



TECHNISCHE UNIVERSITÄT BERLIN

FACHGEBIET BAHNBETRIEB UND INFRASTRUKTUR

BACHELORARBEIT

**Realitätsnahe Fahrzeugsteuerung  
für die Eisenbahnbetriebssimulation  
im Eisenbahn-Betriebs- und  
Experimentierfeld**

*Friedrich Kasper Völkers*

391529

betreut von

Dr.-Ing. Christian BLOME

Berlin, 23. September 2021



## Aufgabenstellung

Im Eisenbahn-Betriebs- und Experimentierfeld (EBuEf) des Fachgebietes Bahnbetrieb und Infrastruktur der Technischen Universität Berlin können Prozesse des Bahnbetriebs unter realitätsnahen Bedingungen simuliert werden. Den Mittelpunkt der Anlagen bilden originale Stellwerke unterschiedlicher Entwicklungsstufen der Eisenbahnsicherungstechnik vom mechanischen Stellwerk bis zu aus einer Betriebszentrale gesteuerten Elektronischen Stellwerken.

Das „Ausgabemedium“ ist eine Modellbahnanlage, die in verkleinertem Maßstab die Abläufe darstellt. Das Betriebsfeld wird in der Lehre im Rahmen der Bachelor- und Masterstudiengänge am Fachgebiet sowie darüber hinaus zur Ausbildung von Fahrdienstleitern, für Schulungen und Weiterbildungen Externer sowie bei öffentlichen Veranstaltungen wie beispielsweise der Langen Nacht der Wissenschaften eingesetzt.

Neben den Stellwerken ist auch bei den Fahrzeugen ein möglichst realitätsnaher Betrieb Teil der umfassenden Eisenbahnbetriebssimulation.

Ziel dieser Arbeit ist die Entwicklung einer Steuerungssoftware, die auf dem (modellseitig nur) punktförmig überwachten Netz die Fahrzeuge kontinuierlich überwacht, um die Fahrzeuge realitätsnäher zu steuern (beispielsweise durch maßstäbliche Beschleunigung oder punktgenaues Anhalten an Bahnsteigen gemäß der aktuellen Zuglänge) und zukünftig auch andere und neue Betriebsverfahren wie Moving Block im EBuEf simulieren zu können.

Teil der kontinuierlichen Überwachung ist die exakte Positionsbestimmung der Fahrzeuge im Netz sowie die Übermittlung der aktuellen Geschwindigkeit.

Beschleunigungs- und Bremsvorgänge sowie Ausrollphasen für optional energieoptimales Fahren sind ebenso zu berücksichtigen. Zur Kalibrierung sind die schon vorhandenen Ortungsmöglichkeiten (Belegung von Gleisabschnitten) zu verwenden.

Weitere zu berücksichtigende Eingangsgrößen aus der vorhandenen Softwarelandschaft im EBuEf sind die Netztopologie (z.B. Streckenlängen, Signalstandorte), die Fahrzeugdaten, die aktuelle Zugbildung sowie die Prüfung (vorhandene API), ob ein Zug an einer Station anhalten muss und ob er abfahren darf. Damit sind in der Simulation Fahrplanteue, Verspätungen sowie Personalausfälle darstellbar.

Die Erkenntnisse sind in einem umfassenden Bericht und einer zusammenfassenden Textdatei darzustellen. Darüber hinaus sind die Ergebnisse der Arbeit ggf. im Rahmen einer Vortragsveranstaltung des Fachgebiets zu präsentieren.

Der Bericht soll in gedruckter Form als gebundenes Dokument sowie in elektronischer Form als ungeschütztes PDF-Dokument eingereicht werden. Methodik und Vorge-

hen bei der Arbeit sind explizit zu beschreiben und auf eine entsprechende Zitierweise ist zu achten. Alle genutzten bzw. verarbeiteten zugrundeliegenden Rohdaten sowie nicht-veröffentlichte Quellen müssen der Arbeit (ggf. in elektronischer Form) beiliegen.

In dem Bericht ist hinter dem Deckblatt der originale Wortlaut der Aufgabenstellung der Arbeit einzuordnen. Weiterhin muss der Bericht eine einseitige Zusammenfassung der Arbeit enthalten. Diese Zusammenfassung der Arbeit ist zusätzlich noch einmal als eigene, unformatierte Textdatei einzureichen.

Für die Bearbeitung der Aufgabenstellung sind die Hinweise zu beachten, die auf der Webseite mit der Adresse [www.railways.tu-berlin.de/?id=66923](http://www.railways.tu-berlin.de/?id=66923) gegeben werden.

Der Fortgang der Abarbeitung ist in engem Kontakt mit dem Betreuer regelmäßig abzustimmen. Hierzu zählen insbesondere mindestens alle vier Wochen kurze Statusberichte in mündlicher oder schriftlicher Form.

## Zusammenfassung

Im Rahmen dieser Arbeit wurde eine Fahrzeugsteuerung entwickelt, welche die Fahrzeuge im eingleisigen Netz des Eisenbahn-Betriebs- und Experimentierfelds (EBuEfs) ansteuert. Dazu wurde ein allgemeingültiger Algorithmus entwickelt, der einen möglichst optimalen Fahrtverlauf ermittelt. Die Berechnung des Fahrtverlaufs basiert auf den gegebenen Infrastrukturabschnitten inklusive deren Länge und zulässiger Höchstgeschwindigkeit, der aktuellen Position und Geschwindigkeit, der Zielposition und der Ankunftszeit. Durch den ermittelten Fahrtverlauf ist eine kontinuierliche Fahrzeugüberwachung möglich und die aktuelle Position der Fahrzeuge ist zu jedem Zeitpunkt bekannt.

Todo-Liste (Beim Korrekturlesen bitte nicht beachten...)

- Was funktioniert nicht
- Formeln?!
- linebreak
- glossaries
- acronyms
- leerzeichen zwischen zahl und einheit
- doppelte leerzeichen
- *\$allTrains* beschreiben

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
<b>2</b>	<b>Grundlagen</b>	<b>2</b>
2.1	Aufbau des Eisenbahn-Betriebs- und Experimentierfelds . . . . .	2
2.2	Aufbau der <i>MySQL</i> -Datenbank . . . . .	3
2.3	Ziele und Prioritätssetzung der Fahrzeugsteuerung . . . . .	3
2.4	Fahrdynamik . . . . .	4
2.5	Aufbau des Projekts . . . . .	4
<b>3</b>	<b>Ablauf der Fahrzeugsteuerung</b>	<b>6</b>
3.1	Einlesen von statischen und mehrfach verwendeten Daten aus der <i>MySQL</i> -Datenbank in den Cache . . . . .	6
3.2	Ermittlung der Session-Daten . . . . .	7
3.3	Ermittlung aller Fahrzeuge im eingleisigen Netz und den zugehörigen Daten . . . . .	8
3.4	Berechnung der Fahrtverläufe aller Fahrzeuge . . . . .	10
3.5	Übermittlung der Echtzeitdaten an die Fahrzeuge . . . . .	14
3.6	Überprüfung nach einer Änderung der Fahrstraße . . . . .	18
3.7	Neukalibrierung der Fahrzeugposition . . . . .	18
3.8	Ermittlung von neuen Fahrzeugen im eingleisigen Netz . . . . .	20
3.9	Fehlerbehebung von Fahrzeugen . . . . .	20
<b>4</b>	<b>Berechnung des Fahrtverlaufs</b>	<b>22</b>
4.1	Ermittlung der Start- und Endposition der einzelnen Infra-Abschnitte unter Berücksichtigung der Zuglänge . . . . .	24
4.2	Berechnung bei einer Beschleunigung auf die maximal mögliche Ge- schwindigkeit . . . . .	27
4.3	Konvertierung der <i>\$keyPoints</i> in Echtzeitdaten . . . . .	29
4.4	Überprüfung des Fahrtverlaufs nach Geschwindigkeitsüberschreitungen	34
4.5	Neuberechnung unter Berücksichtigung der Geschwindigkeitsüber- schreitung . . . . .	34

4.6	Einhaltung der Mindestzeit auf einer Beharrungsfahrt . . . . .	36
4.7	Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs	40
4.8	Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs . . . . .	41
4.9	Einleitung einer Gefahrenbremsung . . . . .	44
<b>5</b>	<b>Beispielrechnung eines Fahrtverlaufs im EBUf</b>	<b>47</b>
<b>6</b>	<b>Visualisierung der Fahrtverläufe</b>	<b>51</b>
<b>7</b>	<b>Formeln</b>	<b>53</b>
7.1	Formeln für gleichmäßig beschleunigte Bewegungen . . . . .	53
7.2	Formeln für gleichförmige Bewegungen . . . . .	55
<b>8</b>	<b>Fazit</b>	<b>57</b>
8.1	Zusammenfassung der Ergebnisse . . . . .	57
8.2	Komplikationen bei dem Betrieb der Fahrzeugsteuerung im EBUf . . .	57
8.2.1	Einhaltung der Zielposition . . . . .	57
8.2.2	Ermittlung der Fahrstraßen . . . . .	58
8.2.3	Kalibrierung der Position . . . . .	58
8.3	Möglichkeiten für eine Weiterentwicklung der Fahrzeugsteuerung . . . .	58
<b>A</b>	<b>Anhang</b>	<b>60</b>
A.1	fahrzeugsteuerung.php . . . . .	60
A.2	functions.php . . . . .	69
A.3	functions_fahrtverlauf.php . . . . .	97
A.4	functions_math.php . . . . .	145
A.5	functions_cache.php . . . . .	146
A.6	functions_db.php . . . . .	152
A.7	global_variables.php . . . . .	158
A.8	speed_over_position.m . . . . .	159

## Abbildungsverzeichnis

1	Schienennetz des EBUfs . . . . .	2
2	Aufbau der Dateistrukturen . . . . .	5
3	Ablauf der Fahrzeugsteuerung . . . . .	6
4	Eigene Darstellung der Positionsbestimmung bei einem Richtungswechsel	16
5	Ablaufplan der Fahrtverlaufsberechnung . . . . .	23
6	Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit . . . . .	26
7	Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit unter Berücksichtigung der Fahrzeuglänge . . . . .	27
8	Fahrtverlaufsberechnung (1. Iterationsschritt) . . . . .	29
9	Fahrtverlaufsberechnung (2. Iterationsschritt) . . . . .	36
10	Fahrtverlaufsberechnung (3. Iterationsschritt) . . . . .	37
11	Fahrtverlaufsberechnung (4. Iterationsschritt) . . . . .	37
12	Einteilung des Fahrtverlaufs in <i>subsections</i> . . . . .	38
13	Fahrtverlauf unter Einhaltung der Mindestzeit . . . . .	40
14	Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit . . . . .	42
15	Fahrtverlauf vor der Anpassung der exakten Ankunftszeit . . . . .	43
16	Fahrtverlauf nach der Anpassung der exakten Ankunftszeit . . . . .	44
17	Ergebnis der Fahrtverlaufsermittlung . . . . .	45
18	Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO . . . . .	48



## Tabellenverzeichnis

1	Beschreibung der wichtigsten Tabellen der <i>MySQL</i> -Datenbank . . . . .	3
2	Aufbau eines Arrays in <i>next_betriebsstellen_data</i> . . . . .	10
3	Aufbau des <i>zeiten</i> -Arrays in <i>next_betriebsstellen_data</i> . . . . .	11
4	Aufbau eines Eintrags aus dem <i>\$allTimes</i> -Array . . . . .	15
5	Verhalten eines Fahrzeugs nach dem Erreichen des Ziels . . . . .	17
6	Übersicht der Fehlermeldungen . . . . .	22
7	Beschreibung der verwendeten Variablen für die Fahrtverlaufsberechnung	24
8	Exemplarische Infrastrukturabschnitt (Infra-Abschnitt)e . . . . .	25
9	Exemplarische Zugdaten . . . . .	25
10	Aufbau des <i>\$subsection</i> -Arrays . . . . .	39
11	Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung vor der Anpassung . . . . .	43
12	Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung nach der Anpassung . . . . .	44
13	<i>\$keyPoints</i> am Beispiel von der Fahrt von XAB nach XZO . . . . .	47
14	Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO . . . . .	49

## Code-Beispiele

1	Initialisierung der Cache Variablen ( <i>fahrzeugsteuerung.php</i> ) . . . . .	7
2	Ermittlung der Real- und Simulationszeit ( <i>fahrzeugsteuerung.php</i> ) . . .	8
3	<i>getCalibratedPosition()</i> ( <i>functions_db.php</i> ) . . . . .	19
4	<i>showErrors()</i> ( <i>functions.php</i> ) . . . . .	21
5	<i>getVMaxBetweenTwoPoints()</i> ( <i>functions_fahrtverlauf.php</i> ) . . . . .	28
6	<i>createTrainChanges()</i> ( <i>functions_fahrtverlauf.php</i> ) . . . . .	30
7	<i>checkIfTrainIsToFastInCertainSections()</i> ( <i>functions_fahrtverlauf.php</i> ) .	35
8	<i>safeTrainChangeToJSONFile()</i> ( <i>functions_fahrtverlauf.php</i> ) . . . . .	51
9	<i>getBrakeDistance()</i> ( <i>functions_math.php</i> ) . . . . .	54
10	<i>getBrakeTime()</i> ( <i>functions_math.php</i> ) . . . . .	55
11	<i>getTargetBrakeSpeedWithDistanceAndStartSpeed()</i> ( <i>functions_math.php</i> )	55
12	<i>distanceWithSpeedToTime()</i> ( <i>functions_math.php</i> ) . . . . .	56
13	<i>calculateDistanceforSpeedFineTuning()</i> ( <i>functions_math.php</i> ) . . . . .	56

## **Abkürzungsverzeichnis**

**EBuEf** Eisenbahn-Betriebs- und Experimentierfeld

**Infra-Abschnitt** Infrastrukturabschnitt

## Glossar

**Beharrungsfahrt** Beschreibt den Teil des Fahrtverlaufs, bei dem die Geschwindigkeit des Fahrzeugs konstant ist.

**Echtzeitdaten** Die Echtzeitdaten beschreiben für jedes Fahrzeug die Position und Geschwindigkeit bei Beschleunigungen und Verzögerungen in  $2km/h$ -Schritten und bei konstanter Geschwindigkeit in in regelmäßigen Distanz-Intervallen in Abhängigkeit von der Simulationszeit.

**Fahrstraße** Beschreibt den Weg, der für das Fahrzeug durch die Stellung der Weichen vorgegeben ist.<sup>1</sup>

**Fahrtverlauf** Der Fahrtverlauf beschreibt die Positionen, Zeiten und Geschwindigkeiten für alle Beschleunigungs- und Bremsvorgänge und den Fahrten auf einer konstanten Geschwindigkeit eines Fahrzeugs von der aktuellen Position bis zum nächsten Halt.

**Realzeit** Die Realzeit beschreibt die Zeit des Rechners/Servers, auf dem die Fahrzeugsteuerung ausgeführt wird.

**Simulationszeit** Die Simulationszeit beschreibt die aktuelle Zeit, an der sich die Fahrzeuge orientieren (Startzeit, Ankunfts- und Abfahrtszeiten etc.). Die Startzeit der Simulation kann in der Session festgelegt werden und beginnt, sobald die Session gestartet wird.

**Unix-Timestamp** Die Unixzeit zählt die Sekunden, die seit dem 1. Januar 1970 00:00 (UTC) vergangen sind und wird im Unix-Timestamp-Format angegeben.<sup>2</sup>

**Zug-ID** Die Zug-ID ordnet den Fahrzeugen die Fahrpläne zu und ist nicht mit der ID des Fahrzeugs, welche dem Eintrag der *id*-Spalte aus der *MySQL*-Tabelle *fahrzeuge* entspricht, zu verwechseln

---

<sup>1</sup> Maschek (2018, S. 114)

<sup>2</sup> The IEEE and The Open Group (2018)

# 1 Einleitung

In dieser Arbeit wird eine Fahrzeugsteuerung für das Eisenbahn-Betriebs- und Experimentierfeld (EBuEf) entwickelt und dokumentiert. Das EBuEf ist eine Einrichtung des Fachgebiets *Bahnbetrieb und Infrastruktur* der Technischen Universität Berlin und bietet die Möglichkeit theoretisch erlerntes Wissen realitätsnah zu vertiefen.<sup>3</sup>

Für die Dokumentierung werden in Kapitel 2 die Grundlagen, die Ausgangssituation, die Herangehensweise und die Ziele beschrieben. Die Funktionsweise der Fahrzeugsteuerung wird in Kapitel 3 in chronologischer Form beschrieben, wobei im Kapitel 4 die Ermittlung des Fahrtverlaufs im Detail beschrieben wird. Damit die Allgemeingültigkeit der Fahrtverlaufsrechnung in Kapitel 4 gezeigt werden kann, wurden Infrastrukturdaten verwendet, die in dieser Form im EBuEf nicht vorkommen. Aus diesem Grund wird in Kapitel 5 die Funktionsweise anhand eines Beispiels im EBuEf gezeigt und mit Hilfe der in Kapitel 7 hergeleiteten Formeln auf die Richtigkeit überprüft.

Der Quellcode der Fahrzeugsteuerung befindet sich im Anhang der Arbeit und wird nur in Ausschnitten innerhalb der Arbeit abgebildet, wenn das der Erläuterung der Funktionsweise dient. Im Quellcode der Fahrzeugsteuerung wird auf Funktionen zugegriffen, welche bereits vorhanden waren und als Grundlage gedient haben. Diese Funktionen werden bei der Erwähnung mit einem Sternchen (\*) gekennzeichnet und nicht näher erläutert.

---

<sup>3</sup> *EBuEf: Eisenbahn-Betriebs- und Experimentierfeld Berlin* (2021)

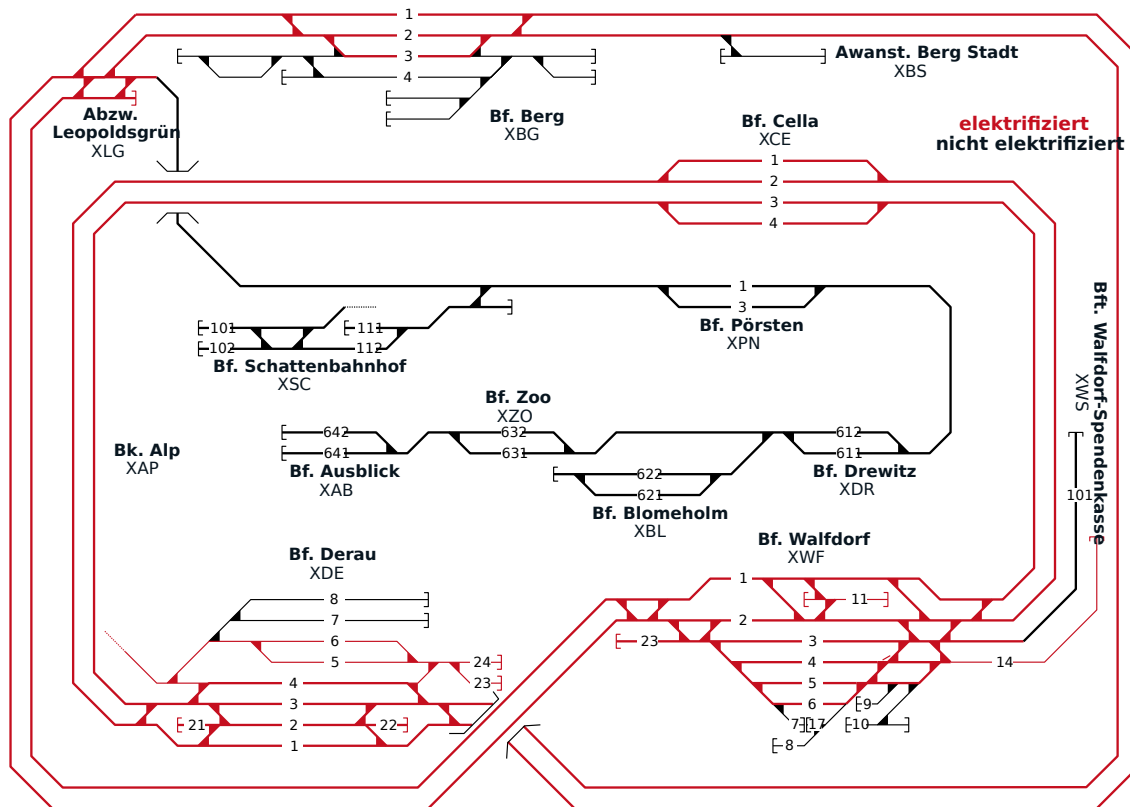


Abbildung 1: Schienennetz des EBUefs (Quelle: [www.ebuef.de/das-betriebsfeld/stellwerke](http://www.ebuef.de/das-betriebsfeld/stellwerke); Letzter Zugriff am: 4. September 2021)

## 2 Grundlagen

### 2.1 Aufbau des Eisenbahn-Betriebs- und Experimentierfelds

Das EBUef ist in ein eingleisiges nicht-elektrifiziertes und ein zweigleisiges elektrifiziertes Streckennetz unterteilt, welche über die Betriebsstelle Leopoldsgrün (XLG) miteinander verbunden sind. In der Abbildung 1 ist das eingleisige Netz in schwarz dargestellt und die zweigleisige Hauptstrecke in rot. Das eingleisige Netz ist in Infrastrukturabschnitte (Infra-Abschnitte) – welche mit Blockstrecken vergleichbar sind und eine Zugfolge im festen Raumabstand ermöglichen – eingeteilt.<sup>4</sup> Die Infra-Abschnitte sind mit der RailCom-Technik ausgestattet, welche über Decoder in den Fahrzeugen den aktuellen Infra-Abschnitt ermittelt und diesen in der *fma*-Tabelle der *MySQL*-Datenbank speichert.<sup>5</sup> Zudem sind in der Datenbank alle Informationen über die Infrastruktur gespeichert. Für die Fahrzeugsteuerung essentiell sind dabei die aktuellen Signalbegriffe

<sup>4</sup> Pachl (2021, S. 7, 42)

<sup>5</sup> RailCom - DCC-Rückmeldeprotokoll (2019)

Name	Beschreibung
<i>fahrplan_sessionfahrplan</i>	Fahrpläne der Session für alle Fahrzeuge
<i>fahrzeuge</i>	Fahrzeuge
<i>fahrzeuge_baureihen</i>	Baureiheninformationen
<i>fahrzeuge_daten</i>	Statische Daten der Fahrzeuge
<i>fma</i>	Freimeldeabschnitte
<i>gbt_fma</i>	Zuordnung der GBT-Abschnitte, FMA-Abschnitte und Infra-Abschnitte
<i>infra_daten</i>	Statische Daten der Infrastruktur
<i>infra_zustand</i>	Zustand der Infrastruktur
<i>signale</i>	Standorte der Signale

Tabelle 1: Beschreibung der wichtigsten Tabellen der *MySQL*-Datenbank

aller Signale und die Längen der Infra-Abschnitte.<sup>6</sup>

Der Betrieb des EBUefs erfordert eine angelegte Session, welche vor dem Start der Fahrzeugsteuerung gestartet werden muss, da die Fahrzeugsteuerung beim Start alle benötigten Informationen der Session einliest.

## 2.2 Aufbau der *MySQL*-Datenbank

Alle Informationen und Daten, die für den Betrieb der Fahrzeugsteuerung benötigt werden, sind in einer *MySQL*-Datenbank gespeichert. In der Tabelle 1 werden die wichtigsten Tabellen der Datenbank aufgelistet und kurz beschrieben.

## 2.3 Ziele und Prioritätssetzung der Fahrzeugsteuerung

Oberste Priorität der Fahrzeugsteuerung hat eine möglichst effiziente Umsetzung und das Einhalten der vorgegebenen Fahrpläne. Für eine effiziente Umsetzung wurden die Zugriffe auf die *MySQL*-Datenbank während des laufenden Betriebs der Fahrzeugsteuerung möglichst gering gehalten und Teile des Quellcodes, welche häufiger verwendet werden, wurden in Funktionen ausgelagert. Die Ermittlung der Fahrtverläufe berücksichtigt für die Einhaltung der Fahrplanzeiten neben den Ankunfts- und Abfahrtszeiten auch die aktuelle Verspätung und versucht diese auszugleichen.

An zweiter Stelle der Prioritätssetzung steht das energieeffiziente Fahren. Damit die Fahrten möglichst energieeffizient sind, fahren die Züge die kleinstmögliche Ge-

<sup>6</sup> *EBUef: Eisenbahn-Betriebs- und Experimentierfeld Berlin* (2021)

geschwindigkeit, bei der das Ziel ohne eine Verspätung erreicht wird. Sollte auch bei der größtmöglichen Geschwindigkeit das Ziel mit einer Verspätung erreicht werden, wird diese Geschwindigkeit gewählt. In dem Fall, dass es für ein Fahrzeug möglich ist mit einer geringeren Geschwindigkeit zu fahren als die maximal zulässige Geschwindigkeit, wird die Geschwindigkeit möglichst am Ende des Fahrtverlaufs reduziert. Dadurch hat das Fahrzeug für den Fall einer Fahrstraßenänderung oder Reduzierung der zulässigen Höchstgeschwindigkeit einen größtmöglichen Zeitpuffer.

## 2.4 Fahrdynamik

In der Realität gibt es vier Bewegungsphasen, in denen sich ein Fahrzeug befinden kann:

- Anfahren
- Beharrungsfahrt
- Auslauf
- Bremsen

Beim Anfahren ist die Antriebskraft größer als die Summe der Widerstandskräfte, wodurch das Fahrzeug beschleunigt und in der Beharrungsfahrt entspricht die Antriebskraft der Summe der Widerstandskräfte, wodurch die Geschwindigkeit des Fahrzeugs konstant bleibt. Für die Reduzierung der Geschwindigkeit kann entweder die Antriebskraft gleich null sein oder eine Bremskraft aufgewendet werden.<sup>7</sup>

Die Widerstandskräfte setzen sich aus dem Streckenwiderstand, dem Fahrzeugwiderstand und dem Anfahrwiderstand zusammen und lassen sich mit den gegebenen Daten des EBUfs nicht vollständig berechnen.<sup>8</sup> Aus diesem Grund werden die Widerstandskräfte bei der Fahrzeugsteuerung nicht berücksichtigt und die Auslaufphase, welche nur von der Widerstandskräften abhängig ist, wird ebenfalls nicht berücksichtigt.

## 2.5 Aufbau des Projekts

In der Darstellung 2 ist der Aufbau des Projekts und die für die Arbeit relevanten Dateien/Ordner dargestellt. Dateien, welche bereits vorhanden waren, sind wie Funktionen

---

<sup>7</sup> Pachl (2021, S. 23 ff.)

<sup>8</sup> Pachl (2021, S. 25 ff.)



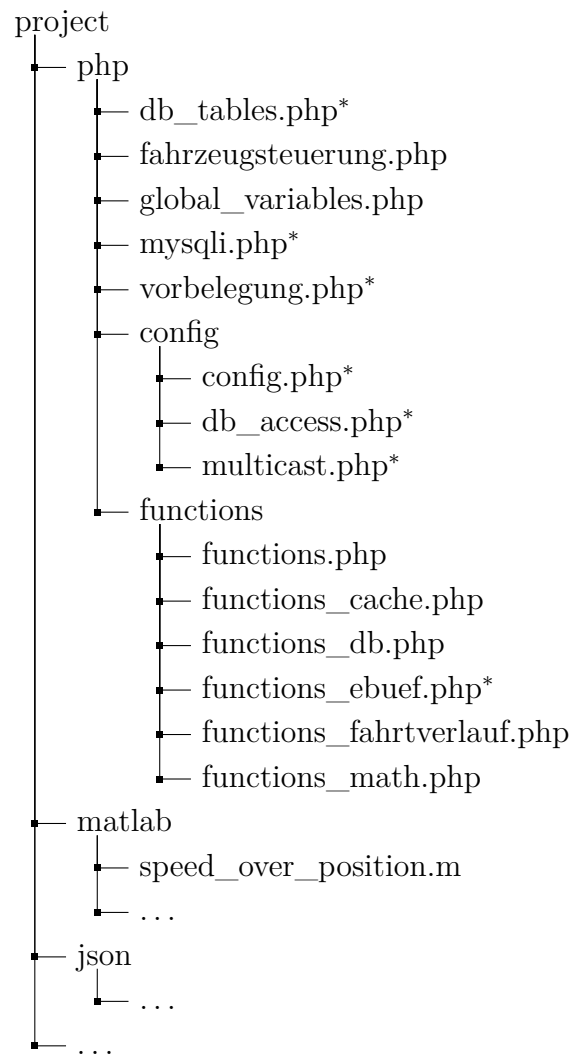


Abbildung 2: Aufbau der Dateistrukturen

mit einem Sternchen (\*) markiert. Die für die Fahrzeugsteuerung essentiellen Dateien befinden sich innerhalb des *php*-Ordners, wobei die Datei *fahrzeugsteuerung.php* die Fahrzeugsteuerung startet und für die Berechnung der Fahrtverläufe auf die Dateien in dem *functions*-Unterordner zugreift. Die benötigten Dateien für den Zugriff auf die *MySQL*-Datenbank befinden sich in dem *config*-Unterordner und global festgelegte Parameter sind in der Datei *global\_variables.php* abgespeichert. Für die Visualisierung (siehe Kapitel 6) der Fahrtverläufe werden die Dateien aus dem *matlab*- und *json*-Ordner benötigt.

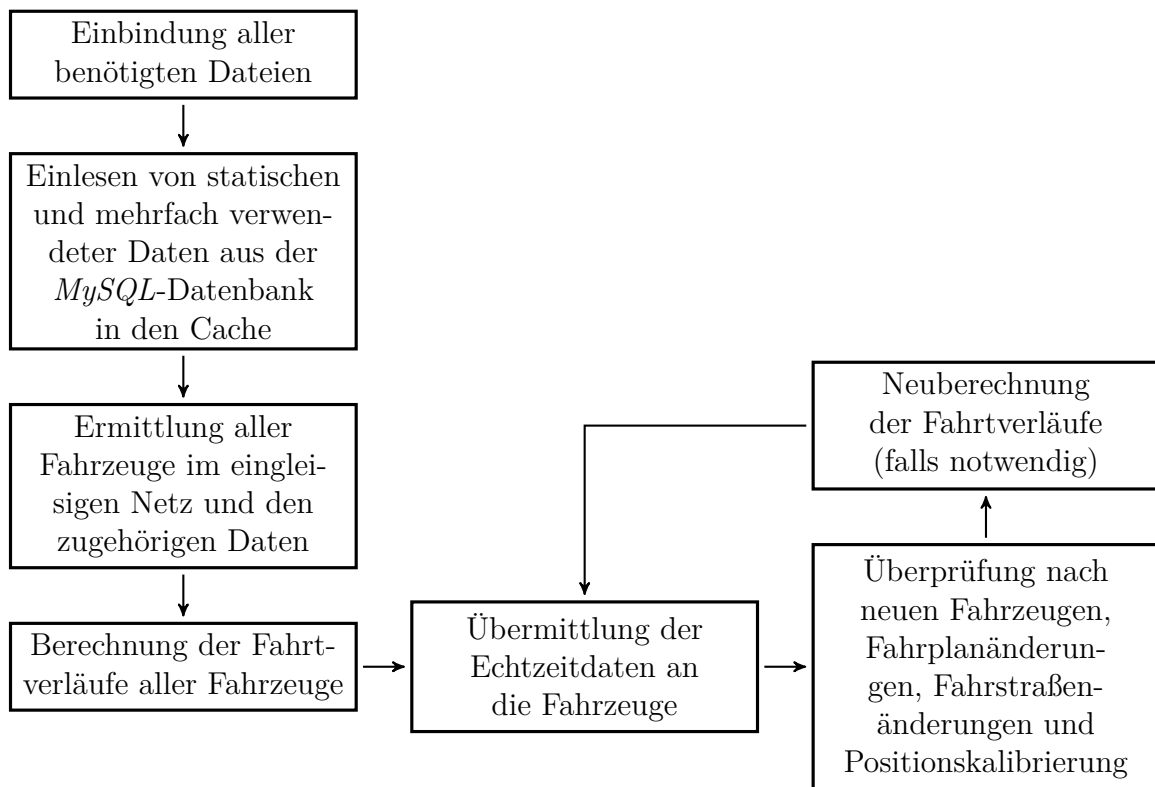


Abbildung 3: Ablauf der Fahrzeugsteuerung

### 3 Ablauf der Fahrzeugsteuerung

Damit die Fahrzeugsteuerung gestartet werden kann, muss die Datei *fahrzeugsteuerung.php* ausgeführt werden. Obligatorisch für die Fahrzeugsteuerung ist die Abschnittsüberwachung (*abschnittueberwachung.php*), welche vor dem Start der Fahrzeugsteuerung ausgeführt werden muss und auf deren Verwendung und Funktionsweise in Kapitel 3.7 eingegangen wird. Der Aufbau dieses Kapitels orientiert sich an dem Ablauf der Fahrzeugsteuerung, welcher in Abbildung 3 schematisch dargestellt wird.

#### 3.1 Einlesen von statischen und mehrfach verwendeten Daten aus der *MySQL*-Datenbank in den Cache

Die Fahrzeugsteuerung benötigt als Grundlage für viele Berechnungen Daten aus der *MySQL*-Datenbank. Damit diese Daten nicht bei jeder Verwendung erneut aus der Datenbank geladen werden müssen und somit die Anzahl an Datenbank-Abfragen möglichst gering gehalten werden kann, werden die wichtigsten Daten beim Programm-

start bzw. bei der ersten Verwendung in den Cache geladen (Code-Beispiel 1). Beispielfähig zu nennen sind hierbei *\$cacheInfraLaenge* (Länge aller Infra-Abschnitte in Metern), *\$cacheHaltepunkte* (zugehörige Infra-Abschnitte für alle Betriebsstellen und Richtung), *\$cacheZwischenhaltepunkte* (zugehörige Infra-Abschnitte für alle Zwischen-Betriebsstellen, die nur einem Infra-Abschnitt zugeordnet sind), *\$cacheGbtToInfra* (Zuordnung der Infra-Abschnitte zu den GBT-Abschnitten) und *\$cacheInfraToGbt* (Zuordnung der GBT-Abschnitte zu den Infra-Abschnitten).

```

1 // Statische Daten einlesen
2 $cacheInfranachbarn = createCacheInfranachbarn();
3 $cacheInfradaten = createCacheInfradaten();
4 $cacheSignaldaten = createCacheSignaldaten();
5 $cacheInfraLaenge = createcacheInfraLaenge();
6 $cacheHaltepunkte = createCacheHaltepunkte();
7 $cacheZwischenhaltepunkte = createChacheZwischenhaltepunkte();
8 $cacheInfraToGbt = createCacheInfraToGbt();
9 $cacheGbtToInfra = createCacheGbtToInfra();
10 $cacheFmaToInfra = createCacheFmaToInfra();
11 $cacheInfraToFma = array_flip($cacheFmaToInfra);
12 $cacheFahrplanSession = createCacheFahrplanSession();
13 $cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
14 $cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
15 $cacheIDTDecoder = createCacheDecoderToAdresse();
16 $cacheDecoderToID = array_flip($cacheIDTDecoder);
17 $cacheAdresseToID = array(); // Filled with data in getAllTrains()
18 $cacheIDToAdresse = array(); // Filled with data in getAllTrains()

```

Code-Beispiel 1: Initialisierung der Cache Variablen (*fahrzeugsteuerung.php*)

### 3.2 Ermittlung der Session-Daten

In der *MySQL*-Tabelle *fahrplan\_session* sind alle Fahrplansessions aufgelistet und der aktuell gültigen wurde der Wert 1 in der *status*-Spalte zugeordnet. Die Daten der gültigen Fahrplansession wurden bei dem Start der Fahrzeugsteuerung in dem Array *\$cacheFahrplanSession* gespeichert und werden benötigt, um die Zeitdifferenz zwischen Real- und Simulationszeit zu ermitteln. Dafür wird im ersten Schritt das Datum der *sim\_startzeit* und *sim\_endzeit*, welche die Start- und Endzeit (Simulationszeit) der Simulation im Unix-Timestamp-Format angeben, auf das aktuelle Datum der Realzeit geändert und im zweiten Schritt mit der Realzeit verglichen (Code-Beispiel 2).

```

1 // Real- und Simulationszeit ermitteln
2 $simulationStartTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_startzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "
    ↳ simulationszeit", null, array("inputtyp"=>"h:i:s"));
3 $simulationEndTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_endzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "
    ↳ simulationszeit", null, array("inputtyp"=>"h:i:s"));
4 $simulationDuration = $cacheFahrplanSession->sim_endzeit -
    ↳ $cacheFahrplanSession->sim_startzeit;
5 $realStartTime = time();
6 $realEndTime = $realStartTime + $simulationDuration;
7 $timeDifference = $simulationStartTimeToday - $realStartTime;

```

Code-Beispiel 2: Ermittlung der Real- und Simulationszeit (*fahrzeugsteuerung.php*)

Das Datum der Simulationszeit wird angepasst, damit auch Fahrplansessions ausgeführt werden können, die nicht dem aktuellen Datum der Realzeit entsprechen. Für die Umwandlung des Datums werden die Zeiten mittels der Funktion *getUhrzeit()*\* in das *hh:mm:ss*-Format umgewandelt und mit derselben Funktion wieder in das Unix-Timestamp-Format zurück umgewandelt.

Für die Ermittlung der Realzeit und der Zeitdifferenz zwischen Real- und Simulationszeit wird die Funktion *time()* aufgerufen, mit der Start-Simulationszeit verglichen und unter der Variable *\$timeDifference* abgespeichert.

Die Differenz zwischen Real- und Simulationszeit ist essenziell, damit die Fahrzeuge zur richtigen Zeit die Echtzeitdaten übermittelt bekommen und die Realzeit in die Simulationszeit umgewandelt werden kann, ohne bei jeder Umrechnung die Funktion *getUhrzeit()*\* aufzurufen.

### 3.3 Ermittlung aller Fahrzeuge im eingleisigen Netz und den zugehörigen Daten

Das eingleisige Netz des EBUfs kann mittels der RailCom-Technik und den Decodern in den Fahrzeugen ermitteln, welches Fahrzeug aktuell welche Infra-Abschnitte belegt. Belegt ein Fahrzeug einen Infra-Abschnitt, wird in der Tabelle *fma* in der Spalte *decoder\_adresse* die Adresse des Fahrzeugs hinterlegt und in der *infra\_zustand*-Tabelle in der Spalte *dir* der Wert 1 hinterlegt. Durch diese Informationen werden alle Fahrzeuge, die sich beim Start des Programms im eingleisigen Netz befinden, mit der Funktion *findTrainsOnTheTracks()* (*functions.php*) eingelesen und die zugehörige Adresse wird der Funktion *prepareTrainForRide()* (*functions.php*) übergeben. Für jedes Fahrzeug, welches dieser Funktion übergeben wird, wird in dem Array *\$allUsedTrains* ein neuer

Eintrag erstellt, für jedes Fahrzeug die exakte Position bestimmt und der Fahrplan geladen. Das Array *\$allUsedTrains* beinhaltet alle Fahrzeuge, die aktuell von der Fahrzeugsteuerung berücksichtigt werden und deren zugehörige Informationen, wobei der Index der ID des Fahrzeugs entspricht.

Bei der Positionsbestimmung wird davon ausgegangen, dass die Fahrzeuge direkt vor dem zugehörigen Signal stehen, da ansonsten die Position nicht exakt ermittelt werden kann. Belegt ein Fahrzeug mehrere Infra-Abschnitte, wird mittels der Fahrtrichtung der Züge der Infra-Abschnitt ermittelt, in dem sich der Zugkopf befindet. Die aktuelle Position wird daraufhin mit dem Infra-Abschnitt und der relativen Position (in Metern) innerhalb des Abschnitts angegeben. Es wird davon ausgegangen, dass das Fahrzeug sich direkt vor dem Signal befindet, wodurch die relative Position der Infra-Abschnittslänge entspricht.

Für die Überprüfung, ob ein Fahrzeug nach Fahrplan fährt, wird die Funktion *getFahrzeugZugIds()*\* (*functions\_ebuef.php*) aufgerufen. Wenn einem Fahrzeug kein Fahrplan zugewiesen wurde (Rückgabewert der Funktion *getFahrzeugZugIds()*\* (*functions\_ebuef.php*) ist ein leeres Array), wird in dem *\$allUsedTrains*-Array dem Fahrzeug unter dem Eintrag *operates\_on\_timetable* der Wert *false* zugewiesen. In dem Fall, dass für das Fahrzeug ein Fahrplan hinterlegt ist (Rückgabewert der Funktion *getFahrzeugZugIds()*\* (*functions\_ebuef.php*) ist ein Array mit allen Zug-IDs), wird mittels der Funktion *getNextBetriebsstellen()* (*functions.php*) der Fahrplan für den ersten Eintrag des Zug-ID Arrays aus der Datenbank geladen. Der Fahrplan wird in dem *\$allUsedTrains*-Array in dem *next\_betriebsstellen\_data*-Array hinterlegt, welches für jede Betriebsstelle ein Array mit den benötigten Daten enthält. Die Indizierung dieser Einträge entspricht dabei den natürlichen Zahlen in aufsteigender Reihenfolge angefangen bei der 0 ( $\mathbb{N}_0$ ). Hierbei werden alle Betriebsstellen hinzugefügt, bei denen ein fahrplanmäßiger Halt vorgesehen ist. Damit ein Fahrzeug nicht erst losfahren kann, wenn die Fahrstraße bis zur nächsten Betriebsstelle mit fahrplanmäßigem Halt gestellt ist, werden auch alle Betriebsstellen ohne fahrplanmäßigem Halt hinzugefügt, welche eindeutig einem Infra-Abschnitt zugeordnet sind (*\$cacheZwischenhaltepunkte*). Das hat den Vorteil, dass Fahrzeuge losfahren können, auch wenn die Fahrstraße noch nicht bis zum nächsten fahrplanmäßigen Halt gestellt ist, das aber nur machen, wenn sichergestellt werden kann, dass die Zwischen-Betriebsstelle auf der Strecke zum nächsten fahrplanmäßigen Halt liegt. In Tabelle 2 ist für eine bessere Übersicht der Aufbau eines Betriebsstellen-Eintrags abgebildet. Für die Ermittlung der Ankunfts- und Abfahrzeiten wird die Funktion *getFahrplanzeiten()*\* (*functions\_ebuef.php*) aufgerufen, welche als Parameter den Namen der Betriebsstelle und die Zug-ID übergeben bekommt. Die zurückgegebenen Daten werden unter dem Eintrag *zeiten* abgespeichert und um

Bezeichnung	Funktion
<i>is_on_fahrstrasse</i> (Boolescher Wert)	Befindet sich die Betriebsstelle auf der Fahrstraße
<i>betriebsstelle</i> (String)	Name der Betriebsstelle
<i>zeiten</i> (Array)	Verspätung und Ankunfts- und Abfahrtszeiten (siehe Tabelle 3)
<i>haltepunkte</i> (Array)	Alle zugehörigen Infra-Abschnitte
<i>fahrplanhalt</i> (Boolescher Wert)	Ist diese Betriebsstelle ein Fahrplanhalt

Tabelle 2: Aufbau eines Arrays in *next\_betriebsstellen\_data*

den Eintrag *verspaetung* ergänzt. Zudem werden die Ankunfts- und Abfahrtszeiten in das Unix-Timestamp-Format mittels der Funktion *getUhrzeit()*\* (*functions\_ebuef.php*) umgewandelt. Der Aufbau des *zeiten*-Arrays ist in der Tabelle 3 dargestellt. Für die Überprüfung, ob eine Betriebsstelle durch die aktuelle Fahrstraße erreichbar ist, müssen den Betriebsstellen die Infra-Abschnitte zugeordnet werden. Dafür werden mit Hilfe der Arrays *\$cacheZwischenhaltepunkte* und *\$cacheHaltepunkte*, jeder Betriebsstelle mögliche Infra-Abschnitte zugeordnet. Die Arrays sind so aufgebaut, dass jeder Betriebsstelle für jede Richtung alle Infra-Abschnitte zugeteilt sind, welchen ein Ausfahrtsignal zugeordnet ist.

Nach der Zuordnung der Infra-Abschnitte zu den Betriebsstellen, wird anhand der aktuellen Positionen der Fahrzeuge überprüft, ob die Fahrzeuge an einer Betriebsstelle des Fahrplans stehen. Stimmt der aktuelle Infra-Abschnitt eines Fahrzeugs mit dem einer Betriebsstelle überein, wird dieser und allen vorherigen der Wert *true* unter der Variablen *angekommen* zugewiesen. Dadurch können Fahrzeuge auch nach Fahrplan fahren, wenn diese nicht an der ersten Betriebsstelle des Fahrplans stehen.

### 3.4 Berechnung der Fahrtverläufe aller Fahrzeuge

Nachdem für alle Fahrzeuge die Fahrplandaten (falls vorhanden) hinterlegt wurden, wird für jedes Fahrzeug die aktuelle Fahrstraße ermittelt. Dafür wird die Funktion *calculateNextSections()* (*functions.php*) aufgerufen und das Array *\$allUsedTrains* für jedes Fahrzeug um die Einträge *next\_sections*, *next\_lenghts* und *next\_v\_max* als Array ergänzt. Diese Arrays speichern die IDs, Längen und zulässigen Höchstgeschwindigkeiten der nächsten Infra-Abschnitte ab, welche auf der Fahrstraße liegen.

