TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT VERKEHRSWISSENSCHAFTEN „FRIEDRICH-LIST“
PROFESSUR FÜR VERKEHRSBTRIEBSLEHRE UND LOGISTIK
PROF. DR. JÖRN SCHÖNBERGER

B-u-S-Sim - Eine C++ - Bibliothek zur Simulation von Fahrzeugbewegungen in ÖPV-Netzwerken

Jörn Schönberger

Dresden, 04.10.2021 Version 2.0
b-u-s-sim.de

Inhaltsverzeichnis

1 Änderungen / Updates der aktuellen Version gegenüber Vorversionen

- Die Platzierung von Netzwerkelementen (Haltestellen, Betriebshöfen) erfolgt nun über die Spezifikation der zugehörigen (realwelt)-Geokoordinaten in der Form (Längengrad;Breitengrad). Hierzu finden sich in Abschnitt 5.3.2 weitere Ausführungen.

- Zusätzlich zum ÖPV-Netzwerk können nun zur besseren geographischen Orientierung verschiedene Typen von Landschaftskomponenten, so.g. *TOI* (''Things-of-Interest") spezifiziert und eingerichtet werden. Details zur Umsetzung stellt Abschnitt 5.2 bereit.

- Der Default-Wert für die Abweichung der Haltestellen-Aufenthaltszeit `BUSSIM_DEVIATION_WAITING_TIME` vom Erwartungswert beträgt jetzt 0 Zeiteinheiten, d.h. es gibt in der Standard-Einstellung keine unsicheren Haltestellenaufenthaltszeiten.

- Betriebshalte und Verkehrshalte werden nun gemeinsam ein- und ausgeblendet durch Drücken der Komstenkombination `<SHIFT>+<S>`.

- Zu den Linienverläufen können nun Richtungspfeile eingeblendet werden. Hierzu gibt es den Menu-Eintrag `show direction indicators`. Diese Richtungspfeile können auch mit der Tastenkombination `<SHIFT>+<D>` aktiviert und deaktiviert werden.

- Ab sofort werden alle Pfeile unabhängig vom Typ durch das Betätigen der Tastenkombination `<SHIFT>+<I>` ein- und ausgeblendet.

- Während der Simulation kann jetzt für jeden Pfeil eingeblendet werden, in welcher Reihenfolge die sich auf dem Pfeil befindlichen Fahrzeuge befinden. Diese Reihenfolgen können mit der Tastenkombination `<SHIFT>+<W>` ein- bzw. ausgeblendet werden.

- Jedes aktivierte Fahrzeug verfügt nun über einen Richtungsanzeiger, der die aktuelle Fahrtrichtung anzeigt.

2 Was ist B-u-S-Sim ?

B-u-S-Sim ist ein in C++ programmiertes Tool, mit dem Bewegungen von Bussen und/oder Schienenfahrzeugen in öffentlichen Personenverkehrs-Netzwerken nachgebildet / simuliert werden können. Es ist derart konzipiert, dass Nutzer mit geringen C++ - Vorkenntnissen ein selbst gewähltes reales ÖPV-Netzwerk nachbauen müssen/können. Durch eine Simulation der daraus resultierenden Prozesse kann beobachtet werden, welche Auswirkungen die getroffenen Entscheidungen haben.
Bei der Konzeption von B-u-S-Sim wurden drei Zielsetzungen verfolgt:

1. Vorhandene rudimentäre Kenntnisse der Programmiersprache sollen in einem anschaulichen Kontext angewendet und vertieft werden. Damit wird die Motivation insb. von Einsteigern in C++ zur weitergehenden Auseinandersetzung mit der C++ - Programmiersprache erhöht.

2. Neben der reinen Programmierarbeit wird die Arbeit mit Daten thematisiert. Oftmals ist es im Zusammenhang mit Programmierarbeiten so, dass die Suche, Zusammenstellung, Aufbereitung, Strukturierung und Codierung von Daten sehr wichtig ist. Diese hohe Bedeutung von Daten soll bei der Arbeit mit B-u-S-Sim verinnerlicht werden.

3. für verkehrsaffine Studierende und Forscher ist es oftmals schwierig, realistische oder realitätsnahe Netzwerke zu untersuchen. B-u-S-Sim versucht daher, Unterstützung bei der Analyse der Performance ganzer Netzwerke oder einzelner Netzkomponenten zu geben.

B-u-S-Sim erhebt keinen Anspruch, reale ÖPV-Netzwerke steuern zu können. Vielmehr soll vereinfachend demonstriert werden, welche Daten/Informationen wie zusammenspielen müssen. Die Simulation der Abläufe in einem solchen System und deren Visualisierungen ist eine Möglichkeit, Wirkungen verschiedener Planungsansätze zu verdeutlichen und zu demonstrieren.
B-u-S-Sim kann daher für verschiedene Zwecke genutzt werden bzw. verschiedene Nutzergruppen ansprechen. Einerseits eignet es sich für Einsteiger in die C++ - Programmierung, da man mit relativ wenig Overhead-Aufwand (Vorarbeiten) schnell anschauliche Ergebnisse erzielen kann. Andererseits eignet sich B-u-S-Sim aber auch zur Erstellung von anschaulichen Beispielen, so dass insb. Lehrende mit verkehrswissenschaftlichen Interessen B-u-S-Sim Sinn stiftend nutzen können.

3 Installation

3.1 Überprüfung und ggf. Aktualisierung der Grafiktreiber

B-u-S-Sim nutzt OpenGL (`https://www.opengl.org/`) zur Erzeugung von Grafiken. Die Software OpenGL kann man sich vereinfachend vorstellen als eine Kollektion von standardisierten Befehlen zur Erzeugung von Computergrafiken, die unabhängig von einer konkreten Programmiersprache und einem konkreten Computer verwendbar sind.
Die einzige Voraussetzung für die Nutzbarkeit der OpenGL-Grafikbefehle ist, dass die Grafikkarte des verwendeten Computers diese zulässt bzw. unterstützt. Dies ist aber in den allermeisten Fällen der Fall.
Um sicher zu gehen, dass die Grafikkarte korrekt installiert ist und die jeweils neuesten Treiber inkl. OpenGL-Befehlsbibliotheken auf den verwendeten Computersystem vorhanden sind, sollten Sie zunächst überprüfen, dass Sie die aktuellen zu der Grafikkarte passenden Treiber installiert haben.
Verwenden Sie hierfür die Administrations-Software bzw. die Systemsteuerung Ihres Rechners.
Falls Sie prüfen möchten, ob OpenGL auf Ihrem System korrekt installiert ist, dann können Sie hierzu das Programm glview verwenden. Dies ist unter der `realtech-vr.com/admin/glview` verfügbar.

