-Decoder 4.1 ausgestattet, welcher eine Stromspannung von 12 Volt und eine Stromstärke von 1.2 Ampere benötigt.[44] Der Arduino Uno kann jedoch nur eine Spannung bis zu 5V verarbeiten und eine Stromstärke von ungefähr 50 mA.[45] Ebenso muss der Arduino die im Kapitel 2.1 erläuterten Signale erzeugen. DCC Signale sind Bi-Polar, springen also zwischen positiver und negativer Spannung hin und her. Der Arduino aber kann nur Signale positiver 5 Volt Spannung ausgeben oder seine Ausgänge auf 0 Volt schalten. Das Schalten der Ausgänge alle 58 µs oder 100 µs würde die komplette Rechenleistung des verbauten Mikrocontrollers benötigen. Daher kommen für die Erzeugung des DCC-Signals nur die Digital-PMW Pins in Frage. Die PMW Pins sind auf dem Arduino mit einer Tilde (~) versehen und können ihr Signal unabhängig vom primären Mikrocontroller, nämlich über Timer, erzeugen. Im Mikrocontroller sind 3 Timer verbaut. *Timer0* und *Timer2* sind 8-Bit Timer, *Timer1* ist ein 16-Bit Timer. Timer0 kontrolliert die Pins 5 und 6, Timer1 die Pins 9 und 10. Timer2 steuert die Pins 3 und 11. Timer2 ist für die Erzeugung der benötigten PMW Signale nicht geeignet, da die Funktionsweise unterschiedlich zu Timer0 und Timer1 ist.[46] Da die Timer jedoch nur periodisch Signale mit derselben Dauer erzeugen, wir aber zwischen 116 µs und 200 µs Signalen wechseln können müssen, um Einsbits und Nullbits zu erzeugen, wurde mit DCC++ die Funktionsweise der Timer so geändert, dass diese die Zeitabstände variieren können. Dadurch kann pro Timer jeweils nur ein Pin gesteuert werden. DCC++ hat die Pins 10 und 5 für diese Ausgabe gewählt. Pin 10 steuert die Hauptstrecke. Pin 5 wird für die Steuerung der Programmierstrecke benutzt. Da die Pins nur eine Stromstärke von 5 Volt abgeben, nutzen wir ein Motor Shield. Dieses kann eine Stromstärke bis zu 18 Volt abgeben. Wie in Kapitel 3.1 erläutert wird, müssen bestimmte Pins auf dem Motor Shield miteinander verbunden werden. Dies sind die Pins 5 und 13 sowie 10 und 12. Der Grund dafür ist, dass die Pins 12(A) und 13(B), wie auf dem Motor Shield beschriftet, die Ausgaberichtung des Stroms der Ausgänge A und B steuern. Ist der jeweilige Pin auf 0 Volt geschaltet, gibt das Motor Shield positive Spannung aus. Ist der Pin auf 5 Volt geschaltet, wird negative Spannung abgegeben. Somit kann der Arduino mithilfe eines Motor Shields ein DCC-Bipolares Signal auf die Schienen geben

[44] PIKO, *PIKO SmartDecoder 4.1.*
[45] Arduino, *Arduino Uno Rev3.*
[46] Atmel, *ATmega328P-Datenblatt.*

und ist in der Lage moderne DCC gesteuerte Modelleisenbahnen zu steuern.

# 3. Programmierung und Umbau des Arduino Uno

Im Folgenden werden die für den jeweiligen Zweck nötigen Umbauten am Arduino und der dafür nötige Programmcode erläutert. Die Möglichkeiten für den Informatikunterricht werden im nächsten Kapitel dargestellt.

## 3.1. Eisenbahnsteuerung

Um die Eisenbahn zu steuern benötigen wir einen Arduino Uno, ein Arduino Motor Shield, ein 12 Volt 1.2 Ampere AC zu DC Netzteil und 2 Verbindungskabel. Je nach Gleishersteller noch zusätzliche 2 Verbindungskabel um das Gleis und das Arduino Motor Shield zu verbinden. Da der Arduino, wie in Kapitel 2.2 erläutert, nur 5 Volt Spannung verarbeiten kann, jedoch das Motor Shield mit 12 Volt zur Benutzung der Lokomotive belastet werden muss und dieses diese Spannung an den Arduino weiterleiten würde, muss am Motor Shield selber eine Anpassung vorgenommen werden. Auf der Rückseite des Motor Shields befindet sich eine Lötstelle mit der Aufschrift Vin Connect (siehe Abbildung 3.1).

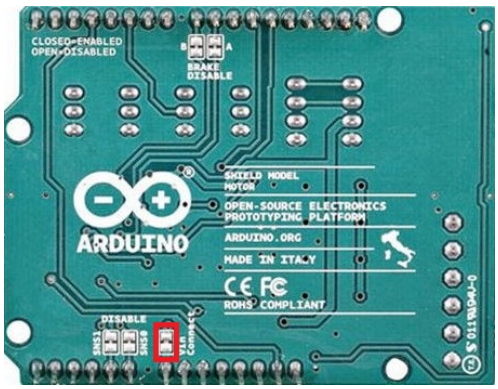



Abbildung 3.1: *Rückseite des Arduino Motor Shields. Lötstelle ist Rot Markiert*

Die Verbindung dieser Lötstelle muss vorsichtig mit einem scharfen Messer durchtrennt werden, damit kein Strom vom Motor Shield zurück zum Arduino fließt. Der Stromfluss vom Arduino zum Motor Shield ist davon nicht betroffen. Danach wird das Motor Shield auf dem Arduino montiert. Hierbei ist darauf zu achten, dass alle Pins eingeführt werden und keiner der Pins abknickt. Anschließend werden die Pins 5 und 13 sowie die Pins 10 und 12 durch die 2 Verbindungskabel miteinander verbunden. Die Gleise werden an den *A* Anschluss des Motor Shields angeschlossen. Möglicherweise genutzte Programmiergleise am *B* Eingang. Diese werden jedoch nicht zwingend benötigt. Das Netzteil wird mit

dem + Leiter an Vin und mit dem - Leiter an GND angeschlossen (siehe Abbildung 3.2). Aus Sicherheitsgründen wird erst im Anschluss das Netzteil in die Steckdose gesteckt.

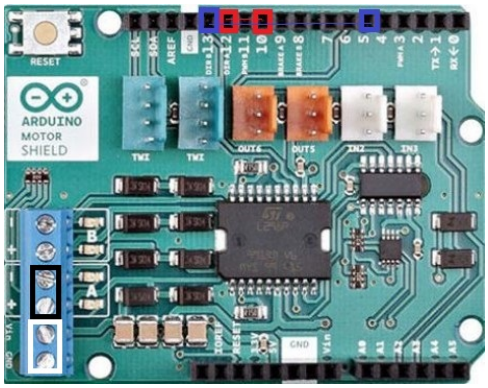



Abbildung 3.2: *Vorderseite des Arduino Motor Shields. Zu verbindende Pins sind Farbig markiert. Schwarz ist der Gleisanschluss. Weiß der Stromanschluss*

Der Arduino wird standardmäßig per USB an den Computer angeschlossen. Eine Anleitung für die Einrichtung des Arduinos ist auf der Website des Herstellers verfügbar. Über die Arduino IDE wird die DCC++ Basestation (https://github.com/DccPlusPlus) auf den Arduino geladen. Anschließend kann über PanelPro[47] die Eisenbahn hinzugefügt werden. Eine Anleitung befindet sich im Anhang A.1. Unter Werkzeuge kann nun ein neuer Fahrregler erstellt werden, welcher die Eisenbahn steuern kann.

## 3.2. Weichensteuerung

Es gibt verschiedene Möglichkeiten Weichen zu steuern. Zum einen gibt es die Möglichkeit direkt über das Arduino Motor Shield oder über einen L298 Motor Driver, der an den Arduino angeschlossen ist, die Weichen per Knopfdruck zu schalten oder per JMRI die Weichen direkt anzusteuern. Der Vorteil des Motor Drivers liegt darin, dass darüber zwei Weichen gesteuert werden können, während über das Motor Shield nur eine einzelne Weiche geschaltet werden kann. Außerdem würde die Weiche dieselben Anschlüsse wie die Gleise verwenden, weshalb diese Variante nicht für den Einsatz empfohlen werden kann.
Die Weichen schalten bei einem Stromimpuls per Magnetspule. Die Magnetspule wird per Wechsel von Plus und Minus des angeschlossenen Gleichstroms gesteuert. Nachfolgend werden verschiedene Möglichkeiten zur Umsetzung einer Weichensteuerung beschrieben. Angefangen mit einer simplen Schaltung per Knopfdruck, die zu einer Schaltung per

[47] Java Model Railroad Interface, *About JMRI*.

DCC-Zentrale umgebaut wird.

### 3.2.1. Eine Weiche steuern

Um eine Weiche steuern zu können, werden, wie für die Eisenbahnsteuerung, ein Arduino Uno, ein Arduino Motor Shield und ein 12 Volt 1.2 Ampere AC zu DC Netzteil benötigt. Zusätzlich eine Steckplatte, Verbindungskabel, zwei 1k Ohm Widerstände, zwei LED's und ein Schaltknopf oder ähnliches. Die Verbindung des Arduinos mit dem Motor Shield und des Netzteils erfolgt wie in Kapitel 3.1 erläutert. Der Aufbau auf der Steckplatte erfolgt gemäß den Abbildungen. Die Weiche wird an Eingang A des Motor Shields angeschlossen, wie vorher die Gleise angeschlossen wurden.
Der Programmcode im Anhang ist einmal für die Steuerung ohne (B.1) und erweitert für die Steuerung mit Lampe (B.2) angegeben. Der Programmcode wird auf den Arduino

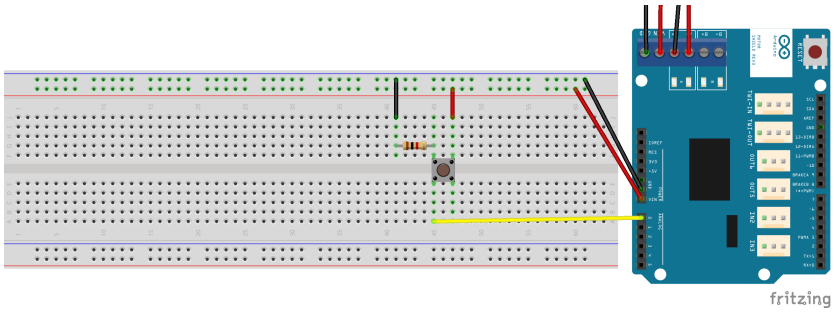



Abbildung 3.3: *Steckplatine für die Weichenschaltung ohne Lampe*

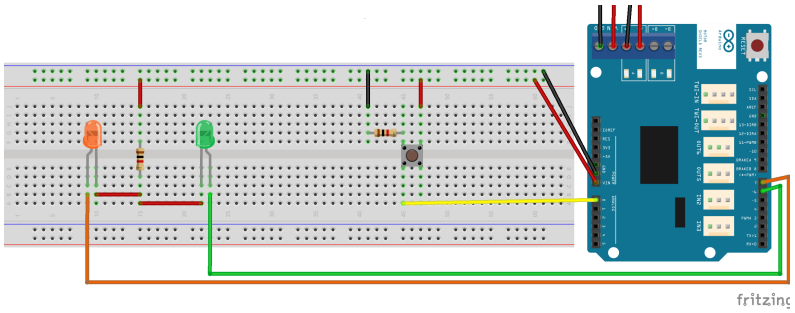



Abbildung 3.4: *Steckplatine für die Weichenschaltung mit Lampe*

geladen und nun wird bei Betätigung des Schaltmechanismus die Weiche geschaltet.

### 3.2.2. Mehrere Weichen steuern

Um zwei (oder mehr Weichen) zu schalten werden ein Arduino Uno, eine Steckplatte, zwei Schaltknöpfe oder ähnliches pro Weiche, zwei 1k Ohm Widerstände pro Weiche, ein L298 Motor Driver pro zwei Weichen, ein 12 Volt 1.2 Ampere AC zu DC Netzteil, 2

Kabel für die Stromzufuhr und 6 Kabel per Weiche benötigt. Der Aufbau wird für zwei Weichen in der Abbildung 3.5 erläutert, erfolgt für mehr als zwei Weichen aber analog auf derselben oder weiteren Steckplatten. Dieser Aufbau ist nur beispielhaft und kann auch mit nur einem Knopf pro Weiche oder etwa einem Potentiometer aufgebaut werden. Weiche 1 wird an die Anschlüsse OUT1 und OUT2 des Motor Drivers angeschlossen und Weiche 2 an die Anschlüsse OUT3 und OUT4. Der Programmcode (B.3) wird auf den Arduino geladen. Nun schaltet die Weiche, je nach Anschluss an den Motor Driver bei einem der Schaltknöpfe geradeaus und in *die Kurve* bei dem anderen Schaltknopf. Sollen die Knöpfe mit der jeweiligen Richtung beschriften werden, muss eventuell der Anschluss der Weiche an OUT1 und OUT2 getauscht werden.

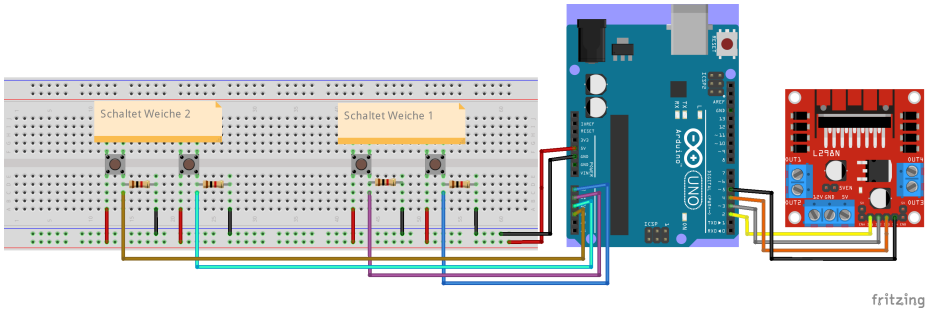



Abbildung 3.5: *Steckplatine für die Weichenschaltung von zwei Weichen*

### *3.2.3. Mehrere Weichen per JMRI steuern*

Per JMRI haben wir die Möglichkeit bis zu 6 Weichen direkt über zwei Arduinos zu steuern. Die Anzahl der Weichen ist hier begrenzt durch den Einsatz eines Arduino Unos, da dieser aufgrund seiner begrenzten Ein- und Ausgänge nicht mehr als drei Motor Driver unterstützt. Durch den Austausch der Arduino Unos mit Arduino Megas könnten bis zu 26 Weichen gesteuert werden. Der eine Arduino wird wie bei Kapitel 3.1 mit dem Motor Shield zusammengebaut und verkabelt. Der andere Arduino wird mit dem Motor Driver verbunden. Das Motor Shield wird nun per Kabel von einem der freien Ausgänge mit den analogen Eingängen des Arduinos verbunden der die Motor Driver steuert.(Abbildung 3.6)

Wenn der Aufbau abgeschlossen ist können das Motor Shield und die Motor Driver an das Netzteil angeschlossen werden. Bei vielen zu schaltenden oder älteren Weichen empfiehlt es sich ein Netzteil mit circa 20 Volt zu verwenden und die Stromzufuhr auf circa 800 Millisekunden zu verlängern.

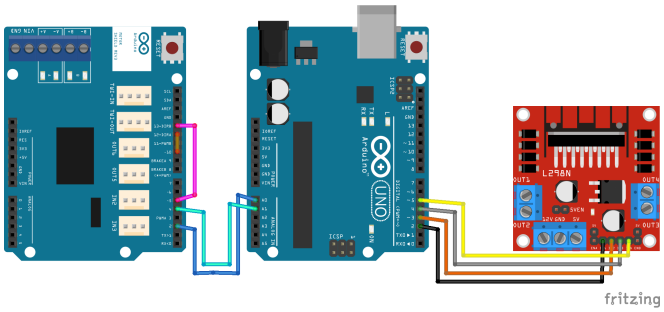



Abbildung 3.6: *Arduino mit Motorshield und Motor Driver für die Weichensteuerung über JMRI*

### *3.2.4. Weichen per Sensor steuern*

Weichen können über einen an den Arduino angeschlossenen Sensor automatisch gesteuert werden. Als Sensoren wurden hier Infrarotsensoren gewählt, aber es ist jede Art von Sensor einsetzbar dessen Output geerdet ist, wenn er aktiviert wird. Pro Weiche werden je nach Umsetzung 1 oder 2 Sensoren zusätzlich zu den in den vorherigen Kapiteln benötigten Materialien benötigt. Die Sensoren benötigen einen 5V Anschluss und eine Erdung die über den Arduino und ein Steckbrett geliefert werden können. Als Eingang für den Arduino können die analogen Eingänge des Arduinos genutzt werden (Abbildung 3.7).

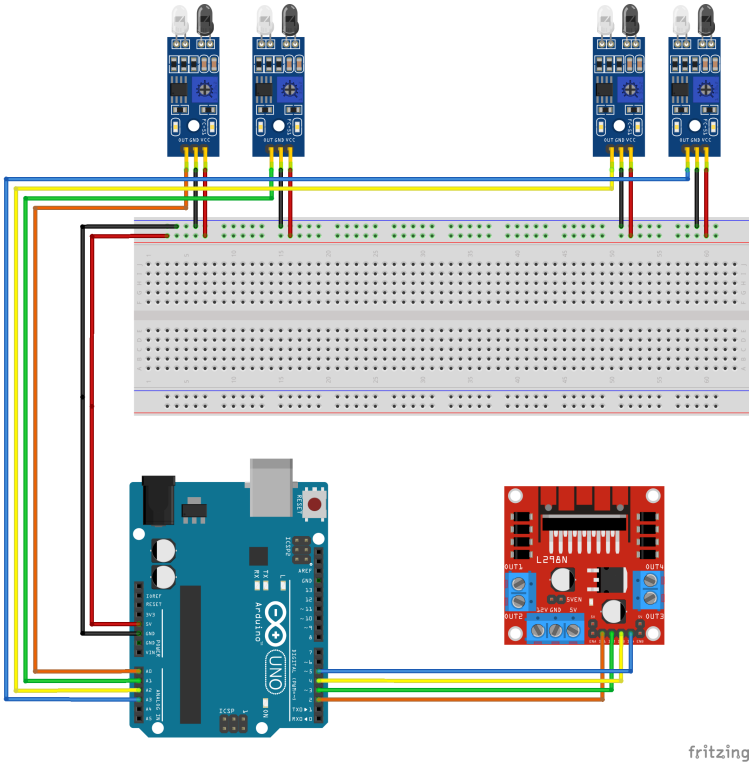



Abbildung 3.7: *Steckplatine für die Weichenschaltung mit Sensoren*

Die Sensoren müssen je nach Umsetzung in der Arduino IDE an den Gleisen angebracht werden, um bei Durchfahrt der Lokomotive die Weiche zu schalten.

## 3.3. Automatisierung mit Sensoren

JMRI unterstützt verschiedene Möglichkeiten Sensoren einzusetzen. Um die Möglichkeit mehrere Sensoren einzusetzen offenzuhalten, wird ein Arduino Mega[48] mit Sensor Shield[49] eingesetzt. Dadurch können bis zu 67 Sensoren gleichzeitig eingesetzt werden. JMRI unterstützt auch die Einrichtung von Sensoren auf dem Arduino der für die DCC-Zentrale genutzt wird. Da mit der Weichensteuerung von mehr als zwei Weichen jedoch ein Großteil der Pins bereits belegt ist, wird der Arduino Mega eingesetzt.[50] Das Sensor Shield wird auf dem Arduino Mega montiert und der Programmcode[51] wird auf den Arduino geladen. Eine Anleitung dazu befindet sich im Anhang D. Die Sensoren werden an die jeweiligen Pins des Sensor Shields angeschlossen(Abbildung 3.7).

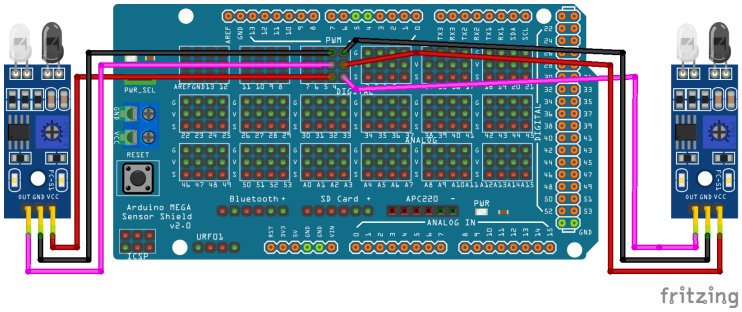



Abbildung 3.8: *Verbindung der Sensoren mit dem Sensor Shield*

In JMRI richten wir das Sensor Shield per Skript ein[52] und fügen die Sensoren entsprechend der Anleitung im Anhang hinzu (E). Die jeweiligen Sensoren heißen nach Einrichtung ”AR*PinNummer*” und können darüber in den Skripten angesprochen werden. Mit den eingerichteten Sensoren lassen sich nun Python bzw. Jython Skripts in JMRI ausführen.

Von JMRI sind einige Beispiele für Skripts[53] online zu finden. Diese müssen jedoch, je nachdem auf welcher DCC Adresse die Lok die gesteuert werden soll, eingerichtet ist abgeändert werden. Eine Anleitung befindet sich im Anhang (E.1). Die Skripts in Kapitel 4.3 dieser Bachelorarbeit sind aufgrund der besseren Unterstützung in Jython geschrieben, unterscheiden sich syntaktisch aber nicht von Python. Der einzige Unterschied liegt darin,

[48] Arduino, *Arduino Mega 2560 Rev3.*
[49] Keyestudio, *Keyestudio MEGA Sensor Shield V1.*
[50] Geoff, *SMA28 JMRI Sensor Channels – Direct Arduino to JMRI Communications - Simple Support for Lots of Detectors.*
[51] Geoff, *SMA28 JMRI Sensor Channels – Direct Arduino to JMRI Communications - Simple Support for Lots of Detectors.*
[52] Geoff, *SMA28 JMRI Sensor Channels – Direct Arduino to JMRI Communications - Simple Support for Lots of Detectors.*
[53] Java Model Railroad Interface, *JMRI:Scripting.*

dass die Importierung von Python spezifischen Bibliotheken in Jython nicht möglich ist, sondern stattdessen auf Java Bibliotheken zugegriffen werden muss.
Durch das Einsetzen von Skripts können verschiedenste Funktionen wie anhalten oder beschleunigen der Lok, schalten von Weichen, Signallampe oder Gleisbelegung automatisiert werden. Hier besteht eine große Variation an Verwendungsmöglichkeiten der Skripts. Die Anleitung für die Verwendung von Skripten innerhalb von JMRI befindet sich im Anhang unter F.2. Die Beispiele bieten für die meisten umsetzbaren Funktionen Skripts in denen diese benutzt werden. Die Funktionen zur Steuerung des einzelnen Zubehörs können aber auch in den Klassen im JMRI Programmcode nachgeschaut werden. Zusätzlich befindet sich im Anhang unter F.1 eine Liste mit den wichtigsten Funktionen der Klassen die für einfache Skripts benötigt werden.

### *3.3.1. Gleisbelegung*

Ein Beispiel für ein Skript, welches in späteren Kapiteln ebenfalls benötigt wird, ist die Umsetzung der Gleisbelegung in JMRI mit dem Arduino. Dieses Skript ist ebenfalls im Anhang zu finden (H.1). Dafür werden neben der Arduino DCC Station, der oben benannte Arduino Mega mit Sensor Shield sowie die gewünschte Zahl an Sensoren entsprechend den Streckenabschnitten die als Belegt markiert werden sollen, benötigt. Die beiden Arduinos werden entsprechend der Anleitungen in den vorherigen Kapiteln eingerichtet und die Sensoren an den Enden der jeweiligen Streckenblöcke angebracht und mit dem Sensor Shield verbunden. Innerhalb des Skripts müssen die Namen der Sensoren und Blöcke sowie die DCC Adresse des Zuges angepasst werden. Danach setzt das Skript mithilfe einer einfachen IF-Abfrage beim Aktivieren des entsprechenden Sensors den zugeordneten Streckenabschnitt in JMRI auf besetzt. Dafür wurden zwei Funktion definiert. Eine um den entsprechenden Streckenabschnitt aktiv zu schalten, die andere um alle anderen Abschnitte wieder inaktiv zu schalten. JMRI weiß bei Durchführung dieses Skriptes nicht welche Lok sich auf den Schienen befindet. Dieses Skript kann genutzt werden um eine entsprechende Greenfoot Funktion zu schreiben, die nur auf ”besetzt” überprüft ohne Rückmeldung des Loknamens. Das entsprechende Greenfoot Beispielprogramm zu der JMRI Schnittstelle verwendet allerdings eine solche Rückmeldung, welche in 3.4 behandelt wird. Zubehör wie Weichen, Lichter oder Signallampen können ebenfalls über Skripts gesteuert werden.

