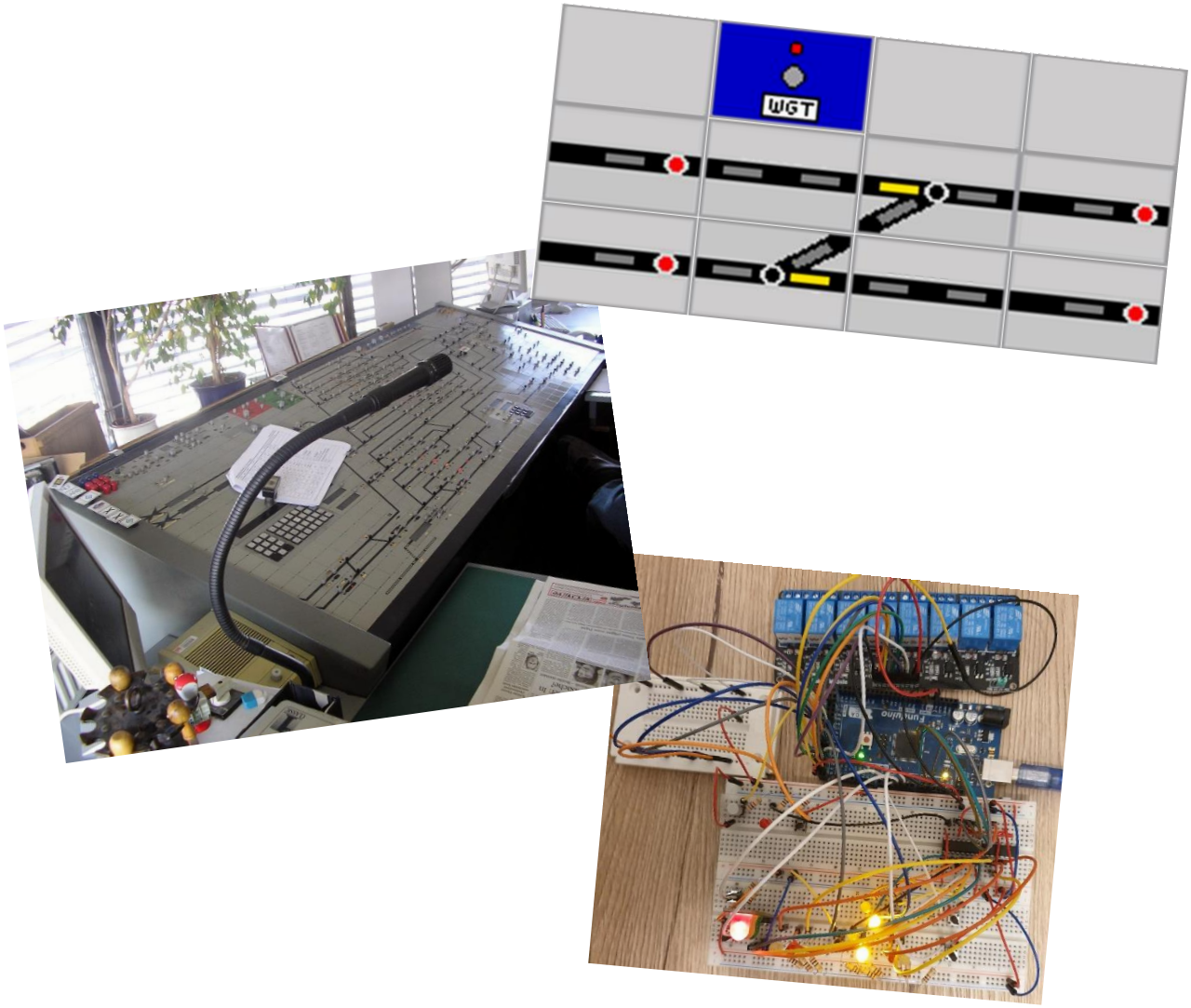
jugend forscht

SpDrS60-Stellwerkssimulation zur Verwendung in beliebigen Schienensystemen



Lennart Klüner

Informatik/Mathematik

Jugend forscht 2023 NRW

Quellen (v. o.) Guido Scholz, spdrs60.sourceforge.net *(abgerufen am 13.01.2023) mit persönlicher Erlaubnis zur Verwendung und Bearbeitung;* commons.wikimedia.org*, abgerufen am 13.01.2023,* HP00 *in der* *Wikipedia auf Deutsch*, CC BY-SA 2.0 DE*, unten eigenes Foto*

Inhalt

Der gesamte Quellcode dieses Projekt ist auf GitHub unter *https://github.com/lk-16/Arduino-signal-box-simulation* gehostet und ist Open-Source. Bei Formatierungsproblemen findet sich unter diesem *Link* eine online Version meiner Arbeit.

Kurzfassung

In meinem Projekt möchte ich meine Stellwerkssimulation aus dem Projekt „Entwicklung einer Simulationssoftware für ein Gleisbildstellwerk“ erweitern. Als Vorbild habe ich das Spurplandrucktastenstellwerk 60 (SpDrS60) der Deutschen Bahn gewählt. Die entwickelte Software simuliert dieses Stellwerk und bietet die Möglichkeit die teure Relaistechnik der SpDrS60-Stellwerke mit Microcontrollertechnik zu ersetzen. Die Fahrwege der Züge werden bei diesem Stellwerkstyp durch das Drücken einer Start- und einer Zieltaste eingestellt.

In diesem Projekt habe ich diese Steuerung der Fahrwege erneuert, so dass das Programm auf verschiedene Stellpulte angewendet werden kann. Der Mikrocontroller soll dazu während der Laufzeit des Programms selbstständig den Fahrweg für die Züge zwischen Start- und Zieltaste bestimmen, so dass nicht mehr alle möglichen Verbindungen zwischen Start- und Zieltasten für jedes Stellpult gesondert programmiert werden müssen.

Das Programm läuft auf einem Arduino Mikrocontroller. Dieser ermittelt mit Hilfe des Backtrackingverfahrens und dem Gleisplan des Stellwerks einen Fahrweg zwischen gedrückter Start- und Zieltaste und simuliert so ein Gleisbildstellpult mit Relaistechnik.