Im ersten Schritt wird überprüft, ob das Fahrzeug aktuell in einem Infra-Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist. Wenn das der Fall ist, wird

Bezeichnung	Funktion
<i>ankunft_soll</i> (String)	Ankunftszeit (hh:mm:ss)
<i>abfahrt_soll</i> (String)	Abfahrtszeit (hh:mm:ss)
<i>ankunft_soll_timestamp</i> (Integer)	Ankunftszeit (Unixtimestamp)
<i>abfahrt_soll_timestamp</i> (Integer)	Abfahrtszeit (Unixtimestamp)
<i>fahrtrichtung</i> (Array)	Fahrtrichtung (Eintrag aus der Tabelle <i>fahrplan_sessionfahrplan</i> )
<i>ist_durchfahrt</i> (Integer)	Fahrplanhalt (Eintrag aus der Tabelle <i>fahrplan_sessionfahrplan</i> )
<i>used_haltepunkt</i> (Integer)	Infra-Abschnitt der Betriebsstelle, welcher auf der Fahrstraße liegt
<i>wendet</i> (Integer)	Wendeauftrag nach Erreichen der Betriebsstelle
<i>verspaetung</i> (Integer)	Verspätung, mit der das Fahrzeug diese Betriebsstelle erreicht hat

Tabelle 3: Aufbau des *zeiten*-Arrays in *next\_betriebsstellen\_data*

den Arrays *next\_sections*, *next\_lenghts* und *next\_v\_max* ein leeres Array zugewiesen. Wenn das Fahrzeug aktuell nicht in einem Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist, wird über die Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*) die aktuelle Fahrstraße ermittelt und der Rückgabewert der Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*) in dem *\$allUsedTrains*-Array unter dem Eintrag *last\_get\_naechste\_abschnitte* gespeichert. Diese Speicherung ist notwendig, um zu überprüfen, ob sich die Fahrstraße geändert hat.

Nach der Ermittlung der Fahrstraße und der Zuordnung der Infra-Abschnitte zu den Betriebsstellen wird im nächsten Schritt überprüft, welche Betriebsstellen des Fahrplans auf der aktuellen Fahrstraße liegen. Dafür iteriert die Funktion *checkIfFahrstrasseIsCorrect()* (*functions.php*) in aufsteigender Reihenfolge über alle Betriebsstellen der Fahrzeuge und die *haltepunkte* der Betriebsstellen werden mit den Werten aus dem Array *next\_sections* verglichen. Bei jedem Aufruf der Funktion wird dem Fahrzeug anfangs (falls das Fahrzeug nach Fahrplan fährt) in dem Array *\$allUsedTrains* der Eintrag *fahrstrasse\_is\_correct* der Wert *false* zugewiesen und erst auf *true* gesetzt, wenn eine Betriebsstelle auf der Fahrstraße liegt. Bei dem Iterieren über die Betriebsstellen wird jeder Betriebsstelle anfangs der Wert *false* für den Eintrag *is\_on\_fahrstrasse* zugeordnet und sobald ein Infra-Abschnitt einer Betriebsstelle in dem Array *next\_sections* ebenfalls vorhanden ist, wird dem Eintrag *is\_on\_fahrstrasse* der Wert *true* zugewiesen und unter dem Eintrag *used\_haltepunkt* der Infra-Abschnitt gespeichert, welcher auf

der Fahrstraße liegt. Bei dem Iterieren über alle Betriebsstellen werden nur die Betriebsstellen beachtet, welche das Fahrzeug noch nicht erreicht hat (*angekommen* == *false*). Für Fahrzeuge ohne Fahrplan wird der Eintrag *fahrstrasse\_is\_correct* direkt auf *true* gesetzt.

Durch die Ermittlung der Fahrstraße kann für jedes Fahrzeug der Fahrtverlauf berechnet werden. Für die Berechnung der Fahrtverläufe wird für jedes Fahrzeug die Funktion *calculateFahrtverlauf()* (*functions.php*) aufgerufen und innerhalb der Funktion überprüft, ob die Fahrstraße richtig eingestellt ist (*fahrstrasse\_is\_correct* == *true*). Wenn die Fahrstraße richtig eingestellt ist, wird zwischen Fahrzeugen unterschieden, die nach Fahrplan fahren und Fahrzeugen, die keinen Fahrplan haben.

Für Fahrzeuge mit Fahrplan muss im ersten Schritt die nächste Betriebsstelle ermittelt werden, an der das Fahrzeug anhalten muss. Dafür wird mit einer *for*-Schleife über alle in *next\_betriebsstellen\_data* hinterlegten Betriebsstellen iteriert, die das Fahrzeug noch nicht angefahren hat (*angekommen* == *false*), die auf der Fahrstraße liegen (*is\_on\_fahrstrasse* == *true*) und die ein fahrplanmäßiger Halt sind (*fahrplanhalt* == *true*). Sobald eine Betriebsstelle gefunden wurde, wird die *for*-Schleife abgebrochen und der Index der Betriebsstelle als *\$nextBetriebsstelleIndex* abgespeichert. Sollte unter den nächsten Betriebsstellen keine dabei sein, auf die diese Kriterien zutreffen, wird in einer zweiten *for*-Schleife nach den selben Kriterien (außer dem des fahrplanmäßigen Halts) nach einer Betriebsstelle gesucht und sobald eine Betriebsstelle gefunden wurde, wird die Schleife abgebrochen und der Index der Betriebsstelle unter der Variablen *\$nextBetriebsstelleIndex* abgespeichert. Sollte eine nächste Betriebsstelle für das Fahrzeug existieren wird in einer dritten *for*-Schleife überprüft, ob zwischen der aktuellen Position und der nächsten Betriebsstelle eine Betriebsstelle ist, bei der das Fahrzeug einen Wendeauftrag bekommt. Sollte eine solche Betriebsstelle existieren, wird diese unter der Variablen *\$nextBetriebsstelleIndex* abgespeichert. In dem Fall, dass keine nächste Betriebsstelle ermittelt werden konnte und das Fahrzeug aktuell eine Geschwindigkeit hat, für die gilt:  $v > 0 \text{ km/h}$ , wird eine Gefahrenbremsung eingeleitet (siehe Kapitel 4.9).

Für alle Fahrzeuge, für die eine nächste Betriebsstelle ermittelt werden konnte, werden im Folgenden alle notwendigen Daten ermittelt. Dazu zählt, ob die Fahrzeuge nach dem Erreichen der Betriebsstelle einen Wendeauftrag erhalten sollen (*wendet*-Eintrag der nächsten Betriebsstelle), in welchen Infra-Abschnitt das Fahrzeug zum Stehen kommen soll (*used\_haltepunkt*-Eintrag der nächsten Betriebsstelle) und an welcher relativen Position innerhalb des Abschnitts das Fahrzeug angehalten soll (Länge des Infra-Abschnitts). Neben den Informationen zur Position müssen die Informationen



zur Zeit ermittelt werden.

Für die Ermittlung der Ankunftszeit muss neben dem zugehörigen Eintrag *ankunft\_soll\_timestamp* der Betriebsstelle die Verspätung berücksichtigt werden. Aus diesem Grund wird im ersten Schritt die zuletzt angefahren Betriebsstelle unter der Variablen *\$prevBetriebsstelle* abgespeichert. Sollte die nächste Betriebsstelle der erste fahrplanmäßige Halt sein (Ankunftszeit nicht definiert), so wird als Start- und Zielzeit (*\$startTime* und *\$endTime*) die aktuelle Simulationszeit verwendet. Wenn die nächste Betriebsstelle nicht dem ersten fahrplanmäßigen Halt entspricht, wird als Zielzeit die Ankunftszeit der Betriebsstelle festgelegt und als Startzeit die Abfahrtszeit der vorherigen Betriebsstelle (*\$prevBetriebsstelle*) plus die eingetragene Verspätung der vorherigen Betriebsstelle. Sollte es zu dem Zeitpunkt der Berechnung keine vorherige Betriebsstelle geben (*\$prevBetriebsstelle == null*), so wird als Startzeit die aktuelle Simulationszeit gewählt. Im zweiten Schritt wird überprüft, ob die Startzeit kleiner als die aktuelle Simulationszeit ist und wenn das der Fall ist, wird die Startzeit gleich der Simulationszeit gesetzt. Im dritten Schritt wird die Startzeit gleich der frühestmöglichen Startzeit des Fahrzeugs (*earliest\_possible\_start\_time*-Eintrag des Fahrzeugs) gesetzt, falls die Startzeit kleiner ist. Der Eintrag *earliest\_possible\_start\_time* der Züge gibt die frühestmögliche Abfahrtszeit der Züge an und wird zum Beispiel bei einem Wendeauftrag auf die aktuelle Simulationszeit gesetzt und um 30 s erhöht.

Für alle Fahrzeuge, die ohne Fahrplan unterwegs sind, wird als Ziel-Infra-Abschnitt der letzte Infra-Abschnitt aus dem Array *last\_get\_naechste\_abschnitte* verwendet, welchem ein Signal zugeordnet ist. Die Ziel-Position innerhalb des Infra-Abschnitts entspricht dabei ebenfalls der Länge des Abschnitts und die Überprüfung, ob ein Wendeauftrag nach dem Erreichen des Ziel-Infra-Abschnitts dem Fahrzeug übermittelt werden soll, wird von dem Signalbegriff abgeleitet. Die Start- und Zielzeit entsprechen der aktuellen Simulationszeit, bzw. der *earliest\_possible\_start\_time*. Sollte keinem der nächsten Infra-Abschnitte aus dem *last\_get\_naechste\_abschnitte*-Array ein Signal zugeordnet sein und die aktuelle Geschwindigkeit des Fahrzeugs ist größer als 0 km/h sein, so wird eine Gefahrenbremsung eingeleitet. Andernfalls wird die Funktion an dieser Stelle abgebrochen und es wird wieder versucht einen Fahrtverlauf zu berechnen, wenn sich die Fahrstraße geändert hat.

Nach der Ermittlung aller notwendigen Daten für die Berechnung des Fahrtverlaufs, wird für jedes Fahrzeug die Funktion *updateNextSpeed()* (*functions\_fahrtverlauf.php*) aufgerufen, welche den Fahrtverlauf berechnet und in Kapitel 4 im Detail beschrieben wird. Wichtig an dieser Stelle ist der Rückgabewert der Funktion, welcher für Fahrzeuge mit Fahrplan die Verspätung in Sekunden angibt, mit der das Fahrzeug die

Ziel-Betriebsstelle erreicht, und wird unter dem Eintrag *verspaetung* der zugehörigen Betriebsstelle gespeichert. Ob ein Fahrzeug eine Betriebsstelle mit einer Verspätung erreicht, kann nur ermittelt werden, wenn die Ankunftszeit definiert ist. Für den Fall, dass für ein Fahrzeug ein Fahrplan hinterlegt ist, das Fahrzeug in einem Infra-Abschnitt steht, welchem keine Betriebsstelle des Fahrplans zugeordnet ist und die Fahrstraße so eingestellt ist, dass das Fahrzeug den ersten fahrplanmäßigen Halt anfahren könnte, kann nicht ermittelt werden, ob das Fahrzeug diese Betriebsstelle mit einer Verspätung erreicht, da für den ersten fahrplanmäßigen Halt in der *MySQL*-Tabelle *fahrplan\_sessionfahrplan* keine Ankunftszeit hinterlegt ist. Aus diesem Grund, wurde in der Datei *global\_variables.php* die Variable *\$globalFirstHaltMinTime* definiert, welche angibt, wie lange ein Fahrzeug an der ersten Betriebsstelle des Fahrplans halten soll. Wenn diese Zeit eingehalten werden kann, wird das Fahrzeug (sofern die Fahrstraße richtig eingestellt ist) zur Abfahrtszeit die Betriebsstelle verlassen. Andernfalls gilt für die Verspätung der ersten Betriebsstelle:

$$\text{Verspätung} = \text{Ankunftszeit} + \$globalFirstHaltMinTime - \text{Abfahrtszeit}$$

### 3.5 Übermittlung der Echtzeitdaten an die Fahrzeuge

Nach dem Aufruf der Funktion *updateNextSpeed()* (*functions\_fahrtverlauf.php*) sind für alle Fahrzeuge – für die ein Fahrtverlauf berechnet wurde – in dem Array *\$allTimes* alle Echtzeitdaten enthalten. Das Array beinhaltet für jedes Fahrzeug wiederum ein Array, welches unter der Adresse des Fahrzeugs abgespeichert ist, und beinhaltet alle Echtzeitdaten eines Fahrzeugs. Der Aufbau eines Array mit Echtzeitdaten ist in Tabelle 4 dargestellt. In einer *while*-Schleife wird über alle Einträge des *\$allTimes*-Arrays iteriert und überprüft, ob der erste Eintrag eines Fahrzeugs Echtzeitdaten enthält, welche an das Fahrzeug übermittelt werden müssen. Dafür wird der Eintrag *live\_time* mit der aktuellen Simulationszeit verglichen und die zugehörigen Echtzeitdaten an das Fahrzeug übermittelt, wenn der Eintrag *live\_time* kleiner als die aktuelle Simulationszeit ist. Nach jedem Durchlauf der *while*-Schleife wird diese mit der Funktion *sleep()* für 0,03 s pausiert. An dieser Stelle wurde sich für einen Wert von 0,03 s entschieden, da so die Position auf einen Meter genau bestimmt werden kann, wenn das Fahrzeug eine Geschwindigkeit von 120 km/h hat.

Wenn für ein Fahrzeug neue Echtzeitdaten vorliegen, wird im ersten Schritt überprüft, ob eine Geschwindigkeitsveränderung vorliegt (*live\_is\_speed\_change == true*) und die neue Geschwindigkeit (falls vorhanden) über die Funktion *sendFahrzeugbefehl()*\* (*functions\_ebuef.php*) dem Fahrzeug übergeben und mittels einer Terminal-Ausgabe

Bezeichnung	Funktion
<i>live_position</i> (Float)	absolute Position (kann weg...)
<i>live_speed</i> (Integer)	Geschwindigkeit des Fahrzeugs
<i>live_time</i> (Float)	Zeit der Übermittlung an das Fahrzeug
<i>live_relative_position</i> (Integer)	relative Position im Infra-Abschnitt
<i>live_section</i> (Integer)	Infra-Abschnitt
<i>live_is_speed_change</i> (Boolescher Wert)	Angabe, ob bei diesen Echtzeitdaten die Geschwindigkeit verändert wird
<i>live_target_reached</i> (Boolescher Wert)	Das Fahrzeug hat sein Ziel erreicht
<i>id</i> (String)	ID des Zugs
<i>wendet</i> (Boolescher Wert)	Angabe, ob ein Wendeauftrag durchgeführt werden soll
<i>betriebsstelle</i> (String)	Name der Betriebsstelle des nächsten Halts
<i>live_all_targets_reached</i> (Integer)	Index der Betriebsstelle, die erreicht wurde

Tabelle 4: Aufbau eines Eintrags aus dem *\$allTimes*-Array

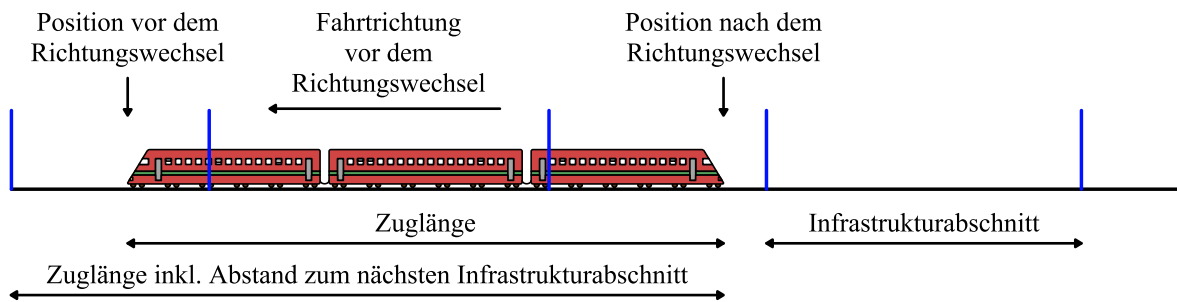


Abbildung 4: Eigene Darstellung der Positionsbestimmung bei einem Richtungswechsel

angezeigt. Im zweiten Schritt wird der aktuelle Infra-Abschnitt, die aktuelle Position innerhalb des Abschnitts und die Geschwindigkeit in dem Array *\$allUsedTrains* abgespeichert.

Sollte das Fahrzeug nach dem Ausführen der Echtzeitdaten einen Wendeauftrag bekommen und dementsprechend der Eintrag *wendet true* sein, so wird die Funktion *changeDirection()* (*functions.php*) aufgerufen. In der Funktion wird neben der Fahrtrichtungsänderung die neue Position ermittelt (die Position eines Fahrzeugs wird durch den Zugkopf beschrieben) und überprüft, ob die Fahrtrichtung geändert werden kann. Damit die Fahrtrichtungsänderung ebenfalls funktioniert, wenn das Fahrzeug nicht am Ende eines Infra-Abschnitts steht, wird für die Ermittlung der neuen Position auf die Fahrzeuglänge der Abstand bis zum Ende Infra-Abschnitts addiert (siehe Abbildung 4). Über den aktuellen und die folgenden Infra-Abschnitte (ermittelt durch die Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*), des aktuellen Infra-Abschnitts und der neuen Fahrtrichtung) wird iteriert und die Summe der Längen gebildet, bis die Fahrzeuglänge (zuzüglich des Abstands bis zum Ende des Infra-Abschnitts) überschritten wird. Der Infra-Abschnitt, in dem die Fahrzeuglänge inkl. des Abstands zum ersten Mal überschritten wird, entspricht dem Infra-Abschnitt der neuen Position.

Sollte die Länge aller nächsten Abschnitte inklusive des aktuellen Abschnitts in der Summe kleiner sein, als die Zuglänge inkl. dem Abstands bis zum Ende des Infra-Abschnitts, kann die neue Position nicht ermittelt werden und dem Fahrzeug wird eine Fehlermeldung übergeben, sodass das Fahrzeug nicht weiter fahren wird. Andernfalls wird die Richtung des Fahrzeugs in der Datenbank geändert und dem Fahrzeug mit der Funktion *sendFahrzeugbefehl()*\* (*functions\_ebuef.php*) die Geschwindigkeit *-4 km/h* (entspricht einem Wendeauftrag) übergeben.

Bei einem Fahrtverlauf kann es vorkommen, dass Fahrzeuge mit Fahrplan auf der

	Fährt jetzt ohne Fahrplan	Fährt jetzt nach Fahrplan
Fuhr davor ohne Fahrplan	1. Fall	2. Fall
Fuhr davor nach Fahrplan	3. Fall	4. Fall

Tabelle 5: Verhalten eines Fahrzeugs nach dem Erreichen des Ziels

Fahrt mehrere Betriebsstellen passieren. Damit dem Eintrag *angekommen* dieser Betriebsstellen auch der Wert *true* zugewiesen werden kann, wird überprüft, ob in den Echtzeitdaten dem Eintrag *live\_all\_targets\_reached* ein Wert zugewiesen ist. Dieser Eintrag enthält – falls das Fahrzeug eine Betriebsstelle erreicht hat – den Index der Betriebsstelle und weist der Betriebsstelle unter dem Eintrag *angekommen* den Wert *true* zu.

Wenn die letzten Echtzeitdaten eines Fahrzeugs übermittelt wurden (*live\_target\_reached == true*) und das Fahrzeug dementsprechend zum Stehen gekommen ist, wird überprüft, wie sich das Fahrzeug als nächstes verhalten soll. Dafür wird zwischen vier Fällen (siehe Tabelle 5) unterschieden. Für die Überprüfung, ob sich der Fahrplan eines Fahrzeugs geändert hat, wird über die Funktion *getFahrzeugZugIds()* (*functions\_ebuef.php*) die aktuelle Zug-ID abgefragt und mit der vorherigen verglichen. In dem **1. Fall** (alte und neue Zug-ID haben beide den Wert *null*) werden dem Fahrzeug keine neue Daten übergeben und ein neuer Fahrtverlauf wird versucht zu berechnen, sobald die Fahrstraße sich verändert hat. In dem **2. und 4. Fall** wird die neue Zug-ID dem Fahrzeug übergeben, der Eintrag *operates\_on\_timetable* auf *true* gesetzt und die Funktionen *getFahrplanAndPositionForOneTrain()* (*functions.php*), *addStopsectionsForTimetable()* (*functions.php*), *calculateNextSections()* (*functions\_fahrtverlauf.php*), *checkIfFahrstrasseIsCorrect()* (*functions.php*) und *calculateFahrtverlauf()* (*functions.php*) aufgerufen. Abgesehen von der ersten Funktion, werden diese Funktionen auch beim Start des Programms ausgeführt, welcher in Kapitel 3.3 beschrieben wird. Die Funktion *getFahrplanAndPositionForOneTrain()* (*functions.php*) ähnelt der in Kapitel 3.3 beschriebenen Funktion *prepareTrainForRide()* (*functions.php*), fügt aber nur die Position und den Fahrplan hinzu, da alle anderen Daten schon eingelesen wurden. In dem **3. Fall** (die neu ermittelte Zug-ID hat den Wert *null*) wird der Eintrag *operates\_on\_timetable* auf *false* gesetzt und die Funktionen *calculateNextSections()* (*functions.php*) und *calculateFahrtverlauf()* (*functions.php*) aufgerufen.

### 3.6 Überprüfung nach einer Änderung der Fahrstraße

Für die Überprüfung, ob sich die Fahrstraße der Züge verändert hat, wird in regelmäßigen Abständen die Fahrstraße der Fahrzeuge ermittelt und mit der aktuell hinterlegten Fahrstraße verglichen. Das Intervall, in dem diese Überprüfung stattfindet kann über die Variable *\$timeCheckFahrstrasseInterval* (*fahrzeugsteuerung.php*) festgelegt werden und ist standardgemäß auf 3 Sekunden festgelegt. Bei der Ermittlung und dem Vergleich der Fahrstraße wird für jedes Fahrzeug die Funktion *compareTwo-NaechsteAbschnitte()* (*functions.php*) aufgerufen. Innerhalb dieser Funktion wird die in Kapitel 3.3 erläuterte Funktion *calculateNextSections()* (*functions.php*) aufgerufen, mit dem Unterschied, dass die ermittelten nächsten Infra-Abschnitte inkl. der Längen und zulässigen Höchstgeschwindigkeiten nicht dem Fahrzeug hinterlegt werden, sondern lokal in der Funktion gespeichert. Damit die ermittelten Daten für ein Fahrzeug berechnet werden, aber nicht dem Fahrzeug hinterlegt werden, kann der Parameter *\$writeResultToTrain* der Funktion *calculateNextSections()* (*functions.php*) (standardgemäß auf *true* gesetzt) auf *false* gesetzt werden. Sollte sich die Fahrstraße geändert haben, wird mit der Funktion *checkIfFahrstrasseIsCorrect()* (*functions.php*) überprüft, ob die Fahrstraße dem Fahrplan (falls vorhanden) entspricht und im Anschluss die Funktion *calculateFahrtverlauf()* (*functions.php*) aufgerufen.

### 3.7 Neukalibrierung der Fahrzeugposition

Für eine genau Fahrzeugsteuerung ist die aktuelle Position der Züge essenziell und muss während der Fahrt kalibriert werden, damit Ungenauigkeiten ausgeglichen werden können. Dafür werden die Daten aus der *MySQL*-Tabelle *fahrzeuge\_abschnitte* benötigt, welche durch die Abschnittsüberwachung ermittelt werden. Die Abschnittsüberwachung schreibt für jedes Fahrzeug den aktuellen Infra-Abschnitt in die Datenbank, sobald der Zugkopf den Abschnitt befährt inklusive der aktuellen Zeit (Realzeit). Für jedes Fahrzeug, welches durch die Übermittlung der Echtzeitdaten in einen neuen Infra-Abschnitt einfährt und seit der Einfahrt in den Abschnitt die Geschwindigkeit nicht verändert hat, wird die aktuelle Position neu ermittelt. Würde sich das Fahrzeug in einem Infra-Abschnitt befinden und hätte seit der Einfahrt die Geschwindigkeit angepasst, könnte mit der Fahrzeugsteuerung die Position nicht neu berechnet werden, da nicht bekannt ist, welche Strecke das Fahrzeug seit der Einfahrt zurückgelegt hat. Aus diesem Grund wird, sobald das Fahrzeug nach den Echtzeitdaten einen neuen Abschnitt befährt und aktuell nicht die Geschwindigkeit anpasst (*live\_is\_speed\_change == false*) dem Eintrag *calibrate\_section\_one* der aktuelle Infra-Abschnitt hinzugefügt und dem Eintrag *calibrate\_section\_two* wird ebenfalls der aktuelle Infra-Abschnitt

```

1 // Kalibriert die Position des Fahrzeugs neu anhand der Daten in der Tabelle
2 // 'fahrzeuge_abschnitte'
3 function getCalibratedPosition ($id, $speed) {
4     global $cacheFahrzeugeAbschnitte;
5     $DB = new DB_MySQL();
6     $positionReturn = $DB->select("SELECT '".DB_TABLE_FAHRZEUGE_ABSCHNITTE."'.'.
        ↳ infra_id','".DB_TABLE_FAHRZEUGE_ABSCHNITTE."'.'.unixtimestamp' FROM '".
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE."' WHERE '".
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE."'.'.fahrzeug_id' = $id")[0];
7     unset($DB);
8     if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
9         if ($positionReturn->unixtimestamp == $cacheFahrzeugeAbschnitte[$id]["
            ↳ unixtimestamp"]) {
10             return array("possible" => false);
11         }
12     }
13     $timeDiff = time() - $positionReturn->unixtimestamp;
14     $position = ($speed / 3.6) * $timeDiff;
15     return array("section" => $positionReturn->infra_id, "position" => $position)
        ↳ ;
16 }

```

Code-Beispiel 3: *getCalibratedPosition()* (*functions\_db.php*)

hinzugefügt, wenn *calibrate\_section\_one* ein Wert zugewiesen ist und dieser nicht dem aktuellen Infra-Abschnitt der Echtzeitdaten entspricht. Sobald das Fahrzeug seine Geschwindigkeit anpasst (*live\_is\_speed\_change == true*), wird beiden Einträgen der Wert *null* zugewiesen. Dadurch ist dem Eintrag *calibrate\_section\_two* nur dann ein Infra-Abschnitt zugewiesen, wenn das Fahrzeug in diesem seit der Einfahrt die Geschwindigkeit nicht verändert hat. Wenn dem Eintrag *\$useRecalibration* aus der Datei *global\_variables.php* der Wert *true* zugewiesen ist, wird in regelmäßigen Abständen überprüft, ob eine Neukalibrierung möglich ist. Das Zeitintervall, in dem die Überprüfung stattfindet ist standardmäßig auf 3 Sekunden eingestellt, kann aber mittels der Variable *\$timeCheckCalibrationInterval* (*fahrzeugsteuerung.php*) angepasst werden.

Für die Neukalibrierung wird die Funktion *getCalibratedPosition()* (*functions.php*) (Code-Beispiel 3) aufgerufen, welche als Rückgabewert die aktuelle relative Position und den aktuellen Infra-Abschnitt zurückgibt.

Sollte die ermittelte Position innerhalb des Infra-Abschnitts größer als die Länge des Infra-Abschnitts sein, welche in dem Array *\$cacheInfraLaenge* abgespeichert ist, wird die Neukalibrierung nicht durchgeführt. Der aktuelle Infra-Abschnitt wird aus der Tabelle *fahrzeuge\_abschnitte* der *MySQL*-Datenbank geladen und durch die aktuelle

Geschwindigkeit des Fahrzeugs und die Differenz der Zeit zwischen dem Einfahren in den Infra-Abschnitt und der aktuellen Zeit wird die relative Position innerhalb des Infra-Abschnitts berechnet.

$\text{relative Position} = \text{Geschwindigkeit} \cdot \text{Zeitdifferenz (aktuelle Zeit} - \text{Zeit des Einfahrens)}$

### 3.8 Ermittlung von neuen Fahrzeugen im eingleisigen Netz

Die Fahrzeugsteuerung betrachtet neben den Fahrzeugen, welche sich schon zu Beginn des Programmstarts im eingleisigen Netz befinden auch alle Fahrzeuge, die nach dem Programmstart hinzugefügt werden. Für alle Fahrzeuge, die beim Start des Programms erkannt werden, wird in dem Array *\$allTrainsOnTheTrack* die zugehörige Adresse gespeichert (*findTrainsOnTheTracks()*) (*functions.php*). Für die Überprüfung, ob Fahrzeuge entfernt wurden oder neu hinzugekommen sind, wird die Funktion *updateAllTrainsOnTheTrack()* (*functions.php*) verwendet. Diese Funktion wird – wie die Neukalibrierung in Kapitel 3.7 – alle 3 Sekunden ausgeführt. Bei dem Aufruf der Funktion werden alle Fahrzeuge geladen, denen in der *fma*-Tabelle aus der Datenbank ein Infra-Abschnitt zugeordnet ist und mit dem Array *\$allTrainsOnTheTrack* verglichen. Fahrzeugadressen, die nicht in dem Array hinterlegt sind, werden in dem Rückgabe-Array unter dem Eintrag *new* zurückgegeben und alle Fahrzeugadressen, die in dem Array enthalten sind, aber bei dem Aufruf der Funktion keinem Infra-Abschnitt zugeordnet sind, werden in dem Rückgabe-Array unter dem Eintrag *removed* zurückgegeben. Nach dem Aufruf der Funktion, werden für alle neuen Fahrzeuge die Funktion *prepareTrainForRide()* (*functions.php*), *addStopsectionsForTimetable()* (*functions.php*), *calculateNextSections()* (*functions.php*), *checkIfTrainReachedHaltepunkt()* (*functions.php*), *checkIfFahrstrasseIsCorrect()* (*functions.php*) und *calculateFahrtverlauf()* (*functions.php*) aufgerufen (siehe Kapitel 3.3 und 3.4). Alle entfernten Fahrzeuge werden aus dem Array *\$allUsedTrains* entfernt und somit nicht mehr von der Fahrzeugsteuerung beachtet.

### 3.9 Fehlerbehebung von Fahrzeugen

Wenn es bei einem Fahrzeug zu einem Konflikt kommt, der eine Steuerung des Fahrzeugs verhindert, wird dem Fahrzeug eine Fehlermeldungs-ID unter dem Eintrag *error* in dem *\$allUsedTrains* zugewiesen. Fahrzeuge, denen eine Fehlermeldung zugeordnet wurde, werden ab diesem Zeitpunkt nicht weiter von der Fahrzeugsteuerung berücksichtigt. Für das Erkennen von Fahrzeugen mit Fehlermeldungen, wird in regelmäßigen Zeitintervallen (*\$timeCheckAllTrainStatusInterval*) die Funktion *showErrors()*



```

1 // Gibt für alle Fahrzeuge die vorhanden Fehlermeldungen an.
2 function showErrors() {
3
4     global $allUsedTrains;
5     global $trainErrors;
6
7     $foundError = false;
8     echo "Hier werden für alle Züge mögliche Fehler angezeigt:\n\n";
9
10    foreach ($allUsedTrains as $trainIndex => $trainValue) {
11        if (sizeof($trainValue["error"]) != 0) {
12            $foundError = true;
13            echo "Zug ID: ", $trainValue["id"], "\n";
14            $index = 1;
15
16            foreach ($trainValue["error"] as $error) {
17                echo "\t", $index, ". Fehler:\t", $trainErrors[$error], "\n";
18                $index++;
19            }
20
21            echo "\n";
22        }
23    }
24
25    if (!$foundError) {
26        echo "Keiner der Züge hat eine Fehlermeldung.\n";
27    }
28 }

```

Code-Beispiel 4: *showErrors()* (*functions.php*)

(*functions.php*) aufgerufen, welche die Fehlermeldungen aller Fahrzeuge ausgibt (Code-Beispiel 4). Für das Beheben einer Fehlermeldung muss das Fahrzeug händisch vom Schienennetz genommen werden, gewartet werden, bis die Fahrzeugsteuerung das Entfernen registriert hat und das Fahrzeug wieder händisch auf das Schienennetz gesetzt werden.

Die möglichen Fehlermeldungen sind in dem Array *\$trainErrors* gespeichert und können um beliebig viele weitere Fehlermeldungen ergänzt werden. Für die Implementierung einer neuen Fehlermeldung muss lediglich die Fehlermeldungs-ID (Index der Fehlermeldung in dem *\$trainErrors*-Array) dem Eintrag *error* aus dem *\$allUsedTrains*-Array hinzugefügt werden, sobald der Konflikt in der Fahrzeugsteuerung auftritt. In Tabelle 6 sind alle Fehlermeldungen aufgelistet, welche aktuell in der Fahrzeugsteuerung implementiert sind.

Fehlermeldungs-ID	Beschreibung
0	Fahrtrichtung des Fahrzeugs musste geändert werden und die Positionsbestimmung war nicht möglich
1	In der Datenbank ist für das Fahrzeug keine Zuglänge angegeben
2	In der Datenbank ist für das Fahrzeug keine v_max angegeben
3	Das Fahrzeug musste eine Gefahrenbremsung durchführen

Tabelle 6: Übersicht der Fehlermeldungen

## 4 Berechnung des Fahrtverlaufs

Der Fahrtverlauf eines Fahrzeuges wird bei der Berechnung auf zwei verschiedenen Arten gespeichert. Einmal in so genannten *\$keyPoints*, welche in einem Array die Start- und Zielgeschwindigkeit (*speed\_0* und *speed\_1*), die Start- und Endposition (*position\_0* und *position\_1*) und die Start- und Endzeit (*time\_0* und *time\_1*) der einzelnen Beschleunigungen bzw. Verzögerungen abspeichern. Für die Überprüfung, ob ein Fahrzeug die zulässige Höchstgeschwindigkeit in einem Infra-Abschnitt überschreitet, und die exakte Positionsbestimmung, werden die *\$keyPoints* in Echtzeitdaten umgewandelt. Die Konvertierung der *\$keyPoints* in die Echtzeitdaten wird in Kapitel 4.3 im Detail beschrieben.

Als Grundlage für die Berechnung des Fahrtverlaufs werden zudem die Variablen *\$indexCurrentSection* und *\$indexTargetSection* benötigt, welche die Indexe der Start- und Ziel-Infra-Abschnitte in Bezug auf das Array *\$next\_sections* beschreiben und die Arrays *\$cumulativeSectionLengthStart* und *\$cumulativeSectionLengthEnd*, welche für jeden Infra-Abschnitt den Abstand zur aktuellen Position von dem Anfang und dem Ende des Infra-Abschnitts angeben.

Der Fahrtverlauf wird mit der Funktion *updateNextSpeed()* (*functions\_fahrtverlauf.php*) berechnet, welche als Parameter unter anderem die Zugdaten aus dem *\$all-UsedTrains*-Array, Start- und Endzeit der Fahrt (*\$startTime* und *\$endTime*), den Ziel-Infra-Abschnitt (*\$targetSection*) und die relative Position in dem Ziel-Infra-Abschnitt (*\$targetPosition*) übergeben bekommt.

In dem folgenden Abschnitt werden die einzelnen Schritte beschrieben, die durchlaufen werden, um den optimalen Fahrtverlauf zu berechnen. In der Darstellung 5 wird der Ablauf grob schematisch dargestellt.

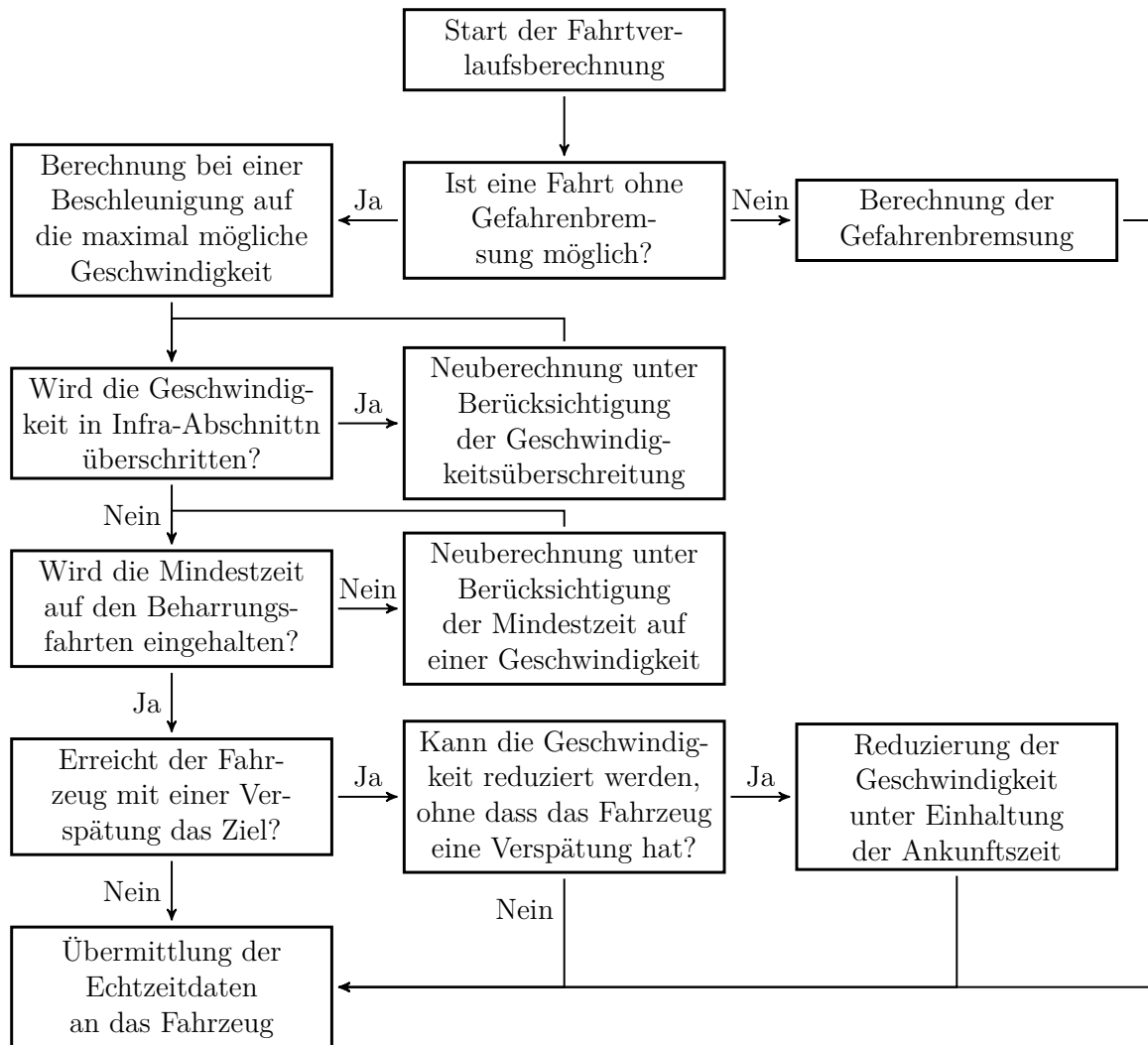


Abbildung 5: Ablaufplan der Fahrverlaufsrechnung

Bezeichnung	Funktion
$\$keyPoint$ (Array)	Beschreibt eine Beschleunigung bzw. Verzögerung ( $position\_0$ , $position\_1$ , $time\_0$ , $time\_1$ , $speed\_0$ , $speed\_1$ )
$\$next\_section$ (Array)	IDs aller Infra-Abschnitte
$\$next\_lengths$ (Array)	Längen aller Infra-Abschnitte
$\$next\_v\_max$ (Array)	Höchstgeschwindigkeit aller Infra-Abschnitte
$\$indexCurrentSection$ (Integer)	Index des aktuellen Infra-Abschnitts
$\$indexTargetSection$ (Integer)	Index des Ziel-Infra-Abschnitts
$\$cumulativeSectionLengthStart$ (Array)	Absolute Startposition aller Infra-Abschnitte
$\$cumulativeSectionLengthEnd$ (Array)	Absolute Endposition aller Infra-Abschnitte
$\$strainPositionChange$ (Array)	Alle absoluten Positionen des Fahrtverlaufs
$\$strainSpeedChange$ (Array)	Alle Geschwindigkeiten des Fahrtverlaufs

Tabelle 7: Beschreibung der verwendeten Variablen für die Fahrtverlaufsberechnung

#### 4.1 Ermittlung der Start- und Endposition der einzelnen Infra-Abschnitte unter Berücksichtigung der Zuglängen

Für die Berechnung eines exemplarischen Fahrtverlaufs wurden die in Tabelle 8 definierten Infra-Abschnitte verwendet. Diese Infra-Abschnitte wurden so gewählt, dass alle Funktionen und die Allgemeingültigkeit des Algorithmus gezeigt werden können und existieren in dieser Form im EBUf nicht. Als exemplarisch gewählte Zugdaten wurden die in Tabelle 9 definierten Daten verwendet.

Die zuvor ermittelten nächsten Infra-Abschnitte inklusive derer Längen und zulässigen Höchstgeschwindigkeit müssen für die Berechnung des Fahrtverlaufs angepasst werden, da ein Fahrzeug erst beschleunigen darf, wenn das komplette Fahrzeug in den Infra-Abschnitt eingefahren ist. In Darstellung 6 sind die Infra-Abschnitte dargestellt, wie sie von der Fahrzeugsteuerung ermittelt wurden. Dabei werden alle Infra-Abschnitte, die das Fahrzeug bereits durchfahren hat oder hinter dem Ziel-Infra-Abschnitt liegen nicht dargestellt. Zudem wird in dem aktuellen Infra-Abschnitt die relative Position von der Länge abgezogen und der Ziel-Infra-Abschnitt wird nur bis zur relativen Zielposition abgebildet. Dementsprechend ist der erste Infra-Abschnitt in der Darstellung 6 der Infra-Abschnitt mit der ID 1001. Dieser hat aufgrund der aktuellen relativen Position des Fahrzeugs eine Länge von 290 m. Und der letzte Infra-Abschnitt ist der Infra-Abschnitt mit der ID 1010 und einer Länge von eben-

Infra-Abschnitts-ID	Länge	zulässige Höchstgeschwindigkeit
1000	300 <i>m</i>	120 <i>km/h</i>
1001	400 <i>m</i>	120 <i>km/h</i>
1002	300 <i>m</i>	120 <i>km/h</i>
1003	400 <i>m</i>	90 <i>km/h</i>
1004	300 <i>m</i>	60 <i>km/h</i>
1005	200 <i>m</i>	60 <i>km/h</i>
1006	400 <i>m</i>	90 <i>km/h</i>
1007	500 <i>m</i>	120 <i>km/h</i>
1008	300 <i>m</i>	120 <i>km/h</i>
1009	400 <i>m</i>	100 <i>km/h</i>
1010	300 <i>m</i>	60 <i>km/h</i>
1011	300 <i>m</i>	40 <i>km/h</i>

Tabelle 8: Exemplarische Infra-Abschnitte

relative Startposition	10 <i>m</i>
relative Zielposition	290 <i>m</i>
aktueller Infra-Abschnitt	1001
Ziel-Infra-Abschnitt	1010
Startgeschwindigkeit	0 <i>km/h</i>
Zielgeschwindigkeit	0 <i>km/h</i>
Zuglänge	50 <i>m</i>
Bremsverzögerung	0,8 <i>m/s</i> <sup>2</sup>
Fahrplan vorhanden	ja
Zeit bis zur nächsten Betriebsstelle	210 <i>s</i>

Tabelle 9: Exemplarische Zugdaten

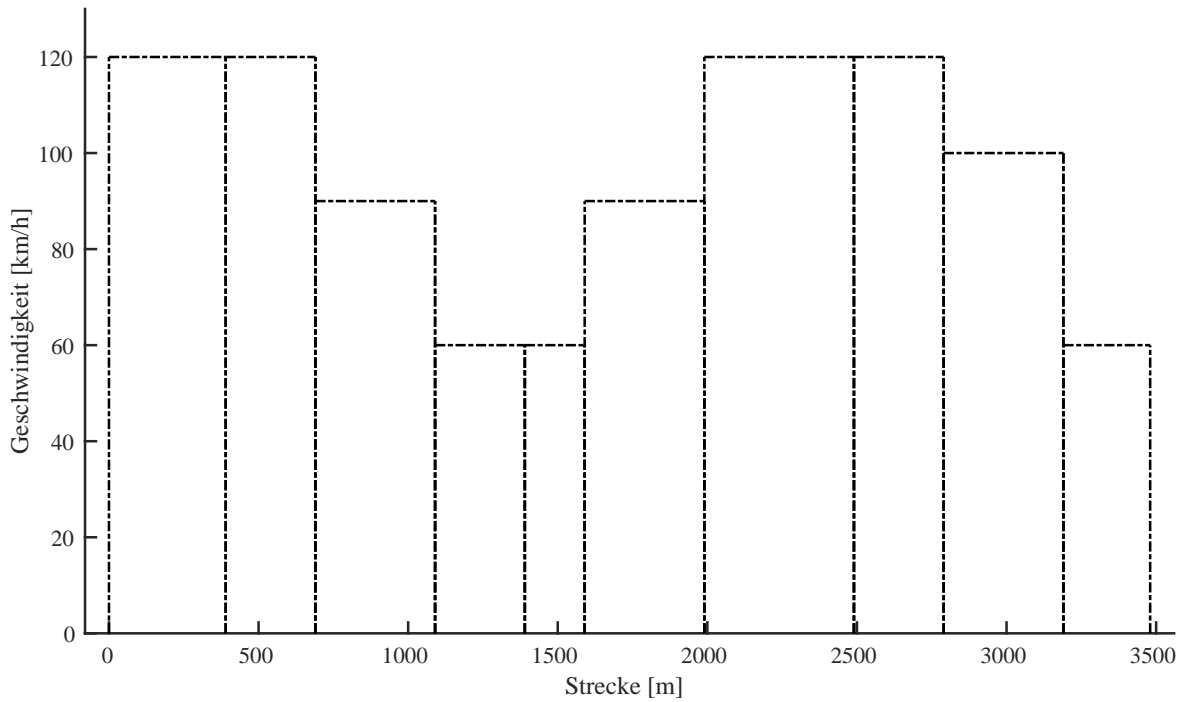


Abbildung 6: Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit

falls 290 m.

