Technische Universitaet Berlin

Bachelorarbeit

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

Friedrich Kasper Völkers

betreut von
Dr.-Ing. Christian Blome

12. 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 Fahrplantreue, 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 Vorgehen 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 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

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code-Beispiele

Abkürzungsverzeichnis

EBuEf Eisenbahn-Betriebs- und Experimentierfeld

Infra-Abschnitt Infrastrukturabschnitt

Glossar

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.

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.

1 Einleitung

In dieser Arbeit wird eine Fahrzeugsteuerung für das 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.[1]

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 **??** die Ermittlung des Fahrtverlaufs im Detail beschrieben ist. Damit die Allgemeingültigkeit der Fahrtverlaufsberechnung in Kapitel **??** gezeigt werden kann, wurden Infrastrukturdaten verwendet, die in dieser Form im EBuEf nicht vorkommen. Aus diesem Grund wird in Kapitel **??** die Funktionsweise anhand eines Beispiels im EBuEf gezeigt und mit Hilfe der in Kapitel **??** 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.

[1] **?**

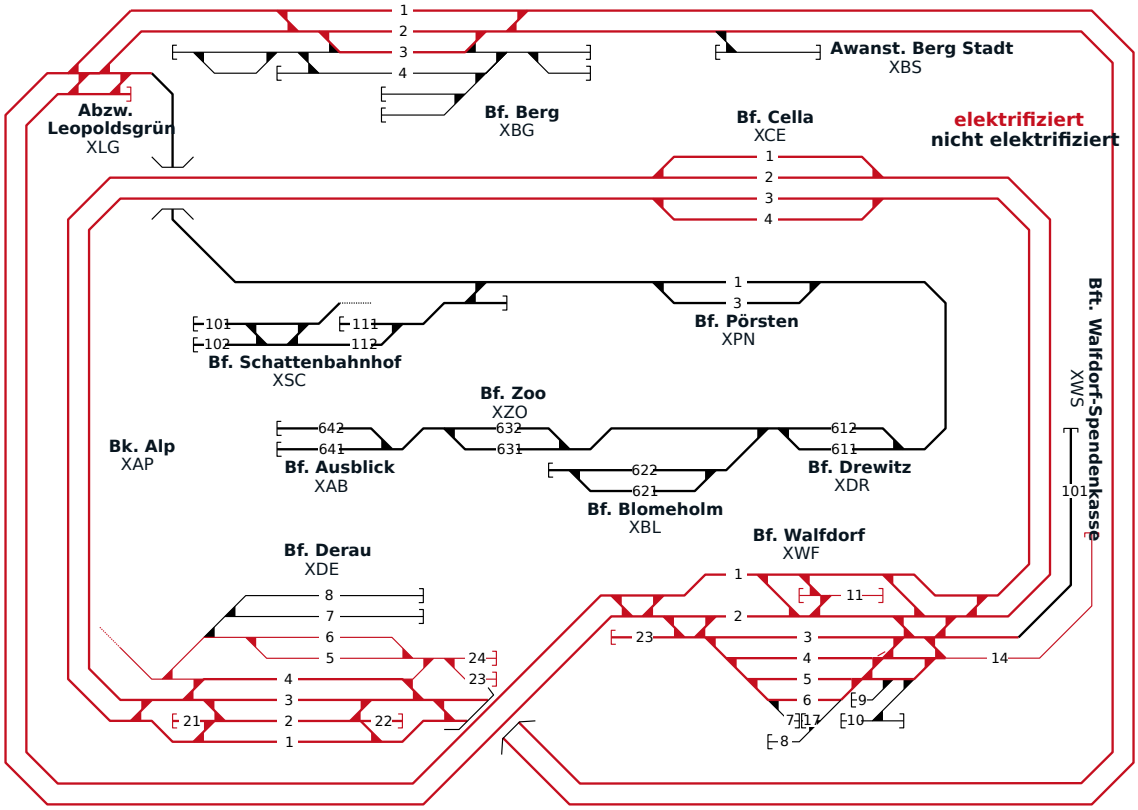

Abbildung 1: Schienennetz des EBuEfs (Quelle: www.ebuef.de/das-betriebsfeld/stellwerke; Letzter Zugriff am: 4. September 2021)

2 Grundlagen

2.1 Aufbau des Eisenbahn-Betriebs- und Experimentierfelds

Das Eisenbahn-Betriebs- und Experimentierfeld (EBuEf) ist unterteilt in ein eingleisiges nicht-elektrifiziertes und ein zweigleisiges elektrifiziertes Streckennetz, welche über die Betriebsstelle Leopoldsgrün 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.[2] Die Infra-Abschnitte sind mit der RailCom-Technik ausgestattet, welche über die Decoder in den Fahrzeugen den aktuellen Infra-Abschnitt ermittelt und diesen in der *fma*-Tabelle der *MySQL*-Datenbank speichert.[3] Zudem sind in der Datenbank alle Informationen über die Infrastruktur gespeichert. Für die Fahrzeugsteuerung essentiell sind dabei die aktuellen Signalbegriffe aller Signale und die Längen der Infra-Abschnitte.[4]

Für den Betrieb des EBuEfs muss eine neue Session angelegt werden und gestartet werden. Der Start der Session muss vor dem Start der Fahrzeugsteuerung erfolgen, da die Fahrzeug-

[2] **?**, S. 7, 42
[3] **?**
[4] **?**

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

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

steuerung beim Start alle benötigten Informationen der Session einliest.

2.2 Aufbau der *MySQL*-Datenbank

Alle Informationen und Daten, die für den Betrieb des EBuEfs benötigt werden, werden 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

An oberster Priorität der Fahrzeugsteuerung steht 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 minimal gehalten und Teile des Quellcodes, welche häufiger verwendet werden, 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 kleinstmöglichste 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 und die Verspätung so möglichst gering gehalten. 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, möglichst viel 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.[5]

Die Widerstandskräfte setzen sich aus dem Streckenwiderstand, dem Fahrzeugwiderstand und dem Anfahrwiderstand zusammen und lassen sich mit den gegebenen Daten nicht vollständig berechnen.[6] 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 gesamten Projekts und die für die Arbeit relevanten Dateien/Ordner dargestellt. Dateien, welche bereits vorhanden waren, sind die Funktionen mit einem Sternchen (*) markiert. Die für die Fahrzeugsteuerung essentiellen Dateien befinden sich innerhalb des *php*-Ordners, wobei die Datei *main.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 *SQL*-Datenbank befinden sich in dem *config*-Unterordner und global festgelegte Parameter sind in der Datei *globalVariables.php* abgespeichert. Für die Visualisierung (siehe Kapitel **??**) der Fahrtverläufe werden die Dateien aus dem *matlab*- und *json*-Ordner benötigt.

[5] **?**, S. 23 ff.
[6] **?**, S. 25 ff.

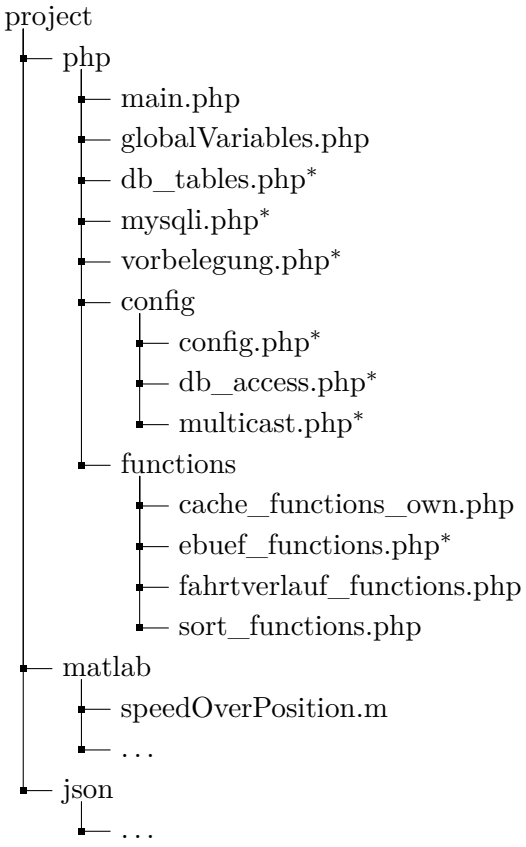

Abbildung 2: Aufbau der Dateistrukturen

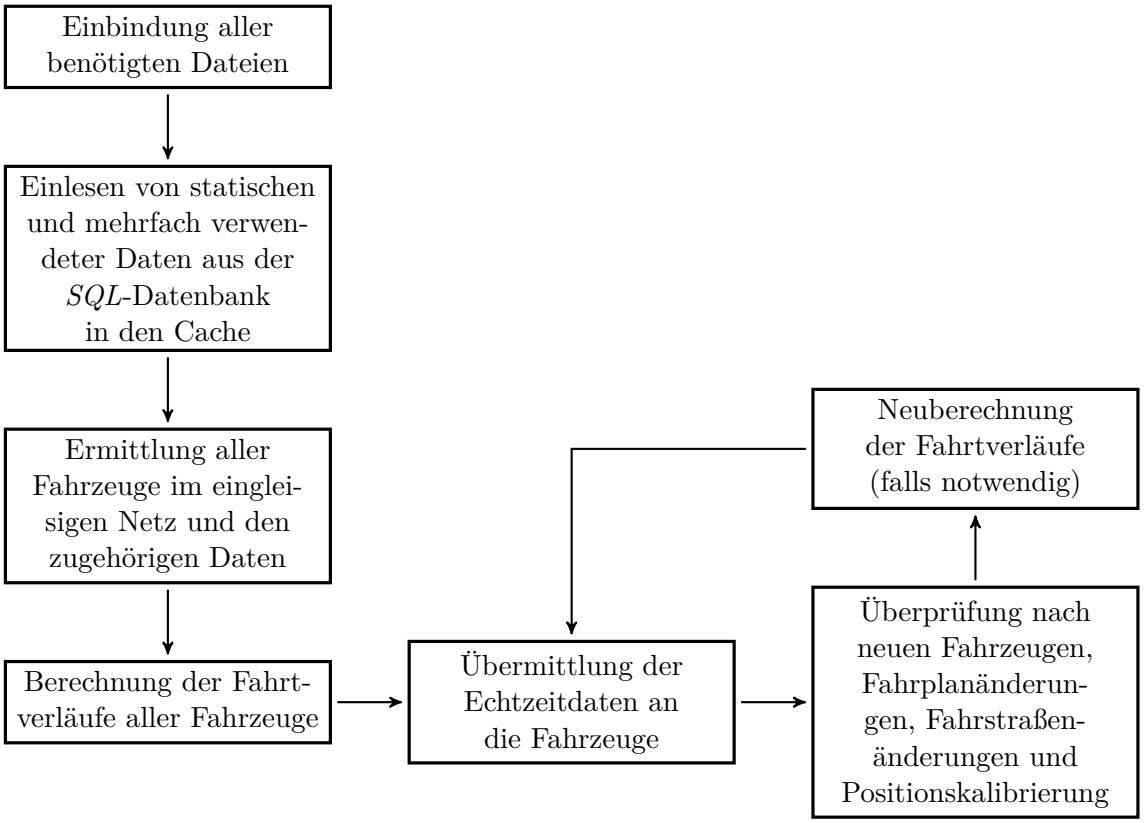

Abbildung 3: Ablauf des Hauptprogramms

3 Ablauf des Hauptprogramms

Um die Fahrzeugsteuerung zu starten, muss die Datei *main.php* ausgeführt werden. Obligatorisch für die Fahrzeugsteuerung ist die Abschnittsüberwachung (*abschnittueberwachung.php*), welche vor dem Start der Fahrzeugsteuerung ausgeführt werden muss. Auf die genaue Verwendung der Abschnittsüberwachung wird in Kapitel **??** eingegangen. In der folgenden Abbildung 3 ist der schematische Ablauf des Hauptprogramms abgebildet, welcher auch grob dem Aufbau dieses Kapitel dient.

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

Die Fahrzeugsteuerung benötigt für viele Berechnungen Daten aus der *SQL*-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 Programmstart bzw. bei der ersten Verwendung in den Cache geladen (Code-Beispiel 1). Beispielhaft zu nennen sind hierbei *$cacheInfraLaenge* (Länge aller Infrastrukturabschnitte in Metern [*m*]), *$cacheHaltepunkte* (zugehörige Infrastrukturabschnitte für alle Betriebsstellen und Richtung), *$cacheZwischenhaltepunkte* (zugehörige Infrastrukturabschnitte für alle Zwischen-Betriebsstellen, die nur einem Infrastrukturabschnitt

zugeordnet sind), *$cacheGbtToInfra* (Zuordnung der Infrastrukturabschnitte zu den GBT-Abschnitten) und *$cacheInfraToGbt* (Zuordnung der GBT-Abschnitte zu den Infrastrukturabschnitten).

```php
$cacheInfranachbarn = createCacheInfranachbarn();
$cacheInfradaten = createCacheInfradaten();
$cacheSignaldaten = createCacheSignaldaten();
$cacheInfraLaenge = createcacheInfraLaenge();
$cacheHaltepunkte = createCacheHaltepunkte();
$cacheZwischenhaltepunkte = createChacheZwischenhaltepunkte();
$cacheInfraToGbt = createCacheInfraToGbt();
$cacheGbtToInfra = createCacheGbtToInfra();
$cacheFmaToInfra = createCacheFmaToInfra();
$cacheInfraToFma = array_flip($cacheFmaToInfra);
$cacheFahrplanSession = createCacheFahrplanSession();
$cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
$cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
$cacheIDTDecoder = createCacheDecoderToAdresse();
$cacheDecoderToID = array_flip($cacheIDTDecoder);
$cacheAdresseToID = array();  // Filled with data in getAllTrains()
$cacheIDToAdresse = array();  // Filled with data in getAllTrains()
```

Code-Beispiel 1: Deklaration und Initialisierung der Cache Variablen (main.php)

3.2 Ermittlung aller Fahrzeuge im eingleisigen Netz und den zugehörigen Daten

ALLTRAINS BESCHREIBEN!

Das eingleisige Netz des EBuEf kann mittels der Railcon Technik und den Decodern in den Fahrzeugen ermitteln, welches Fahrzeug aktuell welche Infrastrukturabschnitte belegt. Belegt ein Fahrzeug einen Infrastrukturabschnitt, 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*() eingelesen und die zugehörige Adresse wird der Funktion *prepareTrainForRide*() übergeben. Für jedes Fahrzeug, welches dieser Funktion übergeben wird, wird in dem Array *$allUsedTrains* ein neuer Eintrag erstellt, wobei der Index der ID des Zugs entspricht. Dabei wird für jedes Fahrzeug die exakte Position bestimmt und der Fahrplan geladen. Bei der Positionsbestimmung wird davon ausgegangen, dass die Fahrzeuge direkt vor dem zugehörigen Signal stehen, da ansonsten die Position nicht exakt ermittelt werden kann. Belegt ein Fahrzeug mehrere Infrastrukturabschnitte, wird mittels der Fahrtrichtung der Züge der Abschnitt ermittelt, in dem sich der Zugkopf befindet. Die aktuelle Position wird daraufhin mit dem Infrastrukturabschnitt und der relativen Position (in Metern) innerhalb des Abschnitts angegeben. Dadurch, dass davon ausgegangen wird, dass das Fahrzeug sich direkt vor dem Signal befindet, entspricht die relative Position der Abschnittslänge. Für die Überprüfung, ob ein Fahrzeug nach Fahrplan fährt oder nicht wird die Funktion *getFahrzeugZugIds*()* aufgerufen. Wenn ein Fahrzeug ohne Fahrplan unterwegs ist (Rückgabewert der Funktion *getFahrzeugZugIds*()* ist ein leeres Array), wird in dem *$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*()* ist ein

Bezeichnung	Funktion
is_on_fahrstrasse (Boolescher Wert)	Befindet sich die Betriebsstelle auf der Fahrstraße
betriebstelle (String)	Name der Betriebsstelle
zeiten (Array)	Verspätung und Ankunfts- und Abfahrtszeiten (siehe Tabelle 3)
haltepunkte (Array)	Alle zugehörigen Infrastrukturabschnitte
fahrplanhalt (Boolescher Wert)	Ist diese Betriebsstelle ein Fahrplanhalt

Tabelle 2: Aufbau eines Arrays in *next_betriebsstellen_data*

Array mit allen Zug-IDs), wird mittels der Funktion *getNextBetriebsstellen*() der Fahrplan für den ersten Eintrag des Zug-ID Arrays aus der Datenbank geladen. Der Fahrplan wird in dem *$allUsedTrains*-Array in dem *next_betriebsstellen_data*-Array hinterlegt, welches für jede Betriebsstelle ein Array mit den benötigten Daten enthält. Die Indizierung dieser Einträge entspricht dabei den natürlichen Zahlen in aufsteigender Reihenfolge angefangen bei der 0. Hierbei werden alle Betriebsstellen hinzugefügt, bei denen ein fahrplanmäßiger Halt vorgesehen ist. Damit ein Fahrzeug nicht erst losfahren kann, wenn die Fahrstraße bis zur nächsten Betriebsstelle gestellt ist, werden auch alle Betriebsstellen hinzugefügt, welche eindeutig einem Infrastrukturabschnitt zugeordnet sind *$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 ist, dass die Zwischen-Betriebsstelle auf dem Weg zum nächsten fahrplanmäßigen Halt liegt. In Tabelle 2 ist für eine bessere Übersicht der Aufbau eines Eintrags abgebildet. Für die Ermittlung der Ankunfts- und Abfahrtzeiten wird die Funktion *getFahrplanzeiten*()* 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 den Eintrag *verspaetung* ergänzt. Zudem werden die Ankunfts- und Abfahrtszeiten in das Unixtimestamp-Format mittels der Funktion *getUhrzeit*()* 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 Infrastrukturabschnitte zugeordnet werden. Dafür werden mittels des *$cacheHaltepunkte*-Arrays, jeder Betriebsstelle mögliche Infrastrukturabschnitte zugeordnet. Das *$cacheHaltepunkte*-Array ist so aufgebaut, dass jeder Betriebsstelle für jede Richtung alle Infrastrukturabschnitte zugeordnet sind, welche ein Ausfahrsignal zugeordnet ist.

3.3 Berechnung der Fahrtverläufe aller Fahrzeuge

Nachdem für alle Fahrzeuge die Fahrplandaten (falls vorhanden) hinterlegt sind, wird für jedes Fahrzeug überprüft, wie die Fahrstraße aktuell eingestellt ist. Dafür wird die Funktion *calculateNextSections*() 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 Infrastrukturabschnitte ab, welche auf der Fahrstraße liegen. Im ersten Schritt wird überprüft, ob das Fahrzeug aktuell in einem Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist.

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

Tabelle 3: Aufbau des *zeiten*-Arrays in *next_betriebsstellen_data*

Wenn das der Fall ist, wird den Arrays *next_sections*, *next_lenghts* und *next_v_max* ein leeres Array zugeordnet. Wenn das Fahrzeug aktuell nicht in einem Abschnitt steht, welchem ein auf Halt stehendes Signal zugeordnet ist, wird über die Funktion *getNaechsteAbschnitte*()* die aktuelle Fahrstraße ermittelt und der Rückgabewert der Funktion *getNaechsteAbschnitte*()* in dem *$allUsedTrains*-Array unter dem Eintrag *last_get_naechste_abschnitte* gespeichert. Diese Speicherung ist notwendig, um zu überprüfen, ob sich die Fahrstraße geändert hat. Nach der Ermittlung der Fahrstraße und der Zuordnung der Infrastrukturabschnitte zu den Betriebsstellen wird im nächsten Schritt überprüft, welche Betriebsstellen des Fahrplans auf der aktuellen Fahrstraße liegen. Dafür wird mittels der Funktion *checkIfFahrstrasseIsCorrrect*() über alle Betriebsstellen der Fahrzeuge in aufsteigender Reihenfolge iteriert und die *haltepunkte* mit den Werten aus dem Array *next_sections* verglichen. Bei jedem Aufruf der Funktion wird dem Fahrzeug anfangs (falls das Fahrzeug nach Fahrplan fährt) in dem Array *$allUsedTrains* der Eintrag *fahrstrasse_is_correct* der Wert *false* zugewiesen und erst dann auf *true* gesetzt, wenn eine Betriebsstelle auf der Fahrstraße liegt. Beim Iterieren über die Betriebsstellen wird jeder Betriebsstelle anfangs der Wert *false* für den Eintrag *is_on_fahrstrasse* zugeordnet. Sobald ein Infrastrukturabschnitt einer Betriebsstelle in dem Array *next_sections* ist, wird dem Eintrag *is_on_fahrstrasse* der Wert *true* zugewiesen und unter dem Eintrag *used_haltepunkt* der Infrastrukturabschnitt gespeichert, welche auf der Fahrstraße liegt. Bei dem Iterieren über alle Betriebsstellen werden nur die Betriebsstellen beachtet, welche das Fahrzeug noch nicht erreicht hat. Dies wird über den Eintrag *angekommen (true/false)* jeder Betriebsstelle ermittelt. 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 *calculateFahrverlauf*() aufgerufen und innerhalb der Funktion überprüft, ob die Fahrstraße aktuell richtig eingestellt ist (*fahrstrasse_is_correct* == *true*). Wenn die Fahrstraße korrekt eingestellt ist, wird zwischen Fahrzeugen unterschieden, die nach Fahrplan unterwegs sind

und Fahrzeugen, die keinen Fahrplan haben. Für Fahrzeuge mit Fahrplan muss im ersten Schritt der nächste Halt ermittelt werden. 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 *$nextBetriebsstelleIndex* abgespeichert. Sollte unter den nächsten Betriebsstellen keine dabei sein, auf die die Kriterien zutreffen, wird in einer zweiten *for*-Schleife nach den selben Kriterien (außer dem des fahrplanmäßigen Halts) nach einer Betriebsstelle gesucht und sobald eine Betriebsstelle gefunden wurde, wird die Schleife abgebrochen und der Index der Betriebsstelle unter der Variablen *$nextBetriebsstelleIndex* abgespeichert. In dem Fall, dass keine nächste Betriebsstelle ermittelt werden konnte und das Fahrzeug aktuell eine Geschwindigkeit hat, für die gilt: $v > 0km/h$, so wird eine Gefahrenbremsung eingeleitet (siehe Kapitel **??**).

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 Infrastrukturabschnitt das Fahrzeug zum Stehen kommen soll (*used_haltepunk*-Eintrag der nächsten Betriebsstelle) und an welcher relativen Position innerhalb des Abschnitts das Fahrzeug angehalten soll (Länge des Infrastrukturabschnitts). Neben den Informationen zur Position, müssen noch die Informationen zur Zeit ermittelt werden. Dafür reicht es nicht aus, die Ankunfts- und Abfahrtszeit aus den Daten der Betriebsstelle zu übernehmen, denn in dem Fall einer Verspätung beispielsweise, würde das Fahrzeug zu kurz an einer Betriebsstelle anhalten. Deswegen, 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 *$endTime*) die aktuelle Simulationszeit genommen. Wenn die nächste Betriebsstelle nicht dem ersten fahrplanmäßigen Halt entspricht, wird als Zielzeit die Ankunftszeit der Betriebsstelle festgelegt und als Startzeit die Abfahrtszeit der vorherigen Betriebsstelle (*$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 überprüft, ob die Startzeit kleiner ist als die frühstmögliche Startzeit des Fahrzeugs (*earliest_possible_start_time*-Eintrag des Fahrzeugs) und dieser Zeit gleichgesetzt, falls das der Fall ist. Der Eintrag *earliest_possible_start_time* der Züge gibt die frühstmögliche Abfahrtzeit der Züge an und wird zum Beispiel bei einem Wendeauftrag auf die aktuelle Simulationszeit gesetzt und um $30s$ erhöht.

Allen Zügen, die ohne Fahrplan unterwegs sind, wird als Ziel-Infrastrukturabschnitt der letzte Infrastrukturabschnitt aus dem Array *last_get_naechste_abschnitte* verwendet, welchem ein Signal zugeordnet ist. Die Ziel-Position innerhalb des Abschnitts entspricht dabei ebenfalls der Länge des Abschnitts und die Überprüfung, ob ein Wendeauftrag nach dem Erreichen des Ziel-Infrastrukturabschnitt 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 Infrastrukturabschnitten aus dem *last_get_naechste_abschnitte*-Array ein Signal zugeordnet sein und die aktuelle Geschwindigkeit des Fahrzeugs ist größer als $0km/h$, so wird eine Gefahrenbremsung ein-

geleitet. Andernfalls wird die Funktion an dieser Stelle abgebrochen und es wird erst dann wieder versucht einen Fahrtverlauf zu berechnen, wenn sie die Fahrstraße geändert hat.

Nach der Ermittlung aller notwendigen Daten für die Berechnung des Fahrtverlaufs, wird für jedes Fahrzeug die Funktion *updateNextSpeed*() aufgerufen, welche den Fahrtverlauf berechnet und in Kapitel **??** im Detail beschrieben wird. Wichtig an dieser Stelle ist allerdings der Rückgabewert der Funktion. Dieser gibt für Fahrzeuge mit Fahrplan die Verspätung in Sekunden an, 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 oder nicht kann nur ermittelt werden, wenn die Ankunftszeit definiert ist. Für den ersten fahrplanmäßigen Halt ist in der *SQL*-Tabelle *fahrplan_sessionfahrplan* allerdings keine Ankunftszeit hinterlegt. Für den Fall, dass für ein Fahrzeug eine Fahrplan hinterlegt ist, das Fahrzeug in einem Infrastrukturabschnitt steht, welcher keiner 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. Aus diesem Grund, wurde in der Datei *globalVariables.php* die Variable *$globalFirstHaltMinTime* definiert, die angibt, wie lange ein Fahrzeug an der ersten Betriebsstelle des Fahrplans halten sollte. Wenn diese Zeit eingehalten werde kann, wird das Fahrzeug (sofern die Fahrstraße richtig eingestellt ist) zur Abfahrtszeit die Betriebsstelle verlassen. Andernfalls gilt für die Verspätung der ersten Betriebsstelle:

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

3.4 Übermittlung der Echtzeitdaten an die Fahrzeuge

Nach dem Aufruf der Funktion *updateNextSpeed*() sind für alle Fahrzeuge, für die ein Fahrtverlauf berechnet wurde, in dem Array *$allTimes* alle Echtzeitdaten enthalten. Das Array beinhaltet für jedes Fahrzeug, für das Echtzeitdaten ermittelt wurden, wiederum ein Array. Dieses Array ist unter der Adresse des Zuges abgespeichert und beinhaltet alle Echtzeitdaten eines Zuges. Der Aufbau eines Array mit Echtzeitdaten ist in Tabelle 4 dargestellt. In einer *while*-Schleife wird über alle Einträge des *$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 wenn dieser kleiner ist als die aktuelle Simulationszeit ist, ausgeführt. Nach jedem Durchlauf der *while*-Schleife wird diese mit der Funktion *sleep*() für $0,03s$ pausiert. An dieser Stelle wurde sich für einen Wert von $0,03s$ entschieden, da so die Position auf einen Meter genau bestimmt werden kann, wenn das Fahrzeug eine Geschwindigkeit von $120km/h$ hat.

Wenn für ein Fahrzeug neue Echtzeitinformationen vorliegen, wird im ersten Schritt überprüft, ob der Eintrag *live_is_speed_change true* ist, wenn das der Fall ist, wird die neue Geschwindigkeit über die Funktion *sendFahrzeugbefehl*()* dem Fahrzeug übergeben und mittels einer Terminal-Ausgabe angezeigt. Im zweiten Schritt wird der aktuelle Infrastrukturabschnitt, 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*() aufgerufen. In der Funktion wird neben der Richtungsänderung auch die neue Position ermittelt (die Position eines Fahrzeugs wird immer durch den Zugkopf beschrie-

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

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

ben) und überprüft, ob die Richtung geändert werden kann. Damit die Richtungsänderung auch funktioniert, wenn das Fahrzeug nicht am Ende eines Infrastrukturabschnitts steht, wird für die Ermittlung der neuen Position auf die Fahrzeuglänge der Abstand bis zum Ende Infrastrukturabschnitts addiert (siehe Abbildung 4). Über den aktuellen und die folgenden Infrastrukturabschnitte (ermittelt durch die Funktion *getNaechsteAbschnitte*()*, des aktuellen Infrastrukturabschnitts und der neuen Richtung) wird iteriert und die Summe der Längen gebildet, bis die Fahrzeuglänge (zuzüglich des Abstands bis zum Ende des Infrastrukturabschnitts) überschritten wird. Der Infrastrukturabschnitt, in dem die Fahrzeuglänge inkl. des Abstands zum ersten Mal überschritten wird, entspricht dem Infrastrukturabschnitt der neuen Position.

Sollte die Länge aller nächsten Abschnitte inklusive des aktuellen Abschnitts in der Sum-

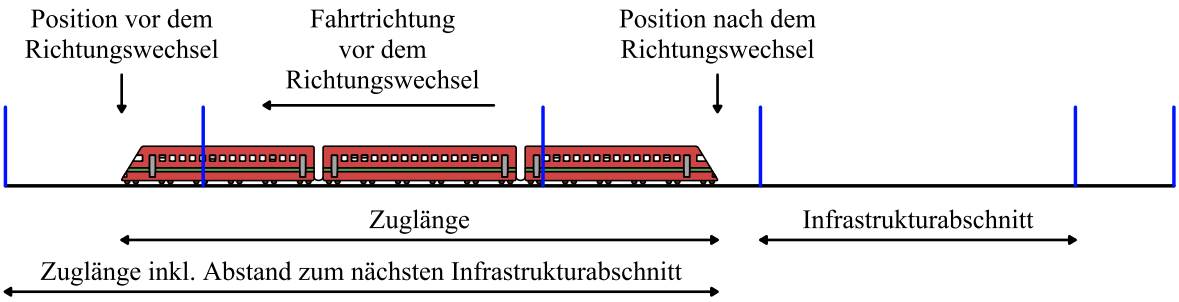

Abbildung 4: Eigene Darstellung der Positionsbestimmung bei einem Richtungswechsel

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

Tabelle 5: Verhalten eines Fahrzeugs nach dem Erreichen des Ziels

me kleiner sein, als die Zuglänge inkl. dem Abstands bis zum Ende des Infrastrukturabschnitts, so 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*()* die Geschwindigkeit $-4km/h$ (entspricht einem Wendeauftrag) übergeben.

Bei einem Fahrtverlauf kann es vorkommen, dass Fahrzeuge mit Fahrplan auf der 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*() 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*(), *addStopsectionsForTimetable*(), *calculateNextSections*(), *checkIfFahrstrasseIsCorrrect*() und *calculateFahrverlauf*() aufgerufen. Abgesehen von der ersten Funktion, werden diese Funktionen auch beim Start des Programms ausgeführt, welcher in Kapitel 3.2 beschrieben wird. Die Funktion *getFahrplanAndPositionForOneTrain*() ähnelt der in Kapitel 3.2 beschrieben Funktion *prepareTrainForRide*(), 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*() und *calculateFahrverlauf*() aufgerufen.

3.5 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* (*main.php*) festgelegt werden und ist standardgemäß auf 3 Sekunden festgelegt. Bei der Ermittlung und dem Vergleich der Fahrstraße wird für jedes Fahrzeug die Funktion *compareTwoNaechsteAbschnitte*() aufgerufen. Innerhalb dieser Funk-

tion wird die in Kapitel 3.2 erläutere Funktion *calculateNextSections*() aufgerufen, mit dem Unterschied, dass die ermittelten nächsten Infrastrukturabschnitte 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*() (standardgemäß auf *true* gesetzt) auf *false* gesetzt werden.

3.6 Neukalibrierung der Fahrzeugposition

Für eine genau Fahrzeugsteuerung ist die aktuelle Position der Züge **essenziäl!** und muss während der Fahrt kalibriert werden, damit Ungenauigkeiten ausgeglichen werden können. Dafür werden die Daten aus der *SQL*-Tabelle *fahrzeuge_abschnitte* benötigt, welche durch Abschnittsüberwachung ermittelt werden. Die Abschnittsüberwachung schreibt für jedes Fahrzeug den aktuellen Infrastrukturabschnitt 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 Infrastrukturabschnitt 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 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. Aus diesem Grund wird, sobald das Fahrzeug laut den Echtzeitdaten einen neuen Abschnitt befährt und aktuell nicht die Geschwindigkeit anpasst (*live_is_speed_change* == *false*), dem Eintrag *calibrate_section_one* der aktuelle Infrastrukturabschnitt hinzugefügt und dem Eintrag *calibrate_section_two* wird ebenfalls der aktuelle Infrastrukturabschnitt hinzugefügt, wenn *calibrate_section_one* ein Wert zugewiesen ist und dieser nicht dem aktuellen Infrastrukturabschnitt der Echtzeitdaten entspricht. Soabld 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 Infrastrukturabschnitt zugewiesen, wenn das Fahrzeug in diesem seit der Einfahrt die Geschwindigkeit nicht verändert hat. Wenn dem Eintrag *$useRecalibration* aus der Datei *globalVariables.php* der Wert *true* zugewiesen ist, wird in regelmäßigen Abständen überprüft, ob eine Neukalibrierung möglich ist. Das Zeitintervall, in dem die Überprüfung stattfindet ist standardmäßig auf auf 3 Sekunden eingestellt, kann aber mittels der Variable *$timeCheckCalibrationInterval* (*main.php*) angepasst werden.

Für die Neukalibrierung wird die Funktion *getCalibratedPosition*() (Code-Beispiel **??**) aufgerufen, welche als Rückgabewert die aktuelle relative Position und den aktuellen Infrastrukturabschnitt zurückgibt. Sollte die ermittelte Position innerhalb des Abschnitts größer als die Länge des Abschnitts, welche in dem Array *$cacheInfraLaenge* abgespeichert ist, sein, wird die Neukalibrierung nicht durchgeführt. Der aktuelle Infrastrukturabschnitt wird aus der Tabelle *fahrzeuge_abschnitte* der Datenbank geladen und durch die aktuelle Geschwindigkeit des Fahrzeugs und die Differenz der Zeit zwischen dem Einfahren in den Abschnitt und der aktuellen Zeit wird die relative Position innerhalb des Abschnitts berechnet.