## 3.4. DCC Zentrale/Gleisbelegung mit Rückmeldung der Lok

Die JMRI-Greenfoot Schnittstelle[54] baut unter anderem auf der Rückmeldefunktion von JMRI auf. Hierbei werden mit speziellen Sensoren, sogenannten Meldern, die DCC Adressen der Loks an JMRI und damit auch Greenfoot zurückgegeben. Damit kann nicht nur eine Gleisbelegung dargestellt werden, wie im Skript welches in 3.3.1 genutzt wird, sondern gleichzeitig der in JMRI vergebene Name der Lok, welche den Gleisblock belegt, zurückgegeben werden. Der Arduino kann diese Funktion standardmäßig nicht ausführen, da mit den zur Verfügung stehenden Mitteln nicht überprüft werden kann welche Lok die Sensoren passiert. Über eine Modifizierung des Gleisbelegungsskripts kann dieses Problem aber gelöst werden. Bei nur einer Lok ist die Lösung einfach. Es wird jedem Gleisblock in JMRI ein Melder zugeordnet. Diese Melder müssen physisch nicht vorhanden sein, sondern es wird per Skript beim Aktivieren des Sensors, welcher den Gleisabschnitt als belegt markiert der Name der Lok an den assoziierten Melder übergeben und aus dem vorherigen Melder gelöscht. Das entsprechend überarbeitete Skript findet sich im Anhang unter H.2. Damit diese Art der Rückmeldung funktioniert müssen einige Änderungen in dem Programmcode des Greenfoot Beispiels vorgenommen werden (G.1).
Der Aufbau gleicht dabei dem der für die Automatisierung mit Sensoren beschrieben wurde. Es werden Sensoren und ein Arduino Mega mit Sensor Shield benötigt, welcher die Sensoren steuert, ein Arduino Uno mit Motor Shield mit DCC++ welcher als DCC-Zentrale dient und bei zusätzlicher Weichenschaltung ein Motor Driver, welcher an einen Arduino auf dem das entsprechenden Programm zur Weichenschaltung läuft 3.2.3. Das Schienenlayout welches benutzt werden soll muss in JMRI angelegt werden (A.2) und muss dem in Greenfoot zu erstellendem Layout gleichen, damit die Darstellung sinnvoll wird.

[54] Bienbeck, „Konzeptionierung einer Design-Based Research Studie zum Einsatz von Computern im Informatikunterricht der Einführungsphase zum Thema Sortierverfahren im Kontext Eisenbahn“.

# 4. Möglichkeiten für den Informatikunterricht

Mit dem Einsatz eines Arduinos können verschiedene Kompetenzbereiche und Inhaltsfelder des Kernlernplans Informatik Nordrhein-Westfalens für die Oberstufe[55] behandelt werden. Im Folgenden werden die in Kapitel 3 erläuterten Möglichkeiten Modelleisenbahnen über einen oder mehrere Arduinos zu steuern, anhand des Kernlernplans und den drei Kriterien die Koubek für die Kontextualisierung von Informatikunterricht aufgestellt hat, auf deren Nutzen analysiert.

## 4.1. Eisenbahnsteuerung

Die Steuerung einer Modelleisenbahn per Arduino kann schlecht analysiert werden, da das dazu notwendige Programm, DCC++, für Schüler ausgesprochen umfangreich ist. So benötigt beispielsweise alleine die Erzeugung des DCC Signals über 450 Zeilen Code. Das Programm kann jedoch als Grundlage für die weitergehende Steuerung der Weichen, Sensoren und Lichter genutzt werden. Es kann die Theorie hinter der Erzeugung der DCC-Signale analysiert werden. In Kapitel 2.1 werden die Bit-Darstellungen der Signale die die DCC-Zentrale auf die Gleise gibt erläutert. Nach einer Einführung in Binärzahlen und einer Erläuterung über die Steuerung von Modelleisenbahnen mit DCC-Signalen können Schülerinnen und Schüler mithilfe von Tabellen verschiedene DCC-Befehle analysieren oder aufstellen. Ein Beispiel dafür befindet sich im Anhang unter C.1. Hierbei wird der inhaltliche Schwerpunkt Digitalisierung aus Inhaltsfeld 4 *Informatiksysteme* sowie der Kompetenzbereich *Darstellen und Interpretieren* behandelt[56].
Die ”Kontextfamilie”[57] ist in der gesamten Bachelorarbeit die Eisenbahn. Als konkreter Kontext kann hier die Steuerung einer Modelleisenbahn genannt werden. Der Kontext wird genutzt um den informatischen Bereich der Binärzahlen anhand der DCC-Steuerung zu unterrichten. Zusätzlich zu dem informatischen Kontext kann hier auch über Sicherheitsrisiken bei der Informationsübertragung gesprochen werden. Dazu könnten aktuelle Beispiele von autonomen Zügen, die Unfälle verursacht haben, als Beispiel dienen. Schülerinnen und Schüler könnten auch anstatt der Analyse der DCC-Signale in Partnerarbeit

[55] Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen, „Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen Informatik“.
[56] Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen, „Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen Informatik“.
[57] Koubek *u. a.,* „Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht“, S. 272.

ein eigenes Binärsystem entwickeln, welches zur Übertragung der Signale genutzt werden soll. Diese Signale können anschließend in Gruppen auf mögliche Sicherheitsrisiken untersucht und verbessert werden. Abschließend wird im Unterrichtsgespräch eine gemeinsame ”ideale” Lösung entwickelt und mit der realen Umsetzung der DCC-Signale verglichen.
Das Programm DCC++ kann außerdem dazu beitragen zu verdeutlichen wie umfangreich Programme auch in Informatiksystemen sein können, in denen die Schülerinnen und Schüler keine Informatik erwarten.

## 4.2. Weichensteuerung

Die Steuerung der Weichen wird in der Arduino IDE in C/C++ programmiert und eignet sich somit für Schulen die auf C/C++ als Programmiersprache setzen. Es gibt verschiedene denkbare Möglichkeiten die Weichensteuerung im Informatikunterricht einzusetzen. Es wäre eine Analyse, im Sinne von Inhaltsfeld 2 *Algorithmen* des Kernlernplans, der ersten vorgestellten Weichensteuerung per Knopfdruck ohne Lampe (3.2.1) denkbar. Diese könnte dann von den Schülerinnen und Schülern eigenständig, schrittweise auf die nachfolgenden Weichensteuerungen modifiziert werden. Dabei würden die Kompetenzerwartungen *Argumentieren* und *Implementieren* behandelt. Die nötigen technischen Schaltungen, also die Steckplatine und die Verbindungen mit dem Arduino selbst könnten dabei vorgegeben werden. Eine andere Möglichkeit wäre es mit einem vorherigen Exkurs in die Technische Informatik und in den Arduino die Schülerinnen und Schüler die Schaltungen selbst erarbeiten zu lassen. Dieser Exkurs kann dabei auch den Einsatz von Computersystemen und Mikrocontrollern in alltäglichen Gegenständen, zu denen die Schülerinnen und Schüler einen Lebensweltbezug haben, verdeutlichen. Gerade durch die Konkretisierung eines solchen Mikroprozessors durch den Arduino wird das ansonsten abstrakte Thema für Schüler verständlicher[58]. Die dabei nötigen Änderungen für die einzelnen Programme sind leicht umzusetzen und basieren alle auf derselben Logik. Rein die Belegung der jeweiligen Ein- und Ausgänge des Arduinos müssen dabei verändert sowie die Logik des Ersetzen eines Knopfes durch die DCC-Zentrale verstanden werden. Es wäre also auch eine Aufgabenstellung möglich in der nur die Vorgabe gegeben wird, dass die Weichen schalten müssen. Dafür müssen gewisse Grundkenntnisse in der Logik der verwendeten Hardware vorhanden sein, sodass die Schülerinnen und Schüler diese Aufgabenstellung

[58] Schubert & Schwill, *Didaktik der Informatik,* S. 216–218.

und die hinter der Weichenschaltung liegenden Logik durchaus selbst erarbeiten können. Eine beispielhafte Unterrichtseinheit à 90 Minuten befindet sich im Anhang unter I und wird in Kapitel 5 betrachtet. Dabei wird auch auf die Schwierigkeiten bei der Umsetzung der Weichenschaltung mit Sensoren eingegangen. Durch die analogen Rückmeldungen der Weichen, schalten per Knopfdruck, Sensor oder eben per JMRI, haben die Schüler ein direktes Erfolgserlebnis, welches daraus folgt, dass nicht nur der Computer "etwas tut" sondern, dass durch die Programmierung etwas außerhalb des Computers geschieht.[59]

## 4.3. Automatisierung mit Sensoren

Die Skripts die für die Automatisierung benötigt werden, werden in Python / Jython oder JavaScript geschrieben. In dieser Arbeit werden nur Jython Skripts vorgestellt, die von der Syntax komplett Python gleichen. Da Python als Programmier-Lehrsprache entwickelt wurde und gegenüber JavaScript den Vorteil hat auch außerhalb von Skripten und Webentwicklung eingesetzt zu werden und zudem eine einfache Syntax zu haben, ist Python in Schulen geläufiger als JavaScript und könnte Schülerinnen und Schülern somit bereits bekannt sein.[60] Einfache Skripts, die beispielsweise einfach nur die Geschwindigkeit der Bahn steuern sobald diese einen Sensor passiert, können ohne Kommentare mit 20 Zeilen Code geschrieben werden, wobei für den Anfang die grobe Struktur mit Importen, Kopfteil der Klasse sowie init() und handle() vorgegeben werden sollten, um Syntaxfehlern vorzubeugen. Komplizierte Skripts die beispielsweise ganze Bahnreisen, mit Bahnhofshalten, akustischen Rückmeldungen der Lok, Weichensteuerung und Gleisbelegung steuern, können ohne Kommentare schon 200 Zeilen benötigen, je nach Ausführung. Somit eignen sich die Skripts für verschieden starke Schülerinnen und Schüler in einem Kurs, die je nach Lerntempo eigene Skripts schreiben können. Ein großes Problem dabei, ist der fehlende Debugger. Die Skripts selbst können zwar mit passender IDE auf Syntaxfehler überprüft werden aber für Logikfehler steht nur die JMRI Systemkonsole zur Verfügung welche keinen Debugger besitzt. Dies kann schnell zu großer Frustration führen, da Fehler nur schwer entdeckt und behoben werden können.[61] Zusätzlich sind die Möglichkeiten von Modelleisenbahnen mithilfe von Skripten zwar vielfältig und erlauben es den Schülerinnen und Schülern mit Kreativität an mögliche Aufgabenstellungen heran-

[59] Bergner & Schroeder, „Informatik Enlightened - Informatik (neu) beleuchtet dank Physical Computing mit Arduino".
[60] Grandell *u. a.,* „Why complicate things? Introducing programming in high school using Python".
[61] Michaeli, „Debugging im Informatikunterricht".

zugehen, jedoch sind die nötigen Funktionen im umfangreichen Programmcode von JMRI zu finden. Also müsste entweder die Lehrkraft die für die Aufgabenstellungen nötigen Funktionen vorgeben, was dem Aspekt der Kreativität und des Selbstbestimmten Lernens entgegenspräche oder die Schülerinnen und Schüler müssten alle nötigen Klassen aus JMRI zur Aufgabenstellung gestellt bekommen und selbst nötige Funktionen suchen. Dieses ist je nach Aufgabenstellung mit viel Zeit verbunden und muss berücksichtigt werden.
Die Arbeit mit Skripten in JMRI kann verschiedene Inhaltsfelder und Kompetenzbereiche abdecken. So müssen etwa aus dem Inhaltsfeld *Daten und ihre Strukturierung*, Eigenschaften, Operationen und Beziehung von Objekten aus dem Kompetenzbereich *Modellieren* untersucht werden. Es müssen selbst Klassen modelliert, Algorithmen analysiert und entworfen und Bibliotheken analysiert werden.
Aber es gibt auch Python spezifische Kompetenzerwartungen, die nicht berücksichtigt werden können, etwa die der Datentypzuordnung. Da Python Attributen und Parametern automatisch Rückgabewerte gibt, ist diese Art der Programmierung zwar weniger fehleranfällig, jedoch gerade für Programmieranfänger später eine große Umstellung, wenn nicht dauerhaft in dieser Programmiersprache programmiert werden soll. Sollte eine Schule im Unterricht mit Python programmieren, sind die Skripts eine Möglichkeit über ein externes Informatiksystem, dem Arduino und einer Modelleisenbahn analoge Ergebnisse zu sehen. In den Unterkapiteln werden trotz der negativen Einschätzung JMRI Skripts einzusetzen, einige Skripts vorgestellt, über die der Einsatz der Greenfoot JMRI Schnittstelle ermöglicht werden soll.

### *4.3.1. Gleisbelegung*

Die Rückmeldung für die Gleisbelegung ist ein Simpler Code, der auf unterschiedliche Weise eingesetzt werden kann. Hauptsächlich ist dieser Code die Hinleitung zu der Gleisbelegung mit Lok Rückmeldung wie im nächsten Kapitel beschrieben. Denkbar wären eine Analyse des Codes mit nachfolgender Weiterentwicklung mit Lok oder gar eine selbstständige Erarbeitung des Codes mit Vorgabe der einzelnen Befehle oder der Befehlsliste aus dem Anhang. In beiden Fällen müssen Schülerinnen und Schüler einfache While-Schleifen und IF-Anweisungen einsetzen um zur Lösung zu kommen. Da die Befehle sehr repetitiv sind, können auch kompliziertere Strukturen wie Dictionarys eingesetzt werden, wie es als Beispiel in dem Skript für die Gleisbelegung mit mehreren Lokomotiven vorgegeben ist. Dadurch wird der Code, der ansonsten aus den gleichen immer wiederkeh-

renden Befehlen besteht, durch mehr Logik mit Einsatz von verschiedenen Schleifen und Datenstrukturen aufgewertet. Dabei ist zu beachten, dass die Programmiersprache Jython nicht auf der aktuellen Python Version basiert, sodass einige Funktionen oder auch Datentypen von Python nicht vollständig unterstützt werden. Dies ist im Normalfall für den Gebrauch im Informatikunterricht aber irrelevant.

## 4.4. DCC Zentrale zur Steuerung mit Greenfoot

Die Steuerung der Lokomotive über Greenfoot mit dem Arduino ist einfach umzusetzen. Es müssen lediglich die Systemnamen der Weichen und der Eisenbahn in *PanelWorld* abgeändert und der Teil des Codes aus *PanelWorld* der für die Streckenerkennung zuständig ist gelöscht werden. Damit lassen sich die Eisenbahn und die Weichen über Greenfoot steuern. Wenn die Gleiserkennung mit Rückmeldung des in JMRI gesetzten Namens der Lokomotive genutzt werden soll, sind die dazu nötigen Änderungen in dem Skript für die Gleisbelegung überschaubar und mit Vorgabe der Befehle, welche aus dem Quellcode von JMRI ausgelesen werden müssen, durchaus umsetzbar für Schülerinnen und Schüler. Die nötigen Änderungen in JMRI selbst, also die zusätzlichen Melder, Zuweisungen, etc., sind recht umständlich und bieten auch keinen Nutzen im Sinne des Kernlernplans. Das Skript kann so abgeändert werden, dass es für mehrere Lokomotiven einsetzbar ist H.3.
Mit dem Skript steht mit Greenfoot eine IDE zur Verfügung die mit Java arbeitet. Dies hat den Vorteil, dass viele Schulen bereits Java nutzen und im speziellen auf Greenfoot im Informatikunterricht setzen. In Greenfoot selbst könnten Schülerinnen und Schüler nun mithilfe des JMRI Greenfoot Interfaces, nach und nach ein Streckenlayout erstellen. Zuerst einfache Kreise und dann können zusätzlich Weichen und die Zugerkennung programmiert werden. Da Greenfoot selbst nur auf die Funktionen der Schnittstelle und nicht auf JMRI zugreift und diese deutlich weniger Funktionen als JMRI selbst hat, ist die Programmierung in Greenfoot zwar beschränkter aber für Schülerinnen und Schüler vermutlich einfacher und übersichtlicher. Möglichkeiten für den Informatikunterricht ergeben sich in Greenfoot viele. Beispielsweise ein Ausbau der Benutzeroberfläche mit einem neuen Kopf für individuelle Zugsteuerung (siehe: 4.4.1) oder die Umstrukturierung auf zwei Strecken mit jeweils einer Lokomotive darauf (siehe: 4.4.2), welche beispielhaft für die verschiedenen Möglichkeiten erläutert werden.

### 4.4.1. Individuelle Zugsteuerung

Die Umsetzung einer Steuerung die Züge individuell anspricht und nicht wie bisher bereits im Programmcode über die Adresse des Zuges festgelegt ist, kann in vielen Arten ausgeführt werden. Eine beispielhafte Umsetzung mit einem sich per Knopfdruck öffnenden Dialogfeld befindet sich im Anhang unter G.2. Der Vorteil dieser Art der Steuerung gegenüber der bisherigen ist die Flexibilität und die Anpassbarkeit des Codes. Wenn mehr als ein Zug auf dem Layout fährt, müssten ansonsten für jeden Zug viele Buttons programmiert werden, damit beide Züge gesteuert werden können. Wenn die DCC-Adresse jedoch nicht als Parameter bei der Erstellung der Buttons übergeben werden muss, sondern erst bei Knopfdruck in ein Dialogfeld eingegeben wird, reicht pro Funktion ein Button über den dann alle Lokomotiven einzeln gesteuert werden können. Aus Sicherheitsgründen empfiehlt es sich jedoch einen einen Notaus Button für alle Lokomotiven zu implementieren um mögliche Zusammenstöße zu verhindern.
Da Schülerinnen und Schüler bei der Umsetzung beispielsweise auf gut dokumentierte Java Bibliotheken wie *javax.swing* zurückgreifen und so nach eigenem Belieben ihre Benutzeroberfläche anpassen können, bietet dieses Thema eine Vielzahl an inhaltlichen Schwerpunkten aus dem Kernlernplan. Ein Schwerpunkt des Inhaltsfeldes *Daten und ihre Strukturierung* ist die Arbeit mit Bibliotheken. Zusätzlich wird so die Kommunikation mit einem Informatiksystem, nämlich dem Arduino, verdeutlicht. Da die Schülerinnen und Schüler einen Befehl per Knopfdruck an die Schnittstelle und damit an den Arduino weitergeben. Der konkrete Kontext ist die Gestaltung einer Benutzeroberfläche zur Steuerung von Modelleisenbahnen. Diese würde sich jedoch auch auf automatisierte oder ferngesteuerte Züge übertragen lassen. Die Schülerinnen und Schüler müssen überlegen welche Möglichkeiten zur Steuerung einer Modelleisenbahn, beziehungsweise in Erweiterung, einer Eisenbahn nötig sind.

### 4.4.2. Zwei getrennte Strecken

Dieses Kapitel soll beispielhaft zeigen, dass die Umsetzung innerhalb von Greenfoot viele Möglichkeiten bietet. Da hier mit Greenfoot, also einer Java Umgebung gearbeitet werden soll, wird nicht weiter auf die nötigen Veränderungen innerhalb des Gleiserkennungsskriptes eingegangen. Das Skript befindet sich im Anhang unter H.4. Innerhalb von Greenfoot können die Schülerinnen und Schüler je nach real vorhandener Vorlage eines Schienenlayouts das Layout anpassen oder gar erstellen. Das zugehörige Layout muss in

JMRI existieren um auf die Blöcke verweisen zu können. In diesem Beispiel wurden zwei Quadrate als Layout gewählt. Die Schülerinnen und Schüler müssen eine Möglichkeit für die Steuerung der einzelnen Layouts beziehungsweise Lokomotiven finden. Dafür kann die Umsetzung von Kapitel 4.4.1 genutzt werden. Die Funktionen für die Zugerkennung funktionieren weiterhin.
Auch hier wäre der Kontext der Gestaltung einer Benutzeroberfläche für die Eisenbahnsteuerung eine Möglichkeit verschiedene Schwerpunkte des Inhaltsfelds *Daten und ihre Strukturierung* zu überprüfen. Durch die Erstellung der Gleisabschnitte und Gleisblöcke müssen die Objekte analysiert und erstellt werden.

# 5. Exemplarische Unterrichtssequenz

Es wird die Unterrichtssequenz beschrieben, die als Beispiel für eine Umsetzung der in Kapitel 4 genannten Möglichkeiten den Arduino einzusetzen, im Anhang aufgezeigt ist. In dieser Unterrichtseinheit wird Greenfoot nicht eingesetzt, sondern es wird sich auf den Arduino als Medium konzentriert. Da die Arduino IDE und Greenfoot mit unterschiedlichen Programmiersprachen arbeiten, sind die beiden Programme eher Alternativen als Ergänzungen für den Einsatz im Informatikunterricht zu sehen. Die Unterrichtseinheiten wurden anhand von Meyers Schema entwickelt und sind eingeteilt in "Einstieg - Arbeitsphase - Ergebnissicherung"[62]. Außerdem orientiert sich der Entwurf an dem 4 Phasenschema aus dem Chemie im Kontext Projekt, welches Koubek auch für die Entwicklung von kontextorientiertem Informatikunterricht vorschlägt.[63] Eine wie von Diethelm geforderte fünfte Phase der Rekontextualisierung[64] kann in einer drei mal 45-minütigen Unterrichtssequenz nicht erfolgen, sondern ist eher für eine längere Reihe von Unterrichtseinheiten gedacht. Die Unterrichtssequenz ist bewusst kurz gehalten und kann daher eher als Einstieg in den Kontext Eisenbahn gesehen werden, in dem die Weichenschaltung vorgestellt wird. Die Sequenz ist in zwei Abschnitte geteilt, von denen einer zwei Unterrichtsstunden und einer eine Unterrichtsstunde à 45 Minuten umfasst. Dieses entspricht der vorgegebenen Unterrichtsdauer pro Woche, die in der Qualifikationsphase 1 in NRW für den Informatikunterricht vorgesehen ist.

Der erste Abschnitt der Sequenz führt in den Kontext Eisenbahn, mit einer geschichtlichen Entwicklung der Stellwerke in Deutschland ein. Da der Unterricht für die Qualifikationsphase gedacht ist, ist es möglich, dass der Kontext Eisenbahn für die Schülerinnen und Schüler nicht neu ist. Möglicherweise vorhandenes Vorwissen kann hier reaktiviert werden und die Schülerinnen und Schüler können darauf zurückgreifen. Im Umgang mit dem Arduino und der Arduino IDE sowie dem Aufbau eines Arduino Programms sollten die Schülerinnen und Schüler bereits Erfahrung haben, ansonsten ist für die Umsetzung der Schaltung mehr Hilfestellung nötig und die einzelnen Komponenten, welche zu benutzen sind müssten genauer erläutert werden. Dies würde den zeitlichen Rahmen einer Unterrichtswoche allerdings überschreiten. In der Erarbeitungsphase wird in Gruppenarbeit an

[62] Meyer, „Unterricht analysieren, planen und auswerten".

[63] Koubek *u. a.,* „Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht", S. 276.

[64] Diethelm, Koubek & Witten, „IniK - Informatik im Kontext - Entwicklungen, Merkmale und Perspektiven", S. 103.

den Aufgaben gearbeitet. Die Gruppen können sich die Arbeit aufteilen, müssen dies aber nicht tun. Falls eine Aufteilung erfolgt muss innerhalb der Gruppe kommuniziert werden, da das Programm und die zugehörige Schaltung ineinandergreifen und aufeinander abgestimmt werden müssen. Dies fördert den Kompetenzbereich Kommunizieren und Kooperieren. Als Inhaltsfelder sind vor allem Algorithmen und Informatiksysteme zu nennen, aber auch Teile aus den anderen Inhaltsfeldern werden bei den Aufgaben abgefragt. In der Ergebnissicherung werden die erarbeiteten Modelle vorgeführt und dabei explizit das Inhaltsfeld Informatik, Mensch und Gesellschaft eingebracht, in dem die Sicherheitsrisikos einer automatisierten Weichenschaltung betrachtet werden.
Die zweite Einheit baut auf den Ergebnissen der ersten Einheit auf und entwickelt diese weiter. Dafür wird in der Einführung an die vorangegangene Stunde erinnert und das Wissen der Schüler reaktiviert. Eine weitere reale Umsetzung der Stellwerke wird vorgestellt, welche die Schülerinnen und Schüler aus ihren Modellen ableiten sollen. Auch hierbei wird wieder hauptsächlich das Inhaltsfeld Algorithmen, durch die Modifizierung des alten Codes behandelt. Die Arbeitsphase findet hier aufgrund der geringeren Arbeit und der dafür nötigen Kommunikation in der Paarprogrammierung statt. Das Ziel der Unterrichtseinheit und auch der gesamten Sequenz ist die Erarbeitung einer Weichenschaltung, die sich über JMRI mithilfe der Arduino-DCC-Zentrale steuern lässt. Damit soll die Weiterarbeit im Kontext ermöglicht und die Arduino-DCC-Zentrale vorgestellt werden, die in zukünftigen Sequenzen genauer betrachtet werden kann. Die abschließende Sicherungsphase der Einheit in zusammengestellten Gruppen sichert die Beteiligung aller Schülerinnen und Schüler, da diese in den Gruppen eingebunden und für die Sicherung ihrer Arbeitsergebnisse, anders wie in einer im Plenum durch die Lehrkraft moderierte Sicherungsphase, selbst verantwortlich sind. Die Lehrkraft sollte den Programmcode der Schüler überprüfen, um sicherzustellen, dass die Stromzufuhr zu den Weichen nicht per Dauerstrom, sondern durch kurze Impulse gewährleistet wird. Die darauf folgende Vorstellung der Ergebnisse erfolgt im Plenum, da nur eine Modelleisenbahn zur Verfügung steht und der Motor Driver an den Strom angeschlossen werden muss. Zwar besteht bei einer Stromleistung von ungefähr 20 Watt kein Verletzungsrisiko für die Schülerinnen und Schüler, jedoch neigen die meisten Netzteile zu einem sofortigen Defekt bei einem Kurzschluss.