Bei der Berücksichtigung der Fahrzeuglänge wird mit einer *for*-Schleife über alle Infra-Abschnitte iteriert und die Zuglänge auf die Länge des Infra-Abschnitts addiert. Von dieser neu ermittelten Endposition des Infra-Abschnitts wird überprüft, ob zwischen der vorherigen Endposition und der neu ermittelten Endposition ein Infra-Abschnitt liegt, dessen zulässige Höchstgeschwindigkeit geringer ist, als die des ursprünglichen Infra-Abschnitts. Wenn dieser Fall eintritt, wird der Infra-Abschnitt nur so weit verlängert, dass keine Höchstgeschwindigkeit der folgenden Infra-Abschnitte überschritten wird. Nach der Ermittlung der neuen Endposition, startet die *for*-Schleife mit dem Infra-Abschnitt, in dem sich die Endposition sich befindet. Sobald der Ziel-Infra-Abschnitt erreicht wurde, wird die Schleife abgebrochen. Die neu ermittelten Infra-Abschnitte werden in den Arrays *\$next\_lengths\_mod* und *\$next\_v\_max\_mod* abgespeichert (analog zu den Arrays *\$next\_lengths* und *\$next\_v\_max*).

Durch diesen Algorithmus kann es dazu kommen, dass sich die Anzahl der Infra-Abschnitte verändert hat, wodurch die Infra-Abschnitte nicht mehr eindeutig mit der Infrastruktur-ID bezeichnet werden können. Mittels *\$next\_lengths\_mod* und *\$next\_v\_max\_mod* werden mit der Funktion *createCumulativeSections()* (*functions\_fahrtverlauf.php*) für jeden Infra-Abschnitt die absolute Start- und Endposition in den Arrays *\$cumulativeSectionLengthStartMod* und *\$cumulativeSectionLengthEndMod* ge-

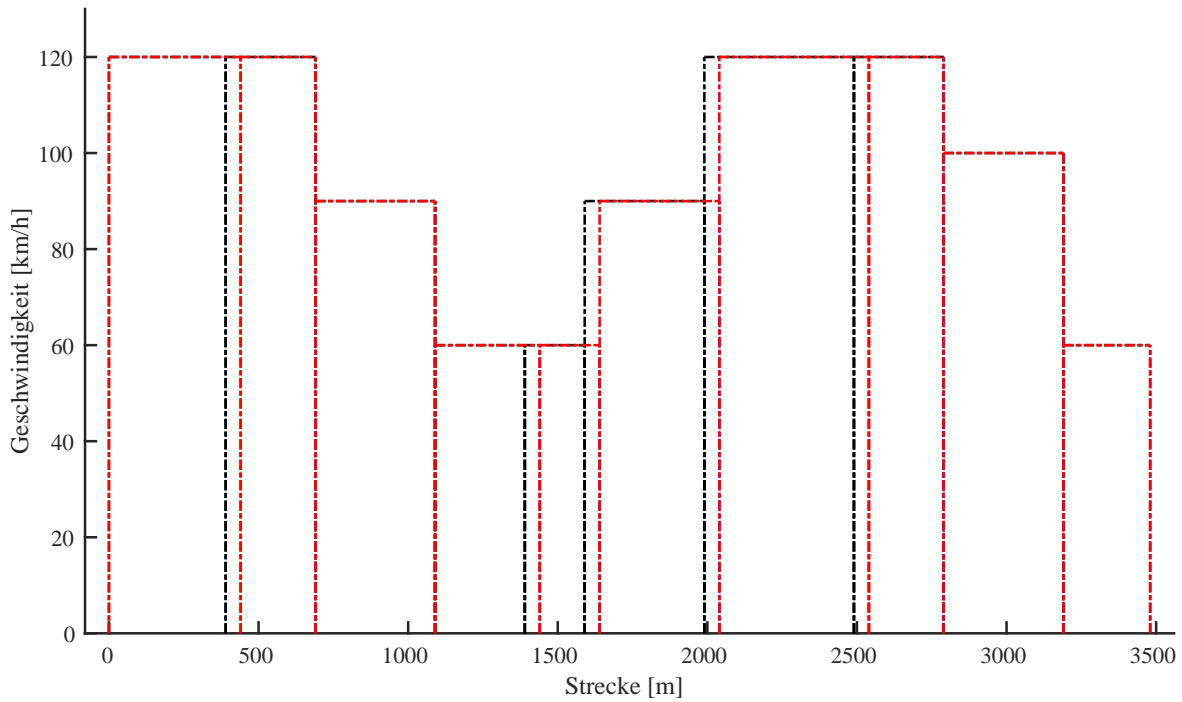


Abbildung 7: Infra-Abschnitte und die zugehörige Höchstgeschwindigkeit unter Berücksichtigung der Fahrzeuglänge

speichert. Diese Umwandlung ist essentiell für die Überprüfung, in welchem Infra-Abschnitt ein Fahrzeug sich aktuell befindet. Die neu berechneten Infra-Abschnitte sind in der Darstellung 7 in rot abgebildet und beschreiben die maximale Geschwindigkeit, die ein Fahrzeug an der jeweiligen Position fahren darf.

## 4.2 Berechnung bei einer Beschleunigung auf die maximal mögliche Geschwindigkeit

Im ersten Schritt der Fahrtverlaufsrechnung wird die Distanz zwischen der aktuellen Position und der Ziel-Position mittels *\$cumulativeSectionLengthStart*, *\$cumulativeSectionLengthEnd*, *\$indexCurrentSection* und *\$indexTargetSection* berechnet. Für die Distanz und die Startgeschwindigkeit wird mit Hilfe der Funktion *getVMaxBetweenTwoPoints()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 5) die maximale Geschwindigkeit ermittelt, auf die das Fahrzeug beschleunigen kann, um bis zum Ziel rechtzeitig bremsen zu können. Dabei wird in 10 *km/h*-Schritten iteriert und der maximale Wert zurückgegeben. Innerhalb der Funktion wird die Funktion *getBrakeDistance()* (*functions\_math.php*) (Code-Beispiel 9) aufgerufen, welche die benötigte Distanz für eine Beschleunigung bzw. Verzögerung berechnet und auf der Gleichung 9 aus Kapitel 7.1 basiert.

```

1 // Ermittelt die maximale Geschwindigkeit zwischen zwei Punkten
2 function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1) {
3
4     global $verzoeigerung;
5     global $globalFloatingPointNumbersRoundingError;
6
7     $v_max = array();
8
9     for ($i = 0; $i <= 120; $i = $i + 10) {
10         if ((getBrakeDistance($v_0, $i, $verzoeigerung) + getBrakeDistance($i, $v_1,
11             ↳ $verzoeigerung)) < ($distance +
12             ↳ $globalFloatingPointNumbersRoundingError)) {
13             array_push($v_max, $i);
14         }
15     }
16
17     if (sizeof($v_max) == 0) {
18         if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
19             echo "Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu
20                 ↳ erreichen.";
21         } else {
22             //emergencyBreak($id);
23         }
24     } else {
25         if ($v_0 == $v_1 && max($v_max) < $v_0) {
26             $v_max = array($v_0);
27         }
28     }
29
30     return max($v_max);
31 }

```

Code-Beispiel 5: *getVMaxBetweenTwoPoints()* (*functions\_fahrtverlauf.php*)



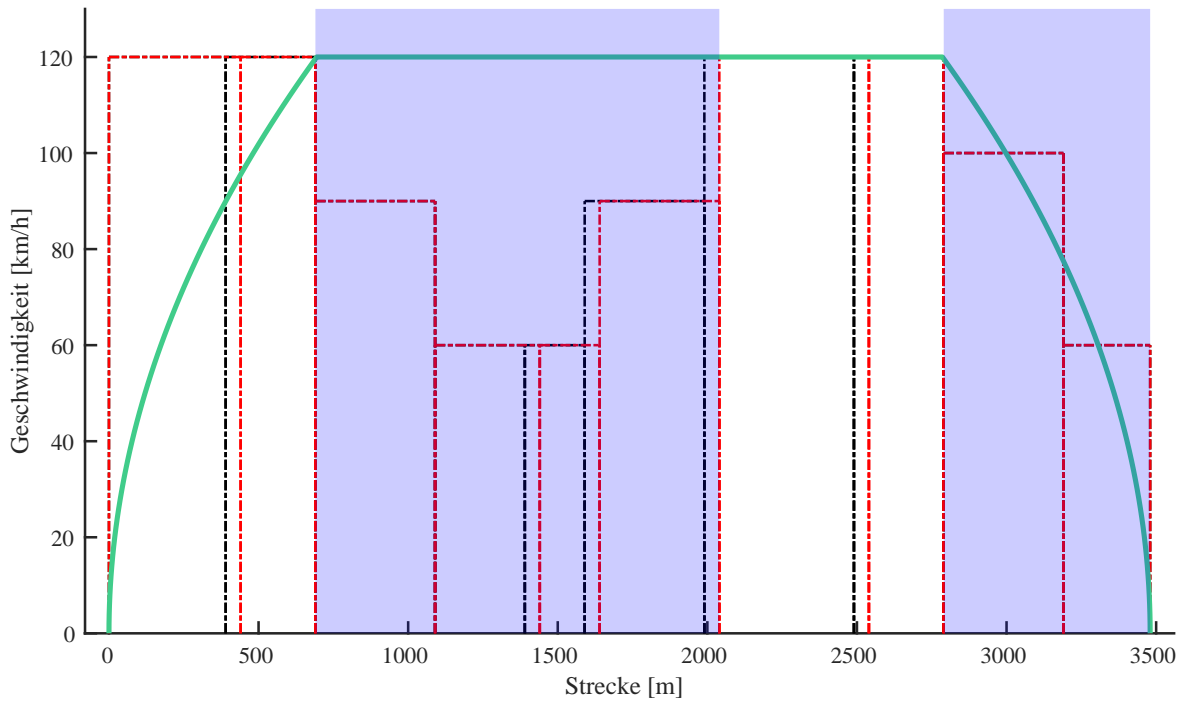


Abbildung 8: Fahrtverlaufs berechnung (1. Iterationsschritt)

Durch die gegebene Startgeschwindigkeit und die größtmögliche Geschwindigkeit wird ein erster Fahrtverlauf berechnet, wobei zwei *keyPoints* erzeugt werden. In der Darstellung 8 ist der erste Iterationsschritt der Fahrtverlaufsermittlung abgebildet, wobei die beiden *keyPoints* in Echtzeitdaten umgewandelt wurden (siehe Kapitel 4.3).

### 4.3 Konvertierung der *keyPoints* in Echtzeitdaten

Die Echtzeitdaten geben in mehreren Arrays den Fahrtverlauf bei Beschleunigungen und Verzögerungen in 2 *km/h*-Schritten und bei Beharrungsfahrten in 1 Meter Abständen an. Die Echtzeitdaten werden mit der Funktion *createTrainChanges()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 6) erzeugt und geben die absolute Position des Fahrzeugs zur aktuellen Position (*\$returnTrainPositionChange*), die Geschwindigkeit (*\$returnTrainSpeedChange*), die Simulationszeit (*\$returnTrainTimeChange*), die relative Position im Infra-Abschnitt (*\$returnTrainRelativePosition*), den Infra-Abschnitt (*\$returnTrainSection*) und ob eine Geschwindigkeitsveränderung vorliegt (*\$returnIsSpeedChange*) an.

Im ersten Schritt wird mit einer *for*-Schleife über alle *keyPoints* iteriert (Code-Beispiel 6; Zeile 24 – 57). Für jeden *keyPoint* wird in einer zweiten *for*-Schleife von der Startgeschwindigkeit (*speed\_0*) bis zur Zielgeschwindigkeit (*speed\_1*) in 2 *km/h*-

Schritten iteriert und für jede Geschwindigkeitsveränderung wird die benötigte Zeit und Strecke errechnet, die Geschwindigkeit gespeichert und dass es sich um eine Geschwindigkeitsanpassung handelt (Code-Beispiel 6; Zeile 32 – 39).

Im Anschluss werden die Echtzeitdaten für die Beharrungsfahrten berechnet. Da Beharrungsfahrten immer zwischen zwei *\$keyPoints* existieren, wird für jeden *\$keyPoint* die Beharrungsfahrt zum nächsten *\$keyPoint* berechnet und der letzte *\$keyPoint* wird ausgelassen (Code-Beispiel 6; Zeile 41 – 56). Das Distanzintervall, in welchem die Echtzeitdaten für Beharrungsfahrten ermittelt werden, kann über die globale Variable *\$globalDistanceUpdateInterval* (*global\_variables.php*) festgelegt werden. Der Standardwert wurde auf 1 m festgelegt.

Für die Ermittlung der zugehörigen Infra-Abschnitte und den relativen Positionen in den Infra-Abschnitten, wird über alle absoluten Positionen (*\$returnTrainPositionChange*) iteriert und in einer zweiten *foreach*-Schleife über alle Anfangspositionen der Infra-Abschnitte (*\$cumulativeSectionLengthStart*). Sobald sich eine absolute Position in einem Infra-Abschnitt befindet, wird der Infra-Abschnitt inklusive der relativen Position ermittelt und die zweite *foreach*-Schleife wird mit dem Befehl *break* abgebrochen (Code-Beispiel 6; Zeile 63 – 96).

```
1 // Echtzeitdatenermittlung eines Fahrtverlaufs auf Grundlage der $keyPoints.
2 // Mit dem Parameter $onlyPositionAndSpeed kann festgelegt werden, ob nur die
3 // Position und Geschwindigkeit berechnet werden soll.
4 function createTrainChanges(bool $onlyPositionAndSpeed) {
5
6     global $keyPoints;
7     global $verzoegerung;
8     global $cumulativeSectionLengthStart;
9     global $cumulativeSectionLengthEnd;
10    global $next_sections;
11    global $indexCurrentSection;
12    global $indexTargetSection;
13    global $currentPosition;
14    global $globalFloatingPointNumbersRoundingError;
15    global $globalDistanceUpdateInterval;
16
17    $returnTrainSpeedChange = array();
18    $returnTrainTimeChange = array();
19    $returnTrainPositionChange = array();
20    $returnTrainRelativePosition = array();
21    $returnTrainSection = array();
```

```

22 $returnIsSpeedChange = array();
23
24 // Ermittelt für alle bis auf den letzten $keyPoint die Echtzeitdaten der
25 // Zeit, Geschwindigkeit und Position
26 for ($i = 0; $i < sizeof($keyPoints); $i++) {
27     array_push($returnTrainTimeChange, $keyPoints[$i]["time_0"]);
28     array_push($returnTrainSpeedChange, $keyPoints[$i]["speed_0"]);
29     array_push($returnTrainPositionChange, $keyPoints[$i]["position_0"]);
30     array_push($returnIsSpeedChange, true);
31
32     $itDir = ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) ? 2 : -2;
33
34     for ($j = ($keyPoints[$i]["speed_0"] + $itDir); $j <= $keyPoints[$i]["
        ↳ speed_1"]; $j = $j + $itDir) {
35         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
        ↳ getBrakeDistance(($j - $itDir), $j, $verzoeigerung)));
36         array_push($returnTrainSpeedChange, $j);
37         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
        ↳ getBrakeTime(($j - $itDir), $j, $verzoeigerung)));
38         array_push($returnIsSpeedChange, true);
39     }
40
41 // Überprüft, ob nach dem $keyPoint eine Beharrungsfahrt stattfindet
42 if ($i != array_key_last($keyPoints)) {
43     // Ermittelt für die Strecke zwischen zwei $keyPoints die Echtzeitdaten
44     // der Zeit, Geschwindigkeit und Position
45     $startPosition = $keyPoints[$i]["position_1"];
46     $endPosition = $keyPoints[$i + 1]["position_0"];
47     $speedToNextKeyPoint = $keyPoints[$i]["speed_1"];
48     $timeForOneTimeInterval = distanceWithSpeedToTime($speedToNextKeyPoint,
        ↳ $globalDistanceUpdateInterval);
49
50     for ($position = $startPosition + $globalDistanceUpdateInterval;
        ↳ $position < $endPosition; $position = $position +
        ↳ $globalDistanceUpdateInterval) {
51         array_push($returnTrainPositionChange, $position);
52         array_push($returnTrainSpeedChange, $speedToNextKeyPoint);
53         array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
        ↳ $timeForOneTimeInterval);

```

```

54     array_push($returnIsSpeedChange, false);
55 }
56 }
57 }
58 array_push($returnTrainPositionChange, $keyPoints[array_key_last($keyPoints)
    ↳ ][ "position_1" ] - getBrakeDistance($keyPoints[array_key_last(
    ↳ $keyPoints)][ "speed_0" ], $keyPoints[array_key_last($keyPoints)][ "
    ↳ speed_1" ], $verzoegerung));
59 array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)][ "
    ↳ speed_0" ]);
60 array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)][ "
    ↳ time_0" ]);
61 array_push($returnIsSpeedChange, true);
62
63 if ($onlyPositionAndSpeed) {
64     return array($returnTrainPositionChange, $returnTrainSpeedChange);
65 } else {
66     // Ermittelt die relativen Positionen innerhalb der Infra-Abschnitte
67     // zu den absoluten Positionen
68     foreach ($returnTrainPositionChange as $absolutPositionKey =>
    ↳ $absolutPositionValue) {
69         foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
    ↳ $sectionStartValue) {
70             if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
    ↳ < $cumulativeSectionLengthEnd[$sectionStartKey]) {
71                 if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
    ↳ $indexTargetSection) {
72                     $returnTrainRelativePosition[$absolutPositionKey] =
    ↳ $absolutPositionValue + $currentPosition;
73                     $returnTrainSection[$absolutPositionKey] = $next_sections[
    ↳ $sectionStartKey];
74                 } else if ($sectionStartKey == $indexCurrentSection) {
75                     $returnTrainRelativePosition[$absolutPositionKey] =
    ↳ $absolutPositionValue + $currentPosition;
76                     $returnTrainSection[$absolutPositionKey] = $next_sections[
    ↳ $sectionStartKey];
77                 } else if ($sectionStartKey == $indexTargetSection) {
78                     $returnTrainRelativePosition[$absolutPositionKey] =
    ↳ $absolutPositionValue - $sectionStartValue;

```

```

79     $returnTrainSection[$absolutPositionKey] = $next_sections[
        ↳ $sectionStartKey];
80 } else {
81     $returnTrainRelativePosition[$absolutPositionKey] =
        ↳ $absolutPositionValue - $sectionStartValue;
82     $returnTrainSection[$absolutPositionKey] = $next_sections[
        ↳ $sectionStartKey];
83 }
84 break;
85 } else if ($absolutPositionKey == array_key_last(
        ↳ $returnTrainPositionChange) && abs($absolutPositionValue -
        ↳ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
        ↳ $globalFloatingPointNumbersRoundingError) {
86     $returnTrainRelativePosition[$absolutPositionKey] =
        ↳ $cumulativeSectionLengthEnd[$sectionStartKey] -
        ↳ $sectionStartValue;
87     $returnTrainSection[$absolutPositionKey] = $next_sections[
        ↳ $sectionStartKey];
88     break;
89 } else {
90     debugMessage("Eine absolute Position konnte kein Infra-Abschnitt und
        ↳ keine relative Position in einem Infra-Abschnitt zugeordnet
        ↳ werden.");
91 }
92 }
93 }
94
95 return array($returnTrainPositionChange, $returnTrainSpeedChange,
        ↳ $returnTrainTimeChange, $returnTrainRelativePosition,
        ↳ $returnTrainSection, $returnIsSpeedChange);
96 }
97 }

```

Code-Beispiel 6: *createTrainChanges()* (*functions\_fahrtverlauf.php*)

#### 4.4 Überprüfung des Fahrtverlaufs nach Geschwindigkeitsüberschreitungen

Für die Überprüfung, ob es bei einem Fahrtverlauf zu einer Überschreitung der zulässigen Höchstgeschwindigkeit kommt, wird nach jeder Berechnung die Funktion *checkIfTrainIsToFastInCertainSections()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 7) aufgerufen. In dieser Funktion wird über alle absoluten Positionen (*\$trainPositionChange*) iteriert, überprüft in welchem Infra-Abschnitt sich diese Position befindet und überprüft, ob die zugehörige Geschwindigkeit aus dem *\$trainSpeedChange*-Array die zulässige Höchstgeschwindigkeit überschreitet. Sobald in einem Infra-Abschnitt eine Geschwindigkeitsüberschreitung vorliegt, wird der zugehörige Index des Infra-Abschnitts in dem *\$failedSections*-Array gespeichert. Diese Infra-Abschnitte sind in der Darstellung 8 Lila hinterlegt.

Als Rückgabewert der Funktion wird ein Array zurückgegeben, welches abspeichert, ob und in welchen Infra-Abschnitten es zu einer Geschwindigkeitsüberschreitung gekommen ist (*failed* und *failed\_sections*).

#### 4.5 Neuberechnung unter Berücksichtigung der Geschwindigkeitsüberschreitung

In dem Fall, dass es zu einer Geschwindigkeitsüberschreitung gekommen ist, wird der Fahrtverlauf neu berechnet. Als Grundlage dafür dienen die *failed\_sections* aus der *checkIfTrainIsToFastInCertainSections()* Funktion (*functions\_fahrtverlauf.php*) (Code-Beispiel 7). Die Funktion *recalculateKeyPoints()* (*functions\_fahrtverlauf.php*) vergleicht dabei immer zwei benachbarte *\$keyPoints* und berechnet in dem Fall einer Geschwindigkeitsüberschreitung mit der Funktion *checkBetweenTwoKeyPoints()* (*functions\_fahrtverlauf.php*) diese neu. In dem Fall, dass zwischen zwei benachbarten *\$keyPoints* die zulässige Höchstgeschwindigkeit überschritten wird, wird die absolute Start- und End-Position dieser Geschwindigkeitsüberschreitung gespeichert.

In dem folgenden Schritt wird wie in dem Abschnitt 4.2 zwischen den Start-Werten des ersten *\$keyPoints* und der ersten Geschwindigkeitsüberschreitung die maximale Geschwindigkeit berechnet und zwei neue *\$keyPoints* erzeugt. Das gleiche passiert zwischen der Position der letzten Geschwindigkeitsüberschreitung und den End-Werten des zweiten *\$keyPoints*. Dadurch wird sichergestellt, dass immer eine gerade Anzahl an *\$keyPoints* existiert und somit in jedem Iterationsschritt zwei benachbarte *\$keyPoints* verglichen werden können. Nachdem alle *\$keyPoint*-Paare überprüft wurden, werden mit Hilfe der *createTrainChanges()*-Funktion (*functions\_fahrtverlauf.php*) die Arrays

```

1 // Überprüft, ob das Fahrzeug in Infra-Abschnitten die zulässige
2 // Höchstgeschwindigkeit überschreitet
3 function checkIfTrainIsToFastInCertainSections() {
4
5     global $trainPositionChange;
6     global $trainSpeedChange;
7     global $cumulativeSectionLengthStartMod;
8     global $next_v_max_mod;
9     global $indexTargetSectionMod;
10
11     $failedSections = array();
12
13     foreach ($trainPositionChange as $trainPositionChangeKey =>
14         ↪ $trainPositionChangeValue) {
15         foreach ($cumulativeSectionLengthStartMod as
16             ↪ $cumulativeSectionLengthStartKey =>
17             ↪ $cumulativeSectionLengthStartValue) {
18             if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
19                 if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
20                     ↪ $cumulativeSectionLengthStartKey - 1]) {
21                     array_push($failedSections, ($cumulativeSectionLengthStartKey - 1));
22                 }
23             }
24             break;
25         } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
26             if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
27                 if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
28                     ↪ $cumulativeSectionLengthStartKey]) {
29                     array_push($failedSections, $cumulativeSectionLengthStartKey);
30                 }
31             }
32             break;
33         }
34     }
35 }
36
37 if (sizeof($failedSections) == 0) {
38     return array("failed" => false);
39 } else {
40     return array("failed" => true, "failed_sections" => array_unique(
41         ↪ $failedSections));
42 }
43 }

```

Code-Beispiel 7: *checkIfTrainIsToFastInCertainSections()* (*functions\_fahrtverlauf.php*)

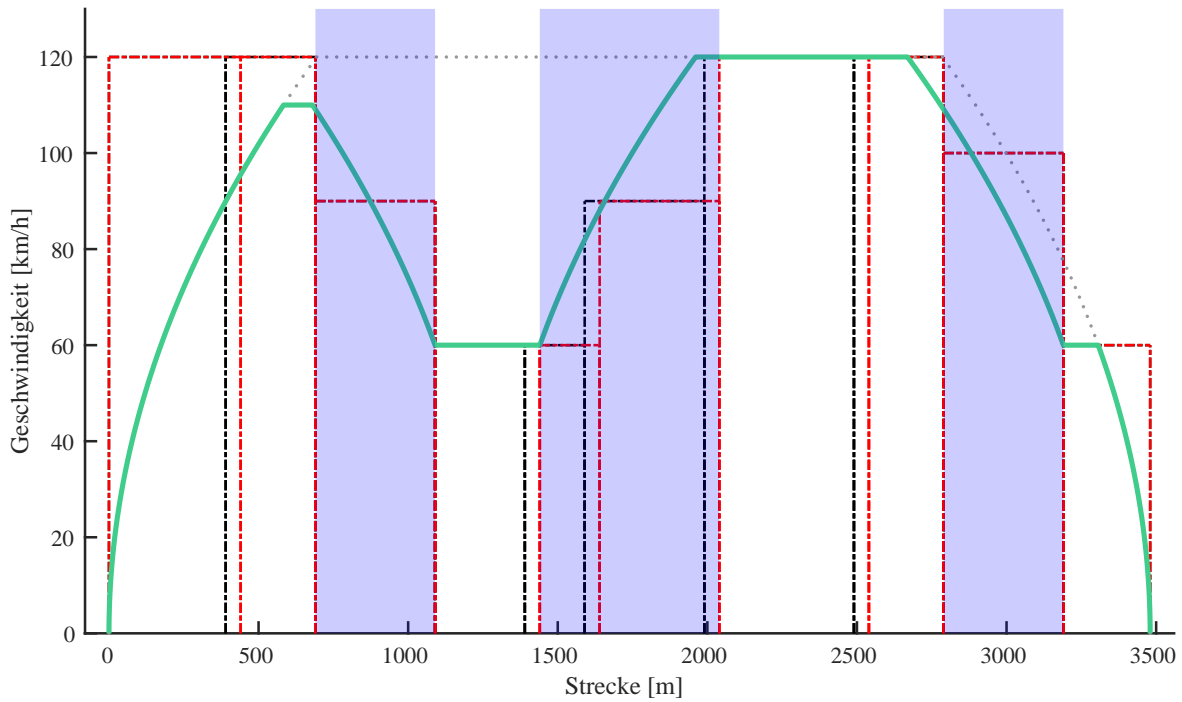


Abbildung 9: Fahrtverlaufs berechnung (2. Iterationsschritt)

*\$strainPositionChange* und *\$strainSpeedChange* erzeugt. Der neu berechnete Fahrtverlauf wird erneut der Funktion *checkIfTrainIsToFastInCertainSections()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 7) übergeben. Dieser Prozess wird solange durchlaufen, bis es zu keiner Geschwindigkeitsüberschreitung mehr kommt. In den folgenden Abbildungen (Darstellung 9, 10 und 11) werden die Ergebnisse der einzelnen Iterationsschritte visuell abgebildet, wobei die grau gepunkteten Linien die Ergebnisse der vorherigen Iterationsschritte darstellen.

#### 4.6 Einhaltung der Mindestzeit auf einer Beharrungsfahrt

Für eine möglichst realitätsnahe Simulation kann über die Variable *\$globalTimeOnOneSpeed* in der Datei *global\_variables.php* eine Mindestzeit festgelegt werden, die ein Fahrzeug auf einer Geschwindigkeit mindestens einhalten muss (Beharrungsfahrt). Ebenfalls kann über die Variablen *\$useMinTimeOnSpeed* und *\$errorMinTimeOnSpeed* festgelegt werden, ob die Funktion aktiviert sein soll und ob es in dem Fall, dass diese Zeit nicht eingehalten werden kann, zu einer Fehlermeldung kommen soll. Im Falle einer Fehlermeldung würde das Fahrzeug nicht losfahren bzw. eine Gefahrenbremsung einleiten, falls das Fahrzeug aktuell eine Geschwindigkeit  $v > 0 \text{ km/h}$  hat.

Wenn auf einem Abschnitt die Mindestzeit nicht eingehalten werden kann, kann eine Beschleunigung später eingeleitet werden, eine Verzögerung vorzeitiger eingeleitet



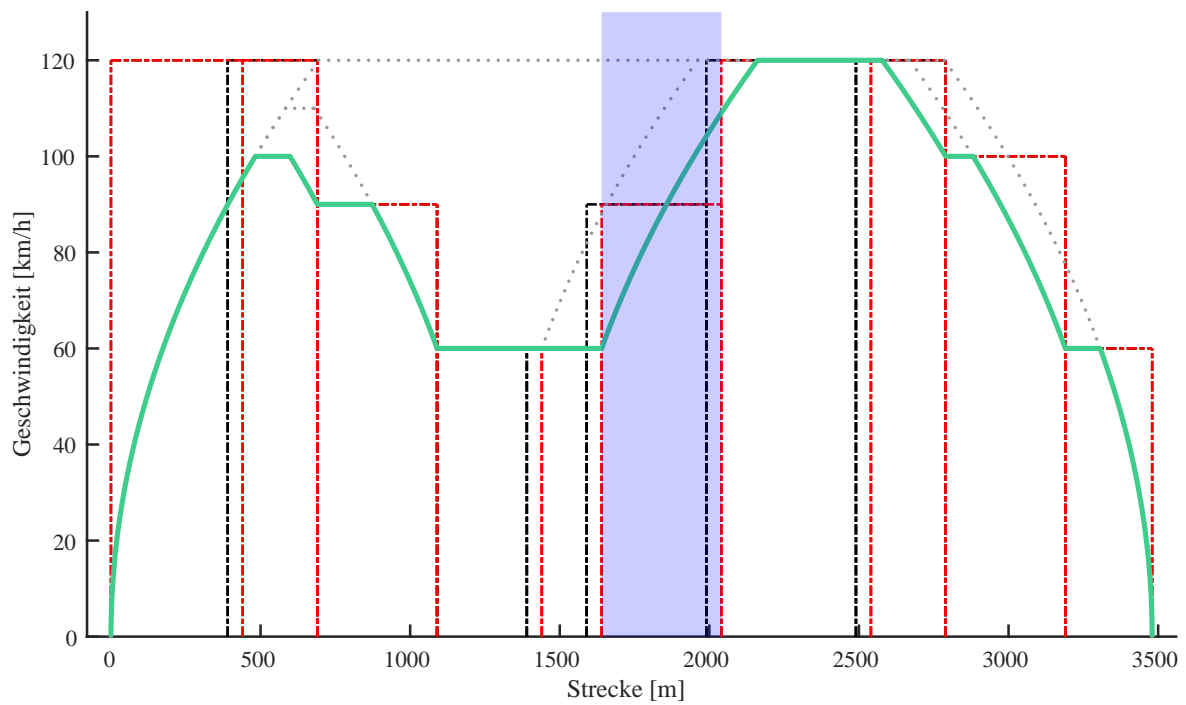


Abbildung 10: Fahrtverlaufsberechnung (3. Iterationsschritt)

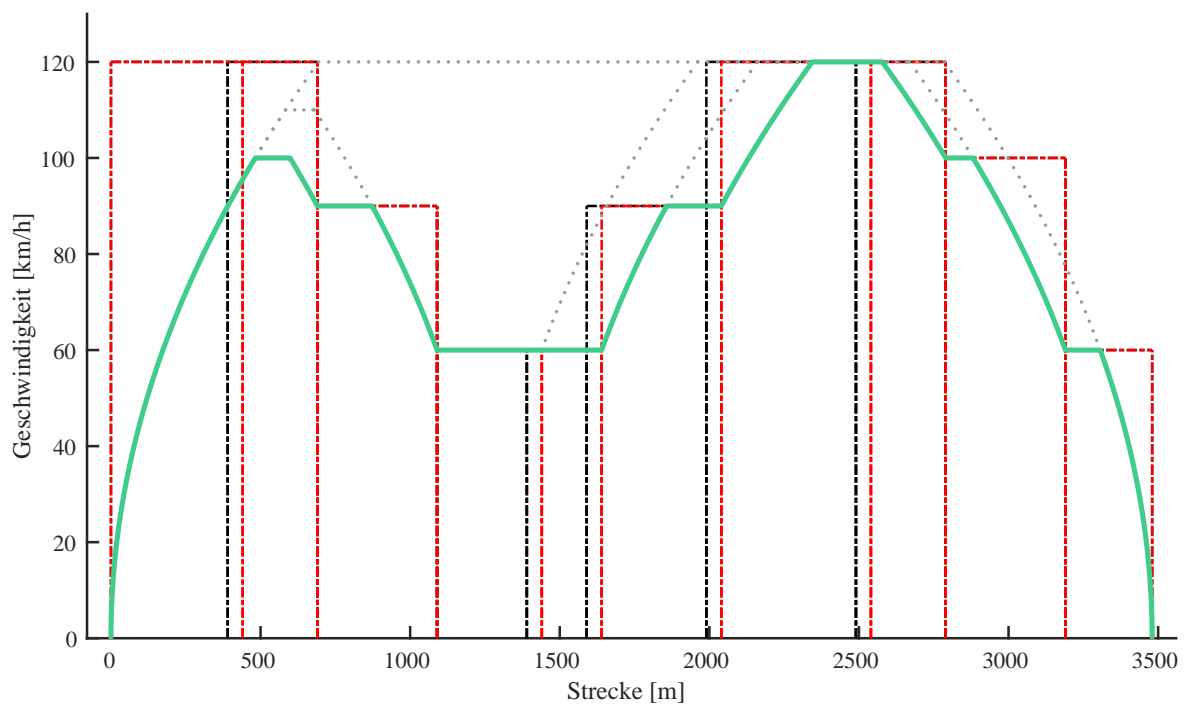


Abbildung 11: Fahrtverlaufsberechnung (4. Iterationsschritt)

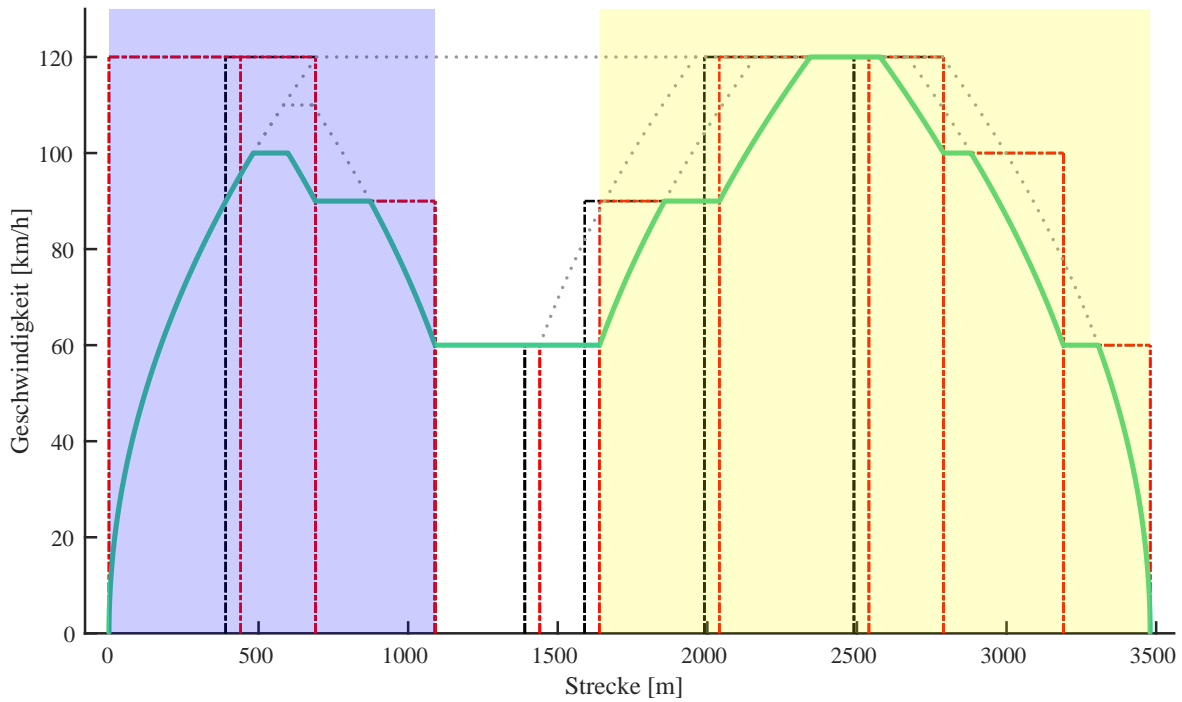


Abbildung 12: Einteilung des Fahrtverlaufs in *subsections*

werden oder auf eine kleinere Geschwindigkeit beschleunigt werden. Dadurch, dass sich eine Verschiebung einer Beschleunigung bzw. Verzögerung auf die nächsten Abschnitte auswirken kann, wird der Fahrtverlauf in *subsections* unterteilt. Eine *subsection* beschreibt dabei den Bereich des Fahrtverlaufs, in dem das Fahrzeug zum ersten Mal beschleunigt und zum letzten Mal abbremst. In der Darstellung 12 wurde der exemplarische Fahrtverlauf somit in zwei *subsections* unterteilt, welche Lila bzw. Gelb hinterlegt sind. Durch diese Einteilung kann verhindert werden, dass es zu Konflikten kommt. Falls die Beschleunigungen bzw. Verzögerungen soweit nach hinten bzw. nach vorne verschoben werden müssen, kann die maximale Geschwindigkeit auf dieser *subsection* reduziert werden und die zur Verfügung stehende Strecke vergrößert werden. Wie in Darstellung 12 zu erkennen wird hierbei im ersten Schritt der Abschnitt zwischen zwei *subsections* ausgelassen. Nach der Ermittlung der *subsections* wird überprüft, ob auf den Abschnitten zwischen den *subsections* die Mindestzeit eingehalten wird. Wenn das nicht der Fall ist, wird der Abschnitt automatisch der in Fahrtrichtung hinteren *subsection* zugeordnet. Dadurch wird sichergestellt, dass das Fahrzeug, wenn es an einer Stelle des Fahrtverlaufs die Geschwindigkeit reduziert, dies möglichst spät tut.

Nachdem die *subsections* mittels der Funktion *createSubsections()* (*func\_tions\_fahrtverlauf.php*) erstellt wurden und mit der Funktion *array\_reverse()* in umgekehrter Reihenfolge in dem Array *subsection\_list* gesammelt wurden, wurde für jede *sub-*

Index	Funktion
<i>max_index</i>	Index des <i>\$keyPoints</i> mit der Beschleunigung auf die maximale Geschwindigkeit in der <i>\$subsection</i>
<i>indexes</i> (Array)	Indexe aller beinhalteten <i>\$keyPoints</i>
<i>is_prev_section</i> (Boolescher Wert)	Berücksichtigung des Abschnitts vor der <i>\$subsection</i>
<i>is_next_section</i> (Boolescher Wert)	Berücksichtigung des Abschnitts nach der <i>\$subsection</i>
<i>failed</i> (Boolescher Wert)	Unterschreitung der Mindestzeit auf der <i>\$subsection</i>

Tabelle 10: Aufbau des *\$subsection*-Arrays

*section* ein Array erzeugt, welches die Variablen aus Tabelle 10 beinhaltet und jede *\$subsection* eindeutig beschreibt.

Bei den *\$subsections*, bei denen die Mindestzeit für die Beharrungsfahrten nicht eingehalten wird (*failed == true*), wird überprüft, ob eine Verschiebung der Beschleunigungen bzw. Verzögerungen möglich ist. Bei der Verschiebung einer Beschleunigung bzw. Verzögerung wird die Differenz zwischen der Mindestzeit einer Beharrungsfahrt (*\$globalTimeOnOneSpeed*) und der Zeit der vorherigen bzw. folgenden Beharrungsfahrt berechnet und die Beschleunigung bzw. Verzögerung um diese Differenz verschoben.

Sollte bei einer Verschiebung die *position\_1* eines *\$keyPoints* hinter *position\_0* des folgenden *\$keyPoints* liegen (bei einer Beschleunigung), wird der zweite *\$keyPoint* gelöscht und die Zielgeschwindigkeit des zweiten *\$keyPoints* wird der Zielgeschwindigkeit des ersten *\$keyPoints* zugewiesen. Gleiches geschieht bei der Verzögerung in umgekehrter Reihenfolge.

Nach der Verschiebung wird überprüft, ob auf allen konstanten Geschwindigkeit die Mindestzeit eingehalten wird. Wenn das der Fall ist, wird die nächste *\$subsection* überprüft. In dem Fall, dass durch die Verschiebung die Mindestzeit nicht eingehalten werden kann, wird die maximale Geschwindigkeit auf dieser *\$subsection* um 10 km/h reduziert, die *\$subsections* neu berechnet und erneut über alle *\$subsection* iteriert. Die Neuberechnung ist notwendig, da durch die Reduzierung der Geschwindigkeit die *\$subsections* anders aufgeteilt sein können.

Wenn alle *\$subsections* die Mindestzeit einhalten, wird der Algorithmus beendet. In der Darstellung 13 ist der Fahrtverlauf unter Einhaltung der Mindestzeit auf einer Geschwindigkeit abgebildet. Für den Fall, dass das Fahrzeug auf einer Geschwindigkeit die Mindestzeit nicht einhält und als nächstes beschleunigen würde, kann die Beschleunigung später eingeleitet werden.

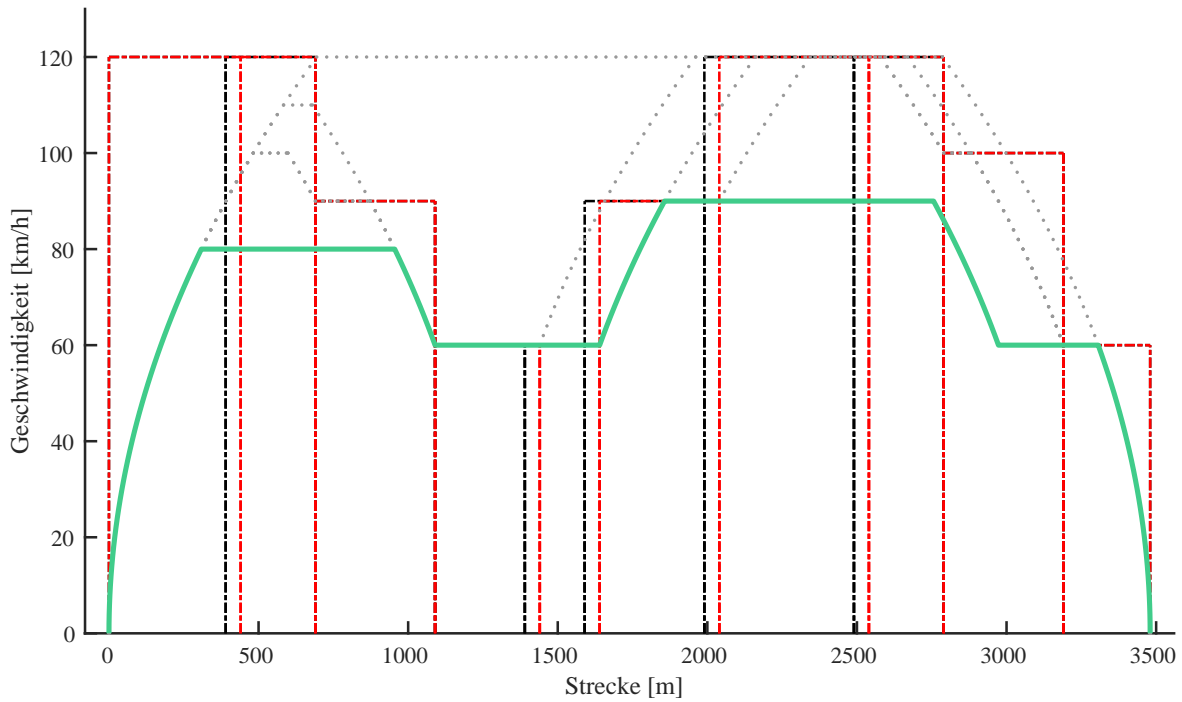


Abbildung 13: Fahrtverlauf unter Einhaltung der Mindestzeit

#### 4.7 Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs

Der berechnete Fahrtverlauf aus den Kapiteln 4.2, 4.4, 4.5 und 4.6 ermittelt die frühestmögliche Ankunftszeit am Ziel. In dem Fall, dass der Zug dadurch mit einer Verspätung am Ziel ankommt, wird der Fahrtverlauf an das Fahrzeug übergeben. Falls das Fahrzeug mit dem Fahrtverlauf zu früh am Ziel ankommen würde, wird überprüft, ob es möglich ist die Geschwindigkeit zu reduzieren, sodass der Zug energieeffizienter fahren kann und ohne Verspätung am Ziel ankommt.

Im ersten Schritt wird mittels der Funktion *checkIfTheSpeedCanBeDecreased()* (*functions\_fahrtverlauf.php*) überprüft, ob die Geschwindigkeit reduziert werden kann. Dabei werden alle *\$keyPoints* ermittelt, bei denen das Fahrzeug beschleunigt und die beim darauffolgenden *\$keyPoint* abbremsen. Für jeden dieser *\$keyPoints* werden die möglichen Geschwindigkeiten ermittelt, welche das Fahrzeug zwischen den beiden *\$keyPoints* fahren könnte. Für die Berechnung dieser Geschwindigkeiten wird als niedrigste Geschwindigkeit die *speed\_0* des ersten *\$keyPoints* bzw. *speed\_1* des zweiten *\$keyPoints* – je nachdem, welche niedriger ist – genommen und in 10 *km/h*-Schritten bis *speed\_1* des ersten *\$keyPoints* abgespeichert. Daraus ergibt sich für jeden *\$keyPoint* eine *range* an möglichen Geschwindigkeiten. Als Rückgabewert der Funktion wird ein Array zurückgegeben, welches die Einträge *possible* und *range* enthält und als *\$return-*

*SpeedDecrease* abgespeichert. Der Eintrag *possible* gibt an, ob das Fahrzeug auf dem gesamten Fahrtverlauf die Geschwindigkeit reduzieren könnte und wird als Boolescher Wert (*true/false*) abgespeichert und in dem Array *range* werden alle Indexe der möglichen *\$keyPoints* inklusive der ermittelten Geschwindigkeiten abgespeichert.