3.7 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 Programm-

```php
function getCalibratedPosition ($id, $speed) {
  global $cacheFahrzeugeAbschnitte;
  $DB = new DB_MySQL();
  $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` = $id")[0];
  unset($DB);
  if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
    if ($positionReturn->unixtimestamp == $cacheFahrzeugeAbschnitte[$id]["unixtimestamp"
        ↪ ]) {
      return array("possible" => false);
    }
  }
  $timeDiff = time() - $positionReturn->unixtimestamp;
  $position = ($speed / 3.6) * $timeDiff;
  return array("possible" => true, "section" => $positionReturn->infra_id, "position" =>
      ↪  $position);
}
```

Code-Beispiel 2: *getCalibratedPosition*() (main.php)

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

start hinzugefügt werden. Für alle Fahrzeuge, die beim Start des Programms erkannt werden, wird in dem Array *$allTrainsOnTheTrack* die zugehörige Adresse gespeichert (*findTrainsOnTheTracks*()). Für die Überprüfung, ob Fahrzeuge entfernt wurden oder neu hinzugekommen sind, wird die Funktion *updateAllTrainsOnTheTrack*() verwendet. Diese Funktion wird - wie die Neukalibrierung in Kapitel **??** - alle 3 Sekunden ausgeführt. Bei dem Aufruf der Funktion werden alle Fahrzeuge geladen, denen in der *fma*-Tabelle aus der Datenbank ein Infrastrukturabschnitt zugeordnet ist und mit dem Array *$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 Infrastrukturabshcnitt 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*(), *addStopsectionsForTimetable*(), *calculateNextSections*(), *checkIfTrainReachedHaltepunkt*(), *checkIfFahrstrasseIsCorrrect*() und *calculateFahrverlauf*() aufgerufen (siehe Kapitel 3.2 und 3.3). Alle entfernten Fahrzeuge werden aus dem Array *$allUsedTrains* entfernt und somit nicht mehr von der Fahrzeugsteuerung beachtet.

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 Abschnitte
$next_lenghts (Array)	Längen aller Abschnitte
$next_v_max (Array)	Höchstgeschwindigkeit aller Abschnitte
$indexCurrentSection (Integer)	Index des aktuellen Abschnitts
$indexTargetSection (Integer)	Index des Ziel-Abschnitts
$cumulativeSectionLengthStart (Array)	Absolute Startposition aller Abschnitte
$cumulativeSectionLengthEnd (Array)	Absolute Endposition aller Abschnitte
$trainPositionChange (Array)	Alle absoluten Positionen des Fahrtverlaufs
$trainSpeedChange (Array)	Alle Geschwindigkeiten des Fahrtverlaufs

Tabelle 6: Beschreibung der verwendeten Variablen für die Fahrtverlaufsberechnung

4 Berechnung des Fahrtverlaufs

Der Fahrtverlauf eines Fahrzeuges wird bei der Berechnung in zwei verschiedenen Arten gespeichert. Einmal in so genannten *$keyPoints*, welche in einem Array die Start- und Zielgeschwindigkeit (*time_0* und *time_0*), die Start- und Endposition (*position_0* und *position_1*) und die Start- und Endzeit (*time_0* und *time_1*) einer Beschleunigung bzw. Verzögerung abspeichern. Für die Überprüfung, ob ein Fahrzeug die zulässige Höchstgeschwindigkeit in einem Infrastrukturabschnitt überschreitet, für die spätere Übermittlung der Echtzeitdaten an das Fahrzeug und die exakte Positionsbestimmung, werden mittels der *$keyPoints* für jede Geschwindigkeitsänderungen (und bei konstanter Geschwindigkeit in 1 Meter Abständen) die aktuelle relative Position innerhalb eines Infrastrukturabschnitts, der Infrastrukturabschnitt, die aktuelle Zeit und die aktuelle Geschwindigkeit in einem Array gespeichert. Der Fahrtverlauf wird mit der Funktion *updateNextSpeed*() berechnet, welche als Parameter unter anderem die Zugdaten aus dem *$allUsedTrains*-Array, Start- und Endzeit der Fahrt (*$startTime* und *$endTime*), den Ziel-Infrastrukturabschnitt (*$targetSection*) und die relative Position in dem Ziel-Infrastrukturabschnitt (*$targetPosition*) übergeben bekommt. Die für die Berechnung benötigten Daten werden in den in Tabelle **??** beschriebenen Variablen gespeichert. In dem folgenden Abschnitt werden die einzelnen Schritte beschrieben, die durchlaufen werden um den optimalen Fahrtverlauf zu berechnen. In der Darstellung **??** wird der Ablauf grob schematisch dargestellt.

4.1 Ermittlung der Start- und Endposition der einzelnen Infrastrukturabschnitte unter Berücksichtigung der Zuglänge

Für die Berechnung eines exemplarischen Fahrtverlaufs wurden die in Tabelle **??** definierten Infrastrukturabschnitte benutzt. Diese Abschnitte wurden so gewählt, sodass alle Funktio-

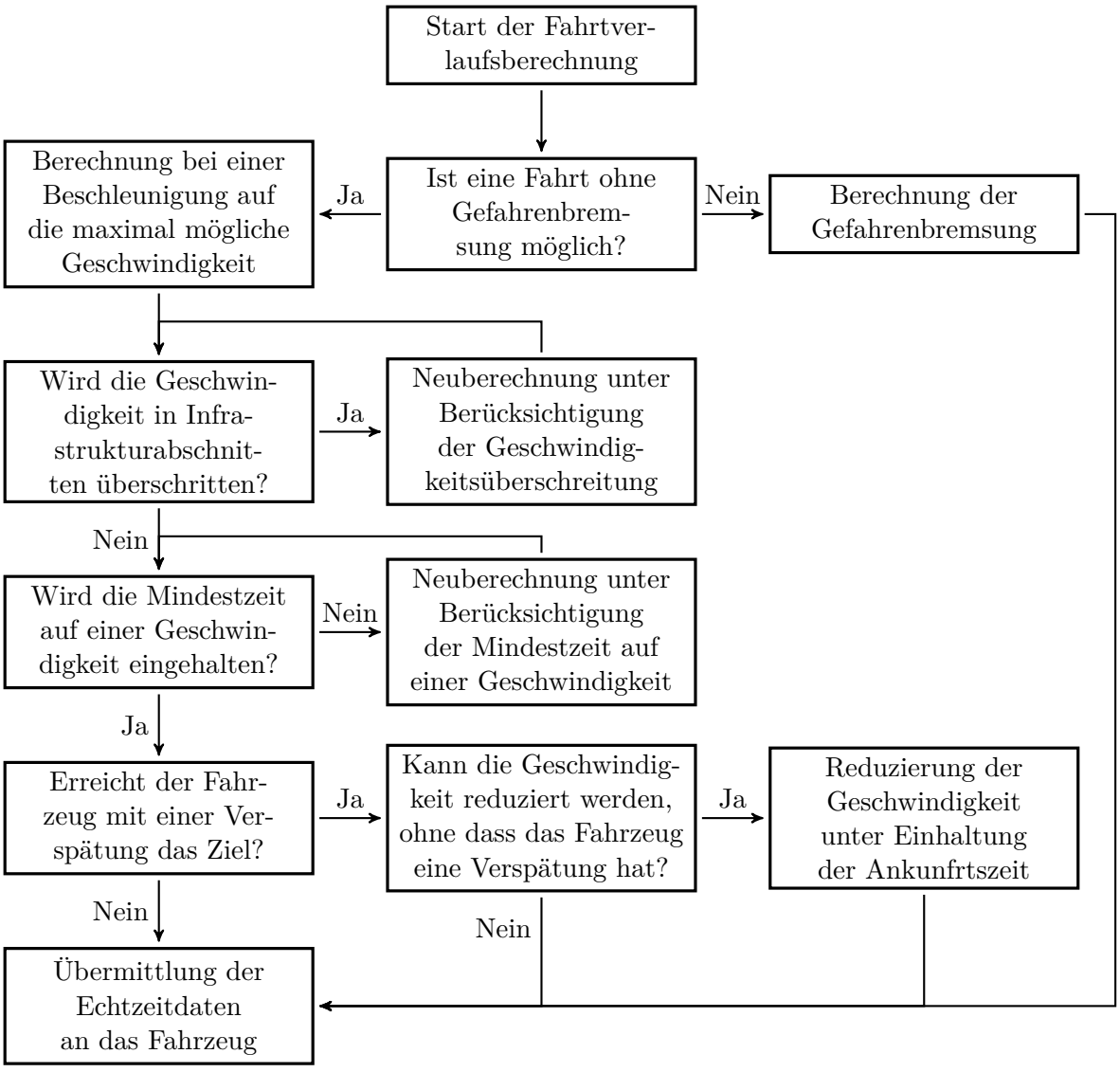

Abbildung 5: Ablaufplan der Fahrtverlaufsberechnung

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

Tabelle 7: Exemplarische Infrastrukturabschnitte

nen und die Allgemeingültigkeit des Algorithmus gezeigt werden können und treten so im EBuEf nicht auf. Als exemplarisch gewählte Zugdaten wurden die in Tabelle **??** definierten Daten verwendet. Die zuvor ermittelten nächsten Infrastrukturabschnitte 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 Infrastrukturabschnitt eingefahren ist. In Darstellung **??** sind die Infrastrukturabschnitte dargestellt, so wie sie von dem Fahrzeug ermittelt wurden. Dabei werden alle Abschnitte, die das Fahrzeug schon durchfahren hat oder hinter dem Zielabschnitt liegen nicht dargestellt. Zudem wird in dem aktuellen Abshcnitt die relative Position von der Länge abgezogen und in der Zielabschnitt wird nur bis zur relativen Zielposition abgebildet. Dementsprechend ist der erste Abschnitt in der Darstellung **??** der Abschnitt mit der ID 1001. Dieser hat aufgrund der aktuellen relativen Position des Fahrzeugs eine Länge von 290 m. Und der letzte Abschnitt ist der Abschnitt mit der ID 1010 und einer Länge von ebenfalls 290 m. Bei der Berücksichtigung der Fahrzeuglänge wird durch alle Infrastrukturabschnitt iteriert und die Zuglänge auf die Länge das Abschnitts addiert. Von dieser neu ermittelten Endposition des Abschnitts wird überprüft, ob zwischen der vorherigen Endposition und der neu ermittelten Endposition ein Infrastrukturabschnitt liegt, dessen zulässige Höchstgeschwindigkeit geringer ist, als die des ursprünglichen Abschnitts. Wenn dieser Fall eintritt, wird der Abschnitt nur so weit verlängert, sodass keine Höchstgeschwindigkeit der folgenden Abschnitte überschritten wird. Von der neu ermittelten Endposition wird überprüft, in welchem Abschnitt diese liegt und mit dem Abschnitt wird dann weiter gerechnet. Sobald der Ziel-Abschnitt erreicht wurde, wird die Schleife abgebrochen. Die neu ermittelten Abschnitte werden in den Arrays *$next_lengths_mod* und *$next_v_max_mod* abgespeichert. Durch diese Algorithmus kann es dazu kommen, dass sich die Anzahl der Abschnitte verändert hat. Dementsprechend können die Abschnitte nicht mehr eindeutig mit der Infrastruktur-ID bezeichnet werden. Mittels *$next_lengths_mod* und *$next_v_max_mod* werden mit der Funktion *createCumulativeSections*() für jeden Abschnitt die absolute Start- und Endpositi-

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

Tabelle 8: Exemplarische Zugdaten

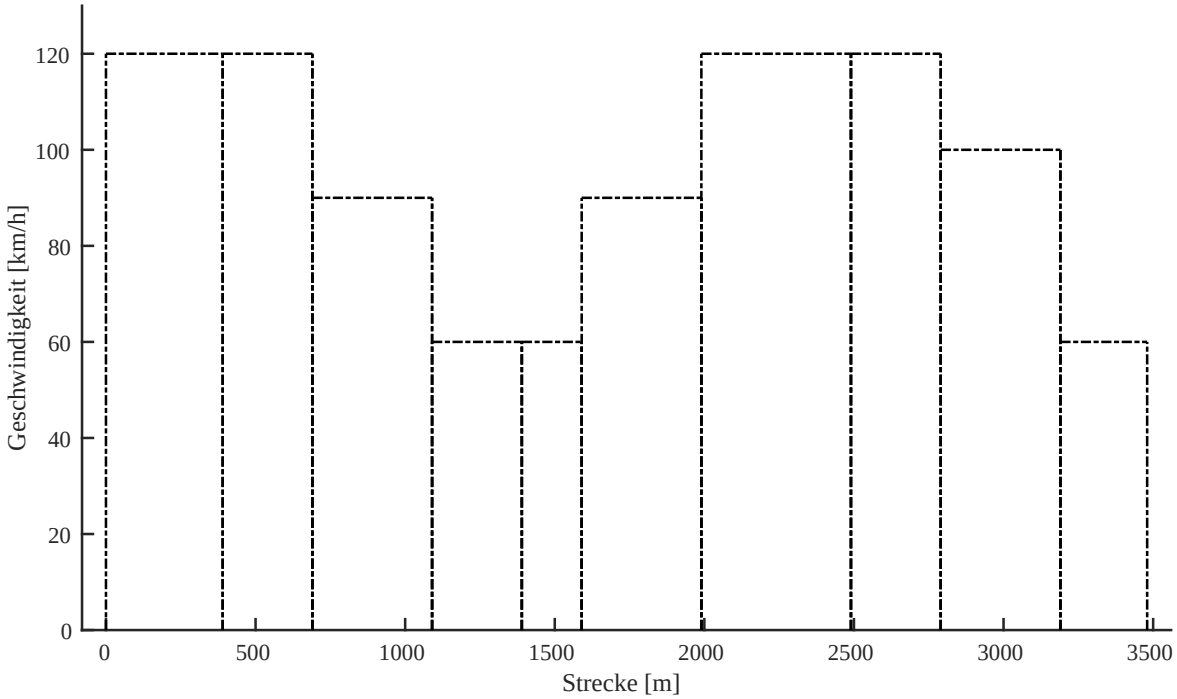

Abbildung 6: Darstellung der Infrastrukturabschnitte und die zugehörige Höchstgeschwindigkeit

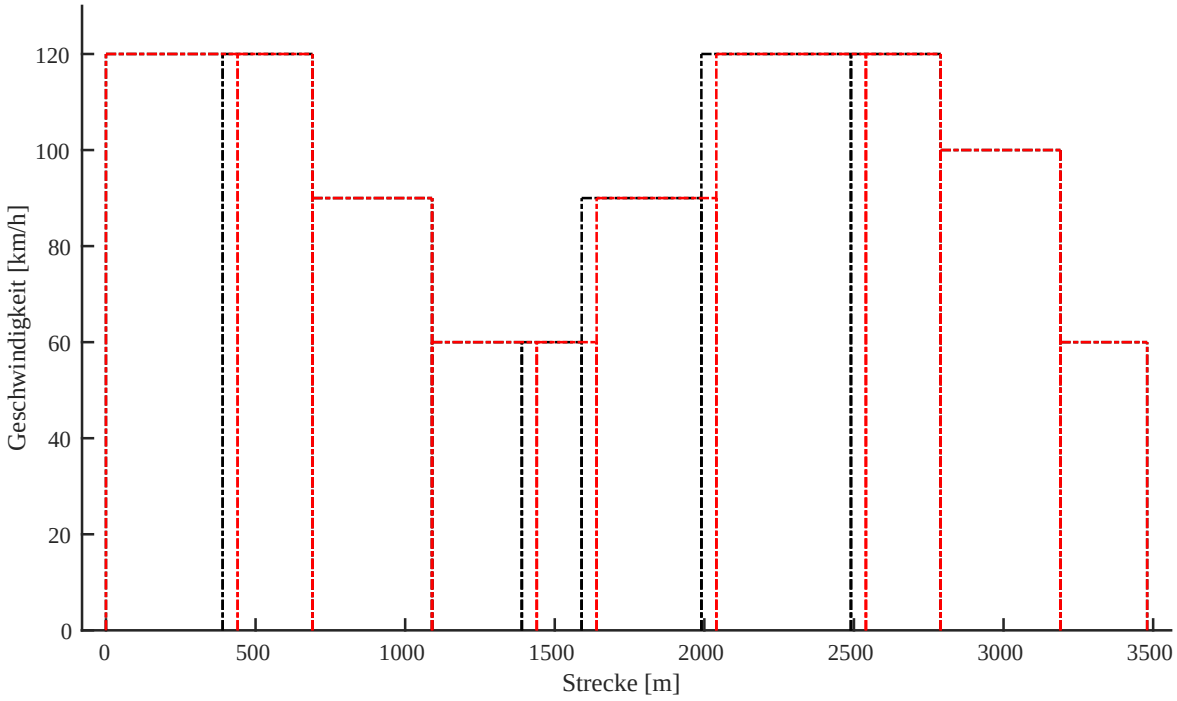

Abbildung 7: Darstellung Infrastrukturabschnitte und die zugehörige Höchstgeschwindigkeit unter Berücksichtigung der Fahrzeuglänge

on in den Arrays *$cumulativeSectionLengthStartMod* und *$cumulativeSectionLengthEndMod* gespeichert. Diese Umwandlung ist essentiell für die Überprüfung, in welchem Abschnitt ein Fahrzeug sich aktuell befindet. Die neu berechneten Abschnitte werden sind in der Darstellung **??** in rot abgebildet und beschreiben die maximale Geschwindigkeit, die ein Fahrzeug fahren darf an der jeweiligen Position.

4.2 Berechnung bei einer Beschleunigung auf die maximal mögliche Geschwindigkeit

Im ersten Schritt für die Distanz zwischen der aktuellen Position und der Ziel-Position mittels *$cumulativeSectionLengthStart*, *$cumulativeSectionLengthEnd*, *$indexCurrentSection* und *$indexTargetSection* berechnet. Für diese Distanz und die Startgeschwindigkeit wird mit Hilfe der Funktion *getVMaxBetweenTwoPoints*() (Code-Beispiel **??**) die maximale Geschwindigkeit ermittelt, die das Fahrzeug aufnehmen kann, um noch bis zum Ziel rechtzeitig bremsen zu können. Dabei wird in 10 km/h-Schritten iteriert und der maximale Wert zurückgegeben. Innerhalb der Funktion wir die Funktion *getBrakeDistance*() (Code-Beispiel **??**) aufgerufen, welche die benötigte Distanz für eine Beschleunigung bzw. Verzögerung berechnet. Durch die gegebene Startgeschwindigkeit und die höchstmögliche Geschwindigkeit wird ein erster Fahrtverlauf berechnet. Dabei werden zwei *$keyPoints* erzeugt. Mithilfe der Funktion *createTrainChanges*() wird aus diesen beiden *$keyPoints* für jede Geschwindigkeitsveränderung die aktuelle absolute Position und Geschwindigkeit ermittelt. An den Positionen, an den das Fahrzeug eine konstante Geschwindigkeit hat, wird in 1 Meter Abständen die absolute Position und die Geschwindigkeit gespeichert. Die ermittelten Daten werden in den Arrays *$trainPositionChange* und *$trainSpeedChange* gespeichert. In der Darstellung **??** ist

```php
function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1) {
  global $verzoegerung;
  global $globalFloatingPointNumbersRoundingError;
  $v_max = array();
  for ($i = 0; $i <= 120; $i = $i + 10) {
    if ((getBrakeDistance($v_0, $i, $verzoegerung) + getBrakeDistance($i, $v_1,
        ↪ $verzoegerung)) < ($distance + $globalFloatingPointNumbersRoundingError)) {
      array_push($v_max, $i);
    }
  }
  if (sizeof($v_max) == 0) {
    if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
      echo "Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu erreichen.";
    } else {
      // TODO: Notbremsung
    }
  } else {
    if ($v_0 == $v_1 && max($v_max) < $v_0) {
      $v_max = array($v_0);
    }
  }
  return max($v_max);
}
```

Code-Beispiel 3: *getVMaxBetweenTwoPoints*()

```php
function getBrakeDistance (float $v_0, float $v_1, float $verzoegerung) {
  if ($v_0 > $v_1) {
    return $bremsweg = 0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
  } if ($v_0 < $v_1) {
    return $bremsweg = -0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
  } else {
    return 0;
  }
}
```

Code-Beispiel 4: *getBrakeDistance*()

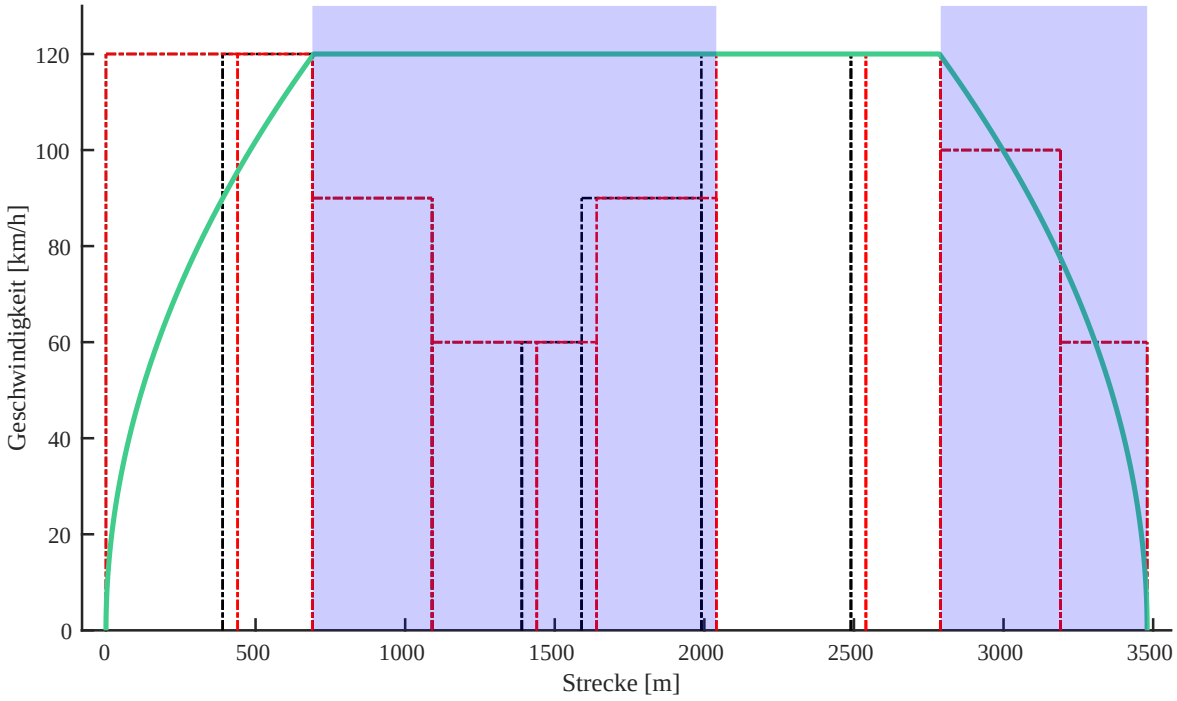

Abbildung 8: Fahrtverlaufberechnung (1. Iteration)

das Ergebnis der 1. Iteration abgebildet.

4.3 Überprüfung des Fahrtverlaufs nach Geschwindigkeitsüberschreitungen

Für die Überprüfung, ob bei einem Fahrtverlauf in manchen Infrastrukturabschnitten die zulässige Höchstgeschwindigkeit überschritten wird, wird nach jeder Berechnung die Funktion *checkIfTrainIsToFastInCertainSections*() (Code-Beispiel **??**) aufgerufen. In dieser Funktion wird über alle absoluten Positionen (*$trainPositionChange*) iteriert, überprüft in welchem Abschnitt sich diese Position befindet und überprüft, ob die zugehörige Geschwindigkeit aus dem *$trainSpeedChange*-Array die zulässige Höchstgeschwindigkeit überschreitet. Sobald in einem Abschnitt eine Geschwindigkeitsüberschreitung vorliegt, wird der zugehörige Index des Abschnitts in dem *$faildSections*-Array gespeichert. Diese Abschnitte sind in der Darstellung **??** Lila hinterlegt. Als Rückgabewert der Funktion wird wird ein Array wiedergegeben, welches abspeichert, ob es zu einer Geschwindigkeitsüberschreitung gekommen ist (*"failed"*) und wenn das der Fall ist auch die Indexe der Abschnitte (*"failed_sections"*).

4.4 Neuberechnung unter Berücksichtigung der Geschwindigkeitsüberschreitung

In dem Fall, dass es zu einer Geschwindigkeitsüberschreitung gekommen ist, wird der Fahrtverlauf neu berechnet. Als Grundlage dafür diesen die *"failed_sections"* aus der *checkIfTrain IsToFastInCertainSections*() Funktion (Code-Beispiel **??**). Die Funktion *recalculateKeyPoints*() vergleicht immer zwei benachbarte *$keyPoints* und berechnet in dem Fall einer Geschwindigkeitsüberschreitung mit der Funktion *checkBetweenTwoKeyPoints*() diese neu. In dem Fall,

```php
function checkIfTrainIsToFastInCertainSections() {
  global $trainPositionChange;
  global $trainSpeedChange;
  global $cumulativeSectionLengthStartMod;
  global $next_v_max_mod;
  global $indexTargetSectionMod;
  $faildSections = array();
  foreach ($trainPositionChange as $trainPositionChangeKey => $trainPositionChangeValue) {
    foreach ($cumulativeSectionLengthStartMod as $cumulativeSectionLengthStartKey => $cumulativeSectionLengthStartValue) {
      if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
        if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[$cumulativeSectionLengthStartKey - 1]) {
          array_push($faildSections, ($cumulativeSectionLengthStartKey -1));
        }
        break;
      } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
        if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
          if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[$cumulativeSectionLengthStartKey]) {
            array_push($faildSections, $cumulativeSectionLengthStartKey);
          }
          break;
        }
      }
    }
  }
  if (sizeof($faildSections) == 0) {
    return array("failed" => false);
  } else {
    return array("failed" => true, "failed_sections" => array_unique($faildSections));
  }
}
```

Code-Beispiel 5: *checkIfTrainIsToFastInCertainSections()*

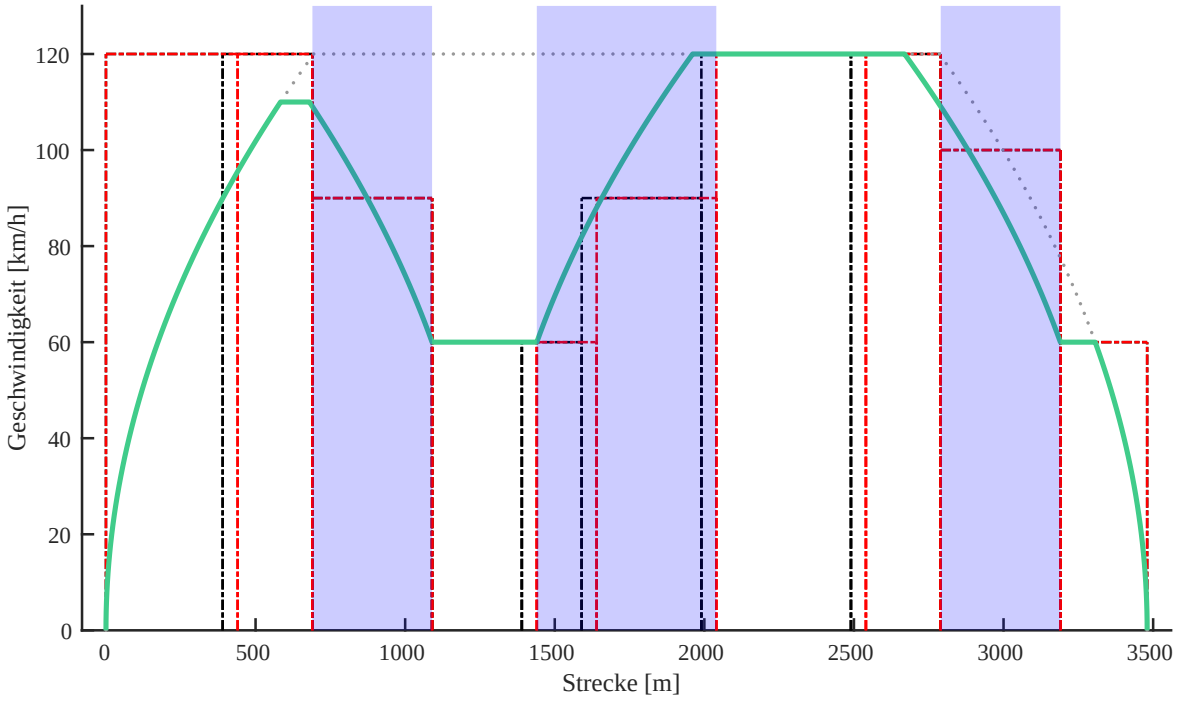

Abbildung 9: Fahrtverlaufberechnung (2. Iteration)

dass zwischen zwei benachbarten *$keyPoints* die zulässige Höchstgeschwindigkeit überschritten wird, wird die absolute Start- und End-Position dieser Geschwindigkeitsüberschreitung gespeichert. Im folgenden Schritt wird wie in dem Abschnitt **??** zwischen den Start-Werten des ersten *$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 es immer eine gerade anzahl an *$keyPoints* gibt und somit in jedem Iterationsschritt zwei benachbarte *$keyPoints* verglichen werden können. Nachdem alle *$keyPoint*-Paare überprüft werden, werden mit Hilfe der *createTrainChanges*() Funktion die Arrays *$trainPositionChange* und *$trainSpeedChange* erzeugt. Dieser neu berechnete Fahrtverlauf wird dann wieder der Funktion *checkIfTrain IsToFastInCertainSections*() Funktion (Code-Beispiel **??**) übergeben. Dieser Prozess wird solange durchlaufen, bis es zu keiner Geschwindigkeitsüberschreitung mehr kommt. In den folgenden Abbildungen (Darstellung **??**, **??** und **??**) werden die Ergebnisse der einzelnen Iterationsschritte visuell abgebildet, wobei die grau gepunkteten Linien die Ergebnisse der vorherigen Iterationsschritte darstellen.

