Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability
https://sdq.kastel.kit.edu/wiki/Programmieren
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2022/23

Abschlussaufgabe 2
Version 1.1

20 Punkte

Ausgabe:	28.02.2023, ca. 12:00 Uhr
Abgabe:	14.03.2023, 12:00 Uhr
Abgabefrist:	29.03.2023, 06:00 Uhr

Änderungen v1.1

- Tippfehler in Unterabschnitt A.2
- Randfall ergänzt in Unterunterabschnitt A.3.1
- Tippfehler int Unterabschnitt A.6

Geschlechtergerechte Sprache

Wenn das generische Maskulinum gewählt wurde, geschieht dies zur besseren Lesbarkeit und zum einfachen Verständnis der Aufgabenstellung. Sofern nicht anders angegeben, beziehen sich Angaben im Sinne der Gleichbehandlung auf Vertretende aller Geschlechter.

Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden.

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich zum Ausschluss der Erfolgskontrolle führen.

Kommunikation und aktuelle Informationen

In unseren *FAQs*[1] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen.

In den *ILIAS-Foren* und in den *Artemis-Foren* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

Fragen zu den Abschlussaufgaben müssen im Artemis-Forum unter *Sonstiges* mit dem Präfix "*AA 1*" bzw. "*AA 2*" im Titel gepostet werden.

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[2] einsehen.

Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der verpflichtenden Tests für eine erfolgreiche Abgabe von Abschlussaufgabe 2 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen zuerst die verpflichtenden Tests bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 17*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `java.lang`, `java.io`, `java.util`, `java.util.regex`, `java.util.stream` und `java.util.function`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.

[1] `https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ`
[2] `https://artemis.praktomat.cs.kit.edu/`

- Halten Sie auch alle anderen Checkstyle-Regeln ein.

Diese folgenden Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe. Dennoch wird Ihre Abgabe durch das Abgabesystem *nicht* automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.
- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Geben Sie im Javadoc-Autoren-Tag nur Ihr u-Kürzel an.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki im ILIAS beschreibt, wie Checkstyle verwendet werden kann.

Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 14.03.2023, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 29.03.2023, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

Prüfungsmodus in Artemis

Wenn Sie mit einer Abschlussaufgabe fertig sind, können Sie diese frühzeitig abgeben. Dazu dient Schaltfläche „Vorzeitig abgeben“. Nach der frühzeitigen Abgabe einer Abschlussaufgabe können Sie keine Änderungen an Ihrer Abgabe mehr vornehmen.

Aufgabe A: Verkehrssimulation

In dieser Aufgabe geht es um den Entwurf und die Implementierung einer Verkehrssimulation, in der Straßennetzwerke simuliert werden. So können unterschiedliche Szenarien oder Verkehrsbedingungen in einer kontrollierten Umgebung getestet und erforscht werden. Die Simulation besteht aus einem Modell, welches von der Realität abstrahiert, und einer Ausführungslogik, welche das Modell schrittweise aktualisiert. Ein *Straßennetzwerk* besteht in der Verkehrssimulation aus *Straßen* und *Straßenknoten* (siehe Abbildung A.1). Auf den Straßen fahren *Autos*, die dynamisch miteinander interagieren. Hierbei müssen mehrere hundert Autos simuliert werden können. *Zeit* wird diskret simuliert, also in festen Simulationsschritten als Zeiteinheit. Diese Simulationsschritte werden im Weiteren als *Tick* bezeichnet.

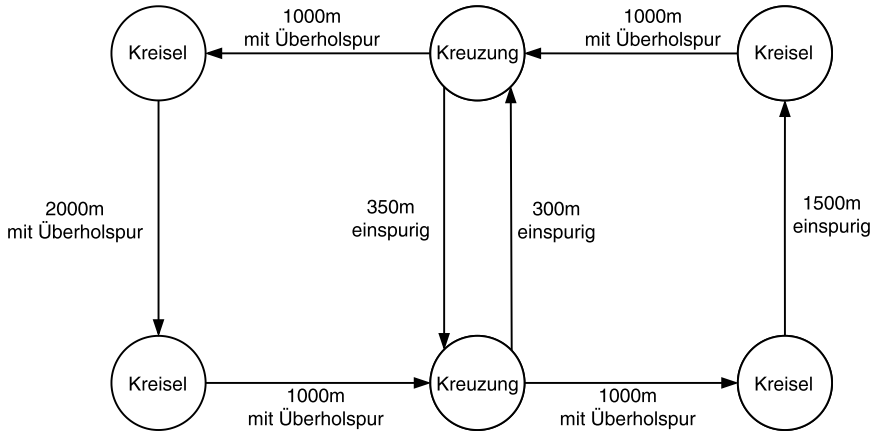

Abbildung A.1: Beispiel eines Straßennetzwerks, welches in dieser Aufgabe simuliert werden kann. Straßenknoten werden als Kreise dargestellt. Straßen werden als Pfeile dargestellt.

A.1 Simulationselemente

Im Folgenden werden die Elemente der Simulation und deren Zusammenhänge beschrieben. Alle Simulationselemente haben einen *Identifikator*, der pro Elementtyp eindeutig ist. Eine Straße und ein Auto können zum Beispiel also den gleichen Identifikator haben. Für zwei Straßen ist das nicht möglich. Der Identifikator ist eine Ganzzahl im Bereich [0, `Integer.MAX_VALUE`].