3.2 Getestete C++ - Compiler

B-u-S-Sim wurde in der Entwicklungsumgebung *Code::Blocks* programmiert. Die Erstellung der Simulationssoftware wurde auf einem Windows 10 - System (64Bit) mit zwei verschiedenen Versionen des C++ - Compilers g++ getestet. Es handelt sind um die sog. 32-Bit-Variante `mingw32-g++` sowie die sog. 64-Bit-Variante `x86_64-w64-mingw32-g++`. Die nachfolgenden Installationsschritte sind bei beiden Compilern gleich, unterscheiden sich jedoch in Details, so dass nachfolgend die Installation für beide Variante separat beschrieben wird.
Die 32-Bit-Version (mit `mingw32-g++`) wurde als Bestandteil des kombinierten Installationspakets von *Code::Blocks* mit MinGW getestet (*Code::Blocks* -Version 17.12). Diese ist verfügbar unter der URL

`https://sourceforge.net/projects/codeblocks/files/Binaries/17.12/Windows/codeblocks-17.12mingw-setup.exe`

Die 64-Bit-Version (mit `x86_64-w64-mingw32-g++`) wurde als Bestandteil der kombinierten Installation von *Code::Blocks* mit MinGW getestet (*Code::Blocks* -Version 20.03). Diese ist verfügbar unter der URL

`https://sourceforge.net/projects/codeblocks/files/Binaries/20.03/Windows/codeblocks-20.03mingw-setup.ex`

Falls Sie *Code::Blocks* noch nicht installiert haben, dann wählen Sie ein der beiden Varianten aus und folgen der passenden nachstehenden Anleitung in3.4 für die 32-Bit-Version (mit `mingw32-g++`) bzw. für die 64-Bit-Version (mit `x86_64-w64-mingw32-g++`) in Abschnitt 3.5.
Falls Sie bereits *Code::Blocks* mit MinGW installiert haben, dann müssen Sie zunächst herausfinden, welche Compiler-Variante bei Ihnen installiert ist. Starten Sie hierzu *Code::Blocks* und öffnen

Sie im Menü `Settings` die Option `Compiler`. In dem sich öffnenden Fenster wählen Sie die Karteikarte `Toolchain executables`. Wenn Sie in der Zeile `C++`-Compiler eine der beiden Compiler-Varianten sehen, wissen Sie, um welche installierte Variante es sich handelt. Falls keine der beiden Varianten ausgewählt ist, durchsuchen Sie die *Code::Blocks* -Installation durch Betätigen der Schältfläche `...`.

3.3 Herunterladen der Bibliothek `freeglut`

B-u-S-Sim nutzt neben den Standard-Befehlen von C++ weitere sog. Befehls-Bibliotheken. Insbesondere wird für die grafische Darstellung die Erweiterung `freeglut` der Standard C++-Version auf dem jeweiligen Rechner benötigt. Diese Software unterliegt eigenen Rechten und Bestimmungen. Für deren Funktionsfähigkeit kann keine Verantwortung übernommen werden. Die für `freeglut` geltenden Rechte sind unbedingt und uneingeschränkt zu beachten.
Bevor wir das erste B-u-S-Sim -Projekt übersetzen können und starten, müssen wir die frei und kostenlos verfügbare `freeglut`-Bibliothek auf dem Rechner installieren. Sobald dies einmal geschehen ist, können neue Projekte darauf zugreifen. Laden Sie `freeglut` unter Verwendung der URL

```
https://www.transmissionzero.co.uk/files/software/development/GLUT/freeglut-MinGW.zip
```

herunter. Nachdem der Download abgeschlossen ist, entpacken Sie die erhaltene zip-Datei. Nach dem Entpacken der zip-Datei stehen drei Verzeichnisse zur Verfügung: `bin`, `include` und `lib`. Die in den Verzeichnissen enthaltenen Dateien müssen wir nun für *Code::Blocks* und den Compiler MinGW nutzbar machen, in dem wir die Dateien an die richtigen Stellen im Dateisystem verschieben bzw. kopieren. MinGW sucht in speziellen, vorgegebenen Verzeichnissen nach zusätzlichen Befehlen in den heruntergeladenen Bibliotheken. Dafür müssen die zugehörigen `.h`-Dateien und die Bibliothek-Dateien (mit der Endungen `.a` bzw. `.dll`) in die vorgesehenen Verzeichnisse auf dem eigenen Rechner kopiert werden.

3.4 32-Bit-Variante `mingw32-g++`

3.4.1 Kopieren der `freeglut`-Header-Dateien

Wir beginnen mit den sog. Header-Dateien aus dem `include`-Verzeichnis

- Wechseln Sie in das Verzeichnis, das durch das Entpacken der heruntergeladenen zip-Datei entstanden ist. Wechsel Sie dort weiter in das `include`-Verzeichnis und dort in das `GL`-Verzeichnis. Sie finden vier `.h`-Dateien dort.
- Kopieren Sie diese in die Zwischenablage.
- Wechseln Sie nun in das Verzeichnis, in dem Sie *Code::Blocks* installiert haben.[1]
- Wechseln Sie dort weiter in das Unterverzeichnis `MinGW\include`.
- Legen Sie dort ein neues Unterverzeichnis `freeglut` an. Kopieren Sie die vier `.h`-Dateien dort hinein.

3.4.2 Kopieren der Bibliotheks-Dateien

Wir fahren fort mit den Bibliotheken im `lib`-Verzeichnis.

- Wechseln Sie in das Unterverzeichnis `lib` des oben entpackten `freeglut`-zip-Archivs. Dort finden Sie ein Unterverzeichnis `x64` und zwei `.a`-Dateien.

[1] Typischerweise ist dies `C:\Program Files (x86)\CodeBlocks` oder `C:\Program Files\CodeBlocks`

- Kopieren Sie die beiden `.a`-Dateien in die Zwischenablage (das Verzeichnis `64` ignorieren Sie)
- Wechseln Sie nun wieder in das Verzeichnis, in dem Sie *Code::Blocks* installiert haben (s.o.)
- Wechseln Sie dort weiter in das Unterverzeichnis `MinGW\lib`.
- Fügen Sie dort die beiden `.a`-Dateien ein.

Nun haben wir die zur Erzeugung des Grafikbildschirms benötigten `freeglut`-Dateien an den richtigen Positionen im Dateisystem hinterlegt.
Im Verzeichnis `bin` befinden sich sog. `.dll`-Dateien. Diese werden durch ein Programm erst zum Zeitpunkt der Programmausführung gelesen. Wir können hier auf das Kopieren dieser Dateien verzichten, da das o.a. B-u-S-Sim -zip-Archiv bereits eine solche `.dll`-Datei enthält. Diese befindet sich im Unterverzeichnis `\bin\release`.

3.4.3 Integration von `freeglut` in *Code::Blocks*

Abschließend müssen wir in *Code::Blocks* noch hinterlegen, dass `MinGW` die `freeglut`-Dateien (und weitere, die mit dem Betriebssystem mitkommen) nutzen soll. Hierzu laden wir das B-u-S-Sim -Projekt in *Code::Blocks* . Im Menü `Settings` wählen wir die Option `Compiler`. Wählen Sie die Karteikarte `Linker settings`. Klicken Sie auf die `Add`-Schaltfläche und tragen Sie in das sich öffnende Fenster den Eintrag

```
C:\Program Files (x86)\CodeBlocks\MinGW\lib\libopengl32.a
```

ein. Wiederholen Sie die `Add`-Aktion zweimal und tragen Sie dabei die Einträge

- `C:\Program Files (x86)\CodeBlocks\MinGW\lib \libfreeglut.a` und
- `C:\Program Files (x86)\CodeBlocks\MinGW\lib\libfreeglut.a`

ein.

3.5 64-Bit-Variante

3.5.1 Kopieren der `freeglut`-Header-Dateien

Wir beginnen mit den sog. Header-Dateien aus dem `include`-Verzeichnis

- Wechseln Sie in das Verzeichnis, das durch das Entpacken der heruntergeladenen zip-Datei entstanden ist. Wechsel Sie dort weiter in das `include`-Verzeichnis und dort in das `GL`-Verzeichnis. Sie finden vier `.h`-Dateien dort.
- Kopieren Sie diese in die Zwischenablage.
- Wechseln Sie nun in das Verzeichnis, in dem Sie *Code::Blocks* installiert haben.[2]
- Wechseln Sie dort weiter in das Unterverzeichnis `MinGW\x86_64-w64-mingw32\include`.
- Legen Sie dort ein neues Unterverzeichnis `freeglut` an. Kopieren Sie die vier `.h`-Dateien dort hinein.