4.5 Einhaltung der Mindestzeit auf einer Geschwindigkeit

1. Ideal: möglichst späte v reduzieren

Für eine möglichst realitätsnahe Simulation kann über die Variable *$globalTimeOnOneSpeed* in der Datei *globalVariables.php* eine Mindestzeit festgelegt werden, die ein Fahrzeug auf einer Geschwindigkeit mindestens einhalten muss. Ebenfalls kann über die Variablen *$useMinTimeOnSpeed* und *$errorMinTimeOnSpeed* festgelegt werden, ob die Funktion aktiviert sein

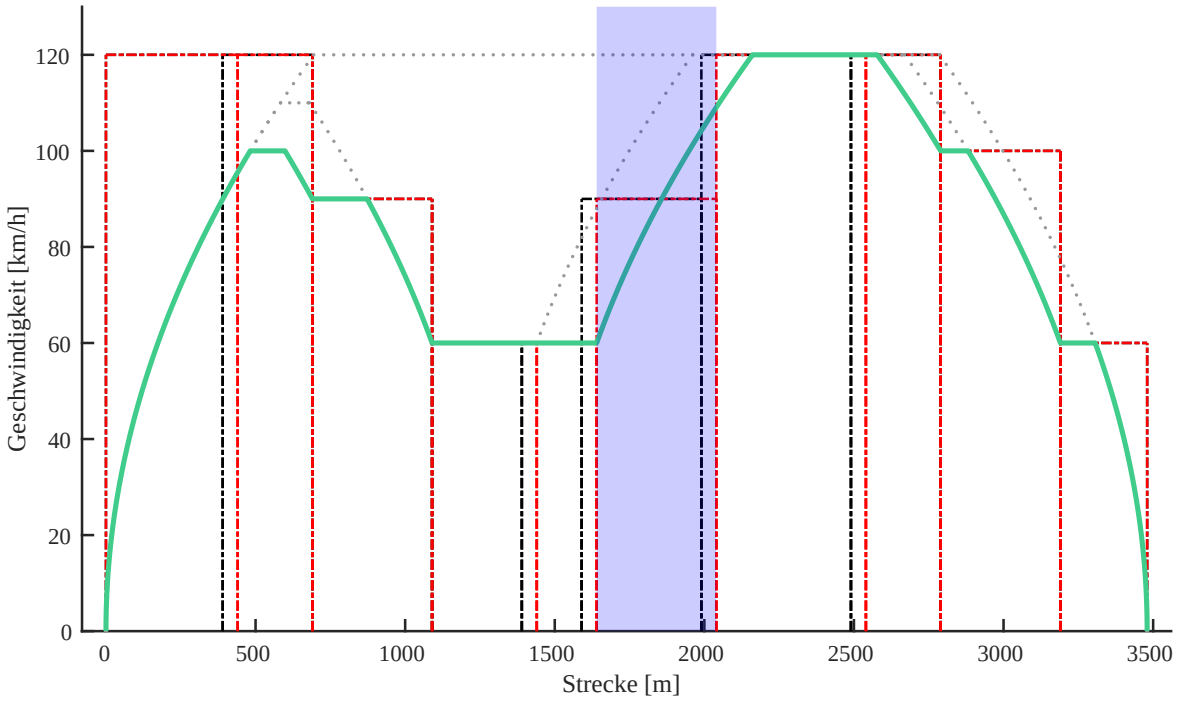

Abbildung 10: Fahrtverlaufberechnung (3. Iteration)

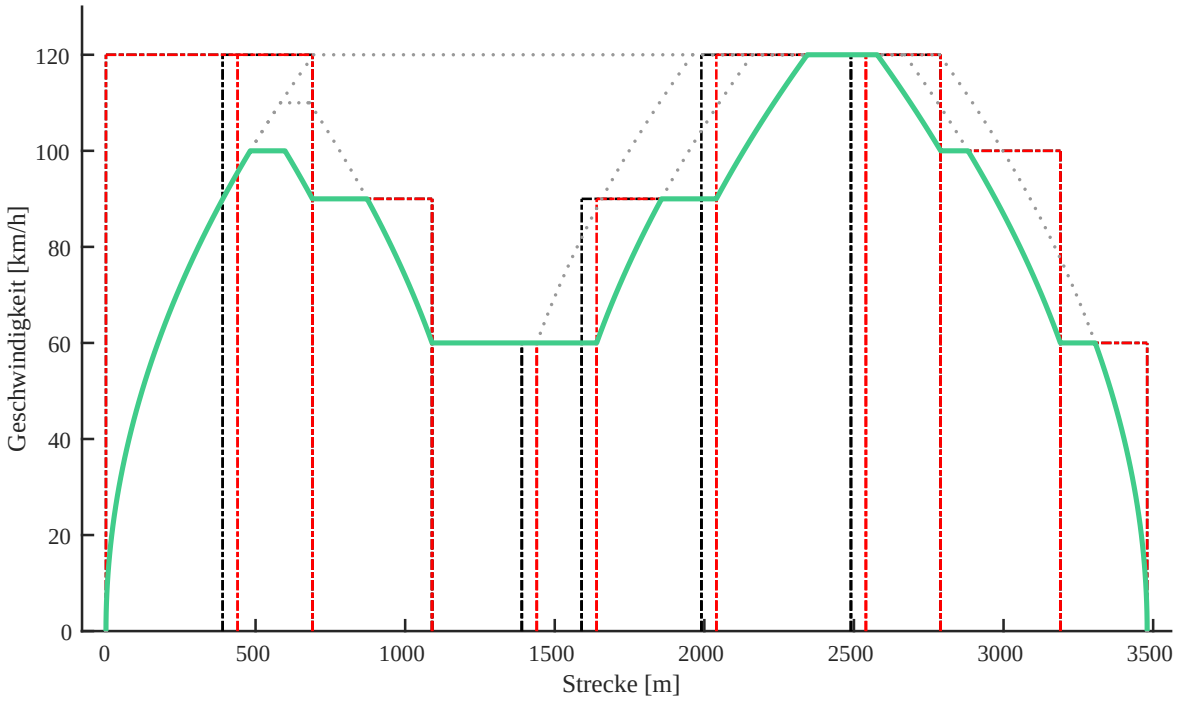

Abbildung 11: Fahrtverlaufberechnung (4. Iteration)

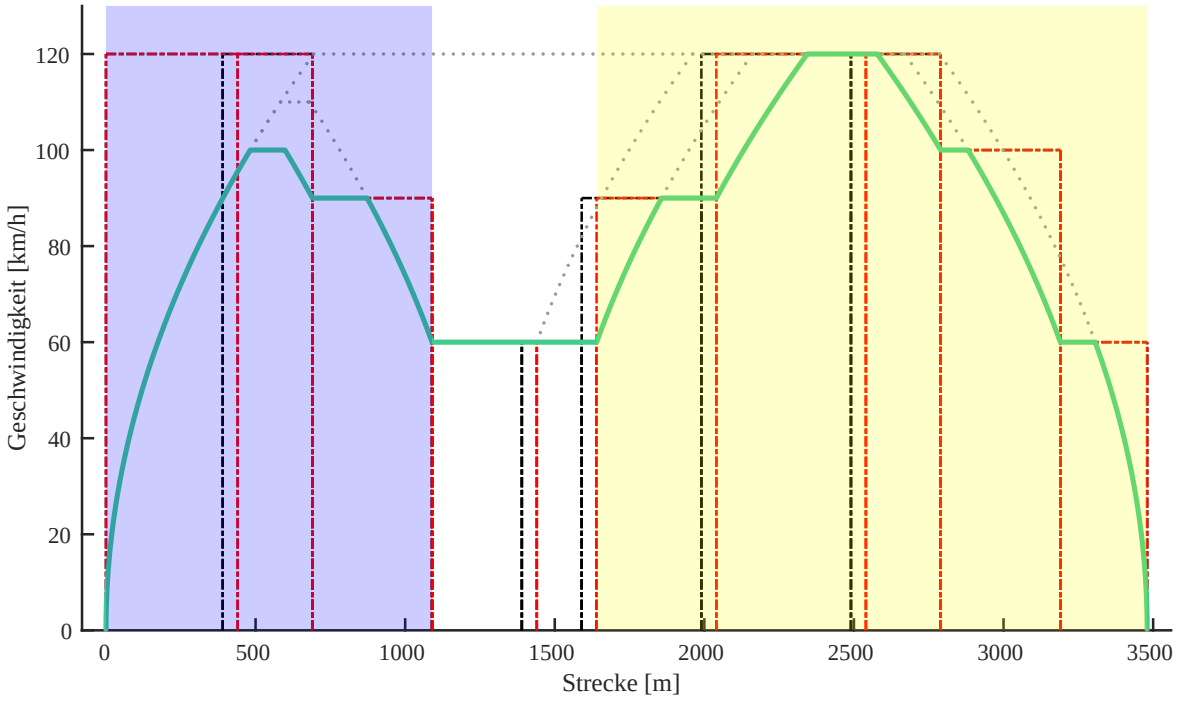

Abbildung 12: Einteilung des Fahrtverlaufs in *$subsections*

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 $v > 0$ hat. Wenn auf einem Abschnitt die Mindestzeit nicht eingehalten werden kann, kann eine Beschleunigung später eingeleitet werden, eine Verzögerung vorzeitiger eingeleitet werden oder auf eine kleinere Geschwindigkeit beschleunigt werden. In dem folgenden Algorithmus werden die ... 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 **??** wurde der exemplarische Fahrtverlauf somit in zwei *$subsection* unterteilt, welche Lila bzw. Gelb hinterlegt sind. Diese Einteilung wird vorgenommen, da sich die Verschiebung einer Beschleunigung bzw. Verzögerung auf die folgenden bzw. vorherigen Abschnitte auswirkt. Durch diese Einteilung kann verhindert werden, dass es dadurch zu Konflikten kommt. Falls die Beschleunigungen bzw. Verzögerungen soweit nach hinten bzw. nach vorne verschoben werden müssen, kann die maximale Geschwindigkeit auf dieser *$subsection* reduziert werden und die zur Verfügung stehende Strecke vergrößert werden. Wie in Darstellung **??** 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 dem 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*() erstellt wurden und mit der Funktion *array_reverse*() in umgekerte Reihenfolge in dem Array *$subsection_list* gesammelt wurden, wird für jede *$subsection* überprüft, ob die Beschleunigungen

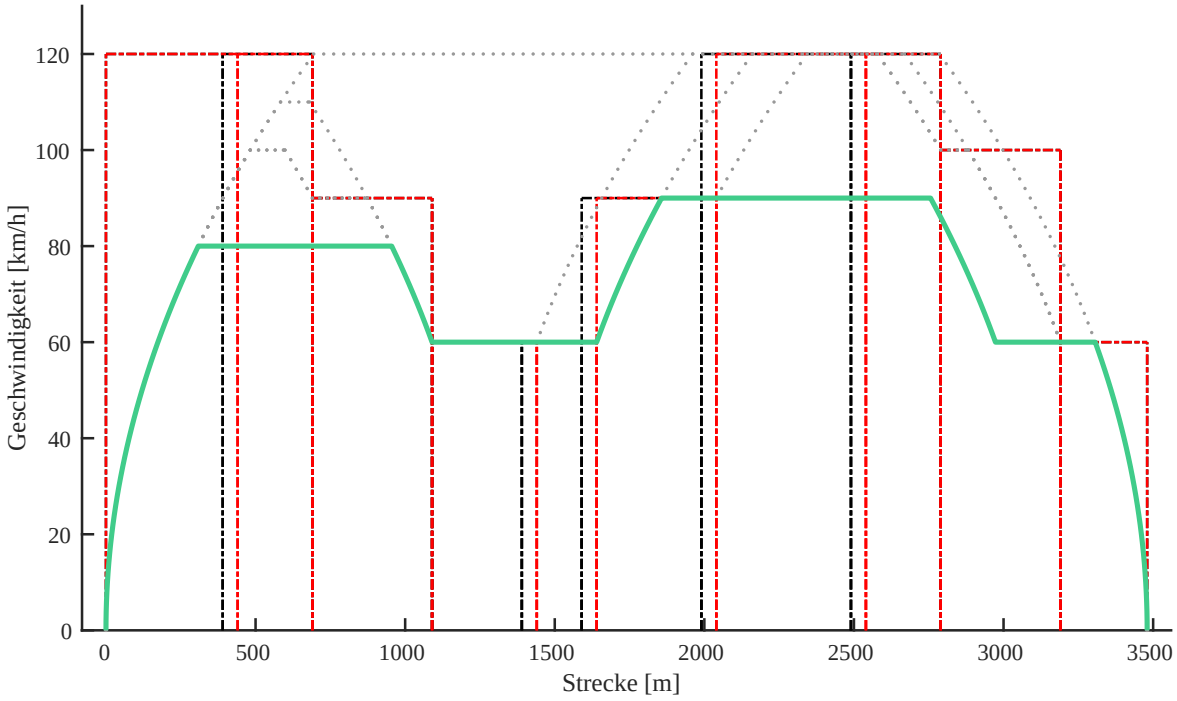

Abbildung 13: Fahrtverlauf unter Einhaltung der Mindestzeit

bzw. Verzögerungen verschoben werden können. Dabei wird über alle konstanten Geschwindigkeiten iteriert, überprüft, ob die Mindestzeit eingehalten wird und wenn das nicht der Fall ist, wird überprüft, ob eine Verschiebung möglich ist. Sollte bei einer Verschiebung die *position_1* des *$keyPoints* hinter *position_0* des zweiten *$keyPoints* liegen (bei einer Beschleunigung), wird der zweite *$keyPoint* gelöscht. Gleiches geschieht bei der Verzögerung in umgekehrter Reihenfolge. Nach der Verschiebung wird überprüft, ob auf allen konstanten Geschwindigkeit die Mindestzeit eingehalten wird. Wenn das der Fall ist, wird die nächste *$subsection* überprüft. In dem Fall, dass durch die Verschiebung die Mindestzeit nicht eingehalten werden kann, wird die maximale Geschwindigkeit auf dieser *$subsection* um $10km/h$ 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 **??** 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.

4.6 Berücksichtigung der Ankunftszeit bei der Berechnung des Fahrtverlaufs

Der berechnete Fahrtverlauf in den Kapiteln **??**, **??**, **??** und **??** ermittelt die frühstmögliche Ankunftszeit am Ziel. In dem Fall, dass der Zug dadurch mit einer Verspätung am Ziel ankommt wird der Fahrtverlauf an das Fahrzeug übergeben. Falls der Zug allerdings 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

Index	Funktion
max_index	ndex des *$keyPoints* mit der Beschleunigung auf die maximale Geschwindigkeit in der *$subsection*
indexes	Indexe aller beinhalteten *$keyPoints*
is_prev_section	Berücksichtigung des Abschnitts vor der *$subsection*
is_next_section	Berücksichtigung des Abschnitts nach der *$subsection*
failed	Unterschreitung der Mindestzeit auf der *$subsection*

Tabelle 9: Aufbau des *$subsection*-Arrays

Verspätung am Ziel ankommt. Ergebnis ist in **??** abgebildet. Im ersten Schritt wird mittels der Funktion *checkIfTheSpeedCanBeDecreased*() 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* - jenachdem, 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* wiedergegeben, welches die Einträge *possible* und *range* enthält und als *$returnSpeedDecrease* abgespeichert. Der Eintrag *possible* gibt an, ob das Fahrzeug auf dem gesamten Fahrtverlauf die Geschwindigkeit reduzieren könnte und wird als Boolescher Wert (*true*/*false*) abgespeichert und und in dem *Array range* werden alle Indexe der möglichen *$keyPoints* inklusive der ermittelten Geschwindigkeiten abgespeichert. In dem in Abbildung **??** dargestellten Fahrtverlauf wären so 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 *$keyPoint* mit dem Index 2 die Geschwindigkeiten 60, 70, 80 und 90 km/h. Wenn eine Reduzierung der Geschwindigkeit möglich ist, wird in einer *while*-Schleife versucht die Geschwindigkeit zu reduzieren, bis das Fahrzeug bei der nächsten Reduzierung mit einer Verspätung am Ziel ankommen würde oder eine weitere Reduzierung nicht möglich ist, da die Maximalgeschwindigkeit auf dem Fahrtverlauf 10 km/h beträgt. Innerhalb der *while*-Schleife ermittelt die Funktion *findMaxSpeed*() aus dem *$returnSpeedDecrease*-Array den *$keyPoint* mit der höchsten Geschwindigkeit. Für den Fall, dass mehrere *$keyPoints* die selbe Höchstgeschwindigkeit haben, wird der letzte dieser *$keyPoints* ermittelt. Im Anschluss wird mit einer *for*-Schleife in 10er-Schritten in absteigender Reihenfolge über die möglichen Geschwindigkeiten iteriert und überprüft, ob durch die Anpassung die Ankunftszeit eingehalten werden kann. Sobald die Ankunftszeit nicht eingehalten werden kann, werden die *$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*() erneut aufgerufen. Das Ergebnis dieser Berechnung ist in der Abbildung **??** zu sehen.

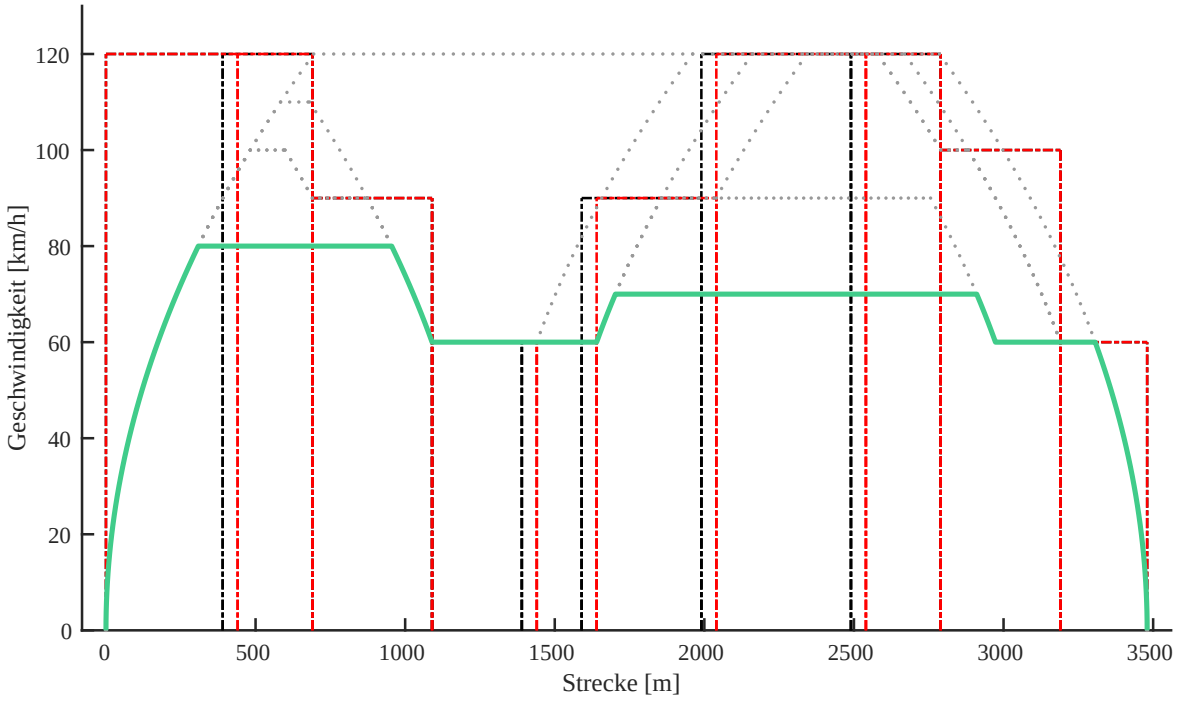

Abbildung 14: Fahrtverlauf mit reduzierter Geschwindigkeit unter Einhaltung der Ankunftszeit

4.7 Berücksichtigung der exakten Ankunftszeit bei der Berechnung des Fahrtverlaufs

Die in Kapitel **??** 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 noch 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 *$keyPoints* vorzeitiger eingeleitet werden kann. Sollte die Zielgeschwindigkeit der Verzögerung 0 km/h sein, wird die Verzögerung unterteilt in eine Verzögerung auf 10 km/h und eine von 10 km/h auf 0 km/h. Die Position der vorzeitig eingeleiteten Verzögerung wird mittels der Funktion *speedFineTuning*() berechnet, welche als Parameter den Betrag der Differenz zwischen aktueller Soll- und Ist-Ankunftszeit und den Index des vorherigen *$keyPoints* übergeben bekommt. In Abbildung **??** werden die Geschwindigkeiten (v_1,v_2), Strecken (s_1,s_2) und Zeiten (t_1,t_2) vor und nach der Verzögerung, welche vorzeitiger eingeleitet werden soll, um eine pünktliche Ankunft am Ziel zu ermöglichen, dargestellt und in Tabelle **??** sind die exakten Werte des exemplarischen Fahrtverlaufs aufgelistet, damit die verwendete Gleichung (Gleichung **??** aus Kapitel **??**) an diesem Beispiel angewandt werden kann. In diesem konkreten Beispiel würde das Fahrzeug 3,31 s (t_Δ) zu früh an der Haltestelle ankommen, wodurch das Fahrzeug für die Zurücklegung der Strecken s_1 und s_2 insgesamt 85,42s ($t_{ges} = t_1 + t_2 + t_\Delta$) zur Verfügung hat. Durch das Einsetzen dieser Werte in die Gleichung **??** aus dem Kapitel **??** ergibt sich für t_3 (t_3,

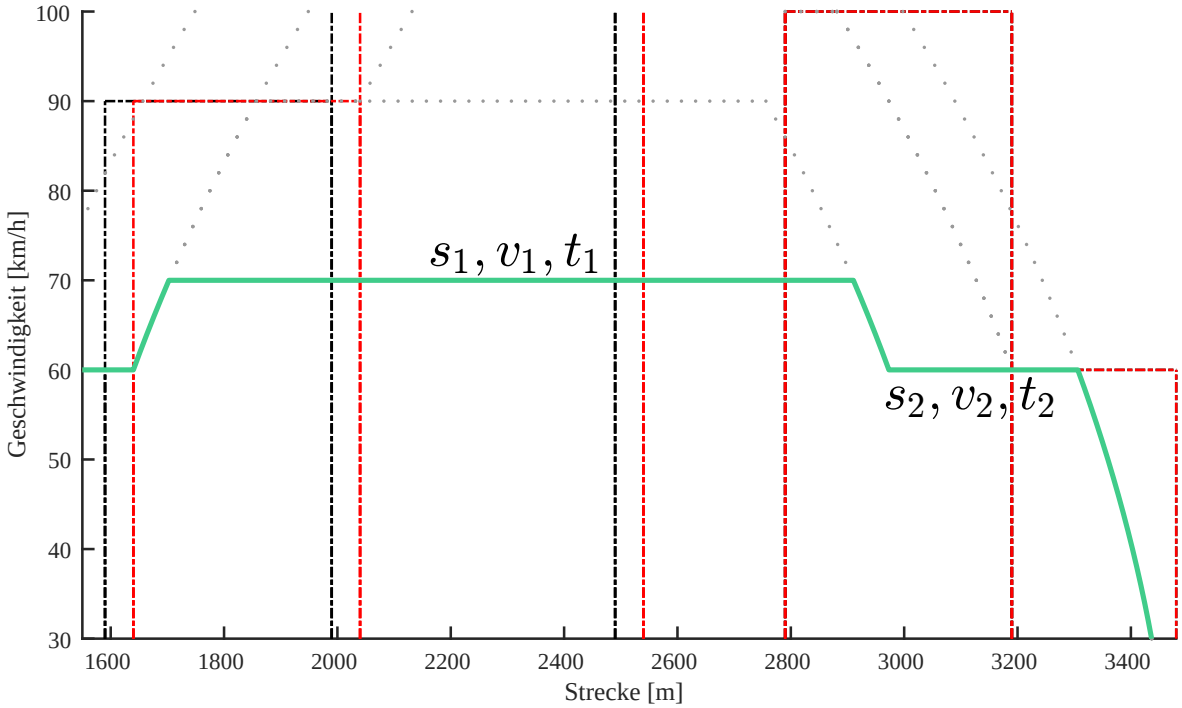

Abbildung 15: speedFineTuning_1

v_1	70 km/h (19,44m/s)
v_2	60 km/h (16,67m/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_Δ	3,31 s
t_{ges}	85,42 s

Tabelle 10: Geschwindigkeiten, Strecken und Zeiten vor und nach der Verzögerung

$$t_3 = \frac{1541m - 16,67m/s \cdot 85,42s}{19,44m/s - 16,67m/s}$$
$$t_3 = 42,26s$$

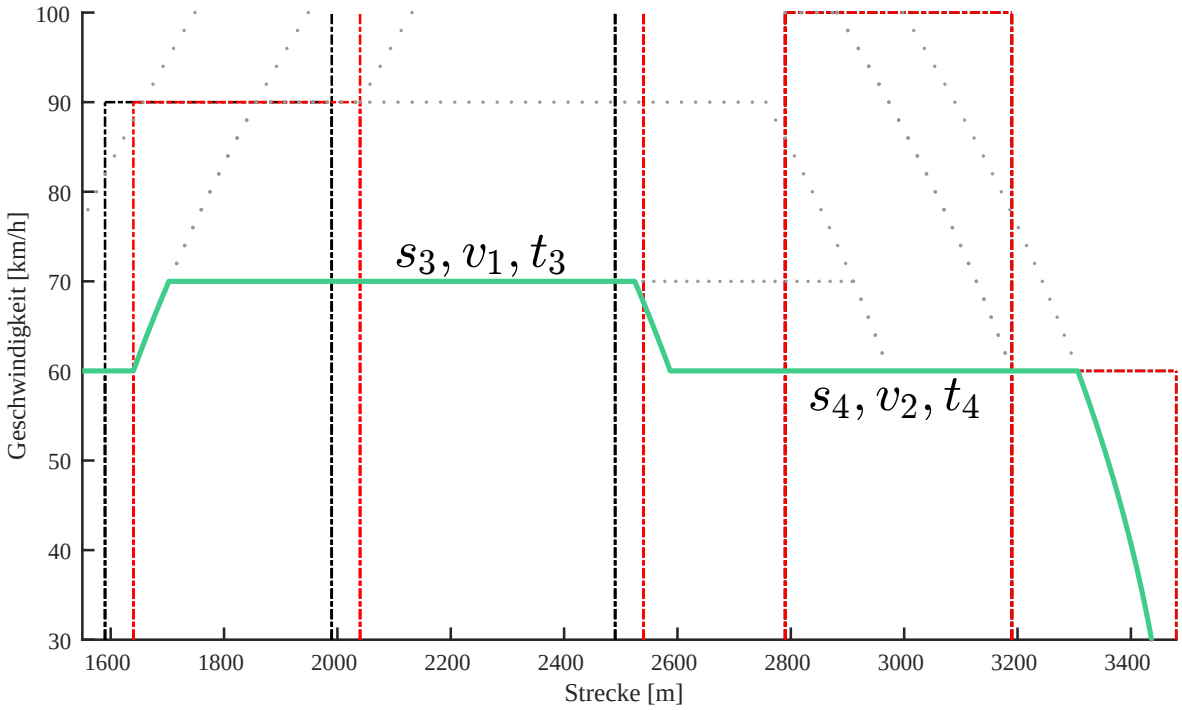

Abbildung 16: speedFineTuning_2

t_4, s_3 und s_4 bezeichnen die Strecken und Zeiten nach der Anpassung) ein Wert von 42,2s. Dementsprechend muss die Verzögerung 19,85s (t_1 - t_3) früher eingeleitet werden Die vorzeitige Einleitung der Verzögerung sorgt dafür, dass das Fahrzeug seinen nächsten Haltepunkt genau pünktlich erreicht und ist in Abbildung **??** dargestellt, wobei durch die gepunktete Linie der Fahrtverlauf vor der Anpassung zu sehen ist. Die neu berechneten Werte sind in Tabelle **??** aufgelistet. Der finale Fahrtverlauf ist in Abbildung **??** dargestellt und kann so dem Fahrzeug übergeben werden.

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

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

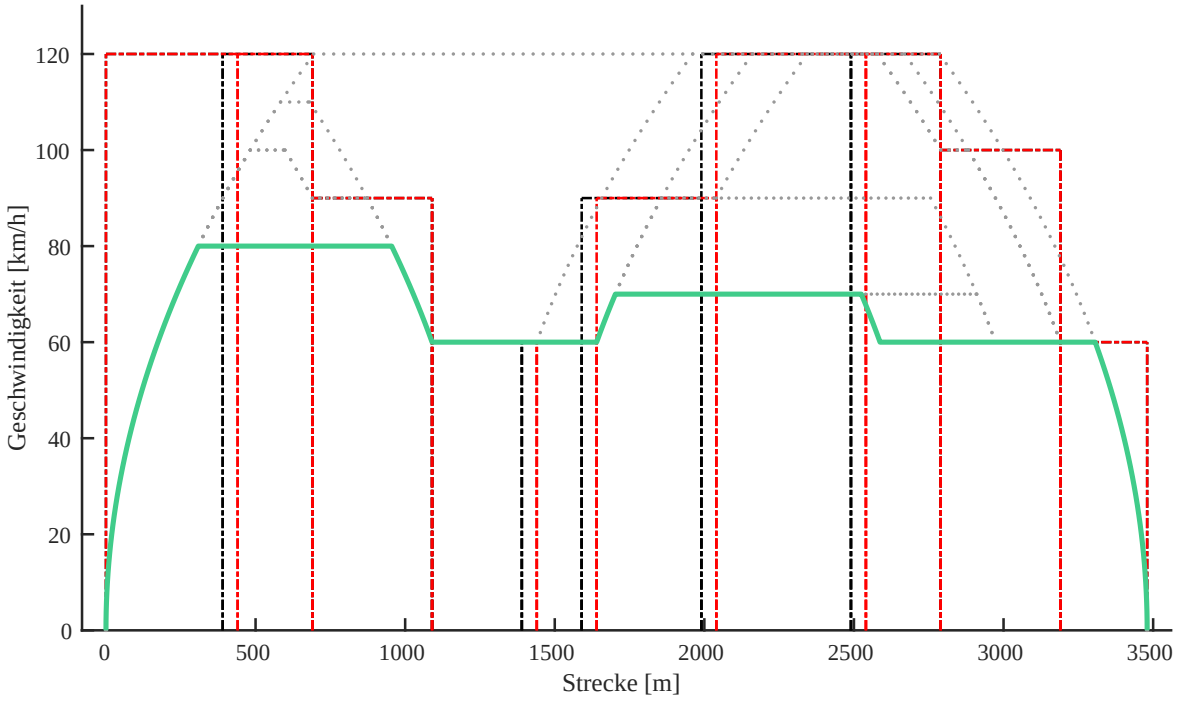

Abbildung 17: Finaler Fahrtverlauf

4.8 Einleitung einer Gefahrenbremsung

Eine Gefahrenbremsung wird eingeleitet, sobald ein Fahrzeug bei einer sofortigen Verzögerung ein auf Halt stehendes Signal überfahren würde, in einem Infrastruktur-Abschnitt die zulässige Höchstgeschwindigkeit überschreiten würde oder an dem nächsten planmäßigen Halt nicht rechtzeitig zum stehen kommen würde. Bei einer Gefahrenbremsung wird mit einer Notbremsverzögerung von $2m/s^2$ abgebremst. Dieser Wert kann in der Datei *globalVariables.php* über die Variable *$globalNotverzoegerung* 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: $v \geq 10km/h$, nach der Geschwindigkeit von $10km/h$ direkt die Geschwindigkeit von $0km/h$ 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 $v < 10km/h$
3. Fahrzeug hat bei der Gefahrenstelle eine Geschwindigkeit von $v \geq 10km/h$

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

Fahrzeug an der Gefahrenstelle hat. Für diese Berechnung wird die Gleichung **??** aus dem Kapitel **??** verwendet. Sollte das Fahrzeug an der Gefahrenstelle eine Geschwindigkeit von $v \geq 10km/h$ haben, bremst das Fahrzeug in $2km/h$-Schritten auf $10km/h$ ab und bekommt nach der Übermittlung der $10km/h$ direkt $0km/h$ übergeben. In dem Fall, dass das Fahrzeug an der Gefahrenstelle langsamer als $10km/h$ ist, bremst das Fahrzeug wie im 1. Fall in $2km/h$-Schritten auf $0km/h$ ab. Bei einer Gefahrenbremsung bekommt das jeweilige Fahrzeug eine Fehlermeldung übermittelt und wird nicht weiterfahren. Das liegt daran, dass durch die Gefahrenbremsung keine genaue Positionsbestimmung vorgenommen werden kann. Damit das Fahrzeug wieder seinen Fahrtbetrieb 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.

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

Tabelle 12: *$keyPoints* am Beispiel der Fahrt von XAB nach XZO

5 Beispielrechnung eines Fahrtverlaufs im EBuEf

Die in Kapitel **??** 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, nachdem 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 Infrastrukturabschnitt 1189, hat die Fahrtrichtung 1 und die Fahrstraße ist so eingestellt, dass das Fahrzeug bis zum Ausfahrsignal im Bahnhof Zoo fahren kann, und dort im Infrastrukturabschnitt 1178 zum Stehen kommen kann. Somit beträgt die Strecke bis zum nächsten Halt 672m und das Fahrzeug hat 115s zur Verfügung. Die Bremsverzögerung des Fahrzeugs beträgt 0,8m/s^2.

Für die Fahrt wurde eine Mindestzeit von 20s festgelegt, welche das Fahrzeug auf einer Geschwindigkeit mindestens fahren muss (*$globalTimeOnOneSpeed = 20*), den Optionen *$useSpeedFineTuning*, *$useMinTimeOnSpeed* und *$slowDownIfTooEarly* wurde der Wert *true* zugewiesen und der Option *$errorMinTimeOnSpeed* der Wert *false*.

In der Tabelle **??** sind die berechneten *$keyPoints* aufgelistet, welche durch die Berechnung des Fahrtverlaufs ermittelt wurden, und in der Darstellung **??** ist der Fahrtverlauf visuell dargestellt. Bei der Berechnung des Fahrtverlaufs wurde laut der Fahrzeugsteuerung die Ankunftszeit exakt eingehalten. Die Zeit-Werte der *$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 **??**) lässt sich der Fahrtverlauf in 5 Abschnitte einteilen. Die Start- und Zielgeschwindigkeit, die Strecke und die Zeit der einzelnen Abschnitt sind in der Tabelle **??** aufgelistet und werden mittels der Formeln aus Kapitel **??** 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:

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

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

Für die Berechnung werden die Strecken und Zeiten in gleichförmige und gleichmäßig be-

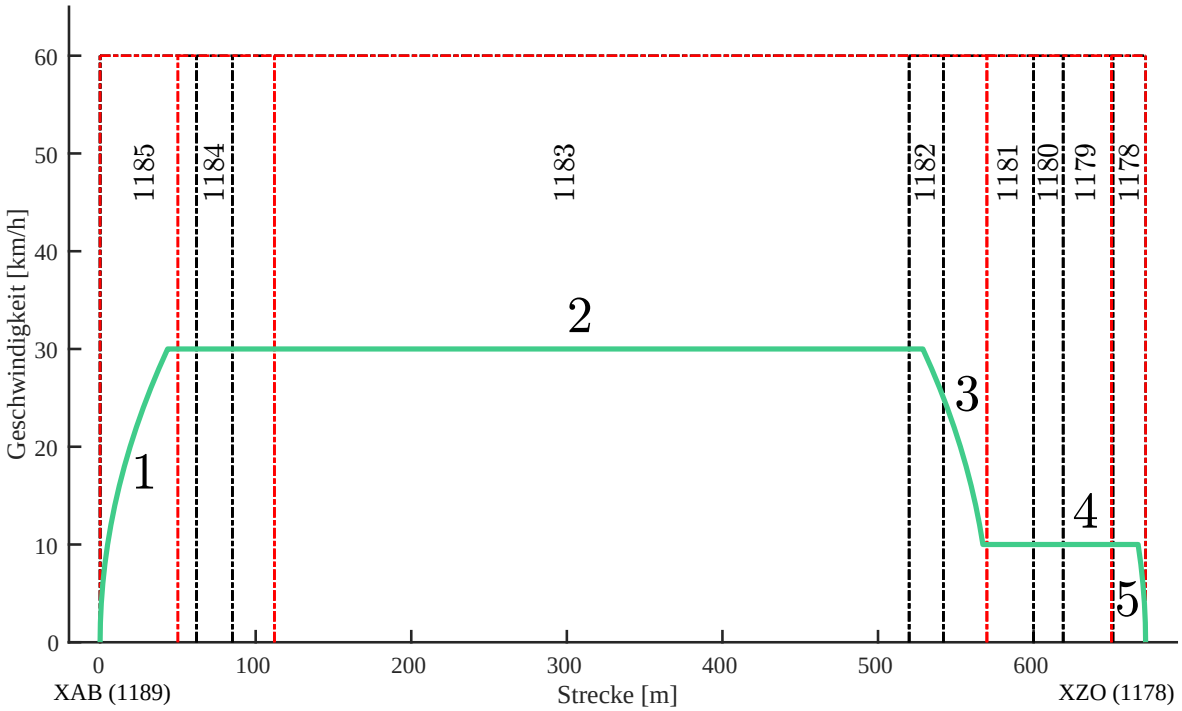

Abbildung 18: Fahrtverlauf für eine Beispielrechnung

Abschnitt	Beschleunigung/Verzögerung	v_0	v_1	Strecke	Zeit
1	ja	0 km/h	30 km/h	43,40m	10,42s
2	nein	30 km/h	30 km/h	485,43m	58,25s
3	ja	30 km/h	10 km/h	38,58m	6,94s
4	nein	10 km/h	10 km/h	99,76m	35,92s
5	ja	10 km/h	0 km/h	4,82m	3,47s
$\sum$	-	-	-	672m[1]	115s

[1] Die Werte in der Strecken-Spalte sind auf zwei Nachkommastellen gerundet und würden durch das Aufsummieren der Strecken von Abschnitt 1 bis 5 eine Gesamtstrecke von 671,99m kommen. Die angegebenen 672m entsprechen der Summe der Abschnitte 1 bis 5, ohne dass die einzelnen Strecken gerundet werden.