Längen, Distanzen, Geschwindigkeiten, Beschleunigungen und Zeiteinheiten sind im Rahmen der Simulation *nicht* mit Nachkommagenauigkeit simuliert. Das heißt, die diese Werte sind immer Ganzzahlen der angegebenen Einheiten.

A.1.1 Straßen

Straßen werden in der Simulation *unidirektional* modelliert. Somit ist die Gegenrichtung einer Straße in der echten Welt, in der Simulation eine andere Straße, deren Eigenschaften unabhängig von der ursprünglichen Straße ist. Eine Straße ist immer mit genau zwei verschiedenen Straßenknoten

verbunden, einem Start- und einem Endknoten. Autos können auf einer Straße nur vom Startknoten zum Endknoten fahren.

Straßen haben folgende Eigenschaften:

- eine Länge in Meter [m] (minimal 10, maximal 10000)
- eine Maximalgeschwindigkeit in Meter pro Tick $\left[\frac{\text{m}}{\text{Tick}}\right]$ (minimal 5, maximal 40)
- eine Straßenart (*einspurig* oder *mit Überholspur*)

Einspurige Straßen Auf einspurigen Straßen können Autos nicht überholen bzw. überholt werden. Die Reihenfolge der Autos auf einer solchen Straße kann sich deshalb nicht ändern, abgesehen davon, dass Autos die Straße verlassen oder betreten.

Straßen mit Überholspur Straßen mit Überholspur haben, wie auch einspurige Straßen, nur eine Fahrspur. Jedoch kann ein Auto das vorausfahrende Auto überholen, vorausgesetzt alle Abstandsregeln (siehe Unterunterabschnitt A.3.1) ermöglichen dies.

A.1.2 Straßenknoten

Ein Straßenknoten verbindet Straßen im Netzwerk. Jeder Straßenknoten kann bis zu vier eingehende und bis zu vier ausgehenden Straßen haben. Jeder Knoten muss aber mindestens eine eingehende und eine ausgehende Straße haben. Autos, die den Straßenknoten überqueren, fahren immer von einer eingehenden Straße zu einer ausgehenden Straße. Um die Simulation zu vereinfachen, werden die Knoten als Punkte interpretiert, deren Überquerung keine Zeit kostet. Autos werden also von dem Ende einer Straße direkt zu Beginn der nächsten versetzt. Die Reihenfolge, in der Straßen mit einem Knoten verbunden werden, ist relevant für die Simulation. Es gibt zwei Arten von Straßenknoten: *Kreisel* und *Kreuzungen*.

Kreisel Ein Kreisel ist ein Straßenknoten, bei dem Autos von allen eingehenden Straßen gleichzeitig den Knoten zu allen ausgehenden Straßen überqueren können. Autos behindern sich also nicht beim Überqueren des Kreisels.

Kreuzung Kreuzungen sind Straßenknoten mit Ampeln, bei dem immer nur die Autos einer eingehenden Straße den Knoten überqueren dürfen. Dabei handelt es sich um die Straße, deren Ampel grün ist. Kreuzungen haben folgende Eigenschaften:

- eine Grünphasendauer in Ticks [Tick] (minimal 3, maximal 10)
- ein Grünphasenindikator, der angibt, welche eingehende Straße grün hat.

Straßen bekommen in der Reihenfolge grün, in der sie mit der Kreuzung verbunden wurden. Zuerst hat also die zuerst verbundene Straße grün, danach die zweite. Hatten alle eingehenden Straßen grün, bekommt als Nächstes wieder die erste Straße grün.

A.1.3 Autos

Autos befinden sich immer genau auf einer Straße. Autos befinden sich insbesondere nie auf einem Straßenknoten. In der Simulation werden Autos als Punkte dargestellt, sie haben also eine Länge von 0 Metern. Allerdings muss jedes Auto immer mindestens 10 Meter Abstand zum nächsten vorher fahrenden Auto halten. Auf einer Straße mit 50 Meter Länge passen also bis zu sechs Autos. Der Mindestabstand gilt auch für das Überqueren eines Straßenknotens. Ein Auto kann also einen Straßenknoten nur dann in eine ausgehende Straße überqueren, wenn das nächste Auto dieser Straße mindestens 10 Meter auf dieser Straße gefahren ist. Um vom Startknoten bis zum Endknoten einer Straße zu fahren, muss ein Auto die gesamte Länge der Straße gefahren haben. Dies geschieht abhängig von der aktuellen Geschwindigkeit des Autos und folgenden Eigenschaften des Autos:

- eine Wunschgeschwindigkeit im Meter pro Tick $\left[\frac{\mathrm{m}}{\mathrm{Tick}}\right]$ (minimal 20, maximal 40)
- eine Beschleunigung in Meter pro Tick^2 $\left[\frac{\mathrm{m}}{\mathrm{Tick}^2}\right]$ (minimal 1, maximal 10)

Zudem haben Autos eine Wunschrichtung. Die Wunschrichtung gibt an, auf welche ausgehende Straße das Auto bei dem nächsten Straßenknoten abbiegen will. Zuerst will ein Auto immer auf die erste ausgehende Straße (in der Reihenfolge, in der die Straßen zum Knoten hinzugefügt wurden) abbiegen. Nach jedem Abbiegevorgang wird die Wunschrichtung erhöht. Nach dem vierten Abbiegevorgang ist die Wunschrichtung wieder die erste Straße. Hat ein Straßenknoten weniger ausgehende Straßen als maximal möglich, kann es sein, dass die gewünschte n-te Straße nicht existiert. In diesem Fall wird die nächste existierende Straße in der Rotation gewählt. Dennoch wird die Wunschrichtung für den nächsten Abbiegevorgang so erhöht, als ob die Straße existiert hätte.