In dem in Abbildung 13 dargestellten Fahrtverlauf wären für den *\$keyPoint* mit dem Index 0 (die Indexe der *\$keyPoints* entsprechen dem Zahlenbereich der  $\mathbb{N}_0$ ) die Geschwindigkeiten 60, 70 und 80 km/h ermittelt worden und für den *\$keyPoint* mit dem Index 2 die Geschwindigkeiten 60, 70, 80 und 90 km/h.

Wenn eine Reduzierung der Geschwindigkeit möglich ist, wird in einer *while*-Schleife versucht die Geschwindigkeit zu reduzieren, bis das Fahrzeug bei der nächsten Reduzierung mit einer Verspätung am Ziel ankommen würde oder eine weitere Reduzierung nicht möglich ist. Innerhalb der *while*-Schleife ermittelt die Funktion *findMaxSpeed()* (*functions\_fahrtverlauf.php*) aus dem *\$returnSpeedDecrease*-Array den *\$keyPoint* mit der höchsten Geschwindigkeit. Für den Fall, dass mehrere *\$keyPoints* die selbe Höchstgeschwindigkeit haben, wird der letzte dieser *\$keyPoints* ermittelt. Im Anschluss wird mit einer *for*-Schleife in 10 km/h-Schritten in absteigender Reihenfolge über die möglichen Geschwindigkeiten iteriert und überprüft, ob durch die Anpassung die Ankunftszeit eingehalten werden kann. Sobald die Ankunftszeit nicht eingehalten werden kann, werden die *\$keyPoints* aus dem vorherigen Iterationsschritt gespeichert und die *while*-Schleife wird abgebrochen. Sollte die *for*-Schleife durchlaufen, ohne dass es zu einer Überschreitung der maximal verfügbaren Zeit kommt, wird die Funktion *checkIfTheSpeedCanBeDecreased()* (*functions\_fahrtverlauf.php*) erneut aufgerufen. Das Ergebnis dieser Berechnung ist in der Abbildung 14 abgebildet.

#### 4.8 Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs

Die in Kapitel 4.6 errechnete Ankunftszeit, beschreibt die spätmöglichste Ankunftszeit am Ziel, ohne dass das Fahrzeug mit einer Verspätung am Ziel ankommt, wenn bei einer Beschleunigung auf eine geringere Zielgeschwindigkeit beschleunigt wird. Dadurch wird das Fahrzeug im Normalfall nicht exakt pünktlich das Ziel erreichen. Über die Variable *\$useSpeedFineTuning* kann festgelegt werden, ob das Fahrzeug eine exakte Ankunftszeit versuchen soll zu erreichen. Wenn diese Funktion aktiviert ist und der Eintrag *possible* aus dem Array *\$returnSpeedDecrease* *true* ist, wird für den letzten *\$keyPoint* aus dem *\$returnSpeedDecrease*-Array überprüft, ob die Verzögerung des nächsten *\$keyPoints* vorzeitiger eingeleitet werden kann. Sollte die Zielgeschwindigkeit der Verzögerung 0 km/h sein, wird die Verzögerung unterteilt in eine Verzögerung auf

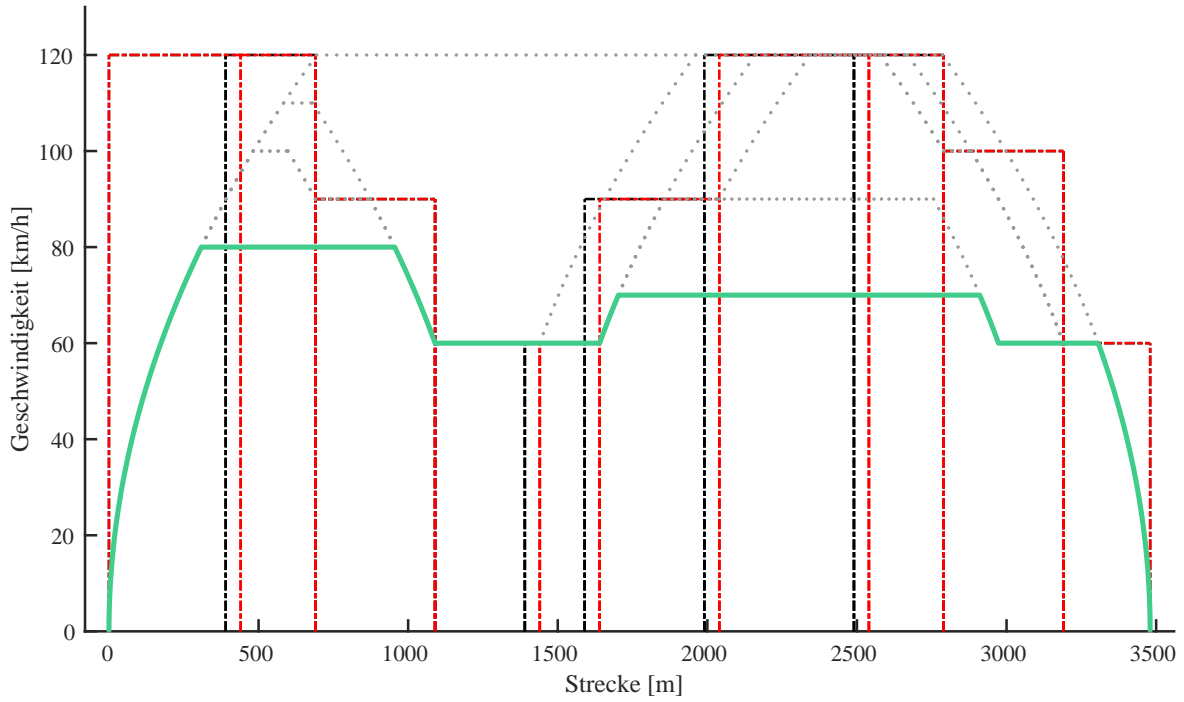


Abbildung 14: Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit

10  $km/h$  und eine von 10  $km/h$  auf 0  $km/h$ . Die Position der vorzeitig eingeleiteten Verzögerung wird mittels der Funktion *speedFineTuning()* (*functions\_fahrtverlauf.php*) berechnet, welche als Parameter den Betrag der Differenz zwischen aktueller Soll- und Ist-Ankunftszeit und den Index des vorherigen *\$keyPoints* übergeben bekommt und auf der Gleichung 19 aus Kapitel 7 basiert. In Abbildung 15 werden die Geschwindigkeiten ( $v_1$ ,  $v_2$ ), Strecken ( $s_1$ ,  $s_2$ ) und Zeiten ( $t_1$ ,  $t_2$ ) vor und nach der Verzögerung – welche vorzeitig eingeleitet werden soll, um eine pünktliche Ankunft am Ziel zu ermöglichen – dargestellt und in Tabelle 11 sind die exakten Werte des exemplarischen Fahrtverlaufs aufgelistet. In diesem konkreten Beispiel würde das Fahrzeug 3,31  $s$  ( $t_\Delta$ ) zu früh an der Betriebsstelle ankommen, wodurch das Fahrzeug für die Zurücklegung der Strecken  $s_1$  und  $s_2$  insgesamt 85,42  $s$  ( $t_{ges} = t_1 + t_2 + t_\Delta$ ) zur Verfügung hat.

Durch das Einsetzen der Werte in die Gleichung 19 aus dem Kapitel 7.2 ergibt sich für  $t_3$  ( $t_3$ ,  $t_4$ ,  $s_3$  und  $s_4$  bezeichnen die Strecken und Zeiten nach der Anpassung) ein Wert von 42,2  $s$ . Dementsprechend muss die Verzögerung 19,85  $s$  ( $t_1 - t_3$ ) früher eingeleitet werden.

$$t_3 = \frac{1541 \text{ m} - 16,67 \text{ m/s} \cdot 85,42 \text{ s}}{19,44 \text{ m/s} - 16,67 \text{ m/s}}$$

$$t_3 = 42,26 \text{ s}$$

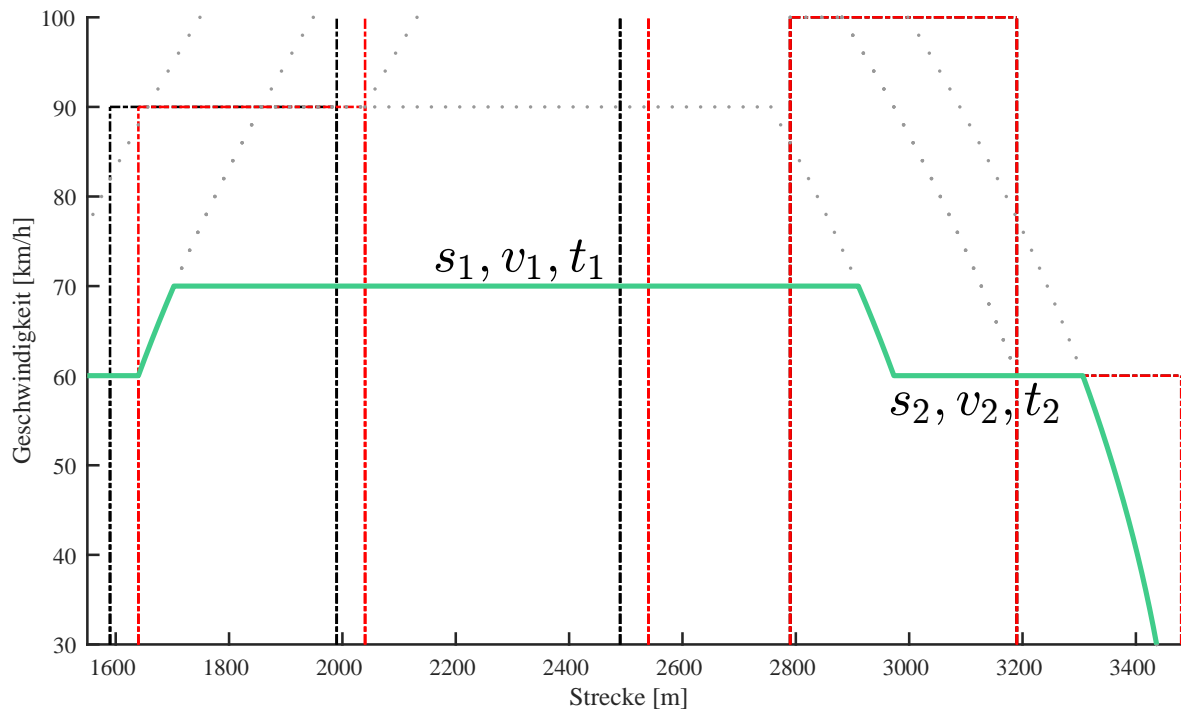


Abbildung 15: Fahrtverlauf vor der Anpassung der exakten Ankunftszeit

$v_1$	70 km/h ( $\approx 19,44$ m/s)
$v_2$	60 km/h ( $\approx 16,67$ m/s)
$s_1$	1207,67 m
$s_2$	333,33 m
$s_{ges}$	1541 m
$t_1$	62,11 s
$t_2$	20 s
$t_{\Delta}$	3,31 s
$t_{ges}$	85,42 s

Tabelle 11: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung vor der Anpassung

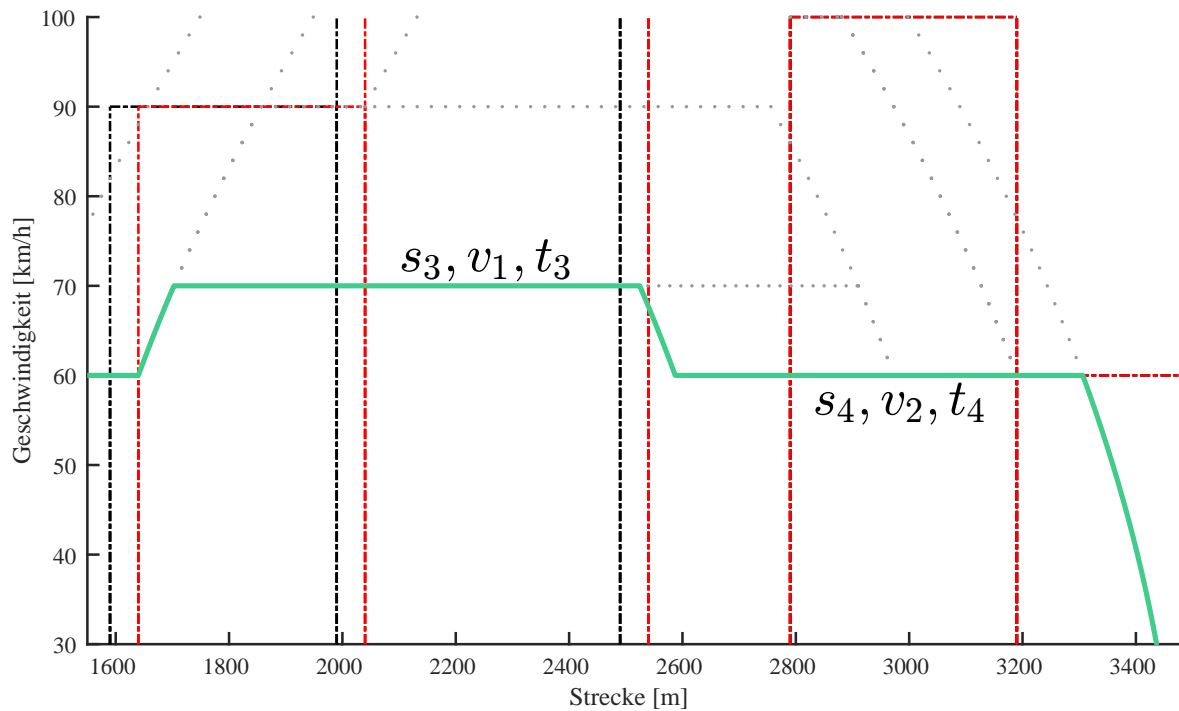


Abbildung 16: Fahrtverlauf nach der Anpassung der exakten Ankunftszeit

$s_3$	821,91 m
$s_4$	719,1 m
$t_3$	42,26 s
$t_4$	43,16 s

Tabelle 12: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung nach der Anpassung

Die vorzeitige Einleitung der Verzögerung sorgt dafür, dass das Fahrzeug die nächste Betriebsstelle genau pünktlich erreicht und ist in Abbildung 16 dargestellt, wobei durch die gepunktete Linie der Fahrtverlauf vor der Anpassung zu sehen ist. Die neu berechneten Werte sind in Tabelle 12 aufgelistet. Der finale Fahrtverlauf ist in Abbildung 17 dargestellt und kann so dem Fahrzeug übergeben werden.

#### 4.9 Einleitung einer Gefahrenbremsung

Eine Gefahrenbremsung wird eingeleitet, sobald ein Fahrzeug bei einer sofortigen Verzögerung ein auf Halt stehendes Signal überfahren würde, in einem Infra-Abschnitt die zulässige Höchstgeschwindigkeit überschreiten würde oder an dem nächsten planmäßigen Halt nicht rechtzeitig zum stehen kommen würde. Bei einer Gefahrenbremsung



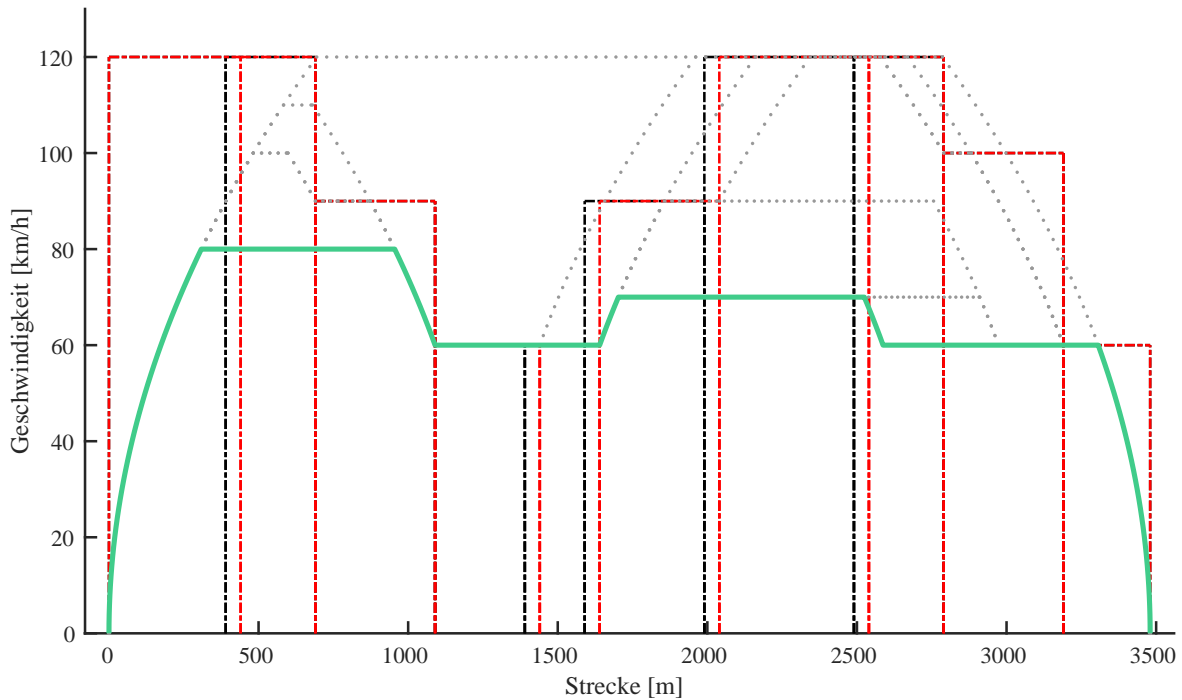


Abbildung 17: Ergebnis der Fahrtverlaufsermittlung

wird mit einer Notbremsverzögerung von  $2 \text{ m/s}^2$  abgebremst. Dieser Wert kann in der Datei *global\_variables.php* über die Variable *\$globalNotverzoeigerung* angepasst werden. Für eine möglichst realitätsnahe Simulation einer Gefahrenbremsung, bei der das Risiko für Fahrzeugschäden möglichst gering ist, wurde sich dafür entschieden, dass die Fahrzeuge – wenn sie an der Gefahrenstelle eine Geschwindigkeit haben, für die gilt:  $v \geq 10 \text{ km/h}$  – nach der Geschwindigkeit von  $10 \text{ km/h}$  direkt die Geschwindigkeit von  $0 \text{ km/h}$  übermittelt bekommen. Dadurch wird bei der Berechnung einer Gefahrenbremsung zwischen drei Fällen unterschieden:

1. Fahrzeug hält mit der Notbremsverzögerung vor der Gefahrenstelle
2. Fahrzeug hat bei der Gefahrenstelle eine Geschwindigkeit von  $v < 10 \text{ km/h}$
3. Fahrzeug hat bei der Gefahrenstelle eine Geschwindigkeit von  $v \geq 10 \text{ km/h}$

Für die Überprüfung, ob das Fahrzeug mit der Notbremsverzögerung vor der Gefahrenstelle zum Stehen kommt, wird mittels der Funktion *getBrakeDistance()* (*functions.php*) der Bremsweg ( $s_{\text{Bremsweg}}$ ) berechnet und mit der Distanz zur Gefahrenstelle ( $s_{\text{Gefahrenstelle}}$ ) verglichen. Sollte für den Bremsweg gelten:  $s_{\text{Bremsweg}} \leq s_{\text{Gefahrenstelle}}$ , wird das Fahrzeug die Gefahrenbremsung einleiten und in  $2 \text{ km/h}$ -Schritten auf  $0 \text{ km/h}$  abbremesen. In dem Fall, dass der Bremsweg länger als die Strecke bis zur Gefahrenstel-

le ist, wird überprüft, welche Geschwindigkeit das Fahrzeug an der Gefahrenstelle hat. Für diese Berechnung wird die Gleichung 11 aus dem Kapitel 7.1 verwendet.

Sollte das Fahrzeug an der Gefahrenstelle eine Geschwindigkeit von  $v \geq 10 \text{ km/h}$  haben, bremst das Fahrzeug in  $2 \text{ km/h}$ -Schritten auf  $10 \text{ km/h}$  ab und bekommt nach der Übermittlung der  $10 \text{ km/h}$  direkt  $0 \text{ km/h}$  übergeben. In dem Fall, dass das Fahrzeug an der Gefahrenstelle langsamer als  $10 \text{ km/h}$  ist, bremst das Fahrzeug wie im 1. Fall in  $2 \text{ km/h}$ -Schritten auf  $0 \text{ km/h}$  ab. Bei einer Gefahrenbremsung bekommt das jeweilige Fahrzeug eine Fehlermeldung übermittelt und wird nicht weiterfahren, da durch die Gefahrenbremsung keine genaue Positionsbestimmung vorgenommen werden kann. Damit das Fahrzeug wieder seinen Fahrbetrieb aufnehmen kann, muss das Fahrzeug händisch von der Anlage genommen werden, gewartet werden, bis die Fahrzeugsteuerung das Entfernen registriert hat und wieder neu positioniert werden.

<i>\$keyPoint</i> -Index	0	1	2
<i>\$speed_0</i>	0 <i>km/h</i>	30 <i>km/h</i>	10 <i>km/h</i>
<i>\$speed_1</i>	30 <i>km/h</i>	10 <i>km/h</i>	0 <i>km/h</i>
<i>\$position_0</i>	0 <i>m</i>	528.83 <i>m</i>	667.18 <i>m</i>
<i>\$position_1</i>	43.40 <i>m</i>	567.41 <i>m</i>	672 <i>m</i>
<i>\$time_0</i> (Unix-Timestamp)	1631088005	1631088073,67	1631088116,53
<i>\$time_1</i> (Unix-Timestamp)	1631088015,41	1631088080,61	1631088120
<i>\$time_0</i> (hh:mm:ss)	10:00:05	10:01:14	10:01:57
<i>\$time_1</i> (hh:mm:ss)	10:00:15	10:01:21	10:02:00

Tabelle 13: *\$keyPoints* am Beispiel von der Fahrt von XAB nach XZO

## 5 Beispielrechnung eines Fahrtverlaufs im EBUf

Die in Kapitel 4 beschriebene Berechnung des Fahrtverlaufs wird in diesem Kapitel an einer Beispielfahrt von Ausblick (XAB) nach Zoo (XZO) exemplarisch gezeigt. Dafür wurde dem Zug ein Fahrplan zugewiesen, nach dem der Zug nach Simulationszeit um 10:00:05 in Ausblick losfahren soll und um 10:02:00 in dem Bahnhof Zoo ankommen soll. Zu Beginn steht der Zug im Infra-Abschnitt 1189 (XAB), hat die Fahrtrichtung 1 und die Fahrstraße ist so eingestellt, dass das Fahrzeug bis zum Ausfahrtsignal im Bahnhof Zoo fahren kann und dort im Infra-Abschnitt 1178 zum Stehen kommen kann. Somit beträgt die Strecke bis zum nächsten Halt 672 *m* und das Fahrzeug hat 115 *s* zur Verfügung. Die Bremsverzögerung des Fahrzeugs beträgt 0,8 *m/s*<sup>2</sup>.

Für die Fahrt wurde eine Mindestzeit von 20 *s* für Beharrungsfahrten (*\$globalTimeOnOneSpeed* = 20) festgelegt, den Optionen *\$useSpeedFineTuning*, *\$useMinTimeOnSpeed* und *\$slowDownIfTooEarly* wurde der Wert *true* zugewiesen und der Option *\$errorMinTimeOnSpeed* der Wert *false*.

In der Tabelle 13 sind die berechneten *\$keyPoints* aufgelistet, welche durch die Berechnung des Fahrtverlaufs ermittelt wurden, und in der Darstellung 18 ist der Fahrtverlauf visuell dargestellt. Bei der Berechnung des Fahrtverlaufs wurde laut der Fahrzeugsteuerung die Ankunftszeit exakt eingehalten. Die Zeit-Werte der *\$keyPoints* geben bei der Berechnung die Simulationszeit im Unix-Timestamp-Format an und sind deswegen ebenfalls im Format *hh:mm:ss* angegeben. Durch die *\$keyPoints* und die Darstellung des Fahrtverlaufs (Abbildung 18) lässt sich der Fahrtverlauf in 5 Abschnitte einteilen. Die Start- und Zielgeschwindigkeit, die Strecke und die Zeit der einzelnen Abschnitte sind in der Tabelle 14 aufgelistet und werden mittels der Formeln aus Ka-

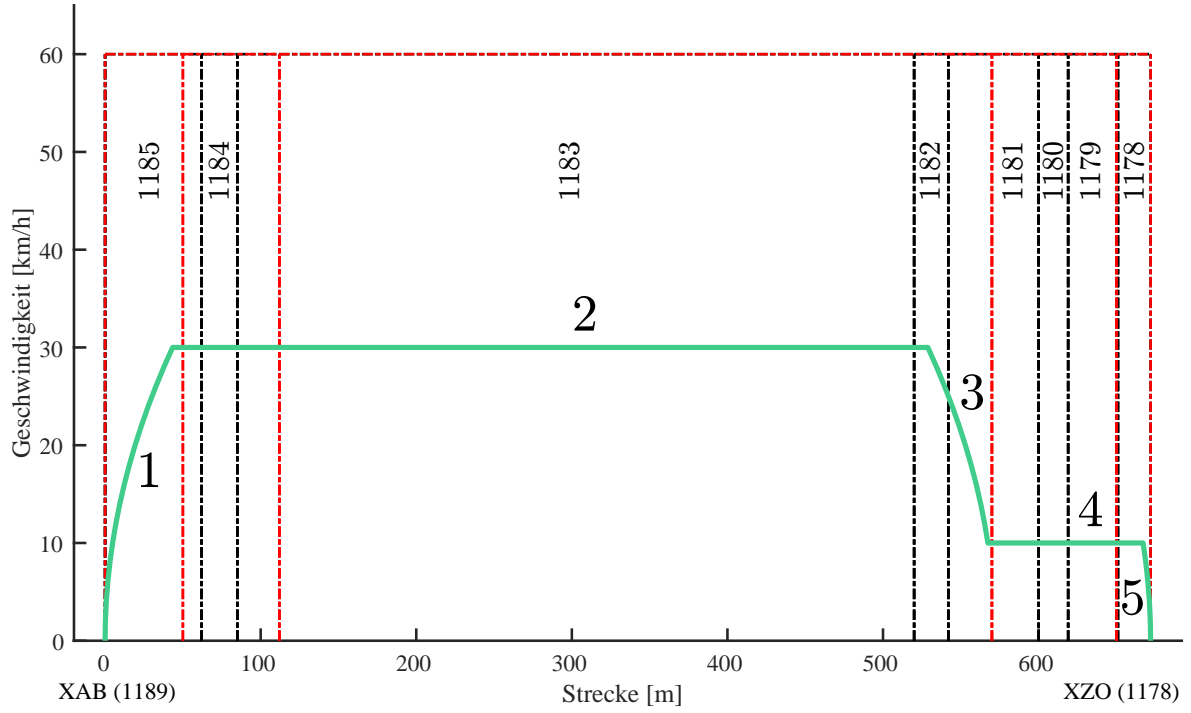


Abbildung 18: Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO

pitel 7 überprüft. Bei der Überprüfung werden die Start- und Zielgeschwindigkeiten als Grundlage genommen und untersucht, ob unter Einhaltung der gegebenen Zeit die selben Werte rauskommen. Damit der berechnet Fahrtverlauf den Vorgaben entspricht, muss gelten:

$$t_{ges} = t_1 + t_2 + t_3 + t_4 + t_5 = 115 \text{ s}$$

$$s_{ges} = s_1 + s_2 + s_3 + s_4 + s_5 = 672 \text{ m}$$

Für die Berechnung werden die Strecken und Zeiten in gleichförmige und gleichmäßig beschleunigte Bewegungen unterteilt:

$$t_{ges} = t_{\text{gleichförmigeBewegungen}} + t_{\text{gleichmäßigBeschleunigteBewegungen}}$$

$$s_{ges} = s_{\text{gleichförmigeBewegungen}} + s_{\text{gleichmäßigBeschleunigteBewegungen}}$$

$$t_{\text{gleichförmigeBewegungen}} = t_2 + t_4$$

$$s_{\text{gleichförmigeBewegungen}} = s_2 + s_4$$

$$t_{\text{gleichmäßigBeschleunigteBewegungen}} = t_1 + t_3 + t_5$$

$$s_{\text{gleichmäßigBeschleunigteBewegungen}} = s_1 + s_3 + s_5$$

Abschnitt	Beschleunigung/ Verzögerung	$v_0$	$v_1$	Strecke	Zeit
1	ja (Beschleunigung)	0 km/h	30 km/h	43,40 m	10,42 s
2	nein	30 km/h	30 km/h	485,43 m	58,25 s
3	ja (Verzögerung)	30 km/h	10 km/h	38,58 m	6,94 s
4	nein	10 km/h	10 km/h	99,76 m	35,92 s
5	ja (Verzögerung)	10 km/h	0 km/h	4,82 m	3,47 s
$\Sigma$	—	—	—	672 m <sup>1</sup>	115 s

<sup>1</sup> Die Werte in der Strecken-Spalte sind auf zwei Nachkommastellen gerundet und würden durch das Aufsummieren der Strecken von Abschnitt 1 – 5 eine Gesamtstrecke von 671,99 m ergeben. Die angegebenen 672 m entsprechen der Summe der Abschnitte 1 – 5, ohne dass die einzelnen Strecken gerundet werden.

Tabelle 14: Fahrtverlauf am Beispiel von der Fahrt von XAB nach XZO

Für die gleichmäßig beschleunigten Bewegungen gilt nach den Gleichungen 9 und 10:

$$t_1 = \left| \frac{\frac{30 \text{ km/h}}{3,6} - \frac{0 \text{ km/h}}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{125}{12} \text{ s} \approx 10,42 \text{ s}$$

$$t_3 = \left| \frac{\frac{10 \text{ km/h}}{3,6} - \frac{30 \text{ km/h}}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{125}{18} \text{ s} \approx 6,94 \text{ s}$$

$$t_5 = \left| \frac{\frac{0 \text{ km/h}}{3,6} - \frac{10 \text{ km/h}}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{125}{36} \text{ s} \approx 3,47 \text{ s}$$

$$s_1 = \frac{1}{2} \cdot \left| \frac{\frac{30 \text{ km/h}^2}{3,6} - \frac{0 \text{ km/h}^2}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{3125}{72} \text{ m} \approx 43,40 \text{ m}$$

$$s_3 = \frac{1}{2} \cdot \left| \frac{\frac{10 \text{ km/h}^2}{3,6} - \frac{30 \text{ km/h}^2}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{3125}{81} \text{ m} \approx 38,58 \text{ m}$$

$$s_5 = \frac{1}{2} \cdot \left| \frac{\frac{0 \text{ km/h}^2}{3,6} - \frac{10 \text{ km/h}^2}{3,6}}{0,8 \text{ m/s}^2} \right| = \frac{3125}{648} \text{ m} \approx 4,82 \text{ m}$$

Dadurch ergibt sich für die Beschleunigungen und Verzögerungen insgesamt eine Strecke von:

$$t_{\text{gleichmässigBeschleunigteBewegungen}} = \frac{125}{6} \text{ s}$$

$$s_{\text{gleichmässigBeschleunigteBewegungen}} = \frac{3125}{36} m$$

Und für die gleichförmigen Bewegungen gilt dementsprechend:

$$t_{\text{gleichförmigeBewegungen}} = \frac{565}{6} s$$

$$s_{\text{gleichförmigeBewegungen}} = \frac{21067}{36} m$$

Für die Berechnung der Strecke und Zeit von der Beharrungsfahrt auf 30 km/h gilt nach der Gleichung 19:

$$t_2 = \frac{s_{\text{gleichförmigeBewegungen}} - \frac{10 \text{ km/h}}{3.6} \cdot t_{\text{gleichförmigeBewegungen}}}{\frac{30 \text{ km/h}}{3.6} - \frac{10 \text{ km/h}}{3.6}}$$

$$t_2 = \frac{\frac{21067}{36} m - \frac{10 \text{ km/h}}{3.6} \cdot \frac{565}{6} s}{\frac{30 \text{ km/h}}{3.6} - \frac{10 \text{ km/h}}{3.6}}$$

$$t_2 = \frac{34951}{600} s \approx 58,25 s$$

Daraus folgt nach der Gleichung 15 für die Abschnitte 2 und 4:

$$t_4 = \frac{7183}{200} s \approx 35,91 s$$

$$s_2 = \frac{34951}{600} s \cdot \frac{30 \text{ km/h}}{3,6}$$

$$s_2 = \frac{34951}{72} m \approx 485,43 m$$

$$s_4 = \frac{7183}{200} s \cdot \frac{10 \text{ km/h}}{3,6}$$

$$s_4 = \frac{7183}{72} m \approx 99,76 m$$

In Summe ergibt das:

$$t_{ges} = \frac{125}{12} s + \frac{34951}{600} s + \frac{125}{18} s + \frac{7183}{200} s + \frac{125}{36} s = 115 s$$

$$s_{ges} = \frac{3125}{72} m + \frac{34951}{72} m + \frac{3125}{81} m + \frac{7183}{72} m + \frac{3125}{648} m = 672 m$$

Wie an den errechneten Werten zu erkennen ist, wurde die Mindestzeit von 20 s auf einer konstanten Geschwindigkeit ( $t_2$  und  $t_4$ ) eingehalten und die Werte stimmen mit den Werten aus der Tabelle 14 überein.

## 6 Visualisierung der Fahrtverläufe

Für die Visualisierung der Fahrtverläufe wurde ein MATLAB-Skript geschrieben, welches aus den Arrays *\$cumulativeSectionLengthStart*, *\$cumulativeSectionLengthEnd*, *\$cumulativeSectionLengthStartMod*, *\$cumulativeSectionLengthEndMod*, *\$trainSpeedChange* und *\$trainPositionChange* eines Fahrtverlaufs den kompletten Fahrtverlauf darstellt. Dieses Skript wurde auch verwendet, um die einzelnen Schritte bei der Kalkulation des Fahrtverlaufs in dieser Arbeit darzustellen (wie z. B. in Abbildung 17).

Damit die Daten aus der Berechnung des Fahrtverlaufs von MATLAB eingelesen werden können, wurde die Funktion *safeTrainChangeToJSONFile()* (*functions\_fahrtverlauf.php*) (Code-Beispiel 8) geschrieben, welche die Daten aus den Arrays als JSON-Datei speichert. Für eine bessere Verdeutlichung des Prozesses bei der Ermittlung des Fahrtverlaufs, werden neben dem Ergebnis auch alle vorherigen Iterationsschritte abgebildet.

Das MATLAB-Skript befindet sich im Anhang dieser Arbeit (siehe A.8) und auf weitere Details bezüglich der Funktionsweise wird im Rahmen dieser Arbeit nicht weiter eingegangen.

```
1 // Wandelt die Daten der Infra-Abschnitte und der Iterationsschritte der
2 // Fahrtverlaufsrechnung in JSON-Dateien um, damit die Fahrtverläufe
3 // visuell dargestellt werden können.
4 function safeTrainChangeToJSONFile(int $indexCurrentSection, int
    ↳ $indexTargetSection, int $indexCurrentSectionMod, int
    ↳ $indexTargetSectionMod, array $speedOverPositionAllIterations) {
5
6     global $trainPositionChange;
7     global $trainSpeedChange;
8     global $next_v_max;
9     global $cumulativeSectionLengthEnd;
10    global $next_v_max_mod;
11    global $cumulativeSectionLengthEndMod;
12    $speedOverPosition = array_map('toArr', $trainPositionChange,
        ↳ $trainSpeedChange);
13    $speedOverPosition = json_encode($speedOverPosition);
14    $fp = fopen('../json/speedOverPosition.json', 'w');
15    fwrite($fp, $speedOverPosition);
16    fclose($fp);
17    $v_maxFromUsedSections = array();
18
```

```

19 for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
20     array_push($v_maxFromUsedSections, $next_v_max[$i]);
21 }
22
23 $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
    ↪ $v_maxFromUsedSections);
24 $VMaxOverPositionsJSoN = json_encode($VMaxOverCumulativeSections);
25 $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
26 fwrite($fp, $VMaxOverPositionsJSoN);
27 fclose($fp);
28
29 $v_maxFromUsedSections = array();
30
31 for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
32     array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
33 }
34
35 $VMaxOverCumulativeSectionsMod = array_map('toArr',
    ↪ $cumulativeSectionLengthEndMod, $v_maxFromUsedSections);
36 $VMaxOverPositionsJSoN = json_encode($VMaxOverCumulativeSectionsMod);
37 $fp = fopen('../json/VMaxOverCumulativeSectionsMod.json', 'w');
38 fwrite($fp, $VMaxOverPositionsJSoN);
39 fclose($fp);
40
41 $jsonReturn = array();
42
43 for ($i = 0; $i < sizeof($speedOverPositionAllIterations); $i++) {
44     $iteration = array_map('toArr', $speedOverPositionAllIterations[$i][0],
    ↪ $speedOverPositionAllIterations[$i][1]);
45     array_push($jsonReturn, $iteration);
46 }
47
48 $speedOverPosition = json_encode($jsonReturn);
49 $fp = fopen('../json/speedOverPosition_prevIterations.json', 'w');
50 fwrite($fp, $speedOverPosition);
51 fclose($fp);
52 }

```

Code-Beispiel 8: *safeTrainChangeToJSONFile()* (*functions\_fahrtverlauf.php*)



## 7 Formeln

Für die im folgenden Kapitel verwendeten Einheiten gilt:

$$\begin{aligned}a &= \text{Bremsverzögerung } [m/s^2] \\v &= \text{Geschwindigkeit } [m/s] \\s &= \text{Strecke } [m] \\t &= \text{Zeit } [s]\end{aligned}$$

### 7.1 Formeln für gleichmäßig beschleunigte Bewegungen

Bei einer gleichmäßig beschleunigten Bewegung gilt:<sup>9</sup>

$$a(t) = a \quad (1)$$

Für die Bestimmung der Geschwindigkeit in Abhängigkeit der Zeit, muss die Beschleunigung  $a(t)$  nach der Zeit  $t$  integriert werden.<sup>10</sup>

$$v(t) = \int a(t) dt \quad (2)$$

Daraus ergibt sich folgende Gleichung für die Geschwindigkeit in Abhängigkeit der Zeit. Die bei der Integration entstehende Integrationskonstante  $v_0$  gibt dabei die Startgeschwindigkeit an.

$$v(t) = a \cdot t + v_0 \quad (3)$$

Für die Bestimmung der benötigten Zeit muss die Geschwindigkeit erneut integriert werden.<sup>11</sup> Die dabei entstehende Integrationskonstante  $s_0$  gibt die bereits zurückgelegte Strecke an.

$$s(t) = \int v(t) dt \quad (4)$$

$$s(t) = \frac{1}{2} \cdot a \cdot t^2 + v_0 \cdot t + s_0 \quad (5)$$

Bei der Verwendung dieser Gleichung werden die Integrationskonstanten  $v_0$  und  $s_0$  gleich 0 gesetzt, damit die Gleichungen allgemein gültig sind. Für die Berechnung des Beschleunigungs- und Abbremsverhalten der Fahrzeuge ist es notwendig zu wissen,

---

<sup>9</sup> Richard & Sander (2011, S. 22)

<sup>10</sup> Richard & Sander (2011, S. 20)

<sup>11</sup> Richard & Sander (2011, S. 20)

welche Strecke ein Fahrzeug zurücklegen muss, um von einer Startgeschwindigkeit  $v_0$  auf eine Zielgeschwindigkeit  $v_1$  zu beschleunigen bzw. abzubremesen. Dafür wird die Gleichung für die Geschwindigkeit  $v(t)$  nach  $t(v)$  umgestellt und in die Gleichung  $s(t)$  eingesetzt. Daraus ergibt sich folgende Gleichung für die Strecke in Abhängigkeit von der Geschwindigkeit:

$$t(v) = \frac{v}{a} \quad (6)$$

$$s(v) = \frac{1}{2} \cdot \frac{v^2}{a} \quad (7)$$

Durch die Festlegung von  $v_0 = 0$  wird so die benötigte Strecke ermittelt, welche ein Fahrzeug bei einer gegebenen Bremsverzögerung  $a$  benötigt, um von  $0 \text{ m/s}$  auf eine gegebenen Zielgeschwindigkeit  $v_1$  zu beschleunigen. Bei der Berechnung des Beschleunigungs- und Abbremsverhaltens wird es aber auch zu Situationen kommen, bei denen ein Fahrzeug eine Startgeschwindigkeit hat, für die gilt  $v_0 \neq 0$ . Um eine allgemein gültige Gleichung aufzustellen, wird für die Ermittlung der benötigten Strecke bei einer gegebenen Start- und Zielgeschwindigkeit die Strecke berechnet, die das Fahrzeug benötigt, um von  $0 \text{ m/s}$  auf  $v_1$  und von  $0 \text{ m/s}$  auf  $v_0$  zu beschleunigen. Für die gesuchte Strecke gilt dann:

$$s(v_0, v_1) = |s(v_1) - s(v_0)| \quad (8)$$

$$s(v_0, v_1) = \frac{1}{2} \cdot \left| \frac{v_1^2 - v_0^2}{a} \right| \quad (9)$$

In der Fahrzeugsteuerung übernimmt diese Berechnung die Funktion *getBrakeDistance()* (Code-Beispiel 9).

```

1 // Ermittlung der Strecke für eine Beschleunigung bzw. Verzögerung
2 function getBrakeDistance (float $v_0, float $v_1, float $verzoeigerung) {
3     return abs(0.5 * ((pow($v_0/3.6, 2) - pow($v_1/3.6, 2))/($verzoeigerung)));
4 }

```

Code-Beispiel 9: *getBrakeDistance()* (*functions\_math.php*)

Neben der Berechnung der Strecke ist auch die benötigte Zeit essenziell. Dafür wird mittels  $t(v)$  die Zeit berechnet, die das Fahrzeug benötigt, um von  $0 \text{ km/h}$  auf  $v_0$  bzw.  $v_1$  zu beschleunigen und aus der Differenz wird die benötigte Zeit berechnet.

$$t(v_0, v_1) = \left| \frac{v_1 - v_0}{a} \right| \quad (10)$$

In der Fahrzeugsteuerung übernimmt diese Berechnung die Funktion *getBrakeTime()* (*functions\_math.php*) (Code-Beispiel 10).

```

1 // Ermittelt die Distanz für Brems- und Verzögerungsvorgänge
2 function getBrakeTime (float $v_0, float $v_1, float $verzögerung) {
3     return abs((( $v_1/3.6)/$verzögerung) - (( $v_0/3.6)/$verzögerung));
4 }

```

Code-Beispiel 10: *getBrakeTime()* (*functions\_math.php*)

Für die Berechnung einer Gefahrenbremsung ist es notwendig zu wissen, welche Geschwindigkeit das Fahrzeug an der Position der Gefahrenstelle hat. Dafür wird die Gleichung (9) nach  $v_2$  umgestellt. Umgesetzt wird diese Gleichung mit der Funktion *getTargetBrakeSpeedWithDistanceAndStartSpeed()* (*functions\_math.php*) (Code-Beispiel 11).

$$v_2(v_1, s) = \sqrt{-2 \cdot s \cdot a} + v_1 \quad (11)$$

```

1 // Ermittelt die Geschwindigkeit, die ein Fahrzeug in einem Bremsvorgang
2 // nach einer gegebenen Distanz hat.
3 function getTargetBrakeSpeedWithDistanceAndStartSpeed (float $distance, float
4     ↳ $verzögerung, int $speed) {
5     return sqrt((-2 * $verzögerung * $distance) + (pow(($speed / 3.6), 2)))*3.6;
6 }

```

Code-Beispiel 11: *getTargetBrakeSpeedWithDistanceAndStartSpeed()* (*functions\_math.php*)

## 7.2 Formeln für gleichförmige Bewegungen

Bei einer gleichförmigen Bewegung gilt der Grundsatz:<sup>12</sup>

$$v(t) = v \quad (12)$$

Für die Berechnung der Strecke gilt wie bei der gleichmäßig beschleunigten Bewegung:<sup>13</sup>

$$s(t) = \int v(t) dt \quad (13)$$

$$s(t) = v \cdot t + s_0 \quad (14)$$

<sup>12</sup> Richard & Sander (2011, S. 22)

<sup>13</sup> Richard & Sander (2011, S. 20)

Damit die Gleichung allgemeingültig ist, wird die Integrationskonstante  $s_0$  gleich 0 gesetzt.

$$s(t) = v \cdot t \quad (15)$$

```

1 // Ermittelt die Zeit, die ein Fahrzeug bei einer gegebenen Strecke für
2 // eine gegebene Distanz benötigt
3 function distanceWithSpeedToTime (int $v, float $distance) {
4     return (($distance)/($v / 3.6));
5 }

```

Code-Beispiel 12: *distanceWithSpeedToTime()* (*functions\_math.php*)

Für die Einhaltung der exakten Ankunftszeit, muss errechnet werden, wie lange das Fahrzeug bei zwei gegebenen Geschwindigkeiten ( $v_1$  und  $v_2$ ) auf den jeweiligen Geschwindigkeiten fahren muss, um die Gesamtstrecke ( $s_{ges}$ ) und die Gesamtzeit ( $t_{ges}$ ) einzuhalten. Für die Zeiten und Strecken gilt:

$$t_{ges} = t_1 + t_2 \quad (16)$$

$$s_{ges} = s_1 + s_2 \quad (17)$$

Durch das Einsetzen der Gleichung (15) in die Gleichung (17) erhält man folgende Gleichung:

$$s_{ges} = v_1 \cdot t_1 + v_2 \cdot t_2 \quad (18)$$

Durch das Umstellen der Gleichung (16) nach  $t_2$  und dem Einsetzen in Gleichung (18) gilt für  $t_1$ :

$$t_1 = \frac{s_{ges} - v_2 \cdot t_{ges}}{v_1 - v_2} \quad (19)$$

```

1 // Ermittelt die Distanz, um die eine Verzögerung "verschoben" werden müsste,
2 // damit die exakte Ankunftszeit eingehalten werden kann.
3 function calculateDistanceforSpeedFineTuning(int $v_0, int $v_1, float
4     ↪ $distance, float $time) : float {
5     return $distance - (($distance - $time * $v_1 / 3.6)/($v_0 / 3.6 - $v_1 /
6     ↪ 3.6)) * ($v_0 / 3.6);
7 }

```

Code-Beispiel 13: *calculateDistanceforSpeedFineTuning()* (*functions\_math.php*)

## 8 Fazit

### 8.1 Zusammenfassung der Ergebnisse

Der Entwickelte Algorithmus ist in der Lage, für eine gegebene Position und Geschwindigkeit, Infra-Abschnitte inklusive deren Längen und zulässigen Höchstgeschwindigkeiten und einer Zielposition den optimalen Fahrtverlauf zu ermitteln, sodass das Fahrzeug ohne eine Überschreitung der zulässigen Höchstgeschwindigkeit und unter Berücksichtigung der Fahrzeuglänge frühestmöglich die Zielposition erreicht. Unter Berücksichtigung der Ankunftszeit wurden Ansätze entwickelt, die dafür sorgen, dass das Fahrzeug – durch eine Reduzierung der Geschwindigkeit – pünktlich das Ziel erreicht und durch eine geringere Geschwindigkeit energiesparsamer fährt. Die Ansätze für die Einhaltung der Ankunftszeit ermitteln nicht den optimalsten Fahrtverlauf, da in vereinzelt Fällen die Geschwindigkeit reduziert wird, obwohl eine Verschiebung von Brems- bzw. Verzögerungsvorgängen ausreichen würde.