Tabelle 13: Fahrtverlauf am Beispiel der Fahrt von XAB nach XZO

schleunigte Bewegungen unterteilt:

$$t_{ges} = t_{gleichförmigeBewegungen} + t_{gleichmässigbeschleunigteBewegungen}$$

$$s_{ges} = s_{gleichförmigeBewegungen} + s_{gleichmässigbeschleunigteBewegungen}$$

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

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

$$t_{gleichmässigbeschleunigteBewegungen} = t_1 + t_3 + t_5$$

$$s_{gleichmässigbeschleunigteBewegungen} = s_1 + s_3 + s_5$$

Für die Beschleunigungen gilt nach den Gleichungen **??** und **??**:

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

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

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

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

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

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

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

$$t_{gleichmässigbeschleunigteBewegungen} = \frac{125}{6}s$$

$$s_{gleichmässigbeschleunigteBewegungen} = \frac{3125}{36}m$$

Und für die gleichförmigen Bewegungen bleiben dementsprechend noch:

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

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

Für die Berechnung der Strecke und Zeit, welche das Fahrzeug auf $30km/h$ fährt gilt nach der Gleichung **??**:

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

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

$$t_2 = \frac{34951}{600}s$$

$$t_2 \approx 58,25s$$

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

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

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

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

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

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

In Summe ergibt das:

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

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

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

Damit die Daten aus der Berechnung des Fahrtverlaufs von MATLAB eingelesen werden können, wurde die Funktion *safeTrainChangeToJSONFile*() (Code-Beispiel **??**) 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.

```
1  function safeTrainChangeToJSONFile(int $indexCurrentSection, int $indexTargetSection,
       ↪ int $indexCurrentSectionMod, int $indexTargetSectionMod, array
       ↪ $speedOverPositionAllIterations) {
2    global $trainPositionChange;
3    global $trainSpeedChange;
4    global $next_v_max;
5    global $cumulativeSectionLengthEnd;
6    global $next_v_max_mod;
7    global $cumulativeSectionLengthEndMod;
8
9    $speedOverPosition = array_map('toArr', $trainPositionChange, $trainSpeedChange);
10   $speedOverPosition = json_encode($speedOverPosition);
11   $fp = fopen('../json/speedOverPosition.json', 'w');
12   fwrite($fp, $speedOverPosition);
13   fclose($fp);
14
15   $v_maxFromUsedSections = array();
16   for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
17     array_push($v_maxFromUsedSections, $next_v_max[$i]);
18   }
19   $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
       ↪ $v_maxFromUsedSections);
20   $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSections);
21   $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
22   fwrite($fp, $VMaxOverPositionsJSon);
23   fclose($fp);
24
25   $v_maxFromUsedSections = array();
26   for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
27     array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
28   }
29   $VMaxOverCumulativeSectionsMod = array_map('toArr', $cumulativeSectionLengthEndMod,
       ↪ $v_maxFromUsedSections);
30   $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSectionsMod);
31   $fp = fopen('../json/VMaxOverCumulativeSectionsMod.json', 'w');
32   fwrite($fp, $VMaxOverPositionsJSon);
33   fclose($fp);
34
```

```
35    $jsonReturn = array();
36    for ($i = 0; $i < sizeof($speedOverPositionAllIterations); $i++) {
37      $iteration = array_map('toArr', $speedOverPositionAllIterations[$i][0],
          ↪ $speedOverPositionAllIterations[$i][1]);
38      array_push($jsonReturn, $iteration);
39    }
40    $speedOverPosition = json_encode($jsonReturn);
41    $fp = fopen('../json/speedOverPosition_prevIterations.json', 'w');
42    fwrite($fp, $speedOverPosition);
43    fclose($fp);
44  }
```

Code-Beispiel 6: *safeTrainChangeToJSONFile*()

Das MATLAB-Skript ist im Anhang (siehe **??**) dieser Arbeit angehängt und auf weitere Details bezüglich der Funktionsweise wird im Rahmen dieser Arbeit nicht weiter eingegangen.

7 Formeln

Für die im folgenden Abschnitt verwendeten Einheiten gilt:

a = Bremsverzoegerung $[m/s^2]$
v = Geschwindigkeit $[m/s]$
s = Strecke $[m]$
t = Zeit $[s]$

7.1 Formeln für gleichmäßig beschleunigte Bewegungen

Bei einer gleichmäßig beschleunigten Bewegung gilt:

$$a(t) = a \tag{1}$$

Für die Bestimmung der Geschwindigkeit in Abhängigkeit der Zeit, muss die Beschleunigung $a(t)$ nach der Zeit t integriert werden.[7]

$$v(t) = \int a(t)\,dt \tag{2}$$

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.

$$v(t) = a \cdot t + v_0 \tag{3}$$

Für die Bestimmung der benötigten Zeit muss die Geschwindigkeit erneut integriert werden.[8] Die dabei entstehende Integrationskonstante s_0 gibt die bereits zurückgelegte Strecke an.

$$s(t) = \int v(t)\,dt \tag{4}$$

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

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

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

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

[7] **?**, S. 20
[8] **?**, S. 20

```
function getBrakeDistance (float $v_0, float $v_1, float $verzoegerung) {
  if ($v_0 > $v_1) {
    return $bremsweg = 0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
  } else if ($v_0 < $v_1) {
    return $bremsweg = -0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung));
  } else {
    return 0;
  }
}
```

Code-Beispiel 7: *getBrakeDistance*()

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 m/s auf eine gegebenen Zielgeschwindigkeit v_1 zu beschleunigen. Bei der Berechnung des Beschleuniguns- und Abbremsverhalten 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 m/s auf v_1 zu beschleunigen und von 0 m/s auf v_0. Für die gesuchte Strecke gilt dann:

$$s(v_0, v_1) = |s(v_1) - s(v_0)| \tag{8}$$

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

In dem Programm übernimmt diese Berechnung die Funktion *getBrakeDistance()*. Damit keine negativen Rückgabewerte entstehen, wird im Falle einer Bremsung ($v_1 < v_0$) das Ergebnis mit -1 multipliziert. Beispiel eines Querverweis (**??**). 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 v_0 auf v_1 zu beschleunigen bzw. abzubremsen und aus der Differenz die benötigte Zeit berechnet.

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

In dem Programm übernimmt diese Berechnung die Funktion *getBrakeTime()*. Damit keine negativen Rückgabewerte entstehen, wird im Falle einer Bremsung ($v_1 < v_0$) das Ergebnis mit -1 multipliziert. Für die Berechnung einer Gefahrenbremsung ist es notwendig zu wissen, welche Geschwindigkeit das Fahrzeug an der Stelle der Gefahrenstelle hat. Dafür wird die Gleichung (**??**) nach v_2 umgestellt.

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

7.2 Formeln für gleichförmige Bewegungen

Bei einer gleichförmigen Bewegung gilt der Grundsatz:

$$v(t) = v \tag{12}$$

```
function getBrakeTime (float $v_0, float $v_1, float $verzoegerung) {
  if ($v_0 < $v_1) {
    return (($v_1/3.6)/$verzoegerung) - (($v_0/3.6)/$verzoegerung);
  } else if ($v_0 > $v_1) {
    return (($v_0/3.6)/$verzoegerung) - (($v_1/3.6)/$verzoegerung);
  } else {
    return 0;
  }
}
```

Code-Beispiel 8: *getBrakeTime*()

Für die Berechnung der Strecke gilt wie bei der gleichmäßig beschleunigten Bewegung:

$$s(t) = \int v(t)\,dt \tag{13}$$

$$s(t) = v \cdot t + s_0 \tag{14}$$

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

$$s(t) = v \cdot t \tag{15}$$

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:

$$t_{ges} = t_1 + t_2 \tag{16}$$

$$s_{ges} = s_1 + s_2 \tag{17}$$

Durch das Einsetzen der Gleichung (**??**) in die Gleichung (**??**) erhält man folgende Gleichung:

$$s_{ges} = v_1 \cdot t_1 + v_2 \cdot t_2 \tag{18}$$

Durch das Umstellen der Gleichung (**??**) nach t_2 und dem Einsetzen in Gleichung (**??**) gilt für t_1:

$$t_1 = \frac{s_{ges} - v_2 \cdot t_{ges}}{v_1 - v_2} \tag{19}$$

8 Fazit

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8.1 Was funktioniert gut?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8.2 Was funktioniert nicht?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8.3 Wie könnte man die Fehler in der Zukunft ausbessern?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

8.4 Was für Erweiterungsmöglichenkeiten gibt es?

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam

voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

A Anhang

A.1 main.php

```php
<?php
// Load all required external files
require 'config/multicast.php';
require 'vorbelegung.php';
require 'functions/sort_functions.php';
require 'functions/cache_functions_own.php';
require 'functions/ebuef_functions.php';
require 'functions/fahrtverlauf_functions.php';
require 'globalVariables.php';

// Set timezone
date_default_timezone_set("Europe/Berlin");

// Reports only errors
error_reporting(1);

// Global Variables
global $useRecalibration;

// Define own train errors
$trainErrors = array();
$trainErrors[0] = "Zug stand falsch herum und war zu lang um die Richtung zu ändern.";
$trainErrors[1] = "In der Datenbank ist für den Zug keine Zuglänge angegeben.";
$trainErrors[2] = "In der Datenbank ist für den Zug keine v_max angegeben.";
$trainErrors[3] = "Zug musste eine Notbremsung durchführen.";

// Load static data from the databse into the cache
$cacheInfranachbarn = createCacheInfranachbarn();
$cacheInfradaten = createCacheInfradaten();
$cacheSignaldaten = createCacheSignaldaten();
$cacheInfraLaenge = createcacheInfraLaenge();
$cacheHaltepunkte = createCacheHaltepunkte();
$cacheZwischenhaltepunkte = createChacheZwischenhaltepunkte();
$cacheInfraToGbt = createCacheInfraToGbt();
$cacheGbtToInfra = createCacheGbtToInfra();
$cacheFmaToInfra = createCacheFmaToInfra();
$cacheInfraToFma = array_flip($cacheFmaToInfra);
$cacheFahrplanSession = createCacheFahrplanSession();
$cacheSignalIDToBetriebsstelle = createCacheToBetriebsstelle();
$cacheFahrzeugeAbschnitte = createCacheFahrzeugeAbschnitte();
$cacheIDTDecoder = createCacheDecoderToAdresse();
$cacheDecoderToID = array_flip($cacheIDTDecoder);
$cacheAdresseToID = array();  // Filled with data in getAllTrains()
$cacheIDToAdresse = array();  // Filled with data in getAllTrains()

// Global variables
$allTrainsOnTheTrack = array();  // All adresses found on the tracks
$allTrains = array();    // All trains with the status 1 or 2
```

```php
$allUsedTrains = array();   // All trains with the status 1 or 2 that are standing on the tracks
$allTimes = array();
$lastMaxSpeedForInfraAndDir = array();

// Get simulation and real time
$simulationStartTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->sim_startzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "simulationszeit", null, array("inputtyp"=>"h:i:s"));
$simulationEndTimeToday = getUhrzeit(getUhrzeit($cacheFahrplanSession->sim_endzeit, "simulationszeit", null, array("outputtyp"=>"h:i:s")), "simulationszeit", null, array("inputtyp"=>"h:i:s"));
$simulationDuration = $cacheFahrplanSession->sim_endzeit - $cacheFahrplanSession->sim_startzeit;
$realStartTime = time();
$realEndTime = $realStartTime + $simulationDuration;
$timeDifference = $simulationStartTimeToday - $realStartTime;

// Start Message
startMessage();

// Load all trains
$allTrains = getAllTrains();

// Loads all trains that are in the rail network and prepares everything for the start
findTrainsOnTheTracks();

// Adds all the stops of the trains.
addStopsectionsForTimetable();

// Checks whether the trains are already at the first scheduled stop or not.
checkIfTrainReachedHaltepunkt();

// Checks if the trains are in the right direction and turns them if it is necessary and possible.
checkIfStartDirectionIsCorrect();
consoleAllTrainsPositionAndFahrplan();
showErrors();

// Determination of the current routes of all trains.
calculateNextSections();

// Checks whether the routes are set correctly.
checkIfFahrstrasseIsCorrrect();

// Calculate driving curve
calculateFahrverlauf();

//$unusedTrains = array_keys($allTimes);
$timeCheckFahrstrasseInterval = 3;
$timeCheckFahrstrasse = $timeCheckFahrstrasseInterval + microtime(true);
$timeCheckAllTrainErrorsInterval = 30;
$timeCheckAllTrainErrors = $timeCheckAllTrainErrorsInterval + microtime(true);
```

```
95  $timeCheckCalibrationInterval = 3;
96  $timeCheckCalibration = $timeCheckCalibrationInterval + microtime(true);
97  $sleeptime = 0.03;
98  while (true) {
99    foreach ($allTimes as $timeIndex => $timeValue) {
100     if (sizeof($timeValue) > 0) {
101       $id = $timeValue[0]["id"];
102       if ((microtime(true) + $timeDifference) > $timeValue[0]["live_time"]) {
103         if ($timeValue[0]["live_is_speed_change"]) {
104           $allUsedTrains[$id]["calibrate_section_one"] = null;
105           $allUsedTrains[$id]["calibrate_section_two"] = null;
106           if ($timeValue[0]["betriebsstelle"] == 'Notbremsung') {
107             sendFahrzeugbefehl($timeValue[0]["id"], intval($timeValue[0]["live_speed"]));
108             $allTrains[$id]["speed"] = intval($timeValue[0]["live_speed"]);
109             echo "Der Zug mit der Adresse ", $timeIndex, " leitet gerade eine
                      ↪ Gefahrenbremsung ein und hat seine Geschwindigkeit auf ", $timeValue
                      ↪ [0]["live_speed"], " km/h angepasst.\n";
110           } else {
111             sendFahrzeugbefehl($timeValue[0]["id"], intval($timeValue[0]["live_speed"]));
112             $allTrains[$id]["speed"] = intval($timeValue[0]["live_speed"]);
113             echo "Der Zug mit der Adresse ", $timeIndex, " hat auf der Fahrt nach ",
                      ↪ $timeValue[0]["betriebsstelle"],
114             " seine Geschwindigkeit auf ", $timeValue[0]["live_speed"], " km/h angepasst
                      ↪ .\n";
115           }
116         } else {
117           if (isset($allUsedTrains[$id]["calibrate_section_one"])) {
118             if ($allUsedTrains[$id]["calibrate_section_one"] != $timeValue[0]["
                      ↪ live_section"]) {
119               $allUsedTrains[$id]["calibrate_section_two"] = $timeValue[0]["live_section"
                        ↪ ];
120             }
121           }
122           $allUsedTrains[$id]["calibrate_section_one"] = $timeValue[0]["live_section"];
123         }
124
125         $allUsedTrains[$id]["current_position"] = $timeValue[0]["live_relative_position"
                ↪ ];
126         $allUsedTrains[$id]["current_speed"] = $timeValue[0]["live_speed"];
127         $allUsedTrains[$id]["current_section"] = $timeValue[0]["live_section"];
128
129         if ($timeValue[0]["wendet"]) {
130           changeDirection($timeValue[0]["id"]);
131         }
132
133         if (isset($timeValue[0]["live_all_targets_reached"])) {
134           $allUsedTrains[$id]["next_betriebsstellen_data"][$timeValue[0]["
                    ↪ live_all_targets_reached"]]["angekommen"] = true;
135           echo "Der Zug mit der Adresse ", $timeIndex, " hat den Halt ", $allUsedTrains[
                    ↪ $id]["next_betriebsstellen_data"][$timeValue[0]["
                    ↪ live_all_targets_reached"]]["betriebstelle"], " erreicht.\n";
136         }
137
```

```
138         if ($timeValue[0]["live_target_reached"]) {
139
140           $currentZugId = $allUsedTrains[$id]["zug_id"];
141           $newZugId = getFahrzeugZugIds(array($id));
142
143           if (sizeof($newZugId) == 0) {
144             $newZugId = null;
145           } else {
146             $newZugId = getFahrzeugZugIds(array($timeValue[0]["id"]));
147             $newZugId = $newZugId[array_key_first($newZugId)]["zug_id"];
148           }
149           if (!($currentZugId == $newZugId && $currentZugId != null)) {
150             if ($currentZugId != null && $newZugId != null) {
151               // The train runs from now on a new timetable
152               $allUsedTrains[$id]["zug_id"] = $newZugId;
153               $allUsedTrains[$id]["operates_on_timetable"] = true;
154               getFahrplanAndPositionForOneTrain($id, $newZugId);
155               addStopsectionsForTimetable($id);
156               checkIfTrainReachedHaltepunkt($id);
157               checkIfStartDirectionIsCorrect($id);
158               calculateNextSections($id);
159               checkIfFahrstrasseIsCorrrect($id);
160               calculateFahrverlauf($id);
161
162             } else if ($currentZugId == null && $newZugId != null) {
163               // The train runs from now on timetable
164               $allUsedTrains[$id]["zug_id"] = $newZugId;
165               $allUsedTrains[$id]["operates_on_timetable"] = true;
166               getFahrplanAndPositionForOneTrain($id);
167               addStopsectionsForTimetable($id);
168               checkIfTrainReachedHaltepunkt($id);
169               checkIfStartDirectionIsCorrect($id);
170               calculateNextSections($id);
171               checkIfFahrstrasseIsCorrrect($id);
172               calculateFahrverlauf($id);
173             } else if ($currentZugId != null && $newZugId == null) {
174               // The train runs from now on without timetable
175               $allUsedTrains[$id]["operates_on_timetable"] = false;
176               calculateNextSections($id);
177               calculateFahrverlauf($id);
178             }
179           }
180         }
181         array_shift($allTimes[$timeIndex]);
182       }
183     }
184   }
185   // Recalibration
186   if ($useRecalibration) {
187     if (microtime(true) > $timeCheckCalibration) {
188       foreach ($allUsedTrains as $trainKey => $trainValue) {
189         if (isset($allUsedTrains[$trainKey]["calibrate_section_two"])) {
```

```
190           $newPosition = getCalibratedPosition($trainKey, $allUsedTrains[$trainKey]["
                  ↪ current_speed"]);
191           if ($newPosition["possible"]) {
192             echo "Die Position des Fahrzeugs mit der ID: ", $trainKey, " wird neu
                    ↪ ermittelt.\n";
193             $position = $newPosition["position"];
194             $section = $newPosition["section"];
195             echo "Die alte Position war Abschnitt: ", $allUsedTrains[$trainKey]["
                    ↪ current_section"], " (", number_format($allUsedTrains[$trainKey]["
                    ↪ current_position"], 2), " m) und die neue Position ist Abschnitt: ",
                    ↪ $section, " (", number_format($position, 2), " m).\n";
196             if ($position > $cacheInfraLaenge[$section]) {
197               echo "Die Position konnte nicht neu kalibriert werden, da die aktuelle
                      ↪ Position im Abschnitt größer ist, als die Länge des Abschnitts.\n";
198             } else {
199               $allUsedTrains[$trainKey]["current_section"] = $section;
200               $allUsedTrains[$trainKey]["current_position"] = $position;
201               calculateNextSections($trainKey);
202               checkIfFahrstrasseIsCorrrect($trainKey);
203               calculateFahrverlauf($trainKey, true);
204               echo "Die Position des Fahrzeugs mit der ID: ", $trainKey, " wurde neu
                      ↪ ermittelt.\n";
205             }
206           }
207         }
208       }
209       $timeCheckCalibration = $timeCheckCalibration + $timeCheckCalibrationInterval;
210     }
211   }
212   // Checking whether the route has changed and whether new vehicles have been placed on
          ↪  the network
213   if (microtime(true) > $timeCheckFahrstrasse) {
214     foreach ($allUsedTrains as $trainID => $trainValue) {
215       compareTwoNaechsteAbschnitte($trainID);
216     }
217     $returnUpdate = updateAllTrainsOnTheTrack();
218     $newTrains = $returnUpdate["new"];
219     $removeTrains = $returnUpdate["removed"];
220
221     if (sizeof($newTrains) > 0) {
222       echo "Neu hinzugefügte Züge: \n";
223       foreach ($newTrains as $newTrain) {
224         $id = $cacheDecoderToID[$newTrain];
225         echo "\tID:\t", $id, "\tAdresse:\t", $newTrain;
226       }
227       echo "\n";
228     }
229     foreach ($newTrains as $newTrain) {
230       $id = $cacheDecoderToID[$newTrain];
231       prepareTrainForRide($newTrain);
232       addStopsectionsForTimetable($id);
233       checkIfTrainReachedHaltepunkt($id);
234       checkIfStartDirectionIsCorrect($id);
```

```
235       consoleAllTrainsPositionAndFahrplan($id);
236       calculateNextSections($id);
237       checkIfFahrstrasseIsCorrrect($id);
238       calculateFahrverlauf($id);
239     }
240     if (sizeof($removeTrains) > 0) {
241       echo "Entfernte Züge:\n";
242       foreach ($removeTrains as $removeTrain) {
243         $id = $cacheDecoderToID[$removeTrain];
244         unset($allUsedTrains[$id]);
245         echo "\tID:\t", $id, "\tAdresse:\t", $removeTrain;
246       }
247       echo "\n";
248     }
249     $timeCheckFahrstrasse = $timeCheckFahrstrasse + $timeCheckFahrstrasseInterval;
250   }
251   // Output of all current data of the vehicles
252   if (microtime(true) > $timeCheckAllTrainErrors) {
253     consoleAllTrainsPositionAndFahrplan();
254     showFahrplan();
255     $timeCheckAllTrainErrors = $timeCheckAllTrainErrors +
          ↪ $timeCheckAllTrainErrorsInterval;
256   }
257   sleep($sleeptime);
258 }
```