Beispiel: Die Wunschrichtung ist die vierte Straße. Allerdings hat der Straßenknoten nur drei ausgehende Straßen. Deshalb biegt das Auto auf die erste Straße ab. Nach dem Abbiegen ist die neue Wunschrichtung die erste Straße.

A.2 Simulationsaufbau

Zu Beginn der Simulation wird das Straßennetzwerk aufgebaut. Dabei wird geprüft, ob das Netzwerk vollständig ist. So darf es zum Beispiel keine Kreuzungen ohne Straßen geben. Zudem dürfen nicht zu viele Autos auf eine Straße gesetzt werden. Ein Netzwerk ohne Autos kann trotzdem valide sein. Ein Netzwerk, das weder Knoten, noch Straßen, noch Autos hat zählt auch als valide.

Auch alle anderen Einschränkungen aus Unterabschnitt A.1 müssen beachtet werden. Der Grünphasenindikator einer Kreuzung indiziert eingehende Straßen in der Reihenfolge, wie die Straßen der Kreuzung hinzugefügt wurden. Wegen der Wunschrichtung von Autos gilt dies ebenfalls für die ausgehende Straßen.

Die Straßen werden vom Ende aus mit ihren Autos befüllt. Dazu wird das erste Auto auf das Ende der Straße gesetzt. Danach werden alle anderen Autos mit dem Mindestabstand von 10 Metern dahinter eingereiht. Eine Straße mit 70 Metern Länge und fünf Autos hat nach dem Aufbau alle Autos auf den letzten ~~50~~ 40 Metern der Straße (siehe Abbildung A.2).

Zu Beginn haben die Autos die Geschwindigkeit 0 $\frac{\mathrm{m}}{\mathrm{Tick}}$. Die initiale Wunschrichtung aller Autos hat den Wert 0. Die Grünphase aller Kreuzungen hat ebenso den Wert 0.

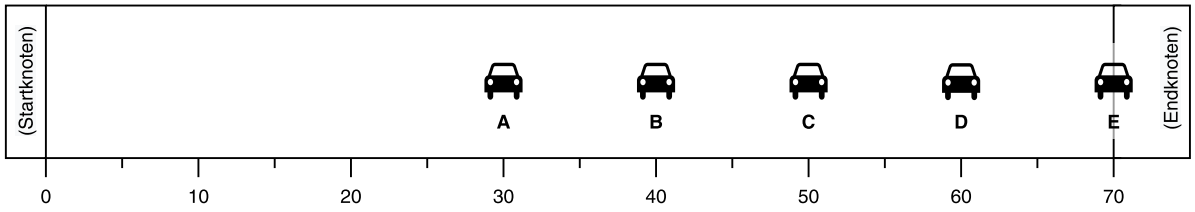

Abbildung A.2: Übersicht über die initiale Befüllung einer Straße der Länge 70 Meter mit fünf Autos. Die Autos wurden in der Reihenfolge (E, D, C, B, A) eingelesen. Auf der Straße können sich maximal 8 Autos gleichzeitig befinden.

A.3 Simulationsschritte

Ein Simulationsschritt entspricht einem Tick. In diesem Schritt wird das Modell der Simulation aktualisiert. Hierbei können sich Autos bewegen und Grünphasen werden gegebenenfalls angepasst. Ein Simulationsschritt überführt das Straßennetzwerk also von einem validen Ausgangszustand immer in einen validen Folgezustand. Die Aktualisierung läuft in folgender Reihenfolge ab:

1. Aktualisierung aller Straßen aufsteigend dem Identifikator nach
2. Aktualisierung aller Kreuzungen aufsteigend dem Identifikator nach

Zum Beispiel werden in einem Netzwerk mit den Straßen S_0 und S_1 sowie den Kreuzungen K_0 und K_1 die Elemente in der Reihenfolge (S_0, S_1, K_0, K_1) aktualisiert (die Nummern entsprechen dem Identifikator). Im Folgenden werden die einzelnen Teilabläufe des Simulationsschrittes näher beschrieben.

A.3.1 Aktualisierung einer Straße

Innerhalb einer Straße werden alle Autos in ihrer Reihenfolge von hinten (Endknoten) nach vorne (Startknoten) aktualisiert.

Im Rahmen einer Aktualisierung eines Autos wird zuerst die Geschwindigkeit angepasst. Dann wird das Auto gegebenenfalls bewegt. Zur Aktualisierung der Geschwindigkeit wird die Formel $v_{neu} = min\left(v_{alt} + a * (1\,\text{Tick}), v_{wunsch}, v_{max}\right)$ verwendet. Hierbei ist v_{neu} die neue Geschwindigkeit des Autos, v_{alt} die bisherige Geschwindigkeit des Autos, v_{wunsch} die Wunschgeschwindigkeit des Autos, v_{max} die Maximalgeschwindigkeit der Straße und a die Beschleunigung des Autos. Die Geschwindigkeit eines vorausfahrenden Autos beeinflusst nicht die Aktualisierung der Geschwindigkeit.