1 Einleitung

1.1 Projektziel

In diesem Projekt möchte ich aufbauend auf den Ergebnissen des letzten Projekts die Fahrstraßensteuerung grundlegend erneuern. So ist mir zum Beispiel bei der Anpassung der Software für mein eigenes Teststellpult aufgefallen, dass das Anlernen der Steuerung der Fahrstraßen wenig komfortabel und sehr aufwendig wird, insbesondere bei größeren Stellwerken. Aus diesem Grund soll in diesem Projekt die Programmierung einer Fahrstraßensteuerung im Vordergrund stehen. Der Algorithmus soll sich selbstständig auf verschiedene Gleissituationen einfacher und schneller anpassen können. Es ist mein Ziel, dass nicht jede Fahrstraße von einem Menschen erkannt und implementiert werden muss. So möchte ich besonders bei größeren Stellwerkssituationen den Aufwand zur Erstellung der Software für das Stellpult verringern.

1.2 Zum Stellwerk

Beim von mir ausgewählten Stellwerk der Bauart SpDrS60 handelt es sich um ein Spurplanstellwerk von Siemens. Bei dieser Bauart steuert der Fahrdienstleiter Weichen und Signale über einen Gleisbildstelltisch. Auf diesem Stelltisch befindet sich eine schematische Nachbildung der Außenanlage. Über Tasten im Stelltisch empfängt die Relaislogik des Stellwerks die Befehle des Fahrdienstleiters und zeigt über Lampen Informationen auf dem Stelltisch an. So werden vom Fahrdienstleiter zum Beispiel Weichen und Signale gesteuert. Das Stellpult arbeitet nach dem Zwei-Tasten-Prinzip, d. h. es müssen zwei Tasten gleichzeitig zum Ausführen einer Bedienhandlung gedrückt werden. So können versehentliche Bedienungen des Stellwerks vermieden werden.

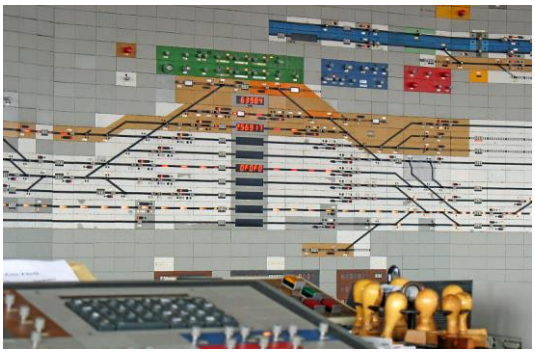


Abbildung 1: SpDrS60 Gleisbildstellwerk in Fürth (Quelle: fuerthwiki.de, Ralph Stenzel, CC BY-SA 3.0 DE)

Spurplanstellwerke gehören zu den elektrischen Gleisbildstellwerken. Im Gegensatz zur festen Verkabelung von Relais, wie es in vorherigen Generationen von Relaisstellwerken üblich war, werden bei diesem Stellwerkstyp einzelne Relaisgruppen über Spurkabel miteinander verbunden. So wird die Stellwerkslogik hergestellt. Nach dem ersten Einbau dieses Stellwerktyps 1963 in Sarstedt wurde diese Bauform zum Standard-Spurplanstellwerk in der Bundesrepublik Deutschland. Trotz der Weiterentwicklung der Stellwerkstechnik und der Einführung von elektronischen Stellwerken (ESTW) und digitalen Stellwerken (DSTW) sowie

Modernisierungen des Stellwerksbestand seitens der Deutschen Bahn werden in 1200 der 3590 Stellwerke[1] der Deutschen Bahn Zugbewegungen mit Relaistechnik gesteuert.

Beim SpDrS60-Stellwerk werden Weichen und Signale nicht einzeln, sondern in Fahrstraßen geschaltet. Diese stellt der Fahrdienstleiter über das Drücken einer Start- und einer Zieltaste auf dem Stelltisch ein. Diese Fahrstraßentasten haben durch ihre Orientierung im schwarzen Gleisband auf dem Stellpult eine Richtung, in welcher sie agieren können. Durch das Drücken der Start- und der Zieltaste durch den Fahrdienstleiter wird überprüft, ob die ausgewählte Strecke von der Start- zur Zieltaste von keiner anderen Fahrstraße belegt ist. Diese Aufgabe übernimmt die Relaislogik im Stellwerk. Ist der Weg frei, werden die Weichen in die richtige Position gestellt und verriegelt. Anschließend wird das Hauptsignal am Anfang der Fahrstraße auf ‚Fahrt' gestellt und der Zug darf einfahren. Sobald der Zug am Hauptsignal vorbeigefahren ist, fällt dieses selbstständig in ‚Halt'-Stellung zurück und die Fahrstraße wird automatisch hinter dem Zug ohne Aktion des Fahrdienstleiters aufgelöst. So kann nach Durchfahren des Zuges sofort eine neue Fahrstraße eingestellt werden.

Beim Einstellen von Fahrstraßen wird zwischen Zugstraßen und Rangierstraßen unterschieden. In diesem Projekt werde ich mich auf Zugstraßen konzentrieren. Zugstraßen haben im Gegensatz zu Rangierstraßen höhere Sicherheitsstandards. So muss das Gleis bei Zugstraßen tatsächlich frei sein. Rangierstraßen dürfen zum Beispiel auch bei durch Wagen besetztem Gleis durchgeführt werden. Fahrstraßen gewähren außerdem Flankenschutz, d. h. das seitliche Hineinfahren in einen Zug durch eine falsch gestellte Weiche wird verhindert (Flankenfahrt), indem die gegenüberliegende Weiche passend gestellt wird.