# 6. ERGEBNISSE/FAZIT

Der Einsatz von einem Arduino im Zusammenhang mit JMRI und Greenfoot kann zu interessanten Ansätzen im Unterricht führen. Zudem kann die ”Aufweichung” des Faches Informatik von einem reinen computerzentrierten, theoretischen Fach wie es in den Kernlernplänen dargestellt wird, hin zu einer zusätzlichen technischen Komponente und dem Physical Computing mit dem Arduino ein umfassenderes Bild der Informatik geben.[65] Unter Zuhilfenahme des Arduinos lassen sich verschiedenste Schaltungen bauen um das Modelleisenbahnzubehör zu steuern. Der Arduino eignet sich hervorragend für die Umsetzung von eingebetteten Systemen in der Schule, um nicht nur theoretisch sondern auch praktisch Inhalte zu vermitteln. Programmiert wird in der Arduino IDE mit einer C/C++ -artigen Sprache. Die Funktionen und Bibliotheken sind alle sehr gut kommentiert und beschrieben, sodass Schülerinnen und Schüler sich selbst mit den Funktionen vertraut machen können und nach eigenen Vorstellungen Schaltungen für die Modelleisenbahn bauen können. Zusätzlich ermöglicht der Arduino mithilfe von DCC++ die Steuerung der Modelleisenbahn als DCC-Zentrale. Der Programmcode dafür ist zwar recht umfangreich, jedoch eignen sich trotzdem einige Klassen des Programms für eine Analyse im Informatikunterricht und DCC++ kann einen tieferen Einblick darin geben wie moderne Modelleisenbahnen funktionieren. JMRI wird in diesem Fall für die Steuerung der Lokomotiven für den kontextualisierten Informatikunterricht benötigt. Der Aufbau des Programms selber ist für Einsteiger beim Thema Modelleisenbahnen nicht besonders übersichtlich. Schülerinnen und Schüler müssten viel Zeit darin investieren das Programm zu verstehen, um auch nur einen Teil der umfangreichen Funktionen nutzen zu können. Der Programmcode ist enorm groß und für Programmieranfänger eher ungeeignet. Zwar bietet JMRI Möglichkeiten für den Informatikunterricht über die Ausführung von Skripten, welche durchaus seinen Reiz haben, die jedoch aufgrund des Fehlens eines ausgereiften Debuggers, wahrscheinlich nicht im Informatikunterricht brauchbar sind. Die Funktionen der Skripts werden allerdings zumindest für die Arbeit mit Greenfoot benötigt, da hierüber die Lok-Rückmeldungen und Besetztmeldungen der Gleise programmiert sind. Mit Greenfoot und der dazugehörigen Schnittstelle für JMRI lässt sich im Informatikunterricht arbeiten. JMRI kann im Hintergrund laufen, alle nötigen Eingaben können von den Lehrerinnen und Lehrern im voraus getätigt und in einer XML Datei gespeichert werden,

[65] Bergner & Schroeder, „Informatik Enlightened - Informatik (neu) beleuchtet dank Physical Computing mit Arduino“.

die den Schülerinnen und Schülern zur Verfügung gestellt wird. So müssen die Schülerinnen und Schüler keine Zeit in das Programm investieren. Da Greenfoot und die Arduino IDE auf Programmieranfänger zugeschnittene Programmierumgebungen sind, verfügen sie über einen Debugger und direkte Fehlerüberprüfung, die für Schülerinnen und Schüler geeignet ist.
Ein Problem ist, dass Greenfoot, JMRI und die Arduino IDE alle mit unterschiedlichen Programmiersprachen arbeiten. So können, gerade in der Schule, wahrscheinlich nicht alle vorgestellten Möglichkeiten zugleich genutzt werden. Es bietet aber auch jede der einzelnen Komponenten eine Vielzahl von Umsetzungsmöglichkeiten für den Informatikunterricht. Die für den Informatikunterricht vermutlich interessanteren Themen sind Greenfoot und die Arduino IDE, da sie aufgrund des vorhandenen Debuggers und der Programmierumgebung an sich sowie der Entwicklung eigens für den Einsatz in einer Lernumgebung, einen einfacheren und frustfreieren Lernvorgang ermöglichen. Die Lehrkräfte müssen für die Umsetzung der einzelnen Abschnitte viele Einstellungen innerhalb von JMRI vornehmen, was zwar gut dokumentiert ist, jedoch trotzdem eine gewisse Einarbeitung benötigt. Für die in dieser Arbeit behandelten Themen sind jeweils hilfreiche Anleitung im Anhang. Ebenso sind alle genutzten Skripts, Greenfoot Projekte und JMRI Einstellungen unter https://github.com/LaBo2709/ArduinoundGreenfoot zu finden. Diese können als Vorlage für die eigenen Layouts benutzt oder nachgebaut werden. Die exemplarische Unterrichtssequenz dient als Beispiel für eine mögliche Umsetzung der Ergebnisse dieser Bachelorarbeit. Da gerade die Arbeit mit dem Arduino eine gewisse Einarbeitung benötigt und die zu benutzbaren Komponenten um Schaltungen zu bauen sehr zahlreich sind, bietet sich eine solche Unterrichtssequenz mit dem Arduino nur an, wenn bereits vorher mit ihm gearbeitet wurde. Ansonsten muss der Zeitrahmen der Unterrichtseinheiten neu festgelegt werden.

# 7. Zusammenfassung/Ausblick

In dieser Bachelorarbeit wurden verschiedene Möglichkeiten untersucht den Kontext Modelleisenbahnen, mithilfe eines Arduinos als DCC-Zentrale sowie der JMRI - Greenfoot Schnittstelle im Informatikunterricht zu nutzen. Der Kontext bietet große Möglichkeiten auch längerfristige Projekte innerhalb des Informatikunterrichts zu begleiten. Die vorgeschlagenen Methoden sind rein theoretischer Natur und wurden nicht in einer echten Unterrichtssituation getestet. Je nach gewählter Programmiersprache können die unterschiedlichen Bestandteile der Bachelorarbeit genutzt werden. Es sind viele weitere Alternativen zum Ausbau der Funktionsweise der vorgestellten Möglichkeiten denkbar. Beispielsweise könnten anstatt der Logik, die die Zugerkennung anhand des alten Standorts berechnet und damit fehleranfällig ist, falls Sensoren falsch ausgelöst oder gar nicht ausgelöst werden durch RFID Scanner ersetzt werden, die in Lokomotiven verbaute RFID Tags ausliest, die vorher beschrieben wurden. Da diese Tags allerdings eine gewisse Größe haben empfiehlt es sich hier mit einer Spurgröße der Lokomotive von H0 zu arbeiten. Die Arbeit mit dem Arduino im Informatikunterricht bietet sich ansonsten auch außerhalb des Kontextes Eisenbahn an. Er ist vielseitig einsetzbar und kann die Schülerinnen und Schüler auch außerhalb des Informatikunterrichts anregen sich mit Informatik zu beschäftigen. Denkbar wären etwa eine AG in der sie mit den Arduino entweder an der Modelleisenbahn, Robotern oder anderen steuerbaren Geräten arbeiten. Er ermöglicht ein spielerisches und praktisches Lernen in einem Fach, welches ansonsten viele rein fachliche, theoretische Inhalte vermittelt und für viele Schülerinnen und Schüler keinen Reiz hat.

# 8. Literaturverzeichnis

## Literatur


1. Nawrath, D. *Kontextorientierung Rekonstruktion einer fachdidaktischen Konzeption für den Physikunterricht* Diss. (Carl von Ossietzky Universität Oldenburg, Feb. 2010), 61–62.
2. Bundesministerium für Bildung und Forschung. *Bundesministerium für Bildung und Forschung* abgerufen am 20.12.2021. `https://www.bmbf.de/bmbf/de/home/home_node.html`.
3. Ministerium für Schule und Bildung des Landes Nordrhein-Westfalen. Kernlehrplan für die Sekundarstufe I Gymnasium in Nordrhein-Westfalen Biologie, 10 (2019).
4. Koubek, J. *u. a. Informatik im Kontext(IniK)* abgerufen am 11.11.2021. `http://www.informatik-im-kontext.de/`.
5. Diethelm, I., Koubek, J. & Witten, H. IniK - Informatik im Kontext - Entwicklungen, Merkmale und Perspektiven. *LOG IN,* 97–105. ISSN: 0720-8642 (2011).
6. Thomas, M. *Eisenbahn im kontextorientierten Informatikunterricht* abgerufen am 19.11.2021. `https://www.uni-muenster.de/imperia/md/content/idmi/ag-thomas/publikationen/2021_eisenbahn_im_kontextorientierten_informatik_unterricht_-_juni_2021_-_marco_thomas.pdf`.
7. Koubek, J. *u. a. Informatik im Kontext (IniK) – Ein integratives Unterrichtskonzept für den Informatikunterricht* in *Zukunft braucht Herkunft – 25 Jahre »INFOS – Informatik und Schule«* (Hrsg. Koerber, B.) (Gesellschaft für Informatik e.V., Bonn, 2009), 268–279.
8. Bayrhuber, H. *u. a.* Biologie im Kontext. *Mathematische und Naturwissenschaftliche Unterricht* **60,** 282–286 (2007).
9. Mikelskis-Seifert, S. & Duit, R. Physik im Kontext. *Mathematische und Naturwissenschaftliche Unterricht* **60,** 265–273 (2007).
10. Di Fuccia, D., Schellenbach-Zell, J. & Ralle, B. Chemie im Kontext. *MATHEMATISCHE UND NATURWISSENSCHAFTLICHE UNTERRICHT* **60,** 274–282 (2007).
11. Smith, G. & Matthews, P. Science, technology and society in transition year: A pilot study. *Irish Educational Studies* **19,** 107–119 (2000).

12. Parchmann, I. *u. a.* “Chemie im Kontext”: A symbiotic implementation of a context-based teaching and learning approach. *International journal of science education* **28,** 1041–1062 (2006).

13. Höttecke, D. *Naturwissenschaftlicher Unterricht im internationalen Vergleich* 81–97 (LIT Verlag Münster, 2007).

14. Hirschfeld, R., Costanza, P. & Nierstrasz, O. M. Context-oriented programming. *Journal of Object technology* **7,** 125–151 (2008).

15. Salvaneschi, G., Ghezzi, C. & Pradella, M. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software* **85,** 1801–1817. ISSN: 0164-1212. https://www.sciencedirect.com/science/article/pii/S016412121200074X (2012).

16. Koubek, J. *Kriterien für die Auswahl von Kontexten* abgerufen am 13.11.2021. https://medienwissenschaft.uni-bayreuth.de/inik/konzepte/kriterienkontexte/.

17. Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen. *Marktuntersuchung Eisenbahnen 2021* abgerufen am 13.11.2021. https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Eisenbahn/Unternehmen_Institutionen/Veroeffentlichungen/Marktuntersuchunge
MarktuntersuchungEisenbahnen/MarktuntersuchungEisenbahnen2021Sondererhebun
pdf?__blob=publicationFile&v=3.

18. Mahnke, F. *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, 2021). https://www.uni-muenster.de/imperia/md/content/idmi/ag-thomas/publikationen/2021_mahnke_eisenbahn_datenbank_masterarbeit.pdf.

19. Bienbeck, L. *Konzeptionierung einer Design-Based Research Studie zum Einsatz von Computern im Informatikunterricht der Einführungsphase zum Thema Sortierverfahren im Kontext Eisenbahn* Masterarbeit (Westfälische Wilhelms-Universität Münster, Institut für Didaktik der Mathematik und der Informatik Arbeitsbereich Didaktik der Informatik, 2021). https://www.uni-muenster.de/imperia/

md/content/idmi/ag-thomas/publikationen/2021_bienbeck_eisenbahn_sortieren.pdf.

20. Allianz pro Schiene. *Trend: Investitionen in die Schieneninfrastruktur in Deutschland von 2009-2020* abgerufen am 13.11.2021. https://www.allianz-pro-schiene.de/themen/infrastruktur/investitionen/.

21. Java Model Railroad Interface. *About JMRI* abgerufen am 17.11.2021. https://www.jmri.org/.

22. Rocrail. *Rocrail - Innovative Software zur Steuerung von Modelleisenbahnen* abgerufen am 03.12.2021. https://wiki.rocrail.net/doku.php?id=start-de.

23. King's College London. *About Greenfoot* abgerufen am 16.11.2021. https://www.greenfoot.org/overview.

24. Kölling, M. *Greenfoot: a highly graphical ide for learning object-oriented programming.* in *ITiCSE* (Hrsg. Amillo, J. *u. a.*) (ACM, 2008), 327. ISBN: 978-1-60558-078-4. http://dblp.uni-trier.de/db/conf/iticse/iticse2008.html#Kolling08.

25. *Lernen am Emsland Informaitk* abgerufen am 16.11.2021. https://emsland-gymnasium-rheine.de/lernen-am-emsland/fachunterricht/informatik/.

26. *Gesamtschule Bad Lippspringe Informatikunterricht* abgerufen am 16.11.2021. https://www.gesamtschule-bad-lippspringe.de/unterricht/f\%C3\%A4cher/informatik/.

27. Grimm, R. *u. a. Informatik - Lehrwerk für die gymnasiale Oberstufe* (Schöningh, 2014).

28. Arduino. *Arduino Uno Rev3* abgerufen am 12.11.2021. https://store.arduino.cc/collections/boards/products/arduino-uno-rev3.

29. Berman, G. *DCC++ Basestation* abgerufen am 16.11.2021. https://github.com/DccPlusPlus.

30. Arduino. *Arduino Motor Shield Rev3* abgerufen am 15.11.2021. https://store.arduino.cc/collections/shields/products/arduino-motor-shield-rev3.

31. National Model Railroad Association. *Communications Standards For Digital Command Control, All Scales* abgerufen am 16.11.2021. https://www.nmra.org/sites/default/files/s-92-2004-07.pdf.

32. Verband der Modelleisenbahner und Eisenbahnfreunde Europas. *Digitales Steuersignal DCC Bitdarstellung* abgerufen am 16.11.2021. https://www.morop.org/downloads/nem/de/nem670_d.pdf.

33. Verband der Hersteller Digitaler Modellbahnprodukte e.V. *RCN-211 DCC-Protokoll Paketstruktur und Adressbereiche* abgerufen am 16.11.2021. http://normen.railcommunity.de/RCN-211.pdf.

34. Verband der Modelleisenbahner und Eisenbahnfreunde Europas. *Digitales Steuersignal DCC Basis-Datenpakete* abgerufen am 16.11.2021. https://www.morop.org/downloads/nem/de/nem671_d.pdf.

35. Verband der Hersteller Digitaler Modellbahnprodukte e.V. *RCN-210 DCC-Protokoll Bit-Übertragung* abgerufen am 16.11.2021. http://normen.railcommunity.de/RCN-210.pdf.

36. PIKO. *PIKO SmartDecoder 4.1* abgerufen am 16.11.2021. https://www.piko-shop.de/is.php?id=22683.

37. Atmel. *ATmega328P-Datenblatt* abgerufen am 15.12.2021. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

38. Arduino. *Arduino Mega 2560 Rev3* abgerufen am 30.11.2021. https://store.arduino.cc/products/arduino-mega-2560-rev3.

39. Keyestudio. *Keyestudio MEGA Sensor Shield V1* abgerufen am 30.11.2021. https://www.keyestudio.com/products/free-shipping-keyestudio-mega-sensor-shield-v1-for-arduino-mega.

40. Geoff. *SMA28 JMRI Sensor Channels – Direct Arduino to JMRI Communications - Simple Support for Lots of Detectors* abgerufen am 30.11.2021. 2018. https://model-railroad-hobbyist.com/node/34392.

41. Java Model Railroad Interface. *JMRI:Scripting* abgerufen am 30.11.2021. https://www.jmri.org/help/en/html/tools/scripting/index.shtml.

42. Ministerium für Schule und Weiterbildung des Landes Nordrhein-Westfalen. Kernlehrplan für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen Informatik (2014).

43. Schubert, S. & Schwill, A. *Didaktik der Informatik* ISBN: 978-3-8274-2653-6. `https://doi.org/10.1007/978-3-8274-2653-6_8` (Spektrum Akademischer Verlag, Heidelberg, 2011).

44. Bergner, N. & Schroeder, U. *Informatik Enlightened - Informatik (neu) beleuchtet dank Physical Computing mit Arduino* in *Informatik allgemeinbildend begreifen* (Hrsg. Gallenbacher, J.) (Gesellschaft für Informatik e.V., Bonn, 2015), 43–52.

45. Grandell, L. *u. a. Why complicate things? Introducing programming in high school using Python* in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (2006), 71–80.

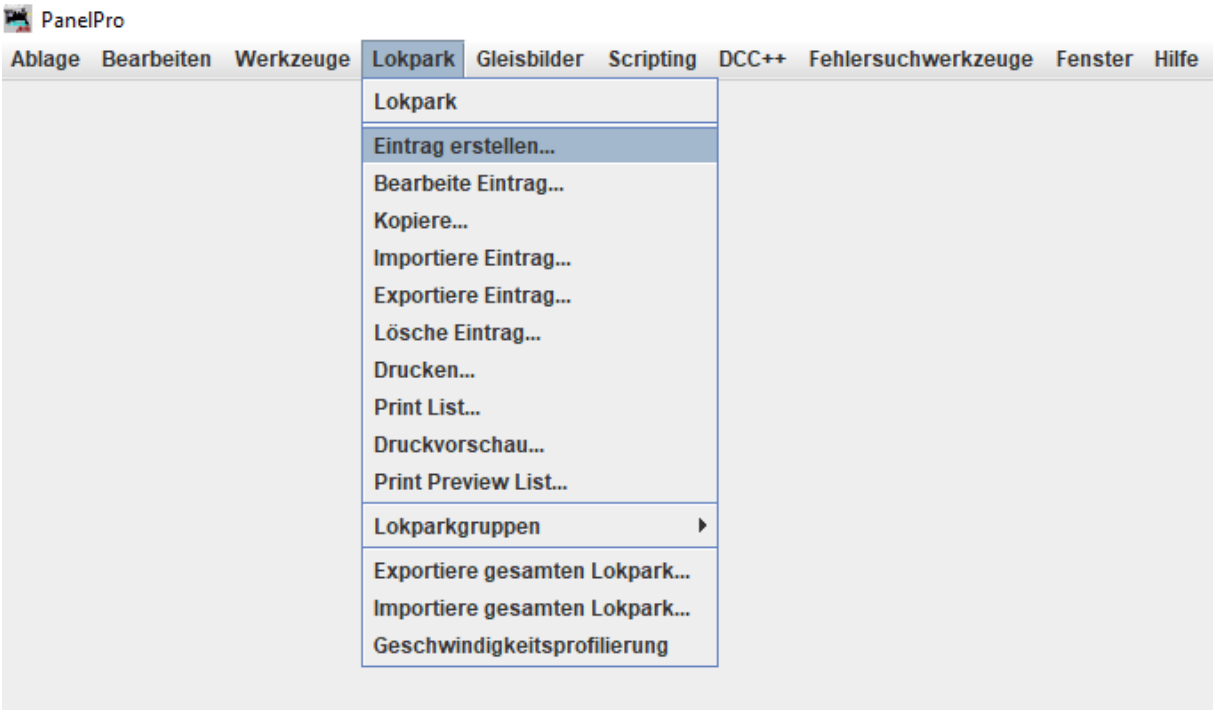
46. Michaeli, T. *Debugging im Informatikunterricht* Diss. (2021).

47. Meyer, H. Unterricht analysieren, planen und auswerten. *Einführung in die Schulpädagogik. Berlin: Cornelsen Scriptor,* 147–156 (2002).

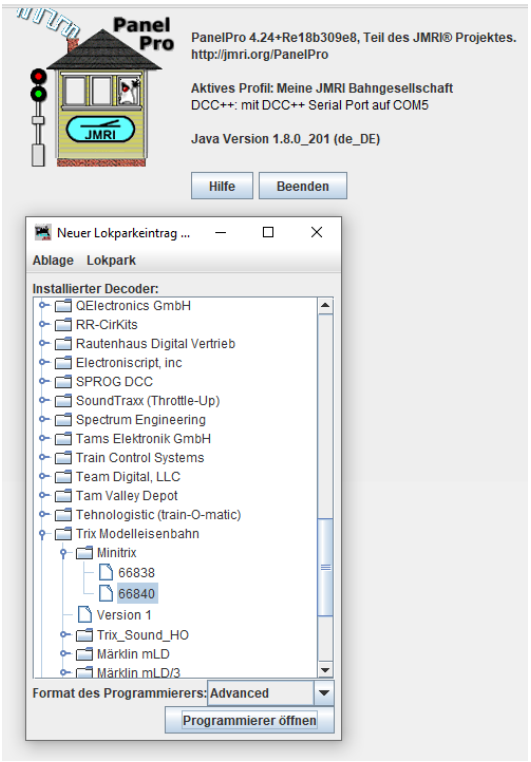
# A. JMRI

## A.1. Anleitung Eisenbahn hinzufügen

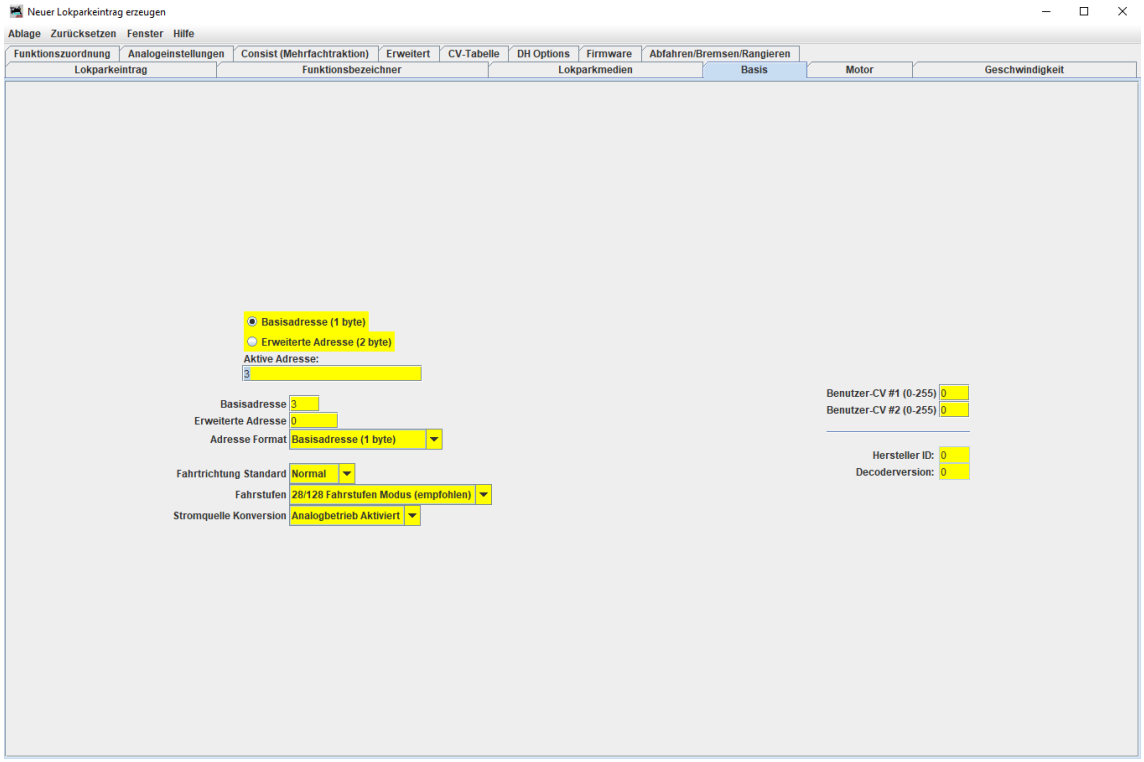
1.Unter Lokpark ”Eintrag erstellen”



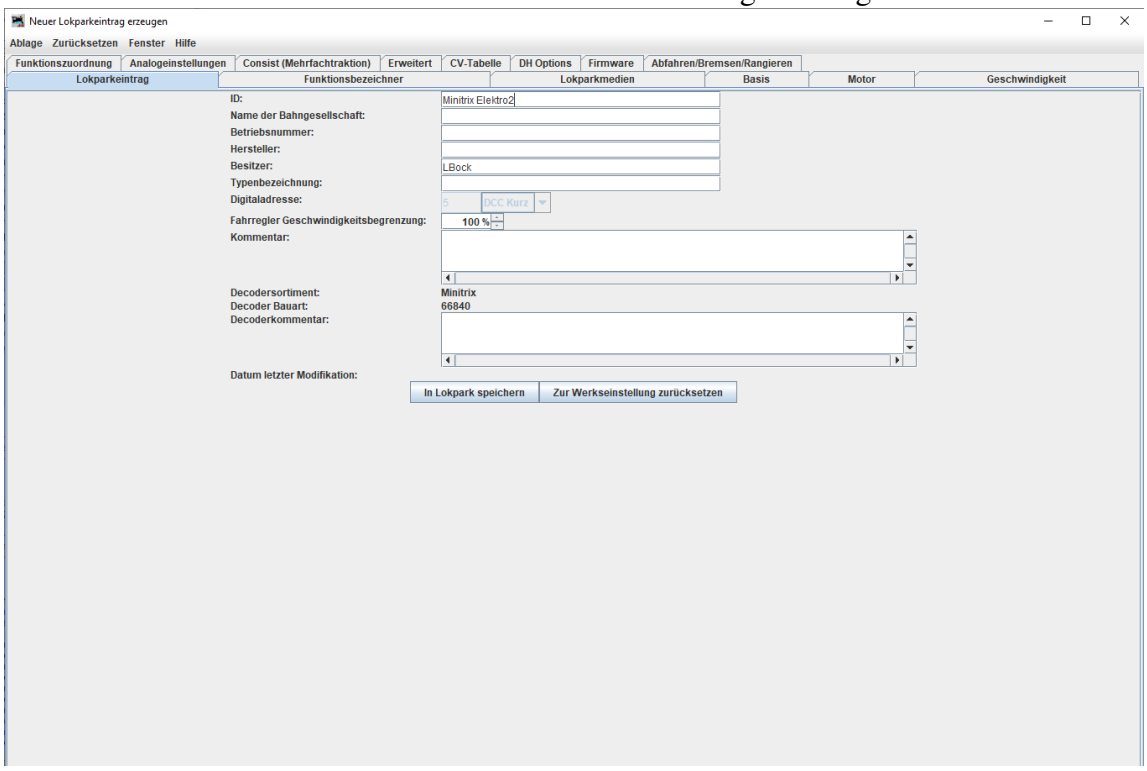

2.Hersteller und Lokdecoder auswählen



3.Unter Basis die aktive DCC Adresse der Lok eingeben. Sollte bei Auslieferung der Lok in der Bedienungsanleitung notiert sein. Ist nicht frei Wählbar und muss eindeutig vergeben sein.