[2] Typischerweise ist dies `C:\Program Files (x86)\CodeBlocks`.

3.5.2 Kopieren der Bibliotheks-Dateien

Wir fahren fort mit den Bibliotheken im `lib`-Verzeichnis.

- Wechseln Sie in das Unterverzeichnis `lib` des oben entpackten `freeglut`-zip-Archivs. Dort finden Sie ein Unterverzeichnis `x64` und zwei `.a`-Dateien.
- Wechseln Sie in das Verzeichnis `x64` und kopieren Sie die beiden `.a`-Dateien in die Zwischenablage.
- Wechseln Sie nun wieder in das Verzeichnis, in dem Sie *Code::Blocks* installiert haben (s.o.)
- Wechseln Sie dort weiter in das Unterverzeichnis `MinGW\x86_64-w64-mingw32\lib`.
- Fügen Sie dort die beiden `.a`-Dateien ein.

Nun haben wir die zur Erzeugung des Grafikbildschirms benötigten `freeglut`-Dateien an den richtigen Positionen im Dateisystem hinterlegt.
Im Verzeichnis `bin\x64` befindet sich die sog. `.dll`-Datei. Diese wird durch ein Programm erst zum Zeitpunkt der Programmausführung gelesen (dynamically linked library). Sie müssen diese Datei kopieren und in das Unterverzeichnis `\bin\release` in Ihrem B-u-S-Sim -Projekt kopieren. Da ist zwar schon eine gleichnamige DLL-Datei enthalten, aber die dort enthaltene DLL-Datei ist eine 32-Bit-Version und muss durch die soeben kopierte 64-Bit-DLL-Datei ersetzt werden.

3.5.3 Integration von `freeglut` in *Code::Blocks*

Abschließend müssen wir in *Code::Blocks* noch hinterlegen, dass `MinGW` die `freeglut`-Dateien (und weitere, die mit dem Betriebssystem mitkommen) nutzen soll. Hierzu laden wir das B-u-S-Sim -Projekt in *Code::Blocks* . Im Menü `Settings` wählen wir die Option `Compiler`. Wählen Sie die Karteikarte `Linker settings`. Klicken Sie auf die `Add`-Schaltfläche und tragen Sie in das sich öffnende Fenster den Eintrag

`C:\Program Files (x86)\CodeBlocks\MinGW\x86_64-w64-mingw32\lib\libopengl32.a`

ein. Wiederholen Sie die `Add`-Aktion zweimal und tragen Sie dabei die Einträge

- `C:\Program Files (x86)\CodeBlocks\MinGW\x86_64-w64-mingw32\lib\libfreeglut.a` und
- `C:\Program Files (x86)\CodeBlocks\MinGW\x86_64-w64-mingw32\lib\libfreeglut.a`

ein.

3.5.4 Anpassen des Suchpfads

Im Suchpfad eines Betriebssystems sind alle Verzeichnisse hinterlegt, in der der Compiler von weiteren benötigten Dateien (z.B. Bibliotheken) sucht. Wir müssen nun noch abschliessend das Verzeichnis, in dem die *.a-Dateien liegen, dem Suchpfad hinzufügen. Suchen Sie im WIndows-Menü nach „Systemumgebungsvariablen bearbeiten“. In dem sich öffnenden Fenster klicken Sie auf die Schaltfläche Umgebungsvariablen und dann wähen Sie im Bereich Systemvariablen den Eintrag Path. Klicken Sie auf „bearbeiten“und fügen Sie über die Schaltfläche „Durchsuchen“den Pfad `C:\Program Files (x86)\CodeBlocks\MinGW\x86_64-w64-mingw32\lib` hinzu. Schliessen Sie mit Ok alle geöffneten Fenster.

3.6 Festlegung des verwendeten C++-Sprachstandards

B-u-S-Sim verwendet einige C++-Befehle bzw. Konventionen, die erst ab der Sprachversion 11 verfügbar sind. Daher muss dem verwendeten Compiler mitgeteilt werden, dass der diese Version nutzen muss. Hierzu ist in *Code::Blocks* eine Konfiguration vorzunehmen. Im Menü Settings in der Kategorie Options ist rechts neben der Option *Have g++ follow the C++11 ISO C++ language standard* der Haken zu setzen.
Damit ist die Konfiguration Ihres Rechners für die Nutzung von abgeschlossen.

3.7 Erstellung des ausführbaren Programms

Wir müssen nun abschließend aus dem entpackten *Code::Blocks* -Projekt ein ausführbares Programm erstellen. Hierzu wählen Sie im *Code::Blocks* -Menü `Build` die Option `Rebuild` aus oder Sie Drücken alternativ die Tastenkombination <STRG>+<F11>.
Sobald die Compilierung und das Verlinken erfolgreich durchgelaufen sind, öffnen Sie eine Kommandozeile (`cmd`) und wechseln Sie in das Projektverzeichnis. Von dort wechseln Sie weiter in das Unterverzeichnis `bin\release`. Durch die Eingabe des Befehls `bus_sim.exe` starten Sie die Simulation. Klicken Sie mit dem Mauszeiger einmal in das geöffnete Simulationsfenster. Drücken Sie anschließend die <a>-Taste, um die Simulation automatisch ablaufen zu lassen.

4 Grundkonzepte von B-u-S-Sim

Nachfolgend werden einige Konzepte beschrieben, die bei der Benutzung von B-u-S-Sim zu berücksichtigen sind.

4.1 Das Sichtbare zuerst ...

Nach dem Start des Programms über die Kommandozeile öffnet sich ein Fenster, in dem das im Simulationsprogramm codierte Netzwerk inkl. Fahrzeuge sowie modellierter weiterer Objekte räumlich dargestellt wird. Die Simulation läuft zeitgesteuert ab. Sie kann in zwei verschiedenen Modi genutzt werden. Im *manuellen Modus* wird ein Zeitschritt durch das Drücken der <+>-Taste ausgelöst. Im *automatischen Modus* wird automatisch nach 1000 Millisekunden die Simulationszeit um eine Zeiteinheit erhöht. Mit fortschreitender Simulationszeit bewegen sich die Fahrzeugsymbole entlang der hinterlegten Linienverläufe durch das Netzwerk (Abbildung 1).
Gestartet und gesteuert wird das Simulationsprogramm über eine Liste von Befehlen. Durch das Klicken mit der rechten Maustaste irgendwo im Simulationsfenster erscheint eine Liste der Befehle. Jeder Befehl kann durch die angezeigte Taste oder durch die Auswahl des Menü-Eintrags ausgeführt werden.
Das Drücken der mittleren Maustaste öffnet eine Liste aller Linienverläufe. Es können einzelne oder alle Linienverläufe ausgeblendet oder (wieder) eingeblendet werden. Voraussetzung hierfür ist, dass Linienverläufe nicht grundsätzlich als unsichtbar spezifiziert sind (vgl. Abschnitt 5.5).