A.2 sort_functions.php

```
1  <?php
2  ini_set('memory_limit', '1024M');
3
4  function startMessage() {
5    global $simulationStartTimeToday;
6    global $simulationEndTimeToday;
7    global $simulationDuration;
8    global $realStartTime;
9    global $realEndTime;
10   global $cacheFahrplanSession;
11
12   $realStartTimeAsHHMMSS = getUhrzeit($realStartTime, "simulationszeit", null, array("
         ↪ outputtyp"=>"h:i:s"));
13   $simulationEndTimeAsHHMMSS = getUhrzeit($simulationEndTimeToday, "simulationszeit",
         ↪ null, array("outputtyp"=>"h:i:s"));
14   $simulationDurationAsHHMMSS = toStd($simulationDuration);
15   $realEndTimeAsHHMMSS = getUhrzeit($realEndTime, "simulationszeit", null, array("
         ↪ outputtyp"=>"h:i:s"));
16   $simulationStartTimeAsHHMMSS = getUhrzeit($simulationStartTimeToday, "simulationszeit"
         ↪ , null, array("outputtyp"=>"h:i:s"));
17   $hashtagLine = "#########################################################################\
         ↪ n";
18   $emptyLine = "#\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t#\n";
19
```

```
20    echo $hashtagLine;
21    echo $emptyLine;
22    echo "#\t\t\t Start der automatischen Zugbeeinflussung\t\t\t\t#\n";
23    echo "#\t\tim Eisenbahnbetriebs- und Experimentierfeld (EBuEf) \t\t#\n";
24    echo "#\t\t\t\t\t\t\t der TU Berlin\t\t\t\t\t\t\t\t#\n";
25    echo "#\t\t\t\t\t\t im eingleisigen Netz \t\t\t\t\t\t#\n";
26    echo $emptyLine;
27    echo "#\t\t\t\t\t\t\t\t____\t\t\t\t\t\t\t\t\t#\n";
28    echo "#\t\t\t\t\t\t\t\t|DD|____T_\t\t\t\t\t\t\t\t#\n";
29    echo "#\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\t @-@-@-oo\\\t\t\t\t\t\t\t\t#\n";
31    echo "#\t\t\t\t\t\t==============================\t\t\t\t\t#\n";
32    echo $emptyLine;
33    echo "#\t Start der Simulation: \t\t\t\t\t\t\t\t\t\t#\n";
34    echo "#\t\t Simulationszeit: \t\t\t", $simulationStartTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
35    echo "#\t\t Realzeit: \t\t\t\t\t", $realStartTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
36    echo $emptyLine;
37    echo "#\t Ende der Simulation: \t\t\t\t\t\t\t\t\t\t#\n";
38    echo "#\t\t Simulationszeit: \t\t\t", $simulationEndTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
39    echo "#\t\t Realzeit: \t\t\t\t\t", $realEndTimeAsHHMMSS, "\t\t\t\t\t\t#\n";
40    echo $emptyLine;
41    echo "#\t Dauer der Simulation: \t\t\t", $simulationDurationAsHHMMSS, "\t\t\t\t\t\t#\n
          ↪ ";
42    echo $emptyLine;
43    echo "#\t Fahrplanname: \t\t\t\t\t", $cacheFahrplanSession->name, "\t\t#\n";
44    echo "#\t Sessionkey: \t\t\t\t\t", $cacheFahrplanSession->sessionkey, "\t\t\t\t\t#\n";
45    echo $emptyLine;
46    echo $hashtagLine, "\n\n";
47  }
48
49  function toStd(float $sekunden) {
50    $stunden = floor($sekunden / 3600);
51    $minuten = floor(($sekunden - ($stunden * 3600)) / 60);
52    $sekunden = round($sekunden - ($stunden * 3600) - ($minuten * 60));
53
54    if ($stunden <= 9) {
55      $strStunden = "0" . $stunden;
56    } else {
57      $strStunden = $stunden;
58    }
59
60    if ($minuten <= 9) {
61      $strMinuten = "0" . $minuten;
62    } else {
63      $strMinuten = $minuten;
64    }
65
66    if ($sekunden <= 9) {
67      $strSekunden = "0" . $sekunden;
68    } else {
69      $strSekunden = $sekunden;
70    }
71
```

```
72    return "$strStunden:$strMinuten:$strSekunden";
73  }
74
75  function getAllAdresses () : array {
76
77    $zustand = array("0", "1");
78    //$zustand = array("0", "1", "2");
79    echo "Alle Züge, die den Zustand ", implode(", ", $zustand), " haben, werden
          ↪ eingelesen.\n\n";
80    $returnAdresses = array();
81    $DB = new DB_MySQL();
82    $adresses = $DB->select("SELECT `".DB_TABLE_FAHRZEUGE."`.`adresse`, `".
          ↪ DB_TABLE_FAHRZEUGE."`.`zustand` FROM `".DB_TABLE_FAHRZEUGE."`");
83    unset($DB);
84
85    foreach ($adresses as $adressIndex => $adressValue) {
86      if (in_array($adressValue->zustand, $zustand)) {
87        array_push($returnAdresses, (int) $adressValue->adresse);
88      }
89    }
90
91    return $returnAdresses;
92  }
93
94  function getAllTrains () : array {
95    global $cacheAdresseToID;
96    global $cacheIDToAdresse;
97    global $globalMinSpeed;
98
99    $allAdresses = getAllAdresses();
100   $DB = new DB_MySQL();
101   $allTrains = array();
102   $id = null;
103
104   foreach ($allAdresses as $adress) {
105     $train_fahrzeuge = get_object_vars($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]);
106     $id = $train_fahrzeuge["id"];
107     $train_daten = $DB->select("SELECT `".DB_TABLE_FAHRZEUGE_DATEN."`.`baureihe` FROM `"
          ↪ .DB_TABLE_FAHRZEUGE_DATEN."` WHERE `".DB_TABLE_FAHRZEUGE_DATEN."`.`id` = $id"
          ↪ )[0]->baureihe;
108     $train_baureihe = $DB->select("SELECT `".DB_TABLE_FAHRZEUGE_BAUREIHEN."`.`vmax` FROM
          ↪  `".DB_TABLE_FAHRZEUGE_BAUREIHEN."` WHERE `".DB_TABLE_FAHRZEUGE_BAUREIHEN."
          ↪ `.`nummer` = $train_daten");
109     if (sizeof($train_baureihe) != 0) {
110       $train_baureihe_return["v_max"] = intval($train_baureihe[0]->vmax);
111     } else {
112       $train_baureihe_return["v_max"] = $globalMinSpeed;
113     }
```

```
114     $id = intval($train_fahrzeuge["id"]);
115     $cacheAdresseToID[intval($train_fahrzeuge["adresse"])] = intval($id);
116     $returnArray = array_merge($train_fahrzeuge, $train_baureihe_return);
117     $allTrains[$id] = $returnArray;
118   }
119   unset($DB);
120   $cacheIDToAdresse = array_flip($cacheAdresseToID);
121   return $allTrains;
122 }
123
124 function updateAllTrainsOnTheTrack () {
125   global $allTrainsOnTheTrack;
126   $newTrains = array();
127   $removedTrains = array();
128   $allTrains = array();
129
130   $DB = new DB_MySQL();
131   $foundTrains = $DB->select("SELECT DISTINCT `".DB_TABLE_FMA."`.`decoder_adresse` FROM
          ↪ `".DB_TABLE_FMA."` WHERE `".DB_TABLE_FMA."`.`decoder_adresse` IS NOT NULL AND `
          ↪ ".DB_TABLE_FMA."`.`decoder_adresse` <> '"."0"."'");
132   unset($DB);
133   foreach ($foundTrains as $train) {
134     array_push($allTrains, intval($train->decoder_adresse));
135     if (!in_array($train->decoder_adresse, $allTrainsOnTheTrack)) {
136       array_push($newTrains, intval($train->decoder_adresse));
137     }
138   }
139   foreach ($allTrainsOnTheTrack as $train) {
140     if (!in_array($train, $allTrains)) {
141       array_push($removedTrains, $train);
142     }
143   }
144   $allTrainsOnTheTrack = $allTrains;
145   return array("new"=>$newTrains, "removed"=>$removedTrains);
146 }
147
148 function findTrainsOnTheTracks () {
149
150   global $allTrainsOnTheTrack;
151
152   $DB = new DB_MySQL();
153   $foundTrains = $DB->select("SELECT DISTINCT `".DB_TABLE_FMA."`.`decoder_adresse` FROM
          ↪ `".DB_TABLE_FMA."` WHERE `".DB_TABLE_FMA."`.`decoder_adresse` IS NOT NULL AND `
          ↪ ".DB_TABLE_FMA."`.`decoder_adresse` <> '"."0"."'");
154   unset($DB);
155   foreach ($foundTrains as $train) {
156     if (!in_array($train->decoder_adresse, $allTrainsOnTheTrack)) {
157       array_push($allTrainsOnTheTrack, intval($train->decoder_adresse));
158       prepareTrainForRide($train->decoder_adresse);
159     }
160   }
161 }
162
```

```php
function prepareTrainForRide(int $adresse) {

  global $allUsedTrains;
  global $allTrains;
  global $cacheAdresseToID;
  global $cacheFmaToInfra;
  global $cacheInfraToFma;
  global $cacheZwischenhaltepunkte;
  global $cacheInfraLaenge;
  global $globalNotverzoegerung;

  $trainID = $cacheAdresseToID[$adresse];
  $zugID = null;
  $keysZwischenhalte = array_keys($cacheZwischenhaltepunkte);
  $allUsedTrains[$trainID]["id"] = $allTrains[$trainID]["id"];
  $allUsedTrains[$trainID]["adresse"] = $allTrains[$trainID]["adresse"];
  $allUsedTrains[$trainID]["zug_id"] = null;
  $allUsedTrains[$trainID]["verzoegerung"] = floatval($allTrains[$trainID]['verzoegerung
      ↪ ']);
  $allUsedTrains[$trainID]["notverzoegerung"] = $globalNotverzoegerung;
  $allUsedTrains[$trainID]["zuglaenge"] = $allTrains[$trainID]["zuglaenge"];
  $allUsedTrains[$trainID]["v_max"] = $allTrains[$trainID]["v_max"];
  $allUsedTrains[$trainID]["dir"] = $allTrains[$trainID]["dir"];
  $allUsedTrains[$trainID]["error"] = array();
  $allUsedTrains[$trainID]["operates_on_timetable"] = false;
  $allUsedTrains[$trainID]["fahrstrasse_is_correct"] = false;
  $allUsedTrains[$trainID]["current_speed"] = intval($allTrains[$trainID]["speed"]);
  $allUsedTrains[$trainID]["current_position"] = null;
  $allUsedTrains[$trainID]["current_section"] = null;
  $allUsedTrains[$trainID]["next_sections"] = array();
  $allUsedTrains[$trainID]["next_lenghts"] = array();
  $allUsedTrains[$trainID]["next_v_max"] = array();
  $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
  $allUsedTrains[$trainID]["next_bs"] = '';
  $allUsedTrains[$trainID]["earliest_possible_start_time"] = null;
  $allUsedTrains[$trainID]["calibrate_section_one"] = null;
  $allUsedTrains[$trainID]["calibrate_section_two"] = null;

  // Check for errors
  if (!($allUsedTrains[$trainID]["zuglaenge"] > 0)) {
    array_push($allUsedTrains[$trainID]["error"], 1);
  }
  if (!isset($allUsedTrains[$trainID]["v_max"])) {
    array_push($allUsedTrains[$trainID]["error"], 2);
  }
  // Get position
  $fma = getPosition($adresse);
  if (sizeof($fma) == 0) {
    $allUsedTrains[$trainID]["current_fma_section"] = null;
    $allUsedTrains[$trainID]["current_section"] = null;
  } elseif (sizeof($fma) == 1) {
    $allUsedTrains[$trainID]["current_fma_section"] = $fma[0];
    $allUsedTrains[$trainID]["current_section"] = $cacheFmaToInfra[$fma[0]];
```

```
215   } else {
216     $infraArray = array();
217     foreach ($fma as $value) {
218       array_push($infraArray, $cacheFmaToInfra[$value]);
219     }
220     $infra = getFrontPosition($infraArray, $allTrains[$trainID]["dir"]);
221     $allUsedTrains[$trainID]["current_fma_section"] = $cacheInfraToFma[$infra];
222     $allUsedTrains[$trainID]["current_section"] = $infra;
223   }
224
225   $allUsedTrains[$trainID]["current_position"] = $cacheInfraLaenge[$allUsedTrains[
          ↪ $trainID]["current_section"]];
226   // Get Zug ID/Check for timetable
227   $timetableIDs = getFahrzeugZugIds(array($trainID));
228   if (sizeof($timetableIDs) != 0) {
229     $timetableID = $timetableIDs[array_key_first($timetableIDs)];
230     $allUsedTrains[$trainID]["zug_id"] = intval($timetableID["zug_id"]);
231     $zugID = intval($timetableID["zug_id"]);
232     $allUsedTrains[$trainID]["operates_on_timetable"] = true;
233
234   } else {
235     $allUsedTrains[$trainID]["zug_id"] = null;
236     $allUsedTrains[$trainID]["operates_on_timetable"] = false;
237   }
238   // Get timetable data
239   if (isset($zugID)) {
240     $nextBetriebsstellen = getNextBetriebsstellen($zugID);
241   }
242   if ($zugID != null && sizeof($nextBetriebsstellen) != 0) {
243     for ($i = 0; $i < sizeof($nextBetriebsstellen); $i++) {
244       if (sizeof(explode("_", $nextBetriebsstellen[$i])) != 2) {
245         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["is_on_fahrstrasse"] =
              ↪ false;
246         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebstelle"] =
              ↪ $nextBetriebsstellen[$i];
247         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
              ↪ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
248         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"] = true;
249       } else if(in_array($nextBetriebsstellen[$i], $keysZwischenhalte)) {
250         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["is_on_fahrstrasse"] =
              ↪ false;
251         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebstelle"] =
              ↪ $nextBetriebsstellen[$i];
252         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
              ↪ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
253         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"] = false
              ↪ ;
254       }
255     }
256     $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array_values($allUsedTrains[
          ↪ $trainID]["next_betriebsstellen_data"]);
257   } else {
258     $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
```

```
259   }
260
261   foreach ($allUsedTrains[$trainID]["next_betriebsstellen_data"] as $betriebsstelleKey
          ↪ => $betriebsstelleValue) {
262     if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["
            ↪ zeiten"]["abfahrt_soll"] != null) {
263       $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
              ↪ ]["abfahrt_soll_timestamp"] = getUhrzeit($betriebsstelleValue["zeiten"]["
              ↪ abfahrt_soll"], "simulationszeit", null, array("inputtyp" => "h:i:s"));
264     } else {
265       $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
              ↪ ]["abfahrt_soll_timestamp"] = null;
266     }
267     if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["
            ↪ zeiten"]["ankunft_soll"] != null) {
268       $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
              ↪ ]["ankunft_soll_timestamp"] = getUhrzeit($betriebsstelleValue["zeiten"]["
              ↪ ankunft_soll"], "simulationszeit", null, array("inputtyp" => "h:i:s"));
269     } else {
270       $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
              ↪ ]["ankunft_soll_timestamp"] = null;
271     }
272     $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"][
            ↪ "verspaetung"] = 0;
273   }
274 }
275
276 function getPosition(int $adresse) {
277   $returnPosition = array();
278   $DB = new DB_MySQL();
279   $position = $DB->select("SELECT `".DB_TABLE_FMA."`.`fma_id` FROM `".DB_TABLE_FMA."`
          ↪ WHERE `".DB_TABLE_FMA."`.`decoder_adresse` = $adresse");
280   unset($DB);
281   if (sizeof($position) != 0) {
282     for ($i = 0; $i < sizeof($position); $i++) {
283       array_push($returnPosition, intval(get_object_vars($position[$i])["fma_id"]));
284     }
285   }
286   return $returnPosition;
287 }
288
289 function getFrontPosition(array $infra, int $dir) : int {
290
291   foreach ($infra as $section) {
292     $nextSections = array();
293     $test = getNaechsteAbschnitte($section, $dir);
294     foreach ($test as $value) {
295       array_push($nextSections, $value["infra_id"]);
296     }
297     if (sizeof(array_intersect($infra, $nextSections)) == 0) {
298       return $section;
299     }
300   }
```

```
301
302   return false;
303 }
304
305 function getNextBetriebsstellen (int $id) : array {
306   $DB = new DB_MySQL();
307   $returnBetriebsstellen = array();
308   $betriebsstellen = $DB->select("SELECT `".DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN."`.`
          ↪ betriebsstelle` FROM `".DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN."` WHERE `".
          ↪ DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN."`.`zug_id` = $id ORDER BY `".
          ↪ DB_TABLE_FAHRPLAN_SESSIONFAHRPLAN."`.`id` ASC");
309   unset($DB);
310   foreach ($betriebsstellen as $betriebsstellenIndex => $betriebsstellenValue) {
311     array_push($returnBetriebsstellen, $betriebsstellenValue->betriebsstelle);
312   }
313   if (sizeof($betriebsstellen) == 0) {
314     debugMessage("Zu dieser Zug ID sind keine nächsten Betriebsstellen im Fahrplan
            ↪ vorhanden.");
315   }
316   return $returnBetriebsstellen;
317 }
318
319 function consoleAllTrainsPositionAndFahrplan($id = false) {
320   global $allUsedTrains;
321   $checkAllTrains = true;
322   if ($id != false) {
323     $checkAllTrains = false;
324   } else {
325     echo "Alle vorhandenen Züge:\n\n";
326   }
327   foreach ($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         $fahrplan = "ja";
334       } else {
335         $fahrplan = "nein";
336       }
337       if (sizeof($train["error"]) != 0) {
338         $error = "ja";
339       } else {
340         $error = "nein";
341       }
342       if (!isset($train["zug_id"])) {
343         $zugId = '-----';
344       } else {
345         $zugId = $train["zug_id"];
346       }
347       echo "Zug ID: ", $train["id"], " (Adresse: ", $train["adresse"], ", Zug ID: ",
            ↪ $zugId, ")\t Fährt nach Fahrplan: ",
```

```
348       $fahrplan, "\t Fahrtrichtung: ", $train["dir"], "\t Infra-Abschnitt: ", $train["
              ↪ current_section"],
349       "\t\tAktuelle relative Position im Infra-Abschnitt: ", number_format($train["
              ↪ current_position"],2), "m\t\tFehler vorhanden:\t", $error, "\n";
350     }
351   }
352   echo "\n";
353 }
354
355 function showFahrplan ($id = false) {
356
357   global $allUsedTrains;
358   $checkAllTrains = true;
359
360   if ($id != false) {
361     $checkAllTrains = false;
362   } else {
363     echo "Alle vorhandenen Fahrpläne:\n\n";
364   }
365
366   foreach ($allUsedTrains as $train) {
367     if ($checkAllTrains || $train["id"] == $id) {
368       $fahrplan = null;
369       $error = null;
370       $zugId = null;
371       if ($train["operates_on_timetable"]) {
372         if (!isset($train["zug_id"])) {
373           $zugId = '-----';
374         } else {
375           $zugId = $train["zug_id"];
376         }
377         $nextStations = '';
378         $lastStation = '';
379         //var_dump($train["next_betriebsstellen_data"]);
380         foreach ($train["next_betriebsstellen_data"] as $bs) {
381           if (!$bs["angekommen"]) {
382             $nextStations = $nextStations . $bs["betriebstelle"] . ' ';
383
384           } else {
385             $lastStation = $bs["betriebstelle"];
386           }
387         }
388         if ($lastStation == '') {
389           $lastStation = '---';
390         }
391         echo "Zug ID: ", $train["id"], " (Adresse: ", $train["adresse"], ", Zug ID: ",
              ↪ $zugId, ")\t Letzte Station: ", $lastStation, " \tNächste Stationen: ",
              ↪ $nextStations, "\n";
392       }
393     }
394   }
395   echo "\n";
396 }
```

```php

function checkIfStartDirectionIsCorrect($id = false) {
  global $allUsedTrains;
  $checkAllTrains = true;
  if ($id != false) {
    $checkAllTrains = false;
    echo "Für den Fall, dass die Fahrtrichtung der Züge nicht mit dem Fahrplan ü
        ↪ bereinstimmt, wird die Richtung verändert:\n\n";
  } else {
    echo "Für den Fall, dass die Fahrtrichtung des Zuges nicht mit dem Fahrplan ü
        ↪ bereinstimmt, wird die Richtung verändert:\n\n";
  }
  foreach ($allUsedTrains as $train) {
    if ($checkAllTrains || $train["id"] == $id) {
      if ($train["operates_on_timetable"]) {
        $endLoop = 0;
        for ($i = 0; $i < sizeof($train["next_betriebsstellen_data"]); $i++) {
          if ($train["next_betriebsstellen_data"][$i]["angekommen"]) {
            $endLoop = $i;
          }
        }
        if ($train["dir"] != $train["next_betriebsstellen_data"][$endLoop]["zeiten"]["
            ↪ fahrtrichtung"][1]) {
          changeDirection($train["id"]);
        }
      }
    }
  }
  echo "\n";
}

function changeDirection (int $id) {
  global $allUsedTrains;
  global $cacheInfraLaenge;
  global $timeDifference;
  global $allTrains;
  $section = $allUsedTrains[$id]["current_section"];
  $position = $allUsedTrains[$id]["current_position"];
  $direction = $allUsedTrains[$id]["dir"];
  $length = $allUsedTrains[$id]["zuglaenge"];
  $newTrainLength = $length + ($cacheInfraLaenge[$section] - $position);
  $newDirection = null;
  $newSection = null;
  $cumLength = 0;
  if ($direction == 0) {
    $newDirection = 1;
  } else {
    $newDirection = 0;
  }
  $newPosition = null;
  $nextSections = getNaechsteAbschnitte($section, $newDirection);
  $currentData = array(0 => array("laenge" => $cacheInfraLaenge[$section], "infra_id" =>
      ↪  $section));
```

```
446   $mergedData = array_merge($currentData, $nextSections);
447   foreach ($mergedData as $sectionValue) {
448     $cumLength += $sectionValue["laenge"];
449     if ($newTrainLength <= $cumLength) {
450       $newSection = $sectionValue["infra_id"];
451       $newPosition = $cacheInfraLaenge[$newSection] - ($cumLength - $newTrainLength);
452       break;
453     }
454   }
455   if ($newPosition == null) {
456     echo "Die Richtung des Zugs mit der ID ", $id, " lässt sich nicht ändern, weil das
          ↪ Zugende auf einem auf Halt stehenden Signal steht.\n";
457     echo "\tDie Zuglänge beträgt:\t", $length, " m\n\tDie Distanz zwischen Zugende und
          ↪ dem auf Halt stehenden Signal beträgt:\t", ($cumLength - ($cacheInfraLaenge[
          ↪ $section] - $position)), " m\n\n";
458     array_push($allUsedTrains[$id]["error"], 0);
459   } else {
460     echo "Die Richtung des Zugs mit der ID: ", $id, " wurde auf ", $newDirection, " geä
          ↪ ndert.\n";
461     $allUsedTrains[$id]["current_section"] = $newSection;
462     $allUsedTrains[$id]["current_position"] = $newPosition;
463     $allUsedTrains[$id]["dir"] = $newDirection;
464     $allUsedTrains[$id]["earliest_possible_start_time"] = FZS_WARTEZEIT_WENDEN + time()
          ↪ + $timeDifference;
465     $allTrains[$id]["dir"] = $newDirection;
466     $DB = new DB_MySQL();
467     $DB->select("UPDATE `".DB_TABLE_FAHRZEUGE."` SET `".DB_TABLE_FAHRZEUGE."`.`dir` =
          ↪ $newDirection WHERE `".DB_TABLE_FAHRZEUGE."`.`id` = $id");
468     unset($DB);
469     sendFahrzeugbefehl($id, -4);
470   }
471 }
472
473 function showErrors() {
474
475   global $allUsedTrains;
476   global $trainErrors;
477
478   $foundError = false;
479   echo "Hier werden für alle Züge mögliche Fehler angezeigt:\n\n";
480
481   foreach ($allUsedTrains as $trainIndex => $trainValue) {
482     if (sizeof($trainValue["error"]) != 0) {
483       $foundError = true;
484       echo "Zug ID: ", $trainValue["id"], "\n";
485       $index = 1;
486       foreach ($trainValue["error"] as $error) {
487         echo "\t", $index, ". Fehler:\t", $trainErrors[$error], "\n";
488         $index++;
489       }
490       echo "\n";
491     }
492   }
```

```
493
494   if (!$foundError) {
495     echo "Keiner der Züge hat eine Fehlermeldung.\n";
496   }
497 }
498
499 // Adds for all trains (if no ID is passed) or for one train (if an ID is passed)
500 // the stops of the schedule (if the train runs according to schedule)
501 function addStopsectionsForTimetable($id = false) {
502   global $allUsedTrains;
503   global $cacheHaltepunkte;
504   global $cacheZwischenhaltepunkte;
505   $checkAllTrains = true;
506   if ($id != false) {
507     $checkAllTrains = false;
508   }
509   foreach ($allUsedTrains as $trainIndex => $trainValue) {
510     if ($checkAllTrains || $trainValue["id"] == $id) {
511       if (sizeof($trainValue["error"]) == 0) {
512         if ($trainValue["operates_on_timetable"]) {
513           foreach ($trainValue["next_betriebsstellen_data"] as $betriebsstelleKey =>
                 ↪ $betriebsstelleValue) {
514             if (in_array($betriebsstelleValue["betriebstelle"], array_keys(
                   ↪ $cacheHaltepunkte))) {
515               $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$betriebsstelleKey
                     ↪ ]["haltepunkte"] = $cacheHaltepunkte[$betriebsstelleValue["
                     ↪ betriebstelle"]][$trainValue["dir"]];
516             } else if (in_array($betriebsstelleValue["betriebstelle"], array_keys(
                   ↪ $cacheZwischenhaltepunkte))) {
517               $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$betriebsstelleKey
                     ↪ ]["haltepunkte"] = array($cacheZwischenhaltepunkte[
                     ↪ $betriebsstelleValue["betriebstelle"]]);
518             } else {
519               $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$betriebsstelleKey
                     ↪ ]["haltepunkte"] = array();
520             }
521           }
522         }
523       }
524     }
525   }
526 }
527
528 function initalFirstLiveData($id = false) {
529
530   global $allUsedTrains;
531   global $allTimes;
532
533   $checkAllTrains = true;
534
535   if ($id != false) {
536     $checkAllTrains = false;
537   }
```

```
538
539   foreach ($allUsedTrains as $trainIndex => $trainValue) {
540     if (($checkAllTrains || $trainValue["id"] == $id)) {
541       $allTimes[$trainValue["adresse"]] = array();
542     }
543   }
544 }
545
546 // Determines for all trains (if no ID is passed) or for one train
547 // (if an ID is passed) the route including the lengths, maximum
548 // allowed speeds and IDs of the next sections.
549 //
550 // The results can be stored directly in the $usedTrains array
551 // ($writeResultToTrain = true) or returned as return
552 // ($writeResultToTrain = false) so that they can be compared
553 // with the previous data.
554 function calculateNextSections($id = false, $writeResultToTrain = true) {
555
556   global $allUsedTrains;
557   global $cacheInfraLaenge;
558   global $globalSpeedInCurrentSection;
559   global $lastMaxSpeedForInfraAndDir;
560
561   $checkAllTrains = true;
562
563   if ($id != false) {
564     $checkAllTrains = false;
565   }
566
567   foreach ($allUsedTrains as $trainIndex => $trainValue) {
568     if (($checkAllTrains || $trainValue["id"] == $id) && sizeof($trainValue["error"]) ==
          ↪ 0) {
569       $dir = $trainValue["dir"];
570       $currentSectionComp = $trainValue["current_section"];
571       $signal = getSignalForSectionAndDirection($currentSectionComp, $dir);
572       $nextSectionsComp = array();
573       $nextVMaxComp = array();
574       $nextLengthsComp = array();
575       $nextSignalbegriff = null;
576       if ($signal != null) {
577         $nextSignalbegriff = getSignalbegriff($signal);
578         $nextSignalbegriff = $nextSignalbegriff[array_key_last($nextSignalbegriff)]["
              ↪ geschwindigkeit"];
579         if ($nextSignalbegriff == -25) {
580           $nextSignalbegriff = 25;
581         } else if ($nextSignalbegriff <= 0) {
582           $nextSignalbegriff = 0;
583         }
584       } else {
585         $nextSignalbegriff = null;
586       }
587       $return = getNaechsteAbschnitte($currentSectionComp, $dir);
588       $allUsedTrains[$trainIndex]["last_get_naechste_abschnitte"] = $return;
```

```
589       if (isset($lastMaxSpeedForInfraAndDir[$trainValue["dir"]][$trainValue["
              ↪ current_section"]])) {
590         $currentVMax = $lastMaxSpeedForInfraAndDir[$trainValue["dir"]][$trainValue["
                ↪ current_section"]];
591       } else {
592         $currentVMax = $globalSpeedInCurrentSection;
593       }
594       array_push($nextSectionsComp, $currentSectionComp);
595       array_push($nextVMaxComp, $currentVMax);
596       array_push($nextLengthsComp, $cacheInfraLaenge[$currentSectionComp]);
597       if (isset($nextSignalbegriff)) {
598         $currentVMax = $nextSignalbegriff;
599       }
600       if ($currentVMax == 0) {
601         if ($writeResultToTrain) {
602           $allUsedTrains[$trainIndex]["next_sections"] = $nextSectionsComp;
603           $allUsedTrains[$trainIndex]["next_lenghts"] = $nextLengthsComp;
604           $allUsedTrains[$trainIndex]["next_v_max"] = $nextVMaxComp;
605         } else {
606           return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
607         }
608       } else {
609         foreach ($return as $section) {
610           array_push($nextSectionsComp, $section["infra_id"]);
611           array_push($nextVMaxComp, $currentVMax);
612           array_push($nextLengthsComp, $cacheInfraLaenge[$section["infra_id"]]);
613           $lastMaxSpeedForInfraAndDir[intval($trainValue["dir"])][intval($section["
                  ↪ infra_id"])] = intval($currentVMax);
614           if ($section["signal_id"] != null) {
615             $signal = $section["signal_id"];
616             $nextSignalbegriff = getSignalbegriff($signal);
617             $nextSignalbegriff = $nextSignalbegriff[array_key_last($nextSignalbegriff)]["
                    ↪ geschwindigkeit"];
618             if ($nextSignalbegriff == -25) {
619               $currentVMax = 25;
620             } else if ($nextSignalbegriff < 0) {
621               $currentVMax = 0;
622             } else {
623               $currentVMax = $nextSignalbegriff;
624             }
625           }
626         }
627         if ($writeResultToTrain) {
628           $allUsedTrains[$trainIndex]["next_sections"] = $nextSectionsComp;
629           $allUsedTrains[$trainIndex]["next_lenghts"] = $nextLengthsComp;
630           $allUsedTrains[$trainIndex]["next_v_max"] = $nextVMaxComp;
631         } else {
632           return array($nextSectionsComp, $nextLengthsComp, $nextVMaxComp);
633         }
634       }
635     }
636   }
637 }
```

```
638
639 // Determines the associated signal (if there is one) for a section and a direction.
640 function getSignalForSectionAndDirection(int $section, int $dir) {
641   $DB = new DB_MySQL();
642   $signal = $DB->select("SELECT `".DB_TABLE_SIGNALE_STANDORTE."`.`id` FROM `".
        ↪ DB_TABLE_SIGNALE_STANDORTE."` WHERE `".DB_TABLE_SIGNALE_STANDORTE."`.`
        ↪ freimelde_id` = $section AND `".DB_TABLE_SIGNALE_STANDORTE."`.`wirkrichtung` =
        ↪ $dir");
643   unset($DB);
644   if ($signal != null) {
645     $signal = intval(get_object_vars($signal[0])["id"]);
646   }
647   return $signal;
648 }
649
650 // Checks for all trains (no ID passed) or for one train (one ID passed)
651 // whether the train is already at the first scheduled stop or not.
652 function checkIfTrainReachedHaltepunkt ($id = false) {
653
654   global $allUsedTrains;
655   global $cacheInfraToGbt;
656   global $cacheGbtToInfra;
657
658   $checkAllTrains = true;
659   if ($id != false) {
660     $checkAllTrains = false;
661   }
662   foreach ($allUsedTrains as $trainIndex => $trainValue) {
663     if ($checkAllTrains || $trainValue["id"] == $id) {
664       $currentInfrasection = $trainValue["current_section"];
665       $currentGbt = $cacheInfraToGbt[$currentInfrasection];
666       $allInfraSections = $cacheGbtToInfra[$currentGbt];
667       for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
668         if (sizeof(array_intersect($trainValue["next_betriebsstellen_data"][$i]["
              ↪ haltepunkte"], $allInfraSections)) != 0) {
669           $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$i]["angekommen"] =
                ↪ true;
670           for ($j = 0; $j < $i; $j++) {
671             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$j]["angekommen"] =
                  ↪ true;
672           }
673         } else {
674           $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$i]["angekommen"] =
                ↪ false;
675         }
676       }
677     }
678   }
679 }
680
681 // Checks for all trains (no ID is passed) or for one train (one ID is passed)
682 // whether the route is currently set correctly so that the next operating
683 // point can be reached according to the timetable.
```

```
684 //
685 // For trains without timetable the route is always correct.
686 function checkIfFahrstrasseIsCorrrect($id = false) {
687   global $allUsedTrains;
688   $checkAllTrains = true;
689   if ($id != false) {
690     $checkAllTrains = false;
691   }
692   foreach ($allUsedTrains as $trainIndex => $trainValue) {
693     if (($checkAllTrains || $trainValue["id"] == $id) && sizeof($trainValue["error"]) ==
          ↪  0) {
694       if ($trainValue["operates_on_timetable"]) {
695         $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = false;
696         foreach ($trainValue["next_betriebsstellen_data"] as $stopIndex => $stopValue) {
697           if (!$stopValue["angekommen"]) {
698             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex]["
                  ↪ is_on_fahrstrasse"] = false;
699             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex]["
                  ↪ used_haltepunkt"] = array();
700             $indexSection = 0;
701             for ($i = 0; $i < sizeof($trainValue["next_sections"]); $i++) {
702               if ($stopValue["haltepunkte"] != null) {
703                 if (in_array($trainValue["next_sections"][$i], $stopValue["haltepunkte"])
                      ↪ ) {
704                   if ($i >= $indexSection) {
705                     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex]["
                          ↪ is_on_fahrstrasse"] = true;
706                     $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex]["
                          ↪ used_haltepunkt"] = $trainValue["next_sections"][$i];
707                     $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = true;
708                     $i = sizeof($trainValue["next_sections"]);
709                     $indexSection = $i;
710                   }
711                 }
712               }
713             }
714           } else {
715             $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][$stopIndex]["
                  ↪ is_on_fahrstrasse"] = true;
716           }
717         }
718       } else {
719         $allUsedTrains[$trainIndex]["fahrstrasse_is_correct"] = true;
720       }
721     }
722   }
723 }
724
725 // Calculates the acceleration and braking curves for all trains (if no ID is passed)
726 // or for one train (if an ID is passed). For trains running according to a timetable,
727 // for all operating points that lie on the currently set route and for trains without
728 // a timetable up to the next red signal.
729 function calculateFahrverlauf($id = false, $recalibrate = false) {
```

```
730
731   global $allUsedTrains;
732   global $cacheInfraLaenge;
733   global $timeDifference;
734   global $simulationStartTimeToday;
735   global $globalFirstHaltMinTime;
736   $checkAllTrains = true;
737   if ($id != false) {
738     $checkAllTrains = false;
739   }
740   foreach ($allUsedTrains as $trainIndex => $trainValue) {
741     $allPossibleStops = array();
742     for($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
743       if ($trainValue["next_betriebsstellen_data"][$i]["fahrplanhalt"]) {
744         array_push($allPossibleStops, $i);
745       }
746     }
747     if (sizeof($trainValue["error"]) == 0 && $trainValue["fahrstrasse_is_correct"]) {
748       if ($checkAllTrains || $trainValue["id"] == $id) {
749         if ($trainValue["operates_on_timetable"]) {
750           $nextBetriebsstelleIndex = null;
751           $allreachedInfras = array();
752           $wendet = false;
753           for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
754             if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"] &&
                     ↪ $trainValue["next_betriebsstellen_data"][$i]["is_on_fahrstrasse"] &&
                     ↪ $trainValue["next_betriebsstellen_data"][$i]["fahrplanhalt"]) {
755               $nextBetriebsstelleIndex = $i;
756               $allUsedTrains[$trainIndex]["next_bs"] = $i;
757               break;
758             }
759           }
760           if (!isset($nextBetriebsstelleIndex)) {
761             for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
762               if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"] &&
                       ↪ $trainValue["next_betriebsstellen_data"][$i]["is_on_fahrstrasse"]) {
763                 $nextBetriebsstelleIndex = $i;
764                 break;
765               }
766             }
767           }
768           if (isset($nextBetriebsstelleIndex)) {
769             if ($allUsedTrains[$trainIndex]["next_bs"] != $trainValue["
                     ↪ next_betriebsstellen_data"][$nextBetriebsstelleIndex]["betriebstelle"]
                     ↪  || $recalibrate) {
770               $allUsedTrains[$trainIndex]["next_bs"] = $trainValue["
                       ↪ next_betriebsstellen_data"][$nextBetriebsstelleIndex]["betriebstelle
                       ↪ "];
771               if (intval($trainValue["next_betriebsstellen_data"][
                       ↪ $nextBetriebsstelleIndex]["zeiten"]["wendet"]) == 1) {
772                 $wendet = true;
773               }
774               for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
```

```
775                 if (!$trainValue["next_betriebsstellen_data"][$i]["angekommen"] &&
                        ↪ $trainValue["next_betriebsstellen_data"][$i]["is_on_fahrstrasse"]
                        ↪ && $i <= $nextBetriebsstelleIndex) {
776                   array_push($allreachedInfras, array("index" => $i, "infra" =>
                          ↪ $trainValue["next_betriebsstellen_data"][$i]["used_haltepunkt"])
                          ↪ );
777                 }
778               }
779               $targetSection = $trainValue["next_betriebsstellen_data"][
                      ↪ $nextBetriebsstelleIndex]["used_haltepunkt"];
780               $targetPosition = $cacheInfraLaenge[$targetSection];
781               $startTime = null;
782               $endTime = null;
783               $prevBetriebsstelle = null;
784               for ($i = 0; $i < sizeof($trainValue["next_betriebsstellen_data"]); $i++) {
785                 if ($trainValue["next_betriebsstellen_data"][$i]["angekommen"]) {
786                   $prevBetriebsstelle = $i;
787                   break;
788                 }
789               }
790               if ($nextBetriebsstelleIndex == 0) {
791                 $startTime = microtime(true) + $timeDifference;
792                 $endTime = $startTime;
793               } else {
794                 $endTime = $trainValue["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["ankunft_soll_timestamp"];
795                 if (isset($prevBetriebsstelle)) {
796                   if ($trainValue["next_betriebsstellen_data"][$prevBetriebsstelle]["
                          ↪ zeiten"]["verspaetung"] > 0) {
797                     $startTime = $trainValue["next_betriebsstellen_data"][
                            ↪ $prevBetriebsstelle]["zeiten"]["abfahrt_soll_timestamp"] +
                            ↪ $trainValue["next_betriebsstellen_data"][
                            ↪ $nextBetriebsstelleIndex - 1]["zeiten"]["verspaetung"];
798                   } else {
799                     $startTime = $trainValue["next_betriebsstellen_data"][
                            ↪ $prevBetriebsstelle]["zeiten"]["abfahrt_soll_timestamp"];
800                   }
801                 } else {
802                   $startTime = microtime(true) + $timeDifference;
803                 }
804               }
805               $reachedBetriebsstele = true;
806               if ($startTime < microtime(true) + $timeDifference) {
807                 $startTime = microtime(true) + $timeDifference;
808               }
809               if (isset($trainValue["earliest_possible_start_time"])) {
810                 if ($startTime < $trainValue["earliest_possible_start_time"]) {
811                   $startTime = $trainValue["earliest_possible_start_time"];
812                 }
813               }
814               $verapetung = updateNextSpeed($trainValue, $startTime, $endTime,
                      ↪ $targetSection, $targetPosition, $reachedBetriebsstele,
                      ↪ $nextBetriebsstelleIndex, $wendet, false, $allreachedInfras);
```

```
815             if ($nextBetriebsstelleIndex != 0) {
816               // TODO: Reicht nicht einer der Einträge aus? Wenn ja, welcher?
817               $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                      ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] = $verapetung;
818               $trainValue["next_betriebsstellen_data"][$nextBetriebsstelleIndex]["
                      ↪ zeiten"]["verspaetung"] = $verapetung;
819             } else {
820               $end = $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                      ↪ $nextBetriebsstelleIndex]["zeiten"]["abfahrt_soll_timestamp"];
821               $start = $startTime;
822               if ($start + $verapetung + $globalFirstHaltMinTime < $end) {
823                 $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] = 0;
824                 $trainValue["next_betriebsstellen_data"][$nextBetriebsstelleIndex]["
                        ↪ zeiten"]["verspaetung"] = 0;
825               } else {
826                 $allUsedTrains[$trainIndex]["next_betriebsstellen_data"][
                        ↪ $nextBetriebsstelleIndex]["zeiten"]["verspaetung"] = $start +
                        ↪ $verapetung + $globalFirstHaltMinTime - $end;
827                 $trainValue["next_betriebsstellen_data"][$nextBetriebsstelleIndex]["
                        ↪ zeiten"]["verspaetung"] = $start + $verapetung +
                        ↪ $globalFirstHaltMinTime - $end;
828               }
829             }
830           }
831         } else {
832           if ($trainValue["current_speed"] > 0) {
833             emergencyBreak($trainValue["id"]);
834           }
835         }
836       } else {
837         $startTime = microtime(true) + $timeDifference;
838         if (isset($trainValue["earliest_possible_start_time"])) {
839           if ($startTime < $trainValue["earliest_possible_start_time"]) {
840             $startTime = $trainValue["earliest_possible_start_time"];
841           }
842         }
843         $endTime = $startTime;
844         $targetSection = null;
845         $targetPosition = null;
846         $reachedBetriebsstele = true;
847         $wendet = false;
848         $signalId = null;
849         for ($i = 0; $i < sizeof($trainValue["last_get_naechste_abschnitte"]); $i++) {
850           if (isset($trainValue["last_get_naechste_abschnitte"][$i]["signal_id"])) {
851             $signalId = $trainValue["last_get_naechste_abschnitte"][$i]["signal_id"];
852             $targetSection = $trainValue["last_get_naechste_abschnitte"][$i]["infra_id"
                    ↪ ];
853             $targetPosition = $cacheInfraLaenge[$targetSection];
854           }
855         }
856         if (!isset($signalId)) {
857           // gibt kein nächstes Signal
```

```
858             if ($trainValue["current_speed"] != 0) {
859               emergencyBreak($trainValue["id"]);
860             }
861           } else {
862             $signal = getSignalbegriff($signalId)[0]["geschwindigkeit"];
863             if ($signal > -25 && $signal < 0) {
864               $wendet = true;
865             }
866             updateNextSpeed($trainValue, $startTime, $endTime, $targetSection,
                  ↪ $targetPosition, $reachedBetriebsstele, $signalId, $wendet, true,
                  ↪ array());
867           }
868         }
869       }
870     } else {
871       if ($trainValue["current_speed"] != 0) {
872         emergencyBreak($trainValue["id"]);
873       }
874     }
875   }
876 }
877
878 function compareTwoNaechsteAbschnitte(int $id) {
879   global $allUsedTrains;
880   global $allTimes;
881   if (sizeof($allUsedTrains[$id]["error"]) == 0) {
882     $newSections = calculateNextSections($id, false);
883     $newNextSection = $newSections[0];
884     $newNextLenghts = $newSections[1];
885     $newNextVMax = $newSections[2];
886     $oldNextSections = $allUsedTrains[$id]["next_sections"];
887     $oldLenghts = $allUsedTrains[$id]["next_lenghts"];
888     $oldNextVMax = $allUsedTrains[$id]["next_v_max"];
889     $currentSectionOld = $allUsedTrains[$id]["current_section"];
890     $keyCurrentSection = array_search($currentSectionOld, $oldNextSections);
891     $keyLatestSection = array_key_last($oldNextSections);
892     $dataIsIdentical = true;
893     $numberOfSection = $keyLatestSection - $keyCurrentSection + 1;
894     $compareNextSections = array();
895     $compareNextLenghts = array();
896     $compareNextVMax = array();
897     for($i = $keyCurrentSection; $i <= $keyLatestSection; $i++) {
898       array_push($compareNextSections, $oldNextSections[$i]);
899       array_push($compareNextLenghts, $oldLenghts[$i]);
900       array_push($compareNextVMax, $oldNextVMax[$i]);
901     }
902     if (sizeof($newNextSection) != ($numberOfSection)) {
903       $dataIsIdentical = false;
904     } else {
905       for ($i = 0; $i < $keyLatestSection - $keyCurrentSection; $i++) {
906         if ($newNextSection[$i] != $compareNextSections[$i] || $newNextLenghts[$i] !=
              ↪ $compareNextLenghts[$i] || $newNextVMax[$i] != $compareNextVMax[$i]) {
907           $dataIsIdentical = false;
```

```
908           break;
909         }
910       }
911     }
912     if (!$dataIsIdentical) {
913       echo "Die Fahrstraße des Zuges mit der ID: ", $id, " hat sich geändert.\n";
914       calculateNextSections($id);
915       $adresse = $allUsedTrains[$id]["adresse"];
916       $allTimes[$adresse] = array();
917       checkIfFahrstrasseIsCorrrect($id);
918       calculateFahrverlauf($id);
919     }
920   }
921 }
922
923 function getCalibratedPosition ($id, $speed) {
924   global $cacheFahrzeugeAbschnitte;
925   $DB = new DB_MySQL();
926   $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` = $id")[0];
927   unset($DB);
928   if (in_array($id, array_keys($cacheFahrzeugeAbschnitte))) {
929     if ($positionReturn->unixtimestamp == $cacheFahrzeugeAbschnitte[$id]["unixtimestamp"
          ↪ ]) {
930       return array("possible" => false);
931     }
932   }
933   $timeDiff = time() - $positionReturn->unixtimestamp;
934   $position = ($speed / 3.6) * $timeDiff;
935   return array("section" => $positionReturn->infra_id, "position" => $position);
936 }
```