Jedes Auto bewegt sich dann entsprechend der angepassten Geschwindigkeit auf der Straße nach vorne (siehe Abbildung A.3). Die Bewegung kann aber vorzeitig beendet werden. Jedes Auto kann nur so weit fahren, wie der Mindestabstand es zulässt. Die Ausnahme hierbei ist das Überholen auf Straßen mit Überholspur. Befindet sich zum Beispiel ein Auto mit einer angepassten Geschwindigkeit von 10 Meter pro Tick allein auf einer Straße mit 15 Metern Abstand zum Ende der Straße, wird es bis auf 5 Meter an das Ende der Straße hinbewegt. Befände sich jedoch ein Auto am Ende der Straße, würde das fahrende Auto nach 5 Metern stoppen.

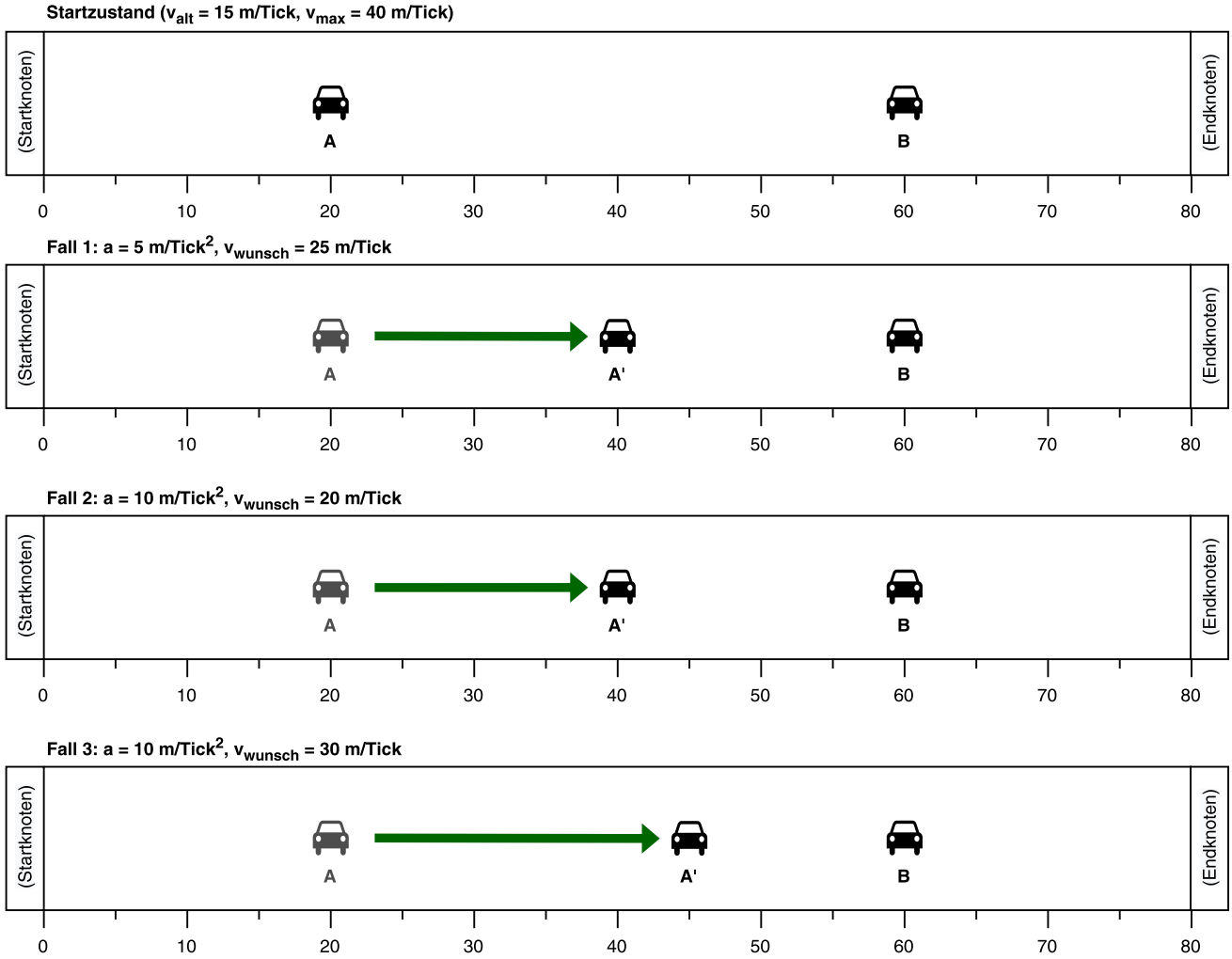

Abbildung A.3: Bewegung des Autos A in einem Tick für verschiedenen Fällen abhängig von Wunschgeschwindigkeit (v_{wunsch}), Beschleunigung (a), aktueller Geschwindigkeit (v_{alt}) sowie Maximalgeschwindigkeit (v_{max}). Nach Abschluss des Ticks steht das Auto an Position A'