4.Unter Lokparkeintrag eine ID(Namen) vergeben. Ist nicht frei Wählbar und muss eindeutig vergeben sein.



## A.2. Gleisbilder erstellen

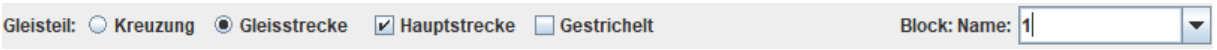
Gleisbilder können unter dem Reiter Gleisbilder ->Anlageeditor erstellt werden.


Um Objekte hinzuzufügen werden diese ausgewählt und mit Shift+Click auf das Gitternetz platziert. Um dabei Weichen, Blöcke, Sensoren, etc. später steuern zu können müssen die entsprechenden Objekte vorher unter Werkzeuge -> Tabellen erstellt werden. Ein Beispiel zur Erstellung von Objekten ist in E.2. Alle Objekte sind in gleicher Weise einzurichten.

Die erstellten Objekte können vor dem platzieren zugewiesen werden indem aus den vorgegeben Objekten gewählt wird.

Weichen:


Gleisblöcke:



# B. Weichenschaltungen

## B.1. Weichenschaltung per Knopf



```
int weiche = 12; // hier wird die Polarität des Schalters gewechselt
int dcmotor = 9; // Teil des Dc Motor Programms
int power = 3; // Strom für die Weiche
int button =A0; //input Pin für den Knopf auf dem Arduino


void setup() {
  pinMode(weiche, OUTPUT); // setzen den PIN 12 als OUTPUT
  pinMode(dcmotor, OUTPUT); // setzen den PIN 9 als OUTPUT
  pinMode(power, OUTPUT); // setzen den PIN 3 als OUTPUT
  pinMode(button, INPUT); // PIN A0 als INPUT
}

// Enum mit den Zuständen
enum SWITCHSTATES
{
  SS_AUS1, // SwitchState aus 1
  SS_AUS2, // SwitchState aus 2
  SS_GERADEAUS, // SwitchState GERADEAUS
  SS_KURVE, // SwitchState KURVE
};

SWITCHSTATES switchState=SS_AUS1; // Standard auf SS_AUS1 setzen

void loop(){
  int knopf_auslesen = analogRead(button); // Der Input A0 für den Knopf wird ausgelesen
  delay(100); //eine Wartezeit von 0.1 Sekunde
// Switch case zum Durchschalten des Enums
  switch(switchState){
    case SS_AUS1:
    ausschalten1(knopf_auslesen);
    break;
    case SS_AUS2:
    ausschalten2(knopf_auslesen);
    break;
    case SS_GERADEAUS:
    geradeaus(knopf_auslesen);
    break;
    case SS_KURVE:
    kurve(knopf_auslesen);
    break;
  }
}

//Ist der Grundzustand. Wenn hier der Knopf betätigt wird, wird der Kurvenzustand aufgerufen
void ausschalten1(int knopf_auslesen){
  digitalWrite(weiche,LOW); // Polarität
  digitalWrite(dcmotor,LOW); // Motor Freigeben
  digitalWrite(power,0); // Strom auf 0 setzen

  if (knopf_auslesen>500){
    switchState=SS_KURVE; //wenn der Knopf gedrückt wird Schalte die Weiche auf KURVE
  }
}

//Schaltet die Weiche auf Kurve und geht in den zweiten Auszustand
void kurve(int knopf_auslesen){
  digitalWrite(weiche,HIGH); // Polarität
  digitalWrite(dcmotor,LOW); // Motor Freigeben
  digitalWrite(power,255); // Strom auf Maximum setzen
  delay(200);
  digitalWrite(weiche,HIGH); // Polarität
  digitalWrite(dcmotor,LOW); // Motor Freigeben
  digitalWrite(power,0); // Strom auf 0 setzen

  switchState=SS_AUS2;
}

//Ist der zweite Grundzustand. Wenn hier der Knopf betätigt wird, wird der Geradeauszustand aufgerufen
void ausschalten2(int knopf_auslesen){
  digitalWrite(weiche,HIGH); // Polarität
```

```
    digitalWrite(dcmotor,LOW); // Motor Freigeben
    digitalWrite(power,0); // Strom auf 0 setzen

    if (knopf_auslesen>500){
      switchState=SS_GERADEAUS; //wenn der Knopf gedrückt wird Schalte die Weiche auf KURVE
    }
  }

  //Schaltet die Weiche auf geradeaus und geht in den zweiten Auszustand
  void geradeaus(int knopf_auslesen){
    digitalWrite(weiche,LOW); // Polarität
    digitalWrite(dcmotor,LOW); // Motor Freigeben
    digitalWrite(power,255); // Strom auf Maximum setzen
    delay(200);
    digitalWrite(weiche,LOW); // Polarität
    digitalWrite(dcmotor,LOW); // Motor Freigeben
    digitalWrite(power,0); // Strom auf Minimum setzen

    switchState=SS_AUS1;
  }
```

## B.2. Weichenschaltung per Knopf mit Lampe

Der Programmcode für die Weichenschaltung über einen Knopf mit einer Lampe als Rückmeldung.



```
int weiche = 12; // hier wird die Polarität des Schalters gewechselt
int dcmotor = 9; // Teil des Dc Motor Programms
int power = 3; // Strom für die Weiche
int button =A0; //input Pin für den Knopf auf dem Arduino
int Green=6; // Grüne LED wird von Digitalem Pin 6 Gesteuert
int Red=7; // Rote LED wird von Digitalem Pin 6 Gesteuert


void setup() {
  pinMode(weiche, OUTPUT); // setzen den PIN 12 als OUTPUT
  pinMode(dcmotor, OUTPUT); // setzen den PIN 9 als OUTPUT
  pinMode(power, OUTPUT); // setzen den PIN 3 als OUTPUT
  pinMode(button, INPUT); // PIN A0 als INPUT
  pinMode(Green, OUTPUT); // PIN 6 als Output
  pinMode(Red, OUTPUT); // PIN 7 als Output
}

// Enum mit den Zuständen
enum SWITCHSTATES
{
  SS_AUS1, // SwitchState aus 1
  SS_AUS2, // SwitchState aus 2
  SS_GERADEAUS, // SwitchState GERADEAUS
  SS_KURVE, // SwitchState KURVE
};

SWITCHSTATES switchState=SS_AUS1; // Standard auf SS_AUS1 setzen

void loop(){
  int knopf_auslesen = analogRead(button); // Der Input A0 für den Knopf wird ausgelesen
  delay(100); //eine Wartezeit von 0.1 Sekunde
// Switch case zum Durchschalten des Enums
  switch(switchState){
    case SS_AUS1:
    ausschalten1(knopf_auslesen);
    break;
    case SS_AUS2:
    ausschalten2(knopf_auslesen);
    break;
    case SS_GERADEAUS:
    geradeaus(knopf_auslesen);
    break;
    case SS_KURVE:
    kurve(knopf_auslesen);
    break;
  }
}
//Ist der Grundzustand. Wenn hier der Knopf betätigt wird, wird der Kurvenzustand aufgerufen
void ausschalten1(int knopf_auslesen){
  digitalWrite(weiche,LOW); // Polarität
  digitalWrite(dcmotor,LOW); // Motor Freigeben
  digitalWrite(power,0); // Strom auf 0 setzen
  digitalWrite(Green,LOW); // Schaltet Grüne LED an
  digitalWrite(Red, HIGH); //Schaltet Rote LED an
  // High und Low getauscht wegen gemeinsamen + und gesteuertem -
  if (knopf_auslesen>500){
    switchState=SS_KURVE; //wenn der Knopf gedrückt wird Schalte die Weiche auf KURVE
  }
}
//Schaltet die Weiche auf Kurve und geht in den zweiten Auszustand
void kurve(int knopf_auslesen){
  digitalWrite(weiche,HIGH); // Polarität
  digitalWrite(dcmotor,LOW); // Motor Freigeben
  digitalWrite(power,255); // Strom auf Maximum setzen
  digitalWrite(Green, HIGH); // Schaltet grüne LED aus
  digitalWrite(Red, HIGH);//Schaltet Rote LED an
  delay(200);
  digitalWrite(weiche,HIGH); // Polarität
  digitalWrite(dcmotor,LOW); // Motor Freigeben
  digitalWrite(power,0); // Strom auf Minimum setzen
  digitalWrite(Green, HIGH); // Schaltet grüne LED aus
```

```
    digitalWrite(Red, HIGH);//Schaltet Rote LED aus
    switchState=SS_AUS2;
  }

  //Ist der zweite Grundzustand. Wenn hier der Knopf betätigt wird, wird der Geradeauszustand aufgerufen
  void ausschalten2(int knopf_auslesen){
    digitalWrite(weiche,HIGH); // Polarität
    digitalWrite(dcmotor,LOW); // Motor Freigeben
    digitalWrite(power,0); // Strom auf 0 setzen
    digitalWrite(Green,HIGH); // Schaltet Grüne LED aus
    digitalWrite(Red, LOW); //Schaltet Rote LED an
    if (knopf_auslesen>500){
      switchState=SS_GERADEAUS; //wenn der Knopf gedrückt wird Schalte die Weiche auf KURVE
    }
  }

  //Schaltet die Weiche auf geradeaus und geht in den zweiten Auszustand
  void geradeaus(int knopf_auslesen){
    digitalWrite(weiche,LOW); // Polarität
    digitalWrite(dcmotor,LOW); // Motor Freigeben
    digitalWrite(power,255); // Strom auf Maximum setzen
    digitalWrite(Green, HIGH); // Schaltet grüne LED aus
    digitalWrite(Red, HIGH);//Schaltet Rote LED aus
    delay(200);
    digitalWrite(weiche,LOW); // Polarität
    digitalWrite(dcmotor,LOW); // Motor Freigeben
    digitalWrite(power,0); // Strom auf Minimum setzen
    digitalWrite(Green, HIGH); // Schaltet grüne LED aus
    digitalWrite(Red, HIGH);//Schaltet Rote LED aus

    switchState=SS_AUS1;
  }
```

## B.3. Weichenschaltung mit zwei Knöpfen

Der Programmcode für die Weichenschaltung über 2 Knöpfe .



```
//Quelle:https://drive.google.com/file/d/1038dZtfMV2y0Y41BSIceeRDO31unvA_s/view wurde als Vorlage benutzt
const int switch1pin1=2; //Motor Driver ist über Pins 2-5 Verbunden. Pin 2 und 3 kontrollieren 1. Weiche
const int switch1pin2=3;
const int switch2pin1=4; // Pins 4 und 5 kontrollieren 2. Weiche
const int switch2pin2=5;
// weiterer Motor Driver an Pins 6-9 anschließen
//const int switch3pin1=6; Für Weiche 3 Pin 6 und 7
//const int switch3pin2=7;
//const int switch4pin1=8; Für Weiche 4 Pin 8 und 9
//const int switch4pin2=9;

const int button1 = A0; // Knopf 1 ist an A0 Angeschlossen und wird Weiche 1 schalten
const int button2 = A1; // Knopf 2 ist an A1 Angeschlossen und wird Weiche 2 schalten
// Für weitere Weichen werden Knöpfe an A2-A5 angeschlossen
// const int button3 = A2; // Knopf 2 ist an A1 Angeschlossen und wird Weiche 1 schalten

int state1=0; // setzt den Aktuellen Status von Knopf1
int state2=0; // setzt den Aktuellen Status von Knopf2
void setup() {
  // Ausgabe Pins werden Festgelegt und der Serielle Monitor gestartet
Serial.begin(9600);// Setzt die Baud Rate auf 9600 und startet den Seriellen Monitor.
pinMode(switch1pin1,OUTPUT); //legt die Digital Pins 2-5 als Ausgabe Pins fest
pinMode(switch1pin2,OUTPUT);
pinMode(switch2pin1,OUTPUT);
pinMode(switch2pin2,OUTPUT);
//usw für weitere Motor Driver
}
// Enum über die Zustände
enum SWITCHSTATES{
  SS_AUS,
  SS_GERADE1,
  SS_KURVE1,
  SS_GERADE2,
  SS_KURVE2,
  //usw für weitere Weichen
};
// Startstatus wird gesetzt
SWITCHSTATES switchState = SS_AUS;
void loop() {
 Serial.print("Status 1:");
 Serial.println(state1); //gibt den Status der ersten Weiche im Seriellen Monitor aus
 Serial.print("Status 2:");
 Serial.println(state2); //gibt den Status der zweiten Weiche im Seriellen Monitor aus
 //usw für weitere Weichen

 int button1Lesen=analogRead(button1); //Speichert den Wert von button1 ab
 int button2Lesen=analogRead(button2); //Speichert den Wert von button2 ab
 Serial.print("Button 1: ");
  Serial.println(button1Lesen);//display in serial monitor what the State of Button 1 is with the text "Button 1" in front.
  Serial.print("Button 2: ");
  Serial.println(button2Lesen);//display in serial monitor what the State of Button 2 is with the text "Button 2" in front.
  delay(200);// 0.2 second pause between readings

 //Switch Case der über den Enum Schaltet und die Funktionen für die Weichenschaltung aufruft
 switch(switchState)
 {
  case SS_AUS:
  switchAus(button1Lesen, button2Lesen);
  break;
  case SS_GERADE1:
  switchGerade1(button1Lesen, button2Lesen);
   break;
  case SS_KURVE1:
  switchKurve1(button1Lesen, button2Lesen);
   break;
  case SS_GERADE2:
  switchGerade2(button1Lesen, button2Lesen);
   break;
  case SS_KURVE2:
  switchKurve2(button1Lesen, button2Lesen);
   break;
 }

}
//switchAus schaltet alle OUTPUTS auf LOW und ist der Ruhezustand
void switchAus(int button1Lesen, int button2Lesen){
  digitalWrite(switch1pin1,LOW);
  digitalWrite(switch1pin2,LOW);
  digitalWrite(switch2pin1,LOW);
  digitalWrite(switch2pin2,LOW);

  if((button1Lesen>1000) && (state1==1)){
    switchState=SS_GERADE1;
  }
  if((button1Lesen>1000) && (state1==0)){
```

```
//Quelle:https://drive.google.com/file/d/1038dZtfMV2y0Y41BSIceeRDO31unvA_s/view wurde als Vorlage benutzt
const int switch1pin1=2; //Motor Driver ist über Pins 2-5 Verbunden. Pin 2 und 3 kontrollieren 1. Weiche
const int switch1pin2=3;
const int switch2pin1=4; // Pins 4 und 5 kontrollieren 2. Weiche
const int switch2pin2=5;
// weiterer Motor Driver an Pins 6-9 anschließen
//const int switch3pin1=6; Für Weiche 3 Pin 6 und 7
//const int switch3pin2=7;
//const int switch4pin1=8; Für Weiche 4 Pin 8 und 9
//const int switch4pin2=9;

const int button1 = A0; // Knopf 1 ist an A0 Angeschlossen und wird Weiche 1 schalten
const int button2 = A1; // Knopf 2 ist an A1 Angeschlossen und wird Weiche 2 schalten
// Für weitere Weichen werden Knöpfe an A2-A5 angeschlossen
// const int button3 = A2; // Knopf 2 ist an A1 Angeschlossen und wird Weiche 1 schalten

int state1=0; // setzt den Aktuellen Status von Knopf1
int state2=0; // setzt den Aktuellen Status von Knopf2
void setup() {
  // Ausgabe Pins werden Festgelegt und der Serielle Monitor gestartet
Serial.begin(9600);// Setzt die Baud Rate auf 9600 und startet den Seriellen Monitor.
pinMode(switch1pin1,OUTPUT); //legt die Digital Pins 2-5 als Ausgabe Pins fest
pinMode(switch1pin2,OUTPUT);
pinMode(switch2pin1,OUTPUT);
pinMode(switch2pin2,OUTPUT);
//usw für weitere Motor Driver
}
// Enum über die Zustände
enum SWITCHSTATES{
  SS_AUS,
  SS_GERADE1,
  SS_KURVE1,
  SS_GERADE2,
  SS_KURVE2,
  //usw für weitere Weichen
};
// Startstatus wird gesetzt
SWITCHSTATES switchState = SS_AUS;
void loop() {
 Serial.print("Status 1:");
 Serial.println(state1); //gibt den Status der ersten Weiche im Seriellen Monitor aus
 Serial.print("Status 2:");
 Serial.println(state2); //gibt den Status der zweiten Weiche im Seriellen Monitor aus
 //usw für weitere Weichen

 int button1Lesen=analogRead(button1); //Speichert den Wert von button1 ab
 int button2Lesen=analogRead(button2); //Speichert den Wert von button2 ab
 Serial.print("Button 1: ");
  Serial.println(button1Lesen);//display in serial monitor what the State of Button 1 is with the text "Button 1" in front.
  Serial.print("Button 2: ");
  Serial.println(button2Lesen);//display in serial monitor what the State of Button 2 is with the text "Button 2" in front.
  delay(200);// 0.2 second pause between readings

 //Switch Case der über den Enum Schaltet und die Funktionen für die Weichenschaltung aufruft
 switch(switchState)
 {
  case SS_AUS:
  switchAus(button1Lesen, button2Lesen);
  break;
  case SS_GERADE1:
  switchGerade1(button1Lesen, button2Lesen);
   break;
  case SS_KURVE1:
  switchKurve1(button1Lesen, button2Lesen);
   break;
  case SS_GERADE2:
  switchGerade2(button1Lesen, button2Lesen);
   break;
  case SS_KURVE2:
  switchKurve2(button1Lesen, button2Lesen);
   break;
 }

}
//switchAus schaltet alle OUTPUTS auf LOW und ist der Ruhezustand
void switchAus(int button1Lesen, int button2Lesen){
  digitalWrite(switch1pin1,LOW);
  digitalWrite(switch1pin2,LOW);
  digitalWrite(switch2pin1,LOW);
  digitalWrite(switch2pin2,LOW);

  if((button1Lesen>1000) && (state1==1)){
    switchState=SS_GERADE1;
  }
  if((button1Lesen>1000) && (state1==0)){
```

## B.4. Weichenschaltung DCC

Der Programmcode für die Weichenschaltung über die Arduino DCC-Zentrale.



```
//Quelle:https://drive.google.com/file/d/1038dZtfMV2y0Y41BSIceeRDO31unvA_s/view wurde als Vorlage benutzt
const int switch1pin1=2; //Motor Driver ist über Pins 2-5 Verbunden. Pin 2 und 3 kontrollieren 1. Weiche
const int switch1pin2=3;
const int switch2pin1=4; // Pins 4 und 5 kontrollieren 2. Weiche
const int switch2pin2=5;
// weiterer Motor Driver an Pins 6-9 anschließen
//const int switch3pin1=6; Für Weiche 3 Pin 6 und 7
//const int switch3pin2=7;
//const int switch4pin1=8; Für Weiche 4 Pin 8 und 9
//const int switch4pin2=9;

const int input1 = A0; // Die DCC-Station ist mit Pin 2 an A0 Angeschlossen und wird Weiche 1 schalten
const int input2 = A1; // Die DCC-Station ist mit Pin 4 an A1 Angeschlossen und wird Weiche 2 schalten



int state1=0; // speichert den Aktuellen Status von Weiche 1
int state2=0; // speichert den Aktuellen Status von Weiche 2
void setup() {
  // Ausgabe Pins werden Festgelegt und der Serielle Monitor gestartet
Serial.begin(9600);// Setzt die Baud Rate auf 9600 und startet den Seriellen Monitor.
pinMode(switch1pin1,OUTPUT); //legt die Digital Pins 2-5 als Ausgabe Pins fest
pinMode(switch1pin2,OUTPUT);
pinMode(switch2pin1,OUTPUT);
pinMode(switch2pin2,OUTPUT);
//usw für weitere Motor Driver
}
// Enum über die Zustände
enum SWITCHSTATES{
  SS_AUS,
  SS_GERADE1,
  SS_KURVE1,
  SS_GERADE2,
  SS_KURVE2,
  //usw für weitere Weichen
};
//Startzustand wird gesetzt
SWITCHSTATES switchState = SS_AUS;

void loop() {
 Serial.print("Status 1:");
 Serial.println(state1); //gibt den Status der ersten Weiche im Seriellen Monitor aus
 Serial.print("Status 2:");
 Serial.println(state2); //gibt den Status der zweiten Weiche im Seriellen Monitor aus
 //usw für weitere Weichen

 int input1read=analogRead(input1); //Speichert den Wert von Eingang A0 ab
 int input2read=analogRead(input2); //Speichert den Wert von Eingang A1 ab

 Serial.print("Input1 : ");
  Serial.println(input1read);// Gibt den Input1 im Seriellen Monitor wieder.
  Serial.print("Input2 : ");
  Serial.println(input2read);//Gibt den Input2 im Seriellen Monitor wieder.
  delay(200);// 0.2 second pause between readings
 switch(switchState)
 {
  case SS_AUS:
  switchAus(input1read, input2read);
  break;
  case SS_GERADE1:
  switchGerade1(input1read, input2read);
   break;
  case SS_KURVE1:
  switchKurve1(input1read, input2read);
   break;
  case SS_GERADE2:
  switchGerade2(input1read, input2read);
   break;
  case SS_KURVE2:
  switchKurve2(input1read, input2read);
   break;
 }
```

```
}
//switchAus schaltet alle OUTPUTS auf LOW und ist der Ruhezustand
void switchAus(int input1read,int input2read){
  digitalWrite(switch1pin1,LOW);
  digitalWrite(switch1pin2,LOW);
  digitalWrite(switch2pin1,LOW);
  digitalWrite(switch2pin2,LOW);

// Die Inputs werden überprüft. Je nach Kabelverbindung müssen die werde für die Schaltung geändert werden
  if((input1read>800)&& (state1==1)){
    switchState=SS_GERADE1;
  }
  if((input2read>800)&& (state2==1)){
    switchState=SS_GERADE2;
  }
  if((input1read<100)&& (state1==0)){
    switchState=SS_KURVE1;
  }
  if((input2read<100)&& (state2==0)){
    switchState=SS_KURVE2;
  }
  //usw für weitere Weichen
}
//switchGerade1 schaltet Weiche 1 auf Gerade indem switch1pin1 auf HIGH geschaltet wird
void switchGerade1(int input1read,int input2read){
  digitalWrite(switch1pin1,HIGH);
  digitalWrite(switch1pin2,LOW);
  digitalWrite(switch2pin1,LOW);
  digitalWrite(switch2pin2,LOW);
  delay(500); //Strom fließt für 0.5 Sekunden zu der Weiche
  state1=0;
  switchState=SS_AUS;
}

//switchKurve1 schaltet Weiche 1 auf Kurve indem switch1pin2 auf HIGH geschaltet wird
void switchKurve1(int input1read,int input2read){
  digitalWrite(switch1pin1,LOW);
  digitalWrite(switch1pin2,HIGH);
  digitalWrite(switch2pin1,LOW);
  digitalWrite(switch2pin2,LOW);
  delay(500); //Strom fließt für 0.5 Sekunden zu der Weiche
  state1=1;
  switchState=SS_AUS; //Schaltet Strom wieder aus und wechselt in den Ruhezustand
}

//switchGerade2 schaltet Weiche 2 auf Gerade indem switch2pin1 auf HIGH geschaltet wird
void switchGerade2(int input1read,int input2read){
  digitalWrite(switch1pin1,LOW);
  digitalWrite(switch1pin2,LOW);
  digitalWrite(switch2pin1,HIGH);
  digitalWrite(switch2pin2,LOW);
  delay(2000); //Strom fließt für 0.5 Sekunden zu der Weiche
  state2=0;
  switchState=SS_AUS;
}
//switchKurve2 schaltet Weiche 2 auf Kurve indem switch2pin2 auf HIGH geschaltet wird
void switchKurve2(int input1read,int input2read){
  digitalWrite(switch1pin1,LOW);
  digitalWrite(switch1pin2,LOW);
  digitalWrite(switch2pin1,LOW);
  digitalWrite(switch2pin2,HIGH);
  delay(2000); //Strom fließt für 0.5 Sekunden zu der Weiche
  state2=1;
  switchState=SS_AUS; //Schaltet Strom wieder aus und wechselt in den Ruhezustand
}
```