A.3 fahrtverlauf_functions

```
1  <?php
2
3  // TODO: current_speed = 0
4  function
5  updateNextSpeed (array $train, float $startTime, float $endTime, int $targetSectionPara,
       ↪  int $targetPositionPara, bool $reachedBetriebsstelle, string $targetSignal, bool
       ↪  $wendet, bool $freieFahrt, array $allreachedInfras) {
6
7    global $useSpeedFineTuning;
8    global $next_sections;
9    global $next_lengths;
10   global $next_v_max;
11   global $allTimes;
12   global $verzoegerung;
13   global $notverzoegerung;
14   global $currentSection;
```

```
15  global $currentPosition;
16  global $currentSpeed;
17  global $targetSpeed;
18  global $targetSection;
19  global $targetPosition;
20  global $targetTime;
21  global $indexCurrentSection;
22  global $indexTargetSection;
23  global $distanceToNextStop;
24  global $trainSpeedChange;
25  global $trainPositionChange;
26  global $trainTimeChange;
27  global $cumulativeSectionLengthEnd;
28  global $cumulativeSectionLengthStart;
29  global $keyPoints;
30  global $allUsedTrains;
31  global $globalIndexBetriebsstelleFreieFahrt;
32  global $cacheSignalIDToBetriebsstelle;
33  global $useMinTimeOnSpeed;
34  global $slowDownIfTooEarly;
35  global $globalFloatingPointNumbersRoundingError;
36
37  $emptyArray = array();
38  $keyPoints = $emptyArray;
39  $cumulativeSectionLengthStart = $emptyArray;
40  $cumulativeSectionLengthEnd = $emptyArray;
41  $next_sections = $train["next_sections"];
42  $next_lengths = $train["next_lenghts"];
43  $next_v_max = $train["next_v_max"];
44  $verzoegerung = $train["verzoegerung"];
45  $notverzoegerung = $train["notverzoegerung"];
46  $train_v_max = $train["v_max"];
47  $currentSection = $train["current_section"];
48  $currentPosition = $train["current_position"];
49  $currentSpeed = $train["current_speed"];
50  $train_length = $train["zuglaenge"];
51
52  $targetSection = $targetSectionPara;
53  $targetPosition = $targetPositionPara;
54
55  $targetSpeed = 0;
56  $targetTime = $endTime;
57
58  $indexCurrentSection = null;
59  $indexTargetSection = null;
60  $timeToNextStop = null;
61  $maxTimeToNextStop = $targetTime - $startTime;
62  $maxSpeedNextSections = 120;
63
64  $indexReachedBetriebsstelle = $targetSignal;
65
66
67  //$targetBetriebsstelle = null;
```

```
68  //$targetTime = $startTime + 210;
69
70  /*
71
72  $currentSection = 1189;
73  $targetSection = 1182;
74
75  $next_sections = array(100, 1189, 101, 102, 103, 104, 105, 106, 107, 108, 1182, 107);
76  //$next_lengths = array(4000, 190, 210, 200, 200, 200, 200, 200, 200, 200);
77  $next_lengths = array(300, 400, 300, 400, 300, 200, 400, 500, 300, 400, 300, 300);
78  $next_v_max = array(120, 120, 120, 90, 60, 60, 90, 120, 120, 100, 60, 40);
79  //$next_v_max = array(100, 90, 80, 70, 60, 50, 40, 40, 80, 100);
80  $currentPosition = 10;
81  $targetPosition = 290;
82  $train_v_max = 120;
83  $train_length = 50;
84  */
85
86
87
88
89  if (!$freieFahrt) {
90    $targetBetriebsstelle = $train["next_betriebsstellen_data"][
        ↪ $indexReachedBetriebsstelle]["betriebstelle"];
91  } else {
92    $targetBetriebsstelle = $cacheSignalIDToBetriebsstelle[intval($targetSignal)];
93  }
94
95  // TODO: Zug steht schon am Ziel (ist das nötig?)
96  if ($targetSection == $currentSection && $targetPosition == $currentPosition) {
97    // Freie Fahrt
98    if ($indexReachedBetriebsstelle == $globalIndexBetriebsstelleFreieFahrt) {
99      $indexReachedBetriebsstelle = -1;
100   }
101   $adress = $train["adresse"];
102   $return = array(array("live_position" => $targetPosition, "live_speed" =>
        ↪ $targetSpeed, "live_time" => $endTime, "live_relative_position" =>
        ↪ $targetPosition, "live_section" => $targetSection, "live_is_speed_change" =>
        ↪ false, "live_target_reached" => $reachedBetriebsstelle, "wendet" => $wendet,
        ↪ "id" => $train["id"], "betriebsstelle_name" => $targetBetriebsstelle));
103   $allTimes[$adress] = array_merge($allTimes[$adress], $return);
104   return 0;
105 }
106
107 // Wenn ein Abschnitt eine Geschwindigkeit zulässt, die größer als die v_max des Zugs
      ↪ ist, wird die Geschwindigkeit auf die v_max des Zuges beschränkt
108 if ($train_v_max != null) {
109   foreach ($next_sections as $sectionKey => $sectionValue) {
110     if ($next_v_max[$sectionKey] > $train_v_max) {
111       $next_v_max[$sectionKey] = $train_v_max;
112     }
113   }
114 }
```

```
115
116   // Index des Start- und Zielabschnitts
117   foreach ($next_sections as $sectionKey => $sectionValue) {
118     if ($sectionValue == $currentSection) {
119       $indexCurrentSection = $sectionKey;
120     }
121     if ($sectionValue == $targetSection) {
122       $indexTargetSection = $sectionKey;
123     }
124   }
125
126   $returnCumulativeSections = createCumulativeSections($indexCurrentSection,
          ↪ $indexTargetSection, $currentPosition, $targetPosition, $next_lengths);
127   $cumulativeSectionLengthStart = $returnCumulativeSections[0];
128   $cumulativeSectionLengthEnd = $returnCumulativeSections[1];
129   $cumLengthEnd = array();
130   $cumLengthStart = array();
131   $sum = 0;
132
133
134
135   foreach ($next_lengths as $index => $value) {
136     if ($index >= $indexCurrentSection) {
137       $cumLengthStart[$index] = $sum;
138       $sum += $value;
139       $cumLengthEnd[$index] = $sum;
140     }
141   }
142
143   $distanceToNextStop = $cumulativeSectionLengthEnd[$indexTargetSection];
144   if (getBrakeDistance($currentSpeed, $targetSpeed, $verzoegerung)> $distanceToNextStop
          ↪ && $currentSpeed != 0) {
145     if (!isset($distanceToNextStop)) {
146       emergencyBreak($train["id"]);
147       return 0;
148     } else {
149       emergencyBreak($train["id"], $distanceToNextStop);
150       return 0;
151     }
152   }
153
154
155   // TODO: Als Funktion ausgliedern
156   global $next_v_max_mod;
157   global $next_lengths_mod;
158   $next_v_max_mod = array();
159   $next_lengths_mod = array();
160
161   global $indexCurrentSectionMod;
162   global $indexTargetSectionMod;
163   $indexCurrentSectionMod = null;
164   $indexTargetSectionMod = null;
165
```

```
166   if ($indexCurrentSection == $indexTargetSection) {
167     $next_lengths_mod = $next_lengths;
168     $next_v_max_mod = $next_v_max;
169     $indexCurrentSectionMod = $indexCurrentSection;
170     $indexTargetSectionMod = $indexTargetSection;
171     $next_lengths_mod[$indexTargetSectionMod] = $targetPosition;
172   } else {
173     $startPosition = 0;
174     $indexStartPosition = null;
175     $indexEndPosition = null;
176     do {
177       $reachedTargetSection = false;
178       for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
179         if ($startPosition >= $cumLengthStart[$j] && $startPosition < $cumLengthEnd[$j])
              ↪ {
180           $indexStartPosition = $j;
181         }
182       }
183       $endPosition = $cumLengthEnd[$indexStartPosition] + $train_length;
184       $current_v_max = $next_v_max[$indexStartPosition];
185       if ($endPosition >= $cumLengthEnd[$indexTargetSection]) {
186         $indexEndPosition = $indexTargetSection;
187         $endPosition = $cumLengthEnd[$indexTargetSection - 1] + $targetPosition;
188         $reachedTargetSection = true;
189       } else {
190         for ($j = $indexCurrentSection; $j <= $indexTargetSection; $j++) {
191           if ($endPosition >= $cumLengthStart[$j] && $endPosition < $cumLengthEnd[$j]) {
192             $indexEndPosition = $j;
193           }
194         }
195       }
196       for ($j = $indexStartPosition + 1; $j <= $indexEndPosition; $j++) {
197         if ($next_v_max[$j] < $current_v_max) {
198           $endPosition = $cumLengthStart[$j];
199           $indexEndPosition = $j - 1;
200         }
201       }
202       if ($reachedTargetSection) {
203         if (!($endPosition >= $distanceToNextStop)) {
204           $reachedTargetSection = false;
205         }
206       }
207       array_push($next_lengths_mod, ($endPosition - $startPosition));
208       array_push($next_v_max_mod, $current_v_max);
209       $startPosition = $endPosition;
210     } while (!$reachedTargetSection);
211     $indexCurrentSectionMod = array_key_first($next_lengths_mod);
212     $indexTargetSectionMod = array_key_last($next_lengths_mod);
213   }
214
215   $returnCumulativeSectionsMod = createCumulativeSections($indexCurrentSectionMod,
        ↪ $indexTargetSectionMod, $currentPosition, $next_lengths_mod[
        ↪ $indexTargetSectionMod], $next_lengths_mod);
```

```
216
217   global $cumulativeSectionLengthStartMod;
218   global $cumulativeSectionLengthEndMod;
219   $cumulativeSectionLengthStartMod = $returnCumulativeSectionsMod[0];
220   $cumulativeSectionLengthEndMod = $returnCumulativeSectionsMod[1];
221
222   $minTimeOnSpeedIsPossible = checkIfItsPossible();
223
224
225   $v_maxFirstIteration = getVMaxBetweenTwoPoints($distanceToNextStop, $currentSpeed,
          ↪ $targetSpeed);
226
227   // Anpassung an die maximale Geschwindigkeit auf der Strecke
228   for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
229     if ($next_v_max[$i] > $maxSpeedNextSections) {
230       $maxSpeedNextSections = $next_v_max[$i];
231     }
232   }
233
234   if ($maxSpeedNextSections < $v_maxFirstIteration) {
235     $v_maxFirstIteration = $maxSpeedNextSections;
236   }
237
238   // Key Points für die erste Iteration erstellen.
239   array_push($keyPoints, createKeyPoint(0, getBrakeDistance($currentSpeed,
          ↪ $v_maxFirstIteration, $verzoegerung), $currentSpeed, $v_maxFirstIteration));
240   array_push($keyPoints, createKeyPoint(($distanceToNextStop - getBrakeDistance(
          ↪ $v_maxFirstIteration, $targetSpeed, $verzoegerung)), $distanceToNextStop,
          ↪ $v_maxFirstIteration, $targetSpeed));
241
242   //function keyPoints => trainChangeArrays
243   $trainChange = convertKeyPointsToTrainChangeArray($keyPoints);
244   $trainPositionChange = $trainChange[0];
245   $trainSpeedChange = $trainChange[1];
246
247   $speedOverPositionAllIterations = array();
248
249   while (checkIfTrainIsToFastInCertainSections()["failed"]) {
250     // saves the keyPoints local
251     $tempKeyPoints = $keyPoints;
252
253     // berechnet die "live Daten"
254     $trainChange = createTrainChanges(true);
255     $trainPositionChange = $trainChange[0];
256     $trainSpeedChange = $trainChange[1];
257
258     array_push($speedOverPositionAllIterations, array($trainPositionChange,
            ↪ $trainSpeedChange));
259
260     // suche nach Fehlern und neuberechnung...
261     $keyPoints = recalculateKeyPoints($tempKeyPoints);
262
263     $localKeyPointsTwo = array();
```

```
264
265     // Löschung von zwei aufernanderfolgenden "unnötigen" KeyPoints
266     for ($i = 0; $i < sizeof($keyPoints); $i++) {
267       if ($i < sizeof($keyPoints) - 1) {
268         if (!($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"])) {
269           array_push($localKeyPointsTwo, $keyPoints[$i]);
270         } else {
271           $i++;
272         }
273       } else {
274         array_push($localKeyPointsTwo, $keyPoints[$i]);
275       }
276     }
277
278     $keyPoints = $localKeyPointsTwo;
279     $trainChange = createTrainChanges(true);
280     $trainPositionChange = $trainChange[0];
281     $trainSpeedChange = $trainChange[1];
282     $count++;
283   }
284
285   // Adding time to first KeyPoint
286   $keyPoints[0]["time_0"] = $startTime;
287   $keyPoints = deleteDoubledKeyPoints($keyPoints);
288   $keyPoints = calculateTimeFromKeyPoints();
289   if ($useMinTimeOnSpeed && $minTimeOnSpeedIsPossible) {
290     array_push($speedOverPositionAllIterations, array($trainPositionChange,
          ↪ $trainSpeedChange));
291     toShortOnOneSpeed();
292   }
293   $trainChange = createTrainChanges(true);
294   $trainPositionChange = $trainChange[0];
295   $trainSpeedChange = $trainChange[1];
296   $timeToNextStop = end($keyPoints)["time_1"] - $keyPoints[0]["time_0"];
297
298   // Train arrives at destination with a delay
299   if (!$freieFahrt) {
300     if ($timeToNextStop > $maxTimeToNextStop) {
301       // TODO: Als allgemeine Info am Anfang der Funktion...
302       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";
303     } else {
304       echo "Aktuell benötigt der Zug mit der Adresse ", $train["adresse"], " ",
            ↪ number_format($timeToNextStop, 2), " Sekunden, obwohl er ", number_format(
            ↪ $maxTimeToNextStop, 2), " Sekunden zur Verfügung hat.\n";
305       if ($slowDownIfTooEarly) {
306         echo "Evtl. könnte der Zug zwischendurch die Geschwindigkeit verringern, um
              ↪ Energie zu sparen.";
307         array_push($speedOverPositionAllIterations, array($trainPositionChange,
              ↪ $trainSpeedChange));
```

```
308         $keyPointsPreviousStep = array();
309         $finish = false;
310         $possibleSpeedRange = null;
311         $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
312         while ($returnSpeedDecrease["possible"] && !$finish) {
313           $possibleSpeedRange = findMaxSpeed($returnSpeedDecrease);
314           if ($possibleSpeedRange["min_speed"] == $possibleSpeedRange["max_speed"]) {
315             break;
316           }
317           $localKeyPoints = $keyPoints;
318           $newCalculatedTime = null;
319           $newKeyPoints = null;
320           for ($i = $possibleSpeedRange["max_speed"]; $i >= $possibleSpeedRange["
                  ↪ min_speed"]; $i = $i - 10) {
321             $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["speed_1"] = $i
                    ↪ ;
322             $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["speed_0"]
                    ↪ = $i;
323             $localKeyPoints[$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"]);
324             $localKeyPoints[$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));
325             $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
326             $newCalculatedTime = $localKeyPoints[array_key_last($localKeyPoints)]["time_1
                    ↪ "];
327             if ($i == 10) {
328               if ($newCalculatedTime > $maxTimeToNextStop) {
329                 $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["speed_1"]
                        ↪ = $i + 10;
330                 $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["
                        ↪ speed_0"] = $i + 10;
331                 $localKeyPoints[$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"]);
332                 $localKeyPoints[$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));
333               }
334               $finish = true;
335               $newKeyPoints = $localKeyPoints;
336               break;
337             }
338             if (($newCalculatedTime - $startTime) > $maxTimeToNextStop) {
```

```
339             if ($i == $possibleSpeedRange["max_speed"]) {
340               $localKeyPoints = $keyPointsPreviousStep;
341               $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
342               $keyPoints = $localKeyPoints;
343               $finish = true;
344               break;
345             }
346             $localKeyPoints[$possibleSpeedRange["first_key_point_index"]]["speed_1"] =
                  ↪ $i + 10;
347             $localKeyPoints[$possibleSpeedRange["first_key_point_index"] + 1]["speed_0"
                  ↪ ] = $i + 10;
348             $localKeyPoints[$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"]);
349             $localKeyPoints[$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));
350             $newKeyPoints = $localKeyPoints;
351             $finish = true;
352             $keyPoints = $localKeyPoints;
353             break;
354           }
355           if ($i == $possibleSpeedRange["min_speed"]) {
356             $newKeyPoints = $localKeyPoints;
357             $newKeyPoints = deleteDoubledKeyPoints($newKeyPoints);
358             $keyPoints = $newKeyPoints;
359             break;
360           }
361           $newKeyPoints = $localKeyPoints;
362         }
363         $keyPointsPreviousStep = $localKeyPoints;
364         if ($newKeyPoints != null) {
365           $keyPoints = $newKeyPoints;
366         }
367         $keyPoints = deleteDoubledKeyPoints($keyPoints);
368         $returnSpeedDecrease = checkIfTheSpeedCanBeDecreased();
369       }
370       $keyPoints = calculateTimeFromKeyPoints();
371       if ($useSpeedFineTuning && $returnSpeedDecrease["possible"]) {
372         $trainChangeReturn = createTrainChanges(true);
373         $trainPositionChange = $trainChangeReturn[0];
374         $trainSpeedChange = $trainChangeReturn[1];
375         $newCalculatedTime = $keyPoints[array_key_last($keyPoints)]["time_1"];
376         speedFineTuning(($maxTimeToNextStop - ($newCalculatedTime - $startTime)),
                  ↪ $returnSpeedDecrease["range"][array_key_last($returnSpeedDecrease["range
                  ↪ "])]["KeyPoint_index"]);
377       }
378       $keyPoints = calculateTimeFromKeyPoints();
379       $timeToNextStop = end($keyPoints)["time_1"] - $keyPoints[0]["time_0"];
```

```
380         echo "\nDurch die Anpassung der Geschwindigkeit benötigt der Zug mit der Adresse
                ↪ ", $train["adresse"], " jetzt ", number_format($timeToNextStop, 2), "
                ↪ Sekunden bis\n";
381         if (abs($timeToNextStop - $maxTimeToNextStop) <
                ↪ $globalFloatingPointNumbersRoundingError) {
382           echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und wird
                  ↪ diesen genau pünktlich erreichen.\n";
383         } else if (($timeToNextStop - $maxTimeToNextStop) > 0) {
384           echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und wird
                  ↪ diesen mit einer Verspätung von ", number_format($timeToNextStop -
                  ↪ $maxTimeToNextStop, 2), " Sekunden erreichen.\n";
385         } else {
386           echo "zum nächsten planmäßigen Halt (", $targetBetriebsstelle, ") und wird
                  ↪ diesen ", number_format($timeToNextStop - $maxTimeToNextStop, 2), "
                  ↪ Sekunden zu früh erreichen.\n";
387         }
388       } else {
389         echo "Dadurch, dass \$slowDownIfTooEarly = true ist, wird das Fahrzeug ",
                ↪ number_format($maxTimeToNextStop - $timeToNextStop, 2), " Sekunden zu frü
                ↪ h am Ziel ankommen.";
390       }
391     }
392   } else {
393     echo "Der Zug mit der Adresse ", $train["adresse"], " fährt aktuell ohne Fahrplan
          ↪ bis zum nächsten auf Halt stehendem Signal (Signal ID: ", $targetSignal, ",
          ↪ Betriebsstelle: ", $targetBetriebsstelle,").\n";
394   }
395
396   $returnTrainChanges = createTrainChanges(false);
397   $trainPositionChange = $returnTrainChanges[0];
398   $trainSpeedChange = $returnTrainChanges[1];
399   $trainTimeChange = $returnTrainChanges[2];
400   $trainRelativePosition = $returnTrainChanges[3];
401   $trainSection = $returnTrainChanges[4];
402   $trainIsSpeedChange = $returnTrainChanges[5];
403   $trainTargetReached = array();
404   $trainBetriebsstelleName = array();
405   $trainWendet = array();
406   $allReachedTargets = array();
407   $allreachedInfrasIndex = array();
408   $allreachedInfrasID = array();
409   $allreachedInfrasUsed = array();
410   foreach ($allreachedInfras as $value) {
411     array_push($allreachedInfrasIndex, $value["index"]);
412     array_push($allreachedInfrasID, $value["infra"]);
413   }
414   foreach ($trainPositionChange as $key => $value) {
415     $trainBetriebsstelleName[$key] = $targetBetriebsstelle;
416     if (array_key_last($trainPositionChange) != $key) {
417       $trainTargetReached[$key] = false;
418       $trainWendet[$key] = false;
419     } else {
420       if ($wendet) {
```

```
421         $trainWendet[$key] = true;
422       } else {
423         $trainWendet[$key] = false;
424       }
425       if ($reachedBetriebsstelle) {
426         $trainTargetReached[$key] = true;
427       } else {
428         $trainTargetReached[$key] = false;
429       }
430     }
431   }
432   for($i = sizeof($trainSection) - 1; $i >= 0; $i--) {
433     if (in_array($trainSection[$i], $allreachedInfrasID) && !in_array($trainSection[$i],
          ↪  $allreachedInfrasUsed)) {
434       array_push($allreachedInfrasUsed, $trainSection[$i]);
435       $Infraindex = array_search($trainSection[$i], $allreachedInfrasID);
436       $allReachedTargets[$i] = $allreachedInfrasIndex[$Infraindex];
437     } else {
438       $allReachedTargets[$i] = null;
439     }
440   }
441   ksort($allReachedTargets);
442   $returnArray = array();
443   $adress = $train["adresse"];
444   $trainID = array();
445   $id = $train["id"];
446   foreach ($trainPositionChange as $key => $value) {
447     $trainID[$key] = $id;
448   }
449   foreach ($trainPositionChange as $trainPositionChangeIndex =>
          ↪ $trainPositionChangeValue) {
450     array_push($returnArray, array("live_position" => $trainPositionChangeValue,
451       "live_speed" => $trainSpeedChange[$trainPositionChangeIndex],
452       "live_time" => $trainTimeChange[$trainPositionChangeIndex],
453       "live_relative_position" => $trainRelativePosition[$trainPositionChangeIndex],
454       "live_section" => $trainSection[$trainPositionChangeIndex],
455       "live_is_speed_change" => $trainIsSpeedChange[$trainPositionChangeIndex],
456       "live_target_reached" => $trainTargetReached[$trainPositionChangeIndex],
457       "id" => $trainID[$trainPositionChangeIndex],
458       "wendet" => $trainWendet[$trainPositionChangeIndex],
459       "betriebsstelle" => $trainBetriebsstelleName[$trainPositionChangeIndex],
460       "live_all_targets_reached" => $allReachedTargets[$trainPositionChangeIndex]));
461   }
462   $allTimes[$adress] = $returnArray;
463   var_dump($keyPoints);
464   var_dump($allTimes[$adress]);
465   safeTrainChangeToJSONFile($indexCurrentSection, $indexTargetSection,
          ↪ $indexCurrentSectionMod, $indexTargetSectionMod,
          ↪ $speedOverPositionAllIterations);
466   return (end($trainTimeChange) - $trainTimeChange[0]) - ($endTime - $startTime);
467 }
468
469 function getBrakeDistance (float $v_0, float $v_1, float $verzoegerung) {
```

```
470   return abs(0.5 * ((pow($v_0/3.6,2)-pow($v_1/3.6, 2))/($verzoegerung)));
471 }
472
473 // TODO: Überarbeitung
474 function getVMaxBetweenTwoPoints(float $distance, int $v_0, int $v_1) {
475   global $verzoegerung;
476   global $globalFloatingPointNumbersRoundingError;
477
478   $v_max = array();
479   for ($i = 0; $i <= 120; $i = $i + 10) {
480     if ((getBrakeDistance($v_0, $i, $verzoegerung) + getBrakeDistance($i, $v_1,
          ↪ $verzoegerung)) < ($distance + $globalFloatingPointNumbersRoundingError)) {
481       array_push($v_max, $i);
482     }
483   }
484   if (sizeof($v_max) == 0) {
485     if ($v_0 == 0 && $v_1 == 0 && $distance > 0) {
486       echo "Der zug müsste langsamer als 10 km/h fahren, um das Ziel zu erreichen.";
487     } else {
488       // TODO: Notbremsung
489     }
490   } else {
491     if ($v_0 == $v_1 && max($v_max) < $v_0) {
492       $v_max = array($v_0);
493     }
494   }
495   return max($v_max);
496 }
497
498 function createKeyPoint (float $position_0, float $position_1, int $speed_0, int
      ↪ $speed_1) {
499   return array("position_0" => $position_0, "position_1" => $position_1, "speed_0" =>
        ↪ $speed_0, "speed_1" => $speed_1);
500 }
501
502 // TODO: Funktion doppelt sich... (evtl. hilfreich, weil diese Funktion nur
      ↪ Geschwindigkeit und Position zurückgibt)
503 function convertKeyPointsToTrainChangeArray (array $keyPoints) {
504   global $verzoegerung;
505
506   $trainSpeedChangeReturn = array();
507   $trainPositionChnageReturn = array();
508
509   array_push($trainPositionChnageReturn, $keyPoints[0]["position_0"]);
510   array_push($trainSpeedChangeReturn, $keyPoints[0]["speed_0"]);
511
512   for ($i = 0; $i <= (sizeof($keyPoints) - 2); $i++) {
513     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
514       for ($j = $keyPoints[$i]["speed_0"]; $j < $keyPoints[$i]["speed_1"]; $j = $j + 2)
            ↪ {
515         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
              ↪ getBrakeDistance($j, ($j + 2), $verzoegerung)));
516         array_push($trainSpeedChangeReturn, ($j + 2));
```

```
517       }
518     } elseif ($keyPoints[$i]["speed_0"] > $keyPoints[$i]["speed_1"]) {
519       for ($j = $keyPoints[$i]["speed_0"]; $j > $keyPoints[$i]["speed_1"]; $j = $j - 2)
              ↪ {
520         array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
                ↪ getBrakeDistance($j, ($j - 2), $verzoegerung)));
521         array_push($trainSpeedChangeReturn, ($j - 2));
522       }
523     }
524     array_push($trainPositionChnageReturn, $keyPoints[$i + 1]["position_0"]);
525     array_push($trainSpeedChangeReturn, $keyPoints[$i + 1]["speed_0"]);
526   }
527
528   if (end($keyPoints)["speed_0"] < end($keyPoints)["speed_1"]) {
529     for ($j = end($keyPoints)["speed_0"]; $j < end($keyPoints)["speed_1"]; $j = $j + 2)
            ↪ {
530       array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
              ↪ getBrakeDistance($j, ($j + 2), $verzoegerung)));
531       array_push($trainSpeedChangeReturn, ($j + 2));
532     }
533   } else if (end($keyPoints)["speed_0"] > end($keyPoints)["speed_1"]) {
534     for ($j = end($keyPoints)["speed_0"]; $j > end($keyPoints)["speed_1"]; $j = $j - 2)
            ↪ {
535       array_push($trainPositionChnageReturn, (end($trainPositionChnageReturn) +
              ↪ getBrakeDistance($j, ($j - 2), $verzoegerung)));
536       array_push($trainSpeedChangeReturn, ($j - 2));
537     }
538   }
539   return array($trainPositionChnageReturn, $trainSpeedChangeReturn);
540 }
541
542 function safeTrainChangeToJSONFile(int $indexCurrentSection, int $indexTargetSection,
      ↪ int $indexCurrentSectionMod, int $indexTargetSectionMod, array
      ↪ $speedOverPositionAllIterations) {
543   global $trainPositionChange;
544   global $trainSpeedChange;
545   global $next_v_max;
546   global $cumulativeSectionLengthEnd;
547   global $next_v_max_mod;
548   global $cumulativeSectionLengthEndMod;
549
550   $speedOverPosition = array_map('toArr', $trainPositionChange, $trainSpeedChange);
551   $speedOverPosition = json_encode($speedOverPosition);
552   $fp = fopen('../json/speedOverPosition.json', 'w');
553   fwrite($fp, $speedOverPosition);
554   fclose($fp);
555
556   $v_maxFromUsedSections = array();
557   for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
558     array_push($v_maxFromUsedSections, $next_v_max[$i]);
559   }
560   $VMaxOverCumulativeSections = array_map('toArr', $cumulativeSectionLengthEnd,
        ↪ $v_maxFromUsedSections);
```

```
561   $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSections);
562   $fp = fopen('../json/VMaxOverCumulativeSections.json', 'w');
563   fwrite($fp, $VMaxOverPositionsJSon);
564   fclose($fp);
565
566   $v_maxFromUsedSections = array();
567   for ($i = $indexCurrentSectionMod; $i <= $indexTargetSectionMod; $i++) {
568     array_push($v_maxFromUsedSections, $next_v_max_mod[$i]);
569   }
570   $VMaxOverCumulativeSectionsMod = array_map('toArr', $cumulativeSectionLengthEndMod,
          ↪ $v_maxFromUsedSections);
571   $VMaxOverPositionsJSon = json_encode($VMaxOverCumulativeSectionsMod);
572   $fp = fopen('../json/VMaxOverCumulativeSectionsMod.json', 'w');
573   fwrite($fp, $VMaxOverPositionsJSon);
574   fclose($fp);
575
576   $jsonReturn = array();
577   for ($i = 0; $i < sizeof($speedOverPositionAllIterations); $i++) {
578     $iteration = array_map('toArr', $speedOverPositionAllIterations[$i][0],
          ↪ $speedOverPositionAllIterations[$i][1]);
579     array_push($jsonReturn, $iteration);
580   }
581   $speedOverPosition = json_encode($jsonReturn);
582   $fp = fopen('../json/speedOverPosition_prevIterations.json', 'w');
583   fwrite($fp, $speedOverPosition);
584   fclose($fp);
585 }
586
587 function checkIfTrainIsToFastInCertainSections() {
588
589   global $trainPositionChange;
590   global $trainSpeedChange;
591   global $cumulativeSectionLengthStartMod;
592   global $next_v_max_mod;
593   global $indexTargetSectionMod;
594
595   $faildSections = array();
596
597   foreach ($trainPositionChange as $trainPositionChangeKey => $trainPositionChangeValue)
          ↪ {
598     foreach ($cumulativeSectionLengthStartMod as $cumulativeSectionLengthStartKey =>
          ↪ $cumulativeSectionLengthStartValue) {
599       if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
600         if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
              ↪ $cumulativeSectionLengthStartKey - 1]) {
601           array_push($faildSections, ($cumulativeSectionLengthStartKey -1));
602         }
603         break;
604       } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
605         if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
606           if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                ↪ $cumulativeSectionLengthStartKey]) {
607             array_push($faildSections, $cumulativeSectionLengthStartKey);
```

```
608           }
609           break;
610         }
611       }
612     }
613   }
614
615   if (sizeof($faildSections) == 0) {
616     return array("failed" => false);
617   } else {
618     return array("failed" => true, "failed_sections" => array_unique($faildSections));
619   }
620 }
621
622 function deleteDoubledKeyPoints($temporaryKeyPoints) {
623
624   do {
625     $foundDoubledKeyPoints = false;
626     $doubledIndex = array();
627     for ($i = 1; $i < (sizeof($temporaryKeyPoints) - 1); $i++) {
628       if ($temporaryKeyPoints[$i]["speed_0"] == $temporaryKeyPoints[$i]["speed_1"]) {
629         $foundDoubledKeyPoints = true;
630         array_push($doubledIndex, $i);
631       }
632     }
633
634     foreach ($doubledIndex as $index) {
635       unset($temporaryKeyPoints[$index]);
636     }
637     $temporaryKeyPoints = array_values($temporaryKeyPoints);
638   } while ($foundDoubledKeyPoints);
639   return $temporaryKeyPoints;
640 }
641
642 function calculateTrainTimeChange() {
643
644   global $keyPoints;
645   global $verzoegerung;
646
647   $returnAllTimes = array();
648   $returnAllTimes[0] = $keyPoints[0]["time_0"];
649
650   for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
651     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
652       for ($j = $keyPoints[$i]["speed_0"] + 2; $j <= $keyPoints[$i]["speed_1"]; $j = $j
          ↪ + 2) {
653         array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($j - 2, $j,
            ↪ $verzoegerung))));
654
655       }
656       array_push($returnAllTimes, (end($returnAllTimes) + distanceWithSpeedToTime(
          ↪ $keyPoints[$i]["speed_1"], ($keyPoints[$i + 1]["position_0"] - $keyPoints[
          ↪ $i]["position_1"]))));
```

```
657
658     } else if ($keyPoints[$i]["speed_0"] > $keyPoints[$i]["speed_1"]) {
659       for ($j = $keyPoints[$i]["speed_0"] - 2; $j >= $keyPoints[$i]["speed_1"]; $j = $j
              ↪ - 2) {
660         array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($j + 2, $j,
                ↪ $verzoegerung))));
661
662       }
663       array_push($returnAllTimes, (end($returnAllTimes) + distanceWithSpeedToTime(
              ↪ $keyPoints[$i]["speed_1"], ($keyPoints[$i + 1]["position_0"] - $keyPoints[
              ↪ $i]["position_1"]))));
664     }
665   }
666
667   if ($keyPoints[array_key_last($keyPoints)]["speed_0"] < $keyPoints[array_key_last(
          ↪ $keyPoints)]["speed_1"]) {
668     for ($i = ($keyPoints[array_key_last($keyPoints)]["speed_0"] + 2); $i <= $keyPoints[
            ↪ array_key_last($keyPoints)]["speed_1"]; $i = $i + 2) {
669       array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($i - 2, $i,
              ↪ $verzoegerung))));
670     }
671   } else if ($keyPoints[array_key_last($keyPoints)]["speed_0"] > $keyPoints[
          ↪ array_key_last($keyPoints)]["speed_1"]) {
672     for ($i = ($keyPoints[array_key_last($keyPoints)]["speed_0"] - 2); $i >= $keyPoints[
            ↪ array_key_last($keyPoints)]["speed_1"]; $i = $i - 2) {
673       array_push($returnAllTimes, (end($returnAllTimes) + (getBrakeTime($i + 2, $i,
              ↪ $verzoegerung))));
674     }
675   }
676   return $returnAllTimes;
677 }
678
679 function getBrakeTime (float $v_0, float $v_1, float $verzoegerung) {
680   if ($v_0 < $v_1) {
681     return (($v_1/3.6)/$verzoegerung) - (($v_0/3.6)/$verzoegerung);
682   } elseif ($v_0 > $v_1) {
683     return (($v_0/3.6)/$verzoegerung) - (($v_1/3.6)/$verzoegerung);
684   } else {
685     return 0;
686   }
687 }
688
689 // TODO: Darf nichts negatives zurückgeben!
690 // TODO: Muss auch was neg. zurück geben für minTimeOnOneSPeed!
691 function distanceWithSpeedToTime (int $v, float $distance) {
692   if ($distance == 0) {
693     return 0;
694   }
695   return (($distance)/($v / 3.6));
696 }
697
698 function calculateTimeFromKeyPoints($inputKeyPoints = null, $skippingKeys = null) {
699
```

```
700   global $keyPoints;
701   global $verzoegerung;
702
703   if ($inputKeyPoints == null) {
704     $localKeyPoints = $keyPoints;
705   } else {
706     $localKeyPoints = $inputKeyPoints;
707   }
708   $keys = array_keys($localKeyPoints);
709   if ($skippingKeys != null) {
710     foreach ($skippingKeys as $skip) {
711       unset($keys[array_search($skip, $keys)]);
712     }
713   }
714   $keys = array_values($keys);
715
716   for ($i = 0; $i < (sizeof($keys) - 1); $i++) {
717     $localKeyPoints[$keys[$i]]["time_1"] = getBrakeTime($localKeyPoints[$keys[$i]]["
          ↪ speed_0"], $localKeyPoints[$keys[$i]]["speed_1"], $verzoegerung) +
          ↪ $localKeyPoints[$keys[$i]]["time_0"];
718     $localKeyPoints[$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"];
719   }
720
721   $localKeyPoints[end($keys)]["time_1"] = getBrakeTime($localKeyPoints[end($keys)]["
        ↪ speed_0"], $localKeyPoints[end($keys)]["speed_1"], $verzoegerung) +
        ↪ $localKeyPoints[end($keys)]["time_0"];
722   return $localKeyPoints;
723 }
724
725 // TODO: Only Position and Speed
726 function createTrainChanges(bool $onlyPositionAndSpeed) {
727
728   global $keyPoints;
729   global $verzoegerung;
730   global $cumulativeSectionLengthStart;
731   global $cumulativeSectionLengthEnd;
732   global $next_sections;
733   global $indexCurrentSection;
734   global $indexTargetSection;
735   global $currentPosition;
736   global $globalFloatingPointNumbersRoundingError;
737   global $globalDistanceUpdateInterval;
738
739   $returnTrainSpeedChange = array();
740   $returnTrainTimeChange = array();
741   $returnTrainPositionChange = array();
742   $returnTrainRelativePosition = array();
743   $returnTrainSection = array();
744   $returnIsSpeedChange = array();
745
```

```
746   // Alle bis auf den letzten Key Point
747   // Erstellt immer alle Daten zwischen KeyPoint Anfang und dem letzten Wert vor dem nä
          ↪ chsten KeyPoint
748   for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
749     array_push($returnTrainTimeChange, $keyPoints[$i]["time_0"]);
750     array_push($returnTrainSpeedChange, $keyPoints[$i]["speed_0"]);
751     array_push($returnTrainPositionChange, $keyPoints[$i]["position_0"]);
752     array_push($returnIsSpeedChange, true);
753     if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
754       // Speichert alle ab dem zweiten Wert bis zum letzten Wert
755       for ($j = ($keyPoints[$i]["speed_0"] + 2); $j <= $keyPoints[$i]["speed_1"]; $j =
              ↪ $j + 2) {
756         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
                ↪ getBrakeDistance(($j - 2), $j, $verzoegerung)));
757         array_push($returnTrainSpeedChange, $j);
758         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (getBrakeTime((
                ↪ $j - 2), $j, $verzoegerung))));
759         array_push($returnIsSpeedChange, true);
760       }
761     } else {
762       for ($j = ($keyPoints[$i]["speed_0"] - 2); $j >= $keyPoints[$i]["speed_1"]; $j =
              ↪ $j - 2) {
763         array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
                ↪ getBrakeDistance(($j + 2), $j, $verzoegerung)));
764         array_push($returnTrainSpeedChange, $j);
765         array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (getBrakeTime((
                ↪ $j + 2), $j, $verzoegerung))));
766         array_push($returnIsSpeedChange, true);
767       }
768     }
769     $startPosition = $keyPoints[$i]["position_1"];
770     $endPosition = $keyPoints[$i + 1]["position_0"];
771     $speedToNextKeyPoint = $keyPoints[$i]["speed_1"];
772     $distanceForOneTimeInterval = $speedToNextKeyPoint / 3.6;
773     for ($position = $startPosition + $distanceForOneTimeInterval; $position <
            ↪ $endPosition; $position = $position + $distanceForOneTimeInterval) {
774       array_push($returnTrainPositionChange, $position);
775       array_push($returnTrainSpeedChange, $speedToNextKeyPoint);
776       array_push($returnTrainTimeChange, end($returnTrainTimeChange) +
              ↪ $globalDistanceUpdateInterval);
777       array_push($returnIsSpeedChange, false);
778     }
779   }
780   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"],$verzoegerung));
781   array_push($returnTrainSpeedChange, $keyPoints[array_key_last($keyPoints)]["speed_0"])
          ↪ ;
782   array_push($returnTrainTimeChange, $keyPoints[array_key_last($keyPoints)]["time_0"]);
783   array_push($returnIsSpeedChange, true);
784   // letzter KeyPoint
785   if ($keyPoints[array_key_last($keyPoints)]["speed_0"] < $keyPoints[array_key_last(
          ↪ $keyPoints)]["speed_1"]) {
```

```
786     for ($j = ($keyPoints[array_key_last($keyPoints)]["speed_0"] + 2); $j <= $keyPoints[
            ↪ array_key_last($keyPoints)]["speed_1"]; $j = $j + 2) {
787       array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
              ↪ getBrakeDistance(($j - 2), $j, $verzoegerung)));
788       array_push($returnTrainSpeedChange, $j);
789       array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (getBrakeTime((
              ↪ $j - 2), $j, $verzoegerung))));
790       array_push($returnIsSpeedChange, true);
791     }
792   } else {
793     for ($j = ($keyPoints[array_key_last($keyPoints)]["speed_0"] - 2); $j >= $keyPoints[
            ↪ array_key_last($keyPoints)]["speed_1"]; $j = $j - 2) {
794       array_push($returnTrainPositionChange, (end($returnTrainPositionChange) +
              ↪ getBrakeDistance(($j + 2), $j, $verzoegerung)));
795       array_push($returnTrainSpeedChange, $j);
796       array_push($returnTrainTimeChange, (end($returnTrainTimeChange) + (getBrakeTime((
              ↪ $j + 2), $j, $verzoegerung))));
797       array_push($returnIsSpeedChange, true);
798     }
799   }
800   if ($onlyPositionAndSpeed) {
801     return array($returnTrainPositionChange, $returnTrainSpeedChange);
802   } else {
803     // Erstellt die relativen Positionen und Abschnitte zu den absoluten Werten.
            ↪ position
804     foreach ($returnTrainPositionChange as $absolutPositionKey => $absolutPositionValue)
            ↪  {
805       foreach ($cumulativeSectionLengthStart as $sectionStartKey => $sectionStartValue)
              ↪ {
806         if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue <
                ↪ $cumulativeSectionLengthEnd[$sectionStartKey]) {
807           if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
                  ↪ $indexTargetSection) {
808             $returnTrainRelativePosition[$absolutPositionKey] = $absolutPositionValue +
                    ↪ $currentPosition;
809           } else if ($sectionStartKey == $indexCurrentSection) {
810             $returnTrainRelativePosition[$absolutPositionKey] = $absolutPositionValue +
                    ↪ $currentPosition;
811           } else if ($sectionStartKey == $indexCurrentSection) {
812             $returnTrainRelativePosition[$absolutPositionKey] = $absolutPositionValue -
                    ↪ $sectionStartValue;
813           } else {
814             $returnTrainRelativePosition[$absolutPositionKey] = $absolutPositionValue -
                    ↪ $sectionStartValue;
815           }
816           break;
817         } else if ($absolutPositionKey == array_key_last($returnTrainPositionChange) &&
                ↪ abs($absolutPositionValue - floatval($cumulativeSectionLengthEnd[
                ↪ $sectionStartKey])) < $globalFloatingPointNumbersRoundingError) {
818           $returnTrainRelativePosition[$absolutPositionKey] = $cumulativeSectionLengthEnd
                  ↪ [$sectionStartKey] - $sectionStartValue;
819           break;
820         } else {
```

```
821           debugMessage("Eine absolute Position konnte keine relativen Position in einem
                  ↪ Abschnitt zugeordnet werden!");
822         }
823       }
824     }
825
826     // Erstellt die relativen Positionen und Abschnitte zu den absoluten Werten. section
827     foreach ($returnTrainPositionChange as $absolutPositionKey => $absolutPositionValue)
            ↪ {
828       foreach ($cumulativeSectionLengthStart as $sectionStartKey => $sectionStartValue)
              ↪ {
829         if ($absolutPositionValue >= $sectionStartValue && $absolutPositionValue <
                ↪ $cumulativeSectionLengthEnd[$sectionStartKey]) {
830           if ($sectionStartKey == $indexCurrentSection && $sectionStartKey ==
                  ↪ $indexTargetSection) {
831             $returnTrainSection[$absolutPositionKey] = $next_sections[$sectionStartKey];
832           } else if ($sectionStartKey == $indexCurrentSection) {
833             $returnTrainSection[$absolutPositionKey] = $next_sections[$sectionStartKey];
834           } else if ($sectionStartKey == $indexTargetSection) {
835             $returnTrainSection[$absolutPositionKey] = $next_sections[$sectionStartKey];
836           } else {
837             $returnTrainSection[$absolutPositionKey] = $next_sections[$sectionStartKey];
838           }
839           break;
840         } else if ($absolutPositionKey == array_key_last($returnTrainPositionChange) &&
                ↪ abs($absolutPositionValue - floatval($cumulativeSectionLengthEnd[
                ↪ $sectionStartKey])) < $globalFloatingPointNumbersRoundingError) {
841           $returnTrainSection[$absolutPositionKey] = $next_sections[$sectionStartKey];
842           break;
843         } else {
844           debugMessage("Eine absolute Position konnte keine relativen Position in einem
                  ↪ Abschnitt zugeordnet werden!");
845         }
846       }
847     }
848     return array($returnTrainPositionChange, $returnTrainSpeedChange,
            ↪ $returnTrainTimeChange, $returnTrainRelativePosition, $returnTrainSection,
            ↪ $returnIsSpeedChange);
849   }
850 }
851
852 // Überprüft immer zwei benachbarte KeyPoints (0+1, 2+3, 4+5, 6+7 etc.)
853 // TODO: schöner schreiben die Funktion... sieht hässlich aus!
854 // TODO: Sollte auch bei einer ungeraden Anzahl an KeyPoints funktionieren, bitte aber
        ↪ nochmal überprüfen
855 function recalculateKeyPoints(array $tempKeyPoints) {
856   $returnKeyPoints = array();
857   $numberOfPairs = sizeof($tempKeyPoints) / 2;
858   for($j = 0; $j < $numberOfPairs; $j++) {
859     $i = $j * 2;
860     $return = checkBetweenTwoKeyPoints($tempKeyPoints, $i);
861     foreach ($return as $keyPoint) {
862       array_push($returnKeyPoints, $keyPoint);
```

```
    }
  }
  return $returnKeyPoints;
}

function checkBetweenTwoKeyPoints(array $temKeyPoints, int $keyPointIndex) {

  global $trainPositionChange;
  global $trainSpeedChange;
  global $cumulativeSectionLengthStartMod;
  global $cumulativeSectionLengthEndMod;
  global $next_v_max_mod;
  global $verzoegerung;
  global $indexTargetSectionMod;

  $failedSections = array();
  $groupedFailedSections = array();
  $returnKeyPoints = array();
  $failedPositions = array();
  $failedSpeeds = array();

  foreach ($trainPositionChange as $trainPositionChangeKey => $trainPositionChangeValue)
      ↪ {
    if ($trainPositionChangeValue >= $temKeyPoints[$keyPointIndex]["position_0"] &&
        ↪ $trainPositionChangeValue <= $temKeyPoints[$keyPointIndex + 1]["position_1"])
        ↪ {
      foreach ($cumulativeSectionLengthStartMod as $cumulativeSectionLengthStartKey =>
          ↪ $cumulativeSectionLengthStartValue) {
        if ($trainPositionChangeValue < $cumulativeSectionLengthStartValue) {
          if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
              ↪ $cumulativeSectionLengthStartKey - 1]) {
            array_push($failedSections, ($cumulativeSectionLengthStartKey - 1));
            array_push($failedSpeeds, $trainSpeedChange[$trainPositionChangeKey]);
            $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                ↪ $trainPositionChangeKey];
          }
          break;
        } else if ($cumulativeSectionLengthStartKey == $indexTargetSectionMod) {
          if ($trainPositionChangeValue > $cumulativeSectionLengthStartValue) {
            if ($trainSpeedChange[$trainPositionChangeKey] > $next_v_max_mod[
                ↪ $cumulativeSectionLengthStartKey]) {
              array_push($failedSections, $cumulativeSectionLengthStartKey);
              array_push($failedSpeeds, $trainSpeedChange[$trainPositionChangeKey]);
              $failedPositions[$trainPositionChangeKey] = $trainPositionChange[
                  ↪ $trainPositionChangeKey];
            }
            break;
          }
        }
      }
    }
  }