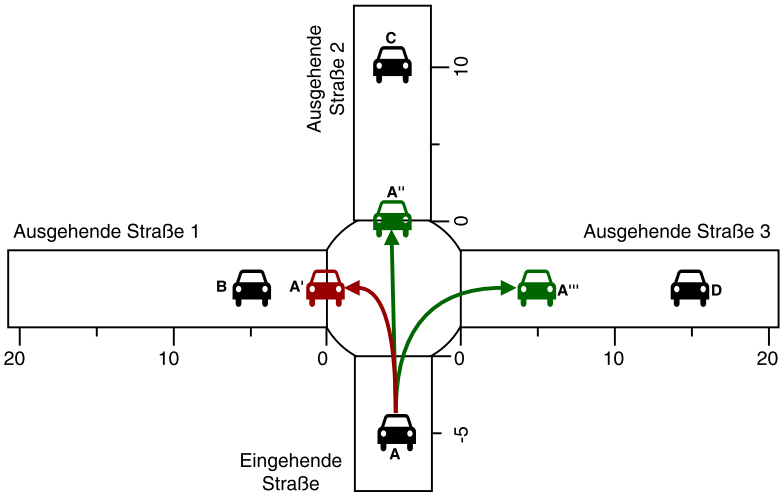

Abbildung A.4: Abbiegevorgänge eines Autos A auf die ausgehenden Straßen eines Knotens. Das Auto fährt mit der (angepassten) Geschwindigkeit 10 Meter pro Tick. Abbiegen auf Position A' ist nicht erlaubt. Abbiegen auf A'' und A''' ist erlaubt. Der tatsächliche Abbiegevorgang hängt von der Wunschrichtung und dem Straßenknoten ab.

Hat ein Auto das Ende der Straße erreicht und kann es im aktuellen Tick mindestens noch einen Meter fahren, kann es über den Straßenknoten abbiegen (abbiegen bezeichnet hierbei das Überqueren des Knotens, unabhängig von der Richtung). Dies geht jedoch, nur wenn nach dem Abbiegevorgang auf der neuen Straße der Mindestabstand zum herausfahrenden Auto eingehalten werden kann. Die verbleibende Fahrtstrecke des Ticks kann nach dem Abbiegen dann noch direkt (also noch während des aktualisieren des Autos) auf der neuen Straße weitergefahren werden. Auch hier muss wie immer der Mindestabstand eingehalten werden (siehe Abbildung A.4). Die Maximalgeschwindigkeit der neuen Straße wird hierbei ignoriert. Unabhängig von der verbleibenden Fahrtstrecke darf ein Auto aber nicht erneut abbiegen. Es bleibt in diesem Fall also am *Ende* der neuen Straße stehen.

Über Kreisel darf immer abgebogen werden. Über Kreuzungen darf nur abgebogen werden, wenn die entsprechende eingehende Straße grün hat. Autos, die während der Aktualisierung ihrer Straße auf eine andere Straße abgebogen sind, dürfen bei der Aktualisierung der anderen Straße nicht erneut bewegt werden. Pro Simulationsschritt wird also jedes Auto nur einmal bewegt.

Befindet sich ein Auto auf einer Straße mit Überholspur, kann es das vorausfahrende Auto gegebenenfalls überholen (siehe Abbildung A.5). Dabei endet die Bewegung nicht bei dem Erreichen des Minimalabstandes zum vorausfahrenden Auto. Stattdessen wird das Auto weiterbewegt, als ob das vorausfahrende Auto nicht existiert. Dies ist jedoch nur möglich, wenn nach dem Ende der Bewegung des Autos erneut alle Mindestabstände (sowohl nach vorne als auch nach hinten) eingehalten werden können (siehe Fall 1 und Fall 3 in Abbildung A.5). Es kann hierbei immer nur ein Auto überholt werden (siehe Fall 2). Wurde ein Auto überholt, kann das überholende Auto nicht im gleichen Tick noch die Kreuzung überqueren (Fall 4). Das bedeutet über einen Straßenknoten wird nie überholt und das vorderste Auto kann nur überholt werden, wenn das überholende Auto am Ende der Straße stehen bleiben kann. Nach dem Abbiegen darf beim Fahren der verbleibenden Fahrtstrecke auf der neuen Straße das Auto nicht mehr überholen. Überholen und Abbiegen findet für ein Auto also nie im gleichen Tick statt.