# C. Arbeitsblätter

## C.1. DCC-Arbeitsblatt

DCC-Signale
Lars Bockstette

Datum:_____ Name:_____

*Aufbau eines DCC-Signals*

| 1111111111111111 | 0 | xxxxxxxx | 0 | $\overset{7}{0}\overset{6}{1}\overset{5}{x}\overset{4}{C}\overset{3}{S_3}\overset{2}{S_2}\overset{1}{S_1}\overset{0}{S_0}$ | 0 | PPPPPPPP | 1 |
|---|---|---|---|---|---|---|---|
| Synchronbits | Startbit | Adressbyte | Startbit | Befehlsbyte | Startbit | Prüfbyte | Endbit |

Figure 1: Darstellung eines DCC Datenpakets im Betriebsmodus Bit 5 des Befehlsbytes ist 1 für Vorwärts und 0 für Rückwärts

*Adressen der DCC-Decoder*

| | |
|---|---|
| 0000-0000 | Nachricht an alle Fahrzeugdecoder |
| 0000-0001 bis 0111-1111 | Fahrzeugdecoder mit 7 Bit Adressen 0AAA-AAAA |
| 1000-0000 bis 1011-1111 | Zubehördecoder mit 11 Bit Adressen 10AA-AAAA 1AAA-DAAR/0AA1 |
| 1100-0000 bis 1110-0111 | Fahrzeugdecoder mit 14 Bit Adressen 11AA-AAAA AAAA-AAAA |
| 1110-1000 bis 1111-1110 | Reserviert für zukünftige Anwendungen |
| 1111-1111 | Leerlauf oder auch Idle-Paket |

Figure 2: Adressbyte Aufteilung(A=Adressbit,D=Datenbit)

*Fahrstufen der Lokomotiven*

| $S_3S_2S_1S_0C$ | Fahrstufe | $S_3S_2S_1S_0C$ | Fahrstufe | $S_3S_2S_1S_0C$ | Fahrstufe | $S_3S_2S_1S_0C$ | Fahrstufe |
|---|---|---|---|---|---|---|---|
| 0 0 0 0 0 | Stop | 0 1 0 0 0 | 5 | 1 0 0 0 0 | 13 | 1 1 0 0 0 | 21 |
| 0 0 0 0 1 | Stop** | 0 1 0 0 1 | 6 | 1 0 0 0 1 | 14 | 1 1 0 0 1 | 22 |
| 0 0 0 1 0 | EStop* | 0 1 0 1 0 | 7 | 1 0 0 1 0 | 15 | 1 1 0 1 0 | 23 |
| 0 0 0 1 1 | EStop** | 0 1 0 1 1 | 8 | 1 0 0 1 1 | 16 | 1 1 0 1 1 | 24 |
| 0 0 1 0 0 | 1 | 0 1 1 0 0 | 9 | 1 0 1 0 0 | 17 | 1 1 1 0 0 | 25 |
| 0 0 1 0 1 | 2 | 0 1 1 0 1 | 10 | 1 0 1 0 1 | 18 | 1 1 1 0 1 | 26 |
| 0 0 1 1 0 | 3 | 0 1 1 1 0 | 11 | 1 0 1 1 0 | 19 | 1 1 1 1 0 | 27 |
| 0 0 1 1 1 | 4 | 0 1 1 1 1 | 12 | 1 0 1 1 1 | 20 | 1 1 1 1 1 | 28 |

Figure 3: C ist das LSB(least significant bit) $S_3$ ist das MSB(most significant bit)

DCC-Signale
Lars Bockstette

Datum:_____ Name:_____

*Analysieren und erstellen Sie anhand der Tabellen DCC Signale*

**1** Analysieren Sie die Folgenden DCC Signale und entscheiden Sie ob der Decoder Sie liest.Falls der Decoder Sie liest beschreiben Sie die Aktionen der Lokomotive nach erhalt des Befehls, falls dies nicht der Fall ist beschreiben Sie die Fehler in dem Signal.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1111111111111111 | 0 | 00000000 | 0 | 01100111 | 0 | 0110111 | 1 |
| | | | | | | | |
| 1111111111111111 | 1 | 11010001 | 0 | 01100000 | 0 | 10110001 | 1 |
| | | | | | | | |
| 1111111111111111 | 0 | 00000011 | 0 | 01010111 | 0 | 010111 | 1 |
| | | | | | | | |
| 1111111111111111 | 0 | 00000011 | 0 | 01010111 | 0 | 010100 | 1 |
| | | | | | | | |

**2** Bestimmen Sie die DCC-Signale die die DCC-Zentrale an die Decoder schicken muss damit folgende Befehle Durchgeführt werden.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Lokomotive 5, Vorwärts, Stufe 17 | | | | | | | |
| 1111111111111111 | 0 | | 0 | | 0 | | 1 |
| Lokomotive 31, Rückwärts, Stufe 28 | | | | | | | |
| 1111111111111111 | 0 | | 0 | | 0 | | 1 |
| Alle Lokomotiven, Stop | | | | | | | |
| 1111111111111111 | 0 | | 0 | | 0 | | 1 |
| Lokomotive 199, Vorwärts, Stufe 10 | | | | | | | |
| 1111111111111111 | 0 | | 0 | | 0 | | 1 |

*Lösung A1:*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1111111111111111 | 0 | 00000000 | 0 | 0110111 | 0 | 0110111 | 1 |
| Wird gelesen. An alle Fahrzeugdecoder: Vorwärts, Stufe 4 | | | | | | | |
| 1111111111111111 | 1 | 11010001 | 0 | 01100000 | 0 | 10110001 | 1 |
| Wird nicht gelesen. 1 Fehler Startbit des Adressbytes ist 1 | | | | | | | |
| 1111111111111111 | 0 | 00000011 | 0 | 01010111 | 0 | 01010111 | 1 |
| Wird nicht gelesen. 1 Fehler Prüfbyte ist Falsch | | | | | | | |
| 1111111111111111 | 0 | 00000011 | 0 | 01010111 | 0 | 01010100 | 1 |
| Wird gelesen. An Lokdecoder Nr 3., Rückwärts Stufe 12 | | | | | | | |

*Lösung A2:*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Lokomotive 5, Vorwärts, Stufe 17 | | | | | | | |
| 1111111111111111 | 0 | 00000101 | 0 | 01101110 | 0 | 01101011 | 1 |
| Lokomotive 31, Rückwärts, Stufe 28 | | | | | | | |
| 1111111111111111 | 0 | 00011111 | 0 | 01011111 | 0 | 01000000 | 1 |
| Alle Lokomotiven, Stop | | | | | | | |
| 1111111111111111 | 0 | 00000000 | 0 | 01x00000 | 0 | 01x00000 | 1 |
| Lokomotive 199, Vorwärts, Stufe 10 | | | | | | | |
| 1111111111111111 | 0 | 11000111 | 0 | 01110110 | 0 | 10110001 | 1 |

# D. Arduino Mega Sensor Code



```
//  Initialize and Scan Sensor Data for Serial Transmission to JMRI Sensor Table
//  Author: Geoff Bunza 2018
//   Version 1.2
//  Transmission starts with a synchronizing character, here the character "A" followed by
//  1 byte with  bit 7 Sensor ON/OFF bit 6-0 sensor # 1-127
//  Transmission is received in JMRI via SerialSensorMux.py Python Script
//  It is assumed that the Arduino sensor mux starts up before the Python Script which will transmit "!!!\n"
//  To indicate the script is running and ready for reception
//  The Arduino will then poll all (up to 70) sensors and transmit thir states for initialization in the JMRI sensor table
//  The serial port assigned to this Arduino must correspond to the Serial Port in the corresponding Sensor Script
//  The Arduino will update JMRI ONLY upon detecting a sensor change minimizing overhead transmission to JMRI
//
#define Sensor_Pin_Max   70  // Max sensor pin NUMBER (plus one) Mega=70,UNO,Pro Mini,Nano=20
#define Sensor_Pin_Start  2  // Starting Sensor Pin number (usually 2 as 0/1 are TX/RX
#define Sensor_Offset     18  // This Offset will be ADDED to the value of each Sensor_Pin to determine the sensor
                              // number sent to JMRI, so pin D12 will set sensor AR:(12+Sensor_Offset) in JMRI
                              // This would allow one Arduino Sensor channel to set sensors 2-69 and another to
                              // Set sensors 70-137 for example; this offset can also be negative
#define Sensors_Active_Low 1 // Set Sensors_Active_Low to 1 if sensors are active LOW
                              // Set Sensors_Active_Low to 0 if sensors are active HIGH
#define open_delay 15         // longer delay to get past script initialization
#define delta_delay 4         // Short delay to allow the script to get all the characters
int i;
char  sensor_state [70];      // up to 70 sensors on a Mega2560
char  new_sensor_state ;      // temp to process the possible state change
char  incomingByte = 0;       // working temp for character processing

void setup(){
    Serial.begin(19200);              // Open serial connection.
    while (Serial.available() == 0);  // wait until we get a charater from JMRI
    incomingByte=Serial.read();       // get the first character
    while ((Serial.available() > 0) && (incomingByte != '!')) incomingByte=Serial.read(); //get past !!!
    while ((Serial.available() > 0) ) incomingByte=Serial.read();                        //flush anything else
    delay(open_delay);                 // take a breath
    for ( i=Sensor_Pin_Start; i<Sensor_Pin_Max; i++)  {  //Initialize all sensors in JMRI and grab each sensor
       pinMode(i, INPUT_PULLUP);       // define each sensor pin as coming in
       sensor_state[i] = (digitalRead( i ))^Sensors_Active_Low;    // read & save each sensor state & invert if necessary
       Serial.print("A"); Serial.print (char((sensor_state[i]<<7)+i+Sensor_Offset));  // send "A <on/off><snesor #>" to JMRI script
       delay(delta_delay);             // in milliseconds, take a short breath as not to overwhelm JMRI's seraial read
    }
}
void loop()  {
   for ( i=Sensor_Pin_Start; i<Sensor_Pin_Max; i++)  {     // scan every sensor over and over for any sensor changes
       new_sensor_state = (digitalRead( i ))^Sensors_Active_Low;  // read & save each sensor state & invert if necessary
       if (new_sensor_state != sensor_state[i] )  {               // check if the sensor changed ->if yes update JMRI
         Serial.print("A"); Serial.print (char((new_sensor_state<<7)+i+Sensor_Offset)); // send "A <on/off><snesor #>" to JMRI script
         sensor_state[i] = new_sensor_state ;                    // save the updated sensor state
         delay(delta_delay);            // in milliseconds, take a short breath as not to overwhelm JMRI's seraial read
       }
   }
}
```

# E. Anleitung Sensor Shield/Sensoren einrichten

Die Einrichtung des Sensor Shields in JMRI wird in den Unterkapiteln beschrieben

## E.1. Skript zur Einrichtung

Das Skript welches zur Einrichtung des Sensor Shields in JMRI ausgeführt wird. In Zeile 85 muss der COM-Anschluss des Shields angepasst werden.

```
1   # Capture Sensor Data from an Arduino Serial Transmission REV2
2   # In the form: Character "A" followed by
3   # 1 byte: bit 7 Sensor ON/OFF bit 6-0 sensor # 1-127
4   # Author: Geoff Bunza 2018 based in part on a script by
5   # Bob Jacobsen as part of the JMRI distribution
6   # Version 1.2
7   # An Automat object to create a separate thread
8   # that can sit there, waiting for each character to arrive
9
10  import jarray
11  import jmri
12  import purejavacomm
13  import java.beans
14
15  class SerialSensorMux(jmri.jmrit.automat.AbstractAutomaton) :
16      # ctor starts up the serial port
17      def __init__(self, portname) :
18          global extport
19          self.portID = purejavacomm.CommPortIdentifier.getPortIdentifier(portname)
20          try:
21              self.port = self.portID.open("JMRI", 50)
22          except purejavacomm.PortInUseException:
23              self.port = extport
24          extport = self.port
25          # set options on port
26          baudrate = 19200
27          self.port.setSerialPortParams(baudrate,
28              purejavacomm.SerialPort.DATABITS_8,
29              purejavacomm.SerialPort.STOPBITS_1,
30              purejavacomm.SerialPort.PARITY_NONE)
31          # Anticipate the Port Opening will restart the Arduino
32          self.waitMsec(2000)
33          # get I/O connections for later
34          self.inputStream = self.port.getInputStream()
35          self.outputStream = self.port.getOutputStream()
36          return
37
38      # init() is the place for your initialization
39      def init(self) :
40          return
41
42      # handle() is called repeatedly until it returns false.
43      def handle(self) :
44          global ttest
45          if ttest == 1 :
46                  self.outputStream.write('!')
47                  self.outputStream.write('!')
48                  self.outputStream.write('!')
49                  self.outputStream.write(0x0D)
50                  ttest = 0
51          # get next character
52          if self.inputStream.read() != 65 :
53                  return 1
54          sensor_num = self.inputStream.read()
55          sensor_state = ( sensor_num>>7 ) & 1
56          sensor_num = sensor_num & 0x7f
57          mesg = "AR%d" % (sensor_num)
58          if sensor_num == 1 and sensor_state == 1 :
59              print "Sensor Read Ends"
60              self.inputStream.close()
61              self.outputStream.close()
62              self.port.close()
63              return 0
64          s = sensors.getByUserName( mesg )
65          if s is None :
66                  print mesg, " Not Available"
67                  return 1
68          if sensor_state == 1 :
69                  s.setKnownState(ACTIVE)
70          if sensor_state == 0 :
71                  s.setKnownState(INACTIVE)
72
```

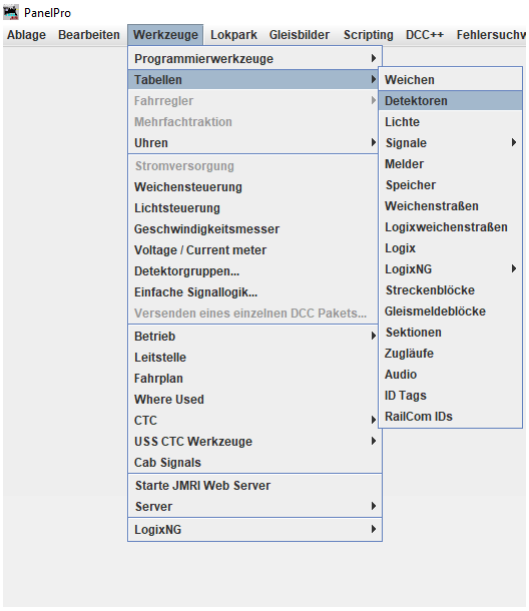
```
73              return 1    # to continue 0 to Kill Script
74
75          def write(self, data) :
76              # now send
77              self.outputStream.write(data)
78              return
79
80          def flush(self) :
81              self.outputStream.flush()
82              return
83      ttest=1
84      # create one of these; provide the name of the serial port
85      a = SerialSensorMux("COM7")
86
87      # set the thread name, so easy to cancel if needed
88      a.setName("SerialSensorMux script")
89
90      # start running
91      a.start();
92
93      # setup  complete
94      a.flush()
95
```

Das Skript welches zum beenden der Verbindung des Sensor Shields mit JMRI genutzt werden kann. In Zeile 29 muss der COM-Anschluss des Shields angepasst werden.

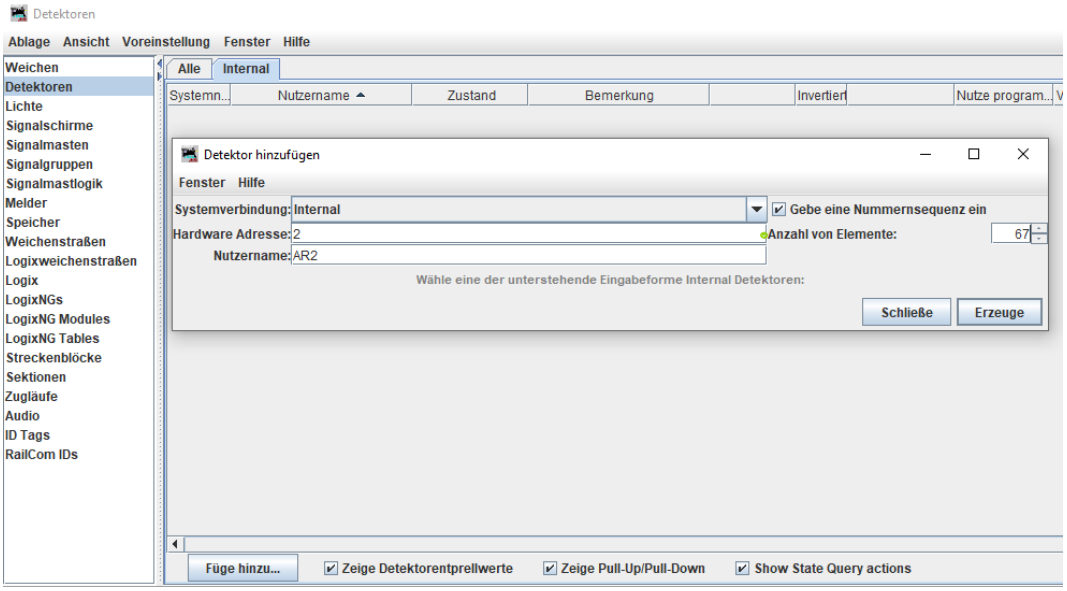
```
1
2   # Capture Sensor Data from an Arduino Serial Transmission
3   # Author: Geoff Bunza 2018 based in part on a script by
4   # Bob Jacobsen as part of the JMRI distribution
5   # Version 1.2
6
7   import jarray
8   import jmri
9   import purejavacomm
10
11  class SerialCloseMux(jmri.jmrit.automat.AbstractAutomaton) :
12
13      # ctor starts up the serial port
14      def __init__(self, portname) :
15          global extport
16          self.portID = purejavacomm.CommPortIdentifier.getPortIdentifier(portname)
17          extport.close()
18          return
19
20      # init() is the place for your initialization
21      def init(self) :
22
23          return
24
25      # handle() is called repeatedly until it returns false.
26      def handle(self) :
27
28            return 0
29
30      def write(self, data) :
31
32          return
33
34      def flush(self) :
35          self.outputStream.flush()
36          return
37
38  # create one of these; provide the name of the serial port
39  a = SerialCloseMux("COM7")
40
41  # set the thread name, so easy to cancel if needed
42  a.setName("SerialCloseMux script")
43
44  # start running
45  a.start();
46
47
```

## E.2. Sensoren einrichten

Um Sensoren hinzuzufügen wird die Tabelle der Sensoren ausgewählt




Der Arduino Mega unterstützt 67 Sensoren. Der Erste Sensor muss frei bleiben. Deswegen Hardware Adresse 2(Entspricht Pin2). Die Skripts dieser Bachelorarbeit Arbeiten mit den Nutzernamen AR2-AR67. Die Namenswahl ist beliebig muss jedoch in den Skripten beachtet werden.
Damit nicht jeder Sensor einzeln hinzugefügt werden muss, setzen wir den Haken bei Nummernsequenz und tragen 67 Elemente ein. JMRI übernimmt Automatisch die Nummerierung der Adressen und Nutzernamen.