Bei der Entwicklung und Testung der Fahrzeugsteuerung wurden die Fahrten am Rechner auf Grundlage der *MySQL*-Datenbank simuliert. Die dabei ermittelten Fahrtverläufe haben die erforderlichen Bedingungen (Ankunfts-, Abfahrts- und Mindesthaltezeit und zulässige Höchstgeschwindigkeit) eingehalten und die Fahrzeuge haben richtig auf Fahrstraßen-Änderungen reagiert (Einleitung einer Gefahrenbremsung und Berücksichtigung der Signalbegriffe). Bei der Verwendung der Fahrzeugsteuerung im EBUef ist es zu Fehlern gekommen, welche in dem folgenden Kapitel 8.2 erläutert werden.

### 8.2 Komplikationen bei dem Betrieb der Fahrzeugsteuerung im EBUef

#### 8.2.1 Einhaltung der Zielposition

Bei Zugfahrten ist es dazu gekommen, dass die Fahrzeuge den Bremsvorgang zu spät eingeleitet haben und an dem Ausfahrtsignal bzw. einem Halt zeigenden Signal vorbei gefahren sind. Für die Fehlerbehebung wurde überprüft, ob die eingetragenen Längen der Infra-Abschnitte aus der *MySQL*-Datenbank mit den realen Werten des EBUefs übereinstimmen. Diese mögliche Ursache konnte ausgeschlossen werden und weitere Ursachen konnten nicht ermittelt werden.

### 8.2.2 Ermittlung der Fahrstraßen

Bei der Ermittlung der Fahrstraßen hat die Funktion *getNaechsteAbschnitte()*\* (*functions\_ebuef.php*) in manchen Fällen nicht die eingestellte Fahrstraße und die erwarteten Infra-Abschnitte wiedergegeben, wodurch für Fahrzeuge kein Fahrtverlauf berechnet wurde.

### 8.2.3 Kalibrierung der Position

Bei der Kalibrierung der Fahrzeugposition wurden Positionen ermittelt, welche stark von der realen Position abwichen. Mögliche Ursachen könnten eine falsche Positionsermittlung bei der Berechnung des Fahrtverlaufs oder eine nicht rechtzeitige Eintragung der aktuellen Abschnitte in die *MySQL*-Tabelle *fahrzeuge\_abschnitte* sein. Keine der beiden Ursachen konnte eindeutig widerlegt oder belegt werden.

## 8.3 Möglichkeiten für eine Weiterentwicklung der Fahrzeugsteuerung

Durch die kontinuierliche Positionsbestimmung der Fahrzeuge ist es in zukünftigen Weiterentwicklungen der Fahrzeugsteuerung möglich, anstatt der Zugfolge im festen Raumabstand, einen Zugbetrieb im Bremswegabstand (Moving Block) zu realisieren, wodurch die Zugfolgezeiten zweier aufeinander folgender Züge deutlich reduziert werden können.<sup>14</sup>

---

<sup>14</sup> Büker et al. (2020, S. 37)

## Literatur

- Büker, T., Hennig, E. & Schotten, S. (2020). Kapazitätsberechnung im moving block – die tücke im detail. *Eisenbahntechnische Rundschau*, 7+8, 32–37. Zugriff auf [www.via-con.de/wp-content/uploads/32\\_37\\_Bueker\\_Hennig\\_Schotten\\_FA.pdf](http://www.via-con.de/wp-content/uploads/32_37_Bueker_Hennig_Schotten_FA.pdf) (Letzter Zugriff am: 21. September 2021)
- Ebuef: *Eisenbahn-Betriebs- und Experimentierfeld Berlin*. (2021). [www.ebuef.de](http://www.ebuef.de). EBuEf e.V.; c/o Technische Universität Berlin; Fachgebiet Bahnbetrieb und Infrastruktur. (Letzter Zugriff am: 11. September 2021)
- Maschek, U. (2018). *Sicherung des Schienenverkehrs* (4. Aufl.). Springer-Verlag. (ISBN: 978-3-658-22878-1)
- Pachl, J. (2021). *Systemtechnik des Schienenverkehrs* (10. Aufl.). Springer-Verlag. (ISBN: 978-3-658-31165-0)
- RailCom - DCC-Rückmeldeprotokoll* (Norm Nr. RCN-217). (2019, Dezember). (Rail-Community – Verband der Hersteller Digitaler Modellbahnprodukte e.V.)
- Richard, H. & Sander, M. (2011). *Technische mechanik. dynamik: Grundlagen - effektiv und anwendungsnah* (2. Aufl.). Vieweg+Teubner Verlag. (ISBN: 978-3-658-05027-6)
- The IEEE and The Open Group. (2018). *The open group base specifications issue 7 – ieee std 1003.1, 2018 edition*. New York, NY, USA: IEEE. Zugriff auf <https://pubs.opengroup.org/onlinepubs/9699919799> (Letzter Zugriff am: 16. September 2021)

## A Anhang

### A.1 fahrzeugsteuerung.php

```
1 <?php
2
3 // Liest alle benötigten Dateien ein
4 require 'config/multicast.php';
5 require 'vorbelegung.php';
6 require 'functions/functions.php';
7 require 'functions/functions_cache.php';
8 require 'functions/functions_db.php';
9 require 'functions/functions_math.php';
10 require 'functions/functions_ebuef.php';
11 require 'functions/functions_fahrtverlauf.php';
12 require 'global_variables.php';
13
14 // Zeitzone setzen
15 date_default_timezone_set('Europe/Berlin');
16
17 // PHP-Fehlermeldungen
18 error_reporting(1);
19
20 // Globale Variablen
21 global $useRecalibration;
22
23 // Fahrzeugfehlermeldungen definieren
24 $trainErrors = array();
25 $trainErrors[0] = 'Fahrtrichtung des Fahrzeugs musste geändert werden und die
    ↳ Positionsbestimmung war nicht möglich.';
26 $trainErrors[1] = 'In der Datenbank ist für das Fahrzeug keine Zuglänge
    ↳ angegeben.';
27 $trainErrors[2] = 'In der Datenbank ist für das Fahrzeug keine v_max angegeben.
    ↳ ';
28 $trainErrors[3] = 'Das Fahrzeug musste eine Gefahrenbremsung durchführen.';
29
30 // Statische Daten einlesen
31 $cacheInfranachbarn = createCacheInfranachbarn();
32 $cacheInfradaten = createCacheInfradaten();
```



```

33 $cacheSignaldaten = createCacheSignaldaten();
34 $cacheInfraLaenge = createCacheInfraLaenge();
35 $cacheHaltepunkte = createCacheHaltepunkte();
36 $cacheZwischenhaltepunkte = createCacheZwischenhaltepunkte();
37 $cacheInfraToGbt = createCacheInfraToGbt();
38 $cacheGbtToInfra = createCacheGbtToInfra();
39 $cacheFmaToInfra = createCacheFmaToInfra();
40 $cacheInfraToFma = array_flip($cacheFmaToInfra);
41 $cacheFahrplanSession = createCacheFahrplanSession();
42 $cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
43 $cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
44 $cacheIDTDecoder = createCacheDecoderToAdresse();
45 $cacheDecoderToID = array_flip($cacheIDTDecoder);
46 $cacheAdresseToID = array(); // Filled with data in getAllTrains()
47 $cacheIDToAdresse = array(); // Filled with data in getAllTrains()
48
49 // Variablendeklaration
50 $allTrainsOnTheTrack = array();
51 $allTrains = array();
52 $allUsedTrains = array();
53 $allTimes = array();
54 $lastMaxSpeedForInfraAndDir = array();
55
56 // Real- und Simulationszeit ermitteln
57 $simulationStartTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_startzeit, 'simulationszeit', null, array('outputtyp' => 'h:i:s')),
    ↳ 'simulationszeit', null, array('inputtyp' => 'h:i:s'));
58 $simulationEndTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->
    ↳ sim_endzeit, 'simulationszeit', null, array('outputtyp' => 'h:i:s')), '
    ↳ simulationszeit', null, array('inputtyp' => 'h:i:s'));
59 $simulationDuration = $cacheFahrplanSession->sim_endzeit -
    ↳ $cacheFahrplanSession->sim_startzeit;
60 $realStartTime = time();
61 $realEndTime = $realStartTime + $simulationDuration;
62 $timeDifference = $simulationStartTimeToday - $realStartTime;
63
64 // Startmeldung
65 startMessage();
66

```

```

67 // Ermittlung aller Fahrzeuge
68 $allTrains = getAllTrains();
69
70 // Ermittlung der Fahrzeuge im eingleisigen Netz
71 findTrainsOnTheTracks();
72
73 // Ermittlung der Fahrpläne der Fahrzeuge
74 addStopsectionsForTimetable();
75
76 // Überprüfung, ob die Fahrzeuge bereits
77 // an einer Betriebsstelle des Fahrplans stehen
78 checkIfTrainReachedHaltepunkt();
79
80 // Überprüfung, ob die Fahrtrichtung der Fahrzeuge mit dem
81 // Fahrplan übereinstimmt. Falls die Richtung nicht übereinstimmt,
82 // wird die Fahrtrichtung der Fahrzeuge geändert
83 checkIfStartDirectionIsCorrect();
84 consoleAllTrainsPositionAndFahrplan();
85 showErrors();
86
87 // Ermittlung der Fahrstraßen aller Fahrzeuge
88 calculateNextSections();
89
90 // Überprüfung, ob die Fahrstraße für die Fahrzeuge mit Fahrplan
91 // richtig eingestellt ist
92 checkIfFahrstrasseIsCorrect();
93
94 // Ermittlung der Fahrtverläufe aller Fahrzeuge
95 calculateFahrtverlauf();
96
97 // Übermittlung der Echtzeitdaten an die Fahrzeuge
98 // $timeCheckFahrstrasseInterval => Überprüfung von Fahrstraßenänderungen
99 // $timeCheckAllTrainErrorsInterval
100 // => Ausgabe der aktuellen Positionen und Fahrplänen
101 // $timeCheckCalibrationInterval => Neukalibrierung der P0sition
102 $timeCheckFahrstrasseInterval = 3;
103 $timeCheckFahrstrasse = $timeCheckFahrstrasseInterval + microtime(true);
104 $timeCheckAllTrainStatusInterval = 30;
105 $timeCheckAllTrainStatus = $timeCheckAllTrainStatusInterval + microtime(true);

```

```

106 $timeCheckCalibrationInterval = 3;
107 $timeCheckCalibration = $timeCheckCalibrationInterval + microtime(true);
108
109 // Zeitintervall, in dem überprüft wird, ob neue Echtzeitdaten vorliegen
110 $sleepTime = 0.03;
111 while (true) {
112
113     // Iteration über alle Fahrzeuge
114     foreach ($allTimes as $timeIndex => $timeValue) {
115         if (sizeof($timeValue) > 0) {
116             $id = $timeValue[0]['id'];
117
118             // Überprüfung, ob der erste Eintrag der Echtzeitdaten in der
119             // Vergangenheit liegt
120             if ((microtime(true) + $timeDifference) > $timeValue[0]['live_time']) {
121
122                 // Überprüfung, ob der Eintrag der Echtzeitdaten eine
123                 // Geschwindigkeitsveränderung beinhaltet
124                 if ($timeValue[0]['live_is_speed_change']) {
125                     $allUsedTrains[$id]['calibrate_section_one'] = null;
126                     $allUsedTrains[$id]['calibrate_section_two'] = null;
127
128                     // Übermittlung der Echtzeitdaten bei einer Gefahrenbremsung
129                     if ($timeValue[0]['betriebsstelle'] == 'Notbremsung') {
130                         sendFahrzeugbefehl($timeValue[0]['id'], intval($timeValue[0]['
131                             ↪ live_speed']));
132                         $allTrains[$id]['speed'] = intval($timeValue[0]['live_speed']);
133                         echo 'Der Zug mit der Adresse ', $timeIndex, ' leitet gerade eine
134                             ↪ Gefahrenbremsung ein und hat seine Geschwindigkeit auf ',
135                             ↪ $timeValue[0]['live_speed'], " km/h angepasst.\n";
136                     } else {
137
138                         // Übermittlung der neuen Geschwindigkeit an das Fahrzeug
139                         sendFahrzeugbefehl($timeValue[0]['id'], intval($timeValue[0]['
140                             ↪ live_speed']));
141                         $allTrains[$id]['speed'] = intval($timeValue[0]['live_speed']);
142                         echo 'Der Zug mit der Adresse ', $timeIndex, ' hat auf der Fahrt
143                             ↪ nach ', $timeValue[0]['betriebsstelle'], ' seine
144                             ↪ Geschwindigkeit auf ', $timeValue[0]['live_speed'], " km/h

```

```

139         ↪ angepasst.\n";
140     }
141 } else {
142     if (isset($allUsedTrains[$id]['calibrate_section_one'])) {
143         if ($allUsedTrains[$id]['calibrate_section_one'] != $timeValue[0]['
144             ↪ live_section']) {
145             $allUsedTrains[$id]['calibrate_section_two'] = $timeValue[0]['
146                 ↪ live_section'];
147         }
148     }
149     $allUsedTrains[$id]['calibrate_section_one'] = $timeValue[0]['
150         ↪ live_section'];
151 }
152
153 // Aktualisierung der Position im $allUsedTrains-Array
154 $allUsedTrains[$id]['current_position'] = $timeValue[0]['
155     ↪ live_relative_position'];
156 $allUsedTrains[$id]['current_speed'] = $timeValue[0]['live_speed'];
157 $allUsedTrains[$id]['current_section'] = $timeValue[0]['live_section'];
158
159 // Überprüfung, ob die Fahrtrichtung geändert werden muss
160 if ($timeValue[0]['wendet']) {
161     changeDirection($timeValue[0]['id']);
162 }
163
164 // Überprüfung, ob das Fahrzeug eine Betriebsstelle erreicht hat
165 if (isset($timeValue[0]['live_all_targets_reached'])) {
166     $allUsedTrains[$id]['next_betriebsstellen_data'][$timeValue[0]['
167         ↪ live_all_targets_reached']]['angekommen'] = true;
168     echo 'Der Zug mit der Adresse ', $timeIndex, ' hat den Halt ',
169         ↪ $allUsedTrains[$id]['next_betriebsstellen_data'][$timeValue[0][
170         ↪ 'live_all_targets_reached']]['betriebsstelle'], " erreicht.\n";
171 }
172
173 // Überprüfung, ob ein (neuer) Fahrplan für das Fahrzeug
174 // vorliegt, wenn das ermittelte Ziel erreicht wurde
175 if ($timeValue[0]['live_target_reached']) {
176     $currentZugId = $allUsedTrains[$id]['zug_id'];

```

```

170     $newZugId = getFahrzeugZugIds(array($id));
171
172     if (sizeof($newZugId) == 0) {
173         $newZugId = null;
174     } else {
175         $newZugId = getFahrzeugZugIds(array($timeValue[0]['id']));
176         $newZugId = $newZugId[array_key_first($newZugId)]['zug_id'];
177     }
178
179     if (!($currentZugId == $newZugId && $currentZugId != null)) {
180         if ($currentZugId != null && $newZugId != null) {
181             // Das Fahrzeug hat einen neuen Fahrplan
182             $allUsedTrains[$id]['zug_id'] = $newZugId;
183             $allUsedTrains[$id]['operates_on_timetable'] = true;
184             getFahrplanAndPositionForOneTrain($id, $newZugId);
185             addStopsectionsForTimetable($id);
186             checkIfTrainReachedHaltepunkt($id);
187             checkIfStartDirectionIsCorrect($id);
188             calculateNextSections($id);
189             checkIfFahrstrasseIsCorrect($id);
190             calculateFahrtverlauf($id);
191         } else if ($currentZugId == null && $newZugId != null) {
192             // Das Fahrzeug hat jetzt einen Fahrplan und
193             // hatte davor keinen
194             $allUsedTrains[$id]['zug_id'] = $newZugId;
195             $allUsedTrains[$id]['operates_on_timetable'] = true;
196             getFahrplanAndPositionForOneTrain($id);
197             addStopsectionsForTimetable($id);
198             checkIfTrainReachedHaltepunkt($id);
199             checkIfStartDirectionIsCorrect($id);
200             calculateNextSections($id);
201             checkIfFahrstrasseIsCorrect($id);
202             calculateFahrtverlauf($id);
203         } else if ($currentZugId != null && $newZugId == null) {
204             // Das Fahrzeug fährt ab jetzt ohne Fahrplan
205             $allUsedTrains[$id]['operates_on_timetable'] = false;
206             calculateNextSections($id);
207             calculateFahrtverlauf($id);
208         }

```

```

209     }
210 }
211 array_shift($allTimes[$timeIndex]);
212 }
213 }
214 }
215
216 // Neukalibrierung der Position
217 if ($useRecalibration) {
218     if (microtime(true) > $timeCheckCalibration) {
219         foreach ($allUsedTrains as $trainKey => $trainValue) {
220             if (isset($allUsedTrains[$trainKey]['calibrate_section_two'])) {
221                 $newPosition = getCalibratedPosition($trainKey, $allUsedTrains[
222                     ↪ $trainKey]['current_speed']);
223                 if ($newPosition['possible']) {
224                     echo 'Die Position des Fahrzeugs mit der ID: ', $trainKey, " wird
225                         ↪ neu ermittelt.\n";
226                     $position = $newPosition['position'];
227                     $section = $newPosition['section'];
228                     echo 'Die alte Position war Abschnitt: ', $allUsedTrains[$trainKey][
229                         ↪ 'current_section'], ' (', number_format($allUsedTrains[
230                         ↪ $trainKey]['current_position'], 2), ' m) und die neue
231                         ↪ Position ist Abschnitt: ', $section, ' (', number_format(
232                         ↪ $position, 2), " m).\n";
233                     if ($position > $cacheInfraLaenge[$section]) {
234                         echo "Die Position konnte nicht neu kalibriert werden, da die
235                             ↪ aktuelle Position im Abschnitt größer ist, als die Länge
236                             ↪ des Abschnitts.\n";
237                     } else {
238                         $allUsedTrains[$trainKey]['current_section'] = $section;
239                         $allUsedTrains[$trainKey]['current_position'] = $position;
240                         calculateNextSections($trainKey);
241                         checkIfFahrstrasseIsCorrect($trainKey);
242                         calculateFahrtverlauf($trainKey, true);
243                         echo 'Die Position des Fahrzeugs mit der ID: ', $trainKey, " wurde
244                             ↪ neu ermittelt.\n";
245                     }
246                 }
247             }
248         }
249     }
250 }

```

```

239     }
240     $timeCheckCalibration = $timeCheckCalibration +
        ↳ $timeCheckCalibrationInterval;
241 }
242 }
243
244 // Überprüfung, ob die Fahrstraße der einzelnen Fahrzeuge sich geändert hat
245 if (microtime(true) > $timeCheckFahrstrasse) {
246     foreach ($allUsedTrains as $trainID => $trainValue) {
247         compareTwoNaechsteAbschnitte($trainID);
248     }
249
250     $returnUpdate = updateAllTrainsOnTheTrack();
251     $newTrains = $returnUpdate['new'];
252     $removeTrains = $returnUpdate['removed'];
253
254     if (sizeof($newTrains) > 0) {
255         echo "Neu hinzugefügte Züge: \n";
256         foreach ($newTrains as $newTrain) {
257             $id = $cacheDecoderToID[$newTrain];
258             echo "\tID:\t", $id, "\tAdresse:\t", $newTrain;
259         }
260         echo "\n";
261     }
262
263     foreach ($newTrains as $newTrain) {
264         $id = $cacheDecoderToID[$newTrain];
265         prepareTrainForRide($newTrain);
266         addStopsectionsForTimetable($id);
267         checkIfTrainReachedHaltepunkt($id);
268         checkIfStartDirectionIsCorrect($id);
269         consoleAllTrainsPositionAndFahrplan($id);
270         calculateNextSections($id);
271         checkIfFahrstrasseIsCorrect($id);
272         calculateFahrtverlauf($id);
273     }
274
275     if (sizeof($removeTrains) > 0) {
276         echo "Entfernte Züge:\n";

```

```

277
278     foreach ($removeTrains as $removeTrain) {
279         $id = $cacheDecoderToID[$removeTrain];
280         unset($allUsedTrains[$id]);
281         echo "\tID:\t", $id, "\tAdresse:\t", $removeTrain;
282     }
283
284     echo "\n";
285 }
286 $timeCheckFahrstrasse = $timeCheckFahrstrasse +
    ↪ $timeCheckFahrstrasseInterval;
287 }
288
289 // Ausgabe der aktuellen Positionen, Fahrplänen
290 // und Fehlermeldungen aller Fahrzeuge
291 if (microtime(true) > $timeCheckAllTrainStatus) {
292     consoleAllTrainsPositionAndFahrplan();
293     showFahrplan();
294     showErrors();
295     $timeCheckAllTrainStatus = $timeCheckAllTrainStatus +
    ↪ $timeCheckAllTrainStatusInterval;
296 }
297
298 sleep($sleeptime);
299 }

```



## A.2 functions.php

```

1 <?php
2 ini_set('memory_limit', '1024M');
3
4 // Zeigt beim Starten der Fahrzeugsteuerung eine Startmeldung im Terminal an,
5 // in der Informationen zur Session angezeigt werden.
6 function startMessage() {
7     global $simulationStartTimeToday;
8     global $simulationEndTimeToday;
9     global $simulationDuration;
10    global $realStartTime;
11    global $realEndTime;
12    global $cacheFahrplanSession;
13
14    $realStartTimeAsHHMMSS = getUhrzeit($realStartTime, 'simulationszeit', null,
        ↳ array('outputtyp' => 'h:i:s'));
15    $simulationEndTimeAsHHMMSS = getUhrzeit($simulationEndTimeToday, '
        ↳ simulationszeit', null, array('outputtyp' => 'h:i:s'));
16    $simulationDurationAsHHMMSS = toStd($simulationDuration);
17    $realEndTimeAsHHMMSS = getUhrzeit($realEndTime, 'simulationszeit', null,
        ↳ array('outputtyp' => 'h:i:s'));
18    $simulationStartTimeAsHHMMSS = getUhrzeit($simulationStartTimeToday, '
        ↳ simulationszeit', null, array('outputtyp' => 'h:i:s'));
19    $hashtagLine = "
        ↳ #####\n";
20    $emptyLine = "#\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
21
22    echo $hashtagLine;
23    echo $emptyLine;
24    echo "#\\t\\t\\t Start der automatischen Zugbeeinflussung\\t\\t\\t\\t#\\n";
25    echo "#\\t\\tim Eisenbahnbetriebs- und Experimentierfeld (EBuEf) \\t\\t#\\n";
26    echo "#\\t\\t\\t\\t\\t\\t\\t der TU Berlin\\t\\t\\t\\t\\t\\t\\t#\\n";
27    echo "#\\t\\t\\t\\t\\t im eingleisigen Netz \\t\\t\\t\\t\\t\\t\\t#\\n";
28    echo $emptyLine;
29    echo "#\\t\\t\\t\\t\\t\\t\\t____\\t\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
30    echo "#\\t\\t\\t\\t\\t\\t\\t|DD|____T_\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
31    echo "#\\t\\t\\t\\t\\t\\t\\t|_ |_____|<\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";
32    echo "#\\t\\t\\t\\t\\t\\t\\t @-@-@-oo\\\\\\t\\t\\t\\t\\t\\t\\t\\t\\t#\\n";

```



```

68 }
69
70 if ($sekunden <= 9) {
71     $strSekunden = '0'. $sekunden;
72 } else {
73     $strSekunden = $sekunden;
74 }
75
76 return "$strStunden:$strMinuten:$strSekunden";
77 }
78
79 // Fügt ein Fahrzeug zur Fahrzeugsteuerung ($allUsedTrains) über die Adresse
80 // hinzu und ermittelt die aktuelle Position und die Fahrplandaten
81 function prepareTrainForRide(int $adresse) {
82
83     global $allUsedTrains;
84     global $allTrains;
85     global $cacheAdresseToID;
86     global $cacheFmaToInfra;
87     global $cacheInfraToFma;
88     global $cacheZwischenhaltepunkte;
89     global $cacheInfraLaenge;
90     global $globalNotverzoegerung;
91
92     $trainID = $cacheAdresseToID[$adresse];
93     $zugID = null;
94     $keysZwischenhalte = array_keys($cacheZwischenhaltepunkte);
95     $allUsedTrains[$trainID]['id'] = $allTrains[$trainID]['id'];
96     $allUsedTrains[$trainID]['adresse'] = $allTrains[$trainID]['adresse'];
97     $allUsedTrains[$trainID]['zug_id'] = null;
98     $allUsedTrains[$trainID]['verzoegerung'] = floatval($allTrains[$trainID]['
        ↪ verzoegerung']);
99     $allUsedTrains[$trainID]['notverzoegerung'] = $globalNotverzoegerung;
100     $allUsedTrains[$trainID]['zuglaenge'] = $allTrains[$trainID]['zuglaenge'];
101     $allUsedTrains[$trainID]['v_max'] = $allTrains[$trainID]['v_max'];
102     $allUsedTrains[$trainID]['dir'] = $allTrains[$trainID]['dir'];
103     $allUsedTrains[$trainID]['error'] = array();
104     $allUsedTrains[$trainID]['operates_on_timetable'] = false;
105     $allUsedTrains[$trainID]['fahrstrasse_is_correct'] = false;

```

```

106 $allUsedTrains[$trainID]['current_speed'] = intval($allTrains[$trainID]['
    ↪ speed']);
107 $allUsedTrains[$trainID]['current_position'] = null;
108 $allUsedTrains[$trainID]['current_section'] = null;
109 $allUsedTrains[$trainID]['next_sections'] = array();
110 $allUsedTrains[$trainID]['next_lenghts'] = array();
111 $allUsedTrains[$trainID]['next_v_max'] = array();
112 $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array();
113 $allUsedTrains[$trainID]['next_bs'] = '';
114 $allUsedTrains[$trainID]['earliest_possible_start_time'] = null;
115 $allUsedTrains[$trainID]['calibrate_section_one'] = null;
116 $allUsedTrains[$trainID]['calibrate_section_two'] = null;
117
118 // Fehlerüberprüfung
119 if (!($allUsedTrains[$trainID]['zuglaenge'] > 0)) {
120     array_push($allUsedTrains[$trainID]['error'], 1);
121 }
122
123 if (!isset($allUsedTrains[$trainID]['v_max'])) {
124     array_push($allUsedTrains[$trainID]['error'], 2);
125 }
126
127 // Positionsermittlung
128 $fma = getPosition($adresse);
129
130 if (sizeof($fma) == 0) {
131     $allUsedTrains[$trainID]['current_fma_section'] = null;
132     $allUsedTrains[$trainID]['current_section'] = null;
133 } elseif (sizeof($fma) == 1) {
134     $allUsedTrains[$trainID]['current_fma_section'] = $fma[0];
135     $allUsedTrains[$trainID]['current_section'] = $cacheFmaToInfra[$fma[0]];
136 } else {
137     $infraArray = array();
138     foreach ($fma as $value) {
139         array_push($infraArray, $cacheFmaToInfra[$value]);
140     }
141     $infra = getFrontPosition($infraArray, $allTrains[$trainID]['dir']);
142     $allUsedTrains[$trainID]['current_fma_section'] = $cacheInfraToFma[$infra];
143     $allUsedTrains[$trainID]['current_section'] = $infra;

```

```

144 }
145
146 $allUsedTrains[$trainID]['current_position'] = $cacheInfraLaenge[
    ↳ $allUsedTrains[$trainID]['current_section']];
147 $timetableIDs = getFahrzeugZugIds(array($trainID));
148
149 if (sizeof($timetableIDs) != 0) {
150     $timetableID = $timetableIDs[array_key_first($timetableIDs)];
151     $allUsedTrains[$trainID]['zug_id'] = intval($timetableID['zug_id']);
152     $zugID = intval($timetableID['zug_id']);
153     $allUsedTrains[$trainID]['operates_on_timetable'] = true;
154 } else {
155     $allUsedTrains[$trainID]['zug_id'] = null;
156     $allUsedTrains[$trainID]['operates_on_timetable'] = false;
157 }
158
159 // Ermittlung der Fahrplaninformationen
160 if (isset($zugID)) {
161     $nextBetriebsstellen = getNextBetriebsstellen($zugID);
162 }
163
164 if ($zugID != null && sizeof($nextBetriebsstellen) != 0) {
165     for ($i = 0; $i < sizeof($nextBetriebsstellen); $i++) {
166         if (sizeof(explode('_', $nextBetriebsstellen[$i])) != 2) {
167             $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['
                ↳ is_on_fahrstrasse'] = false;
168             $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['betriebsstelle
                ↳ '] = $nextBetriebsstellen[$i];
169             $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['zeiten'] =
                ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
170             $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['fahrplanhalt'
                ↳ ] = true;
171         } else if (in_array($nextBetriebsstellen[$i], $keysZwischenhalte)) {
172             $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['
                ↳ is_on_fahrstrasse'] = false;
173             $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['betriebsstelle
                ↳ '] = $nextBetriebsstellen[$i];
174             $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['zeiten'] =
                ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);

```

```

175         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['fahrplanhalt'
176             ↳ ] = false;
177     }
178     $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array_values(
179         ↳ $allUsedTrains[$trainID]['next_betriebsstellen_data']);
180 } else {
181     $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array();
182 }
183 foreach ($allUsedTrains[$trainID]['next_betriebsstellen_data'] as
184     ↳ $betriebsstelleKey => $betriebsstelleValue) {
185     if ($allUsedTrains[$trainID]['next_betriebsstellen_data'][$
186         ↳ $betriebsstelleKey]['zeiten']['abfahrt_soll'] != null) {
187         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
188             ↳ ]['zeiten']['abfahrt_soll_timestamp'] = getUhrzeit(
189             ↳ $betriebsstelleValue['zeiten']['abfahrt_soll'], 'simulationszeit',
190             ↳ null, array('inputtyp' => 'h:i:s'));
191     } else {
192         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
193             ↳ ]['zeiten']['abfahrt_soll_timestamp'] = null;
194     }
195     if ($allUsedTrains[$trainID]['next_betriebsstellen_data'][$
196         ↳ $betriebsstelleKey]['zeiten']['ankunft_soll'] != null) {
197         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
198             ↳ ]['zeiten']['ankunft_soll_timestamp'] = getUhrzeit(
199             ↳ $betriebsstelleValue['zeiten']['ankunft_soll'], 'simulationszeit',
200             ↳ null, array('inputtyp' => 'h:i:s'));
201     } else {
202         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
203             ↳ ]['zeiten']['ankunft_soll_timestamp'] = null;
204     }
205     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey]['
206         ↳ zeiten']['versaetung'] = 0;
207 }
208 }
209 // Positionsermittlung einer Zuges, wenn das Fahrzeug mehrere
210 // Infrastrukturabschnitte belegt.

```

```

200 function getFrontPosition(array $infra, int $dir) {
201
202     foreach ($infra as $section) {
203         $nextSections = array();
204         $test = getNaechsteAbschnitte($section, $dir);
205
206         foreach ($test as $value) {
207             array_push($nextSections, $value['infra_id']);
208         }
209
210         if (sizeof(array_intersect($infra, $nextSections)) == 0) {
211             return $section;
212         }
213     }
214
215     return false;
216 }
217
218 // Ermittelt für ein Fahrzeug und die zugehörige Zug-ID den Fahrplan
219 function getFahrplanAndPositionForOneTrain (int $trainID, int $zugID) {
220
221     global $cacheZwischenhaltepunkte;
222     global $allUsedTrains;
223
224     $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array();
225     $keysZwischenhalte = array_keys($cacheZwischenhaltepunkte);
226
227     $nextBetriebsstellen = getNextBetriebsstellen($zugID);
228
229     if ($zugID != null && sizeof($nextBetriebsstellen) != 0) {
230         for ($i = 0; $i < sizeof($nextBetriebsstellen); $i++) {
231             if (sizeof(explode('_', $nextBetriebsstellen[$i])) != 2) {
232                 $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['
                     ↳ is_on_fahrstrasse'] = false;
233                 $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['betriebsstelle
                     ↳ '] = $nextBetriebsstellen[$i];
234                 $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['zeiten'] =
                     ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);

```

```

235     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['fahrplanhalt'
        ↳ ] = true;
236 } else if(in_array($nextBetriebsstellen[$i], $keysZwischenhalte)) {
237     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['
        ↳ is_on_fahrstrasse'] = false;
238     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['betriebsstelle
        ↳ '] = $nextBetriebsstellen[$i];
239     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['zeiten'] =
        ↳ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
240     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$i]['fahrplanhalt'
        ↳ ] = false;
241 }
242 }
243 $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array_values(
        ↳ $allUsedTrains[$trainID]['next_betriebsstellen_data']);
244 } else {
245     $allUsedTrains[$trainID]['next_betriebsstellen_data'] = array();
246 }
247
248 foreach ($allUsedTrains[$trainID]['next_betriebsstellen_data'] as
        ↳ $betriebsstelleKey => $betriebsstelleValue) {
249     if ($allUsedTrains[$trainID]['next_betriebsstellen_data'][$
        ↳ $betriebsstelleKey]['zeiten']['abfahrt_soll'] != null) {
250         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
            ↳ ]['zeiten']['abfahrt_soll_timestamp'] = getUhrzeit(
            ↳ $betriebsstelleValue['zeiten']['abfahrt_soll'], 'simulationszeit',
            ↳ null, array('inputtyp' => 'h:i:s'));
251     } else {
252         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
            ↳ ]['zeiten']['abfahrt_soll_timestamp'] = null;
253     }
254
255     if ($allUsedTrains[$trainID]['next_betriebsstellen_data'][$
        ↳ $betriebsstelleKey]['zeiten']['ankunft_soll'] != null) {
256         $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
            ↳ ]['zeiten']['ankunft_soll_timestamp'] = getUhrzeit(
            ↳ $betriebsstelleValue['zeiten']['ankunft_soll'], 'simulationszeit',
            ↳ null, array('inputtyp' => 'h:i:s'));
257     } else {

```



```

258     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey
        ↳ [['zeiten']['ankunft_soll_timestamp'] = null;
259 }
260
261     $allUsedTrains[$trainID]['next_betriebsstellen_data'][$betriebsstelleKey]['
        ↳ zeiten']['verspaetung'] = 0;
262 }
263 }
264
265 // Gibt in der Konsole für alle Züge (oder nur einen,
266 // wenn eine ID übergeben wird) die aktuellen Daten
267 // (Adresse, ID, Zug ID, Position, Fahrplan vorhanden,
268 // Fehler vorhanden und die Fahrtrichtung) aus.
269 function consoleAllTrainsPositionAndFahrplan($id = false) {
270
271     global $allUsedTrains;
272
273     $checkAllTrains = true;
274
275     if ($id != false) {
276         $checkAllTrains = false;
277     } else {
278         echo "Alle vorhandenen Züge:\n\n";
279     }
280
281     foreach ($allUsedTrains as $train) {
282         if ($checkAllTrains || $train['id'] == $id) {
283             $fahrplan = null;
284             $error = null;
285             $zugId = null;
286             if ($train['operates_on_timetable']) {
287                 $fahrplan = 'ja';
288             } else {
289                 $fahrplan = 'nein';
290             }
291
292             if (sizeof($train['error']) != 0) {
293                 $error = 'ja';
294             } else {

```

```

295     $error = 'nein';
296 }
297
298 if (!isset($train['zug_id'])) {
299     $zugId = '-----';
300 } else {
301     $zugId = $train['zug_id'];
302 }
303
304 echo 'Zug ID: ', $train['id'], ' (Adresse: ', $train['adresse'], ', Zug
    ↳ ID: ', $zugId, ")\\t Führt nach Fahrplan: ",
305 $fahrplan, "\\t Fahrtrichtung: ", $train['dir'], "\\t Infra-Abschnitt: ",
    ↳ $train['current_section'],
306 "\\t\\tAktuelle relative Position im Infra-Abschnitt: ", number_format(
    ↳ $train['current_position'],2), "m\\t\\tFehler vorhanden:\\t", $error,
    ↳ "\\n";
307 }
308 }
309 echo "\\n";
310 }
311
312 // Zeigt für alle Züge, die nach Fahrplan fahren (oder nur für einen Zug,
313 // wenn eine ID übergeben wird) die zuletzt erreichte Betriebsstelle und
314 // die nächsten Betriebsstellen an.
315 function showFahrplan ($id = false) {
316
317     global $allUsedTrains;
318
319     $checkAllTrains = true;
320
321     if ($id != false) {
322         $checkAllTrains = false;
323     } else {
324         echo "Alle vorhandenen Fahrpläne:\\n\\n";
325     }
326
327     foreach ($allUsedTrains as $train) {
328         if ($checkAllTrains || $train['id'] == $id) {
329             $fahrplan = null;

```

```

330     $error = null;
331     $zugId = null;
332     if ($train['operates_on_timetable']) {
333
334         if (!isset($train['zug_id'])) {
335             $zugId = '-----';
336         } else {
337             $zugId = $train['zug_id'];
338         }
339
340         $nextStations = '';
341         $lastStation = '';
342
343         foreach ($train['next_betriebsstellen_data'] as $bs) {
344             if (!$bs['angekommen']) {
345                 $nextStations = $nextStations . $bs['betriebsstelle'] . ' ';
346
347             } else {
348                 $lastStation = $bs['betriebsstelle'];
349             }
350         }
351
352         if ($lastStation == '') {
353             $lastStation = '---';
354         }
355
356         echo 'Zug ID: ', $train['id'], ' (Adresse: ', $train['adresse'], ', Zug
            ↳ ID: ', $zugId, ")\t Letzte Station: ", $lastStation, " \t Nächste
            ↳ Stationen: ", $nextStations, "\n";
357     }
358 }
359 }
360 echo "\n";
361 }
362
363 // Über prüft für alle Fahrzeuge die nach Fahrplan fahren (oder nur für ein
364 // Fahrzeug, wenn eine ID übergeben wird), ob die Fahrtrichtung mit dem
365 // Fahrplan übereinstimmt, und ob diese geändert werden muss. Wenn die
366 // Fahrtrichtung geändert werden muss, wird die Funktion changeDirection()

```

```

367 // aufgerufen
368 function checkIfStartDirectionIsCorrect($id = false) {
369
370     global $allUsedTrains;
371
372     $checkAllTrains = true;
373
374     if ($id != false) {
375         $checkAllTrains = false;
376         echo "Für den Fall, dass die Fahrtrichtung der Züge nicht mit dem Fahrplan
           ↳ übereinstimmt, wird die Richtung verändert:\n\n";
377     } else {
378         echo "Für den Fall, dass die Fahrtrichtung des Zuges nicht mit dem Fahrplan
           ↳ übereinstimmt, wird die Richtung verändert:\n\n";
379     }
380
381     foreach ($allUsedTrains as $train) {
382         if ($checkAllTrains || $train['id'] == $id) {
383             if ($train['operates_on_timetable']) {
384                 $endLoop = 0;
385                 for ($i = 0; $i < sizeof($train['next_betriebsstellen_data']); $i++) {
386                     if ($train['next_betriebsstellen_data'][$i]['angekommen']) {
387                         $endLoop = $i;
388                     }
389                 }
390
391                 if ($train['dir'] != $train['next_betriebsstellen_data'][$endLoop]['
           ↳ zeiten']['fahrtrichtung'][1]) {
392                     changeDirection($train['id']);
393                 }
394             }
395         }
396     }
397     echo "\n";
398 }
399
400 // Ändert die Fahrtrichtung eines Zuges, wenn das möglich ist. Sollte
401 // das Fahrzeug seine Richtung ändern müssen und ist dies nicht möglich,
402 // so wird dem Fahrzeug eine Fehlermeldung (Fehlerstatus = 0) hinzugefügt.