2 Ergebnisse des Projekts „Simulationssoftware für ein Gleisbildstellwerk"

In meinem letzten Projekt habe ich für die aktiven Bestandteile des Stellpults wie zum Beispiel die Weichen, Signale und Zugtasten einzelnen Klassen zugeordnet. So verfügt zum Beispiel ein Objekt der Klasse ‚Weiche' über die Funktionen, die eine Weiche besitzt. Die Klasse ‚Actor' stellt als Oberklasse zum Beispiel die Ansteuerung der Schieberegister zur Verfügung. Diese Funktion wird von allen Klassen bis auf die Klasse ‚Zugtaste' benötigt, um die LEDs auf dem Stellpult anzusteuern. In der Klasse ‚Zugtaste' wird diese Funktion nicht benötigt. Die Klasse ‚Signal' bildet die allgemeinen Funktionen eines Signals nach, beispielsweise habe ich das Sperren der Signale in dieser Klasse umgesetzt. In der Klasse ‚Hauptsignal' finden sich dann zusätzlich Methoden, um die Signalbilder eines Hauptsignals umzusetzen.

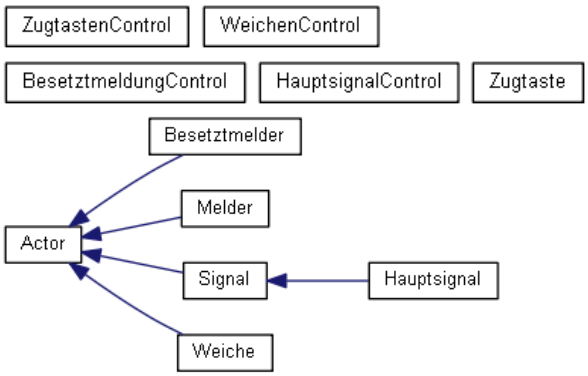


Abbildung 2: Klassenhierarchie des Projekts (Quelle: eigene Darstellung, erstellt mit Doxygen/Graphviz)

Je nach Größe des Stellpults entsteht so eine unterschiedliche große Anzahl an Objekten. Um diese Objekte zu verwalten, habe ich in meinem letzten Projekt für Klassen mit einer großen Anzahl von Objekten auf dem Stelltisch eine Klasse erstellt, welche diese Objekte verwaltet. Diese Klassen übernehmen das Initialisieren der Objekte und ermöglichten einen einheitlichen Zugriff auf die einzelnen Objekte. Der Klassenname setzt sich aus dem Namen der Klasse, die verwaltet wird und „Control" zusammen und ist so leicht zuzuordnen. Insbesondere diese „Control"-Klassen sollen in meinem Projekt einer besseren Lösung weichen.

[1]Aus dem Infrastrukturzustands- und -entwicklungsbericht 2021, abgerufen am 12.11.2022 unter *https://www.eba.bund.de/SharedDocs/Downloads/DE/Finanzierung/IZB/IZB_2021.html*

3 Vorgehensweise, Materialien und Methoden

3.1 Modellierung des Gleisplans als Graph

Im Rahmen meiner in der Oberstufe verpflichtenden Facharbeit habe ich mich mit Graphentheorie beschäftigt. Dabei entstand die Idee, die Graphentheorie auch zur Verbesserung der Fahrstraßensteuerung zu verwenden und den Gleisplan als Graph zu modellieren. Dieser kann dann vom Mikrocontroller durchsucht werden, um eine Fahrstraße zu finden und einzustellen. Dabei symbolisiert jedes Tischfeld, das ein Gleiselement enthält, einen Knoten innerhalb des Graphen. Jeder Knoten verfügt über die Attribute, die zu seinem entsprechenden Tischfeld gehören. Das können zum Beispiel Besetztmelder, Weichen oder Signale sein. Die Attribute eines Tischfeldes werden also direkt in einem Knoten gespeichert und nicht wie zuvor in einer speziellen Klasse. Über Kanten sollen diese Knoten so verknüpft werden, dass sie ein Abbild des Gleisplans ergeben. Den Graphen habe ich ungerichtet modelliert, da Zugbewegungen in beide Richtungen möglich sind und daher die Richtung der Kanten unerheblich ist.

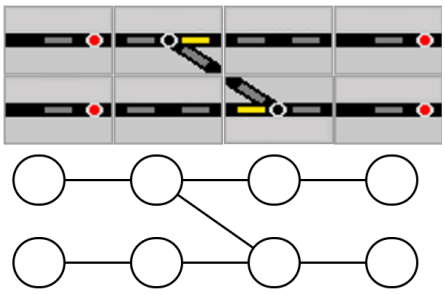


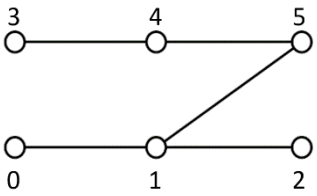
Abbildung 3: Gleisbildstellpult (oben) und der zugehörige Graph (unten) (Quelle: (v.o.) Guido Scholz, spdrs60.sourceforge.net (abgerufen am 13.01.2023) mit persönlicher Erlaubnis zur Verwendung und Bearbeitung; eigene Grafik

3.2 Speicherung des Gleisplans

Es gibt hauptsächlich zwei Möglichkeiten, einen Graphen in einem informatischen System zu speichern. Dazu kann ein Graph unter Verwendung einer Adjazenzmatrix oder einer Adjazenzliste gespeichert werden.

3.2.1 Adjazenzmatrix

Bei einer Adjazenzmatrix werden mit einem zweidimensionalen Feld der Länge n-Knoten und der Breite n-Knoten die Kanten zwischen den Knoten angegeben. Ob der Knoten i und der Knoten j benachbart sind, lässt sich mit $A(j,i)$ feststellen. Enthält das Feld $A(j,i)$ eine 1, so existiert zwischen diesen beiden Knoten eine Verbindung.



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Abbildung 4: ungerichteter Graph mit seiner Adjazenzmatrix (Quelle: eigene Darstellung, erstellt mit yEd)

Diese Art der Speicherung ermöglicht es leicht und mit hoher Geschwindigkeit, einzelne Verbindungen zwischen Knoten innerhalb des Graphen zu überprüfen. Der Speicherplatzverbrauch liegt dabei konstant bei n^2, wobei n die Anzahl der Knoten repräsentiert. Da der Gleisplangraph ungerichtet ist, führt diese Art der Speicherung allerdings zu einer Redundanz der Daten, da $A(j,i) = A(i,j)$ ist. Die Daten in der Matrix sind also gespiegelt. Sollen alle Nachbarn eines Knoten i ausgegeben werden, so muss die gesamte Zeile $A(i)$ der Länge n durchsucht werden.