```

```
908   // Alle Sections zwischen denn beiden KeyPoints, bei denen die v_max überschritten
          ↪ wird
909   $failedSections = array_unique($failedSections);
910
911   // Info: Der Index der failedPositions entspricht dem Index der trainChanges
912
913   // Wenn es kein Fehler gibt, werden die beiden KeyPoints zurückgegeben und wen es
          ↪ einen Fehler gibt, wird der
914   // erste der beiden KeyPoints im $returnKeyPoints gespeichert
915   if (sizeof($failedSections) == 0) {
916     return array($temKeyPoints[$keyPointIndex], $temKeyPoints[$keyPointIndex + 1]);
917   } else {
918     $returnKeyPoints[0]["speed_0"] = $temKeyPoints[$keyPointIndex]["speed_0"];
919     $returnKeyPoints[0]["position_0"] = $temKeyPoints[$keyPointIndex]["position_0"];
920   }
921
922   // Einteilung der benachbarten failedSections in zusammenhängende Gruppen
923   $previous = NULL;
924   $index = 0;
925   foreach($failedSections as $key => $value) {
926     if($value > $previous + 1) {
927       $index++;
928     }
929     $groupedFailedSections[$index][] = $value;
930     $previous = $value;
931   }
932
933   // Durch alle Gruppen gehen
934   foreach ($groupedFailedSections as $groupSectionsIndex => $groupSectionsValue) {
935     $firstFailedPositionIndex = null;
936     $lastFailedPositionIndex = null;
937     $firstFailedPosition = null;
938     $lastFailedPosition = null;
939     $lastElement = array_key_last($returnKeyPoints);
940     $failedSection = null;
941
942     // Ermittlung der Section mit der kleinsten v_max von allen failed Sections in der
          ↪ Gruppe
943     if (sizeof($groupSectionsValue) == 1) {
944       $failedSection = $groupSectionsValue[0];
945     } else {
946       $slowestSpeed = 200;
947       for ($i = 0; $i <= (sizeof($groupSectionsValue) - 1); $i++) {
948         if ($next_v_max_mod[$groupSectionsValue[$i]] < $slowestSpeed) {
949           $slowestSpeed = $next_v_max_mod[$groupSectionsValue[$i]];
950           $failedSection = $groupSectionsValue[$i];
951         }
952       }
953     }
954
955     // Start- und Endposition der $failedSection
956     $failedSectionStart = $cumulativeSectionLengthStartMod[$failedSection];
957     $failedSectionEnd = $cumulativeSectionLengthEndMod[$failedSection];
```

```
958     //$vMaxInFailedSection = $next_v_max[$failedSection];
959
960     // Bestimmung der ersten und letzten Position, in der es in der failed Section
961     // zu einer Geschwindigkeitsüberschreitung kommt
962     foreach ($failedPositions as $failPositionIndex => $failPositionValue) {
963       if ($failPositionValue > $failedSectionStart && $failPositionValue <
            ↪ $failedSectionEnd) {
964         if ($firstFailedPositionIndex == null) {
965           $firstFailedPositionIndex = $failPositionIndex;
966         }
967         $lastFailedPositionIndex = $failPositionIndex;
968       }
969     }
970
971     // Bestimmung des letzten Punktes, bei dem die Geschwindigkeit noch nicht zu schnell
           ↪  war
972     // Wenn der Punkt davor außerhalb der failedSection liegt => Startpunkt = Anfang der
           ↪  Section
973     // Wenn der Punkt davor innnerhalb der failed Section liegt => Startpunkt = der
           ↪ Punkt davor
974     if ($firstFailedPositionIndex != 0) {
975       if ($trainPositionChange[$firstFailedPositionIndex - 1] < $failedSectionStart) {
976         $firstFailedPosition = $failedSectionStart;
977       } else {
978         $firstFailedPosition = $trainPositionChange[$firstFailedPositionIndex - 1];
979       }
980     } else {
981       $firstFailedPosition = $failedSectionStart;
982     }
983
984     // Bestimmung des ersten Punktes, bei dem die Geschwindigkeit nicht mehr zu schnell
           ↪ war
985     // Beschreibung: siehe $firstFailedPosition Berechnung
986     if ($lastFailedPositionIndex != array_key_last($trainPositionChange)) {
987       if ($trainPositionChange[$lastFailedPositionIndex + 1] > $failedSectionEnd) {
988         $lastFailedPosition = $failedSectionEnd;
989       } else {
990         $lastFailedPosition = $trainPositionChange[$lastFailedPositionIndex + 1];
991       }
992     } else {
993       $lastFailedPosition = $failedSectionEnd;
994     }
995     $returnKeyPoints[$lastElement + 1]["position_1"] = $firstFailedPosition;
996     $returnKeyPoints[$lastElement + 1]["speed_1"] = $next_v_max_mod[$failedSection];
997     $returnKeyPoints[$lastElement + 2]["position_0"] = $lastFailedPosition;
998     $returnKeyPoints[$lastElement + 2]["speed_0"] = $next_v_max_mod[$failedSection];
999   }
1000
1001  // Hinzufügen von dem "Ende"
1002  $returnKeyPoints[array_key_last($returnKeyPoints) + 1]["position_1"] = $temKeyPoints[
          ↪ $keyPointIndex + 1]["position_1"];
1003  $returnKeyPoints[array_key_last($returnKeyPoints)]["speed_1"] = $temKeyPoints[
          ↪ $keyPointIndex + 1]["speed_1"];
```

```
1004    $numberOfPairs = sizeof($returnKeyPoints) / 2;
1005    for($j = 0; $j < $numberOfPairs; $j++) {
1006      $i = $j * 2;
1007      $distance = $returnKeyPoints[$i + 1]["position_1"] - $returnKeyPoints[$i]["
              ↪ position_0"];
1008      $vMax = getVMaxBetweenTwoPoints($distance, $returnKeyPoints[$i]["speed_0"],
              ↪ $returnKeyPoints[$i + 1]["speed_1"]);
1009      // TODO: Der Teil kann weg, getVMaxBetweenTwoPoints() kann nicht -10 zurückgeben
1010      if ($vMax == -10) {
1011        $returnKeyPoints[$i]["position_0"] = $returnKeyPoints[$i + 1]["position_1"] - (
              ↪ getBrakeDistance($returnKeyPoints[$i]["speed_0"], $returnKeyPoints[$i + 1][
              ↪ "speed_1"], $verzoegerung));
1012        $distance = $returnKeyPoints[$i + 1]["position_1"] - $returnKeyPoints[$i]["
              ↪ position_0"];
1013        $vMax = getVMaxBetweenTwoPoints($distance, $returnKeyPoints[$i]["speed_0"],
              ↪ $returnKeyPoints[$i + 1]["speed_1"]);
1014      }
1015      $returnKeyPoints[$i]["speed_1"] = $vMax; //TODO
1016      $returnKeyPoints[$i]["position_1"] = $returnKeyPoints[$i]["position_0"] +
              ↪ getBrakeDistance($returnKeyPoints[$i]["speed_0"], $vMax, $verzoegerung);
1017      $returnKeyPoints[$i + 1]["speed_0"] = $vMax;
1018      $returnKeyPoints[$i + 1]["position_0"] = $returnKeyPoints[$i + 1]["position_1"] -
              ↪ getBrakeDistance($vMax, $returnKeyPoints[$i + 1]["speed_1"], $verzoegerung);
1019    }
1020    return $returnKeyPoints;
1021  }
1022
1023  // Wenn ein Key Point beschleunigt und der nächste Key Point abbremst, wird
1024  // die Geschwindigkeit zwischen den beiden KeyPoints als $v_maxBetweenKeyPoints
1025  // gespeichert und als $v_minBetweenKeyPoints der größere Wert von
1026  // $keyPoints[$i]["speed_0"] und $keyPoints[$i + 1]["speed_1"]
1027  function checkIfTheSpeedCanBeDecreased() {
1028    global $keyPoints;
1029    global $returnPossibleSpeed;
1030    $returnPossibleSpeed = array();
1031    for ($i = 0; $i < (sizeof($keyPoints) - 1); $i++) {
1032      $v_maxBetweenKeyPoints = $keyPoints[$i]["speed_1"];
1033      $v_minBetweenKeyPoints = null;
1034      if ($keyPoints[$i]["speed_0"] < $v_maxBetweenKeyPoints && $keyPoints[$i + 1]["
              ↪ speed_1"] < $v_maxBetweenKeyPoints) {
1035        $v_minBetweenKeyPoints = $keyPoints[$i]["speed_0"];
1036        if ($keyPoints[$i + 1]["speed_1"] > $v_minBetweenKeyPoints) {
1037          $v_minBetweenKeyPoints = $keyPoints[$i + 1]["speed_1"];
1038        }
1039      }
1040      if (isset($v_minBetweenKeyPoints)) {
1041        if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints >= 10) {
1042          $v_minBetweenKeyPoints = 10;
1043        } else if ($v_minBetweenKeyPoints == 0 && $v_maxBetweenKeyPoints == 10) {
1044          $v_minBetweenKeyPoints = null;
1045        }
1046      }
1047      if ($v_minBetweenKeyPoints != null) {
```

```
1048        if ($v_minBetweenKeyPoints % 10 != 0) {
1049          $rest = $v_minBetweenKeyPoints % 10;
1050          $v_minBetweenKeyPoints = $v_minBetweenKeyPoints - $rest + 10;
1051        }
1052        array_push($returnPossibleSpeed, array("KeyPoint_index" => $i, "values" => range(
                ↪ $v_minBetweenKeyPoints, $v_maxBetweenKeyPoints, 10)));
1053      }
1054    }
1055    if (sizeof($returnPossibleSpeed) > 0) {
1056      return array("possible" => true, "range" => $returnPossibleSpeed);
1057    } else {
1058      return array("possible" => false, "range" => array());
1059    }
1060  }
1061
1062  function speedFineTuning(float $timeDiff, int $index) {
1063    global $keyPoints;
1064    global $verzoegerung;
1065    global $globalTimeOnOneSpeed;
1066    global $useMinTimeOnSpeed;
1067    $speed_0 = $keyPoints[$index]["speed_1"];
1068    $speed_1 = null;
1069    $availableDistance = $keyPoints[$index + 1]["position_0"] - $keyPoints[$index]["
          ↪ position_1"];
1070    $timeBetweenKeyPoints = $keyPoints[$index + 1]["time_0"] - $keyPoints[$index]["time_1"
          ↪ ];
1071    $availableTime = $timeBetweenKeyPoints + $timeDiff;
1072    if ($keyPoints[$index + 1]["speed_1"] != 0) {
1073      $speed_1 = $keyPoints[$index + 1]["speed_1"];
1074      $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index + 1]["
            ↪ speed_0"], $keyPoints[$index + 1]["speed_1"], $availableDistance,
            ↪ $availableTime);
1075      if ($useMinTimeOnSpeed) {
1076        if (distanceWithSpeedToTime($speed_0, $availableDistance - $lengthDifference) >
              ↪ $globalTimeOnOneSpeed && distanceWithSpeedToTime($speed_1,
              ↪ $lengthDifference) > $globalTimeOnOneSpeed) {
1077          $keyPoints[$index + 1]["position_0"] = $keyPoints[$index + 1]["position_0"] -
                ↪ $lengthDifference;
1078          $keyPoints[$index + 1]["position_1"] = $keyPoints[$index + 1]["position_1"] -
                ↪ $lengthDifference;
1079        }
1080      } else {
1081        $keyPoints[$index + 1]["position_0"] = $keyPoints[$index + 1]["position_0"] -
              ↪ $lengthDifference;
1082        $keyPoints[$index + 1]["position_1"] = $keyPoints[$index + 1]["position_1"] -
              ↪ $lengthDifference;
1083      }
1084    } else if ($keyPoints[$index + 1]["speed_0"] > 10) {
1085      $speed_1 = 10;
1086      $lengthDifference = calculateDistanceforSpeedFineTuning($keyPoints[$index + 1]["
            ↪ speed_0"],10, $availableDistance, $availableTime);
1087      if ($useMinTimeOnSpeed) {
```

```
1088        if (distanceWithSpeedToTime($speed_0, $availableDistance - $lengthDifference) >
                ↪ $globalTimeOnOneSpeed && distanceWithSpeedToTime($speed_1,
                ↪ $lengthDifference) > $globalTimeOnOneSpeed) {
1089          $firstKeyPoint = createKeyPoint(($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);
1090          $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_1"] -
                  ↪ getBrakeDistance(10, 0, $verzoegerung)),$keyPoints[$index + 1]["
                  ↪ position_1"],10,$keyPoints[$index + 1]["speed_1"]);
1091          $keyPoints[$index + 1] = $secondKeyPoint;
1092          array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1093        }
1094      } else {
1095        $firstKeyPoint = createKeyPoint(($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);
1096        $secondKeyPoint = createKeyPoint(($keyPoints[$index + 1]["position_1"] -
              ↪ getBrakeDistance(10, 0, $verzoegerung)),$keyPoints[$index + 1]["position_1"
              ↪ ],10,$keyPoints[$index + 1]["speed_1"]);
1097        $keyPoints[$index + 1] = $secondKeyPoint;
1098        array_splice( $keyPoints, ($index + 1), 0, array($firstKeyPoint));
1099      }
1100    }
1101  }
1102
1103  function calculateDistanceforSpeedFineTuning(int $v_0, int $v_1, float $distance, float
        ↪ $time) : float {
1104    return $distance - (($distance - $time * $v_1 / 3.6)/($v_0 / 3.6 - $v_1 / 3.6)) * (
          ↪ $v_0 / 3.6);
1105  }
1106
1107  // Sucht den KeyPoint der zu maximalen Geschwindigkeit beschleunigt
1108  // Wenn die maximale Geschwindigkeit mehrfach erreciht wird, wird
1109  // der letzte dieser KeyPoints genommen
1110  //
1111  // Zu dem Index wird auch noch die Speed Range abgespeichert wie bei
1112  // checkIfTheSpeedCanBeDecreased()
1113  // TODO: Kann man diese beiden Funktionen kombinieren?
1114  function findMaxSpeed(array $speedDecrease) {
1115    $maxSpeed = 0;
1116    $minSpeed = 0;
1117    $keyPointIndex = null;
1118
1119    for ($i = 0; $i < sizeof($speedDecrease["range"]); $i++) {
1120      if (max($speedDecrease["range"][$i]["values"]) >= $maxSpeed) {
1121        $maxSpeed = max($speedDecrease["range"][$i]["values"]);
1122        $minSpeed = min($speedDecrease["range"][$i]["values"]);
1123        $keyPointIndex = $speedDecrease["range"][$i]["KeyPoint_index"];
1124      }
1125    }
```

```
1126   return array("min_speed" => $minSpeed, "max_speed" => $maxSpeed, "
           ↪ first_key_point_index" => $keyPointIndex);
1127 }
1128
1129 // Beim Start der Berechnung
1130 function checkIfItsPossible() {
1131   global $currentSpeed;
1132   global $distanceToNextStop;
1133   global $verzoegerung;
1134   global $globalTimeOnOneSpeed;
1135   global $errorMinTimeOnSpeed;
1136
1137   $minTimeIsPossible = true;
1138   if ($currentSpeed == 0) {
1139     $distance_0 = getBrakeDistance(0, 10, $verzoegerung);
1140     $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1141     $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 - $distance_1)
           ↪ ;
1142     if ($time < $globalTimeOnOneSpeed) {
1143       $minTimeIsPossible = false;
1144       if ($errorMinTimeOnSpeed) {
1145         // TODO: Notbremsung einleiten/Zug bekommt Fehler
1146       }
1147       echo "Der Zug schafft es ohne Notbremsung am Ziel anzukommen, kann aber nicht die
           ↪ mind. Zeit einhalten.\n";
1148     }
1149     // Sollte egal sein, da der Zug schon vor der Berechnung auf v_0 war und somit die
           ↪ Zeit unbekannt ist.
1150   } else {
1151     if (getBrakeDistance($currentSpeed, 0, $verzoegerung) != $distanceToNextStop) {
1152       $distance_0 = getBrakeDistance($currentSpeed, 10, $verzoegerung);
1153       $distance_1 = getBrakeDistance(10, 0, $verzoegerung);
1154       $time = distanceWithSpeedToTime(10, $distanceToNextStop - $distance_0 -
           ↪ $distance_1);
1155       if ($time < $globalTimeOnOneSpeed) {
1156         $minTimeIsPossible = false;
1157         if ($errorMinTimeOnSpeed) {
1158           // TODO: Notbremsung einleiten/Zug bekommt Fehler
1159         }
1160         echo "Der Zug schafft es, ohne Notbremsung am Ziel anzukommen.\n";
1161       }
1162     }
1163   }
1164   return $minTimeIsPossible;
1165 }
1166
1167 // TODO: General Options:
1168 // 1. Soll generell überprüft werden, ob der Zug die mind. Zeit auf einer
       ↪ Geschwindigkeit einhält
1169 // 2. Soll es zu einem Fehler kommen, wenn der Zug diese Bedingung nicht erfüllen kann?
1170 // => Der Zug könnte ein Error-Statement bekommen
1171 function toShortOnOneSpeed () {
1172
```

```
1173    global $keyPoints;
1174    global $verzoegerung;
1175
1176    $index = 0;
1177    $localKeyPoints = $keyPoints;
1178    $subsections = createSubsections($localKeyPoints);
1179    $breakesOnly = false;
1180
1181    // Sobald in einer Section die Geschwindigkeit verändert werden müsste, wird erstmal
          ↪ die Geschwindigkeit angepasst und dann neu berechnet...
1182    while (toShortInSubsection($subsections)) {
1183      $breakesOnly = true;
1184      foreach ($subsections as $sectionKey => $sectionValue) {
1185        if ($sectionValue["failed"]) {
1186          if (!$sectionValue["brakes_only"]) {
1187            $breakesOnly = false;
1188          }
1189          $return = postponeSubsection($localKeyPoints, $sectionValue);
1190          if (!$return["fail"]) {
1191            $localKeyPoints = $return["keyPoints"];
1192          } else {
1193            if (!$sectionValue["brakes_only"]) {
1194              $localKeyPoints[$sectionValue["max_index"]]["speed_1"] -= 10;
1195              $localKeyPoints[$sectionValue["max_index"] + 1]["speed_0"] -= 10;
1196              $localKeyPoints[$sectionValue["max_index"]]["position_1"] = $localKeyPoints[
                      ↪ $sectionValue["max_index"]]["position_0"] + getBrakeDistance(
                      ↪ $localKeyPoints[$sectionValue["max_index"]]["speed_0"],
                      ↪ $localKeyPoints[$sectionValue["max_index"]]["speed_1"], $verzoegerung)
                      ↪ ;
1197              $localKeyPoints[$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"],
                      ↪  $verzoegerung);
1198              $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1199              $localKeyPoints = deleteDoubledKeyPoints($localKeyPoints);
1200              break;
1201            }
1202          }
1203        }
1204      }
1205      $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1206      $localKeyPoints = array_values($localKeyPoints);
1207      $subsections = createSubsections($localKeyPoints);
1208      if ($breakesOnly) {
1209        break;
1210      }
1211    }
1212    $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints);
1213    $keyPoints = $localKeyPoints;
1214  }
1215
1216
```

```
1217 function postponeSubsectionOld (array $localKeyPoints, array $subsection) {
1218
1219   //global $keyPoints;
1220   global $globalTimeOnOneSpeed;
1221
1222   $keyPoints = $localKeyPoints;
1223
1224   $indexMaxSection = array_search($subsection["max_index"], $subsection["indexes"]);
1225   $indexLastKeyPoint = array_key_last($subsection["indexes"]);
1226
1227   if ($subsection["is_prev_section"]) {
1228     $timeDiff = $keyPoints[$subsection["indexes"][0]]["time_0"] - $keyPoints[$subsection
          ↪ ["indexes"][0] - 1]["time_1"] - $globalTimeOnOneSpeed;
1229     if ($timeDiff < 0) {
1230       $positionDiff = abs($timeDiff) * $keyPoints[$subsection["indexes"][0]]["speed_0"]
            ↪ / 3.6;
1231       //$keyPoints[$subsection["indexes"][0]]["time_0"] -= $timeDiff;
1232       //$keyPoints[$subsection["indexes"][0]]["time_1"] -= $timeDiff;
1233       $keyPoints[$subsection["indexes"][0]]["position_0"] += $positionDiff;
1234       $keyPoints[$subsection["indexes"][0]]["position_1"] += $positionDiff;
1235       $keyPoints = calculateTimeFromKeyPoints();
1236
1237     }
1238   }
1239
1240   for ($i = 1; $i <= $indexMaxSection; $i++) {
1241     $timeDiff = $keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[
          ↪ $subsection["indexes"][$i] - 1]["time_1"] - $globalTimeOnOneSpeed;
1242     if ($timeDiff < 0) {
1243       $positionDiff = abs($timeDiff) * $keyPoints[$subsection["indexes"][$i]]["speed_0"]
            ↪  / 3.6;
1244       //$keyPoints[$subsection["indexes"][$i]]["time_0"] -= $timeDiff;
1245       //$keyPoints[$subsection["indexes"][$i]]["time_1"] -= $timeDiff;
1246       $keyPoints[$subsection["indexes"][$i]]["position_0"] += $positionDiff;
1247       $keyPoints[$subsection["indexes"][$i]]["position_1"] += $positionDiff;
1248       $keyPoints = calculateTimeFromKeyPoints();
1249     }
1250   }
1251
1252   if ($subsection["is_next_section"]) {
1253     $timeDiff = $keyPoints[$indexLastKeyPoint + 1]["time_0"] - $keyPoints[
          ↪ $indexLastKeyPoint]["time_1"] - $globalTimeOnOneSpeed;
1254     if ($timeDiff < 0) {
1255       $positionDiff = abs($timeDiff) * $keyPoints[$indexLastKeyPoint]["speed_1"] / 3.6;
1256       //$keyPoints[$indexLastKeyPoint]["time_0"] += $timeDiff;
1257       //$keyPoints[$indexLastKeyPoint]["time_1"] += $timeDiff;
1258       $keyPoints[$indexLastKeyPoint]["position_0"] -= $positionDiff;
1259       $keyPoints[$indexLastKeyPoint]["position_1"] -= $positionDiff;
1260       $keyPoints = calculateTimeFromKeyPoints();
1261     }
1262   }
1263
1264   for ($i = $indexLastKeyPoint - 1; $i > $indexMaxSection; $i--) {
```

```
1265      $timeDiff = $keyPoints[$subsection["indexes"][$i + 1]]["time_0"] - $keyPoints[
              ↪ $subsection["indexes"][$i]]["time_1"] - $globalTimeOnOneSpeed;
1266      if ($timeDiff < 0) {
1267        $positionDiff = abs($timeDiff) * $keyPoints[$indexLastKeyPoint]["speed_1"] / 3.6;
1268        //$keyPoints[$subsection["indexes"][$i]]["time_0"] += $timeDiff;
1269        //$keyPoints[$subsection["indexes"][$i]]["time_1"] += $timeDiff;
1270        $keyPoints[$subsection["indexes"][$i]]["position_0"] -= $positionDiff;
1271        $keyPoints[$subsection["indexes"][$i]]["position_1"] -= $positionDiff;
1272        $keyPoints = calculateTimeFromKeyPoints();
1273      }
1274    }
1275
1276
1277    return $keyPoints;
1278  }
1279
1280  function postponeSubsection (array $localKeyPoints, array $subsection) {
1281
1282    global $globalTimeOnOneSpeed;
1283    global $verzoegerung;
1284
1285    $deletedKeyPoints = array();
1286    $numberOfKeyPoints = sizeof($subsection["indexes"]);
1287    $indexMaxSection = array_search($subsection["max_index"], $subsection["indexes"]);
1288    $indexLastKeyPoint = array_key_last($subsection["indexes"]);
1289
1290    if ($subsection["is_prev_section"]) {
1291      $timeDiff = $localKeyPoints[$subsection["indexes"][0]]["time_0"] - $localKeyPoints[
              ↪ $subsection["indexes"][0] - 1]["time_1"] - $globalTimeOnOneSpeed;
1292      if ($timeDiff < 0) {
1293
1294        $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection["indexes"][0]]["
              ↪ speed_0"] / 3.6;
1295        if (!($localKeyPoints[$subsection["indexes"][0]]["position_1"] + $positionDiff >
              ↪ $localKeyPoints[$subsection["indexes"][$indexMaxSection + 1]]["position_0"
              ↪ ])) {
1296          $localKeyPoints[$subsection["indexes"][0]]["position_0"] += $positionDiff;
1297          $localKeyPoints[$subsection["indexes"][0]]["position_1"] += $positionDiff;
1298          // Es muss einen nächsten KeyPoint geben, da der Zug hier beschleunigt und er
                ↪ Ende der Strecke gilt v = 0
1299          if ($localKeyPoints[$subsection["indexes"][0]]["position_1"] > $localKeyPoints[
                ↪ $subsection["indexes"][0] + 1]["position_0"]) {
1300            array_push($deletedKeyPoints, $subsection["indexes"][0] + 1);
1301            $numberOfKeyPoints -= 1;
1302            $v_0 = $localKeyPoints[$subsection["indexes"][0]]["speed_0"];
1303            $v_1 = $localKeyPoints[$subsection["indexes"][0] + 1]["speed_1"];
1304            $localKeyPoints[$subsection["indexes"][0]]["position_1"] = $localKeyPoints[
                  ↪ $subsection["indexes"][0]]["position_0"] + getBrakeDistance($v_0, $v_1,
                  ↪ $verzoegerung);
1305            $localKeyPoints[$subsection["indexes"][0]]["speed_1"] = $v_1;
1306          }
1307          $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints, $deletedKeyPoints);
1308        }
```

```
    }
  }

  for ($i = 1; $i <= $indexMaxSection; $i++) {
    if (!in_array($subsection["indexes"][$i], $deletedKeyPoints)) {
      $timeDiff = $localKeyPoints[$subsection["indexes"][$i]]["time_0"] -
          ↪ $localKeyPoints[$subsection["indexes"][$i] - 1]["time_1"] -
          ↪ $globalTimeOnOneSpeed;
      if ($timeDiff < 0) {
        $positionDiff = abs($timeDiff) * $localKeyPoints[$subsection["indexes"][$i]]["
            ↪ speed_0"] / 3.6;
        if (!($localKeyPoints[$subsection["indexes"][$i]]["position_1"] + $positionDiff >
            ↪  $localKeyPoints[$subsection["indexes"][$indexMaxSection + 1]]["
            ↪ position_0"])) {
          $localKeyPoints[$subsection["indexes"][$i]]["position_0"] += $positionDiff;
          $localKeyPoints[$subsection["indexes"][$i]]["position_1"] += $positionDiff;
          if ($i < $indexMaxSection && $localKeyPoints[$subsection["indexes"][$i]]["
              ↪ position_1"] > $localKeyPoints[$subsection["indexes"][$i] + 1]["
              ↪ position_0"]) {
            array_push($deletedKeyPoints, ($subsection["indexes"][$i] + 1));
            $numberOfKeyPoints -= 1;
            $v_0 = $localKeyPoints[$subsection["indexes"][$i]]["speed_0"];
            $v_1 = $localKeyPoints[$subsection["indexes"][$i] + 1]["speed_1"];
            $localKeyPoints[$subsection["indexes"][$i]]["position_1"] = $localKeyPoints[
                ↪ $subsection["indexes"][$i]]["position_0"] + getBrakeDistance($v_0,
                ↪ $v_1, $verzoegerung);
            $localKeyPoints[$subsection["indexes"][$i]]["speed_1"] = $v_1;
          }
          $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints, $deletedKeyPoints
              ↪ );
        }
      }
    }
  }
  if ($subsection["is_next_section"]) {
    $timeDiff = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint] + 1]["time_0"
        ↪ ] - $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["time_1"] -
        ↪ $globalTimeOnOneSpeed;
    if ($timeDiff < 0) {
      $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]["speed_1"] /
          ↪ 3.6;
      if (!($localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_0"] -
          ↪ $positionDiff < $localKeyPoints[$subsection["indexes"][$indexMaxSection]]["
          ↪ position_0"])) {
        $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_0"] -=
            ↪ $positionDiff;
        $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_1"] -=
            ↪ $positionDiff;
        if ($localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_0"] <
            ↪ $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint] - 1]["
            ↪ position_1"]) {
          array_push($deletedKeyPoints, ($subsection["indexes"][$indexLastKeyPoint] - 1))
              ↪ ;
```

```
1342          $numberOfKeyPoints -= 1;
1343          $v_0 = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint] - 1]["speed_0
                  ↪ "];
1344          $v_1 = $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["speed_1"];
1345          $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_0"] =
                  ↪ $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["position_1"
                  ↪ ] - getBrakeDistance($v_0, $v_1, $verzoegerung);
1346          $localKeyPoints[$subsection["indexes"][$indexLastKeyPoint]]["speed_0"] = $v_0;
1347        }
1348        $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints, $deletedKeyPoints);
1349      }
1350    }
1351  }
1352
1353  for ($i = $indexLastKeyPoint - 1; $i > $indexMaxSection; $i--) {
1354    if (!in_array($i, $deletedKeyPoints)) {
1355      $timeDiff = $localKeyPoints[$subsection["indexes"][$i + 1]]["time_0"] -
              ↪ $localKeyPoints[$subsection["indexes"][$i]]["time_1"] -
              ↪ $globalTimeOnOneSpeed;
1356      if ($timeDiff < 0) {
1357        $positionDiff = abs($timeDiff) * $localKeyPoints[$indexLastKeyPoint]["speed_1"] /
                  ↪  3.6;
1358        if (!($localKeyPoints[$subsection["indexes"][$i]]["position_0"] - $positionDiff <
                  ↪  $localKeyPoints[$subsection["indexes"][$indexMaxSection]]["position_0"])
                  ↪ ) {
1359          $localKeyPoints[$subsection["indexes"][$i]]["position_0"] -= $positionDiff;
1360          $localKeyPoints[$subsection["indexes"][$i]]["position_1"] -= $positionDiff;
1361          if ($i > ($indexMaxSection + 1) && $localKeyPoints[$subsection["indexes"][$i]][
                  ↪ "position_0"] < $localKeyPoints[$subsection["indexes"][$i] - 1]["
                  ↪ position_1"]) {
1362            array_push($deletedKeyPoints, ($subsection["indexes"][$i] - 1));
1363            $numberOfKeyPoints -= 1;
1364            $v_0 = $localKeyPoints[$subsection["indexes"][$i] - 1]["speed_0"];
1365            $v_1 = $localKeyPoints[$subsection["indexes"][$i]]["speed_1"];
1366            $localKeyPoints[$subsection["indexes"][$i]]["position_0"] = $localKeyPoints[
                      ↪ $subsection["indexes"][$i]]["position_1"] - getBrakeDistance($v_0,
                      ↪ $v_1, $verzoegerung);
1367            $localKeyPoints[$subsection["indexes"][$i]]["speed_0"] = $v_0;
1368          }
1369          $localKeyPoints = calculateTimeFromKeyPoints($localKeyPoints, $deletedKeyPoints
                  ↪ );
1370        }
1371      }
1372    }
1373  }
1374  // Info: Erster und letzter KeyPoint aus $subsection["indexes"] sind auf jedenfall
          ↪ noch vorhanden
1375  $keys = $subsection["indexes"];
1376  foreach ($deletedKeyPoints as $index) {
1377    unset($keys[array_search($index, $keys)]);
1378  }
1379  $keys = array_values($keys);
1380  $failed = false;
```

```
1381    if ($subsection["is_prev_section"]) {
1382      // Geht, weil die subsections von hinten nach vorne kontrolliert werden.
1383      // Und weil Start- und End-KeyPoint immmer gleich bleiben.
1384      if ($localKeyPoints[$keys[0]]["time_0"] - $localKeyPoints[$keys[0] - 1]["time_1"] <
              ↪ $globalTimeOnOneSpeed) {
1385        $failed = true;
1386      }
1387    }
1388    if ($subsection["is_next_section"]) {
1389      // Geht, weil die subsections von hinten nach vorne kontrolliert werden.
1390      // Und weil Start- und End-KeyPoint immmer gleich bleiben.
1391      if ($localKeyPoints[end($keys) + 1]["time_0"] - $localKeyPoints[end($keys)]["time_1"
              ↪ ] < $globalTimeOnOneSpeed) {
1392        $failed = true;
1393      }
1394    }
1395    for ($i = 1; $i < sizeof($keys); $i++) {
1396      if ($localKeyPoints[$keys[$i]]["time_0"] - $localKeyPoints[$keys[$i - 1]]["time_1"]
              ↪ < $globalTimeOnOneSpeed) {
1397        $failed = true;
1398        break;
1399      }
1400    }
1401    if ($failed) {
1402      return array("fail" => true, "keyPoints" => array());
1403    } else {
1404      foreach ($deletedKeyPoints as $index) {
1405        unset($localKeyPoints[$index]);
1406      }
1407      return array("fail" => false, "keyPoints" => $localKeyPoints);
1408    }
1409  }
1410
1411  // Es werden nur "komplette" Subsectionsbetrachtet. Der Zug MUSS beschleunigen und
       ↪ abbremsen!
1412  function createSubsections (array $localKeyPoints) {
1413    global $globalTimeOnOneSpeed;
1414
1415    $keyPoints = $localKeyPoints;
1416    $subsections = array();
1417    $subsection = array("max_index" => null, "indexes" => array(), "is_prev_section" =>
           ↪ false, "is_next_section" => false);
1418    $maxIndex = null;
1419
1420    // Wenn die erste Geschwindigkeit die maximale Geschwindigkeit der ersten Subsection
           ↪ ist.
1421    // TODO: Bei v_0 != 0 hat der erste KeyPoint v_0 == v_1... Kommt es da zu Konflikten?
1422    for($i = 0; $i < sizeof($keyPoints); $i++) {
1423      // subsection zu ende
1424      if ($i > 0) {
1425        if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"] && $keyPoints[$i - 1]["
             ↪ speed_0"] > $keyPoints[$i - 1]["speed_1"] || $i == sizeof($keyPoints) - 1)
             ↪ {
```

```
      if ($i == sizeof($keyPoints) - 1) {
        array_push($subsection["indexes"], $i);
      }
      array_push($subsections, $subsection);
      $subsection["indexes"] = array();
    }
  }
  if ($keyPoints[$i]["speed_0"] < $keyPoints[$i]["speed_1"]) {
    $subsection["max_index"] = $i;
  }
  array_push($subsection["indexes"], $i);
}

// Check if middle section failed
for ($i = 1; $i < sizeof($subsections); $i++) {
  //$firstIndex = $subsections[$i]["max_index"] + 1;
  $firstIndex = $subsections[$i]["indexes"][array_key_first($subsections[$i]["indexes"]
      ↪ ])];
  if ($keyPoints[$firstIndex]["time_0"] - $keyPoints[$firstIndex - 1]["time_1"] <
      ↪ $globalTimeOnOneSpeed) {
    $subsections[$i]["is_prev_section"] = true;
    $subsections[$i]["failed"] = true;
  } else {
    $subsections[$i]["is_prev_section"] = false;
    $subsections[$i]["failed"] = false;
  }
}

for ($i = sizeof($subsections) - 1; $i >= 0; $i--) {
  $isFirstSubsection = false;
  $isLastSubsection = false;
  if ($i == 0) {
    $isFirstSubsection = true;
  }
  if ($i == sizeof($subsections) - 1) {
    $isLastSubsection = true;
  }
  if ($subsections[$i]["failed"] || failOnSubsection($keyPoints, $subsections[$i])) {
    $subsections[$i]["failed"] = true;
    if (!$isFirstSubsection) {
      $subsections[$i]["is_prev_section"] = true;
    }
    if (!$isLastSubsection) {
      if (!$subsections[$i + 1]["is_prev_section"]) {
        $subsections[$i]["is_next_section"] = true;
      }
    }
  } else {
    $subsections[$i]["failed"] = false;
  }
}