# F. Skripte

## F.1. Befehle für die Skripts

| Typ | Befehl | Beschreibung |
|---|---|---|
| Lokomotiven | | |
| | self.throttleVariable=self.getThrottle(DCCADRESSE, True/False) | Zuweisen einer Lok zu einer Variabel. True=Lange DCCADRESSE |
| | self.throttleVariable.setIsForward(True/False) | Setzt die Richtung der Lok |
| | self.throttleVariable.setSpeedSetting(x) | setzt die Geschwindigkeit der Lok 0<=x<=1 |
| | self.throttleVariable.getIsForward() | gibt True zurück wenn die Lok Vorwärts fährt und False bei Rückwärts |
| Blocks | | |
| | self.blockVariable=blocks.provideBlock(SSystemname") | Zuweisen eines Blocks zu einer Variable. Übergebener Parameter ist der Systemname aus JMRI des Blocks |
| | self.blockVariable.goingActive() | setzt den Block auf Besetzt |
| | self.blockVariable.goingInactive() | setzt den Block auf Frei |
| | self.blockVariable.getReporter() | Gibt den mit dem Block in JMRI assozierten Melder aus |
| Weichen | | |
| | self.weichenVariable=turnouts.provideTurnout(SSystemname") | Zuweisen einer Weiche zu einer Variable. Übergebener Parameter ist der Systemname aus JMRI der Weiche |
| | self.weichenVariable.setState(THROWN/CLOSED) | Überführt die Weiche in THROWN/CLOSED |
| | self.weichenVariable.invertTurnoutState(CLOSED/THROWN) | überführt die Weiche falls sie CLOSED ist in THROWN oder andersherum. |
| Lampen | | |
| | self.lampenVariable=lights.provideLight(SSystemname") | Zuweisen einer Lampe zu einer Variable. Übergebener Parameter ist der Systemname aus JMRI der Lampe |
| | self.lampenVariable.setCommandedState(ON/OFF) | Schaltet die Lampe an/aus. Weitere Befehle wie Mittelhell sind der Lights.Java Klasse von JMRI zu entnehmen |
| | self.lampenVariable.getKnownState() | gibt den aktuellen Status der Lampe aus. |
| Melder | | |
| | self.reportervariable=reporters.provideReporter(SSystemname") | Zuweisen eines Melders zu einer Variable. Übergebener Parameter ist der Systemname aus JMRI des Melders |
| | self.reportervariable.setReport(Eingabe") | Rückgabe sollte die Form DCCADRESSE+ëntersöder DCCADRESSE+ ëxits"haben. |
| | self.reportervariable.getCurrentReport() | Gibt den aktuellen Report des Melders Zurück oder NULL wenn keiner vorhanden ist |
| | self.reportervariable.getLastReport() | Gibt den Report zurück der Vor dem aktuellen Report stattgefunden hat wenn der CurrentReport NULL ist. |
| Sensoren | | |
| | self.SensorVariable = sensors.provideSensor(SSystemname") | Zuweisen eines Sensors zu einer Variable. Übergebener Parameter ist der Systemname aus JMRI des Sensors |
| | self.SensorVariable.getState() | Übergibt den Status des Sensors. ACTIVE / INACTIVE |
| | self.SensorVariable.setKnownState(ACTIVE/INACTIVE) | Übergibt ACTIVE/INACTIVE an den Sensor. |
| | self.SensorVariable.getReporter() | Gibt den mit dem Sensor assozierten Melder zurück |

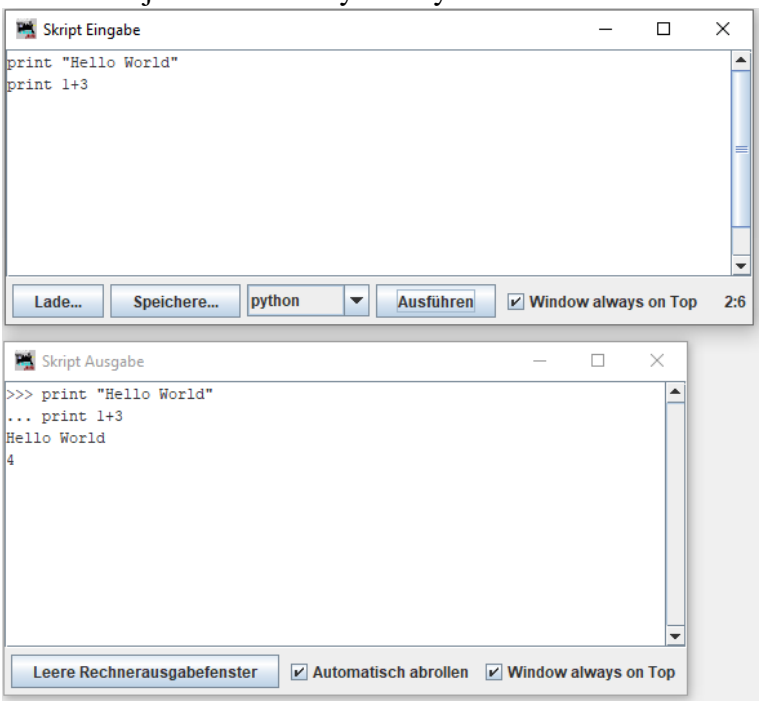
Abbildung F.1: *Einfache Befehle für die Skriptnutzung in JMRI für die Wichtigsten Klassen.*

## F.2. Anleitung Skriptdurchführung

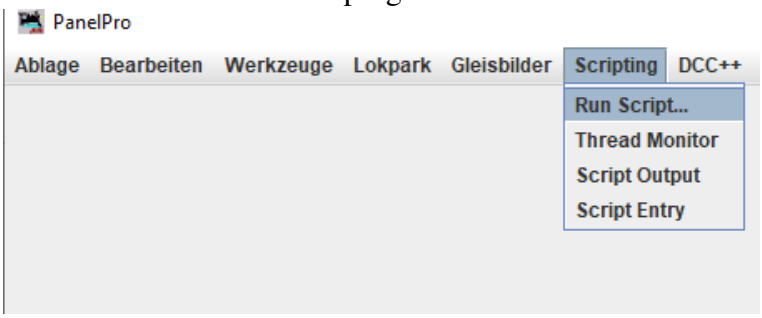
Die Manuelle Eingabe des Programmcodes erfolgt über *Script Entry* Die Ausgabe


erfolgt über *Script Output*

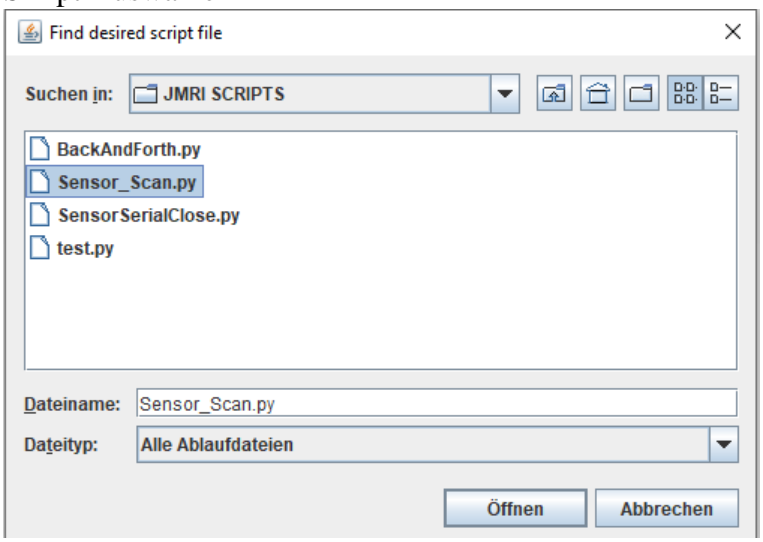
Hier kann jede Art von Python/Jython Code in der Konsole ausgeführt werden.




Die Skripts die in dieser Bachelorarbeit benutzt wurden, wurden außerhalb der Konsole programmiert und über *Run Script* ausgeführt

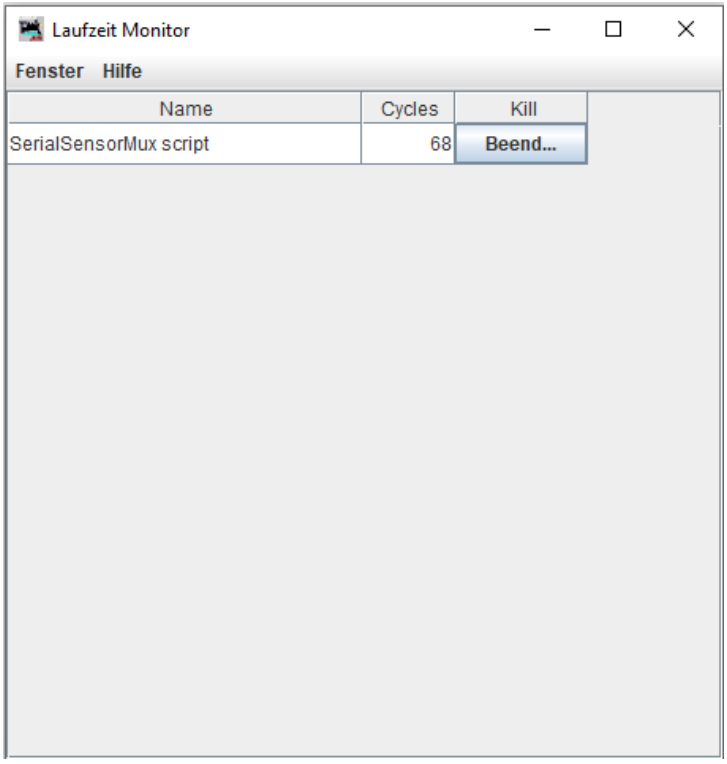



Skript Auswählen



War die Ausführung des Skripts erfolgreich erscheint ein Task im Laufzeit Monitor


Hier sind alle derzeit ausgeführten Skripts aufgeführt mit Namen und Anzahl der Programmdurchläufe.



# G. GREENFOOT

## G.1. Greenfoot Code Änderungen

Die Namen der einzelnen Gleisabschnitte in der Greenfoot Klasse *PanelWorld* in der Funktion *prepare()* müssen mit den Usernamen der Blöcke in JMRI übereinstimmen. Die Gleisabschnitte erhalten die Parameter Gleistyp, Username der Blöcke in JMRI.
Zusätzlich muss der Zug Username aus JMRI in derselben Funktion bei allen Buttons geändert werden sowie die Usernamen der Weichen auf die Usernamen in JMRI angepasst werden.

## G.2. Greenfoot individuelle Zugsteuerung

Der Programmcode ist für einen Button der die Geschwindigkeit einer Lokomotive über ein Popup Panel steuert

ButtonScript 2021-Dez.-14 11:02 Page 1

```
1  import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot und MouseInfo)
2  import de.wwu.jmrigreenfootinterface.*;
3  import javax.swing.*;
4  /**
5   * A button that lets a specific train execute a custom script.
6   *
7   * @author Leonard Bienbeck,Lars Bockstette
8   * @version 1.0.1
9   */
10 public class ButtonScript extends Button
11 {
12     @Override
13     public void onClick() {
14       JTextField xField = new JTextField(5);
15       JTextField yField = new JTextField(5);
16 
17       JPanel myPanel = new JPanel();
18       myPanel.add(new JLabel("Lokomotive:"));
19       myPanel.add(xField);
20       myPanel.add(Box.createHorizontalStrut(15)); // a spacer
21       myPanel.add(new JLabel("Geschwindigkeit:"));
22       myPanel.add(yField);
23 
24       int result = JOptionPane.showConfirmDialog(null, myPanel,
25                 "Bitte geben Sie die DCC Adresse und die Geschwindigkeit ein.",
JOptionPane.OK_CANCEL_OPTION);
26       if (result == JOptionPane.OK_OPTION) {
27         // Error abfangen wenn ungültige Eingabe in Geschwindigkeit geschieht
28          try {
29           int geschwindigkeit = Integer.parseInt(yField.getText());
30           // Error abfangen wenn ungültige Eingabe in Lokomotive geschieht
31           try {
32               Integer.parseInt(xField.getText());
33               System.out.println("DCC Adresse: " + xField.getText());
34               System.out.println("Geschwindigkeit: " + yField.getText());
35              JMRI.getInterface().addLocomotive("S"+xField.getText(),
"S"+xField.getText());
36              JMRI.getInterface().setSpeed("S"+xField.getText(), geschwindigkeit);
37             }
38             catch(NumberFormatException e) {
39                System.out.println("Die DCC Adresse einer Lokomotive ist eine
Integer");
40             }
41 
42         }
43         catch (NumberFormatException e) {
44           System.out.println("Geschwindigkeits Eingabe muss ein Integer sein");
45         }
46       }
47     }
48 }
49 
50 
```

# H. Skripts zur Gleisbelegung

## H.1. Ohne Lokrückmeldung

Das Skript setzt innerhalb von JMRI einen Gleisabschnitt auf belegt

```
1  import jarray
2  import jmri
3
4
5  class Belegtstatuts(jmri.jmrit.automat.AbstractAutomaton) :
6      strecke = 0
7      def init(self):
8          # init() wird beim Start einmalig aufgerufen(Ähnlich der Setup() funktion
           des Arduinos)
9          print "Es beginnt init(self)"
10
11         # Die Sensoren werden deklariert
12
13         self.Sensor1 = sensors.provideSensor("AR22")
14         self.Sensor2 = sensors.provideSensor("AR23")
15         self.Sensor3 = sensors.provideSensor("AR24")
16         self.Sensor4 = sensors.provideSensor("AR25")
17         self.Sensor5 = sensors.provideSensor("AR26")
18         self.Sensor6 = sensors.provideSensor("AR27")
19         print "Sensoren Fertig"
20
21         # Die DCC Adresse der Lok(In JMRI Lokpark nachschaubar).
22         # Wenn die Lange Adresse vergeben wurde muss "False" zu "True" geändert werden
23         self.throttle1 = self.getThrottle(3, False)  # short address 3
24
25         # Die Blöcke werden deklariert
26         self.Links=blocks.provideBlock("IB:AUTO:0003")
27         self.Oben=blocks.provideBlock("IB:AUTO:0001")
28         self.Rechts=blocks.provideBlock("IB:AUTO:0004")
29         self.Unten=blocks.provideBlock("IB:AUTO:0002")
30         self.Weiche=blocks.provideBlock("IB:AUTO:0005")
31
32         # setze die lok in bewegung
33
34         forward = True #Ändern für Vorwärts oder Rückwärts
35
36
37         self.throttle1.setIsForward(forward)
38         # Warte 1 Sekunde und Setze die Geschwindigkeit(0-1, 1 entspricht 100%)
39         self.waitMsec(1000)
40         print "Setze Geschwindigkeit"
41         self.throttle1.setSpeedSetting(0.5)
42         #Aktuellen Standort der Lok Ermitteln
43         print "Aktivierung des Ersten Sensors Abwarten"
44         self.inaktivSetzen()
45         return
46
47     def aktivSetzen(self, b, s):  #b ist der Block, s ist der Sensor
48         b.goingActive()
49         print s,"Aktiviert"
50         print b, "Besetzt"
51         return
52
53     def inaktivSetzen(self):
54         self.Links.goingInactive()
55         self.Oben.goingInactive()
56         self.Rechts.goingInactive()
57         self.Unten.goingInactive()
58         self.Weiche.goingInactive()
59
60         return
61
62     def handle(self):
63         # handle() wird wiederholt ausgeführt bis das Programm beendet wird.
64         print "Es beginnt handle(self)"
65         self.waitMsec(500)
66
67         forward=self.throttle1.getIsForward()
68         if(forward==True):
69             if(self.Sensor1.getState()==ACTIVE):
70                 self.inaktivSetzen()
71                 self.aktivSetzen(self.Links,self.Sensor1)
```

```
72                elif(self.Sensor2.getState()==ACTIVE):
73                    self.inaktivSetzen()
74                    self.aktivSetzen(self.Unten,self.Sensor2)
75                elif(self.Sensor3.getState()==ACTIVE):
76                    self.inaktivSetzen()
77                    self.aktivSetzen(self.Rechts,self.Sensor3)
78                elif(self.Sensor4.getState()==ACTIVE):
79                    self.inaktivSetzen()
80                    self.aktivSetzen(self.Oben,self.Sensor4)
81                elif(self.Sensor5.getState()==ACTIVE):
82                    self.inaktivSetzen()
83                    self.aktivSetzen(self.Weiche,self.Sensor5)
84                elif(self.Sensor6.getState()==ACTIVE):
85                    self.inaktivSetzen()
86                    self.aktivSetzen(self.Unten,self.Sensor6)
87            elif(forward==False):
88                if(self.Sensor1.getState()==ACTIVE):
89                    self.inaktivSetzen()
90                    self.aktivSetzen(self.Oben,self.Sensor1)
91                elif(self.Sensor2.getState()==ACTIVE):
92                    self.inaktivSetzen()
93                    self.aktivSetzen(self.Links,self.Sensor2)
94                elif(self.Sensor3.getState()==ACTIVE):
95                    self.inaktivSetzen()
96                    self.aktivSetzen(self.Unten,self.Sensor3)
97                elif(self.Sensor4.getState()==ACTIVE):
98                    self.inaktivSetzen()
99                    self.aktivSetzen(self.Rechts,self.Sensor4)
100               elif(self.Sensor5.getState()==ACTIVE):
101                   self.inaktivSetzen()
102                   self.aktivSetzen(self.Unten,self.Sensor5)
103               elif(self.Sensor6.getState()==ACTIVE):
104                   self.inaktivSetzen()
105                   self.aktivSetzen(self.Weiche,self.Sensor6)
106               print "Lok faehrt Rückwaerts"
107
108
109           # Startet Skript von Vorne
110           print "Ende des Loops"
111           return 1
112           # Beenden Über JMRI-Laufzeitmonitor und Fahrregler der Lok
113
114
115   # end of class definition
116
117   # start one of these up
118   Belegtstatuts().start()
119
```

## H.2. Mit einer Lokomotive

Setzt in JMRI Gleisblöcke auf belegt mit Namensrückgabe für 1 Lok.

```
1  import jarray
2  import jmri
3
4
5  class Belegtstatuts(jmri.jmrit.automat.AbstractAutomaton) :
6      strecke = 0
7      def init(self):
8          # init() wird beim Start einmalig aufgerufen(Ähnlich der Setup() funktion
       des Arduinos)
9          print "Es beginnt init(self)"
10
11         # Die Sensoren werden deklariert
12
13         self.Sensor1 = sensors.provideSensor("AR22")
14         self.Sensor2 = sensors.provideSensor("AR23")
15         self.Sensor3 = sensors.provideSensor("AR24")
16         self.Sensor4 = sensors.provideSensor("AR25")
17         self.Sensor5 = sensors.provideSensor("AR26")
18         self.Sensor6 = sensors.provideSensor("AR27")
19         self.Report1 = reporters.provideReporter("IR100")
20         self.Report2 = reporters.provideReporter("IR101")
21         self.Report3 = reporters.provideReporter("IR102")
22         self.Report4 = reporters.provideReporter("IR103")
23         self.Report5 = reporters.provideReporter("IR104")
24         print "Sensoren Fertig"
25
26         # Die DCC Adresse der Lok(In JMRI Lokpark nachschaubar).
27         # Wenn die Lange Adresse vergeben wurde muss "False" zu "True" geändert werden
28         self.throttle1 = self.getThrottle(3, False)  # short address 3
29
30         # Die Blöcke werden deklariert
31         self.Links=blocks.provideBlock("IB:AUTO:0002")
32         self.Oben=blocks.provideBlock("IB:AUTO:0001")
33         self.Rechts=blocks.provideBlock("IB:AUTO:0005")
34         self.Unten=blocks.provideBlock("IB:AUTO:0003")
35         self.Weiche=blocks.provideBlock("IB:AUTO:0004")
36
37         # setze die lok in bewegung
38
39         forward = True #Ändern für Vorwärts oder Rückwärts
40
41
42         self.throttle1.setIsForward(forward)
43         # Warte 1 Sekunde und Setze die Geschwindigkeit(0-1, 1 entspricht 100%)
44         self.waitMsec(1000)
45         print "Setze Geschwindigkeit"
46         self.throttle1.setSpeedSetting(0.5)
47         #Aktuellen Standort der Lok Ermitteln
48         print "Aktivierung des Ersten Sensors Abwarten"
49         self.inaktivSetzen()
50         return
51
52     def aktivSetzen(self, b, s):  #b ist der Block, s ist der Sensor
53         b.goingActive()
54         b.getReporter().setReport("3 Enter")
55         print b.getReporter(),"Aktiviert"
56         print s,"Aktiviert"
57         print b, "Besetzt"
58         return
59
60     def inaktivSetzen(self):
61         self.Links.goingInactive()
62         self.Oben.goingInactive()
63         self.Rechts.goingInactive()
64         self.Unten.goingInactive()
65         self.Weiche.goingInactive()
66         self.Links.getReporter().setReport("3 exits")
67         self.Oben.getReporter().setReport("3 exits")
68         self.Rechts.getReporter().setReport("3 exits")
69         self.Unten.getReporter().setReport("3 exits")
70         self.Weiche.getReporter().setReport("3 exits")
71
```

```
72            return
73
74        def handle(self):
75            # handle() wird wiederholt ausgeführt bis das Programm beendet wird.
76            print "Es beginnt handle(self)"
77            self.waitMsec(500)
78
79            forward=self.throttle1.getIsForward()
80            if(forward==True):
81                if(self.Sensor1.getState()==ACTIVE):
82                    self.inaktivSetzen()
83                    self.aktivSetzen(self.Links,self.Sensor1)
84                elif(self.Sensor2.getState()==ACTIVE):
85                    self.inaktivSetzen()
86                    self.aktivSetzen(self.Unten,self.Sensor2)
87                elif(self.Sensor3.getState()==ACTIVE):
88                    self.inaktivSetzen()
89                    self.aktivSetzen(self.Rechts,self.Sensor3)
90                elif(self.Sensor4.getState()==ACTIVE):
91                    self.inaktivSetzen()
92                    self.aktivSetzen(self.Oben,self.Sensor4)
93                elif(self.Sensor5.getState()==ACTIVE):
94                    self.inaktivSetzen()
95                    self.aktivSetzen(self.Weiche,self.Sensor5)
96                elif(self.Sensor6.getState()==ACTIVE):
97                    self.inaktivSetzen()
98                    self.aktivSetzen(self.Unten,self.Sensor6)
99            elif(forward==False):
100               if(self.Sensor1.getState()==ACTIVE):
101                   self.inaktivSetzen()
102                   self.aktivSetzen(self.Oben,self.Sensor1)
103               elif(self.Sensor2.getState()==ACTIVE):
104                   self.inaktivSetzen()
105                   self.aktivSetzen(self.Links,self.Sensor2)
106               elif(self.Sensor3.getState()==ACTIVE):
107                   self.inaktivSetzen()
108                   self.aktivSetzen(self.Unten,self.Sensor3)
109               elif(self.Sensor4.getState()==ACTIVE):
110                   self.inaktivSetzen()
111                   self.aktivSetzen(self.Rechts,self.Sensor4)
112               elif(self.Sensor5.getState()==ACTIVE):
113                   self.inaktivSetzen()
114                   self.aktivSetzen(self.Unten,self.Sensor5)
115               elif(self.Sensor6.getState()==ACTIVE):
116                   self.inaktivSetzen()
117                   self.aktivSetzen(self.Weiche,self.Sensor6)
118               print "Lok faehrt Rückwaerts"
119
120
121           # Startet Skript von Vorne
122           print "Ende des Loops"
123           return 1
124           # Beenden Über JMRI-Laufzeitmonitor und Fahrregler der Lok
125
126
127   # end of class definition
128
129   # start one of these up
130   Belegtstatuts().start()
131
```