4.2 Zeitmodell

Eine Zeiteinheit in B-u-S-Sim repräsentiert eine Realzeit-Minute. Die aktuelle Systemzeit wird in der Variablen `CURRENT_TIME` gespeichert und iterativ in einer Schleife sukzessive fortgeschrieben. In jedem Schleifendurchlauf wird sie um `BUSSIM_TIME_STEP` Zeiteinheiten erhöht. Das Ende eines Simulationslaufs (in der Regel eines Fahrplantags) wird eingeleitet, sobald der Simulationsendzeitpunkt `BUSSIM_END_OF_DUTY` erreicht wird. Die Werte dieser Konstanten werden in der im Projekt beinhalteten Datei `bussim_global.cpp` festgelegt und können dort auch verändert werden.
Eine Simulation startet immer zum Simulationszeitpunkt `CURRENT_TIME=0`.Standardmäßig beträgt das Zeitinkrement `BUSSIM_TIME_STEP` 0.1, d.h. in jedem Durchlauf wird ein rechnerischer Fortschritt der Simulationszeit um 0.1 Sekunden realisiert. Ohne Änderung der Grundeinstellungen wird das

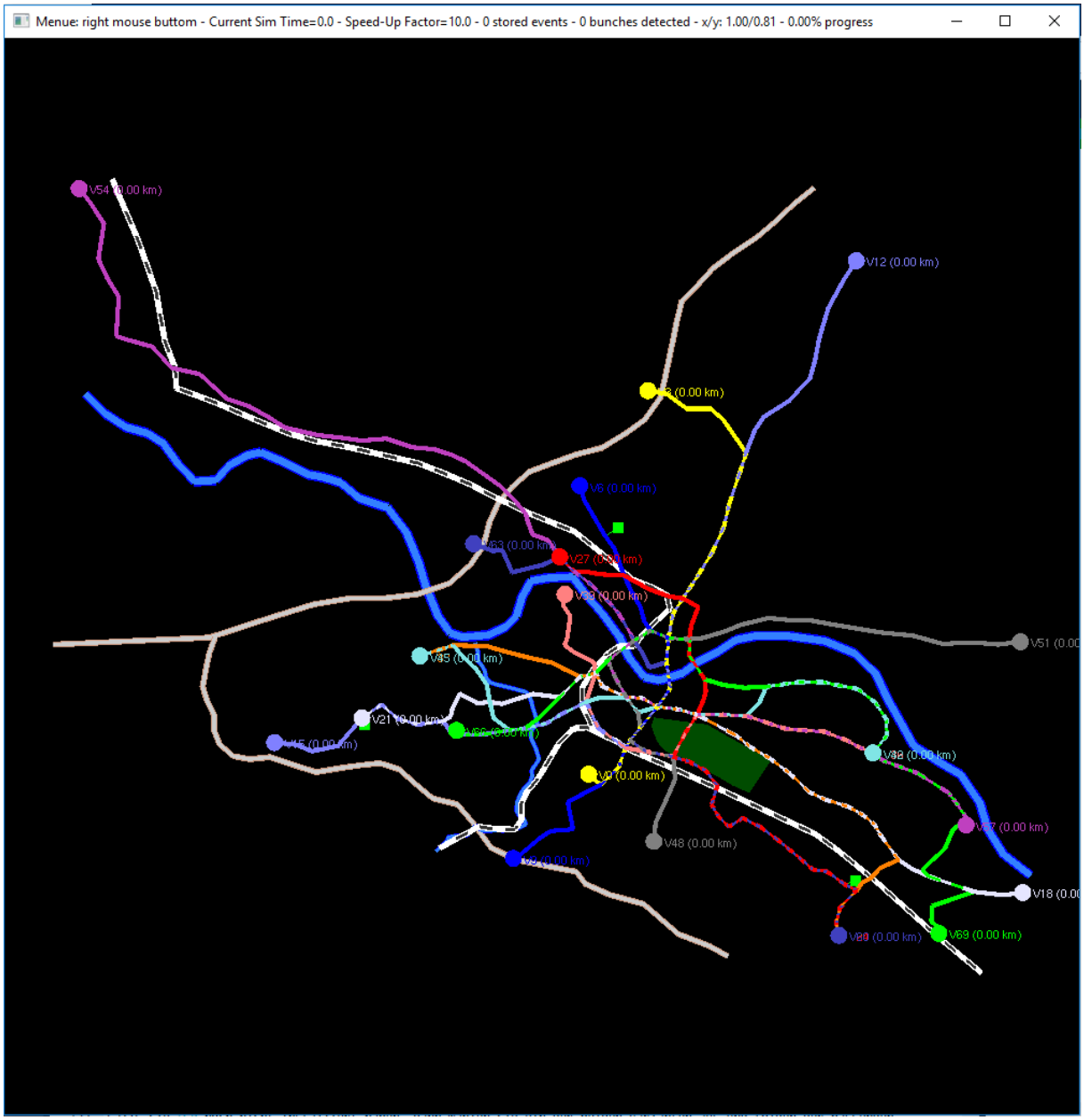

Abbildung 1: Das Simulationsfenster von B-u-S-Sim

Ende eines Simulationslaufs nach simulierten 24h, d.h. nach $24 \cdot 60 = 1440$ Simulationsminuten oder $1440 \cdot 4 = 14400$ Iterationen erreicht. Fahrzeuge, die zum Zeitpunkt des Erreichens des Zeitpunkts `BUSSIM_END_OF_DUTY` noch „unterwegs“sind, fahren noch bis zum Ende ihres aktuell bedienten Linienverlaufs und stoppen dort ihre Weiterfahrt endgültig. Daher endet die Simulation teilweise erst deutlich nach Erreichen des Simulationszeitpunkts `BUSSIM_END_OF_DUTY`.
Die Simulation kann in zwei verschiedenen Modi ablaufen. Im *manuellen Modus* wird ein Zeitschritt durch das Drücken der <+>-Taste ausgelöst. Im *automatischen Modus* wird automatisch nach 1000 Millisekunden die Simulationszeit `CURRENT_TIME` um `BUSSIM_TIME_STEP` Minuten erhöht.
Der manuelle Modus ist nach dem Start als Ausführungsmodus voreingestellt. Durch das Betätigen der <a>-Taste kann zu jeder Zeit in den automatischen Modus gewechselt werden. Sobald dann später die <m>-Taste gedrückt wird, wechselt die Simulation zurück in den manuellen Modus.
Im automatischen Modus ist es grundsätzlich möglich, den Ablauf der Simulation zu beschleunigen. Hierfür kann der Beschleunigungsfaktor `BUSSIM_INITIAL_SPEEDUP_FACTOR` verwendet werden. Dieser steht initial auf dem Wert 10 und kann durch das Drücken der Tasten <s> („slower“) und <f> („faster“) verändert werden. Im automatischen Modus wird der nächste Zeitschritt dann bereits nach 1000/`BUSSIM_INITIAL_SPEEDUP_FACTOR` Millisekunden durchgeführt. Das Inkrement des Simulationszeitpunktes ändert sich durch eine Variation des Beschleunigungsfaktors nicht. Aus numerischen Gründen ist es aber möglich, dass sich die Abläufe und Ereigniszeitpunkte der Simulation bei verschiedenen Beschleunigungsfaktoren geringfügig unterscheiden.