3.2.2 Adjazenzliste

Eine Adjazenzliste besteht aus einer zweidimensionalen verketteten Liste mit der Länge n-Knoten. Während an erster Stelle alle Knoten des Graphen stehen, folgen in einer Liste die Nachbarn dieser Knoten. So hat der Knoten 1 aus Abbildung 5 zum Beispiel die Nachbarn null, zwei und fünf und der Knoten zwei den Knoten eins.

Die Größe der Adjazenzliste ist abhängig von der Dichte des Graphen und der Menge der Knoten, die Geschwindigkeit ist abhängig vom Grad der Knoten des Graphen. Bei dieser Art der Speicherung können zusätzlich die Nachbarn eines bestimmten Knotens im Gegensatz zur Adjazenzmatrix einfach und schnell ausgegeben werden.

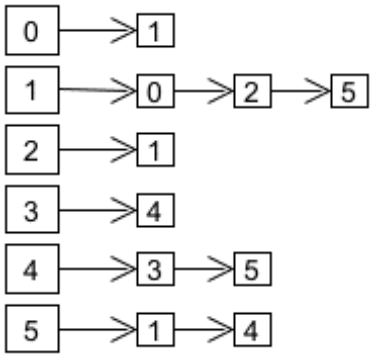


Abbildung 5: Adjazenzliste mit einer verketteten Liste zum Graphen aus Abbildung 4 (Quelle: eigene Darstellung, erstellt mit yEd)

3.2.3 Umsetzung des Graphen für das Stellpult

Bei der Implementierung des Graphen für das Stellpult habe ich eine abgeänderte Form der Adjazenzliste verwendet. Die Speicherung wurde statt mit einer verketteten Liste mit einer Matrix der Größe $n * 3$ umgesetzt (siehe Abbildung 6). Die Matrix hat also eine feste Länge abhängig von der Anzahl der Knoten n, da die Anzahl der Knoten sich nicht verändert und jeder Knoten eine maximale Anzahl von drei Nachbarn hat. Das ist zum Beispiel bei einer Weiche der Fall. Einem Knoten wird bei der Implementierung des Graphen eine feste Nummer zugeordnet. So können über $B(i,l)$, die Nachbarn des Knoten i an der Position $l = \{0,1,2\}$ ermittelt werden und es wird die Nummer des Nachbarknotens an der Position l zurückgegeben. So kann der Nachbar des Knotens i identifiziert werden. Die Positionen im Array werden dabei aufsteigend von Position null an mit Knotennummern aufgefüllt. Hat ein Knoten weniger als drei Nachbarn, so wird für die unbelegten Positionen in der Matrix -1 zurückgegeben. Die Rückgabe liegt damit außerhalb des Wertebereichs für Knotennummern $W_{KnotenNr} = \{x \in R | x \geq 0\}$. Die Gleissymbolobjekte werden in einem Array der Länge n gespeichert.

$$B = \begin{pmatrix} 1 & -1 & -1 \\ 0 & 2 & 5 \\ 1 & -1 & -1 \\ 4 & -1 & -1 \\ 3 & 5 & -1 \\ 1 & 4 & -1 \end{pmatrix}$$

Abbildung 6: Umsetzung der Matrix für das Stellpult zum Graph aus Abbildung 4 (Quelle: eigene Darstellung)

Die Verwendung einer Matrix führt zwar zu einem höheren Speicherplatzverbrauch als bei einer normalen Adjazenzliste, da nicht benötigte Werte der Matrix wie bei einer Adjazenzmatrix mit Standardwerten befüllt werden. Sie liegt allerdings deutlich unter dem Speicherplatzverbrauch einer Adjazenzmatrix und bietet außerdem den Vorteil, dass Weichen einfach mit der Abfrage $B(i,3) > -1$ erkannt werden können, ohne eine Liste zu durchlaufen und die Einträge zu zählen.

Mit dieser Art der Speicherung wird eine wesentliche Funktion des Programms, das Bestimmen des nächsten Nachbarn eines Knoten, auf eine konstante Laufzeit reduziert $O(3)$. Zusätzlich würde die reale Laufzeit nochmals geringer ausfallen, da Operationen innerhalb eines Arrays schneller ablaufen als in einer verketteten Liste, weil der interne Overhead bei Arrayabfragen geringer ausfällt als beim Durchsuchen einer Liste. Diese Art der Speicherung bietet somit eine auf die Besonderheiten des Gleisplangraphen angepasste Möglichkeit, die Vorteile der Adjazenzliste und der Adjazenzmatrix bei konstanter Geschwindigkeit und vergleichsweise geringem Speicherplatzverbrauch zu kombinieren.

3.3 Testen des Programmcodes

Während der gesamten Entwicklungsphase habe ich mit meinem selbst gebauten Teststellpult den Programmcode getestet. Dieses Stellpult besteht aus einem zweigleisigen Bahnhof mit einem Gleiswechsel und einem Abzweig. Es bildet mit Tastern und LEDs ein SpDrS60 Stellpult nach. Die Strecken auf dem Teststellpult können in beide Richtungen befahren werden. Über die selbst entwickelten Besetztmeldemodule habe ich zusätzlich die Selbstauflösung der Fahrstraßen getestet. So ließ sich die entwickelte Software in unterschiedlichen Situationen testen und ich konnte Fehler im Programmcode leichter erkennen.