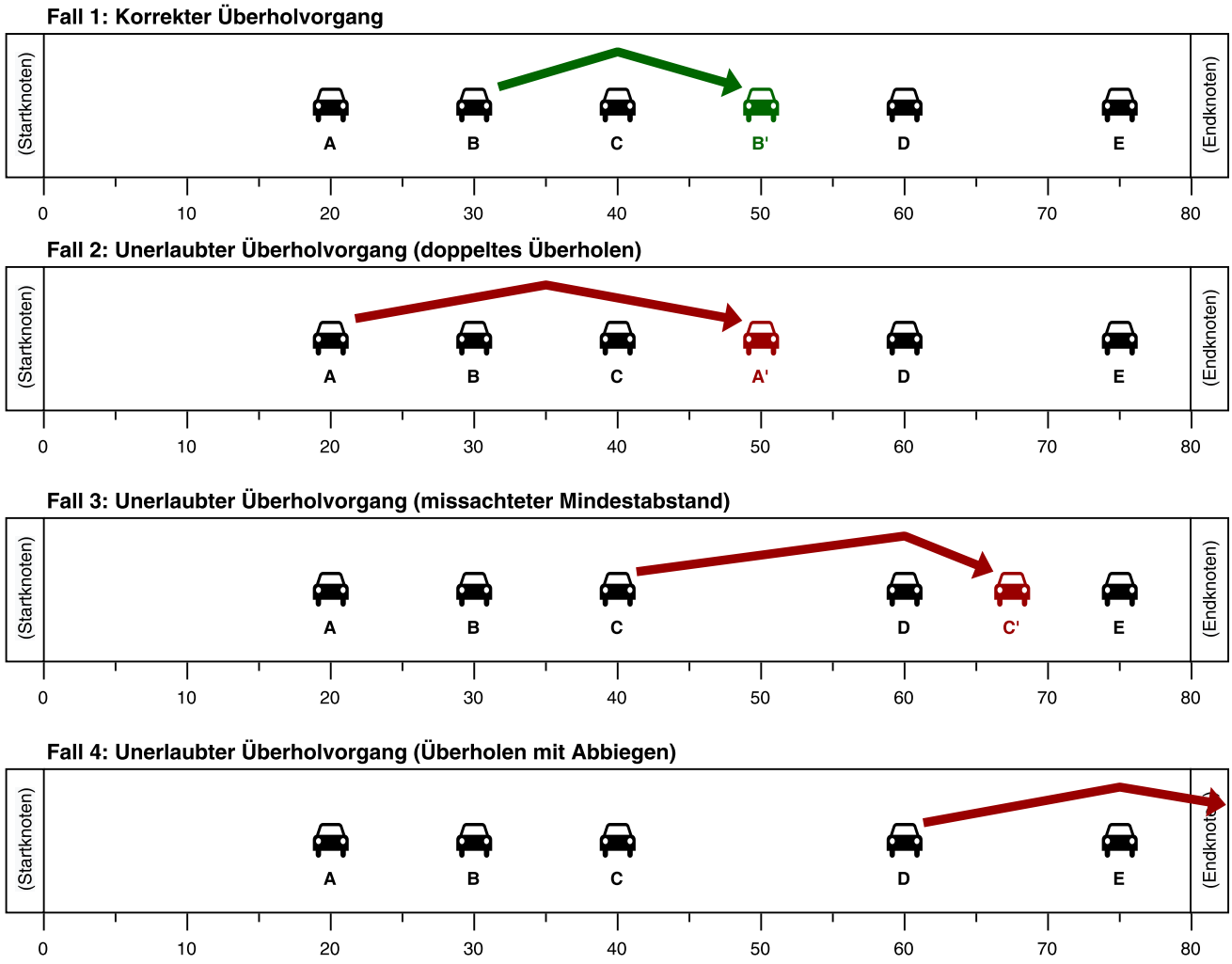

Abbildung A.5: Übersicht über erlaubte (Fall 1) und unerlaubte (Fall 2-4) Überholvorgänge auf einer Straße mit Überholspur. Fall 1 ist nur erlaubt, falls zusätzliche Bedingungen wie Restdistanz im aktuellen Tick gegeben sind.

Wird ein Auto in einem Tick nicht bewegt, wird die Geschwindigkeit des Autos nach der Geschwindigkeitsanpassung auf 0 gesetzt. Das reine Abbiegen zählt hierbei jedoch nicht als Bewegung. Dies ist zum Beispiel in einem Stau der Fall, also wenn der Abstand zum vorausfahrenden Auto bereits Mindestabstand entspricht und nicht überholt werden kann.

A.3.2 Aktualisierung einer Kreuzung

Zur Aktualisierung einer Kreuzung muss nur die Grünschaltung angepasst werden. Dafür wird die verbleibende Dauer der aktuellen Grünphase um einen Tick reduziert. Ist die verbleibende Dauer dann 0, müssen die Ampeln umgeschaltet werden. Dazu wird der Grünphasenindikator, welcher angibt, welche eingehende Straße grün hat, um eins erhöht. Falls er die letzte eingehende Straße indiziert, wird er stattdessen auf 0 gesetzt. Anschließend muss die Grünphasendauer wieder auf den Ausgangswert der Kreuzung gesetzt werden.

A.4 Simulationsdateien

Die Netzwerkstruktur und alle Simulationselemente werden aus Text-Dateien geladen. Dafür werden immer genau drei Dateien benötigt: die Datei *streets.sim*, die Datei *crossings.sim*, sowie die Datei *cars.sim*. Die Datei *streets.sim* enthält Informationen über alle Straßen, inklusive an welche Straßenknoten sie angebunden sind. Die Datei *crossings.sim* enthält zusätzlich Informationen über die Straßenknoten. Die Datei *cars.sim* enthält Informationen über alle Autos, inklusive auf welcher Straße sie starten.

A.4.1 `SimulationFileLoader`

Die Klasse stellt drei Methoden bereit, also jeweils eine pro Datei. Die Klasse erwartet als Konstruktor-Parameter den Dateipfad, welcher angibt, in welchem Ordner die Simulationsdateien liegen. Die Methoden lesen die Dateien jeweils ein und geben den Inhalt der Datei zeilenweise als Liste von Zeichenketten zurück. Jede Zeile ist dabei das Eingabeformat für ein Simulationselement. Der Inhalt dieser Liste, inklusive der syntaktischen Korrektheit des Eingabeformats, muss selbst überprüft werden. Beachten Sie die Javadoc-Kommentare der entsprechenden Methoden.

A.4.2 Eingabeformat

Im Folgenden wird das Eingabeformat für alle drei Dateien diskutiert. Platzhalter für Werte werden dabei mit eckigen Klammern gekennzeichnet: `[Platzhalter]`

Hinweis: Straßen sind die einzigen Simulationselemente, welche keine expliziten Identifikatoren im Eingabeformat besitzen. Hier müssen die Identifikatoren deshalb dynamisch vergeben werden. Dies geschieht in der Einlesereihenfolge (also in der Datei *streets.sim* von oben nach unten) startend mit dem Identifikator `0`.