4.3 Raum-Zeit-Modell

Nachdem wir nun wissen, wie die fortschreitende Zeit innerhalb der Simulation gesteuert wird, müssen wir erklären, wie Fahrzeuge mit fortschreitender Zeit auf den vorgegebenen Streckenverläufen weiterfahren. Dies ist auch deshalb notwendig, um zu verstehen, welche Eingangsdaten Sie in die Simulation integrieren müssen, um ein reales Netzwerk möglichst echt darzustellen.
Um die Orientierung auch in komplexen Netzwerken zu erleichtern können sog. *Things-of-Interests* (TOI) modelliert und angezeigt werden. Ein TOI kann z.B. ein Fluss oder sonstiges Gewässer, eine Autobahn, eine Eisenbahnstrecke, ein Wald bzw. eine Grünfläche oder ein Gebirge sein.
B-u-S-Sim verwendet einen mathematischen Graphen $\mathcal{G} := (\mathcal{V}, \mathcal{A}, \rho)$ zur Speicherung des Netzwerks. Die festen Orte werden als Haltestellen in der Knotenmenge $\mathcal{V}$ hinterlegt. Damit diese korrekt am Bildschirm angezeigt werden können, müssen die Geo-Koordinaten des Punktes bekannt sein.
Jeder Streckenabschnitt $(a; b) \in \mathcal{A}$ stellt einen Streckenabschnitt dar, auf dem ein Fahrzeug von der Haltestelle $a \in \mathcal{A}$ ohne Zwischenhalt zur nächsten Haltestelle $b \in \mathcal{A}$ fährt. Somit bilden die vorhandenen Streckenabschnitte die Pfeilmenge $\mathcal{A}$. Wichtig für die Bestimmung von Fahrtzeiten und Ankunftszeiten ist die Länge (in Kilometern) $\rho(a)$ eines Streckenabschnitts $a \in \mathcal{A}$.
Ein Linienverlauf (eine Linie) ist nun durch eine Sequenz verbundener Pfeile im Graphen $\mathcal{G}$ definiert. Enthalten zwei Linienverläufe einen identischen Knoten, so stellt dieser eine Umsteigehaltestelle dar. Der Linienverlauf gibt des Weg vor, den ein Fahrzeug durch das Netzwerk nehmen muss. Mit fortschreitender Zeit bewegt sich das Fahrzeug auf diesem Weg weiter. Um wie viele Kilometer sich das Fahrzeug je Zeitschritt auf dem Linienverlauf fortbewegt hängt von dessen Geschwindigkeit ab. Somit muss für jedes Fahrzeug eine Geschwindigkeit v hinterlegt werden.
Hat ein Fahrzeug das Ende eines Linienverlaufs erreicht, so wechselt es auf einen anderen Linienverlauf. Somit muss zu jedem Linienverlauf abgespeichert werden, welches der nächste zu befahrene Linienverlauf ist. Hierfür wird für jedes Fahrzeug ein sog. Umlaufplan spezifiziert.
Die aktuelle Position eines Fahrzeugs wird durch drei Informationen bestimmt. Einerseits ist zu jedem Zeitpunkt für jedes Fahrzeug der Pfeil $\vec{a} \in \mathcal{A}$ hinterlegt, auf dem sich das Fahrzeug gerade befindet. Die exakte Position des Fahrzeuge auf dem Pfeil $\vec{a} \in \mathcal{A}$ wird durch den Wert ϕ festgelegt. Dieser gibt die Prozentzahl der Länge von $\vec{a}$ an, den das Fahrzeug schon auf dem Pfeil gefahren ist. Wenn $\vec{o}$ der Ortsvektor der Starthaltestelle des Pfeils $\vec{a}$ ist, dann ist die aktuelle Fahrzeugposition $\vec{o} + \phi \cdot \vec{a}$ (wir setzen hier vereinfachend voraus, dass der durch $\vec{a}$ repräsentierte Streckenabschnitt eine gerade Strecke ist). Bei

jedem Zeitschritt wird der Wert ϕ für jedes Fahrzeug entsprechend der aktuellen Geschwindigkeit des Fahrzeugs erhöht. Vereinfachend nehmen wir an, dass ein Fahrzeug entweder mit konstanter Geschwindigkeit unterwegs ist oder gerade an einer Haltestelle wartet. Falls das Fahrzeug das Ende eines Pfeils erreicht hat (d.h. ϕ den Wert 1 annimmt), wird $\vec{a}$ auf den im Linienverlauf, dem das Fahrzeug gerade folgt, direkt nachfolgenden Pfeil gesetzt. Dadurch wird es möglich eine zeitabhängige Fahrzeugbewegung im Raum (bzw. in der Ebene) abzubilden bzw. nachzubilden. Abschließend ist für ein Fahrzeug der aktuell bediente Linienverlauf hinterlegt, damit bei Erreichen des Endpunktes eines Pfeils eindeutig geregelt ist, welcher nächste Pfeil befahren werden muss.

4.4 Events

Während der Durchführung einer Simulation geschehen verschiedene Dinge, die relevant bzw. wichtig sind. B-u-S-Sim folgt dem Konzept, diese sog. **Events** während einer Simulation nacheinander mit einem Zeitstempel versehen zu erfassen, zu speichern und nach dem Abschluss der Simulation strukturiert am Bildschirm zeitlich sortiert auszugeben.

Wert	Konstante	Erklärung
0	BUSSIM_ARRIVAL	ein Fahrzeug erreicht eine Haltestelle
1	BUSSIM_DEPARTURE	ein Fahrzeug verlässt eine Haltestelle
2	BUSSIM_VEHICLE_ACTIVATION	ein Fahrzeug beginnt seine erste Aktivität
3	BUSSIM_VEHICLE_DEACTIVATION	ein Fahrzeug beendet seine aktuelle Aktivität, kann aber später reaktivert werden
4	BUSSIM_VEHICLE_END_OF_WORK	ein Fahrzeug beendet seine letzte Aktivität für den Rest des Tages
5	BUSSIM_VEHICLE_BLOCKAGE	ein Fahrzeug wird von einem vorausfahrenden Fahrzeug ausgebremst

Tabelle 1: Events in B-u-S-Sim

Die überwachten Events in B-u-S-Sim sind in Tabelle 1 zusammengestellt und erklärt. In der Header-Datei `bussim_global.h` werden diese Konstanten definiert.
Jedes einzelne Event wird mit einem Zeitstempel versehen in einer Liste während der Simulation abgespeichert. Zusätzlich wird zu dem Event eine Erklärung erzeugt, die verbal alle relevanten Informationen wie die Fahrzeugnummer, die Haltestelle oder den Streckenabschnitt enthält. Am Ende einer Simulation wird diese Liste mit durch Tabulatoren separierten Spalten am Bildschirm ausgegeben. Mit dem Startbefehl `bus_sim.exe > result.txt` kann diese Liste dann am Ende der Simulation in die Text-Daten `result.txt` geschrieben werden. Diese Textdatei kann dann einfach in z.B. Excel importiert und dort ausgewertet werden. Ein kleines Beispiel zur Demonstration wird in Abschnitt 6 besprochen.
Als einziges Event verursacht das Event 5 (BUSSIM_VEHICLE_BLOCKAGE) schon während des Simulationsablaufs eine sichtbare Aktivität. Im Simulationsfenster wird an der Stelle der Blockierung ein grauer Punkt ausgegeben. Je mehr Blockierungen an dieser Stelle stattfinden, desto heller wird der Punkt. Diese Punkte können durch das Drücken der <B>-Taste anzeigt bzw. ausgeblendet werden.

5 Konfiguration der Simulation

Dieses Kapitel demonstriert die Abbildung eines Linienverlaufs in B-u-S-Sim und die vorbereitenden Schritte, die zur Durchführung eines Simulationslaufs notwendig sind. Anhand der DVB-Straßenbahn-

linie 13 wird die Konstruktion des Linienverlaufs erklärt und die Spezifikation der Fahrzeuge und deren Aktivitäten beschrieben.