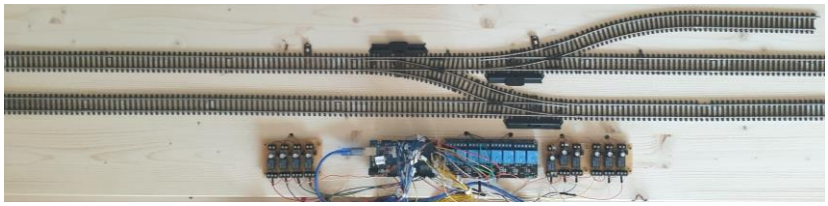


Abbildung 8: Teststrecke für das Stellpult: v. l. Besetztmelder, Arduino, Relais für die Weichen, Besetztmelder (eigenes Foto)

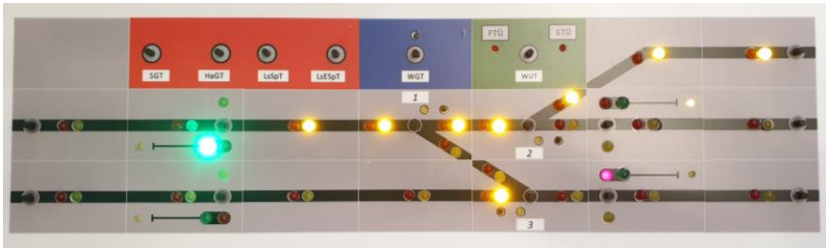


Abbildung 7: selbstgebautes Teststellpult mit eingestellter Fahrstraße (eigenes Foto)

3.4 Durchsuchen des Graphen nach einer Fahrstraße

Nachdem ich den Graphen implementiert hatte, habe ich mich damit beschäftigt, welches Verfahren ich zur Bestimmung eines Weges durch den Graphen nutzen kann. Da in meinem Fall der Graph eine Abbildung des schematischen Stellpultes ist, ist die Anzahl der Tischfelder nicht relativ zu der Länge des Gleises in der Realität. Aus diesem Grund ist den Kanten kein Gewicht zugewiesen. Bei meiner Recherche zum Thema Graphentheorie bin ich auf das Verfahren Backtracking gestoßen, das verwendet werden kann, um unterschiedliche Problemstellungen systematisch zu lösen. Ich habe mich nach einer weiteren Recherche für dieses Verfahren zum Bestimmen einer Fahrstraße entschieden, da bei einem Gleisbildstellpult in vielen Fällen nur eine Fahrstraße zwischen Start- und Zieltaste besteht, also nur die Suche nach einem Weg im Vordergrund der Suche steht. Ausgehend von der Starttaste versucht das Programm, mit diesem Verfahren einen Weg zur Zieltaste zu finden. Dieser Weg soll dann als Fahrstraße ausgeführt werden.

Beim Backtracking versucht das Programm, so lange einen Weg zu verfolgen, bis festgestellt wird, dass dieser zur Lösung geführt hat oder nicht. Im Gegensatz zur erschöpfenden Suche wird die Suche nach einem Weg dann abgebrochen, wenn dieser Weg offensichtlich zu keiner Lösung führt. Dieses Vorgehen verbessert die Laufzeiten des Programms. Knoten, die bereits durch das Programm untersucht wurden, werden markiert. Führt ein Weg nicht zur Lösung, wird der letzte Schritt rückgängig gemacht. Das Programm macht also einen track back und es wird ein anderer Nachbarknoten verwendet. Der Weg wird so lange angepasst, bis eine Lösung gefunden wurde, es keine Lösungsmöglichkeiten mehr gibt oder jeder Knoten im Graphen markiert ist. Wurde ein Weg gefunden, wird dieser Pfad als Weg markiert.

4. Ergebnisse

4.1 Besonderheiten bei der Modellierung des Schienensystems als Graph

Auf Grundlage des Backtrackingverfahren, das auf die Tiefensuche zurückgreift, habe ich begonnen, die Fahrstraßensteuerung zu implementieren. Dabei bin ich auf einige Besonderheiten gestoßen, welche ich in meiner Modellierung des Stellwerks als Graph nicht berücksichtigt hatte.

4.1.1 Alle Signale werden geschaltet

Nach dem Einstellen der ersten Fahrstraße wurden nicht nur die Signale in Fahrtrichtung der Fahrstraße gestellt, sondern auch Signale entgegen der Fahrtrichtung auf ‚Fahrt' gestellt (siehe Abbildung 9). Da Signale allerdings nur in einer Richtung gelten, müssen diese je nach Fahrstraßenrichtung auf ‚Fahrt' gestellt werden oder nicht. Um diesem Problem entgegenzuwirken, habe ich den Signalen eine Richtung in Form einer booleschen-Variable zugeordnet: True, wenn das Signal von links nach rechts auf dem Stellpult angeordnet ist, sonst false. Das Programm vergleicht die Richtung des Signals mit der Richtung der Fahrstraße und schaltet das Signal gegebenenfalls auf „Fahrt". Die Richtung der Fahrstraße wird bestimmt, indem die Nummern des aktuellen und des vorherigen Knotens verglichen werden. Da die Knoten ebenfalls von links nach rechts und von oben nach unten nummeriert sind, lässt sich so beurteilen, ob Fahrstraße und Hauptsignal die gleiche Richtung teilen.

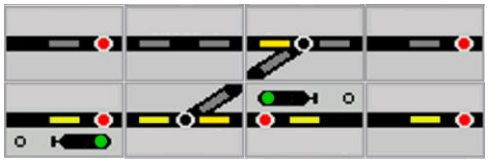


Abbildung 9: Fahrstraße von unten links nach unten rechts. Signale in beiden Fahrtrichtungen werden fälschlicherweise geschaltet (Quelle: (geändert) Guido Scholz, spdrs60.sourceforge.net (abgerufen am 13.01.2023) mit persönlicher Erlaubnis zur Verwendung und Bearbeitung)

4.1.2 Einschränken der möglichen Fahrstraßen im Graphen

Außerdem bestand durch die Modellierung als Graph das Problem, dass der Backtracking-Algorithmus mehr Wege erkannte als es mögliche Fahrstraßen gibt. So wurden auch Wege über zwei Weichenenden als Fahrstraßen erkannt (siehe Abbildung 11). Diese Wege sind in der Modellierung des Schienensystems als Graph richtig, müssen aber für die Realität eingeschränkt werden. So bestehen bei Weichen zum Beispiel nur eingeschränkte Möglichkeiten, diese aus unterschiedlichen Richtungen zu befahren. Diese Abhängigkeiten habe ich zusätzlich zu den Verbindungen der Kanten in der abgewandelten Adjazenzmatrix gespeichert. Dazu werden die Nachbarn eines Weichenknotens in einer festen Reihenfolge gespeichert (siehe Abbildung 10). Der Knoten K1 am Weichenanfang wird hierzu an erster Position gespeichert, der Nachbar in gerader Richtung danach und als letztes der Nachbar in abzweigender Richtung gespeichert.