```

```

403 function changeDirection (int $id) {
404
405     global $allUsedTrains;
406     global $cacheInfraLaenge;
407     global $timeDifference;
408     global $allTrains;
409
410     $section = $allUsedTrains[$id]['current_section'];
411     $position = $allUsedTrains[$id]['current_position'];
412     $direction = $allUsedTrains[$id]['dir'];
413     $length = $allUsedTrains[$id]['zuglaenge'];
414     $newTrainLength = $length + ($cacheInfraLaenge[$section] - $position);
415     $newDirection = null;
416     $newSection = null;
417     $cumLength = 0;
418
419     if ($direction == 0) {
420         $newDirection = 1;
421     } else {
422         $newDirection = 0;
423     }
424
425     $newPosition = null;
426     $nextSections = getNaechsteAbschnitte($section, $newDirection);
427     $currentData = array(0 => array('laenge' => $cacheInfraLaenge[$section], '
        ↳ infra_id' => $section));
428     $mergedData = array_merge($currentData, $nextSections);
429
430     foreach ($mergedData as $sectionValue) {
431         $cumLength += $sectionValue['laenge'];
432
433         if ($newTrainLength <= $cumLength) {
434             $newSection = $sectionValue['infra_id'];
435             $newPosition = $cacheInfraLaenge[$newSection] - ($cumLength -
                ↳ $newTrainLength);
436             break;
437         }
438     }
439

```

```

440 if ($newPosition == null) {
441     echo 'Die Richtung des Zugs mit der ID ', $id, " lässt sich nicht ändern,
        ↳ weil das Zugende auf einem auf Halt stehenden Signal steht.\n";
442     echo "\tDie Zuglänge beträgt:\t", $length, " m\n\tDie Distanz zwischen
        ↳ Zugende und dem auf Halt stehenden Signal beträgt:\t", ($cumLength -
        ↳ ($cacheInfraLaenge[$section] - $position)), " m\n\n";
443     array_push($allUsedTrains[$id]['error'], 0);
444 } else {
445     echo 'Die Richtung des Zugs mit der ID: ', $id, ' wurde auf ',
        ↳ $newDirection, " geändert.\n";
446     $allUsedTrains[$id]['current_section'] = $newSection;
447     $allUsedTrains[$id]['current_position'] = $newPosition;
448     $allUsedTrains[$id]['dir'] = $newDirection;
449     $allUsedTrains[$id]['earliest_possible_start_time'] = FZS_WARTEZEIT_WENDEN
        ↳ + time() + $timeDifference;
450     $allTrains[$id]['dir'] = $newDirection;
451     $DB = new DB_MySQL();
452     $DB->select('UPDATE '.DB_TABLE_FAHRZEUGE.' SET '.DB_TABLE_FAHRZEUGE.'"'.
        ↳ dir' = $newDirection WHERE '".DB_TABLE_FAHRZEUGE.'"'.id' = $id");
453     unset($DB);
454     sendFahrzeugbefehl($id, -4);
455 }
456 }
457
458 // Gibt für alle Fahrzeuge die vorhanden Fehlermeldungen an.
459 function showErrors() {
460
461     global $allUsedTrains;
462     global $trainErrors;
463
464     $foundError = false;
465     echo "Hier werden für alle Züge mögliche Fehler angezeigt:\n\n";
466
467     foreach ($allUsedTrains as $trainIndex => $trainValue) {
468         if (sizeof($trainValue['error']) != 0) {
469             $foundError = true;
470             echo 'Zug ID: ', $trainValue['id'], "\n";
471             $index = 1;
472

```

```

473     foreach ($trainValue['error'] as $error) {
474         echo "\t", $index, ". Fehler:\t", $trainErrors[$error], "\n";
475         $index++;
476     }
477
478     echo "\n";
479 }
480 }
481
482 if (!$foundError) {
483     echo "Keiner der Züge hat eine Fehlermeldung.\n";
484 }
485 }
486
487 // Fügt allen Fahrzeugen (oder nur einem Fahrzeug,
488 // wenn eine ID übergeben wird), die nach Fahrplan
489 // fahren, mögliche Halte-Infrastrukturabschnitte hinzu.
490 function addStopsectionsForTimetable($id = false) {
491
492     global $allUsedTrains;
493     global $cacheHaltepunkte;
494     global $cacheZwischenhaltepunkte;
495
496     $checkAllTrains = true;
497
498     if ($id != false) {
499         $checkAllTrains = false;
500     }
501
502     foreach ($allUsedTrains as $trainIndex => $trainValue) {
503         if ($checkAllTrains || $trainValue['id'] == $id) {
504             if (sizeof($trainValue['error']) == 0) {
505                 if ($trainValue['operates_on_timetable']) {
506                     foreach ($trainValue['next_betriebsstellen_data'] as
507                         ↳ $betriebsstelleKey => $betriebsstelleValue) {
508                         if (in_array($betriebsstelleValue['betriebsstelle'], array_keys(
509                             ↳ $cacheHaltepunkte))) {
510                             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$
511                                 ↳ $betriebsstelleKey]['haltepunkte'] = $cacheHaltepunkte[

```

```

509         ↪ $betriebsstelleValue['betriebsstelle']][$strainValue['dir']];
    } else if (in_array($betriebsstelleValue['betriebsstelle'],
        ↪ array_keys($cacheZwischenhaltepunkte))) {
510        $allUsedTrains[$strainIndex]['next_betriebsstellen_data']['
            ↪ $betriebsstelleKey]['haltepunkte'] = array(
            ↪ $cacheZwischenhaltepunkte[$betriebsstelleValue['
            ↪ betriebsstelle']]);
511    } else {
512        $allUsedTrains[$strainIndex]['next_betriebsstellen_data']['
            ↪ $betriebsstelleKey']['haltepunkte'] = array();
513    }
514 }
515 }
516 }
517 }
518 }
519 }
520
521 // Ermittelt für alle Fahrzeuge (wenn keine ID übergeben wird) oder für ein
522 // Fahrzeug (wenn eine ID übergeben wird) die Fahrstraße inkl. der Längen,
523 // der zulässigen Höchstgeschwindigkeiten und der IDs der nächsten Abschnitte.
524 //
525 // Die Ergebnisse können direkt im Array $usedTrains gespeichert werden
526 // ($writeResultToTrain = true) oder als return zurückgegeben werden
527 // ($writeResultToTrain = false), so dass sie verglichen werden können
528 // mit den vorherigen Daten verglichen werden können.
529 function calculateNextSections($id = false, $writeResultToTrain = true) {
530
531     global $allUsedTrains;
532     global $cacheInfraLaenge;
533     global $globalSpeedInCurrentSection;
534     global $lastMaxSpeedForInfraAndDir;
535
536     $checkAllTrains = true;
537
538     if ($id != false) {
539         $checkAllTrains = false;
540     }
541

```



```

542 foreach ($allUsedTrains as $trainIndex => $trainValue) {
543     if (($checkAllTrains || $trainValue['id'] == $id) && sizeof($trainValue['
        ↳ error']) == 0) {
544         $dir = $trainValue['dir'];
545         $currentSectionComp = $trainValue['current_section'];
546         $signal = getSignalForSectionAndDirection($currentSectionComp, $dir);
547         $nextSectionsComp = array();
548         $nextVMaxComp = array();
549         $nextLengthsComp = array();
550         $nextSignalbegriff = null;
551
552         if ($signal != null) {
553             $nextSignalbegriff = getSignalbegriff($signal);
554             $nextSignalbegriff = $nextSignalbegriff[array_key_last(
                ↳ $nextSignalbegriff)]['geschwindigkeit'];
555             if ($nextSignalbegriff == -25) {
556                 $nextSignalbegriff = 25;
557             } else if ($nextSignalbegriff <= 0) {
558                 $nextSignalbegriff = 0;
559             }
560         } else {
561             $nextSignalbegriff = null;
562         }
563
564         $return = getNaechsteAbschnitte($currentSectionComp, $dir);
565         $allUsedTrains[$trainIndex]['last_get_naechste_abschnitte'] = $return;
566
567         if (isset($lastMaxSpeedForInfraAndDir[$trainValue['dir']][$trainValue['
            ↳ current_section']])) {
568             $currentVMax = $lastMaxSpeedForInfraAndDir[$trainValue['dir']][
                ↳ $trainValue['current_section']];
569         } else {
570             $currentVMax = $globalSpeedInCurrentSection;
571         }
572
573         array_push($nextSectionsComp, $currentSectionComp);
574         array_push($nextVMaxComp, $currentVMax);
575         array_push($nextLengthsComp, $cacheInfraLaenge[$currentSectionComp]);
576

```

```

577     if (isset($nextSignalbegriff)) {
578         $currentVMax = $nextSignalbegriff;
579     }
580
581     if ($currentVMax == 0) {
582         if ($writeResultToTrain) {
583             $allUsedTrains[$trainIndex]['next_sections'] = $nextSectionsComp;
584             $allUsedTrains[$trainIndex]['next_lengths'] = $nextLengthsComp;
585             $allUsedTrains[$trainIndex]['next_v_max'] = $nextVMaxComp;
586         } else {
587             return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
588         }
589     } else {
590         foreach ($return as $section) {
591             array_push($nextSectionsComp, $section['infra_id']);
592             array_push($nextVMaxComp, $currentVMax);
593             array_push($nextLengthsComp, $cacheInfraLaenge[$section['infra_id']]);
594             $lastMaxSpeedForInfraAndDir[intval($trainValue['dir'])][intval(
595                 ↳ $section['infra_id'])] = intval($currentVMax);
596             if ($section['signal_id'] != null) {
597                 $signal = $section['signal_id'];
598                 $nextSignalbegriff = getSignalbegriff($signal);
599                 $nextSignalbegriff = $nextSignalbegriff[array_key_last(
600                     ↳ $nextSignalbegriff)]['geschwindigkeit'];
601                 if ($nextSignalbegriff == -25) {
602                     $currentVMax = 25;
603                 } else if ($nextSignalbegriff < 0) {
604                     $currentVMax = 0;
605                 } else {
606                     $currentVMax = $nextSignalbegriff;
607                 }
608             }
609         }
610     }
611     if ($writeResultToTrain) {
612         $allUsedTrains[$trainIndex]['next_sections'] = $nextSectionsComp;
613         $allUsedTrains[$trainIndex]['next_lengths'] = $nextLengthsComp;
614         $allUsedTrains[$trainIndex]['next_v_max'] = $nextVMaxComp;
615     } else {
616         return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);

```

```

614     }
615 }
616 }
617 }
618 }
619
620 // Prüft für alle Fahrzeuge (falls keine ID übergeben wird)
621 // oder für ein Fahrzeug (falls eine ID übergeben wird),
622 // ob das Fahrzeug bereits am ersten fahrplanmäßigen
623 // Halt ist oder nicht.
624 function checkIfTrainReachedHaltepunkt ($id = false) {
625
626     global $allUsedTrains;
627     global $cacheInfraToGbt;
628     global $cacheGbtToInfra;
629
630     $checkAllTrains = true;
631
632     if ($id != false) {
633         $checkAllTrains = false;
634     }
635
636     foreach ($allUsedTrains as $trainIndex => $trainValue) {
637         if ($checkAllTrains || $trainValue['id'] == $id) {
638             $currentInfrasection = $trainValue['current_section'];
639             $currentGbt = $cacheInfraToGbt[$currentInfrasection];
640             $allInfraSections = $cacheGbtToInfra[$currentGbt];
641             for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data']); $i++)
642                 ↪ {
643                 if (sizeof(array_intersect($trainValue['next_betriebsstellen_data'][$i][
644                     ↪ 'haltepunkte'], $allInfraSections)) != 0) {
645                     $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$i]['
646                         ↪ angekommen'] = true;
647                     for ($j = 0; $j < $i; $j++) {
648                         $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$j]['
649                             ↪ angekommen'] = true;
650                     }
651                 } else {

```

```

648         $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$i]['
           ↳ angekommen'] = false;
649     }
650 }
651 }
652 }
653 }
654
655 // Prüft für alle Fahrzeuge (falls keine ID übergeben wird)
656 // oder für ein Fahrzeug (falls eine ID übergeben wird),
657 // ob die Fahrstraße aktuell richtig eingestellt ist,
658 // sodass die nächste Betriebsstelle laut Fahrplan
659 // erreicht werden kann.
660 //
661 // Für Züge ohne Fahrplan ist der Fahrweg immer korrekt.
662 function checkIfFahrstrasseIsCorrect($id = false) {
663
664     global $allUsedTrains;
665
666     $checkAllTrains = true;
667
668     if ($id != false) {
669         $checkAllTrains = false;
670     }
671
672     foreach ($allUsedTrains as $trainIndex => $trainValue) {
673         if (($checkAllTrains || $trainValue['id'] == $id) && sizeof($trainValue['
           ↳ error']) == 0) {
674             if ($trainValue['operates_on_timetable']) {
675                 $allUsedTrains[$trainIndex]['fahrstrasse_is_correct'] = false;
676                 foreach ($trainValue['next_betriebsstellen_data'] as $stopIndex =>
           ↳ $stopValue) {
677                     if (!$stopValue['angekommen']) {
678                         $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$stopIndex
           ↳ ]['is_on_fahrstrasse'] = false;
679                         $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$stopIndex
           ↳ ]['used_haltepunkt'] = array();
680                         $indexSection = 0;
681                         for ($i = 0; $i < sizeof($trainValue['next_sections']); $i++) {

```

```

682         if ($stopValue['haltepunkte'] != null) {
683             if (in_array($trainValue['next_sections'][$i], $stopValue['
                ↳ haltepunkte'])) {
684                 if ($i >= $indexSection) {
685                     $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$
                        ↳ $stopIndex]['is_on_fahrstrasse'] = true;
686                     $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$
                        ↳ $stopIndex]['used_haltepunkt'] = $trainValue['
                            ↳ next_sections'][$i];
687                     $allUsedTrains[$trainIndex]['fahrstrasse_is_correct'] = true;
688                     $i = sizeof($trainValue['next_sections']);
689                     $indexSection = $i;
690                 }
691             }
692         }
693     }
694 } else {
695     $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][$stopIndex
        ↳ ]['is_on_fahrstrasse'] = true;
696 }
697 }
698 } else {
699     $allUsedTrains[$trainIndex]['fahrstrasse_is_correct'] = true;
700 }
701 }
702 }
703 }
704
705 // Berechnet die Beschleunigungs- und Bremskurven für alle Züge (wenn keine ID
706 // übergeben wird) oder für einen Zug (wenn eine ID übergeben wird). Für Züge -
707 // die nach Fahrplan fahren - für alle Betriebsstellen, die auf der aktuell
708 // eingestellten Strecke liegen und für Züge ohne Fahrplan bis zum nächsten
709 // roten Signal.
710 function calculateFahrtverlauf($id = false, $recalibrate = false) {
711
712     global $allUsedTrains;
713     global $cacheInfraLaenge;
714     global $timeDifference;
715     global $globalFirstHaltMinTime;

```

```

716
717 $checkAllTrains = true;
718
719 if ($id != false) {
720     $checkAllTrains = false;
721 }
722
723 foreach ($allUsedTrains as $trainIndex => $trainValue) {
724     $allPossibleStops = array();
725     for($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data']); $i++) {
726         if ($trainValue['next_betriebsstellen_data'][$i]['fahrplanhalt']) {
727             array_push($allPossibleStops, $i);
728         }
729     }
730     if (sizeof($trainValue['error']) == 0 && $trainValue['
        ↳ fahrstrasse_is_correct']) {
731         if ($checkAllTrains || $trainValue['id'] == $id) {
732             if ($trainValue['operates_on_timetable']) {
733                 $nextBetriebsstelleIndex = null;
734                 $allreachedInfras = array();
735                 $wendet = false;
736                 for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data']); $i
                    ↳ ++)) {
737                     if (!$trainValue['next_betriebsstellen_data'][$i]['angekommen'] &&
                        ↳ $trainValue['next_betriebsstellen_data'][$i]['
                        ↳ is_on_fahrstrasse'] && $trainValue['next_betriebsstellen_data
                        ↳ '][$i]['fahrplanhalt']) {
738                         $nextBetriebsstelleIndex = $i;
739                         $allUsedTrains[$trainIndex]['next_bs'] = $i;
740                         break;
741                     }
742                 }
743                 if (!isset($nextBetriebsstelleIndex)) {
744                     for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data']);
                        ↳ $i++) {
745                         if (!$trainValue['next_betriebsstellen_data'][$i]['angekommen'] &&
                            ↳ $trainValue['next_betriebsstellen_data'][$i]['
                            ↳ is_on_fahrstrasse']) {
746                             $nextBetriebsstelleIndex = $i;

```

```

747         break;
748     }
749 }
750 }
751 if (isset($nextBetriebsstelleIndex)) {
752     if ($allUsedTrains[$trainIndex]['next_bs'] != $trainValue['
        ↳ next_betriebsstellen_data'][$nextBetriebsstelleIndex]['
        ↳ betriebsstelle'] || $recalibrate) {
753         $allUsedTrains[$trainIndex]['next_bs'] = $trainValue['
        ↳ next_betriebsstellen_data'][$nextBetriebsstelleIndex]['
        ↳ betriebsstelle'];
754         if (intval($trainValue['next_betriebsstellen_data'][
        ↳ $nextBetriebsstelleIndex]['zeiten']['wendet']) == 1) {
755             $wendet = true;
756         }
757         for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data'])
        ↳ ; $i++) {
758             if (!$trainValue['next_betriebsstellen_data'][$i]['angekommen']
        ↳ && $trainValue['next_betriebsstellen_data'][$i]['
        ↳ is_on_fahrstrasse'] && $i <= $nextBetriebsstelleIndex) {
759                 array_push($allreachedInfras, array('index' => $i, 'infra' =>
        ↳ $trainValue['next_betriebsstellen_data'][$i]['
        ↳ used_haltepunkt']));
760             }
761         }
762         $targetSection = $trainValue['next_betriebsstellen_data'][
        ↳ $nextBetriebsstelleIndex]['used_haltepunkt'];
763         $targetPosition = $cacheInfraLaenge[$targetSection];
764         $startTime = null;
765         $endTime = null;
766         $prevBetriebsstelle = null;
767         for ($i = 0; $i < sizeof($trainValue['next_betriebsstellen_data'])
        ↳ ; $i++) {
768             if ($trainValue['next_betriebsstellen_data'][$i]['angekommen'])
        ↳ {
769                 $prevBetriebsstelle = $i;
770                 break;
771             }
772         }

```

```

773     if ($nextBetriebsstelleIndex == 0) {
774         $startTime = microtime(true) + $timeDifference;
775         $endTime = $startTime;
776     } else {
777         $endTime = $trainValue['next_betriebsstellen_data'][
            ↳ $nextBetriebsstelleIndex]['zeiten']['
            ↳ ankunft_soll_timestamp'];
778         if (isset($prevBetriebsstelle)) {
779             if ($trainValue['next_betriebsstellen_data'][
                ↳ $prevBetriebsstelle]['zeiten']['verspaetung'] > 0) {
780                 $startTime = $trainValue['next_betriebsstellen_data'][
                    ↳ $prevBetriebsstelle]['zeiten']['abfahrt_soll_timestamp
                    ↳ '] + $trainValue['next_betriebsstellen_data'][
                    ↳ $nextBetriebsstelleIndex - 1]['zeiten']['verspaetung'
                    ↳ ];
781             } else {
782                 $startTime = $trainValue['next_betriebsstellen_data'][
                    ↳ $prevBetriebsstelle]['zeiten']['abfahrt_soll_timestamp
                    ↳ '];
783             }
784         } else {
785             $startTime = microtime(true) + $timeDifference;
786         }
787     }
788     $reachedBetriebsstele = true;
789
790     if ($startTime < microtime(true) + $timeDifference) {
791         $startTime = microtime(true) + $timeDifference;
792     }
793
794     if (isset($trainValue['earliest_possible_start_time'])) {
795         if ($startTime < $trainValue['earliest_possible_start_time']) {
796             $startTime = $trainValue['earliest_possible_start_time'];
797         }
798     }
799
800     $verapetung = updateNextSpeed($trainValue, $startTime, $endTime,
        ↳ $targetSection, $targetPosition, $reachedBetriebsstele,
        ↳ $nextBetriebsstelleIndex, $wendet, false, $allreachedInfras

```



```

    ↪ );
801
802     if ($nextBetriebsstelleIndex != 0) {
803         $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
            ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
            ↪ $verapetung;
804         $trainValue['next_betriebsstellen_data'][
            ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
            ↪ $verapetung;
805     } else {
806         $end = $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
            ↪ $nextBetriebsstelleIndex]['zeiten']['
            ↪ abfahrt_soll_timestamp'];
807         $start = $startTime;
808         if ($start + $verapetung + $globalFirstHaltMinTime < $end) {
809             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] = 0;
810             $trainValue['next_betriebsstellen_data'][
                ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] = 0;
811         } else {
812             $allUsedTrains[$trainIndex]['next_betriebsstellen_data'][
                ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
                ↪ $start + $verapetung + $globalFirstHaltMinTime - $end;
813             $trainValue['next_betriebsstellen_data'][
                ↪ $nextBetriebsstelleIndex]['zeiten']['verspaetung'] =
                ↪ $start + $verapetung + $globalFirstHaltMinTime - $end;
814         }
815     }
816 }
817 } else {
818     if ($trainValue['current_speed'] > 0) {
819         emergencyBreak($trainValue['id']);
820     }
821 }
822 } else {
823     $startTime = microtime(true) + $timeDifference;
824     if (isset($trainValue['earliest_possible_start_time'])) {
825         if ($startTime < $trainValue['earliest_possible_start_time']) {
826             $startTime = $trainValue['earliest_possible_start_time'];

```

```

827     }
828 }
829 $endTime = $startTime;
830 $targetSection = null;
831 $targetPosition = null;
832 $reachedBetriebsstele = true;
833 $wendet = false;
834 $signalId = null;
835 for ($i = 0; $i < sizeof($trainValue['last_get_naechste_abschnitte']);
      ↪ $i++) {
836     if (isset($trainValue['last_get_naechste_abschnitte'][$i]['signal_id']
      ↪ '')) {
837         $signalId = $trainValue['last_get_naechste_abschnitte'][$i]['
      ↪ signal_id'];
838         $targetSection = $trainValue['last_get_naechste_abschnitte'][$i]['
      ↪ infra_id'];
839         $targetPosition = $cacheInfraLaenge[$targetSection];
840     }
841 }
842 if (!isset($signalId)) {
843     // gibt kein nächstes Signal
844     if ($trainValue['current_speed'] != 0) {
845         emergencyBreak($trainValue['id']);
846     }
847 } else {
848     $signal = getSignalbegriff($signalId)[0]['geschwindigkeit'];
849
850     if ($signal > -25 && $signal < 0) {
851         $wendet = true;
852     }
853
854     updateNextSpeed($trainValue, $startTime, $endTime, $targetSection,
      ↪ $targetPosition, $reachedBetriebsstele, $signalId, $wendet,
      ↪ true, array());
855 }
856 }
857 }
858 } else {
859     if ($trainValue['current_speed'] != 0) {

```

```

860         emergencyBreak($trainValue['id']);
861     }
862 }
863 }
864 }
865
866 // Vergleicht für ein Fahrzeug die zuletzt ermittelte
867 // Fahrstraße mit der aktuellen Fahrstraße und berechnet
868 // den Fahrtverlauf neu, wenn das nötig ist.
869 function compareTwoNaechsteAbschnitte(int $id) {
870
871     global $allUsedTrains;
872     global $allTimes;
873
874     if (sizeof($allUsedTrains[$id]['error']) == 0) {
875         $newSections = calculateNextSections($id, false);
876         $newNextSection = $newSections[0];
877         $newNextLenghts = $newSections[1];
878         $newNextVMax = $newSections[2];
879         $oldNextSections = $allUsedTrains[$id]['next_sections'];
880         $oldLenghts = $allUsedTrains[$id]['next_lenghts'];
881         $oldNextVMax = $allUsedTrains[$id]['next_v_max'];
882         $currentSectionOld = $allUsedTrains[$id]['current_section'];
883         $keyCurrentSection = array_search($currentSectionOld, $oldNextSections);
884         $keyLatestSection = array_key_last($oldNextSections);
885         $dataIsIdentical = true;
886         $numberOfSection = $keyLatestSection - $keyCurrentSection + 1;
887         $compareNextSections = array();
888         $compareNextLenghts = array();
889         $compareNextVMax = array();
890
891         for($i = $keyCurrentSection; $i <= $keyLatestSection; $i++) {
892             array_push($compareNextSections, $oldNextSections[$i]);
893             array_push($compareNextLenghts, $oldLenghts[$i]);
894             array_push($compareNextVMax, $oldNextVMax[$i]);
895         }
896
897         if (sizeof($newNextSection) != ($numberOfSection)) {
898             $dataIsIdentical = false;

```

```

899 } else {
900     for ($i = 0; $i < $keyLatestSection - $keyCurrentSection; $i++) {
901         if ($newNextSection[$i] != $compareNextSections[$i] || $newNextLenghts[
            ↳ $i] != $compareNextLenghts[$i] || $newNextVMax[$i] !=
            ↳ $compareNextVMax[$i]) {
902             $dataIsIdentical = false;
903             break;
904         }
905     }
906 }
907
908 if (!$dataIsIdentical) {
909     echo 'Die Fahrstraße des Zuges mit der ID: ', $id, " hat sich geändert.\n
        ↳ ";
910     calculateNextSections($id);
911     $adresse = $allUsedTrains[$id]['adresse'];
912     $allTimes[$adresse] = array();
913     checkIfFahrstrasseIsCorrect($id);
914     calculateFahrtverlauf($id);
915 }
916 }
917 }

```

### A.3 functions\_fahrtverlauf.php

```
1 <?php
2
3 // Berechnet den Fahrtverlauf eines Fahrzeugs
4 function updateNextSpeed (array $train, float $startTime, float $endTime, int
    ↳ $targetSectionPara, int $targetPositionPara, bool $reachedBetriebsstelle
    ↳ , string $indexReachedBetriebsstelle, bool $wendet, bool $freieFahrt,
    ↳ array $allreachedInfras) {
5
6     global $useSpeedFineTuning;
7     global $next_sections;
8     global $next_lengths;
9     global $next_v_max;
10    global $allTimes;
11    global $verzoegerung;
12    global $notverzoegerung;
13    global $currentSection;
14    global $currentPosition;
15    global $currentSpeed;
16    global $targetSpeed;
17    global $targetSection;
18    global $targetPosition;
19    global $targetTime;
20    global $indexCurrentSection;
21    global $indexTargetSection;
22    global $distanceToNextStop;
23    global $trainSpeedChange;
24    global $trainPositionChange;
25    global $trainTimeChange;
26    global $cumulativeSectionLengthEnd;
27    global $cumulativeSectionLengthStart;
28    global $keyPoints;
29    global $allUsedTrains;
30    global $globalIndexBetriebsstelleFreieFahrt;
31    global $cacheSignalIDToBetriebsstelle;
32    global $useMinTimeOnSpeed;
33    global $slowDownIfTooEarly;
34    global $globalFloatingPointNumbersRoundingError;
35
```

```

36 $emptyArray = array();
37 $keyPoints = $emptyArray;
38 $cumulativeSectionLengthStart = $emptyArray;
39 $cumulativeSectionLengthEnd = $emptyArray;
40 $next_sections = $train['next_sections'];
41 $next_lengths = $train['next_lengths'];
42 $next_v_max = $train['next_v_max'];
43 $verzoegerung = $train['verzoegerung'];
44 $notverzoegerung = $train['notverzoegerung'];
45 $train_v_max = $train['v_max'];
46 $currentSection = $train['current_section'];
47 $currentPosition = $train['current_position'];
48 $currentSpeed = $train['current_speed'];
49 $train_length = $train['zuglaenge'];
50 $targetSection = $targetSectionPara;
51 $targetPosition = $targetPositionPara;
52 $targetSpeed = 0;
53 $targetTime = $endTime;
54 $indexCurrentSection = null;
55 $indexTargetSection = null;
56 $timeToNextStop = null;
57 $maxTimeToNextStop = $targetTime - $startTime;
58 $maxSpeedNextSections = 120;
59
60 if (!$freieFahrt) {
61     $targetBetriebsstelle = $train['next_betriebsstellen_data'][
        ↳ $indexReachedBetriebsstelle]['betriebsstelle'];
62 } else {
63     $targetBetriebsstelle = $cacheSignalIDToBetriebsstelle[intval(
        ↳ $indexReachedBetriebsstelle)];
64 }
65
66 // Überprüfung, ob das Fahrzeug bereits am Ziel steht
67 if ($targetSection == $currentSection && $targetPosition == $currentPosition)
    ↳ {
68     if ($currentSpeed > 0) {
69         emergencyBreak($train['id']);
70     } else {
71         $allTimes[$train['adresse']] = array();

```

```

72     return 0;
73 }
74 }
75
76 // Wenn ein Infra-Abschnitt eine Geschwindigkeit zulässt,
77 // die größer als die zulässige Höchstgeschwindigkeit des
78 // Fahrzeugs ist, wird die Geschwindigkeit des
79 // Infra-Abschnitts reduziert.
80 if ($train_v_max != null) {
81     foreach ($next_sections as $sectionKey => $sectionValue) {
82         if ($next_v_max[$sectionKey] > $train_v_max) {
83             $next_v_max[$sectionKey] = $train_v_max;
84         }
85     }
86 }
87
88 // Ermittlung der Indexe des Start- und Zielabschnitts
89 foreach ($next_sections as $sectionKey => $sectionValue) {
90     if ($sectionValue == $currentSection) {
91         $indexCurrentSection = $sectionKey;
92     }
93
94     if ($sectionValue == $targetSection) {
95         $indexTargetSection = $sectionKey;
96     }
97 }
98
99 // Berechnet die kumulierten Abstände jedes Infra-Abschnitts für den Anfang
100 // und das Ende der Infra-Abschnitt von der aktuellen Fahrzeugposition
101 $returnCumulativeSections = createCumulativeSections($indexCurrentSection,
102     ↪ $indexTargetSection, $currentPosition, $targetPosition, $next_lengths)
103     ↪ ;
104 $cumulativeSectionLengthStart = $returnCumulativeSections[0];
105 $cumulativeSectionLengthEnd = $returnCumulativeSections[1];
106 $cumLengthEnd = array();
107 $cumLengthStart = array();
108 $sum = 0;
109
110 foreach ($next_lengths as $index => $value) {

```

```

109     if ($index >= $indexCurrentSection) {
110         $cumLengthStart[$index] = $sum;
111         $sum += $value;
112         $cumLengthEnd[$index] = $sum;
113     }
114 }
115
116 // Ermittlung der Distanz bis zum Ziel
117 $distanceToNextStop = $cumulativeSectionLengthEnd[$indexTargetSection];
118 if (getBrakeDistance($currentSpeed, $targetSpeed, $verzoegerung)>
119     ↪ $distanceToNextStop && $currentSpeed != 0) {
119     if (!isset($distanceToNextStop)) {
120         emergencyBreak($train['id']);
121
122         return 0;
123     } else {
124         emergencyBreak($train['id'], $distanceToNextStop);
125
126         return 0;
127     }
128 }
129
130 // Ermittlung der Längen und zulässigen Höchstgeschwindigkeiten der
131 // Infra-Abschnitte inkl. Zuglänge
132 global $next_v_max_mod;
133 global $next_lengths_mod;
134 global $indexCurrentSectionMod;
135 global $indexTargetSectionMod;
136
137 $next_v_max_mod = array();
138 $next_lengths_mod = array();
139 $indexCurrentSectionMod = null;
140 $indexTargetSectionMod = null;
141
142 if ($indexCurrentSection == $indexTargetSection) {
143     $next_lengths_mod = $next_lengths;
144     $next_v_max_mod = $next_v_max;
145     $indexCurrentSectionMod = $indexCurrentSection;
146     $indexTargetSectionMod = $indexTargetSection;

```



```

147     $next_lengths_mod[$indexTargetSectionMod] = $targetPosition;
148 } else {
149     $startPosition = 0;
150     $indexStartPosition = null;
151     $indexEndPosition = null;
152
153     do {
154         $reachedTargetSection = false;
155
156         for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
157             if ($startPosition >= $cumLengthStart[$j] && $startPosition <
158                 ↪ $cumLengthEnd[$j]) {
159                 $indexStartPosition = $j;
160             }
161         }
162
163         $endPosition = $cumLengthEnd[$indexStartPosition] + $strain_length;
164         $current_v_max = $next_v_max[$indexStartPosition];
165
166         if ($endPosition >= $cumLengthEnd[$indexTargetSection]) {
167             $indexEndPosition = $indexTargetSection;
168             $endPosition = $cumLengthEnd[$indexTargetSection - 1] + $targetPosition;
169             $reachedTargetSection = true;
170         } else {
171             for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
172                 if ($endPosition >= $cumLengthStart[$j] && $endPosition <
173                     ↪ $cumLengthEnd[$j]) {
174                     $indexEndPosition = $j;
175                 }
176             }
177
178             for ($j = $indexStartPosition + 1; $j <= $indexEndPosition; $j++) {
179                 if ($next_v_max[$j] < $current_v_max) {
180                     $endPosition = $cumLengthStart[$j];
181                     $indexEndPosition = $j - 1;
182                 }
183             }

```

```

184     if ($reachedTargetSection) {
185         if (!($endPosition >= $distanceToNextStop)) {
186             $reachedTargetSection = false;
187         }
188     }
189
190     array_push($next_lengths_mod, ($endPosition - $startPosition));
191     array_push($next_v_max_mod, $current_v_max);
192     $startPosition = $endPosition;
193 } while (!$reachedTargetSection);
194
195 $indexCurrentSectionMod = array_key_first($next_lengths_mod);
196 $indexTargetSectionMod = array_key_last($next_lengths_mod);
197 }
198
199 // Berechnet die kumulierten Abstände jedes Infra-Abschnitts für den Anfang
200 // und das Ende der Infra-Abschnitt von der aktuellen Fahrzeugposition
201 // inkl. der Fahrzeuglänge
202 $returnCumulativeSectionsMod = createCumulativeSections(
203     ↪ $indexCurrentSectionMod, $indexTargetSectionMod, $currentPosition,
204     ↪ $next_lengths_mod[$indexTargetSectionMod], $next_lengths_mod);
205
206 global $cumulativeSectionLengthStartMod;
207 global $cumulativeSectionLengthEndMod;
208
209 $cumulativeSectionLengthStartMod = $returnCumulativeSectionsMod[0];
210 $cumulativeSectionLengthEndMod = $returnCumulativeSectionsMod[1];
211 $minTimeOnSpeedIsPossible = checkIfItsPossible($train['id']);
212 $v_maxFirstIteration = getVMaxBetweenTwoPoints($distanceToNextStop,
213     ↪ $currentSpeed, $targetSpeed, $train['id']);
214
215 // Anpassung an die maximale Geschwindigkeit auf der Strecke
216 for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
217     if ($next_v_max[$i] < $maxSpeedNextSections) {
218         $maxSpeedNextSections = $next_v_max[$i];
219     }
220 }
221
222 if ($maxSpeedNextSections < $v_maxFirstIteration) {
223     $v_maxFirstIteration = $maxSpeedNextSections;

```

```

220 }
221
222 // Key Points für die erste Iteration erstellen.
223 array_push($keyPoints, createKeyPoint(0, getBrakeDistance($currentSpeed,
    ↳ $v_maxFirstIteration, $verzoegerung), $currentSpeed,
    ↳ $v_maxFirstIteration));
224 array_push($keyPoints, createKeyPoint(($distanceToNextStop - getBrakeDistance
    ↳ ($v_maxFirstIteration, $targetSpeed, $verzoegerung)),
    ↳ $distanceToNextStop, $v_maxFirstIteration, $targetSpeed));
225
226 //$trainChange = convertKeyPointsToTrainChangeArray($keyPoints);
227 $trainChange = createTrainChanges(true);
228 $trainPositionChange = $trainChange[0];
229 $trainSpeedChange = $trainChange[1];
230 $speedOverPositionAllIterations = array();
231
232 // Überprüfung, ob das Fahrzeug in Infra-Abschnitten zu schnell ist
233 while (checkIfTrainIsTooFastInCertainSections()['failed']) {
234     $tempKeyPoints = $keyPoints;
235
236     // Berechnung der Echtzeitdaten
237     $trainChange = createTrainChanges(true);
238     $trainPositionChange = $trainChange[0];
239     $trainSpeedChange = $trainChange[1];
240
241     // Hinzufügen der Echtzeitdaten des vorherigen Iterationsschritt
242     // für die Visualisierung
243     array_push($speedOverPositionAllIterations, array($trainPositionChange,
        ↳ $trainSpeedChange));
244
245     // Überprüfung, ob durch den Fahrtverlauf zulässige Höchst-
246     // geschwindigkeiten überschritten werden
247     $keyPoints = recalculateKeyPoints($tempKeyPoints, $train['id']);
248     $localKeyPointsTwo = array();
249
250     // Entfernen von doppelten $keyPoints
251     for ($i = 0; $i < sizeof($keyPoints); $i++) {
252         if ($i < sizeof($keyPoints) - 1) {

```

```

253     if (!($keyPoints[$i]['speed_0'] == $keyPoints[$i]['speed_1'] &&
        ↪ $keyPoints[$i]['speed_0'] == $keyPoints[$i + 1]['speed_0'] &&
        ↪ $keyPoints[$i]['speed_0'] == $keyPoints[$i + 1]['speed_1'])) {
254         array_push($localKeyPointsTwo, $keyPoints[$i]);
255     } else {
256         $i++;
257     }
258 } else {
259     array_push($localKeyPointsTwo, $keyPoints[$i]);
260 }
261 }
262
263 // Berechnung der Echtzeitdaten nach der Neukalibrierung
264 $keyPoints = $localKeyPointsTwo;
265 $trainChange = createTrainChanges(true);
266 $trainPositionChange = $trainChange[0];
267 $trainSpeedChange = $trainChange[1];
268 }
269
270 // Fügt die aktuelle Zeit zum ersten $keyPoint hinzu
271 $keyPoints[0]['time_0'] = $startTime;
272 $keyPoints = deleteDoubledKeyPoints($keyPoints);
273 $keyPoints = calculateTimeFromKeyPoints();
274
275 if ($useMinTimeOnSpeed && $minTimeOnSpeedIsPossible) {
276     array_push($speedOverPositionAllIterations, array($trainPositionChange,
        ↪ $trainSpeedChange));
277     toShortOnOneSpeed();
278 }
279
280 // Ermittlung der Echtzeitdaten
281 $trainChange = createTrainChanges(true);
282 $trainPositionChange = $trainChange[0];
283 $trainSpeedChange = $trainChange[1];
284 $timeToNextStop = end($keyPoints)['time_1'] - $keyPoints[0]['time_0'];
285
286 // Überprüfung, ob das Fahrzeug mit einer Verspätung am Ziel ankommt.
287 // Fahrzeuge, die ohne Fahrplan fahren, werden nicht betrachtet.
288 if (!$freieFahrt) {

```

```

289 if ($timeToNextStop > $maxTimeToNextStop) {
290     echo 'Der Zug mit der Adresse ', $train['adresse'], ' wird mit einer
        ↳ Verspätung von ', number_format($timeToNextStop -
        ↳ $maxTimeToNextStop, 2), ' Sekunden im nächsten planmäßigen Halt (',
        ↳ $targetBetriebsstelle,") ankommen.\n";
291 } else {
292     echo 'Aktuell benötigt der Zug mit der Adresse ', $train['adresse'], ' ',
        ↳ number_format($timeToNextStop, 2), ' Sekunden, obwohl er ',
        ↳ number_format($maxTimeToNextStop, 2), " Sekunden zur Verfügung hat
        ↳ .\n";
293
294 if ($slowDownIfTooEarly) {
295     echo 'Evtl. könnte der Zug zwischendurch die Geschwindigkeit verringern,
        ↳ um Energie zu sparen.';
296
297     array_push($speedOverPositionAllIterations, array($trainPositionChange,
        ↳ $trainSpeedChange));
298     $keyPointsPreviousStep = array();
299     $finish = false;
300     $possibleSpeedRange = null;
301     $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
302
303     while ($returnSpeedDecrease['possible'] && !$finish) {
304         $possibleSpeedRange = findMaxSpeed($returnSpeedDecrease);
305
306         if ($possibleSpeedRange['min_speed'] == $possibleSpeedRange['max_speed
            ↳ ']) {
307             break;
308         }
309
310         $localKeyPoints = $keyPoints;
311         $newCalculatedTime = null;
312         $newKeyPoints = null;
313
314         for ($i = $possibleSpeedRange['max_speed']; $i >= $possibleSpeedRange[
            ↳ 'min_speed']; $i = $i - 10) {
315             $localKeyPoints[$possibleSpeedRange['first_key_point_index']][
                ↳ 'speed_1'] = $i;

```

```

316     $localKeyPoints[$possibleSpeedRange['first_key_point_index'] + 1]['
        ↳ speed_0'] = $i;
317     $localKeyPoints[$possibleSpeedRange['first_key_point_index']][
        ↳ position_1'] = (getBrakeDistance($localKeyPoints[
        ↳ $possibleSpeedRange['first_key_point_index']][
        ↳ 'speed_0'], $i,
        ↳ $verzoegerung) + $localKeyPoints[$possibleSpeedRange[
        ↳ first_key_point_index']][
        ↳ 'position_0']);
318     $localKeyPoints[$possibleSpeedRange['first_key_point_index'] + 1][
        ↳ position_0'] = ($localKeyPoints[$possibleSpeedRange[
        ↳ first_key_point_index'] + 1][
        ↳ 'position_1'] - getBrakeDistance
        ↳ ($i, $localKeyPoints[$possibleSpeedRange[
        ↳ first_key_point_index'] + 1][
        ↳ 'speed_1'], $verzoegerung));
319     $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
320     $newCalculatedTime = $localKeyPoints[array_key_last($localKeyPoints)
        ↳ ][
        ↳ 'time_1'];
321
322     if ($i == 10) {
323         if ($newCalculatedTime > $maxTimeToNextStop) {
324             $localKeyPoints[$possibleSpeedRange['first_key_point_index']][
        ↳ speed_1'] = $i + 10;
325             $localKeyPoints[$possibleSpeedRange['first_key_point_index'] +
        ↳ 1][
        ↳ 'speed_0'] = $i + 10;
326             $localKeyPoints[$possibleSpeedRange['first_key_point_index']][
        ↳ position_1'] = (getBrakeDistance($localKeyPoints[
        ↳ $possibleSpeedRange['first_key_point_index']][
        ↳ 'speed_0'],
        ↳ ($i + 10), $verzoegerung) + $localKeyPoints[
        ↳ $possibleSpeedRange['first_key_point_index']][
        ↳ 'position_0'
        ↳ ]);
327             $localKeyPoints[$possibleSpeedRange['first_key_point_index'] +
        ↳ 1][
        ↳ 'position_0'] = ($localKeyPoints[$possibleSpeedRange[
        ↳ first_key_point_index'] + 1][
        ↳ 'position_1'] -
        ↳ getBrakeDistance(($i + 10), $localKeyPoints[
        ↳ $possibleSpeedRange['first_key_point_index'] + 1][
        ↳ 'speed_1
        ↳ '], $verzoegerung));
328         }
329
330         $finish = true;
331         $newKeyPoints = $localKeyPoints;
332         break;

```

```

333     }
334     if (($newCalculatedTime - $startTime) > $maxTimeToNextStop) {
335         if ($i == $possibleSpeedRange['max_speed']) {
336             $localKeyPoints = $keyPointsPreviousStep;
337             $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
338             $keyPoints = $localKeyPoints;
339             $finish = true;
340             break;
341         }
342         $localKeyPoints[$possibleSpeedRange['first_key_point_index']]['speed_1'] = $i + 10;
343         $localKeyPoints[$possibleSpeedRange['first_key_point_index'] + 1]['speed_0'] = $i + 10;
344         $localKeyPoints[$possibleSpeedRange['first_key_point_index']]['position_1'] = (getBrakeDistance($localKeyPoints[
            ↪ $possibleSpeedRange['first_key_point_index']]['speed_0'], (
            ↪ $i + 10), $verzoeigerung) + $localKeyPoints[
            ↪ $possibleSpeedRange['first_key_point_index']]['position_0'
            ↪ ]);
345         $localKeyPoints[$possibleSpeedRange['first_key_point_index'] + 1]['position_0'] = ($localKeyPoints[$possibleSpeedRange['
            ↪ first_key_point_index'] + 1]['position_1'] -
            ↪ getBrakeDistance(($i + 10), $localKeyPoints[
            ↪ $possibleSpeedRange['first_key_point_index'] + 1]['speed_1'
            ↪ ], $verzoeigerung));
346         $newKeyPoints = $localKeyPoints;
347         $finish = true;
348         $keyPoints = $localKeyPoints;
349
350         break;
351     }
352     if ($i == $possibleSpeedRange['min_speed']) {
353         $newKeyPoints = $localKeyPoints;
354         $newKeyPoints = deleteDoubledKeyPoints($newKeyPoints);
355         $keyPoints = $newKeyPoints;
356         break;
357     }
358     $newKeyPoints = $localKeyPoints;
359 }

```

```

360     $keyPointsPreviousStep = $localKeyPoints;
361
362     if ($newKeyPoints != null) {
363         $keyPoints = $newKeyPoints;
364     }
365
366     $keyPoints = deleteDoubledKeyPoints($keyPoints);
367     $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
368 }
369
370 $keyPoints = calculateTimeFromKeyPoints();
371
372 if ($useSpeedFineTuning && $returnSpeedDecrease['possible']) {
373     $trainChangeReturn = createTrainChanges(true);
374     $trainPositionChange = $trainChangeReturn[0];
375     $trainSpeedChange = $trainChangeReturn[1];
376     $newCalculatedTime = $keyPoints[array_key_last($keyPoints)]['time_1'];
377     speedFineTuning(($maxTimeToNextStop - ($newCalculatedTime - $startTime
        ↳ )), $returnSpeedDecrease['range'][array_key_last(
        ↳ $returnSpeedDecrease['range'])]['KeyPoint_index']);
378 }
379
380 $keyPoints = calculateTimeFromKeyPoints();
381 $timeToNextStop = end($keyPoints)['time_1'] - $keyPoints[0]['time_0'];
382
383 echo "\nDurch die Anpassung der Geschwindigkeit benötigt der Zug mit der
    ↳ Adresse ", $train['adresse'], ' jetzt ', number_format(
    ↳ $timeToNextStop, 2), " Sekunden bis\n";
384
385 if (abs($timeToNextStop - $maxTimeToNextStop) <
    ↳ $globalFloatingPointNumbersRoundingError) {
386     echo 'zum nächsten planmäßigen Halt (', $targetBetriebsstelle, ") und
        ↳ wird diesen genau pünktlich erreichen.\n";
387 } else if (($timeToNextStop - $maxTimeToNextStop) > 0) {
388     echo 'zum nächsten planmäßigen Halt (', $targetBetriebsstelle, ') und
        ↳ wird diesen mit einer Verspätung von ', number_format(
        ↳ $timeToNextStop - $maxTimeToNextStop, 2), " Sekunden erreichen
        ↳ .\n";
389 } else {

```



```

390     echo 'zum nächsten planmäßigen Halt (', $targetBetriebsstelle, ') und
        ↳ wird diesen ', number_format($timeToNextStop -
        ↳ $maxTimeToNextStop, 2), " Sekunden zu früh erreichen.\n";
391 }
392 } else {
393     echo "Dadurch, dass \$slowDownIfTooEarly = true ist, wird das Fahrzeug "
        ↳ , number_format($maxTimeToNextStop - $timeToNextStop, 2), '
        ↳ Sekunden zu früh am Ziel ankommen.';
394 }
395 }
396 } else {
397     echo 'Der Zug mit der Adresse ', $train['adresse'], ' fährt aktuell ohne
        ↳ Fahrplan bis zum nächsten auf Halt stehendem Signal (Signal ID: ',
        ↳ $indexReachedBetriebsstelle, ', Betriebsstelle: ',
        ↳ $targetBetriebsstelle, ").\n";
398 }
399
400 // Berechnung der Echtzeitdaten
401 $returnTrainChanges = createTrainChanges(false);
402 $trainPositionChange = $returnTrainChanges[0];
403 $trainSpeedChange = $returnTrainChanges[1];
404 $trainTimeChange = $returnTrainChanges[2];
405 $trainRelativePosition = $returnTrainChanges[3];
406 $trainSection = $returnTrainChanges[4];
407 $trainIsSpeedChange = $returnTrainChanges[5];
408 $trainTargetReached = array();
409 $trainBetriebsstelleName = array();
410 $trainWendet = array();
411 $allReachedTargets = array();
412 $allreachedInfrasIndex = array();
413 $allreachedInfrasID = array();
414 $allreachedInfrasUsed = array();
415
416 foreach ($allreachedInfras as $value) {
417     array_push($allreachedInfrasIndex, $value['index']);
418     array_push($allreachedInfrasID, $value['infra']);
419 }
420
421 foreach ($trainPositionChange as $key => $value) {

```

```

422     $trainBetriebsstelleName[$key] = $targetBetriebsstelle;
423     if (array_key_last($trainPositionChange) != $key) {
424         $trainTargetReached[$key] = false;
425         $trainWendet[$key] = false;
426     } else {
427         if ($wendet) {
428             $trainWendet[$key] = true;
429         } else {
430             $trainWendet[$key] = false;
431         }
432
433         if ($reachedBetriebsstelle) {
434             $trainTargetReached[$key] = true;
435         } else {
436             $trainTargetReached[$key] = false;
437         }
438     }
439 }
440
441 for($i = sizeof($trainSection) - 1; $i >= 0; $i--) {
442     if (in_array($trainSection[$i], $allreachedInfrasID) && !in_array(
443         ↪ $trainSection[$i], $allreachedInfrasUsed)) {
444         array_push($allreachedInfrasUsed, $trainSection[$i]);
445         $Infracindex = array_search($trainSection[$i], $allreachedInfrasID);
446         $allReachedTargets[$i] = $allreachedInfrasIndex[$Infracindex];
447     } else {
448         $allReachedTargets[$i] = null;
449     }
450 }
451 ksort($allReachedTargets);
452 $returnArray = array();
453 $address = $train['adresse'];
454 $trainID = array();
455 $id = $train['id'];
456
457 foreach ($trainPositionChange as $key => $value) {
458     $trainID[$key] = $id;
459 }