Straßen haben das folgende Eingabeformat (keine Leerzeichen):

`[Startknoten]-->[Endknoten]:[Länge]m,[Art]x,[Limit]max`

Beispiel: `3-->1:300m,2x,130max`

- `[Startknoten]` und `[Endknoten]` geben den Identifikator des jeweiligen Straßenknotens an.
- `[Länge]` gibt die Länge der Straße in Meter an.
- `[Art]` ist die Straßenart, also entweder `1` für einspurig oder `2` für einspurig mit Überholspur.
- `[Limit]` ist die Maximalgeschwindigkeit in Meter pro Tick.

Kreuzungen haben das folgende Eingabeformat (keine Leerzeichen):

`[ID]:[Dauer]t`

Beispiel: `0:5t`

- `[ID]` gibt den Identifikator des Straßenknotens an.
- `[Dauer]` gibt die Grünphasendauer in Ticks an. Ist der Wert `0`, ist der Knoten ein Kreisel. Ist der Wert größer als `0`, ist der Knoten eine Kreuzung.

Autos haben das folgende Eingabeformat (keine Leerzeichen):

`[ID],[Straße],[v],[a]`

Beispiel: `5,2,20,5`

- `[ID]` gibt den Identifikator des Autos an.
- `[Straße]` gibt den Identifikator der Straße an, auf der das Auto startet.
- `[v]` gibt die Wunschgeschwindigkeit in Meter pro Tick an.
- `[a]` gibt die Beschleunigung in Meter pro Tick^2 an.

A.5 Benutzerschnittstelle

Nach dem Start des Programmes erfolgt die Interaktion über die Kommandozeile. Nach dem Starten des Programms kann der Nutzer eine Reihe von gültigen Befehlen eingeben. Ein gültiger Befehl besteht immer aus einem Schlüsselwort und je nach Befehl zudem aus einem durch ein Leerzeichen getrennten Parameter:

`[Schlüsselwort]␣[Parameter]`

Ungültige Befehle (egal ob durch inkorrekte Befehlsbezeichnung oder inkorrekte Parameter) führen zu einer Fehlermeldung, welche auf der Kommandozeile ausgegeben wird. Jede Fehlermeldung beginnt mit der Zeichenkette `Error:` (keine Leerzeichen). Danach soll eine sinnvolle Nachricht, passend zum verursachenden Fehler, folgen. Fehlermeldungen bestehen immer aus einer Zeile und enthalten keine Zeilenumbrüche.

A.5.1 Load-Befehl

Befehlsschema: `load [path]`

Der Load-Befehl lädt die Simulationselemente aus den Eingabedateien. Nutzen Sie dazu den `SimulationFileLoader` und den Dateipfad, welcher als Parameter `[path]` dem Load-Befehl übergeben wird (es ist hierbei egal, ob der Pfad relativ oder absolut ist). Der Dateipfad ist eine Zeichenkette und kann direkt dem `SimulationFileLoader` Konstruktor übergeben werden. Der Inhalt der Zeichenkette muss hierbei *nicht* validiert werden. Ein Dateipfad kann keine Leerzeichen enthalten. Sie müssen eventuelle Fehler des `SimulationFileLoader` entsprechend weiterverarbeiten. Der `SimulationFileLoader` liest die Simulationselemente in der Reihenfolge ein, in der sie in den entsprechenden Dateien aufgelistet sind. Straßenknoten werden erzeugt und mit den passenden Straßen verbunden. Hierbei wird in der Reihenfolge vorgegangen, die durch das Einlesen definiert wurde. Danach werden die Autos in der Reihenfolge des Einlesens auf die Straßen verteilt (siehe Unterabschnitt A.2). Nach dem erfolgreichen Laden wird `READY`(Großbuchstaben) auf der Kommandozeile ausgegeben. Fehler müssen zum Beispiel ausgegeben werden, wenn der Inhalt der Dateien, welche der `SimulationFileLoader` lädt, nicht zum erwarteten Eingabeformat passt, wenn die Werte der Simulationselemente nicht im erlaubten Bereich liegen oder wenn die Simulationselemente ein unvollständiges bzw. inkorrektes Netzwerk bilden.

Der Load-Befehl kann auch mehrfach ausgeführt werden. Hierbei werden alle Schritte des Load-Befehls normal ausgeführt und das aktuelle Netzwerk nach dem erfolgreichen Laden verworfen. Dies kann genutzt werden, um das Netzwerk in den Ausgangszustand zu bringen.

Beispielinteraktion

```
1 > load files/basic
2 READY
```

Beispielinteraktion

```
1 > load load/invalid
2 Error: Street 7 cannot have more than 15 cars.
```

A.5.2 Simulate-Befehl

Befehlsschema: `simulate [ticks]`

Der Simulate-Befehl simuliert eine bestimmte Anzahl an Simulationsschritten, welche durch eine Ganzzahl `[ticks]` spezifiziert wird. Die Simulationsschritte werden nacheinander durchgeführt (siehe Unterabschnitt A.3). Nach der Durchführung aller Simulationsschritte wird `READY` ausgegeben. Der Befehl `simulate 10` hat die gleiche Auswirkung wie das zehnfache Aufrufen des Befehls `simulate 1`. Fehler müssen ausgegeben werden, wenn `[ticks]` keine gültige Ganzzahl oder negativ ist. Zudem darf der Simulate-Befehl nicht vor dem ersten Aufruf des Load-Befehls aufgerufen werden.