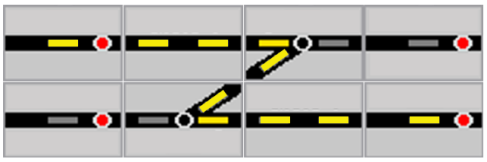


Abbildung 11: Bsp. nicht umsetzbare Fahrstraße (Quelle: (geändert) Guido Scholz, spdrs60.sourceforge.net (abgerufen am 13.01.2023) mit persönlicher Erlaubnis zur Verwendung und Bearbeitung)

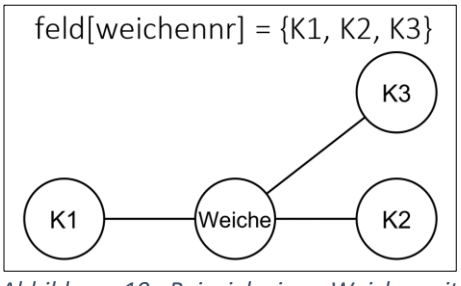


Abbildung 10: Beispiel einer Weiche mit angrenzenden Knoten (Quelle: eigene Darstellung)

Schon bei der Bestimmung des Fahrweges können so Wege ausgeschlossen und die Laufzeit verkürzt werden. Dazu wird mit dem Vorgängerknoten und dem aktuellen Weichenknoten die Richtung der Fahrstraße bestimmt. Mit dieser Methode[2] wird die nächste sinnvolle Knotennummer ermittelt.

[2] int Graph::nextWay(int knotenNr, int vorgaengerNr, boolean vorgaengerAktiv)

4.2 Einstellen und Auflösen einer Fahrstraße

4.2.1 Einstellen einer Fahrstraße

Finden und Markieren eines Weges beim Stellwerk

Das Einstellen einer Fahrstraße läuft immer nach demselben Schema ab. Werden zwei Fahrstraßentasten gedrückt, wird mit den beiden gedrückten Tasten als Parameter das Einstellen der Fahrstraße ausgelöst. Innerhalb dieser Methode[3] wird zuerst überprüft, ob beide Tasten die gleiche Richtung haben und eine Fahrstraße somit möglich ist. Anschließend beginnt die Suche nach einem Weg zwischen den Fahrstraßentasten. Dazu werden zunächst alle Markierungen der Knoten zurückgesetzt. Via Backtracking wird dann ein Weg zwischen den beiden Gleissymbolen, auf denen die Fahrstraßentasten gedrückt wurden, ermittelt.[4] Die Methode sucht mittels des Backtracking-Verfahrens rekursiv den Weg und markiert diesen beim Trackback mit einer Wegnummer, wenn das Ziel gefunden wurde. Dabei wird zusätzlich die Länge des gefundenen Weges ermittelt und zurückgegeben. Wird kein Weg gefunden, wird die Zahl -1 zurückgegeben. Die Wegnummern werden aufsteigend vergeben. Da der von mir verwendete ATmega16U2 auf dem Arduinoboard 16-bit Werte verarbeitet, liegt der Wertebereich einer Wegnummer bei $W = \{0 \leq n \leq 65.535\}$. Damit kann es 65.535 aktive Fahrstraßen geben, da die Null als Standardwert für nicht verwendete Gleissymbole reserviert ist.

Die als Wegmarkierung vergebene Nummer kann jederzeit geändert werden. Erst, wenn ein Knoten zum Fahrstraßenelement wird, kann diese Nummer nicht mehr geändert werden, bis das Tischfeld mit der Fahrstraßennummer wieder zurückgesetzt wird.

Ermitteln der Starttaste

Ist der Weg markiert, wird überprüft, welche der beiden Tasten die Start- und welche die Zieltaste ist. Das ist notwendig, um die Richtung der Fahrstraße zu bestimmen, so dass nur Signale in Fahrtrichtung geschaltet werden. Außerdem verläuft das automatische Auflösen der Fahrstraße in dieser Richtung.

Um die Starttaste zu bestimmen, habe ich die Richtung der Fahrstraßentasten und die Nummern der nächsten Knoten entlang des Weges ausgehend von einer Taste verwendet. Bei dieser Bestimmung spielen vier Situationen eine Rolle (siehe Abbildung 12): Zum einen die zwei Richtungen, mit oder gegen die Nummerierung der Knoten und zum anderen die Richtung der Fahrstraßentasten. Ist zum Beispiel die Richtung der Fahrstraßentasten ‚false' und der Knoten der Taste 1 hat eine größere Nummer als der nächste Knoten entlang des Weges, so ist die Taste eins die Starttaste (siehe Abbildung 12 oben links). Der Knoten dieser Taste wird dann zum Anfang der Fahrstraße erklärt.[5]

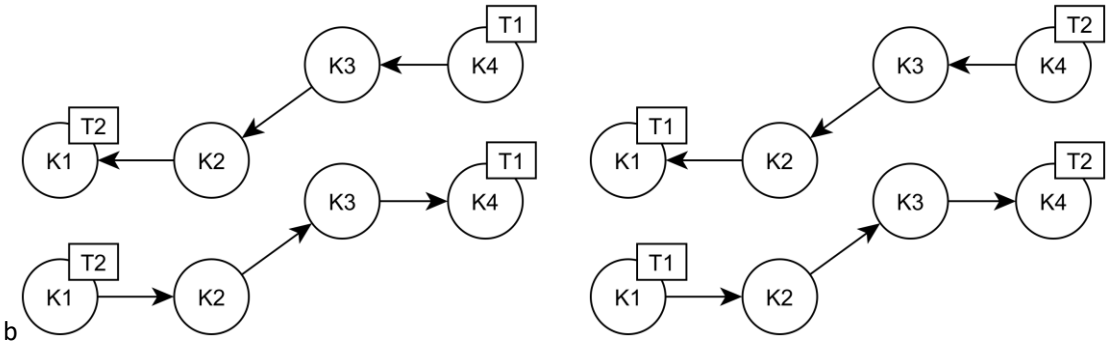


Abbildung 12: Situationen für die Ermittlung der Starttaste der Fahrstraße, die Pfeile geben dabei die Richtung true (nach rechts) oder false (nach links) der Fahrstraßentasten an
(Quelle: eigene Darstellung)