5.1 Basis-Objekte in B-u-S-Sim

Objekt	Beschreibung
BUSSIM_POINT	Repräsentation einer Haltestelle (inkl. Endhaltestellen oder Betriebshalte / Depots)
BUSSIM_ARC	Eine Verbindung von zwei benachbarten Haltestellen
BUSSIM_TOI	geometrische Objekte verschiedener Farbe, die wichtige geographische Referenzen in der gezeigten Simulationskarte darstellen.
BUSSIM_NETWORK	Gruppierung der gespeicherten Haltestellen und Verbindungen sowie anderer die gesamte Simulation betreffender Eigenschaften und Informationen
BUSSIM_LINE	Ein Streckenverlauf (Linienverlauf) von einer Endhaltestelle zu einer anderen Endhaltestelle
BUSSIM_VEHICLE	Ein Fahrzeug
BUSSIM_EVENT	Ein Ereignis, dass während der Simulation auftritt
BUSSIM_BUNCHPOINT	Ein Ort im Netzwerk, an dem ein Fahrzeug durch ein anderes blockiert wird
BUSSIM_DUTY	Ein Fahrauftrag für ein Fahrzeug (umfasst das Abfahren eines Linienverlaufs)
BUSSIM_ROTATION	Ein Umlaufplan, d.h. eine Liste von nacheinander durch ein spezifiziertes Fahrzeug abzufahrenden Fahraufträgen

Tabelle 2: B-u-S-Sim -Objekte

B-u-S-Sim stellt verschiedene C++ - Datenobjekte zur Verfügung, um effizient komplexe Personenverkehrsnetze darzustellen, die einen Linienverkehr anbieten. Diese Objekte sind in Tabelle 2 zusammengestellt. Zu jedem Objekt ***OBJECT*** existiert im `Code::Blocks` -Projekt eine Header-Datei `BUSSIM_OBJECT.h` sowie eine Quellcode-Datei `BUSSIM_OBJECT.cpp`. Typischerweise muss sich der Nutzer nicht mit den ***BUSSIM_EVENT*** - sowie den ***BUSSIM_BUNCHPOINT***-Objekten auseinandersetzen, da diese für die Konfiguration einer Simulation irrelevant sind.
Bei der Einrichtung einer Simulation müssen jedoch die übrigen fünf Objekte spezifiziert werden. Da sie teilweise voneinander abhängig sind, muss hierbei eine vorgegebene Reihenfolge berücksichtigt werden.

1. Spezifikation wichtiger geographischer Referenzobjekte durch ***BUSSIM_TOI***-OBjekte.
2. Einrichten der Haltestellen, d.h. Spezifikationen der ***BUSSIM_POINT***-Objekte.
3. Einrichten der Streckenabschnitte, d.h. Spezifikationen der ***BUSSIM_ARC***-Objekte.
4. Definition der Linienverläufe, der ***BUSSIM_LINE***-Objekte.
5. Konfiguration der Fahrzeuge (***BUSSIM_VEHICLE***-Objekte) und
6. Festlegung der zugehörigen Fahrten der Fahrzeuge durch das simulierte Netzwerk („Umlaufplan“) in einem ***BUSSIM_ROTATION***-Objekt.

Im ***BUSSIM_NETWORK***-Objekt werden alle Objekte, die für die Simulation benötigt werden, strukturiert abgespeichert.

TOI-Typ	Farbe	Bedeutung bzw. Inhalt	Polygonzug
BUSSIM_TOI_WATER	blau	Fluss	offen
BUSSIM_TOI_MOUNTAIN	dunkel-grau	Geländeerhebung (Berg) oder Senke (Tal)	geschlossen
BUSSIM_TOI_HIGHWAY	grau/rot	wichtige Straße	offen
BUSSIM_TOI_FOREST	dunkel-grün	Grünfläche oder Waldstück	geschlossen
BUSSIM_TOI_LAKE	blau	Gewässer (See etc.)	geschlossen
BUSSIM_TOI_RAILTRACK	grau/weiß	Eisenbahnstrecke	offen

Tabelle 3: Mögliche Werte für das Attribut `type` einer `BUSSIM_TOI`-Instanz

5.2 Festlegung Things-of-Interests

Things-of-Interests (TOIs) beschreiben geographische Objekte, deren Einblendung im Simulationsfenster eine bessere Orientierung ermöglichen. In Abbildung 1 ist beispielsweise der Verlauf der Dresden von West/Nordwest nach Südost durchquerenden Elbe als dicke blaue Linie gezeigt. Weitere TOIs sind in der gleichen Abbildung Eisenbahnstrecken (weiß-grau), Autobahnen (grau) und ein Park (dunkelgrünes Viereck). Tabelle 3 beinhaltet eine Liste der in B-u-S-Sim verfügbaren TOI-Typen.
Jedes TOI wird durch eine Liste von zwei-dimensionalen Geo-Koordinaten (Längengrad;Breitengrad) definiert. Beginnend mit dem ersten gespeicherten Punkt wird ein Polygon-Zug durch die nachfolgenden Punkte gebildet. Je nach Typ wird der Polygonzug geschlossen (und eingefärbt) oder er bleibt offen (dann wird nur die Linie angezeigt). Für jedes TOI muss im Simulationsprogramm eine ***BUSSIM_TOI***-Instanz eingerichtet und mit Werte gefüllt werden.

Attribut	Typ	Bedeutung bzw. Inhalt
title	char[256]	verbale Beschreibung des TOIs
type	int	Spezifikation des TOI-Types gemäß Tabelle 3
width	double	die Standardbreite eines offenen Polygonzugs wird reduziert (< 1) oder vergrößert (> 1)

Tabelle 4: Attribute für das Objekt `BUSSIM_TOI`

Jedes B-u-S-Sim - Projekt beinhaltet genau eine Instanz eines ***BUSSIM_NETWORK***-Objekts. Dieses hat den Namen `NET`. Dieses Objekt enthält ein Array `TOI` mit `TOIS` Einträgen. Jedes einzelne Array-Feld enthält die Informationen zu genau einem TOI (vgl. Tabelle 4). Um die Position und die Gestalt eines TOIs abzuspeichern, werden dessen Geo-Koordinaten sukzessive mit der Methode `BUSSIM_TOI::add_coordinate_longlat(double _long, double _lat)` dem Objekt hinzugefügt. Die Festlegung der TOI-Eigenschaften (inkl. der Geokoordinaten) erfolgt in der Methode `BUSSIM_NETWORK::specify_tois(void)`.
Wir betrachten die Situation, dass für die Elbe als `BUSSIM_TOI_WATER`-Objekt-Instanz als ersten TOI spezifzieren wollen. Somit müssen wir die Attribute für die Objekt-Instanz `NET.TOI[0]` festlegen.
Zunächst wird das Attribut `title` gesetzt: `strcpy(this->TOI[0].title,"River Elbe");`.
Anschließend deklariert `this->TOI[0].type = BUSSIM_TOI_WATER;` das TOI `TOI[0]` als Fluss-Objekt.
Nachdem die Grunddaten dieses TOI festgelegt wurden, ist die Liste der Geo-Koordinaten aufzubauen, die für eine passende graphische Darstellung des TOI benötigt werden. Diese Geo-Koordinaten können z.B. einer elektronischen Karten wie GoogleMaps oder openstreetmap entnommen werden.
Die Codierung der Geo-Koordinaten des Elbe-Verlaufs zeigt Abbildung 2.
Nach dem Einfügen der Quellcode-Fragmente initialisieren Sie das Netzwerk in der Datei `main.cpp` mit dem Befehl `class BUSSIM_NETWORK NET(1, 0, 0, 0, 0);`. Erstellen Sie das ausführ-

```
1  void BUSSIM_NETWORK::specify_tois(void)
2  {
3      // this procedure specifies the tois shown in the simulation
4
5      // this is the Elbe river crossing the city of Dresden
6      strcpy(this->TOI[0].title,"River Elbe");
7      this->TOI[0].type = BUSSIM_TOI_WATER;
8
9      this->TOI[0].add_coordinate_longlat(13.7000464,51.0679028);
10     this->TOI[0].add_coordinate_longlat(13.701763,51.0720017);
11     this->TOI[0].add_coordinate_longlat(13.7045096,51.0754532);
12     this->TOI[0].add_coordinate_longlat(13.7100028,51.0761003);
13     this->TOI[0].add_coordinate_longlat(13.7172125,51.076316);
14     this->TOI[0].add_coordinate_longlat(13.7213324,51.0724332);
15     this->TOI[0].add_coordinate_longlat(13.7261389,51.06855);
16     this->TOI[0].add_coordinate_longlat(13.7316321,51.0633719);
17     this->TOI[0].add_coordinate_longlat(13.7343787,51.0590564);
18     this->TOI[0].add_coordinate_longlat(13.7391852,51.0547405);
19     this->TOI[0].add_coordinate_longlat(13.7436484,51.0540931);
20     this->TOI[0].add_coordinate_longlat(13.7508582,51.0556038);
21     this->TOI[0].add_coordinate_longlat(13.7553214,51.0579775);
22     this->TOI[0].add_coordinate_longlat(13.7615012,51.0601354);
23     this->TOI[0].add_coordinate_longlat(13.7669943,51.0627246);
24     this->TOI[0].add_coordinate_longlat(13.7728308,51.0635877);
25     this->TOI[0].add_coordinate_longlat(13.7834738,51.0635877);
26     this->TOI[0].add_coordinate_longlat(13.7944602,51.0625089);
27     this->TOI[0].add_coordinate_longlat(13.8033865,51.0599196);
28     this->TOI[0].add_coordinate_longlat(13.8099097,51.0556038);
29 }
```