## H.3. Mit mehreren Lokomotiven

Setzt in JMRI Gleisblöcke auf belegt mit Namensrückgabe für mehrere Loks.

```
1   import jarray
2   import jmri
3
4
5   class Belegtstatuts(jmri.jmrit.automat.AbstractAutomaton) :
6       strecke = 0
7       global letzterSensor
8
9       #Gibt den vorherigen Block zurück. Muss je nach Layout geändert werden
10      def checkPrev(self, b,t):#b ist der Block, t ist die Richtung der Throttles
11          if t:
12              if(b==self.Links):
13                  return self.Oben
14              elif(b==self.Unten):
15                  return self.Links
16              elif(b==self.Oben):
17                  return self.Rechts
18              elif(b==self.Weiche):
19                  return self.Links
20              elif(b==self.Rechts):
21
                    if(self.reportBlock[self.Weiche].getCurrentReport().endswith("Enter"))
                    :
22                      return self.Weiche
23                  else:
24                      return self.Unten
25          else:
26              if(b==self.Rechts):
27                  return self.Oben
28              elif(b==self.Unten):
29                  return self.Rechts
30              elif(b==self.Oben):
31                  return self.Links
32              elif(b==self.Weiche):
33                  return self.Rechts
34              elif(b==self.Links):
35
                    if(self.reportBlock[self.Weiche].getCurrentReport().endswith("Enter"))
                    :
36                      return self.Weiche
37                  else:
38                      return self.Unten
39          return
40
41      def init(self):
42          # init() wird beim Start einmalig aufgerufen(Ähnlich der Setup() funktion
            des Arduinos)
43          print "Es beginnt init(self)"
44
45          # Die Sensoren werden deklariert
46          self.throttledict={} #Hier werden die einzelnen Throttles der Züge gespeichert
47          self.positionendict={} # Hier werden die Aktuellen Positionen der Züge
            gespeichert
48          self.sensorDict={}  # Hier werden die Sensoren abgespeichert mit der Nummer
            dem Objekt zugeordnet
49          self.reportDict={} # Hier werden die Melder abgespeichert mit der Nummer dem
            Objekt zugeordnet
50
51
52
53          # Die Blöcke werden deklariert
54          self.Links=blocks.provideBlock("IB:AUTO:0002")
55          self.Oben=blocks.provideBlock("IB:AUTO:0001")
56          self.Rechts=blocks.provideBlock("IB:AUTO:0005")
57          self.Unten=blocks.provideBlock("IB:AUTO:0003")
58          self.Weiche=blocks.provideBlock("IB:AUTO:0004")
59          self.Links.goingInactive()
60          self.Oben.goingInactive()
61          self.Rechts.goingInactive()
62          self.Unten.goingInactive()
63          self.Weiche.goingInactive()
64
```

```
65          #Hier werden die Adressen der Loks geändert
66          Loks=[3,4] #3 Loks mit den Adressen 3,4,...
67          Sensoren=[22,23,24,25,26,27] #Hier können weitere Sensoren eingetragen werden
68          Melder=[100,101,102,103,104] #Hier können weitere Sensoren eingetragen werden
69
            self.fwdSensoMelder={Sensoren[0]:self.Links,Sensoren[1]:self.Unten,Sensoren[2]
            :self.Rechts,Sensoren[3]:self.Oben,Sensoren[4]:self.Weiche,Sensoren[5]:self.Un
            ten}
70
            self.bckSensoMelder={Sensoren[0]:self.Oben,Sensoren[1]:self.Links,Sensoren[2]:
            self.Unten,Sensoren[3]:self.Rechts,Sensoren[4]:self.Unten,Sensoren[5]:self.Wei
            che}
71          #Die Melder werden den Streckenabschnitten zugeordnet
72
            self.reportBlock={self.Links:self.Links.getReporter(),self.Oben:self.Oben.getR
            eporter(),self.Rechts:self.Rechts.getReporter(),self.Unten:self.Unten.getRepor
            ter(),self.Weiche:self.Weiche.getReporter()}
73          #Hier wird die Reihenfolge festgehalten
74          self.fwd=[self.Links,self.Unten,self.Weiche,self.Rechts,self.Oben]
75
76          print self.reportBlock
77          # Die DCC Adresse der Lok(In JMRI Lokpark nachschaubar).
78          # Wenn die Lange Adresse vergeben wurde muss "False" zu "True" geändert werden
79          Start=[self.Links,self.Rechts]
80          self.positionendict={self.Oben:Loks[0],self.Unten:Loks[1]}#Hier die
            StartPositionen der Lok eingeben, PRO BLOCK NUR 1 ZUG
81          for i in Sensoren:
82              sensor=sensors.provideSensor("AR"+str(i))
83              self.sensorDict[i]=sensor
84
85          for i in Melder:
86              reporter=reporters.provideReporter("IR"+str(i))
87              self.reportDict[i]=reporter
88          print "Sensoren und Melder Fertig"
89          #Die Throttles werden erstellt und dem ThrottleDict hinzugefügt
90          for i in Loks:
91              throttle=self.getThrottle(i, False)
92              self.throttledict[i]=throttle
93          print "Dies sind die Züge im Roster"
94          for keys,values in self.throttledict.items():
95              print(keys)
96              print(values)
97
98          print "Und hier Starten Sie"
99          for keys,values in self.positionendict.items():
100             print(keys)
101             print(values)
102         # setze die lok in bewegung
103
104         forward = True #Ändern für Vorwärts oder Rückwärts
105         #Kann auch in Listen verwaltet werden und dann seperat per Lok zugeordnet
            werden. Lohnt sich jedoch nur bei Großen
106         #Schienenlayouts da ansonsten zusammenstöße passieren könnten.
107         #Dieses Skript ist darauf ausgelegt, dass wenn die Richtung geändert wird.
            Dies Bei ALLEN Loks gleichzeitig geschieht
108
109         # Damit zusammenstöße verhindert werden
110
111         for key in self.throttledict:
112             self.throttledict[key].setIsForward(forward)
113
114         # Warte 1 Sekunde und Setze die Geschwindigkeit(0-1, 1 entspricht 100%)
115         self.waitMsec(1000)
116         print "Setze Geschwindigkeit"
117         for key in self.throttledict:
118             self.throttledict[key].setSpeedSetting(0.5)
119         #Aktuellen Standort der Lok Ermitteln
120         print "Aktivierung des Ersten Sensors Abwarten"
121         for key in self.reportDict:
122             self.reportDict[key].setReport("")
123         return
124
```

```
125     def aktivSetzen(self, b, s,r,l):  #b ist der Block, s ist der Sensor,r der
        Reporter,l die Lok
126         b.goingActive()
127         r.setReport(str(l)+" Enter")
128         print r,"Aktiviert"
129         print s,"Aktiviert"
130         print b, "Besetzt"
131         return
132 
133     def inaktivSetzen(self,b,r,l): #b ist der Block, r der Reporter,l die Lok
134         b.goingInactive()
135         r.setReport(str(l)+" exits")
136         return
137 
138 
139     #Checkt ob ein Sensor Aktiv wird und gibt den Entsprechenden Sensor aus nachdem
        er wieder auf Inaktiv geschaltet wird
140     #So werden sensoren nur für den Bruchteil einer Sekunde Aktiv geschaltet
141     #Dies ist wichtig falls 2 Loks zur selben zeit Sensoren Aktivieren
142     def anySensor(self):
143         global letzterSensor
144         for key in self.sensorDict:
145             if(self.sensorDict[key].getState()==ACTIVE):
146                 self.sensorDict[key].setState(INACTIVE)
147                 letzterSensor=key
148                 print "key:" + str(key)
149                 print "letzterSensor:"+str(letzterSensor)
150                 return letzterSensor
151         return False
152 
153     def handle(self):
154         global letzterSensor
155         # handle() wird wiederholt ausgeführt bis das Programm beendet wird.
156         print "Es beginnt handle(self)"
157         self.waitMsec(500)
158 
159 
160         if(self.throttledict[3].getIsForward):
161             for key in self.throttledict:
162                 #Alle Loks werden auf dieselbe Richtung wie Lok 1 gebracht
163                 self.throttledict[key].setIsForward(True)
164             #Falls ein Sensor Signal gibt
165             if (self.anySensor()):
166                 #gehe die Sensoren durch und schaue welcher der Letzte Sensor war
                    der Signal gegeben hat
167                 for key in self.sensorDict:
168                     if(key==letzterSensor):
169                         #Der Letzte Block auf dem die Lok war wird gesucht
170                         letzterBlock=self.checkPrev(self.fwdSensoMelder[key],True)
171                         #Dem PositionenDict wird die Aktuelle Position der Bahn
                            mitgeteilt und die Alte position wird gelöscht
172 
                            self.positionendict[self.fwdSensoMelder[key]]=self.positionend
                            ict.pop(letzterBlock)
173                         #Der Block, der Sensor , der Reporter und die Lok werden an
                            aktivSetzen() übergeben
174 
                            self.aktivSetzen(self.fwdSensoMelder[key],key,self.reportBlock
                            [self.fwdSensoMelder[key]],self.positionendict[self.fwdSensoMe
                            lder[key]])
175                         print "TOEFFTOEFF:"
176                         print self.positionendict[self.fwdSensoMelder[key]]
177                         #Der Block, der Reporter und die Lok werden inaktiv setzen
                            übergeben
178 
                            self.inaktivSetzen(letzterBlock,self.reportBlock[letzterBlock]
                            ,self.positionendict[self.fwdSensoMelder[key]])
179         else:
180             for key in self.throttledict:#Alle Loks werden auf dieselbe Richtung wie
                Lok 1 gebracht
181                 self.throttledict[key].setIsForward(False)
182             if (self.anySensor()):
```

```
183                     #gehe die Sensoren durch und schaue welcher der Letzte Sensor war
                        der Signal gegeben hat
184                     for key in self.sensorDict:
185                         if(key==letzterSensor):
186                             #Der Letzte Block auf dem die Lok war wird gesucht
187                             letzterBlock=self.checkPrev(self.bckSensoMelder[key],True)
188                             #Dem PositionenDict wird die Aktuelle Position der Bahn
                                mitgeteilt und die Alte position wird gelöscht
189
                                self.positionendict[self.bckSensoMelder[key]]=self.positionend
                                ict.pop(letzterBlock)
190                             #Der Block, der Sensor , der Reporter und die Lok werden an
                                aktivSetzen() übergeben
191
                                self.aktivSetzen(self.bckSensoMelder[key],key,self.reportBlock
                                [self.bckSensoMelder[key]],self.positionendict[self.bckSensoMe
                                lder[key]])
192                             print "TOEFFTOEFF:"
193                             print self.positionendict[self.bckSensoMelder[key]]
194                             #Der Block, der Reporter und die Lok werden inaktiv setzen
                                übergeben
195
                                self.inaktivSetzen(letzterBlock,self.reportBlock[letzterBlock]
                                ,self.positionendict[self.bckSensoMelder[key]])
196             # Startet Skript von Vorne
197             print "Ende des Loops"
198             return 1
199             # Beenden Über JMRI-Laufzeitmonitor und Fahrregler der Lok
200
201
202     # end of class definition
203
204     # start one of these up
205     Belegtstatuts().start()
206
```

## H.4. Für zwei Layouts

Setzt in JMRI Gleisblöcke belegt mit Namensrückgabe,mehrere Loks & Layouts.

```
1   import jarray
2   import jmri
3
4
5   class Belegtstatuts(jmri.jmrit.automat.AbstractAutomaton) :
6       strecke = 0
7       global letzterSensor
8
9       #Gibt den vorherigen Block zurück. Muss je nach Layout geändert werden
10      def checkPrev(self, b,t):#b ist der Block, t ist die Richtung der Throttles
11          if t:
12              if(b==self.LLinks):
13                  return self.LOben
14              elif(b==self.LUnten):
15                  return self.LLinks
16              elif(b==self.LRechts):
17                  return self.LUnten
18              elif(b==self.LOben):
19                  return self.LRechts
20              elif(b==self.RLinks):
21                  return self.ROben
22              elif(b==self.RUnten):
23                  return self.RLinks
24              elif(b==self.RRechts):
25                  return self.RUnten
26              elif(b==self.ROben):
27                  return self.RRechts
28          else:
29              if(b==self.LLinks):
30                  return self.LUnten
31              elif(b==self.LUnten):
32                  return self.LRechts
33              elif(b==self.LRechts):
34                  return self.LOben
35              elif(b==self.LOben):
36                  return self.LLinks
37              elif(b==self.RLinks):
38                  return self.RUnten
39              elif(b==self.RUnten):
40                  return self.RRechts
41              elif(b==self.RRechts):
42                  return self.ROben
43              elif(b==self.ROben):
44                  return self.RLinks
45          return
46
47      def init(self):
48          # init() wird beim Start einmalig aufgerufen(Ähnlich der Setup() funktion
            des Arduinos)
49          print "Es beginnt init(self)"
50
51          # Die Sensoren werden deklariert
52          self.throttledict={} #Hier werden die einzelnen Throttles der Züge gespeichert
53          self.positionendict={} # Hier werden die Aktuellen Positionen der Züge
            gespeichert
54          self.LsensorDict={}
55          self.LreportDict={}
56          self.RsensorDict={}
57          self.RreportDict={}
58          self.LinkesLayout=[]
59          self.RechtesLayout=[]
60
61
62          # Die Blöcke werden deklariert
63          self.LLinks=blocks.provideBlock("IBLLinks")
64          self.LOben=blocks.provideBlock("IBOLinks")
65          self.LRechts=blocks.provideBlock("IBRLinks")
66          self.LUnten=blocks.provideBlock("IBULinks")
67          self.LLinks.goingInactive()
68          self.LOben.goingInactive()
69          self.LRechts.goingInactive()
70          self.LUnten.goingInactive()
```

```
71          self.RLinks=blocks.provideBlock("IBLRechts")
72          self.ROben=blocks.provideBlock("IBORechts")
73          self.RRechts=blocks.provideBlock("IBRRechts")
74          self.RUnten=blocks.provideBlock("IBURechts")
75          self.RLinks.goingInactive()
76          self.ROben.goingInactive()
77          self.RRechts.goingInactive()
78          self.RUnten.goingInactive()
79
80          #Hier werden die Adressen der Loks geändert
81          Loks=[3,4] #3 Loks mit den Adressen 3,4,...
82          LSensoren=[10,11,12,13]
83          RSensoren=[14,15,16,17]#Hier können weitere Sensoren eingetragen werden
84          LMelder=[100,101,102,103] #Hier können weitere Melder eingetragen werden
85          RMelder=[104,105,106,107]
86          #Die Reihenfolge der Streckenabschnitte wird den Sensoren zugeteilt
87
            self.LfwdSensoMelder={LSensoren[0]:self.LLinks,LSensoren[1]:self.LUnten,LSenso
            ren[2]:self.LRechts,LSensoren[3]:self.LOben}
88
            self.LbckSensoMelder={LSensoren[0]:self.LOben,LSensoren[1]:self.LLinks,LSensor
            en[2]:self.LUnten,LSensoren[3]:self.LRechts}
89
            self.RfwdSensoMelder={RSensoren[0]:self.RLinks,RSensoren[1]:self.RUnten,RSenso
            ren[2]:self.RRechts,RSensoren[3]:self.ROben}
90
            self.RbckSensoMelder={RSensoren[0]:self.ROben,RSensoren[1]:self.RLinks,RSensor
            en[2]:self.RUnten,RSensoren[3]:self.RRechts}
91          #Die Melder werden den Streckenabschnitten zugeordnet
92
            self.reportBlock={self.LLinks:self.LLinks.getReporter(),self.LOben:self.LOben.
            getReporter(),self.LRechts:self.LRechts.getReporter(),self.LUnten:self.LUnten.
            getReporter(),self.RLinks:self.RLinks.getReporter(),self.ROben:self.ROben.getR
            eporter(),self.RRechts:self.RRechts.getReporter(),self.RUnten:self.RUnten.getR
            eporter()}
93          #Hier wird die Reihenfolge festgehalten
94
95
96          print self.reportBlock
97          # Die DCC Adresse der Lok(In JMRI Lokpark nachschaubar).
98          # Wenn die Lange Adresse vergeben wurde muss "False" zu "True" geändert werden
99          self.positionendict={self.LLinks:Loks[0],self.RLinks:Loks[1]}#Hier die
            StartPositionen der Lok eingeben, PRO BLOCK NUR 1 ZUG
100
101         for key in self.positionendict:
102             throttle=self.getThrottle(self.positionendict[key], False)
103             if (key == self.LLinks or key == self.LRechts or key == self.LOben or
                key == self.LUnten):
104                 self.LinkesLayout.append(throttle)
105             else:
106                 self.RechtesLayout.append(throttle)
107
108         for i in LSensoren:
109             sensor=sensors.provideSensor("AR"+str(i))
110             self.LsensorDict[i]=sensor
111
112         for i in RSensoren:
113             sensor=sensors.provideSensor("AR"+str(i))
114             self.RsensorDict[i]=sensor
115
116
117         for i in LMelder:
118             reporter=reporters.provideReporter("IR"+str(i))
119             self.LreportDict[i]=reporter
120
121         for i in RMelder:
122             reporter=reporters.provideReporter("IR"+str(i))
123             self.RreportDict[i]=reporter
124
125
126
127         print "Sensoren und Melder Fertig"
```

```
128             #Die Throttles werden erstellt und dem ThrottleDict hinzugefügt
129             for i in Loks:
130                 throttle=self.getThrottle(i, False)
131                 self.throttledict[i]=throttle
132             print "Dies sind die Züge im Roster"
133             for keys,values in self.throttledict.items():
134                 print(keys)
135                 print(values)
136 
137             print "Und hier Starten Sie"
138             for keys,values in self.positionendict.items():
139                 print(keys)
140                 print(values)
141             # setze die lok in bewegung
142 
143             forward = True #Ändern für Vorwärts oder Rückwärts
144             #Kann auch in Listen verwaltet werden und dann seperat per Lok zugeordnet
                werden. Lohnt sich jedoch nur bei Großen
145             #Schienenlayouts da ansonsten zusammenstöße passieren könnten.
146             #Dieses Skript ist darauf ausgelegt, dass wenn die Richtung geändert wird.
                Dies Bei ALLEN Loks Pro Layout gleichzeitig geschieht
147             # Damit zusammenstöße verhindert werden
148 
149             for key in self.throttledict:
150                 self.throttledict[key].setIsForward(forward)
151 
152             # Warte 1 Sekunde und Setze die Geschwindigkeit(0-1, 1 entspricht 100%)
153             self.waitMsec(1000)
154             print "Setze Geschwindigkeit"
155             for key in self.throttledict:
156                 self.throttledict[key].setSpeedSetting(0.5)
157             #Aktuellen Standort der Lok Ermitteln
158             print "Aktivierung des Ersten Sensors Abwarten"
159             for key in self.LreportDict:
160                 self.LreportDict[key].setReport("")
161             return
162             for key in self.RreportDict:
163                 self.RreportDict[key].setReport("")
164             return
165 
166         def aktivSetzen(self, b, s,r,l):  #b ist der Block, s ist der Sensor,r der
            Reporter,l die Lok
167             b.goingActive()
168             r.setReport(str(l)+" Enter")
169             print r,"Aktiviert"
170             print s,"Aktiviert"
171             print b, "Besetzt"
172             return
173 
174         def inaktivSetzen(self,b,r,l): #b ist der Block, r der Reporter,l die Lok
175             b.goingInactive()
176             r.setReport(str(l)+" exits")
177             return
178 
179 
180         #Checkt ob ein Sensor Aktiv wird und gibt den Entsprechenden Sensor aus nachdem
            er wieder auf Inaktiv geschaltet wird
181         #So werden sensoren nur für den Bruchteil einer Sekunde Aktiv geschaltet
182         #Dies ist wichtig falls 2 Loks zur selben zeit Sensoren Aktivieren
183         def LanySensor(self):
184             global letzterSensor
185             for key in self.LsensorDict:
186                 if(self.LsensorDict[key].getState()==ACTIVE):
187                     self.LsensorDict[key].setState(INACTIVE)
188                     letzterSensor=key
189                     print "key:" + str(key)
190                     print "letzterSensor:"+str(letzterSensor)
191                     return letzterSensor
192             return False
193 
194         def RanySensor(self):
195             global letzterSensor
```

```
196             for key in self.RsensorDict:
197                 if(self.RsensorDict[key].getState()==ACTIVE):
198                     self.RsensorDict[key].setState(INACTIVE)
199                     letzterSensor=key
200                     print "key:" + str(key)
201                     print "letzterSensor:"+str(letzterSensor)
202                     return letzterSensor
203             return False
204 
205         def handle(self):
206             global letzterSensor
207             # handle() wird wiederholt ausgeführt bis das Programm beendet wird.
208             print "Es beginnt handle(self)"
209             self.waitMsec(500)
210             #Das Linke Layout wird überprüft
211             if(self.LinkesLayout[0].getIsForward()):
212                 #for i in range(len(self.LinkesLayout)):
213                     #Alle Loks werden auf dieselbe Richtung wie Lok 1 gebracht
214                     #self.LinkesLayout[i].setIsForward(True)
215                 #Falls ein Sensor Signal gibt
216                 if (self.LanySensor()):
217                     #gehe die Sensoren durch und schaue welcher der Letzte Sensor war
                        der Signal gegeben hat
218                     for key in self.LsensorDict:
219                         if(key==letzterSensor):
220                             #Der Letzte Block auf dem die Lok war wird gesucht
221                             letzterBlock=self.checkPrev(self.LfwdSensoMelder[key],True)
222                             #Dem PositionenDict wird die Aktuelle Position der Bahn
                                mitgeteilt und die Alte position wird gelöscht
223 
                                self.positionendict[self.LfwdSensoMelder[key]]=self.positionen
                                dict.pop(letzterBlock)
224                             #Der Block, der Sensor , der Reporter und die Lok werden an
                                aktivSetzen() übergeben
225 
                                self.aktivSetzen(self.LfwdSensoMelder[key],key,self.reportBloc
                                k[self.LfwdSensoMelder[key]],self.positionendict[self.LfwdSens
                                oMelder[key]])
226                             print self.positionendict[self.LfwdSensoMelder[key]]
227                             #Der Block, der Reporter und die Lok werden inaktiv setzen
                                übergeben
228 
                                self.inaktivSetzen(letzterBlock,self.reportBlock[letzterBlock]
                                ,self.positionendict[self.LfwdSensoMelder[key]])
229             else:
230                 for i in range(len(self.LinkesLayout)):#Alle Loks werden auf dieselbe
                    Richtung wie Lok 1 gebracht
231                     self.LinkesLayout[i].setIsForward(False)
232                 if (self.LanySensor()):
233                     #gehe die Sensoren durch und schaue welcher der Letzte Sensor war
                        der Signal gegeben hat
234                     for key in self.LsensorDict:
235                         if(key==letzterSensor):
236                             #Der Letzte Block auf dem die Lok war wird gesucht
237                             letzterBlock=self.checkPrev(self.LbckSensoMelder[key],False)
238                             #Dem PositionenDict wird die Aktuelle Position der Bahn
                                mitgeteilt und die Alte position wird gelöscht
239 
                                self.positionendict[self.LbckSensoMelder[key]]=self.positionen
                                dict.pop(letzterBlock)
240                             #Der Block, der Sensor , der Reporter und die Lok werden an
                                aktivSetzen() übergeben
241 
                                self.aktivSetzen(self.LbckSensoMelder[key],key,self.reportBloc
                                k[self.LbckSensoMelder[key]],self.positionendict[self.LbckSens
                                oMelder[key]])
242                             print self.positionendict[self.LbckSensoMelder[key]]
243                             #Der Block, der Reporter und die Lok werden inaktiv setzen
                                übergeben
244 
                                self.inaktivSetzen(letzterBlock,self.reportBlock[letzterBlock]
                                ,self.positionendict[self.LbckSensoMelder[key]])
```

```
245
246             #Das Rechte Layout wird überprüft
247             if(self.RechtesLayout[0].getIsForward()):
248                 for i in range(len(self.RechtesLayout)):
249                     #Alle Loks werden auf dieselbe Richtung wie Lok 1 gebracht
250                     self.RechtesLayout[i].setIsForward(True)
251                 #Falls ein Sensor Signal gibt
252                 if (self.RanySensor()):
253                     #gehe die Sensoren durch und schaue welcher der Letzte Sensor war
                        der Signal gegeben hat
254                     for key in self.RsensorDict:
255                         if(key==letzterSensor):
256                             #Der Letzte Block auf dem die Lok war wird gesucht
257                             letzterBlock=self.checkPrev(self.RfwdSensoMelder[key],True)
258                             print "letzterBlock:---------------"
259                             print self.RfwdSensoMelder[key]
260                             print letzterBlock
261                             #Dem PositionenDict wird die Aktuelle Position der Bahn
                                mitgeteilt und die Alte position wird gelöscht
262
                                self.positionendict[self.RfwdSensoMelder[key]]=self.positionen
                                dict.pop(letzterBlock)
263                             #Der Block, der Sensor , der Reporter und die Lok werden an
                                aktivSetzen() übergeben
264
                                self.aktivSetzen(self.RfwdSensoMelder[key],key,self.reportBloc
                                k[self.RfwdSensoMelder[key]],self.positionendict[self.RfwdSens
                                oMelder[key]])
265                             print self.positionendict[self.RfwdSensoMelder[key]]
266                             #Der Block, der Reporter und die Lok werden inaktiv setzen
                                übergeben
267
                                self.inaktivSetzen(letzterBlock,self.reportBlock[letzterBlock]
                                ,self.positionendict[self.RfwdSensoMelder[key]])
268             else:
269                 for i in range(len(self.RechtesLayout)):#Alle Loks werden auf dieselbe
                    Richtung wie Lok 1 gebracht
270                     self.RechtesLayout[i].setIsForward(False)
271                 if (self.RanySensor()):
272                     #gehe die Sensoren durch und schaue welcher der Letzte Sensor war
                        der Signal gegeben hat
273                     for key in self.RsensorDict:
274                         if(key==letzterSensor):
275                             #Der Letzte Block auf dem die Lok war wird gesucht
276                             letzterBlock=self.checkPrev(self.RbckSensoMelder[key],False)
277                             #Dem PositionenDict wird die Aktuelle Position der Bahn
                                mitgeteilt und die Alte position wird gelöscht
278
                                self.positionendict[self.RbckSensoMelder[key]]=self.positionen
                                dict.pop(letzterBlock)
279                             #Der Block, der Sensor , der Reporter und die Lok werden an
                                aktivSetzen() übergeben
280
                                self.aktivSetzen(self.RbckSensoMelder[key],key,self.reportBloc
                                k[self.RbckSensoMelder[key]],self.positionendict[self.RbckSens
                                oMelder[key]])
281                             print self.positionendict[self.RbckSensoMelder[key]]
282                             #Der Block, der Reporter und die Lok werden inaktiv setzen
                                übergeben
283
                                self.inaktivSetzen(letzterBlock,self.reportBlock[letzterBlock]
                                ,self.positionendict[self.RbckSensoMelder[key]])
284             # Startet Skript von Vorne
285             print "Ende des Loops"
286             return 1
287             # Beenden Über JMRI-Laufzeitmonitor und Fahrregler der Lok
288
289
290     # end of class definition
291
292     # start one of these up
293     Belegtstatuts().start()
```

# I. UNTERRICHTSENTWURF

## I.1. Didaktisch-methodische Handreichung

**Kernanliegen:** Die Schülerinnen und Schüler erarbeiten aus einem realweltlichen Kontext ein informatisches Modell einer Weichenschaltung. Hierfür werden Stellwerke und deren Entwicklung an Beispielen aufgezeigt und von den Schülerinnen und Schülern nachgebaut. Die Schülerinnen und Schüler müssen eine Weichenschaltung in der Arduino IDE entwickeln, die entweder mithilfe von einem Steckbrett eine manuelle Schaltung realisiert oder eine Möglichkeit finden, die Signale von verbundenen Sensoren zu nutzen um Weichen automatisch zu schalten. Zudem muss ein Algorithmus entwickelt werden der dem Arduino signalisiert, dass bei gegebenem Signal die Weiche geschaltet werden muss.
**Paradigma:** Physical ☒ Plugged ☒ Unplugged ☐
**Unterrichtliche Einbettung (Vorkenntnisse, Fortführung):** Die Schülerinnen und Schüler sollten Grundkenntnisse im Umgang mit dem Arduino und der Arduino IDE haben. Die Unterrichtseinheit kann unabhängig davon stattfinden ob der Kontext Eisenbahn bereits zuvor im Unterricht verwendet wurde.

**Jahrgangsstufe:** Q1(Qualifikationsphase 1 der gymnasialen Oberstufe)
**Dauer:**3 Unterrichtseinheiten à 45 Minuten
**Inhaltsfeld:** Algorithmen, Informatiksysteme
**Zentrale Kompetenzbereiche:** Modellieren, Darstellen und Interpretieren, Implementieren, Argumentieren,
**Inhaltlicher Schwerpunkt:** Nutzung von Informatiksystemen, Analyse, Entwurf und Implementierung von Algorithmen
**Gesellschaftliche Dimensionen:** Technik, Wirtschaft

Beschreibung: Diese Unterrichtseinheit soll den Schülerinnen und Schülern Kontext Eisenbahn, in diesem Fall genauer die Weichenschaltung verdeutlichen. Der inhaltliche Schwerpunkt ist die Nutzung von Informatiksystemen aus dem Inhaltsfeld der Informatiksysteme sowie ”Analyse, Entwurf und Implementierung von Algorithmen” des Inhaltsfelds Algorithmen. Die Schülerinnen und Schüler sollten Grundkenntnisse im Umgang mit dem Arduino sowie der Arduino IDE und zugehöriger Programmiersprache C/C++ besitzen. Die Schülerinnen und Schüler lernen die geschichtliche Entwicklung von Stellwerken zur Weichenschaltung kennen und sollen eine automatisierte Weichenschaltung oder eine Stellwerk ähnliche Schaltung per Steckbrett realisieren. Anschließend werden die Vor- und Nachteile beider Varianten besprochen. Besonders im Hinblick auf Sicherheitsaspekte der Automatisierung. In der zweiten Unterrichtseinheit wird die entworfene Weichenschaltung auf eine Variante weiterentwickelt, die in der Lage ist die Weichen per JMRI über die DCC-Zentrale zu schalten.