```

```

460 foreach ($trainPositionChange as $trainPositionChangeIndex =>
    ↪ $trainPositionChangeValue) {
461     array_push($returnArray, array('live_position' => $trainPositionChangeValue
    ↪ ,
462     'live_speed' => $trainSpeedChange[$trainPositionChangeIndex],
463     'live_time' => $trainTimeChange[$trainPositionChangeIndex],
464     'live_relative_position' => $trainRelativePosition[
    ↪ $trainPositionChangeIndex],
465     'live_section' => $trainSection[$trainPositionChangeIndex],
466     'live_is_speed_change' => $trainIsSpeedChange[$trainPositionChangeIndex],
467     'live_target_reached' => $trainTargetReached[$trainPositionChangeIndex],
468     'id' => $trainID[$trainPositionChangeIndex],
469     'wendet' => $trainWendet[$trainPositionChangeIndex],
470     'betriebsstelle' => $trainBetriebsstelleName[$trainPositionChangeIndex],
471     'live_all_targets_reached' => $allReachedTargets[
    ↪ $trainPositionChangeIndex]));
472 }
473
474 $allTimes[$adress] = $returnArray;
475 safeTrainChangeToJSONFile($indexCurrentSection, $indexTargetSection,
    ↪ $indexCurrentSectionMod, $indexTargetSectionMod,
    ↪ $speedOverPositionAllIterations);
476
477 return (end($trainTimeChange) - $trainTimeChange[0]) - ($endTime - $startTime
    ↪ );
478 }
479
480 // Ermittelt die maximale Geschwindigkeit zwischen zwei Punkten
481 function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1, int $id)
    ↪ {
482
483     global $verzoeigerung;
484     global $globalFloatingPointNumbersRoundingError;
485
486     $v_max = array();
487
488     for ($i = 0; $i <= 120; $i = $i + 10) {
489         if ((getBrakeDistance($v_0, $i, $verzoeigerung) + getBrakeDistance($i, $v_1,
            ↪ $verzoeigerung)) < ($distance +

```

```

        ↪ $globalFloatingPointNumbersRoundingError)) {
490     array_push($v_max, $i);
491 }
492 }
493
494 if (sizeof($v_max) == 0) {
495     if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
496         echo 'Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu
        ↪ erreichen.';
497     } else {
498         emergencyBreak($id);
499     }
500 } else {
501     if ($v_0 == $v_1 && max($v_max) < $v_0) {
502         $v_max = array($v_0);
503     }
504 }
505
506 return max($v_max);
507 }
508
509 // Erstellt einen $keyPoint
510 function createKeyPoint (float $position_0, float $position_1, int $speed_0,
        ↪ int $speed_1) {
511     return array('position_0' => $position_0, 'position_1' => $position_1, '
        ↪ speed_0' => $speed_0, 'speed_1' => $speed_1);
512 }
513
514 // Ermittelt aus den $keyPoint die Echtzeitdaten
515 // (nur Geschwindigkeit und Position)
516 function convertKeyPointsToTrainChangeArray (array $keyPoints) {
517
518     global $verzoeigerung;
519
520     $trainSpeedChangeReturn = array();
521     $trainPositionChnageReturn = array();
522     array_push($trainPositionChnageReturn, $keyPoints[0]['position_0']);
523     array_push($trainSpeedChangeReturn, $keyPoints[0]['speed_0']);
524

```

```

525 for ($i = 0; $i <= (sizeof($keyPoints) - 2); $i++) {
526     if ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1']) {
527         for ($j = $keyPoints[$i]['speed_0']; $j < $keyPoints[$i]['speed_1']; $j =
528             ↪ $j + 2) {
529             array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
530                 ↪ + getBrakeDistance($j, ($j + 2), $verzoegerung)));
531             array_push($trainSpeedChangeReturn, ($j + 2));
532         }
533     } elseif ($keyPoints[$i]['speed_0'] > $keyPoints[$i]['speed_1']) {
534         for ($j = $keyPoints[$i]['speed_0']; $j > $keyPoints[$i]['speed_1']; $j =
535             ↪ $j - 2) {
536             array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn)
537                 ↪ + getBrakeDistance($j, ($j - 2), $verzoegerung)));
538             array_push($trainSpeedChangeReturn, ($j - 2));
539         }
540     }
541     array_push($trainPositionChnageReturn, $keyPoints[$i + 1]['position_0']);
542     array_push($trainSpeedChangeReturn, $keyPoints[$i + 1]['speed_0']);
543 }
544
545 if (end($keyPoints)['speed_0'] < end($keyPoints)['speed_1']) {
546     for ($j = end($keyPoints)['speed_0']; $j < end($keyPoints)['speed_1']; $j =
547         ↪ $j + 2) {
548         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
549             ↪ getBrakeDistance($j, ($j + 2), $verzoegerung)));
550         array_push($trainSpeedChangeReturn, ($j + 2));
551     }
552 } else if (end($keyPoints)['speed_0'] > end($keyPoints)['speed_1']) {
553     for ($j = end($keyPoints)['speed_0']; $j > end($keyPoints)['speed_1']; $j =
554         ↪ $j - 2) {
555         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
556             ↪ getBrakeDistance($j, ($j - 2), $verzoegerung)));
557         array_push($trainSpeedChangeReturn, ($j - 2));
558     }
559 }
560
561 return array($trainPositionChnageReturn, $trainSpeedChangeReturn);
562 }

```

```

556 // Wandelt die Daten der Infra-Abschnitte und der Iterationsschritte der
557 // Fahrtverlaufsrechnung in JSON-Dateien um, damit die Fahrtverläufe
558 // visuell dargestellt werden können.
559 function safeTrainChangeToJSONFile(int $indexCurrentSection, int
    ↳ $indexTargetSection, int $indexCurrentSectionMod, int
    ↳ $indexTargetSectionMod, array $speedOverPositionAllIterations) {
560
561     global $trainPositionChange;
562     global $trainSpeedChange;
563     global $next_v_max;
564     global $cumulativeSectionLengthEnd;
565     global $next_v_max_mod;
566     global $cumulativeSectionLengthEndMod;
567
568     $speedOverPosition = array_map('toArr', $trainPositionChange,
    ↳ $trainSpeedChange);
569     $speedOverPosition = json_encode($speedOverPosition);
570     $fp = fopen('../json/speedOverPosition.json', 'w');
571     fwrite($fp, $speedOverPosition);
572     fclose($fp);
573
574     $v_maxFromUsedSections = array();
575
576     for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
577         array_push($v_maxFromUsedSections, $next_v_max[$i]);
578     }
579
580     $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
    ↳ $v_maxFromUsedSections);
581     $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSections);
582     $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
583     fwrite($fp, $VMaxOverPositionsJSon);
584     fclose($fp);
585
586     $v_maxFromUsedSections = array();
587
588     for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
589         array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
590     }

```

```

591
592 $VMaxOverCumulativeSectionsMod = array_map('toArr',
    ↪ $cumulativeSectionLengthEndMod, $v_maxFromUsedSections);
593 $VMaxOverPositionsJSoN = json_encode($VMaxOverCumulativeSectionsMod);
594 $fp = fopen('../json/VMaxOverCumulativeSectionsMod.json', 'w');
595 fwrite($fp, $VMaxOverPositionsJSoN);
596 fclose($fp);
597
598 $jsonReturn = array();
599
600 for ($i = 0; $i < sizeof($speedOverPositionAllIterations); $i++) {
601     $iteration = array_map('toArr', $speedOverPositionAllIterations[$i][0],
    ↪ $speedOverPositionAllIterations[$i][1]);
602     array_push($jsonReturn, $iteration);
603 }
604
605 $speedOverPosition = json_encode($jsonReturn);
606 $fp = fopen('../json/speedOverPosition_prevIterations.json', 'w');
607 fwrite($fp, $speedOverPosition);
608 fclose($fp);
609 }
610
611 // Überprüft, ob das Fahrzeug in Infra-Abschnitten die zulässige
612 // Höchstgeschwindigkeit überschreitet
613 function checkIfTrainIsToFastInCertainSections() {
614
615     global $trainPositionChange;
616     global $trainSpeedChange;
617     global $cumulativeSectionLengthStartMod;
618     global $next_v_max_mod;
619     global $indexTargetSectionMod;
620
621     $faillSections = array();
622
623     foreach ($trainPositionChange as $trainPositionChangeKey =>
    ↪ $trainPositionChangeValue) {
624         foreach ($cumulativeSectionLengthStartMod as
    ↪ $cumulativeSectionLengthStartKey =>
    ↪ $cumulativeSectionLengthStartValue) {

```

```

625     if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
626         if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
            ↳ $cumulativeSectionLengthStartKey - 1]) {
627             array_push($faieldSections, ($cumulativeSectionLengthStartKey - 1));
628         }
629
630         break;
631     } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
632         if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
633             if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                ↳ $cumulativeSectionLengthStartKey]) {
634                 array_push($faieldSections, $cumulativeSectionLengthStartKey);
635             }
636
637             break;
638         }
639     }
640 }
641 }
642
643 if (sizeof($faieldSections) == 0) {
644     return array('failed' => false);
645 } else {
646     return array('failed' => true, 'failed_sections' => array_unique(
        ↳ $faieldSections));
647 }
648 }
649
650 // Löscht $keyPoint, bei denen Start- und Zielgeschwindigkeit identisch ist.
651 // Der erste $keyPoint wird dabei nicht betrachtet.
652 function deleteDoubledKeyPoints($temporaryKeyPoints) {
653     do {
654         $foundDoubledKeyPoints = false;
655         $doubledIndex = array();
656
657         for ($i = 1; $i < (sizeof($temporaryKeyPoints) - 1); $i++) {
658             if ($temporaryKeyPoints[$i]['speed_0'] == $temporaryKeyPoints[$i][
                ↳ speed_1']) {
659                 $foundDoubledKeyPoints = true;

```



```

660     array_push($doubledIndex, $i);
661 }
662 }
663
664 foreach ($doubledIndex as $index) {
665     unset($temporaryKeyPoints[$index]);
666 }
667
668 $temporaryKeyPoints = array_values($temporaryKeyPoints);
669 } while ($foundDoubledKeyPoints);
670
671 return $temporaryKeyPoints;
672 }
673
674 // Ermittelt die Zeiten der $keyPoint ausgehend vom ersten $keyPoint. Mit dem
675 // Parameter $inputKeyPoints können $keyPoints übergeben werden, bei den die
676 // Zeit ermittelt wird. Wenn keine $keyPoints übergeben werden, werden die
677 // globalen $keyPoints verwendet. Mit dem Parameter $skippingKeys können
678 // $keyPoints übersprungen werden. Das auslassen von $keyPoints ist für die
679 // Funktion postponeSubsection() relevant.
680 function calculateTimeFromKeyPoints($inputKeyPoints = null, $skippingKeys =
    ↪ null) {
681
682     global $keyPoints;
683     global $verzoegerung;
684
685     if ($inputKeyPoints == null) {
686         $localKeyPoints = $keyPoints;
687     } else {
688         $localKeyPoints = $inputKeyPoints;
689     }
690
691     $keys = array_keys($localKeyPoints);
692
693     if ($skippingKeys != null) {
694         foreach ($skippingKeys as $skip) {
695             unset($keys[array_search($skip, $keys)]);
696         }
697     }

```

```

698
699     $keys = array_values($keys);
700
701     for ($i = 0; $i < (sizeof($keys) - 1); $i++) {
702         $localKeyPoints[$keys[$i]]['time_1'] = getBrakeTime($localKeyPoints[$keys[
            ↳ $i]]['speed_0'], $localKeyPoints[$keys[$i]]['speed_1'],
            ↳ $verzoeigerung) + $localKeyPoints[$keys[$i]]['time_0'];
703         $localKeyPoints[$keys[$i] + 1]['time_0'] = distanceWithSpeedToTime(
            ↳ $localKeyPoints[$keys[$i]]['speed_1'], ($localKeyPoints[$keys[$i] +
            ↳ 1]['position_0']) - $localKeyPoints[$keys[$i]]['position_1']) +
            ↳ $localKeyPoints[$keys[$i]]['time_1'];
704     }
705
706     $localKeyPoints[end($keys)]['time_1'] = getBrakeTime($localKeyPoints[end(
            ↳ $keys)]['speed_0'], $localKeyPoints[end($keys)]['speed_1'],
            ↳ $verzoeigerung) + $localKeyPoints[end($keys)]['time_0'];
707
708     return $localKeyPoints;
709 }
710
711 // Echtzeitdatenermittlung eines Fahrtverlaufs auf Grundlage der $keyPoints.
712 // Mit dem Parameter $onlyPositionAndSpeed kann festgelegt werden, ob nur die
713 // Position und Geschwindigkeit berechnet werden soll.
714 function createTrainChanges(bool $onlyPositionAndSpeed) {
715
716     global $keyPoints;
717     global $verzoeigerung;
718     global $cumulativeSectionLengthStart;
719     global $cumulativeSectionLengthEnd;
720     global $next_sections;
721     global $indexCurrentSection;
722     global $indexTargetSection;
723     global $currentPosition;
724     global $globalFloatingPointNumbersRoundingError;
725     global $globalDistanceUpdateInterval;
726
727     $returnTrainSpeedChange = array();
728     $returnTrainTimeChange = array();
729     $returnTrainPositionChange = array();

```

```

730 $returnTrainRelativePosition = array();
731 $returnTrainSection = array();
732 $returnIsSpeedChange = array();
733
734 // Ermittelt für alle bis auf den letzten $keyPoint die Echtzeitdaten der
735 // Zeit, Geschwindigkeit und Position
736 for ($i = 0; $i < sizeof($keyPoints); $i++) {
737     array_push($returnTrainTimeChange, $keyPoints[$i]['time_0']);
738     array_push($returnTrainSpeedChange, $keyPoints[$i]['speed_0']);
739     array_push($returnTrainPositionChange, $keyPoints[$i]['position_0']);
740     array_push($returnIsSpeedChange, true);
741
742     $itDir = ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1']) ? 2 : -2;
743
744     for ($j = ($keyPoints[$i]['speed_0'] + $itDir); $j <= $keyPoints[$i]['
        ↳ speed_1']; $j = $j + $itDir) {
745         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
        ↳ getBrakeDistance(($j - $itDir), $j, $verzoeigerung)));
746         array_push($returnTrainSpeedChange, $j);
747         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (
        ↳ getBrakeTime(($j - $itDir), $j, $verzoeigerung))));
748         array_push($returnIsSpeedChange, true);
749     }
750
751 // Überprüft, ob nach dem $keyPoint eine Beharrungsfahrt stattfindet
752 if ($i != array_key_last($keyPoints)) {
753     // Ermittelt für die Strecke zwischen zwei $keyPoints die Echtzeitdaten
754     // der Zeit, Geschwindigkeit und Position
755     $startPosition = $keyPoints[$i]['position_1'];
756     $endPosition = $keyPoints[$i + 1]['position_0'];
757     $speedToNextKeyPoint = $keyPoints[$i]['speed_1'];
758     $timeForOneTimeInterval = distanceWithSpeedToTime($speedToNextKeyPoint,
        ↳ $globalDistanceUpdateInterval);
759
760     for ($position = $startPosition + $globalDistanceUpdateInterval;
        ↳ $position < $endPosition; $position = $position +
        ↳ $globalDistanceUpdateInterval) {
761         array_push($returnTrainPositionChange, $position);
762         array_push($returnTrainSpeedChange, $speedToNextKeyPoint);

```

```

763     array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
        ↳ $timeForOneTimeInterval);
764     array_push($returnIsSpeedChange, false);
765 }
766 }
767 }
768 array_push($returnTrainPositionChange, $keyPoints[array_key_last($keyPoints)
        ↳ ][ 'position_1' ] - getBrakeDistance($keyPoints[array_key_last(
        ↳ $keyPoints)][ 'speed_0' ], $keyPoints[array_key_last($keyPoints)][ '
        ↳ speed_1' ], $verzoegerung));
769 array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)][ '
        ↳ speed_0' ]);
770 array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)][ '
        ↳ time_0' ]);
771 array_push($returnIsSpeedChange, true);
772
773 if ($onlyPositionAndSpeed) {
774     return array($returnTrainPositionChange, $returnTrainSpeedChange);
775 } else {
776     // Ermittelt die relativen Positionen innerhalb der Infra-Abschnitte
777     // zu den absoluten Positionen
778     foreach ($returnTrainPositionChange as $absolutPositionKey =>
        ↳ $absolutPositionValue) {
779         foreach ($cumulativeSectionLengthStart as $sectionStartKey =>
            ↳ $sectionStartValue) {
780             if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue
                ↳ < $cumulativeSectionLengthEnd[$sectionStartKey]) {
781                 if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
                    ↳ $indexTargetSection) {
782                     $returnTrainRelativePosition[$absolutPositionKey] =
                        ↳ $absolutPositionValue + $currentPosition;
783                     $returnTrainSection[$absolutPositionKey] = $next_sections[
                        ↳ $sectionStartKey];
784                 } else if ($sectionStartKey == $indexCurrentSection) {
785                     $returnTrainRelativePosition[$absolutPositionKey] =
                        ↳ $absolutPositionValue + $currentPosition;
786                     $returnTrainSection[$absolutPositionKey] = $next_sections[
                        ↳ $sectionStartKey];
787                 } else if ($sectionStartKey == $indexTargetSection) {

```

```

788     $returnTrainRelativePosition[$absolutPositionKey] =
789         ↳ $absolutPositionValue - $sectionStartValue;
790     $returnTrainSection[$absolutPositionKey] = $next_sections[
791         ↳ $sectionStartKey];
792 } else {
793     $returnTrainRelativePosition[$absolutPositionKey] =
794         ↳ $absolutPositionValue - $sectionStartValue;
795     $returnTrainSection[$absolutPositionKey] = $next_sections[
796         ↳ $sectionStartKey];
797 }
798 break;
799 } else if ($absolutPositionKey == array_key_last(
800     ↳ $returnTrainPositionChange) && abs($absolutPositionValue -
801     ↳ floatval($cumulativeSectionLengthEnd[$sectionStartKey])) <
802     ↳ $globalFloatingPointNumbersRoundingError) {
803     $returnTrainRelativePosition[$absolutPositionKey] =
804         ↳ $cumulativeSectionLengthEnd[$sectionStartKey] -
805         ↳ $sectionStartValue;
806     $returnTrainSection[$absolutPositionKey] = $next_sections[
807         ↳ $sectionStartKey];
808     break;
809 } else {
810     debugMessage('Einer absoluten Position konnte kein Infra-Abschnitt und
811         ↳ keine relative Position in einem Infra-Abschnitt zugeordnet
812         ↳ werden.');
```

```

813 $returnKeyPoints = array();
814 $numberOfPairs = sizeof($tempKeyPoints) / 2;
815
816 for($j = 0; $j < $numberOfPairs; $j++) {
817     $i = $j * 2;
818     $return = checkBetweenTwoKeyPoints($tempKeyPoints, $i, $id);
819
820     foreach ($return as $keyPoint) {
821         array_push($returnKeyPoints, $keyPoint);
822     }
823 }
824
825 return $returnKeyPoints;
826 }
827
828 // Überprüft, ob zwischen zwei $keyPoints die zulässige Höchstgeschwindigkeit
829 // überschritten wird
830 function checkBetweenTwoKeyPoints(array $temKeyPoints, int $keyPointIndex, int
    ↪ $id) {
831
832     global $trainPositionChange;
833     global $trainSpeedChange;
834     global $cumulativeSectionLengthStartMod;
835     global $cumulativeSectionLengthEndMod;
836     global $next_v_max_mod;
837     global $verzoegerung;
838     global $indexTargetSectionMod;
839
840     $failedSections = array();
841     $groupedFailedSections = array();
842     $returnKeyPoints = array();
843     $failedPositions = array();
844     $failedSpeeds = array();
845
846     foreach ($trainPositionChange as $trainPositionChangeKey =>
    ↪ $trainPositionChangeValue) {
847         if ($trainPositionChangeValue >= $temKeyPoints[$keyPointIndex]['position_0'
    ↪ ] && $trainPositionChangeValue <= $temKeyPoints[$keyPointIndex + 1][
    ↪ 'position_1']) {

```

```

848     foreach ($cumulativeSectionLengthStartMod as
      ↳ $cumulativeSectionLengthStartKey =>
      ↳ $cumulativeSectionLengthStartValue) {
849         if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
850             if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
              ↳ $cumulativeSectionLengthStartKey - 1]) {
851                 array_push($failedSections, ($cumulativeSectionLengthStartKey - 1));
852                 array_push($failedSpeeds, $trainSpeedChange[$trainPositionChangeKey
                  ↳ ]);
853                 $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                  ↳ $trainPositionChangeKey];
854             }
855             break;
856         } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
857             if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
858                 if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                  ↳ $cumulativeSectionLengthStartKey]) {
859                     array_push($failedSections, $cumulativeSectionLengthStartKey);
860                     array_push($failedSpeeds, $trainSpeedChange[
                        ↳ $trainPositionChangeKey]);
861                     $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                        ↳ $trainPositionChangeKey];
862                 }
863                 break;
864             }
865         }
866     }
867 }
868 }
869
870 // Alle Infra_abschnitte zwischen denn beiden KeyPoints, bei denen die
871 // zulässige Höchstgeschwindigkeit überschritten wird
872 $failedSections = array_unique($failedSections);
873
874 // Wenn es kein Fehler gibt, werden die beiden KeyPoints zurückgegeben und
875 // wen es einen Fehler gibt, wird der erste der beiden KeyPoints im
876 // $returnKeyPoints gespeichert
877 if (sizeof($failedSections) == 0) {

```

```

878     return array($stemKeyPoints[$keyPointIndex], $stemKeyPoints[$keyPointIndex +
      ↪ 1]);
879 } else {
880     $returnKeyPoints[0]['speed_0'] = $stemKeyPoints[$keyPointIndex]['speed_0'];
881     $returnKeyPoints[0]['position_0'] = $stemKeyPoints[$keyPointIndex]['
      ↪ position_0'];
882 }
883
884 // Einteilung der benachbarten failedSections in zusammenhängende Gruppen
885 $previous = NULL;
886 $index = 0;
887 foreach($failedSections as $key => $value) {
888     if($value > $previous + 1) {
889         $index++;
890     }
891     $groupedFailedSections[$index][] = $value;
892     $previous = $value;
893 }
894
895 // Iteration über die zusammenhängenden $failedSections
896 foreach ($groupedFailedSections as $groupSectionsIndex => $groupSectionsValue
      ↪ ) {
897     $firstFailedPositionIndex = null;
898     $lastFailedPositionIndex = null;
899     $firstFailedPosition = null;
900     $lastFailedPosition = null;
901     $lastElement = array_key_last($returnKeyPoints);
902     $failedSection = null;
903
904     // Ermittlung der Section mit der kleinsten v_max von allen $failedSections
905     // in der Gruppe
906     if (sizeof($groupSectionsValue) == 1) {
907         $failedSection = $groupSectionsValue[0];
908     } else {
909         $slowestSpeed = 200;
910         for ($i = 0; $i <= (sizeof($groupSectionsValue) - 1); $i++) {
911             if ($next_v_max_mod[$groupSectionsValue[$i]] < $slowestSpeed) {
912                 $slowestSpeed = $next_v_max_mod[$groupSectionsValue[$i]];
913                 $failedSection = $groupSectionsValue[$i];

```



```

914     }
915 }
916 }
917
918 // Start- und Endposition der $failedSection
919 $failedSectionStart = $cumulativeSectionLengthStartMod[$failedSection];
920 $failedSectionEnd = $cumulativeSectionLengthEndMod[$failedSection];
921
922 // Bestimmung der ersten und letzten Position, in der es in der
923     ↳ $failedSection
924 // zu einer Geschwindigkeitsüberschreitung kommt
925 foreach ($failedPositions as $failPositionIndex => $failPositionValue) {
926     if ($failPositionValue > $failedSectionStart && $failPositionValue <
927         ↳ $failedSectionEnd) {
928         if ($firstFailedPositionIndex == null) {
929             $firstFailedPositionIndex = $failPositionIndex;
930         }
931         $lastFailedPositionIndex = $failPositionIndex;
932     }
933 }
934
935 // Bestimmung des letzten Punktes, bei dem die Geschwindigkeit noch
936 // nicht zu schnell war
937 //
938 // Wenn der Punkt davor außerhalb der failedSection liegt
939 // => Startpunkt = Anfang der Section
940 // Wenn der Punkt davor innerhalb der failed Section liegt
941 // => Startpunkt = der Punkt davor
942 if ($firstFailedPositionIndex != 0) {
943     if ($trainPositionChange[$firstFailedPositionIndex - 1] <
944         ↳ $failedSectionStart) {
945         $firstFailedPosition = $failedSectionStart;
946     } else {
947         $firstFailedPosition = $trainPositionChange[$firstFailedPositionIndex -
948             ↳ 1];
949     }
950 } else {
951     $firstFailedPosition = $failedSectionStart;
952 }

```

```

949
950 // Bestimmung der ersten Position, bei dem die Geschwindigkeit nicht
951 // mehr zu hoch war.
952 if ($lastFailedPositionIndex != array_key_last($trainPositionChange)) {
953     if ($trainPositionChange[$lastFailedPositionIndex + 1] >
954         ↪ $failedSectionEnd) {
955         $lastFailedPosition = $failedSectionEnd;
956     } else {
957         $lastFailedPosition = $trainPositionChange[$lastFailedPositionIndex +
958             ↪ 1];
959     }
960 } else {
961     $lastFailedPosition = $failedSectionEnd;
962 }
963
964 $returnKeyPoints[$lastElement + 1]['position_1'] = $firstFailedPosition;
965 $returnKeyPoints[$lastElement + 1]['speed_1'] = $next_v_max_mod[
966     ↪ $failedSection];
967
968 $returnKeyPoints[$lastElement + 2]['position_0'] = $lastFailedPosition;
969 $returnKeyPoints[$lastElement + 2]['speed_0'] = $next_v_max_mod[
970     ↪ $failedSection];
971 }
972
973 // Zielwerte des letzten $keyPoint vom zweiten $keyPoint übernehmen
974 $returnKeyPoints[array_key_last($returnKeyPoints) + 1]['position_1'] =
975     ↪ $temKeyPoints[$keyPointIndex + 1]['position_1'];
976 $returnKeyPoints[array_key_last($returnKeyPoints)]['speed_1'] = $temKeyPoints
977     ↪ [$keyPointIndex + 1]['speed_1'];
978 $numberOfPairs = sizeof($returnKeyPoints) / 2;
979
980 for($j = 0; $j < $numberOfPairs; $j++) {
981     $i = $j * 2;
982     $distance = $returnKeyPoints[$i + 1]['position_1'] - $returnKeyPoints[$i][
983         ↪ 'position_0'];
984     $vMax = getVMaxBetweenTwoPoints($distance, $returnKeyPoints[$i]['speed_0'],
985         ↪ $returnKeyPoints[$i + 1]['speed_1'], $id);
986     $returnKeyPoints[$i]['speed_1'] = $vMax;
987     $returnKeyPoints[$i]['position_1'] = $returnKeyPoints[$i]['position_0'] +
988         ↪ getBrakeDistance($returnKeyPoints[$i]['speed_0'], $vMax,

```

```

    ↪ $verzögerung);
979 $returnKeyPoints[$i + 1]['speed_0'] = $vMax;
980 $returnKeyPoints[$i + 1]['position_0'] = $returnKeyPoints[$i + 1]['
    ↪ position_1'] - getBrakeDistance($vMax, $returnKeyPoints[$i + 1]['
    ↪ speed_1'], $verzögerung);
981 }
982
983 return $returnKeyPoints;
984 }
985
986 // Wenn ein Key Point beschleunigt und der nächste Key Point abbremst, wird
987 // die Geschwindigkeit zwischen den beiden KeyPoints als $v_maxBetweenKeyPoints
988 // gespeichert und als $v_minBetweenKeyPoints der größere Wert von
989 // $keyPoints[$i]["speed_0"] und $keyPoints[$i + 1]["speed_1"]
990 function checkIfTheSpeedCanBeDecreased() {
991
992     global $keyPoints;
993     global $returnPossibleSpeed;
994
995     $returnPossibleSpeed = array();
996
997     for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
998         $v_maxBetweenKeyPoints = $keyPoints[$i]['speed_1'];
999         $v_minBetweenKeyPoints = null;
1000
1001         if ($keyPoints[$i]['speed_0'] < $v_maxBetweenKeyPoints && $keyPoints[$i +
            ↪ 1]['speed_1'] < $v_maxBetweenKeyPoints) {
1002             $v_minBetweenKeyPoints = $keyPoints[$i]['speed_0'];
1003             if ($keyPoints[$i + 1]['speed_1'] > $v_minBetweenKeyPoints) {
1004                 $v_minBetweenKeyPoints = $keyPoints[$i + 1]['speed_1'];
1005             }
1006         }
1007
1008         if (isset($v_minBetweenKeyPoints)) {
1009             if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints >= 10) {
1010                 $v_minBetweenKeyPoints = 10;
1011             } else if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints == 10) {
1012                 $v_minBetweenKeyPoints = null;
1013             }

```

```

1014     }
1015
1016     if ($v_minBetweenKeyPoints != null) {
1017         if ($v_minBetweenKeyPoints % 10 != 0) {
1018             $rest = $v_minBetweenKeyPoints % 10;
1019             $v_minBetweenKeyPoints = $v_minBetweenKeyPoints - $rest + 10;
1020         }
1021
1022         array_push($returnPossibleSpeed, array('KeyPoint_index' => $i, 'values'
            ↳ => range($v_minBetweenKeyPoints, $v_maxBetweenKeyPoints, 10)));
1023     }
1024 }
1025 if (sizeof($returnPossibleSpeed) > 0) {
1026     return array('possible' => true, 'range' => $returnPossibleSpeed);
1027 } else {
1028     return array('possible' => false, 'range' => array());
1029 }
1030 }
1031
1032 // Wenn in 'global_variables.php' der Variablen $useSpeedFineTuning
1033 // der Wert 'true' zugewiesen ist und das Fahrzeug zu früh an der
1034 // nächsten Betriebsstelle ankommt, wird überprüft, ob durch eine
1035 // vorzeitige Einleitung einer Verzögerung die exakte Ankunftszeit
1036 // eingehalten werden kann.
1037 function speedFineTuning(float $timeDiff, int $index) {
1038
1039     global $keyPoints;
1040     global $verzoegerung;
1041     global $globalTimeOnOneSpeed;
1042     global $useMinTimeOnSpeed;
1043
1044     $speed_0 = $keyPoints[$index]['speed_1'];
1045     $speed_1 = null;
1046     $availableDistance = $keyPoints[$index + 1]['position_0'] - $keyPoints[$index
        ↳ ]['position_1'];
1047     $timeBetweenKeyPoints = $keyPoints[$index + 1]['time_0'] - $keyPoints[$index
        ↳ ]['time_1'];
1048     $availableTime = $timeBetweenKeyPoints + $timeDiff;
1049

```

```

1050 if ($keyPoints[$index + 1]['speed_1'] != 0) {
1051     $speed_1 = $keyPoints[$index + 1]['speed_1'];
1052     $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
        ↳ 1]['speed_0'], $keyPoints[$index + 1]['speed_1'],
        ↳ $availableDistance, $availableTime);
1053
1054     if ($useMinTimeOnSpeed) {
1055         if (distanceWithSpeedToTime($speed_0, $availableDistance -
            ↳ $lengthDifference) > $globalTimeOnOneSpeed &&
            ↳ distanceWithSpeedToTime($speed_1, $lengthDifference) >
            ↳ $globalTimeOnOneSpeed) {
1056             $keyPoints[$index + 1]['position_0'] = $keyPoints[$index + 1]['
                ↳ position_0'] - $lengthDifference;
1057             $keyPoints[$index + 1]['position_1'] = $keyPoints[$index + 1]['
                ↳ position_1'] - $lengthDifference;
1058         }
1059     } else {
1060         $keyPoints[$index + 1]['position_0'] = $keyPoints[$index + 1]['position_0
            ↳ '] - $lengthDifference;
1061         $keyPoints[$index + 1]['position_1'] = $keyPoints[$index + 1]['position_1
            ↳ '] - $lengthDifference;
1062     }
1063 } else if ($keyPoints[$index + 1]['speed_0'] > 10) {
1064     $speed_1 = 10;
1065     $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index +
        ↳ 1]['speed_0'],10, $availableDistance, $availableTime);
1066
1067     if ($useMinTimeOnSpeed) {
1068         if (distanceWithSpeedToTime($speed_0, $availableDistance -
            ↳ $lengthDifference) > $globalTimeOnOneSpeed &&
            ↳ distanceWithSpeedToTime($speed_1, $lengthDifference) >
            ↳ $globalTimeOnOneSpeed) {
1069             $firstKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_0'] -
                ↳ $lengthDifference),($keyPoints[$index + 1]['position_0'] -
                ↳ $lengthDifference + getBrakeDistance($keyPoints[$index + 1]['
                ↳ speed_0'],10, $verzoegerung)), $keyPoints[$index + 1]['speed_0'
                ↳ ],10);
1070             $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_1'] -
                ↳ getBrakeDistance(10, 0, $verzoegerung)), $keyPoints[$index + 1]['

```

```

1071         ↪ position_1'],10,$keyPoints[$index + 1]['speed_1']);
1072     $keyPoints[$index + 1] = $secondKeyPoint;
1073     array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1074 }
1075 } else {
1076     $firstKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_0'] -
1077         ↪ $lengthDifference),($keyPoints[$index + 1]['position_0'] -
1078         ↪ $lengthDifference + getBrakeDistance($keyPoints[$index + 1]['
1079         ↪ speed_0'],10, $verzoegerung)), $keyPoints[$index + 1]['speed_0'],10)
1080         ↪ ;
1081     $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]['position_1'] -
1082         ↪ getBrakeDistance(10, 0, $verzoegerung)), $keyPoints[$index + 1]['
1083         ↪ position_1'],10,$keyPoints[$index + 1]['speed_1']);
1084     $keyPoints[$index + 1] = $secondKeyPoint;
1085     array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1086 }
1087 }
1088 }
1089 // Sucht den KeyPoint der zu maximalen Geschwindigkeit beschleunigt
1090 // Wenn die maximale Geschwindigkeit mehrfach erreicht wird, wird
1091 // der letzte dieser KeyPoints genommen
1092 //
1093 // Zu dem Index wird auch die Speed Range abgespeichert wie bei
1094 // checkIfTheSpeedCanBeDecreased()
1095 function findMaxSpeed(array $speedDecrease) {
1096     $maxSpeed = 0;
1097     $minSpeed = 0;
1098     $keyPointIndex = null;
1099
1100     for ($i = 0; $i < sizeof($speedDecrease['range']); $i++) {
1101         if (max($speedDecrease['range'][$i]['values']) >= $maxSpeed) {
1102             $maxSpeed = max($speedDecrease['range'][$i]['values']);
1103             $minSpeed = min($speedDecrease['range'][$i]['values']);
1104             $keyPointIndex = $speedDecrease['range'][$i]['KeyPoint_index'];
1105         }
1106     }
1107     return array('min_speed' => $minSpeed, 'max_speed' => $maxSpeed, '
1108         ↪ first_key_point_index' => $keyPointIndex);

```

```

1102 }
1103
1104 // Überprüft beim Start der Fahrtverlaufsrechnung,
1105 // ob es möglich ist einen Fahrtverlauf zu ermitteln
1106 function checkIfItsPossible(int $id) {
1107
1108     global $currentSpeed;
1109     global $distanceToNextStop;
1110     global $verzoegerung;
1111     global $globalTimeOnOneSpeed;
1112     global $errorMinTimeOnSpeed;
1113
1114     $minTimeIsPossible = true;
1115
1116     if ($currentSpeed == 0) {
1117         $distance_0 = getBrakeDistance(0, 10, $verzoegerung);
1118         $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1119         $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
            ↳ $distance_1);
1120
1121         if ($time < $globalTimeOnOneSpeed) {
1122             $minTimeIsPossible = false;
1123
1124             if ($errorMinTimeOnSpeed) {
1125                 emergencyBreak($id);
1126             }
1127
1128             echo "Der Zug schafft es ohne eine Gefahrenbremsung am Ziel anzukommen,
            ↳ kann aber nicht die mind. Zeit einhalten.\n";
1129         }
1130     } else {
1131         if (getBrakeDistance($currentSpeed, 0, $verzoegerung) !=
            ↳ $distanceToNextStop) {
1132             $distance_0 = getBrakeDistance($currentSpeed, 10, $verzoegerung);
1133             $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1134             $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
            ↳ $distance_1);
1135
1136             if ($time < $globalTimeOnOneSpeed) {

```

```

1137     $minTimeIsPossible = false;
1138
1139     if ($errorMinTimeOnSpeed) {
1140         emergencyBreak($id);
1141     }
1142
1143     echo "Der Zug schafft es, ohne eine Gefahrenbremsung am Ziel anzukommen
        ↳ .\n";
1144 }
1145 }
1146 }
1147 return $minTimeIsPossible;
1148 }
1149
1150 // Überprüft, ob der vorgeschriebene Wert aus der Variablen
1151 // $globalTimeOnOneSpeed eingehalten wird, falls die
1152 // Variable $useMinTimeOnSpeed den Wert 'true' hat
1153 function toShortOnOneSpeed () {
1154
1155     global $keyPoints;
1156     global $verzoegerung;
1157
1158     $localKeyPoints = $keyPoints;
1159     $subsections = createSubsections($localKeyPoints);
1160
1161     while (toShortInSubsection($subsections)) {
1162         $breakesOnly = true;
1163         foreach ($subsections as $sectionKey => $sectionValue) {
1164             if ($sectionValue['failed']) {
1165                 if (!$sectionValue['brakes_only']) {
1166                     $breakesOnly = false;
1167                 }
1168
1169                 $return = postponeSubsection($localKeyPoints, $sectionValue);
1170
1171                 if (!$return['fail']) {
1172                     $localKeyPoints = $return['keyPoints'];
1173                 } else {
1174                     if (!$sectionValue['brakes_only']) {

```



```

1175     $localKeyPoints[$sectionValue['max_index']]['speed_1'] -= 10;
1176     $localKeyPoints[$sectionValue['max_index'] + 1]['speed_0'] -= 10;
1177     $localKeyPoints[$sectionValue['max_index']]['position_1'] =
        ↳ $localKeyPoints[$sectionValue['max_index']]['position_0'] +
        ↳ getBrakeDistance($localKeyPoints[$sectionValue['max_index']]['
        ↳ 'speed_0'], $localKeyPoints[$sectionValue['max_index']]['
        ↳ speed_1'], $verzoeigerung);
1178     $localKeyPoints[$sectionValue['max_index'] + 1]['position_0'] =
        ↳ $localKeyPoints[$sectionValue['max_index'] + 1]['position_1']
        ↳ - getBrakeDistance($localKeyPoints[$sectionValue['max_index'
        ↳ ] + 1]['speed_0'], $localKeyPoints[$sectionValue['max_index'
        ↳ + 1]['speed_1'], $verzoeigerung);
1179     $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1180     $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
1181     break;
1182 }
1183 }
1184 }
1185 }
1186 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1187 $localKeyPoints = array_values($localKeyPoints);
1188 $subsections = createSubsections($localKeyPoints);
1189
1190 if ($breakesOnly) {
1191     break;
1192 }
1193 }
1194 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1195 $keyPoints = $localKeyPoints;
1196 }
1197
1198 // Überprüft, ob innerhalb einer $subsection Beschleunigungs- und
1199 // Bremsvorgänge später bzw. früher eingeleitet werden können.
1200 function postponeSubsection (array $localKeyPoints, array $subsection) {
1201
1202     global $globalTimeOnOneSpeed;
1203     global $verzoeigerung;
1204
1205     $deletedKeyPoints = array();

```

```

1206 $numberOfKeyPoints = sizeof($subsection['indexes']);
1207 $indexMaxSection = array_search($subsection['max_index'], $subsection['
    ↳ indexes']);
1208 $indexLastKeyPoint = array_key_last($subsection['indexes']);
1209
1210 if ($subsection['is_prev_section']) {
1211     $timeDiff = $localKeyPoints[$subsection['indexes'][0]]['time_0'] -
        ↳ $localKeyPoints[$subsection['indexes'][0] - 1]['time_1'] -
        ↳ $globalTimeOnOneSpeed;
1212     if ($timeDiff < 0) {
1213         $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection['indexes'
        ↳ ][0]]['speed_0'] / 3.6;
1214         if (!($localKeyPoints[$subsection['indexes'][0]]['position_1'] +
        ↳ $positionDiff > $localKeyPoints[$subsection['indexes'][
        ↳ $indexMaxSection + 1]]['position_0'])) {
1215             $localKeyPoints[$subsection['indexes'][0]]['position_0'] +=
        ↳ $positionDiff;
1216             $localKeyPoints[$subsection['indexes'][0]]['position_1'] +=
        ↳ $positionDiff;
1217             if ($localKeyPoints[$subsection['indexes'][0]]['position_1'] >
        ↳ $localKeyPoints[$subsection['indexes'][0] + 1]['position_0']) {
1218                 array_push($deletedKeyPoints, $subsection['indexes'][0] + 1);
1219                 $numberOfKeyPoints -= 1;
1220                 $v_0 = $localKeyPoints[$subsection['indexes'][0]]['speed_0'];
1221                 $v_1 = $localKeyPoints[$subsection['indexes'][0] + 1]['speed_1'];
1222                 $localKeyPoints[$subsection['indexes'][0]]['position_1'] =
        ↳ $localKeyPoints[$subsection['indexes'][0]]['position_0'] +
        ↳ getBrakeDistance($v_0, $v_1, $verzoegerung);
1223                 $localKeyPoints[$subsection['indexes'][0]]['speed_1'] = $v_1;
1224             }
1225             $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
        ↳ $deletedKeyPoints);
1226         }
1227     }
1228 }
1229
1230 for ($i = 1; $i <= $indexMaxSection; $i++) {
1231     if (!in_array($subsection['indexes'][$i], $deletedKeyPoints)) {

```

```

1232     $timeDiff = $localKeyPoints[$subsection['indexes'][$i]]['time_0'] -
        ↳ $localKeyPoints[$subsection['indexes'][$i] - 1]['time_1'] -
        ↳ $globalTimeOnOneSpeed;
1233     if ($timeDiff < 0) {
1234         $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection['indexes'][$
            ↳ $i]]['speed_0'] / 3.6;
1235         if (!(($localKeyPoints[$subsection['indexes'][$i]]['position_1'] +
            ↳ $positionDiff > $localKeyPoints[$subsection['indexes'][$
            ↳ $indexMaxSection + 1]]['position_0']))) {
1236             $localKeyPoints[$subsection['indexes'][$i]]['position_0'] +=
                ↳ $positionDiff;
1237             $localKeyPoints[$subsection['indexes'][$i]]['position_1'] +=
                ↳ $positionDiff;
1238             if ($i < $indexMaxSection && $localKeyPoints[$subsection['indexes'][$i]
                ↳ ]['position_1'] > $localKeyPoints[$subsection['indexes'][$i] +
                ↳ 1]['position_0']) {
1239                 array_push($deletedKeyPoints, ($subsection['indexes'][$i] + 1));
1240                 $numberOfKeyPoints -= 1;
1241                 $v_0 = $localKeyPoints[$subsection['indexes'][$i]]['speed_0'];
1242                 $v_1 = $localKeyPoints[$subsection['indexes'][$i] + 1]['speed_1'];
1243                 $localKeyPoints[$subsection['indexes'][$i]]['position_1'] =
                    ↳ $localKeyPoints[$subsection['indexes'][$i]]['position_0'] +
                    ↳ getBrakeDistance($v_0, $v_1, $verzoegerung);
1244                 $localKeyPoints[$subsection['indexes'][$i]]['speed_1'] = $v_1;
1245             }
1246             $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
                ↳ $deletedKeyPoints);
1247         }
1248     }
1249 }
1250 }
1251
1252 if ($subsection['is_next_section']) {
1253     $timeDiff = $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint] +
        ↳ 1]['time_0'] - $localKeyPoints[$subsection['indexes'][$
        ↳ $indexLastKeyPoint]]['time_1'] - $globalTimeOnOneSpeed;
1254     if ($timeDiff < 0) {
1255         $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]['
            ↳ speed_1'] / 3.6;

```

```

1256 if (!($localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]][
    ↳ position_0'] - $positionDiff < $localKeyPoints[$subsection['indexes
    ↳ '][$indexMaxSection]['position_0'])) {
1257 $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]][position_0'
    ↳ ] -= $positionDiff;
1258 $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]][position_1'
    ↳ ] -= $positionDiff;
1259 if ($localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]][
    ↳ position_0'] < $localKeyPoints[$subsection['indexes'][$
    ↳ $indexLastKeyPoint] - 1][position_1']) {
1260 array_push($deletedKeyPoints, ($subsection['indexes'][$
    ↳ $indexLastKeyPoint] - 1));
1261 $numberOfKeyPoints -= 1;
1262 $v_0 = $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint] -
    ↳ 1][speed_0'];
1263 $v_1 = $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]][
    ↳ speed_1'];
1264 $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]][
    ↳ position_0'] = $localKeyPoints[$subsection['indexes'][$
    ↳ $indexLastKeyPoint]][position_1'] - getBrakeDistance($v_0,
    ↳ $v_1, $verzoegerung);
1265 $localKeyPoints[$subsection['indexes'][$indexLastKeyPoint]][speed_0']
    ↳ = $v_0;
1266 }
1267 $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
    ↳ $deletedKeyPoints);
1268 }
1269 }
1270 }
1271
1272 for ($i = $indexLastKeyPoint - 1; $i > $indexMaxSection; $i--) {
1273 if (!in_array($i, $deletedKeyPoints)) {
1274 $timeDiff = $localKeyPoints[$subsection['indexes'][$i + 1]][time_0'] -
    ↳ $localKeyPoints[$subsection['indexes'][$i]][time_1'] -
    ↳ $globalTimeOnOneSpeed;
1275 if ($timeDiff < 0) {
1276 $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint][
    ↳ speed_1'] / 3.6;