Abbildung 2: Spezifikation der Sequenz der Geo-Koordinaten, die den Verlauf der Elbe im Großraum Dresden beschreiben

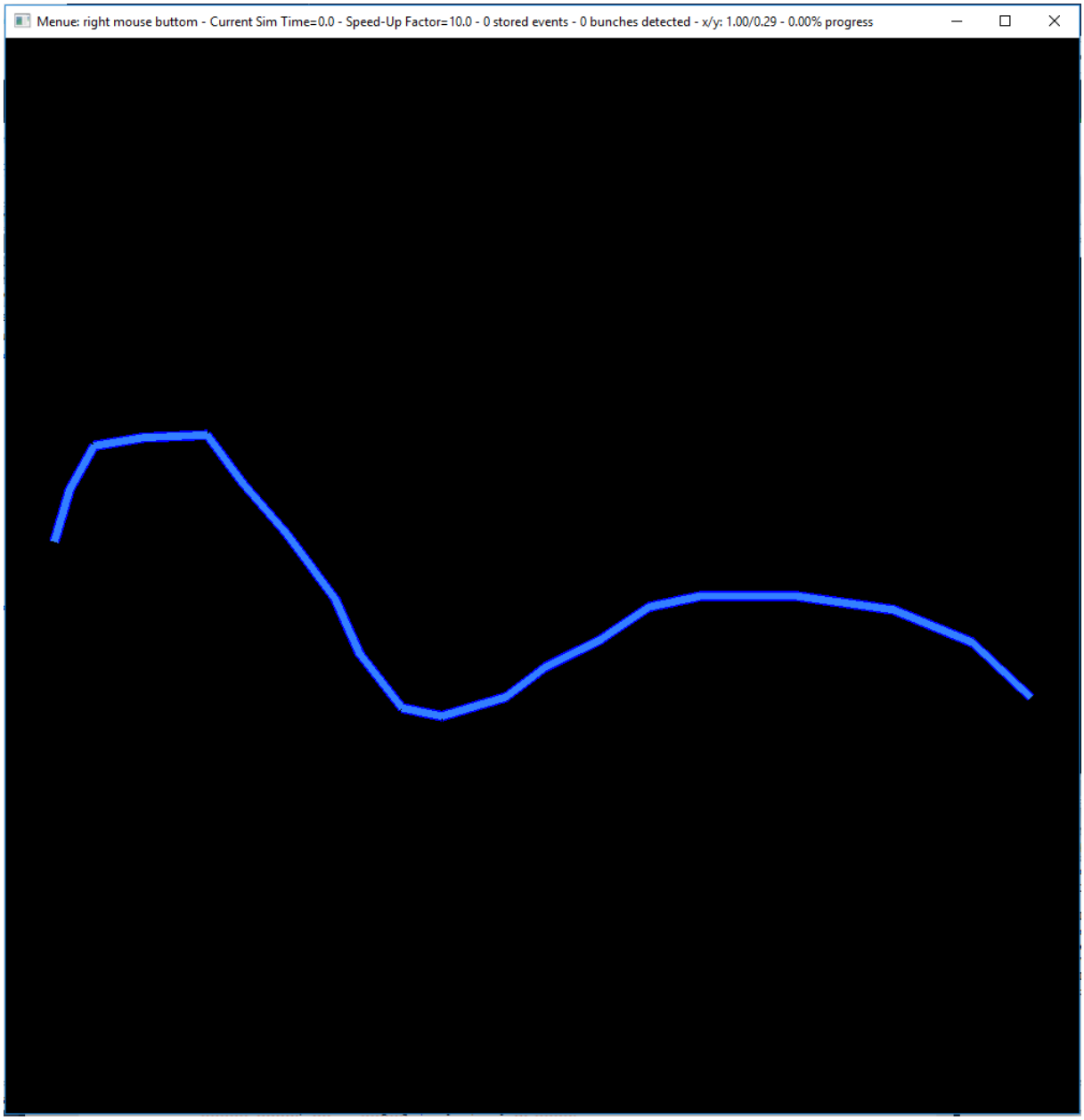

Abbildung 3: Elbe als TOI in der Simulationskarte

bare Programm in *Code::Blocks* . Starten Sie anschließend das erstellte Simulationsprogramm. Klicken Sie anschließend mit dem Mauszeiger in das Simulationsfenster. Abbildung 3 zeigt die graphische Darstellung der Elbe im Simulationsprogramm.

5.3 Einrichten der Infrastruktur

Unser Ziel ist es, als nächstes die Infrastruktur zu hinterlegen, die für die Abbildung des Streckenverlaufs der DVB-Straßenbahnlinie 13 (Prohlis↔Mickten) benötigt werden. Die Liste der Haltestellen, die von dieser Straßenbahnlinie bedient werden, ist in Anhang A zu finden.

5.3.1 Netzwerk-Objekte

Neben dem Array `TOI` zum Ablegen der TOIs enthält ein Netzwerk-Objekt auch ein Array der Länge `POINTS` mit dem Namen `PT`, wobei `POINTS` die Anzahl zu speichernder Haltestellen angibt. Somit enthält das Netzwerkobject `NET` die Knoten `PT[0]`,`PT[1]`,...,`PT[POINTS-1]`, die die Haltestellen im Netzwerk repräsentieren.