**Übersicht über den geplanten Verlauf des Unterrichts**

1. 1.Stunde
    a) Einführung in den Kontext und die Problemstellung
    b) Erarbeitung einer Schaltung und eines Algorithmus zur Weichenschaltung
    c) Sicherung
    d) Erarbeitungsphase für Vor- und Nachteile der vorgestellten Lösungen
    e) Sicherung und Sicherheitsaspekte von Automatisierung
2. 2.Stunde
    a) Wiederholung der Ergebnisse
    b) Modifizierung des Programms
    c) Sicherung

Tabelle I.1: *Artikulationsschema UE1 90 Min.: Entwicklung einer Arduino Weichenschaltung für Modelleisenbahnen*

| **Phase** (Zeitangaben) | **Unterrichtsinhalte** | **Sozial- / Arbeitsformen** | **Material / Medien** | **Didaktisch-methodischer Kommentar** |
|---|---|---|---|---|
| **Einstieg** 20 Min. | Eine Modelleisenbahn ist im Kursraum aufgebaut und wird von der Lehrkraft mit einer Arduino basierten DCC-Zentrale gesteuert und vorgeführt. Das Schienenlayout enthält zwei Weichen. Der Lehrkörper erläutert die Funktionsweise der Weichenschaltung. (Je nach Weiche verschieden). Ein kurzes Stromsignal auf das eine Anschlusskabel schaltet die Weichen, ein Stromsignal auf das andere Kabel lässt die Weiche zurückschalten. Es werden reale Beispiele von Stellwerken vorgeführt und die geschichtliche Entwicklung mit einbezogen. Die Frage wie die Entwicklung weitergehen könnte soll aufkommen. Das Stichwort ”Automatisierung” sollte fallen oder erwähnt werden. Die zentrale Unterrichtsfrage ”Wie können diese Weichen per Arduino oder automatisiert per Sensor geschaltet werden?” und die nachfolgende Frage der Vor- und Nachteile beider Systeme werden genannt. Die SuS dürfen entscheiden ob welche der Lösungen sie bevorzugen und werden dafür in Gruppen à 3 Personen eingeteilt. Je nach bevorzugter Umsetzung und können sich das benötigte Material abholen. Die Schülerinnen und Schüler erhalten einen Überblick über das Stundenziel und wissen was von Ihnen verlangt wird. | Plenum | Modell-eisenbahn, Arduino, Arduino Zubehör (Steck-bretter, Druck-knöpfe, Verbin-dungs-kabel, Wider-stände, LED-Lampen, etc.) | Es soll die Problemstellung vorgestellt werden, die die Schülerinnen und Schüler in der Unterrichtseinheit lösen sollen. Die Schülerinnen und Schüler sollen einen geschichtlichen Bezug zu der Aufgabe erhalten und wenn nötig mit dem Stichwort ”Automatisierung” in die richtige Richtung gelenkt werden. Die freie Auswahl des genauen Themas der Stunde soll die Schülerinnen und Schüler eigenverantwortlicher und motivierter handeln lassen.<br>Falls keine Gruppe die automatisierte Version übernimmt, muss dies in der Abschlussdiskussion berücksichtigt werden wo über die Möglichkeiten und Risiken gesprochen werden soll.<br>Die Arbeit in Dreiergruppen soll die Kompetenzen im Bereich ”Kommunizieren und Kooperieren” fördern. |

| | | | | |
|---|---|---|---|---|
| **Erarbeitung I:**<br>35 Min. | Die Schülerinnen und Schüler erarbeiten anhand des Materials was ihnen zur Verfügung gestellt wird die Schaltungen(Arbeitsblatt ”Weichenschaltung per Steckplatte” oder ”Automatisierte Weichenschaltung”). Je nach Thema sollte den Schülerinnen und Schülern auffallen, dass die Arbeit gut aufteilbar ist. Einer übernimmt die analoge Arbeit und setzt die Schaltung bzw. den Arduino und die Sensoren zusammen, die anderen beiden arbeiten im Pair Programming den Algorithmus aus. | Gruppenarbeit/ Paarprogrammierung | Arbeitsblatt ”Weichenschaltung per Steckplatte” oder ”Automatisierte Weichenschaltung”, Arduino, Arduino Zubehör (Steckbretter, Druckknöpfe, Verbindungskabel, Widerstände, LED-Lampen, etc.), Computer | SuS führen das Projekt anhand der Arbeitsblätter eigenständig durch, wenden bereits erlerntes Wissen an und bauen ihre eigenen Kompetenzen weiter aus. Die offene Aufgabenstellung ermöglicht verschiedene Lösungen auf unterschiedlichen Niveaus und dadurch differenziertes arbeiten. Die Schülerinnen und Schüler helfen sich in den Gruppen gegenseitig und müssen zur Umsetzung des Projektes miteinander kommunizieren. Die Lehrkraft kann bei Bedarf individuelle Hilfestellungen geben |
| **Sicherung I:**<br>20 Min. | Die einzelnen Gruppen stellen ihre Lösungen, moderiert von der Lehrkraft vor und führen sie an der Modelleisenbahn aus. Dabei kann der Programmcode über den Beamer vorgestellt und per Plenum, wenn nötig, verbessert werden.<br>Falls keine Gruppe die automatisierte Version mit den Sensoren gewählt hat, stellt die Lehrkraft diese vor. | Plenum | s.o., Beamer | Die Schülerinnen und Schüler bilden ihre Kompetenzen im Bereich Kommunizieren und Argumentieren weiter aus. Zusätzlich können die Schülerinnen und Schüler, wenn nötig, der vorstellenden Gruppe Hilfe beim Debuggen ihrer Programme geben, sodass ebenfalls die Kompetenzen im Bereich Implementierung ausgebildet werden. Das gemeinsame Zusammenfassen der Ergebnisse dient der Verständniskontrolle und ermöglicht für alle Schülerinnen und Schüler das Erreichen des Stundenziels. |

| | | | | |
|---|---|---|---|---|
| **Erarbeitung II:** 5 Min. | Die Gruppen bekommen den Arbeitsauftrag die Vor- und Nachteile der beiden Möglichkeiten zur Weichenschaltung zu besprechen. Auch im Hinblick auf eine Umsetzung für reale Stellwerke. | Gruppenarbeit | - | Die Schülerinnen und Schüler betrachten den Kontext auch von einem gesellschaftlichen Aspekt aus. Die Gruppenarbeit fördert dabei weiterhin die Ausbildung der Kompetenzen im Bereich Kommunikation und auch Argumentieren. |
| **Sicherung II:** 10 Min. | Die Vor- und Nachteile der beiden Möglichkeiten werden im Plenum gesammelt. Dabei soll vorallem auch auf Sicherheitsrisiken eingegangen werden die eine Automatisierung von Stellwerken mit sich bringt. Sollten die Schülerinnen und Schüler diese Nachteile der automatisierten Version nicht einbringen muss die Lehrkraft die Schülerinnen und Schüler zu diesen Punkten hinleiten. | Gruppenarbeit | - | Es wird hier Sichergestellt, dass sich die Schülerinnen und Schüler mit den Sicherheitsrisiken einer etwaigen Automatisierung von Zügen oder Weichenschaltungen auseinandersetzen. Aber auch die Vor- und Nachteile für die weitere Umsetzung im Informatikunterricht soll beleuchtet werden. |

Tabelle I.2: *Artikulationsschema UE2 45 Min.: Erweiterung der Arduino Weichenschaltung zur Schaltung per JMRI*

| **Phase** (Zeitangaben) | **Unterrichtsinhalte** | **Sozial-/ Arbeitsformen** | **Material / Medien** | **Didaktisch-methodischer Kommentar** |
|---|---|---|---|---|
| **Einstieg** 10 Min. | Die Lehrkraft moderiert ein Unterrichtsgespräch in dem ein Rückblick auf die erarbeiteten Weichenschaltungen erfolgt. Die Lehrkraft stellt eine weitere Variante der Stellwerke, die elektronischen Stellwerke, genauer vor. Die Lehrkraft erläutert kurz das erstellte Layout in JMRI über das die Weichen am Ende der Stunde gesteuert werden soll. | Plenum | Beamer | Im Unterrichtsgespräch rekapitulieren die Schülerinnen und Schüler die entwickelten Schaltungen mit ihren Vor- und Nachteilen um in dieser Unterrichtseinheit an das gelernte anzuknüpfen. Die Frage nach einem besseren System kommt auf. Die Vorführung der zu erreichenden Steuerung verschafft den SuS Klarheit über die Ziele der Stunde. |

| | | | | |
|---|---|---|---|---|
| **Erarbeitung I:** 20 Min. | Die Schülerinnen und Schüler entwickeln aus ihrer Schaltung der letzten Einheit, mithilfe des Arbeitsblattes ”Weichenschaltung per JMRI” eine Schaltung die mithilfe der Arduino-DCC-Zentrale die Weichen schalten kann. | Paarprogrammierung | Arbeitsblatt ”Weichenschaltung per JMRI” , Arduino, Computer | Die Schülerinnen und Schüler werden bewusst nicht in die Gruppen der letzten Einheit eingeteilt, sondern es wird in der Paarprogrammierung gearbeitet da hier keine weitere Arbeitsteilung stattfinden kann. |
| **Sicherung I:** 15 Min. | Es werden Gruppen gebildet à 3 Paaren in denen die Ergebnisse vorgestellt werden und über die Ergebnisse diskutiert wird. Diese Phase sollte nicht mehr als 8 Minuten dauern. Anschließend führen die Gruppen ihre Lösung im Plenum an der Modelleisenbahn vor. | Gruppenarbeit / Plenum | Modelleisenbahn, Computer, die Entwickelten Schaltungen der SuS, Arduino | Die Gruppenphase der Ergebnissicherung sichert die Beteiligung der Schülerinnen und Schüler und lässt jede Gruppe eine funktionierende Version erarbeiten die vorgestellt werden kann. Da nur eine Modelleisenbahn mit Weichen zur Verfügung steht müssen die Tests im Plenum stattfinden. Das Arbeitsblatt gibt die zu benutzenden Pins vor, da die Änderung in JMRI für eine so kurze Sicherungsphase zu viel Zeit in Anspruch nehmen würde |

# I.2. Arbeitsblatt Weichenschaltung per Steckplatte

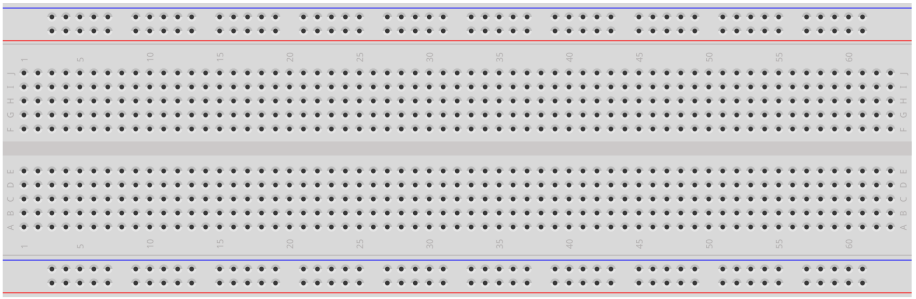
Weichenschaltung per Steckplatine
Lars Bockstette

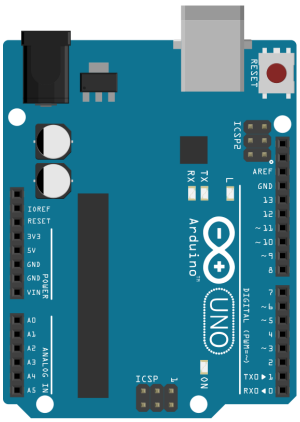
Datum:_____ Name:_____

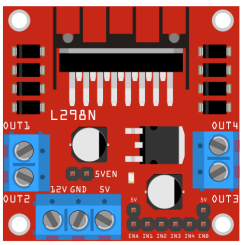
*Entwickeln Sie ein Programm in der Arduino IDE die mithilfe eines Arduinos und einem Motor Driver eine Weiche schalten kann*

1 Nutzen Sie zur Lösung dieses Problems das bereitgestellte Steckbrett und entwickeln Sie auf diesem eine Schaltung zur Signalübertragung an den Arduino

Zeichnen Sie ihre Lösung in die untenstehende Grafik ein.






fritzing

Die Eingänge "IN1" und "IN2" des Motor Drivers steuern die Ausgänge "OUT1" und "OUT2", und die Eingänge "IN3" und "IN4" entsprechend die Ausgänge "OUT3" und "OUT4".

Die Schaltungen werden in der Überprüfung an der bereitgestellten Weiche getestet und auch erst dann wird der Motor Driver an den Strom angeschlossen.

**Hinweise**:

- Achten Sie darauf den Strom nur für Kurze Zeit (<1s) auf die Weichen zu geben, da diese bei längerer Belastung sehr heiß werden.
- Achten Sie darauf 1k Ohm Widerstände in den Schaltungen zu verwenden.
- Als Eingang für Sensoren eignen sich die PINS A0 bis A5 da diese standardmäßig als Eingang deklariert sind. Ausgänge müssen in der Arduino IDE als solche deklariert werden.
- Weitere Hilfestellungen zu den Funktionen sind falls nötig auf der Rückseite.

Das Internet steht zur Recherche von Funktionen und Schaltungen zur Verfügung.

**2** Falls Sie hilfe bei der Programmierung benötigen ist hier eine Liste aller Nötigen befehle zur Umsetzung

pinMode(Integer,OUTPUT/INPUT) // Legt den entsprechenden Pin des Arduinos als Eingang oder Ausgang fest.

analogRead(Integer) // Liest den Stromwert vom entsprechenden Pin des Arduinos aus.

digitalWrite(Integer, LOW/HIGH) // Schaltet den entsprechenden Pin des Arduinos auf Stromausgabe oder keine Stromausgabe.

delay(x) // lässt das Programm x Millisekunden Pausieren.

**Tipps**: Die Weiche kann nicht in den Zustand schalten in dem sie sich Bereits befindet. Darum eignen sich hier besonders gut **Switch-Anweisungen** die auf einzelne Funktionen verweisen.

# I.3. Arbeitsblatt automatisierte Weichenschaltung

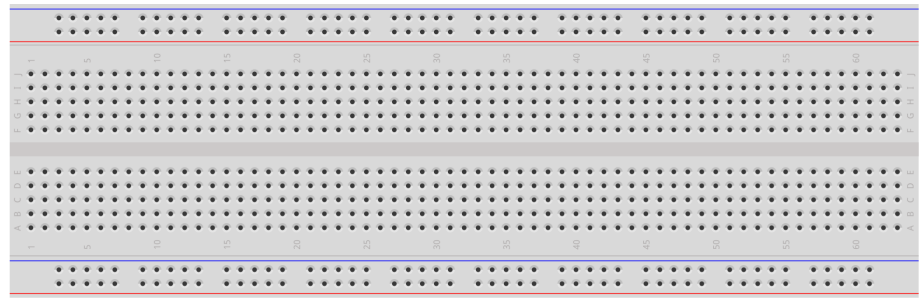
Automatisierte Weichenschaltung
Lars Bockstette

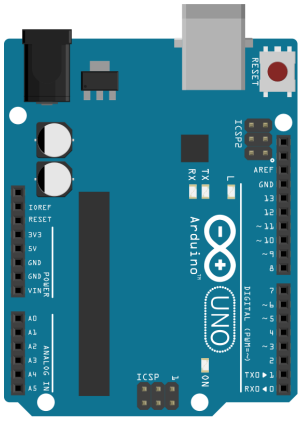
Datum:_____ Name:_____

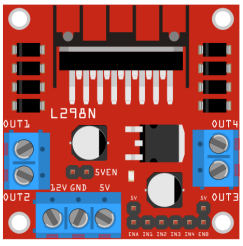
*Entwickeln Sie ein Programm in der Arduino IDE die mithilfe eines Arduinos und einem Motor Driver eine Weiche schalten kann*

**1** Nutzen Sie zur Lösung dieses Problem die bereitgestellten Sensoren eine Steckplatte und den Arduino mit seinem Motor Driver

Zeichnen Sie ihre Lösung in die untenstehende Grafik ein.






fritzing

Die Eingänge "IN1" und "IN2" des Motor Drivers steuern die Ausgänge "OUT1" und "OUT2", und die Eingänge "IN3" und "IN4" entsprechend die Ausgänge "OUT3" und "OUT4".

Die Schaltungen werden in der Überprüfung an der bereitgestellten Weiche getestet und auch erst dann wird der Motor Driver an den Strom angeschlossen.

**Hinweise**:

- Achten Sie darauf den Strom nur für Kurze Zeit (<1s) auf die Weichen zu geben, da diese bei längerer Belastung sehr heiß werden.
- Die Sensoren die Sie benutzen benötigen 5V Strom und eine Erdung. Der Arduino liefert diese über die Pins auf der Linken Seite "5V" und GND.
- Als Eingang für Sensoren eignen sich die PINS A0 bis A5 da diese Standardmäßig als Eingang deklariert sind. Ausgänge müssen in der Arduino IDE als solche deklariert werden.
- Weitere Hilfestellungen zu den Funktionen sind falls nötig auf der Rückseite.

Das Internet steht zur Recherche von Funktionen und Schaltungen zur Verfügung.

**2** Falls Sie Hilfe bei der Programmierung benötigen ist hier eine Liste aller nötigen Befehle zur Umsetzung

pinMode(Integer,OUTPUT/INPUT) // Legt den entsprechenden Pin des Arduinos als Eingang oder Ausgang fest.

analogRead(Integer) // Liest den Stromwert vom entsprechenden Pin des Arduinos aus.

digitalWrite(Integer, LOW/HIGH) // Schaltet den entsprechenden Pin des Arduinos auf Stromausgabe oder keine Stromausgabe.

delay(x) // lässt das Programm x Millisekunden Pausieren.

**Tipps**: Die Weiche kann nicht in den Zustand schalten in dem sie sich Bereits befindet. Darum eignen sich hier besonders gut **Switch-Anweisungen** die auf einzelne Funktionen verweisen.

# I.4. Arbeitsblatt Weichenschaltung per JMRI

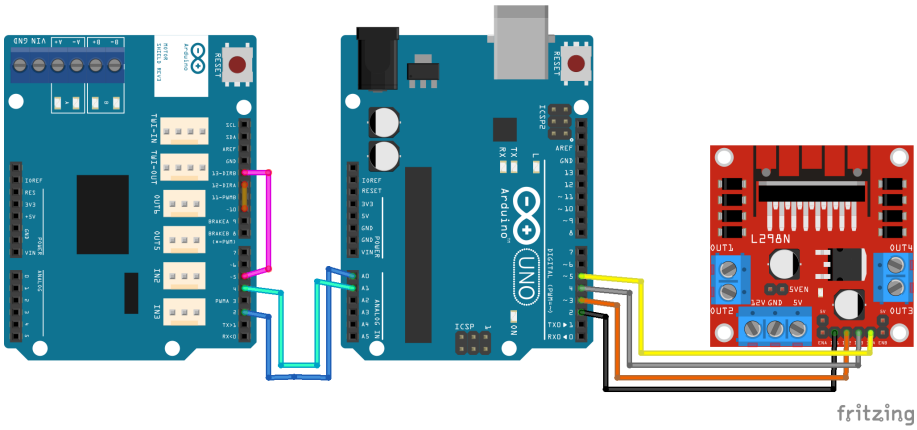
Weichenschaltung per Java Model Railroad Interface
Lars Bockstette

Datum:_____ Name:_____

*Entwickeln Sie ein Programm in der Arduino IDE die mithilfe der Arduino DCC-Zentrale und einem Motor Driver eine Weiche schalten kann*

**1** Entwickeln sie aus den Lösungen der letzten Stunde einen Algorithmus in der Arduino IDE der die geforderte Weichenschaltung umsetzt

In der untenstehenden Grafik sehen Sie die Verbindung zwischen der DCC-Zentrale(links), dem Arduino(mitte) welchen Sie programmieren sollen und dem Motor Driver(rechts). Die DCC-Zentrale sendet aus Pin 2 das Signal für die 1. Weiche und aus Pin 4 das Signal für die 2. Weiche.




Die Schaltungen werden in der Überprüfung an der bereitgestellten Weiche getestet und auch erst dann wird der Motor Driver an den Strom angeschlossen.

**Hinweis**:

- Achten Sie darauf den Strom nur für Kurze Zeit (¡1s) auf die Weichen zu geben, da diese bei längerer Belastung sehr heiß werden.

Das Internet steht zur Recherche von Funktionen und Schaltungen zur Verfügung.

Beispielhafte Umsetzungen des Greenfoot Codes für die Arbeitsblätter befinden sich in B.1. Hier folgen die zugehörigen beispielhaften Umsetzungen der technischen Komponente der Arbeitsblätter für die erste Unterrichtseinheit, da in der zweiten Unterrichtseinheit keine technische Umsetzung nötig ist.

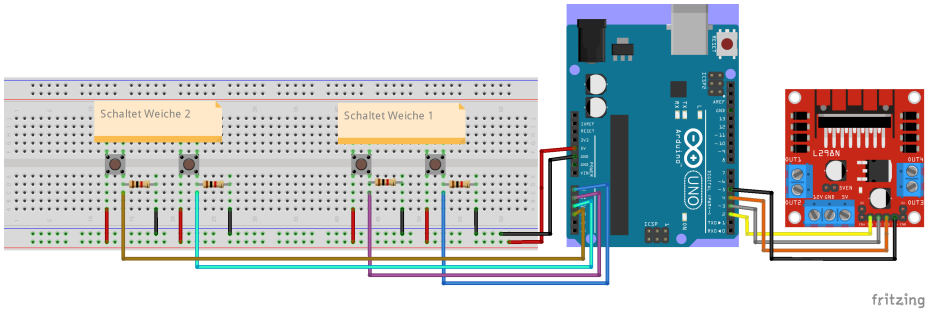



Abbildung I.1: *Steckplatine für die Weichenschaltung von zwei Weichen*

Weitere mögliche Schaltungen wurden in Kapitel 3.2 behandelt.
Die Umsetzung mit den Sensoren könnte wie folgt aussehen:

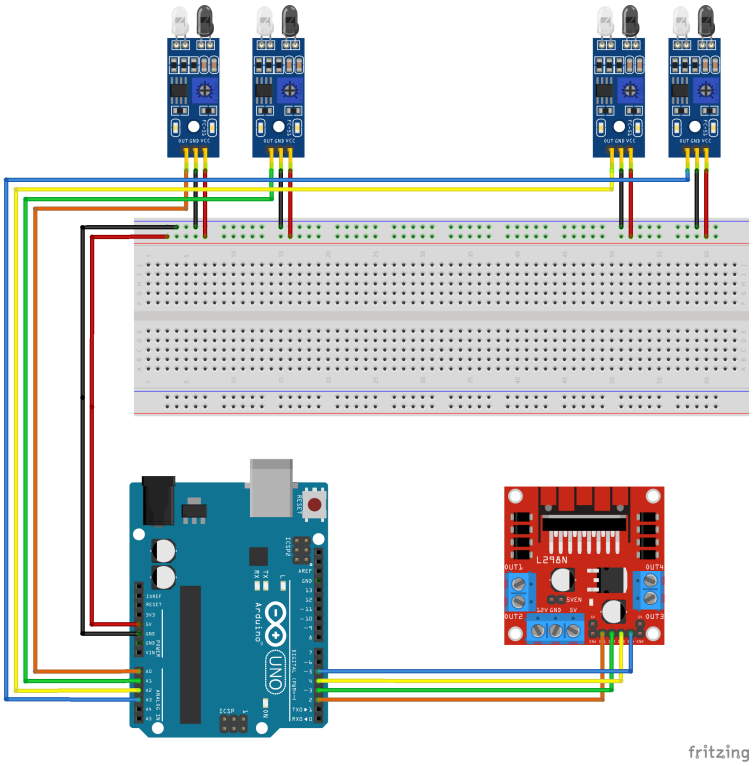



Abbildung I.2: *Steckplatine für die Weichenschaltung mit Sensoren*

Die Schülerinnen und Schüler müssen dabei die Sensoren passend an den Weichen positionieren. In diesem Beispiel steuern die Sensoren an A0 und A1 die 1.Weiche und die Sensoren an A2 und A3 die 2.Weiche.