Beispielinteraktion

```
> simulate 5
READY
```

Beispielinteraktion

```
> simulate 1
Error: Street network is yet to be loaded.
```

A.5.3 Position-Befehl

Befehlsschema: `position [ID]`

Der Position-Befehl gibt Informationen über die Position und die Geschwindigkeit eines bestimmten Autos aus. Der Parameter `[ID]` ist der Identifikator eines Autos und spezifiziert dadurch, welches Auto gemeint ist. Nach dem Aufruf des Befehls werden die Informationen wie folgt ausgegeben:

`Car [ID] on Street [S] with speed [v] and position [P]`

Dabei ist:

- `[S]` der Identifikator der Straße auf der sich das Auto befindet
- `[v]` die aktuelle Geschwindigkeit des Autos in Meter pro Tick
- `[P]` die Distanz, welche das Auto auf der aktuellen Straße bereits gefahren ist.

Fehler müssen ausgegeben werden, wenn `[ID]` kein gültiger Identifikator eines Autos ist und wenn der Position-Befehl vor dem ersten Aufruf des Load-Befehls aufgerufen wird. Jedoch darf der Befehl aufgerufen werden, bevor der Simulate-Befehl das erste Mal aufgerufen wurde.

Beispielinteraktion

```
> position 19
Car 19 on street 7 with speed 20 and position 1370
```

Beispielinteraktion

```
> position 744
Error: There is no car with the identifier 744.
```

A.5.4 Quit-Befehl

Befehlsschema: `quit`

Der Quit-Befehl beendet das Programm. Beachten Sie, dass für das Testen Ihrer Abgabe dieser Befehl essenziell ist, da viele der Tests `quit` am Ende einer Testsequenz verwenden, um Ihr Programm zu beenden. Stellen Sie daher sicher, dass der Befehl in jeder Situation funktioniert. Beachten Sie

bitte nochmals, dass Sie hierfür **nicht** `System.exit()` oder andere ausgeschlossene Funktionen verwenden dürfen.

A.6 Beispielinteraktion

Im Folgenden wird eine Beispielinteraktion mit ~~drei~~ zwei bestimmten Netzwerken gezeigt. Neben diesem Netzwerk befinden sich noch andere Netzwerke im ausgelieferten Beispielprojekt. Erstellen Sie auch eigene Netzwerke mit verschiedenen Eigenschaften, um Ihre Lösung zu testen. Ihre Lösung muss für beliebige, valide Netzwerke funktionieren.

Das folgende Netzwerk (*files/basic*) wird zum Beispiel im Rahmen der Beispielinteraktion zuerst eingelesen:

Datei *streets.sim*:

```
0-->1:40m,1x,15max
1-->0:60m,1x,35max
```

Datei *crossings.sim*:

```
0:0t
1:0t
```

Datei *cars.sim* (Auszug):

```
0,0,40,5
1,1,30,10
```

Die Interaktion kann dann beispielsweise wie folgt ablaufen:

Beispielinteraktion

```
%> java TrafficNetworkSimulation
> load files/basic
READY
> position 0
Car 0 on street 0 with speed 0 and position 40
> simulate 1
READY
> position 0
Car 0 on street 1 with speed 5 and position 5
> position 1
Car 1 on street 0 with speed 10 and position 10
> simulate 3
READY
> position 0
Car 0 on street 1 with speed 20 and position 50
```

Beispielinteraktion

```
> position 1
Car 1 on street 1 with speed 15 and position 15
> simulate 2
READY
> position 0
Car 0 on street 0 with speed 15 and position 30
> position 1
Car 1 on street 0 with speed 30 and position 10
> load files/advanced
READY
> simulate 29
READY
> position 0
Car 0 on street 5 with speed 25 and position 92
> position 1
Car 1 on street 5 with speed 18 and position 42
> position 3
Car 3 on street 2 with speed 10 and position 70
> position 7
Car 7 on street 1 with speed 9 and position 38
> position 12
Car 12 on street 2 with speed 12 and position 90
> position 17
Car 17 on street 1 with speed 10 and position 18
> position 22
Car 22 on street 4 with speed 0 and position 370
> position 27
Car 27 on street 6 with speed 0 and position 190
> position 32
Car 32 on street 6 with speed 0 and position 150
> quit
```

A.7 Tipp: Visualisierung des Straßennetzwerks

Das Straßennetzwerk, welches in der Datei *streets.sim* definiert ist, ist *Mermaid*[3] kompatibel. Deswegen können die Straßennetzwerke im Mermaid Live Web-Editor[4] visualisiert werden. Ersetzen Sie dazu den Mermaid-Beispielcode im Web-Editor mit der Zeile `stateDiagram-v2`[5]. Kopieren Sie danach den Inhalt der Datei *streets.sim* darunter. Falls sie statt den IDs der Straßenknoten die Eigenschaften sehen wollen, kopieren sie zusätzlich den Inhalt der Datei *crossings.sim* darunter. Autos können nicht visualisiert werden. Die Visualisierung der Straßennetzwerke dient lediglich als Hilfsmittel und ist für die Umsetzung der Abschlussaufgabe **nicht** benötigt.

[3] `https://mermaid.js.org/intro`
[4] `https://mermaid.live`
[5] `https://mermaid.js.org/syntax/stateDiagram.html`