```

```php
  // $subsections[$i]["max_index"] = null heißt, dass der Zug auf einer subsection nicht beschleunigt!
  for ($i = 0; $i < sizeof($subsections); $i++) {
    if (!isset($subsections[$i]["max_index"])) {
      $subsections[$i]["brakes_only"] = true;
      $subsections[$i]["max_index"] = $subsections[$i]["indexes"][0];
    } else {
      $subsections[$i]["brakes_only"] = false;
    }
  }
  $subsections = array_values($subsections);
  return array_reverse($subsections);
}

function failOnSubsection(array $keyPoints, array $subsection) {
  global $globalTimeOnOneSpeed;
  $failed = false;
  for ($i = 1; $i < sizeof($subsection["indexes"]); $i++) {
    if ($keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[$subsection["indexes"][$i] - 1]["time_1"] < $globalTimeOnOneSpeed) {
      $failed = true;
      break;
    }
  }
  return $failed;
}

function checkForPostponement(array $localKeyPoints, array $subsection) {
  global $globalTimeOnOneSpeed;

  $keyPoints = $localKeyPoints;
  $timeBeforeMax = 0;
  $timeAfterMax = 0;
  $foundShortSectionBeforeMax = false;
  $foundShortSectionAfterMax = false;
  $indexMaxSection = array_search($subsection["max_index"], $subsection["indexes"]);
  $indexLastKeyPoint = array_key_last($subsection["indexes"]);
  $timeOnMax = $keyPoints[$subsection["max_index"] + 1]["time_0"] - $keyPoints[$subsection["max_index"]]["time_1"] - $globalTimeOnOneSpeed;
  if ($timeOnMax < 0) {
    return false;
  }
  if ($subsection["is_prev_section"]) {
    $timeDiff = $keyPoints[$subsection["indexes"][0]]["time_0"] - $keyPoints[$subsection["indexes"][0] - 1]["time_1"] - $globalTimeOnOneSpeed;
    if ($timeDiff < 0) {
      $timeBeforeMax += $timeDiff;
      $foundShortSectionBeforeMax = true;
    }
  }

  if ($subsection["is_next_section"]) {
```

```
    $timeDiff = $keyPoints[$subsection["indexes"][array_key_last($subsection["indexes"])
        ↪ ] + 1]["time_0"] - $keyPoints[$subsection["indexes"][array_key_last(
        ↪ $subsection["indexes"])]]["time_1"] - $globalTimeOnOneSpeed;
    if ($timeDiff < 0) {
      $timeAfterMax += $timeDiff;
      $foundShortSectionAfterMax = true;
    }
  }

  for ($i = 1; $i <= $indexMaxSection; $i++) {
    if ($keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[$subsection["
        ↪ indexes"][$i] - 1]["time_1"] < $globalTimeOnOneSpeed ||
        ↪ $foundShortSectionBeforeMax) {
      $foundShortSectionBeforeMax = true;
      $timeBeforeMax += $keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[
          ↪ $subsection["indexes"][$i] - 1]["time_1"] - $globalTimeOnOneSpeed;
    }
  }

  for ($i = $indexLastKeyPoint; $i > $indexMaxSection + 1; $i--) {
    if ($keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[$subsection["
        ↪ indexes"][$i] - 1]["time_1"] < $globalTimeOnOneSpeed ||
        ↪ $foundShortSectionAfterMax) {
      $foundShortSectionAfterMax = true;
      $timeAfterMax += $keyPoints[$subsection["indexes"][$i]]["time_0"] - $keyPoints[
          ↪ $subsection["indexes"][$i] - 1]["time_1"] - $globalTimeOnOneSpeed;
    }
  }

  if ($timeBeforeMax > 0) {
    $timeBeforeMax = 0;
  }

  if ($timeAfterMax > 0) {
    $timeAfterMax = 0;
  }

  // true = kann verschoben werden...
  if ($timeOnMax + $timeBeforeMax + $timeAfterMax >= 0) {
    return true;
  } else {
    return false;
  }
}

function toShortInSubsection (array $subsections) {
  $foundError = false;
  foreach ($subsections as $subsection) {
    if ($subsection["failed"]) {
      $foundError = true;
      break;
    }
  }
```

```
1569   return $foundError;
1570 }
1571
1572 function createCumulativeSections ($indexCurrentSection, $indexTargetSection,
       ↪ $currentPosition, $targetPosition, $next_lengths) {
1573   $cumLength = array();
1574   $sum = 0;
1575
1576   foreach ($next_lengths as $index => $value) {
1577     if ($index >= $indexCurrentSection) {
1578       $sum += $value;
1579       $cumLength[$index] = $sum;
1580     }
1581   }
1582   // Berechnung der kummulierten Start- und Endlängen der Abschnitte
1583   // TODO: Geht das auch, wenn Start- und Zielabschnitt der selbe sind?
1584   for ($i = $indexCurrentSection; $i <= $indexTargetSection; $i++) {
1585     if ($indexCurrentSection == $indexTargetSection) {
1586       $cumulativeSectionLengthStart[$i] = 0;
1587       $cumulativeSectionLengthEnd[$i] = $targetPosition - $currentPosition;
1588     } else {
1589       if ($i == $indexCurrentSection) {
1590         $cumulativeSectionLengthStart[$i] = 0;
1591         $cumulativeSectionLengthEnd[$i] = $cumLength[$i] - $currentPosition;
1592       } else if ($i == $indexTargetSection) {
1593         $cumulativeSectionLengthStart[$i] = $cumLength[$i - 1] - $currentPosition;
1594         $cumulativeSectionLengthEnd[$i] = $cumLength[$i - 1] + $targetPosition -
             ↪ $currentPosition;
1595       } else {
1596         $cumulativeSectionLengthStart[$i] = $cumLength[$i - 1] - $currentPosition;
1597         $cumulativeSectionLengthEnd[$i] = $cumLength[$i] - $currentPosition;
1598       }
1599     }
1600   }
1601   return array($cumulativeSectionLengthStart, $cumulativeSectionLengthEnd);
1602 }
1603
1604 function getFahrplanAndPositionForOneTrain (int $trainID, int $zugID) {
1605
1606   global $cacheZwischenhaltepunkte;
1607   global $allUsedTrains;
1608
1609   $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
1610   $keysZwischenhalte = array_keys($cacheZwischenhaltepunkte);
1611
1612   // Get timetable data
1613   $nextBetriebsstellen = getNextBetriebsstellen($zugID);
1614
1615   if ($zugID != null && sizeof($nextBetriebsstellen) != 0) {
1616     for ($i = 0; $i < sizeof($nextBetriebsstellen); $i++) {
1617       if (sizeof(explode("_", $nextBetriebsstellen[$i])) != 2) {
1618         $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["is_on_fahrstrasse"] =
             ↪ false;
```

```
1619        $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebstelle"] =
                ↪ $nextBetriebsstellen[$i];
1620        $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
                ↪ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
1621        $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"] = true;
1622      } else if(in_array($nextBetriebsstellen[$i], $keysZwischenhalte)) {
1623        $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["is_on_fahrstrasse"] =
                ↪ false;
1624        $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["betriebstelle"] =
                ↪ $nextBetriebsstellen[$i];
1625        $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["zeiten"] =
                ↪ getFahrplanzeiten($nextBetriebsstellen[$i], $zugID);
1626        $allUsedTrains[$trainID]["next_betriebsstellen_data"][$i]["fahrplanhalt"] = false
                ↪ ;
1627      }
1628    }
1629    $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array_values($allUsedTrains[
            ↪ $trainID]["next_betriebsstellen_data"]);
1630  } else {
1631    $allUsedTrains[$trainID]["next_betriebsstellen_data"] = array();
1632  }
1633
1634  foreach ($allUsedTrains[$trainID]["next_betriebsstellen_data"] as $betriebsstelleKey
        ↪ => $betriebsstelleValue) {
1635    if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["
          ↪ zeiten"]["abfahrt_soll"] != null) {
1636      $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
            ↪ ]["abfahrt_soll_timestamp"] = getUhrzeit($betriebsstelleValue["zeiten"]["
            ↪ abfahrt_soll"], "simulationszeit", null, array("inputtyp" => "h:i:s"));
1637    } else {
1638      $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
            ↪ ]["abfahrt_soll_timestamp"] = null;
1639    }
1640    if ($allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["
          ↪ zeiten"]["ankunft_soll"] != null) {
1641      $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
            ↪ ]["ankunft_soll_timestamp"] = getUhrzeit($betriebsstelleValue["zeiten"]["
            ↪ ankunft_soll"], "simulationszeit", null, array("inputtyp" => "h:i:s"));
1642    } else {
1643      $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"
            ↪ ]["ankunft_soll_timestamp"] = null;
1644    }
1645    $allUsedTrains[$trainID]["next_betriebsstellen_data"][$betriebsstelleKey]["zeiten"][
          ↪ "verspaetung"] = 0;
1646  }
1647}
1648
1649function toArr(){
1650  return func_get_args();
1651}
1652
1653function emergencyBreak ($id, $distanceToNextStop = 0) {
1654
```

```
1655    global $allUsedTrains;
1656    global $timeDifference;
1657    global $allTimes;
1658
1659    $time = microtime(true) + $timeDifference;
1660    $currentSpeed = $allUsedTrains[$id]["current_speed"];
1661    $targetSpeed = 0;
1662    $notverzoegerung = $allUsedTrains[$id]["notverzoegerung"];
1663    $currentSection = $allUsedTrains[$id]["current_section"];
1664
1665    echo "Der Zug mit der Adresse: ", $allUsedTrains[$id]["adresse"], " leitet jetzt eine
            ↪ Notbremsung ein.\n";
1666    $returnArray = array();
1667    if (getBrakeDistance($currentSpeed, $targetSpeed, $notverzoegerung) <=
            ↪ $distanceToNextStop) {
1668      for ($i = $currentSpeed; $i >= 0; $i = $i - 2) {
1669        array_push($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));
1670        $time = $time + getBrakeTime($i, $i - 1, $notverzoegerung);
1671      }
1672    } else {
1673      $targetSpeedNotbremsung = getTargetBrakeSpeedWithDistanceAndStartSpeed(
              ↪ $distanceToNextStop, $notverzoegerung, $currentSpeed);
1674      $speedBeforeStop = intval($targetSpeedNotbremsung / 2) * 2;
1675      if ($speedBeforeStop >= 10) {
1676        for ($i = $currentSpeed; $i >= 10; $i = $i - 2) {
1677          array_push($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));
1678        $time = $time + getBrakeTime($i, $i - 1, $notverzoegerung);
1679        }
1680        array_push($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" => $id, "wendet" => false, "betriebsstelle" => 'Notbremsung', "
              ↪ live_all_targets_reached" => null));
1681      } else {
1682        array_push($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));
1683        $time = $time + getBrakeTime($currentSpeed, $currentSpeed - 1, $notverzoegerung);
1684        array_push($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));
```

```
1685      }
1686    }
1687
1688    $allTimes[$allUsedTrains[$id]["adresse"]] = $returnArray;
1689    array_push($allUsedTrains[$id]["error"], 3);
1690    return 0;
1691  }
1692
1693  function getTargetBrakeSpeedWithDistanceAndStartSpeed (float $distance, float
       ↪ $verzoegerung, int $speed) {
1694    return sqrt((-2 * $verzoegerung * $distance) + (pow(($speed / 3.6), 2)))*3.6;
1695  }
```

A.4 cache_functions_own.php

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

```
34      $bksig = "BkSig";
35      $vsig = "VSig";
36      $ja = "ja";
37      if ($betriebsstelleKey == 'XAB' || $betriebsstelleKey == "XBL") {
38        $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` = '$ja'");
39        unset($DB);
40      } else if ($betriebsstelleKey == 'XTS') {
41        $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` = '$bksig'");
42        unset($DB);
43      } else if ($betriebsstelleKey == 'XLG') {
44        $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` != '$vsig'");
45        unset($DB);
46      } else {
47        $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` = '$asig'");
48        unset($DB);
49      }
50
51      foreach ($haltepunkte as $haltepunkt) {
52        if ($haltepunkt->wirkrichtung == 0) {
53          array_push($returnArray[$betriebsstelleKey][0], intval($haltepunkt->freimelde_id)
                ↪ );
54        } elseif ($haltepunkt->wirkrichtung == 1) {
55          array_push($returnArray[$betriebsstelleKey][1], intval($haltepunkt->freimelde_id)
                ↪ );
56        }
57      }
58    }
59    $returnArray["XSC"][1] = array(734, 732, 735, 733, 692); // In der Datenbank ist für
          ↪ Richtung 1 für diese Abschnitte fahrplanhalt auf nein eingestellt
60    return $returnArray;
61  }
62
63  function createChacheZwischenhaltepunkte() {
```

```
64    $DB = new DB_MySQL();
65    $allZwischenhalte = array();
66    $returnArray = array();
67    $zwischenhalte = $DB->select("SELECT DISTINCT `".DB_TABLE_SIGNALE_STANDORTE."`.`
          ↪ betriebsstelle` FROM `".DB_TABLE_SIGNALE_STANDORTE."` WHERE `".
          ↪ DB_TABLE_SIGNALE_STANDORTE."`.`betriebsstelle` IS NOT NULL");
68    unset($DB);
69    foreach ($zwischenhalte as $halt) {
70      array_push($allZwischenhalte, $halt->betriebsstelle);
71    }
72    foreach ($allZwischenhalte as $halt) {
73      $DB = new DB_MySQL();
74      $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");
75      unset($DB);
76      if (sizeof($zwischenhalte) == 1) {
77        if (sizeof(explode("_", $halt)) == 2) {
78          $returnArray[$halt] = intval($zwischenhalte[0]->freimelde_id);
79        }
80      }
81    }
82    return $returnArray;
83  }
84
85  function createCacheInfraToGbt () {
86    $DB = new DB_MySQL();
87    $infraArray = array();
88    $returnArray = array();
89    $allInfra = $DB->select("SELECT `".DB_TABLE_FMA_GBT."`.`infra_id` FROM `".
          ↪ DB_TABLE_FMA_GBT."` WHERE `".DB_TABLE_FMA_GBT."`.`infra_id` IS NOT NULL");
90    unset($DB);
91    foreach ($allInfra as $infra) {
92      array_push($infraArray, intval($infra->infra_id));
93    }
94    foreach ($infraArray as $infra) {
95      $DB = new DB_MySQL();
96      $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;
97      unset($DB);
98      $returnArray[$infra] = intval($gbt);
99    }
100   return $returnArray;
101 }
102
103 function createCacheGbtToInfra () {
104
105   $DB = new DB_MySQL();
106
107   $returnArray = array();
108
```

```
109   $allGbt = $DB->select("SELECT DISTINCT `".DB_TABLE_FMA_GBT."`.`gbt_id` FROM `".
          ↪ DB_TABLE_FMA_GBT."` WHERE `".DB_TABLE_FMA_GBT."`.`gbt_id` IS NOT NULL");
110   unset($DB);
111
112   foreach ($allGbt as $gbt) {
113     $DB = new DB_MySQL();
114     $gbt = $gbt->gbt_id;
115     $infras = $DB->select("SELECT `".DB_TABLE_FMA_GBT."`.`infra_id` FROM `".
          ↪ DB_TABLE_FMA_GBT."` WHERE `".DB_TABLE_FMA_GBT."`.`gbt_id` = '$gbt'");
116     unset($DB);
117     $returnArray[$gbt] = array();
118     foreach ($infras as $infra) {
119       array_push($returnArray[$gbt], intval($infra->infra_id));
120     }
121   }
122   return $returnArray;
123 }
124
125 function createCacheFmaToInfra () {
126   $DB = new DB_MySQL();
127   $returnArray = array();
128   $fmaToInfra = $DB->select("SELECT `".DB_TABLE_FMA_GBT."`.`infra_id`, `".
          ↪ DB_TABLE_FMA_GBT."`.`fma_id` FROM `".DB_TABLE_FMA_GBT."` WHERE `".
          ↪ DB_TABLE_FMA_GBT."`.`fma_id` IS NOT NULL");
129   unset($DB);
130   foreach ($fmaToInfra as $value) {
131     $returnArray[intval($value->fma_id)] = intval($value->infra_id);
132   }
133   return $returnArray;
134 }
135
136 function createCacheToBetriebsstelle() {
137   $DB = new DB_MySQL();
138   $returnArray = array();
139   $fmaToInfra = $DB->select("SELECT `".DB_TABLE_SIGNALE_STANDORTE."`.`id`, `".
          ↪ DB_TABLE_SIGNALE_STANDORTE."`.`betriebsstelle` FROM `".
          ↪ DB_TABLE_SIGNALE_STANDORTE."`");
140   unset($DB);
141   foreach ($fmaToInfra as $value) {
142     $returnArray[intval($value->id)] = $value->betriebsstelle;
143   }
144   return $returnArray;
145 }
146
147 function createCacheFahrzeugeAbschnitte () {
148   $DB = new DB_MySQL();
149   $returnArray = array();
150   $fahrzeugeAbschnitte = $DB->select("SELECT `".DB_TABLE_FAHRZEUGE_ABSCHNITTE."`.`
          ↪ fahrzeug_id`, `".DB_TABLE_FAHRZEUGE_ABSCHNITTE."`.`infra_id`, `".
          ↪ DB_TABLE_FAHRZEUGE_ABSCHNITTE."`.`unixtimestamp` FROM `".
          ↪ DB_TABLE_FAHRZEUGE_ABSCHNITTE."`");
151   unset($DB);
152   foreach ($fahrzeugeAbschnitte as $fahrzeug) {
```

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

A.5 globalVariables.php

```
1  <?php
2
3  $globalNotverzoegerung = 2; // Bremsverzögerung bei einer Notbremsung
4  $globalMinSpeed = 10; // Maximale Geschwindigkeit, wenn keine vorgegeben ist
5  $globalSpeedInCurrentSection = 60; // Maximale Geschwindigkeit im aktuellen Abschnitt
6  $globalFirstHaltMinTime = 20; // calculateFahrverlauf -> Zeit fürs Wenden...
7  $globalIndexBetriebsstelleFreieFahrt = 9999999999999;
8  $globalFloatingPointNumbersRoundingError = 0.0000000001;
9  $globalTimeOnOneSpeed = 20;
10 $globalDistanceUpdateInterval = 1;
11
12 $useSpeedFineTuning = true;
13 $useMinTimeOnSpeed = true;
14 $errorMinTimeOnSpeed = false;
15 $slowDownIfTooEarly = true;
16 $useRecalibration = true;
```

A.6 speedOverPosition.m

```
%% Load cumulative sections

fname = '../json/VMaxOverCumulativeSections.json';
fid = fopen(fname);
raw = fread(fid,inf);
str = char(raw');
fclose(fid);
vmaxOverPosition = jsondecode(str);
vmaxOverPosition_Position = vmaxOverPosition(:,1);
vmaxOverPosition_v_max = vmaxOverPosition(:,2);

%% Load modified cumulative sections

fname = '../json/VMaxOverCumulativeSectionsMod.json';
fid = fopen(fname);
raw = fread(fid,inf);
str = char(raw');
fclose(fid);
vmaxOverPosition_mod = jsondecode(str);
vmaxOverPosition_Position_mod = vmaxOverPosition_mod(:,1);
vmaxOverPosition_v_max_mod = vmaxOverPosition_mod(:,2);

%% Load speed over position

fname = '../json/speedOverPosition.json';
fid = fopen(fname);
raw = fread(fid,inf);
str = char(raw');
fclose(fid);
val = jsondecode(str);

speedOverPosition_x_v1 = val(:,1);
speedOverPosition_y_v1 = val(:,2);

%% Load speed over position (all iterationsteps)

fname_it = '../json/speedOverPosition_prevIterations.json';
fid_it = fopen(fname_it);
raw_it = fread(fid_it,inf);
str_it = char(raw_it');
fclose(fid_it);
val_it = jsondecode(str_it);

%% Plot

hold on
figure(1)

% Plot infrastructuresections

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
```

Literatur

Ebuef: Eisenbahn-Betriebs- und Experimentierfeld Berlin. (2021). www.ebuef.de. EBuEf e.V.; c/o Technische Universität Berlin; Fachgebiet Bahnbetrieb und Infrastruktur. (Letzter Zugriff am: 11. September 2021)

Pachl, J. (2021). *Systemtechnik des schienenverkehrs.* Springer-Verlag.

Railcom - DCC-Rückmeldeprotokoll (Norm Nr. RCN-217). (2019, Dezember). (RailCommunity – Verband der Hersteller Digitaler Modellbahnprodukte e.V.)

Richard, H. & Sander, M. (2011). *Technische mechanik. dynamik: Grundlagen - effektiv und anwendungsnah.* Vieweg+Teubner Verlag.