[3] boolean Graph::fahrstrasseEinstellen(Zugtaste *taste1, Zugtaste *taste2)
[4] int Graph::wegSuchen(Gleissymbol *start, Gleissymbol *ziel, Gleissymbol *vorgaenger)
[5] Gleissymbol::setAnfang(boolean status)

Ausführen einer Fahrstraße auf dem Stellpult

Wenn der Start der Fahrstraße bestimmt wurde, wird diese vom Anfangsknoten aus umgesetzt.[6] Dazu durchläuft das Programm die Knoten entlang der Fahrstraße und stellt für jeden Knoten alle Attribute ein. Bereits besuchte Knoten werden markiert.[7] So werden Besetztmelder durch das Setzen des Knotens als Fahrstraßenelement aktiviert, so dass sie nicht nur bei ‚besetzt sein' aufleuchten. Alle Signale in Fahrtrichtung werden auf ‚Fahrt' gestellt und Weichen in die richtige Position gebracht. Zur Bestimmung der richtigen Weichenposition wertet eine Methode[8] den Vorgängerknoten, den aktuellen Knoten mit der Weiche und den folgenden Knoten aus und gibt zurück, welche Weichenposition gewählt werden muss.

4.2.2 Auflösen einer Fahrstraße

Das Auflösen einer Fahrstraße erfolgt automatisch. Sobald ein Zug das auf ‚Fahrt' stehende Signal überquert hat, fällt dieses wieder in ‚Halt'-Stellung. Anschließend löst sich die Fahrstraße hinter dem Zug auf und eine neue Fahrstraße kann an derselben Position eingestellt werden. Für das Auflösen der Fahrstraßen und für weitere Funktionen, wie zum Beispiel die Besetztmeldung, wird das Stellpult fortlaufend aktualisiert.[9] Um das Auflösen der Fahrstraße zu ermöglichen, wird kontrolliert, ob die Besetztmelder eines als Anfang markierten Knotens einer Fahrstraße besetzt sind. Ist dieses der Fall, wird der Anfang auf den nächsten Knoten der Fahrstraße gesetzt und der aktuelle Knoten aus der Fahrstraße entfernt, indem die Markierung als Fahrstraßenelement deaktiviert wird. Das Verändern des Anfangs der Fahrstraße wird so lange wiederholt, bis der Zug die Fahrstraße durchquert hat und sich die Fahrstraße selbstständig aufgelöst hat.

4.3 Welche Fahrwege werden mit dem Backtracking-Verfahren bestimmt?

Beim Durchführen des Backtrackingverfahrens bestimmt die Ausgabe des nächsten Weges vom aktuellen Knoten aus, welcher Weg von diesem Verfahren bestimmt wird. Gibt es zwei mögliche Wege über das Gleisfeld, bestimmt das Programm den Weg, der möglichst lange Weichen gerade überfährt. Dies liegt darin begründet, das als nächster Knoten ein Knoten in gerader Weichenposition bevorzugt ausgegeben wird. So muss ein Zug erst möglichst spät über eine Weiche fahren. Dies erhöht den Fahrkomfort der Passagiere, da die Fahrdauer in gerader Richtung maximiert wird.

4.4 Laufzeit des Backtrackings

Neben der Speicherung des Graphen spielt auch die Laufzeit des Backtracking-Verfahrens eine Rolle, um die Effizienz des Programms zur Ermittlung von Fahrstraßen zu bestimmen. Dies gilt besonders bei großen Stellwerken mit einem Gleisplangraphen mit vielen Knoten. Um die Zeitkomplexität des Backtracking-Verfahren abzuschätzen, habe ich diese im schlechtesten Fall bestimmt. Das ist der Fall, wenn der gesamte Graph traversiert wird und erst am letzten Knoten eine Lösung gefunden wird. Das ist zum Beispiel bei der Suche einer Fahrstraße zwischen Knoten 0 und Knoten 5 der Fall, da zuerst der Weg über die Weiche in gerader Richtung überprüft wird und erst danach die Knoten in Richtung Kurve überprüft werden. Da bei dem Besuch jedes Knotens alle Kanten von jeder Seite, also zweimal, überquert werden, liegt die Laufzeit im schlechtesten Fall bei $O(|E| * 2) = O(|E|)$, wobei $|E|$ die Anzahl der Kanten in einem Graphen ist. Damit entwickelt sich die Laufzeit des Programms im schlechtestenfalls linear.

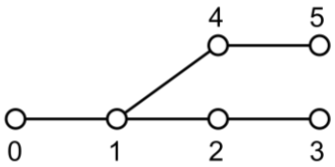


Abbildung 13:Laufzeit des Backtracking, Worst-case-Laufzeit von Knoten 1 zu Knoten 5 (erstellt mit yEd)

[6] Graph::symbolZuFahrstrasse(int knotenNr)

[7] Gleissymbol::setMarkierung(boolean status)

[8] boolean Graph::richtungGerade(int vorgaenger, int weichensymbolNr, int nachfolger)

[9] Graph::updateSymbole()

Die tatsächliche Laufzeit ist aber in vielen Fällen besser, da zum einen vor Beginn der Suche überprüft wird, ob die Fahrstraßentasten die gleiche Richtung zugewiesen haben und so eine Fahrstraße überhaupt möglich ist, zum anderen durch die beschränkten Passiermöglichkeiten der Weichen Teile des Graphen sofort durch den Algorithmus ausgeschlossen werden.