```

```

1277     if (!($localKeyPoints[$subsection['indexes'][$i]]['position_0'] -
        ↪ $positionDiff < $localKeyPoints[$subsection['indexes'][$
        ↪ $indexMaxSection]]['position_0'])) {
1278     $localKeyPoints[$subsection['indexes'][$i]]['position_0'] -=
        ↪ $positionDiff;
1279     $localKeyPoints[$subsection['indexes'][$i]]['position_1'] -=
        ↪ $positionDiff;
1280     if ($i > ($indexMaxSection + 1) && $localKeyPoints[$subsection['
        ↪ indexes'][$i]]['position_0'] < $localKeyPoints[$subsection['
        ↪ indexes'][$i] - 1]]['position_1']) {
1281     array_push($deletedKeyPoints, ($subsection['indexes'][$i] - 1));
1282     $numberOfKeyPoints -= 1;
1283     $v_0 = $localKeyPoints[$subsection['indexes'][$i] - 1]['speed_0'];
1284     $v_1 = $localKeyPoints[$subsection['indexes'][$i]]['speed_1'];
1285     $localKeyPoints[$subsection['indexes'][$i]]['position_0'] =
        ↪ $localKeyPoints[$subsection['indexes'][$i]]['position_1'] -
        ↪ getBrakeDistance($v_0, $v_1, $verzoegerung);
1286     $localKeyPoints[$subsection['indexes'][$i]]['speed_0'] = $v_0;
1287     }
1288     $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints,
        ↪ $deletedKeyPoints);
1289     }
1290     }
1291     }
1292 }
1293
1294 $keys = $subsection['indexes'];
1295
1296 foreach ($deletedKeyPoints as $index) {
1297     unset($keys[array_search($index, $keys)]);
1298 }
1299
1300 // Ordnet die Abschnitte zwischen zwei $subsection den $subsections zu
1301 $keys = array_values($keys);
1302 $failed = false;
1303
1304 if ($subsection['is_prev_section']) {
1305     if ($localKeyPoints[$keys[0]]['time_0'] - $localKeyPoints[$keys[0] - 1][
        ↪ time_1'] < $globalTimeOnOneSpeed) {

```

```

1306     $failed = true;
1307 }
1308 }
1309
1310 if ($subsection['is_next_section']) {
1311     if ($localKeyPoints[end($keys) + 1]['time_0'] - $localKeyPoints[end($keys)
1312         ↳ ]['time_1'] < $globalTimeOnOneSpeed) {
1313         $failed = true;
1314     }
1315 }
1316
1317 for ($i = 1; $i < sizeof($keys); $i++) {
1318     if ($localKeyPoints[$keys[$i]]['time_0'] - $localKeyPoints[$keys[$i - 1]]['
1319         ↳ time_1'] < $globalTimeOnOneSpeed) {
1320         $failed = true;
1321         break;
1322     }
1323 }
1324
1325 if ($failed) {
1326     return array('fail' => true, 'keyPoints' => array());
1327 } else {
1328     foreach ($deletedKeyPoints as $index) {
1329         unset($localKeyPoints[$index]);
1330     }
1331
1332     return array('fail' => false, 'keyPoints' => $localKeyPoints);
1333 }
1334 }
1335
1336 // Erstellt mittels der $keyPoints die $subsections
1337 function createSubsections (array $localKeyPoints) {
1338
1339     global $globalTimeOnOneSpeed;
1340
1341     $keyPoints = $localKeyPoints;
1342     $subsections = array();
1343     $subsection = array('max_index' => null, 'indexes' => array(), '
1344         ↳ is_prev_section' => false, 'is_next_section' => false);

```

```

1342 $maxIndex = null;
1343
1344 for($i = 0; $i < sizeof($keyPoints); $i++) {
1345     if ($i > 0) {
1346         if ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1'] && $keyPoints[
            ↳ $i - 1]['speed_0'] > $keyPoints[$i - 1]['speed_1'] || $i == sizeof(
            ↳ $keyPoints) - 1) {
1347             if ($i == sizeof($keyPoints) - 1) {
1348                 array_push($subsection['indexes'], $i);
1349             }
1350
1351             array_push($subsections, $subsection);
1352             $subsection['indexes'] = array();
1353         }
1354     }
1355
1356     if ($keyPoints[$i]['speed_0'] < $keyPoints[$i]['speed_1']) {
1357         $subsection['max_index'] = $i;
1358     }
1359
1360     array_push($subsection['indexes'], $i);
1361 }
1362
1363 // Überprüfung der Abschnitte zwischen zwei $subsections
1364 for ($i = 1; $i < sizeof($subsections); $i++) {
1365     $firstIndex = $subsections[$i]['indexes'][array_key_first($subsections[$i][
        ↳ 'indexes'])];
1366
1367     if ($keyPoints[$firstIndex]['time_0'] - $keyPoints[$firstIndex - 1]['time_1
        ↳ ''] < $globalTimeOnOneSpeed) {
1368         $subsections[$i]['is_prev_section'] = true;
1369         $subsections[$i]['failed'] = true;
1370     } else {
1371         $subsections[$i]['is_prev_section'] = false;
1372         $subsections[$i]['failed'] = false;
1373     }
1374 }
1375
1376 for ($i = sizeof($subsections) - 1; $i >= 0; $i--) {

```

```

1377     $isFirstSubsection = false;
1378     $isLastSubsection = false;
1379
1380     if ($i == 0) {
1381         $isFirstSubsection = true;
1382     }
1383
1384     if ($i == sizeof($subsections) - 1) {
1385         $isLastSubsection = true;
1386     }
1387
1388     if ($subsections[$i]['failed'] || failOnSubsection($keyPoints, $subsections
        ↪ [$i])) {
1389         $subsections[$i]['failed'] = true;
1390
1391         if (!$isFirstSubsection) {
1392             $subsections[$i]['is_prev_section'] = true;
1393         }
1394
1395         if (!$isLastSubsection) {
1396             if (!$subsections[$i + 1]['is_prev_section']) {
1397                 $subsections[$i]['is_next_section'] = true;
1398             }
1399         }
1400     } else {
1401         $subsections[$i]['failed'] = false;
1402     }
1403 }
1404
1405 for ($i = 0; $i < sizeof($subsections); $i++) {
1406     if (!isset($subsections[$i]['max_index'])) {
1407         $subsections[$i]['brakes_only'] = true;
1408         $subsections[$i]['max_index'] = $subsections[$i]['indexes'][0];
1409     } else {
1410         $subsections[$i]['brakes_only'] = false;
1411     }
1412 }
1413
1414 $subsections = array_values($subsections);

```



```

1415
1416     return array_reverse($subsections);
1417 }
1418
1419 // Überprüfung, ob es in einer $subsection zu einer Unterschreitung
1420 // der Mindestzeit kommt
1421 function failOnSubsection(array $keyPoints, array $subsection) {
1422
1423     global $globalTimeOnOneSpeed;
1424
1425     $failed = false;
1426
1427     for ($i = 1; $i < sizeof($subsection['indexes']); $i++) {
1428         if ($keyPoints[$subsection['indexes'][$i]]['time_0'] - $keyPoints[
1429             ↪ $subsection['indexes'][$i] - 1]['time_1'] < $globalTimeOnOneSpeed) {
1430             $failed = true;
1431             break;
1432         }
1433     }
1434
1435     return $failed;
1436 }
1437
1438 function toShortInSubsection (array $subsections) {
1439
1440     $foundError = false;
1441
1442     foreach ($subsections as $subsection) {
1443         if ($subsection['failed']) {
1444             $foundError = true;
1445             break;
1446         }
1447     }
1448
1449     return $foundError;
1450 }
1451
1452 function createCumulativeSections ($indexCurrentSection, $indexTargetSection,
1453     ↪ $currentPosition, $targetPosition, $next_lengths) {

```

```

1452
1453 $cumLength = array();
1454 $sum = 0;
1455
1456 foreach ($next_lengths as $index => $value) {
1457     if ($index >= $indexCurrentSection) {
1458         $sum += $value;
1459         $cumLength[$index] = $sum;
1460     }
1461 }
1462 // Berechnung der kumulierten Start- und Endlängen der Abschnitte
1463 for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
1464     if ($indexCurrentSection == $indexTargetSection) {
1465         $cumulativeSectionLengthStart[$i] = 0;
1466         $cumulativeSectionLengthEnd[$i] = intval($targetPosition -
            ↪ $currentPosition);
1467     } else {
1468         if ($i == $indexCurrentSection) {
1469             $cumulativeSectionLengthStart[$i] = 0;
1470             $cumulativeSectionLengthEnd[$i] = intval($cumLength[$i] -
            ↪ $currentPosition);
1471         } else if ($i == $indexTargetSection) {
1472             $cumulativeSectionLengthStart[$i] = intval($cumLength[$i - 1] -
            ↪ $currentPosition);
1473             $cumulativeSectionLengthEnd[$i] = intval($cumLength[$i - 1] +
            ↪ $targetPosition - $currentPosition);
1474         } else {
1475             $cumulativeSectionLengthStart[$i] = intval($cumLength[$i - 1] -
            ↪ $currentPosition);
1476             $cumulativeSectionLengthEnd[$i] = intval($cumLength[$i] -
            ↪ $currentPosition);
1477         }
1478     }
1479 }
1480
1481 return array($cumulativeSectionLengthStart, $cumulativeSectionLengthEnd);
1482 }
1483
1484 function toArr(){

```

```

1485     return func_get_args();
1486 }
1487
1488 // Ermittelt die Echtzeitdaten für eine Gefahrenbremsung
1489 function emergencyBreak ($id, $distanceToNextStop = 0) {
1490
1491     global $allUsedTrains;
1492     global $timeDifference;
1493     global $allTimes;
1494
1495     $targetSpeed = 0;
1496     $returnArray = array();
1497     $time = microtime(true) + $timeDifference;
1498     $currentSpeed = $allUsedTrains[$id]['current_speed'];
1499     $notverzoegerung = $allUsedTrains[$id]['notverzoegerung'];
1500     $currentSection = $allUsedTrains[$id]['current_section'];
1501
1502     echo 'Der Zug mit der Adresse: ', $allUsedTrains[$id]['adresse'], " leitet
        ↳ jetzt eine Gefahrenbremsung ein.\n";
1503
1504     if (getBrakeDistance($currentSpeed, $targetSpeed, $notverzoegerung) <=
        ↳ $distanceToNextStop) {
1505         for ($i = $currentSpeed; $i >= 0; $i = $i - 2) {
1506             array_push($returnArray, array('live_position' => 0, 'live_speed' => $i,
                ↳ 'live_time' => $time, 'live_relative_position' => 0, 'live_section'
                ↳ => $currentSection, 'live_is_speed_change' => true, '
                ↳ live_target_reached' => false, 'id' => $id, 'wendet' => false, '
                ↳ betriebsstelle' => 'Notbremsung', 'live_all_targets_reached' =>
                ↳ null));
1507             $time = $time + getBrakeTime($i, $i - 1, $notverzoegerung);
1508         }
1509     } else {
1510         $targetSpeedNotbremsung = getTargetBrakeSpeedWithDistanceAndStartSpeed(
            ↳ $distanceToNextStop, $notverzoegerung, $currentSpeed);
1511         $speedBeforeStop = intval($targetSpeedNotbremsung / 2) * 2;
1512
1513         if ($speedBeforeStop >= 10) {
1514             for ($i = $currentSpeed; $i >= 10; $i = $i - 2) {

```

```

1515     array_push($returnArray, array('live_position' => 0, 'live_speed' => $i,
        ↳ 'live_time' => $time, 'live_relative_position' => 0, '
        ↳ live_section' => $currentSection, 'live_is_speed_change' => true,
        ↳ 'live_target_reached' => false, 'id' => $id, 'wendet' => false,
        ↳ 'betriebsstelle' => 'Notbremsung', 'live_all_targets_reached' =>
        ↳ null));
1516     $time = $time + getBrakeTime($i, $i - 1, $notverzögerung);
1517 }
1518 array_push($returnArray, array('live_position' => 0, 'live_speed' => 0, '
        ↳ live_time' => $time, 'live_relative_position' => 0, 'live_section'
        ↳ => $currentSection, 'live_is_speed_change' => true, '
        ↳ live_target_reached' => false, 'id' => $id, 'wendet' => false, '
        ↳ betriebsstelle' => 'Notbremsung', 'live_all_targets_reached' =>
        ↳ null));
1519 } else {
1520     array_push($returnArray, array('live_position' => 0, 'live_speed' =>
        ↳ $currentSpeed, 'live_time' => $time, 'live_relative_position' => 0,
        ↳ 'live_section' => $currentSection, 'live_is_speed_change' => true,
        ↳ 'live_target_reached' => false, 'id' => $id, 'wendet' => false, '
        ↳ betriebsstelle' => 'Notbremsung', 'live_all_targets_reached' =>
        ↳ null));
1521     $time = $time + getBrakeTime($currentSpeed, $currentSpeed - 1,
        ↳ $notverzögerung);
1522     array_push($returnArray, array('live_position' => 0, 'live_speed' => 0, '
        ↳ live_time' => $time, 'live_relative_position' => 0, 'live_section'
        ↳ => $currentSection, 'live_is_speed_change' => true, '
        ↳ live_target_reached' => false, 'id' => $id, 'wendet' => false, '
        ↳ betriebsstelle' => 'Notbremsung', 'live_all_targets_reached' =>
        ↳ null));
1523 }
1524 }
1525
1526 $allTimes[$allUsedTrains[$id]['adresse']] = $returnArray;
1527 array_push($allUsedTrains[$id]['error'], 3);
1528
1529 return 0;
1530 }

```

## A.4 functions\_math.php

```
1 <?php
2
3 // Ermittelt die Geschwindigkeit, die ein Fahrzeug in einem Bremsvorgang
4 // nach einer gegebenen Distanz hat.
5 function getTargetBrakeSpeedWithDistanceAndStartSpeed (float $distance, float
    ↳ $verzoeigerung, int $speed) {
6     return sqrt((-2 * $verzoeigerung * $distance) + (pow(($speed / 3.6), 2)))*3.6;
7 }
8
9 // Ermittelt die Distanz, um die eine Verzögerung "verschoben" werden müsste,
10 // damit die exakte Ankunftszeit eingehalten werden kann.
11 function calculateDistanceforSpeedFineTuning(int $v_0, int $v_1, float
    ↳ $distance, float $time) {
12     return $distance - (($distance - $time * $v_1 / 3.6)/($v_0 / 3.6 - $v_1 /
    ↳ 3.6)) * ($v_0 / 3.6);
13 }
14
15 // Ermittelt die Distanz für Brems- und Verzögerungsvorgänge
16 function getBrakeTime (float $v_0, float $v_1, float $verzoeigerung) {
17     return abs((($v_1/3.6)/$verzoeigerung) - (($v_0/3.6)/$verzoeigerung));
18 }
19
20 // Ermittelt die Zeit, die ein Fahrzeug bei einer gegebenen Strecke für
21 // eine gegebene Distanz benötigt
22 function distanceWithSpeedToTime (int $v, float $distance) {
23     return (($distance)/($v / 3.6));
24 }
25
26 // Ermittlung der Strecke für eine Beschleunigung bzw. Verzögerung
27 function getBrakeDistance (float $v_0, float $v_1, float $verzoeigerung) {
28     return abs(0.5 * ((pow($v_0/3.6,2) - pow($v_1/3.6, 2))/($verzoeigerung)));
29 }
```

## A.5 functions\_cache.php

```
1 <?php
2
3 // Ermittelt die Längen aller Infra-Abschnitte
4 function createcacheInfraLaenge() {
5     $DB = new DB_MySQL();
6     $returnArray = array();
7     $infraLaenge = $DB->select('SELECT '.DB_TABLE_INFRAZUSTAND.'.'. 'id', ' '.
        ↳ DB_TABLE_INFRAZUSTAND.'.'. 'laenge' FROM ' '.DB_TABLE_INFRAZUSTAND.' '
        ↳ WHERE ' '.DB_TABLE_INFRAZUSTAND.'.'. 'type' = ' '. 'gleis'.'.'');
8     unset($DB);
9     foreach ($infraLaenge as $data) {
10         if ($data->laenge != null) {
11             $returnArray[$data->id] = intval($data->laenge);
12         }
13     }
14     return $returnArray;
15 }
16
17 // Ermittelt die zugehörigen Infra-Abschnitte zu den Betriebsstellen
18 function createCacheHaltepunkte() {
19
20     $DB = new DB_MySQL();
21     $returnArray = array();
22
23     $betriebsstellen = $DB->select('SELECT ' '.DB_TABLE_BETRIEBSSTELLEN_DATEN.'.'. '
        ↳ parent_kuerzel' FROM ' '.DB_TABLE_BETRIEBSSTELLEN_DATEN.' ' WHERE ' '.
        ↳ DB_TABLE_BETRIEBSSTELLEN_DATEN.'.'. 'parent_kuerzel' IS NOT NULL');
24     unset($DB);
25
26     foreach ($betriebsstellen as $betriebsstelle) {
27         $returnArray[$betriebsstelle->parent_kuerzel][0] = array();
28         $returnArray[$betriebsstelle->parent_kuerzel][1] = array();
29     }
30
31     foreach ($returnArray as $betriebsstelleKey => $betriebsstelleValue) {
32         $DB = new DB_MySQL();
33         $name = $betriebsstelleKey;
34         $name .= '%';
```

```

35 $asig = 'ASig';
36 $bksig = 'BkSig';
37 $vsig = 'VSig';
38 $ja = 'ja';
39 if ($betriebsstelleKey == 'XAB' || $betriebsstelleKey == 'XBL') {
40     $haltepunkte = $DB->select('SELECT '.DB_TABLE_SIGNALE_STANDORTE.'.'.
        ↳ freimelde_id', '.DB_TABLE_SIGNALE_STANDORTE.'.'.wirkrichtung' FROM
        ↳ '.DB_TABLE_SIGNALE_STANDORTE.'.' WHERE '.
        ↳ DB_TABLE_SIGNALE_STANDORTE.'"'.betriebsstelle' LIKE '$name' AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'.'.freimelde_id' IS NOT NULL AND '.
        ↳ DB_TABLE_SIGNALE_STANDORTE.'"'.fahrplanhalt' = '$ja'");
41     unset($DB);
42 } else if ($betriebsstelleKey == 'XTS') {
43     $haltepunkte = $DB->select('SELECT '.DB_TABLE_SIGNALE_STANDORTE.'.'.
        ↳ freimelde_id', '.DB_TABLE_SIGNALE_STANDORTE.'.'.wirkrichtung' FROM
        ↳ '.DB_TABLE_SIGNALE_STANDORTE.'.' WHERE '.
        ↳ DB_TABLE_SIGNALE_STANDORTE.'"'.betriebsstelle' LIKE '$name' AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'.'.freimelde_id' IS NOT NULL AND '.
        ↳ DB_TABLE_SIGNALE_STANDORTE . '"'.signaltyp' = '$bksig'");
44     unset($DB);
45 } else if ($betriebsstelleKey == 'XLG') {
46     $haltepunkte = $DB->select('SELECT '.DB_TABLE_SIGNALE_STANDORTE.'.'.
        ↳ freimelde_id', '.DB_TABLE_SIGNALE_STANDORTE.'.'.wirkrichtung' FROM
        ↳ '.DB_TABLE_SIGNALE_STANDORTE.'.' WHERE '.
        ↳ DB_TABLE_SIGNALE_STANDORTE.'"'.betriebsstelle' LIKE '$name' AND '".
        ↳ DB_TABLE_SIGNALE_STANDORTE.'.'.freimelde_id' IS NOT NULL AND '.
        ↳ DB_TABLE_SIGNALE_STANDORTE . '"'.signaltyp' != '$vsig'");
47     unset($DB);
48 } else {
49     $haltepunkte = $DB->select('SELECT '.DB_TABLE_SIGNALE_STANDORTE.'.'.
        ↳ freimelde_id', '.DB_TABLE_SIGNALE_STANDORTE.'.'.wirkrichtung'
        ↳ FROM '.DB_TABLE_SIGNALE_STANDORTE.'.' WHERE '.
        ↳ DB_TABLE_SIGNALE_STANDORTE . '"'.betriebsstelle' LIKE '$name' AND '
        ↳ " . DB_TABLE_SIGNALE_STANDORTE.'.'.freimelde_id' IS NOT NULL AND '
        ↳ '.DB_TABLE_SIGNALE_STANDORTE . '"'.signaltyp' = '$asig'");
50     unset($DB);
51 }
52
53 foreach ($haltepunkte as $haltepunkt) {

```

```

54     if ($haltepunkt->wirkrichtung == 0) {
55         array_push($returnArray[$betriebsstelleKey][0], intval($haltepunkt->
            ↳ freimelde_id));
56     } elseif ($haltepunkt->wirkrichtung == 1) {
57         array_push($returnArray[$betriebsstelleKey][1], intval($haltepunkt->
            ↳ freimelde_id));
58     }
59 }
60 }
61 $returnArray['XSC'][1] = array(734, 732, 735, 733, 692); // In der Datenbank
    ↳ ist für Richtung 1 für diese Abschnitte fahrplanhalt auf nein
    ↳ eingestellt
62 return $returnArray;
63 }
64
65 // Ermittelt die zugehörigen Infra-Abschnitte zu den Zwischen-Betriebsstellen
66 function createChacheZwischenhaltepunkte() {
67     $DB = new DB_MySQL();
68     $allZwischenhalte = array();
69     $returnArray = array();
70     $zwischenhalte = $DB->select('SELECT DISTINCT `'.DB_TABLE_SIGNALE_STANDORTE.'
        ↳ `'.betriebsstelle' FROM `'.DB_TABLE_SIGNALE_STANDORTE.'` WHERE `'.
        ↳ DB_TABLE_SIGNALE_STANDORTE.'`'.betriebsstelle' IS NOT NULL');
71     unset($DB);
72     foreach ($zwischenhalte as $halt) {
73         array_push($allZwischenhalte, $halt->betriebsstelle);
74     }
75     foreach ($allZwischenhalte as $halt) {
76         $DB = new DB_MySQL();
77         $zwischenhalte = $DB->select('SELECT `'.DB_TABLE_SIGNALE_STANDORTE.'`'.
            ↳ freimelde_id' FROM `'.DB_TABLE_SIGNALE_STANDORTE.'` WHERE `'.
            ↳ DB_TABLE_SIGNALE_STANDORTE.'`'.betriebsstelle' = '$halt' AND `'.
            ↳ DB_TABLE_SIGNALE_STANDORTE.'`'.freimelde_id' IS NOT NULL');
78         unset($DB);
79         if (sizeof($zwischenhalte) == 1) {
80             if (sizeof(explode('_', $halt)) == 2) {
81                 $returnArray[$halt] = intval($zwischenhalte[0]->freimelde_id);
82             }
83         }

```



```

84     }
85     return $returnArray;
86 }
87
88 function createCacheInfraToGbt () {
89     $DB = new DB_MySQL();
90     $infraArray = array();
91     $returnArray = array();
92     $allInfra = $DB->select('SELECT '' .DB_TABLE_FMA_GBT.''. 'infra_id' FROM '' .
        ↳ DB_TABLE_FMA_GBT.''. WHERE '' .DB_TABLE_FMA_GBT.''. 'infra_id' IS NOT
        ↳ NULL');
93     unset($DB);
94     foreach ($allInfra as $infra) {
95         array_push($infraArray, intval($infra->infra_id));
96     }
97     foreach ($infraArray as $infra) {
98         $DB = new DB_MySQL();
99         $gbt = $DB->select('SELECT '' .DB_TABLE_FMA_GBT.''. 'gbt_id' FROM '' .
        ↳ DB_TABLE_FMA_GBT.''. WHERE '' .DB_TABLE_FMA_GBT.''. 'infra_id' = '
        ↳ $infra')[0]->gbt_id;
100         unset($DB);
101         $returnArray[$infra] = intval($gbt);
102     }
103     return $returnArray;
104 }
105
106 function createCacheGbtToInfra () {
107
108     $DB = new DB_MySQL();
109
110     $returnArray = array();
111
112     $allGbt = $DB->select('SELECT DISTINCT '' .DB_TABLE_FMA_GBT.''. 'gbt_id' FROM '
        ↳ '.DB_TABLE_FMA_GBT.''. WHERE '' .DB_TABLE_FMA_GBT.''. 'gbt_id' IS NOT
        ↳ NULL');
113     unset($DB);
114
115     foreach ($allGbt as $gbt) {
116         $DB = new DB_MySQL();

```

```

117     $gbt = $gbt->gbt_id;
118     $infras = $DB->select('SELECT '.DB_TABLE_FMA_GBT.'.'. 'infra_id' FROM '.
        ↳ DB_TABLE_FMA_GBT.' ' WHERE '.DB_TABLE_FMA_GBT.'".'. 'gbt_id' = '$gbt'")
        ↳ ;
119     unset($DB);
120     $returnArray[$gbt] = array();
121     foreach ($infras as $infra) {
122         array_push($returnArray[$gbt], intval($infra->infra_id));
123     }
124 }
125 return $returnArray;
126 }
127
128 function createCacheFmaToInfra () {
129     $DB = new DB_MySQL();
130     $returnArray = array();
131     $fmaToInfra = $DB->select('SELECT '.DB_TABLE_FMA_GBT.'.'. 'infra_id', '.
        ↳ DB_TABLE_FMA_GBT.'.'. 'fma_id' FROM '.DB_TABLE_FMA_GBT.' ' WHERE '.
        ↳ DB_TABLE_FMA_GBT.'.'. 'fma_id' IS NOT NULL');
132     unset($DB);
133     foreach ($fmaToInfra as $value) {
134         $returnArray[intval($value->fma_id)] = intval($value->infra_id);
135     }
136     return $returnArray;
137 }
138
139 function createCacheToBetriebsstelle() {
140     $DB = new DB_MySQL();
141     $returnArray = array();
142     $fmaToInfra = $DB->select('SELECT '.DB_TABLE_SIGNAL_E_STANDORTE.'.'. 'id', '.
        ↳ DB_TABLE_SIGNAL_E_STANDORTE.'.'. 'betriebsstelle' FROM '.
        ↳ DB_TABLE_SIGNAL_E_STANDORTE.' ');
143     unset($DB);
144     foreach ($fmaToInfra as $value) {
145         $returnArray[intval($value->id)] = $value->betriebsstelle;
146     }
147     return $returnArray;
148 }
149

```

```

150 function createCacheFahrzeugeAbschnitte () {
151     $DB = new DB_MySQL();
152     $returnArray = array();
153     $fahrzeugeAbschnitte = $DB->select('SELECT '.DB_TABLE_FAHRZEUGE_ABSCHNITTE.'
        ↳ '.fahrzeug_id', '.DB_TABLE_FAHRZEUGE_ABSCHNITTE.'.'.infra_id', '.
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'.'.unixtimestamp' FROM '.
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.'');
154     unset($DB);
155     foreach ($fahrzeugeAbschnitte as $fahrzeug) {
156         $returnArray[intval($fahrzeug->fahrzeug_id)][intval($fahrzeug
        ↳ ->infra_id)];
157         $returnArray[intval($fahrzeug->fahrzeug_id)][intval($fahrzeug->unixtimestamp)] = intval(
        ↳ $fahrzeug->unixtimestamp);
158     }
159     return $returnArray;
160 }
161
162 function createCacheDecoderToAdresse () {
163     $DB = new DB_MySQL();
164     $returnArray = array();
165     $decoderToAdresse = $DB->select('SELECT '.DB_TABLE_FAHRZEUGE.'.'.id', '.
        ↳ DB_TABLE_FAHRZEUGE.'.'.adresse' FROM '.DB_TABLE_FAHRZEUGE.'');
166     unset($DB);
167     foreach ($decoderToAdresse as $fahrzeug) {
168         $returnArray[intval($fahrzeug->id)] = intval($fahrzeug->adresse);
169     }
170     return $returnArray;
171 }
172
173 function createCacheFahrplanSession() {
174     $DB = new DB_MySQL();
175     $fahrplanData = $DB->select('SELECT * FROM '.DB_TABLE_FAHRPLAN_SESSION.'.'.
        ↳ WHERE '.DB_TABLE_FAHRPLAN_SESSION.'.'.status' = '".1'."');
176     unset($DB);
177
178     return $fahrplanData[0];
179 }

```

## A.6 functions\_db.php

```
1 <?php
2
3 // Ermittelt die Daten aller Fahrzeuge.
4 function getAllTrains () {
5
6     global $cacheAdresseToID;
7     global $cacheIDToAdresse;
8     global $globalTrainVMax;
9
10    $allAdresses = getAllAdresses();
11    $DB = new DB_MySQL();
12    $allTrains = array();
13    $id = null;
14
15    foreach ($allAdresses as $adress) {
16        $train_fahrzeuge = get_object_vars($DB->select('SELECT '.
            ↳ DB_TABLE_FAHRZEUGE.'.'. 'id', ''.DB_TABLE_FAHRZEUGE.'.'. 'adresse', ''.
            ↳ DB_TABLE_FAHRZEUGE.'.'. 'speed', ''.DB_TABLE_FAHRZEUGE.'.'. 'dir', ''.
            ↳ DB_TABLE_FAHRZEUGE.'.'. 'zugtyp', ''.DB_TABLE_FAHRZEUGE.'.'. 'zuglaenge
            ↳ ', ''.DB_TABLE_FAHRZEUGE.'.'. 'verzoegerung', ''.DB_TABLE_FAHRZEUGE.'
            ↳ '. 'zustand' FROM ''.DB_TABLE_FAHRZEUGE.'.' WHERE ''.
            ↳ DB_TABLE_FAHRZEUGE.'.'. 'adresse' = $adress")[0]);
17        $id = $train_fahrzeuge['id'];
18        $train_daten = $DB->select('SELECT ''.DB_TABLE_FAHRZEUGE_DATEN.'.'. 'baureihe
            ↳ ' FROM ''.DB_TABLE_FAHRZEUGE_DATEN.'.' WHERE ''.
            ↳ DB_TABLE_FAHRZEUGE_DATEN.'.'. 'id' = $id")[0]->baureihe;
19        $train_baureihe = $DB->select('SELECT ''.DB_TABLE_FAHRZEUGE_BAUREIHEN.'.'. '
            ↳ vmax' FROM ''.DB_TABLE_FAHRZEUGE_BAUREIHEN.'.' WHERE ''.
            ↳ DB_TABLE_FAHRZEUGE_BAUREIHEN.'.'. 'nummer' = $train_daten");
20
21        if (sizeof($train_baureihe) != 0) {
22            $train_baureihe_return['v_max'] = intval($train_baureihe[0]->vmax);
23        } else {
24            $train_baureihe_return['v_max'] = $globalTrainVMax;
25        }
26
27        $id = intval($train_fahrzeuge['id']);
28        $cacheAdresseToID[intval($train_fahrzeuge['adresse'])] = intval($id);
```

```

29     $returnArray = array_merge($train_fahrzeuge, $train_baureihe_return);
30     $allTrains[$id] = $returnArray;
31 }
32
33 unset($DB);
34
35 $cacheIDToAdresse = array_flip($cacheAdresseToID);
36
37 return $allTrains;
38 }
39
40 // Ermittelt alle Fahrzeuge im eingleisigen Netz und gibt die neu hinzugefügten
41 // und entfernten Fahrzeuge getrennt zurück.
42 function updateAllTrainsOnTheTrack () {
43
44     global $allTrainsOnTheTrack;
45     $newTrains = array();
46     $removedTrains = array();
47     $allTrains = array();
48     $DB = new DB_MySQL();
49     $foundTrains = $DB->select('SELECT DISTINCT ``.DB_TABLE_FMA.``.‘
        ↳ decoder_adresse‘ FROM ``.DB_TABLE_FMA.`` WHERE ``.DB_TABLE_FMA.``.‘
        ↳ decoder_adresse‘ IS NOT NULL AND ``.DB_TABLE_FMA.``.‘.‘decoder_adresse‘
        ↳ <> ‘”.’0’.””’);
50     unset($DB);
51
52     foreach ($foundTrains as $train) {
53         array_push($allTrains, intval($train->decoder_adresse));
54         if (!in_array($train->decoder_adresse, $allTrainsOnTheTrack)) {
55             array_push($newTrains, intval($train->decoder_adresse));
56         }
57     }
58
59     foreach ($allTrainsOnTheTrack as $train) {
60         if (!in_array($train, $allTrains)) {
61             array_push($removedTrains, $train);
62         }
63     }
64 }

```

```

65     $allTrainsOnTheTrack = $allTrains;
66     return array('new' =>$newTrains, 'removed' =>$removedTrains);
67 }
68
69 // Ermittelt alle Fahrzeuge im eingleisigen Netz.
70 function findTrainsOnTheTracks () {
71
72     global $allTrainsOnTheTrack;
73
74     $DB = new DB_MySQL();
75     $foundTrains = $DB->select('SELECT DISTINCT '''.DB_TABLE_FMA.''. '
        ↳ decoder_adresse' FROM '''.DB_TABLE_FMA.''. ' WHERE '''.DB_TABLE_FMA.''. '
        ↳ decoder_adresse' IS NOT NULL AND '''.DB_TABLE_FMA.''. 'decoder_adresse'
        ↳ <> '''. '0'.'''');
76     unset($DB);
77
78     foreach ($foundTrains as $train) {
79         if (!in_array($train->decoder_adresse, $allTrainsOnTheTrack)) {
80             array_push($allTrainsOnTheTrack, intval($train->decoder_adresse));
81             prepareTrainForRide($train->decoder_adresse);
82         }
83     }
84 }
85
86 // Bestimmung der Position eines Zuges über die Adresse.
87 function getPosition(int $adresse) {
88
89     $returnPosition = array();
90     $DB = new DB_MySQL();
91     $position = $DB->select('SELECT '''.DB_TABLE_FMA.''. 'fma_id' FROM '''.
        ↳ DB_TABLE_FMA.''. ' WHERE '''.DB_TABLE_FMA.''. 'decoder_adresse' = $adresse"
        ↳ );
92     unset($DB);
93
94     if (sizeof($position) != 0) {
95         for ($i = 0; $i < sizeof($position); $i++) {
96             array_push($returnPosition, intval(get_object_vars($position[$i])['fma_id'
                ↳ '']));
97         }

```

```

98     }
99
100     return $returnPosition;
101 }
102
103 // Ermittelt die Fahrplandaten eines Zuges
104 function getNextBetriebsstellen (int $id) {
105
106     $DB = new DB_MySQL();
107     $returnBetriebsstellen = array();
108     $betriebsstellen = $DB->select('SELECT ``.DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.`
        ↳ ``.``betriebsstelle`` FROM ``.DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.` ` WHERE
        ↳ ``.DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.` ``.``zug_id`` = $id ORDER BY ``.
        ↳ DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN.` ``.``id`` ASC`');
109     unset($DB);
110
111     foreach ($betriebsstellen as $betriebsstellenIndex => $betriebsstellenValue)
        ↳ {
112         array_push($returnBetriebsstellen, $betriebsstellenValue->betriebsstelle);
113     }
114
115     if (sizeof($betriebsstellen) == 0) {
116         debugMessage('Zu dieser Zug ID sind keine nächsten Betriebsstellen im
        ↳ Fahrplan vorhanden.');
```

```

128     unset($DB);
129
130     if ($signal != null) {
131         $signal = intval(get_object_vars($signal[0])['id']);
132     }
133
134     return $signal;
135 }
136
137 // Kalibriert die Position des Fahrzeugs neu anhand der Daten in der Tabelle
138 // 'fahrzeuge_abschnitte'
139 function getCalibratedPosition ($id, $speed) {
140
141     global $cacheFahrzeugeAbschnitte;
142
143     $DB = new DB_MySQL();
144     $positionReturn = $DB->select('SELECT ``.DB_TABLE_FAHRZEUGE_ABSCHNITTE.`.`.`
        ↳ infra_id`,`.DB_TABLE_FAHRZEUGE_ABSCHNITTE.`.`.`unixtimestamp` FROM ``.
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.`.` WHERE ``.
        ↳ DB_TABLE_FAHRZEUGE_ABSCHNITTE.`.`.`fahrzeug_id` = $id")[0];
145     unset($DB);
146
147     if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
148         if ($positionReturn->unixtimestamp == $cacheFahrzeugeAbschnitte[$id]['
            ↳ unixtimestamp']) {
149             return array('possible' => false);
150         }
151     }
152
153     $timeDiff = time() - $positionReturn->unixtimestamp;
154     $position = ($speed / 3.6) * $timeDiff;
155
156     return array('section' => $positionReturn->infra_id, 'position' => $position)
        ↳ ;
157 }
158
159 // Liest die Adressen aller Fahrzeuge ein.
160 function getAllAdresses () {
161

```



```

162 $zustand = array('0', '1');
163 $returnAdresses = array();
164
165 echo 'Alle Züge, die den Zustand ', implode(', ', $zustand), " haben, werden
    ↳ eingelesen.\n\n";
166
167 $DB = new DB_MySQL();
168 $adresses = $DB->select('SELECT '.DB_TABLE_FAHRZEUGE.'.'. 'adresse', ' '.
    ↳ DB_TABLE_FAHRZEUGE.'.'. 'zustand' FROM '.DB_TABLE_FAHRZEUGE.'');
169 unset($DB);
170
171 foreach ($adresses as $adressIndex => $adressValue) {
172     if (in_array($adressValue->zustand, $zustand)) {
173         array_push($returnAdresses, (int) $adressValue->adresse);
174     }
175 }
176
177 return $returnAdresses;
178 }

```

## A.7 global\_variables.php

```
1 <?php
2
3 // Ermittlung der exakten Ankunftszeit
4 $useSpeedFineTuning = true;
5 // Mindestzeit für Beharrungsfahrten verwenden
6 $useMinTimeOnSpeed = true;
7 // Fehlermeldung, wenn die Mindestzeit für
8 // Beharrungsfahrten nicht eingehalten werden kann
9 $errorMinTimeOnSpeed = false;
10 // Anpassung der Geschwindigkeit, wenn das Fahrzeug
11 // zu früh an der Ziel-Betriebsstellen ankommen würde
12 $slowDownIfTooEarly = true;
13 // Neukalibrierung der Position verwenden
14 $useRecalibration = true;
15
16 // Bremsverzögerung bei einer Gefahrenbremsung [m/s^2]
17 $globalNotverzoeigerung = 2;
18 // Maximale Geschwindigkeit, wenn keine vorgegeben ist [km/h]
19 $globalTrainVMax = 10;
20 // Maximale Geschwindigkeit im aktuellen Abschnitt [km/h]
21 $globalSpeedInCurrentSection = 60;
22 // Mindesthaltezeit am ersten fahrplanmäßigen Halt [s]
23 $globalFirstHaltMinTime = 20;
24 // Kulanzbereich beim Rechnen mit float-Werten
25 $globalFloatingPointNumbersRoundingError = 0.0000000001;
26 // Mindestzeit für Beharrungsfahrten [s]
27 $globalTimeOnOneSpeed = 20;
28 // Distanzintervall für Beharrungsfahrten [m]
29 $globalDistanceUpdateInterval = 1;
```

## A.8 speed\_over\_position.m

```
1 %% Load cumulative sections
2
3 fname = '../json/VMaxOverCumulativeSections.json';
4 fid = fopen(fname);
5 raw = fread(fid,inf);
6 str = char(raw');
7 fclose(fid);
8 vmaxOverPosition = jsondecode(str);
9 vmaxOverPosition_Position = vmaxOverPosition(:,1);
10 vmaxOverPosition_v_max = vmaxOverPosition(:,2);
11
12 %% Load modified cumulative sections
13
14 fname = '../json/VMaxOverCumulativeSectionsMod.json';
15 fid = fopen(fname);
16 raw = fread(fid,inf);
17 str = char(raw');
18 fclose(fid);
19 vmaxOverPosition_mod = jsondecode(str);
20 vmaxOverPosition_Position_mod = vmaxOverPosition_mod(:,1);
21 vmaxOverPosition_v_max_mod = vmaxOverPosition_mod(:,2);
22
23 %% Load speed over position
24
25 fname = '../json/speedOverPosition.json';
26 fid = fopen(fname);
27 raw = fread(fid,inf);
28 str = char(raw');
29 fclose(fid);
30 val = jsondecode(str);
31
32 speedOverPosition_x_v1 = val(:,1);
33 speedOverPosition_y_v1 = val(:,2);
34
35 %% Load speed over position (all iterationsteps)
36
37 fname_it = '../json/speedOverPosition_prevIterations.json';
38 fid_it = fopen(fname_it);
```

```

39 raw_it = fread(fid_it,inf);
40 str_it = char(raw_it');
41 fclose(fid_it);
42 val_it = jsondecode(str_it);
43
44 %% Plot
45
46 hold on
47 figure(1)
48
49 % Plot infrastructuresections
50
51 p = line([0 0], [0 vmaxOverPosition_v_max(1)], 'LineStyle', '-.', 'LineWidth', 2, '
    ↳ color', 'black', 'DisplayName', ['Infra-Abschnitte']);
52 line([0 vmaxOverPosition_Position(1)], [vmaxOverPosition_v_max(1)
    ↳ vmaxOverPosition_v_max(1)], 'LineStyle', '-.', 'LineWidth', 2, 'color', 'black
    ↳ ', 'HandleVisibility', 'off');
53
54 for i = 1:size(vmaxOverPosition_Position) - 1
55     line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i + 1)], [
        ↳ vmaxOverPosition_v_max(i + 1) vmaxOverPosition_v_max(i + 1)], '
        ↳ LineStyle', '-.', 'LineWidth', 2, 'color', 'black', 'HandleVisibility', '
        ↳ off');
56     line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i)], [0
        ↳ vmaxOverPosition_v_max(i + 1)], 'LineStyle', '-.', 'LineWidth', 2, 'color'
        ↳ ', 'black', 'HandleVisibility', 'off');
57     line([vmaxOverPosition_Position(i) vmaxOverPosition_Position(i)], [0
        ↳ vmaxOverPosition_v_max(i)], 'LineStyle', '-.', 'LineWidth', 2, 'color', '
        ↳ black', 'HandleVisibility', 'off');
58     line([vmaxOverPosition_Position(i + 1) vmaxOverPosition_Position(i + 1)], [0
        ↳ vmaxOverPosition_v_max(i + 1)], 'LineStyle', '-.', 'LineWidth', 2, 'color
        ↳ ', 'black', 'HandleVisibility', 'off');
59 end
60
61 % Plot modified iterationsteps (incl. trainlength)
62 line([0 0], [0 vmaxOverPosition_v_max_mod(1)], 'LineStyle', '-.', 'LineWidth', 2, '
    ↳ color', 'red', 'HandleVisibility', 'off');
63 line([0 vmaxOverPosition_Position_mod(1)], [vmaxOverPosition_v_max_mod(1)
    ↳ vmaxOverPosition_v_max_mod(1)], 'LineStyle', '-.', 'LineWidth', 2, 'color', '

```

```

        ↪ red', 'DisplayName', ['Infra-Abschnitte' newline 'inkl. Zuglänge']);
64
65 for i = 1:size(vmaxOverPosition_Position_mod) - 1
66     line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i + 1)
        ↪ ], [vmaxOverPosition_v_max_mod(i + 1) vmaxOverPosition_v_max_mod(i +
        ↪ 1)], 'LineStyle', '-.', 'LineWidth', 2, 'color', 'red', 'HandleVisibility',
        ↪ 'off');
67     line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i)], [0
        ↪ vmaxOverPosition_v_max_mod(i + 1)], 'LineStyle', '-.', 'LineWidth', 2, '
        ↪ color', 'red', 'HandleVisibility', 'off');
68     line([vmaxOverPosition_Position_mod(i) vmaxOverPosition_Position_mod(i)], [0
        ↪ vmaxOverPosition_v_max_mod(i)], 'LineStyle', '-.', 'LineWidth', 2, 'color
        ↪ ', 'red', 'HandleVisibility', 'off');
69     line([vmaxOverPosition_Position_mod(i + 1) vmaxOverPosition_Position_mod(i +
        ↪ 1)], [0 vmaxOverPosition_v_max_mod(i + 1)], 'LineStyle', '-.', '
        ↪ LineWidth', 2, 'color', 'red', 'HandleVisibility', 'off');
70 end
71
72 % Plot all iterationsteps
73 for i = 1:length(val_it)
74     %plot(val_it{i}(:,1),val_it{i}(:,2),'.','markersize',8,'Color', [0.6 0.6
        ↪ 0.6], 'DisplayName', legend_name);
75
76 end
77
78 % PLOT speedcurve
79
80 plot(speedOverPosition_x_v1,speedOverPosition_y_v1,'LineWidth',4,'Color', [0.25
        ↪ 0.80 0.54], 'DisplayName', 'Fahrtverlauf');
81
82 %% Fillobjects
83
84 %fill([0, 0, 1090, 1090, 0], [0, 200, 200, 0, 0], 'b','facealpha',.2,'LineStyle
        ↪ ', 'none');
85
86 %% Adding text
87
88 %text(20,17,'1','Interpreter','latex','fontsize', 40);
89

```

```

90 %% Format plot
91
92 p.LineWidth = 2;
93 box off
94 fontSize = 18;
95 xlabel("Strecke [m]", 'FontSize', fontSize);
96 ylabel("Geschwindigkeit [km/h]", 'FontSize', fontSize);
97 x0=10;
98 y0=10;
99 width=1100;
100 height=600;
101 axis([-80 max(vmaxOverPosition_Position)+80 0 max(vmaxOverPosition_v_max)+10]);
102 axis([-20 max(vmaxOverPosition_Position)+20 0 max(vmaxOverPosition_v_max)+5]);
103 set(gcf, 'position', [x0,y0,width,height]);
104 set(gcf, 'PaperPositionMode', 'auto');
105 set(gca, 'FontSize', 18);
106 set(gca, 'Linewidth', 2);
107
108 t = gca;
109 exportgraphics(t, 'SpeedOverPosition.pdf', 'ContentType', 'vector');
110 hold off

```

## Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken wurden als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Berlin, 1. Oktober 2021

Ort, Datum

A handwritten signature in blue ink, consisting of stylized, cursive letters, positioned above a horizontal line.

Unterschrift