5 Anwendungsmöglichkeiten

Auf Grund der Anwendungsmöglichkeiten der Simulationssoftware im Modellbahnbereich habe ich meine Steuerungssoftware mit anderen Modellbahninteressierten in einem Modellbahnforum diskutiert. So konnte ich einige neue Ideen und Anregungen zur Verbesserung der Software aus Modellbahnperspektive gewinnen. Neben dem Anwendungsbereich der Modelleisenbahn habe ich mich mit der Deutschen Bahn AG in Verbindung gesetzt, um mein Projekt vorzustellen und über Anwendungsmöglichkeiten ins Gespräch zu kommen. Über einen Kontakt von einem MINT-Camp wurde ich an Herrn Michael Binzen, Mitarbeiter der DB Systel, weitergeleitet. Die DB-Systel ist der Digitalpartner der Deutschen Bahn AG. In einem einstündigen Gespräch habe ich ihm mein Projekt vorgestellt und wir haben uns über die Anwendungsmöglichkeiten bei der Bahn unterhalten. Er sagte mir seine Unterstützung beim Finden eines Ansprechpartners bei der DB Netz AG, dem Schieneninfrastrukturunternehmen der Deutschen Bahn, zu. Insbesondere im Bereich der Lehrstellwerke sah Herr Binzen einen Anwendungsbereich für meine Simulationssoftware, da diese auf marktgängiger und günstiger Hardware aufbaue und die Software durch ihre Struktur modular und damit anpassbar sei. Derzeit stehe ich mit Herrn Binzen in Kontakt, um diese Zusammenarbeit auszubauen.

6 Ergebnisdiskussion

In meinem Projekt „SpDrS60-Stellwerkssimulation zur Verwendung in beliebigen Schienensystemen“ habe ich mir zum Ziel gesetzt die Fahrstaßensteuerung zu verbessern, so dass ich sie einfach auf verschiedene Stellwerkssituationen anpassen kann. Die Modellierung des Schienensystems als Graph und die Verwendung des Backtracking-Verfahrens bietet eine solche flexible Lösung zum Bestimmen, Ausführen und Auflösen der Fahrstraßen auf Stellwerken mit verschiedenen Gleisanlagen. Innerhalb meiner Arbeit musste ich dazu die Modellierung immer wieder an die Bedingungen des Stellpults und der Realität anpassen. Insbesondere die objektorientierte Grundlage meines Projektes hat die Umsetzung des Graphen unterstützt und mir eine gute Basis zur Weiterentwicklung geboten.

Neben der neuen Fahrstraßensteuerung hat die Implementierung die Struktur und die Übersichtlichkeit des Programms deutlich verbessert. Alle Attribute eines Tischfeldes sind nun einem Objekt der Klasse ‚Gleissymbol‘ zugeordnet. Durch die neue Struktur werden die Abhängigkeiten zwischen den einzelnen Teilen eines Tischfeldes deutlicher. Auch die Klasse ‚Graph‘ hat zur Übersichtlichkeit innerhalb des Quellcodes beigetragen. Besonders deutlich wird dieses innerhalb des Hauptprogramms, da die Fahrstraßensteuerung durch die neue Implementierung deutlich weniger Raum einnimmt. So hat die Anpassung zu einer flexiblen und anpassbaren Lösung geführt, welche zusätzlich leichter nachzuvollziehen ist und es so auch anderen erlaubt, das Programm anzuwenden. Diese Möglichkeit möchte ich anderen Interessierten mit der Veröffentlichung auf GitHub bieten.

Erweitern ließe sich das Projekt um einen Algorithmus, welcher nicht nur einen Weg ermittelt, sondern bei mehreren möglichen Wegen einen optimalen Weg bestimmt. Zum Beispiel einen Weg mit möglichst wenig Weichenüberquerungen. Zu diesem Thema wäre auch eine weitere Hintergrundrecherche notwendig, um herauszufinden, wie eine solche Situation in der Realität entschieden wird.

Auch mein Kontakt zur Deutschen Bahn AG hat mein Projekt weiterentwickelt und mich bestärkt, dieses Thema sowohl aus Modellbahnperspektive als auch aus Ausbildungs- und Schulungsperspektive weiterzuentwickeln.

7 Literaturverzeichnis

5.3 Backtracking und Branch&Bound-Verfahren. (2009). In L. Suhl, & M. Taïeb, *Optimierungssyteme Modelle, Verfahren, Software, Anwendungen 2. Auflage* (S. 144-160). Berlin Heidelberg: Springer-Verlag.

Breymann, U. (2007). *C++ Einführung und professionelle Programmierung, 9. Auflage.* München: Carl Hanser Verlag.

Deutsche Bahn AG. (April 2022). *www.eba.bund.de.* Abgerufen am 12. November 2022 von https://www.eba.bund.de/SharedDocs/Downloads/DE/Finanzierung/IZB/IZB_2021.html

Deutsche Bundesbahn. (1983). *DS 482/9 Vorschrift für die Bedienung von Signalanlagen-Spurplanstellwerk Sp Dr 60 mit Informationen zu den einzelnen Tischfeldern.* Abgerufen am 03. Oktober 2020 von joguts-eisenbahnwelt.de: https://joguts-eisenbahnwelt.de/wp-content/uploads/2020/07/DB_DS-482-9-Vorschrift-f%C3%BCr-die-Bedienung-von-Signalanlagen-Spurplanstellwerk-Sp-Dr-60_19831101_B0.pdf

Helder, R. (kein Datum). *Bedienungsanleitung für Spurplandrucktastenstellwerk*. Abgerufen am 28. September 2022 von http://home.planet.nl/~helde862/PctWin/SpDrS60/spdrs60.html

Kötting, H. (kein Datum). *Informationen zur Stellwerkstechnik von Gleisbildstellwerken*. Abgerufen am 30. Dezember 2022 von stellwerk.de: http://stellwerke.de/formen/seite2_4.html

Letschert, T. (08. Juni 2016). Entwurf von Algorithmen II. *Algorithmen und Datenstrukturen*. Abgerufen am 13. Januar 2023 von https://homepages.thm.de/~hg51/Veranstaltungen/A_D/Folien/ad-09.pdf

Turau, V. (1996). *Algorithmische Graphentheorie.* Addison-Wesley.

vers. Autoren. (kein Datum). *C++-Programmierung: Ausdrücke und Operatoren.* Abgerufen am 03. November 2022 von Wikibooks (Wikimedia Foundation): https://de.wikibooks.org/wiki/C%2B%2B-Programmierung:_Ausdr%C3%BCcke_und_Operatoren

Willemer, A. (2020). *Dynamische Strukturen.* Abgerufen am 21. 10 2020 von Willemers Informatik-Ecke: http://willemer.de/informatik/cpp/dynamic.htm