5.3.2 Haltestellen

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Knotens
world_longitude	double	Längengrad des Knotens (x-Koordinate)
world_latitude	double	Breitengrad des Knotens (y-Koordinate)
screen_x	double	x-Koordinate des Knotens am Bildschirm (wird durch B-u-S-Sim berechnet)
screen_y	double	y-Koordinate des Knotens am Bildschirm (wird durch B-u-S-Sim berechnet)
label_pos	int	gibt Richtung der Label-Ausgabe am Bildschirm an (Werte 0,1,...,8)
label_x_offset	double	horizontaler Abstand der Knotennummer vom Knoten am Bildschirm
label_y_offset	double	vertikaler Anstand der Knotennummer vom Knoten am Bildschirm
type	int	legt den Typ der Haltestelle (regulärer Halte im Linienbetrieb oder Betriebshalt) fest

Tabelle 5: Attribute für das Objekt `BUSSIM_POINT`

Die Haltestellen werden als Bestandteil des Netzwerks gespeichert. Hierzu wird die zum Netzwerk-Objekt gehörende Funktion `BUSSIM_NETWORK::specify_nodes_longlat` genutzt. Der zu ergänzende Quellcode befindet sich in der Datei `bussim_network.cpp`.
Abbildung 4 zeigt die Befehle zur Positionierung der Geo-Koordinaten $(13.7221358; 51.0339496)$ der in `PT[0]` hinterlegten Haltestelle. Damit die Abstände zwischen den Haltestellen möglichst einfach errechnet und damit alle Haltestellen sinnvoll am Bildschirm positioniert werden können, müssen alle Haltestellen in einem kartesischen Koordinatensystem angeordnet werden. Typischerweise kann für jede Haltestelle relativ einfach (z.B. über www.openstreetmap.org), die jeweilige Geo-Position bestehend aus der Angabe der Werte für den Längengrad und den Breitengrad ermittelt werden. Anschließend muss dann aus der paarweisen Differenz von Längen- bzw. Breitengrad zwischen je zwei Haltestellen-Geo-Positionen der Abstand in Kilometern bestimmt werden. Dies kann näherungsweise sehr einfach geschehen und diese Näherung ist für unsere Zwecke ausreichend. Seien $(l_1; b_1)$ sowie $(l_2; b_2)$ die Geo-Koordinaten zweier Haltestellen H_1 und H_2 ausgedrückt in Längengrad (l) und Breitengrad (b) in Grad.

```
1  void BUSSIM_NETWORK::specify_nodes_longlat(void)
2  {
3      // added in version 1.05
4      // specifies the points using geo-coordinates
5      // after coordinate specification these are transformed into (world_x;
           world_y)-pairs
6      // here you type in the node information of your network
7
8      // Spezifikation der Verkehrshalte
9      this->PT[0].world_longitude=13.7987202; this->PT[0].world_latitude
           =50.9993055;
10     this->PT[1].world_longitude=13.7978505; this->PT[1].world_latitude
           =51.0019235;
11     this->PT[2].world_longitude=13.7990222; this->PT[2].world_latitude
           =51.0052175;
12     this->PT[3].world_longitude=13.8042152; this->PT[3].world_latitude
           =51.0089377;
13     this->PT[4].world_longitude=13.8003263; this->PT[4].world_latitude
           =51.0103155;
14     this->PT[5].world_longitude=13.7952035; this->PT[5].world_latitude
           =51.0129712;
15     this->PT[6].world_longitude=13.7907414; this->PT[6].world_latitude
           =51.0134771;
16     this->PT[7].world_longitude=13.7873736; this->PT[7].world_latitude
           =51.0155091;
17     this->PT[8].world_longitude=13.7836863; this->PT[8].world_latitude
           =51.0176679;
18     this->PT[9].world_longitude=13.7803055; this->PT[9].world_latitude
           =51.0196443;
19     this->PT[10].world_longitude=13.7747082; this->PT[10].world_latitude
           =51.0225596;
20     this->PT[11].world_longitude=13.769357; this->PT[11].world_latitude
           =51.0245206;
21     this->PT[12].world_longitude=13.7650843; this->PT[12].world_latitude
           =51.0226629;
22     this->PT[13].world_longitude=13.7618549; this->PT[13].world_latitude
           =51.0248476;
23     this->PT[14].world_longitude=13.7597683; this->PT[14].world_latitude
           =51.0273837;
24     this->PT[15].world_longitude=13.7615109; this->PT[15].world_latitude
           =51.0314038;
25     this->PT[16].world_longitude=13.7579935; this->PT[16].world_latitude
           =51.0346843;
26     this->PT[17].world_longitude=13.7529649; this->PT[17].world_latitude
           =51.0365759;
27     this->PT[18].world_longitude=13.7480175; this->PT[18].world_latitude
           =51.0377002;
28     this->PT[19].world_longitude=13.7517587; this->PT[19].world_latitude
           =51.0428411;
29     this->PT[20].world_longitude=13.7553024; this->PT[20].world_latitude
           =51.0466861;
```

Abbildung 4: Spezifikation der Haltestellen

```
1       \begin{lstlisting}
2       this->PT[21].world_longitude=13.7574188; this->PT[21].world_latitude
            =51.0501163;
3       this->PT[22].world_longitude=13.7579853; this->PT[22].world_latitude
            =51.0518303;
4       this->PT[23].world_longitude=13.7576199; this->PT[23].world_latitude
            =51.054302;
5       this->PT[24].world_longitude=13.7528767; this->PT[24].world_latitude
            =51.0594439;
6       this->PT[25].world_longitude=13.7528809; this->PT[25].world_latitude
            =51.0629002;
7       this->PT[26].world_longitude=13.7535012; this->PT[26].world_latitude
            =51.0658187;
8       this->PT[27].world_longitude=13.7556933; this->PT[27].world_latitude
            =51.0698441;
9       this->PT[28].world_longitude=13.7509442; this->PT[28].world_latitude
            =51.0716791;
10      this->PT[29].world_longitude=13.7469434; this->PT[29].world_latitude
            =51.0720774;
11      this->PT[30].world_longitude=13.7406711; this->PT[30].world_latitude
            =51.0736034;
12      this->PT[31].world_longitude=13.7329574; this->PT[31].world_latitude
            =51.0766036;
13      this->PT[32].world_longitude=13.7257934; this->PT[32].world_latitude
            =51.0768443;
14      this->PT[33].world_longitude=13.7217588; this->PT[33].world_latitude
            =51.0773648;
15      this->PT[34].world_longitude=13.7176658; this->PT[34].world_latitude
            =51.0774949;
16      this->PT[35].world_longitude=13.713299; this->PT[35].world_latitude
            =51.0806;
17
18      // Spezifikation des Betriebshofs
19      this->PT[36].world_longitude=13.80377; this->PT[36].world_latitude
            =51.01093;
20      this->PT[36].type = BUSSIM_NODE_SERVICE;
21
22      // points specification ends here
23
24      // set the ID and label position
25      for(int i=0 ; i < this->POINTS ; i++)
26      {
27          this->PT[i].ID = i;
28          this->PT[i].label_pos = BUSSIM_DEFAULT_LABEL_POS;
29      }
30  }
```

Abbildung 4: Spezifikation der Haltestellen

Da in Deutschland je Grad Längengrad-Differenz eine Kilometer-Differenz von ungefähr 71km besteht, kann der horizontale Abstand zwischen H_1 und H_2 bestimmt werden durch $d_x(H_1; H_2) := 71km \cdot (l_1 - l_2)$. Da zwei Breitengrade immer einen Abstand von ca. 111km je Grad Differenz besitzen, kann der vertikale Abstand zwischen H_1 und H_2 bestimmt werden durch $d_y(H_1; H_2) := 111km \cdot (b_1 - b_2)$.
Damit B-u-S-Sim die Geo-Koordinaten in kartesische Koordinaten zur Anzeige am Bildschirm und zur Bestimmung der Abstände umrechnen kann, müssen die o.a. Umrechnungsfaktoren spezifiziert werden. Hierzu müssen in der Datei `bussim_global.h` die Konstanten `BUSSIM_LONG_DIST_KM` und `BUSSIM_LAT_DIST_KM` auf die jeweils für die dargestellte Region eingestellt werden. Standardmäßig werden die vorgenannten Werte verwendet.

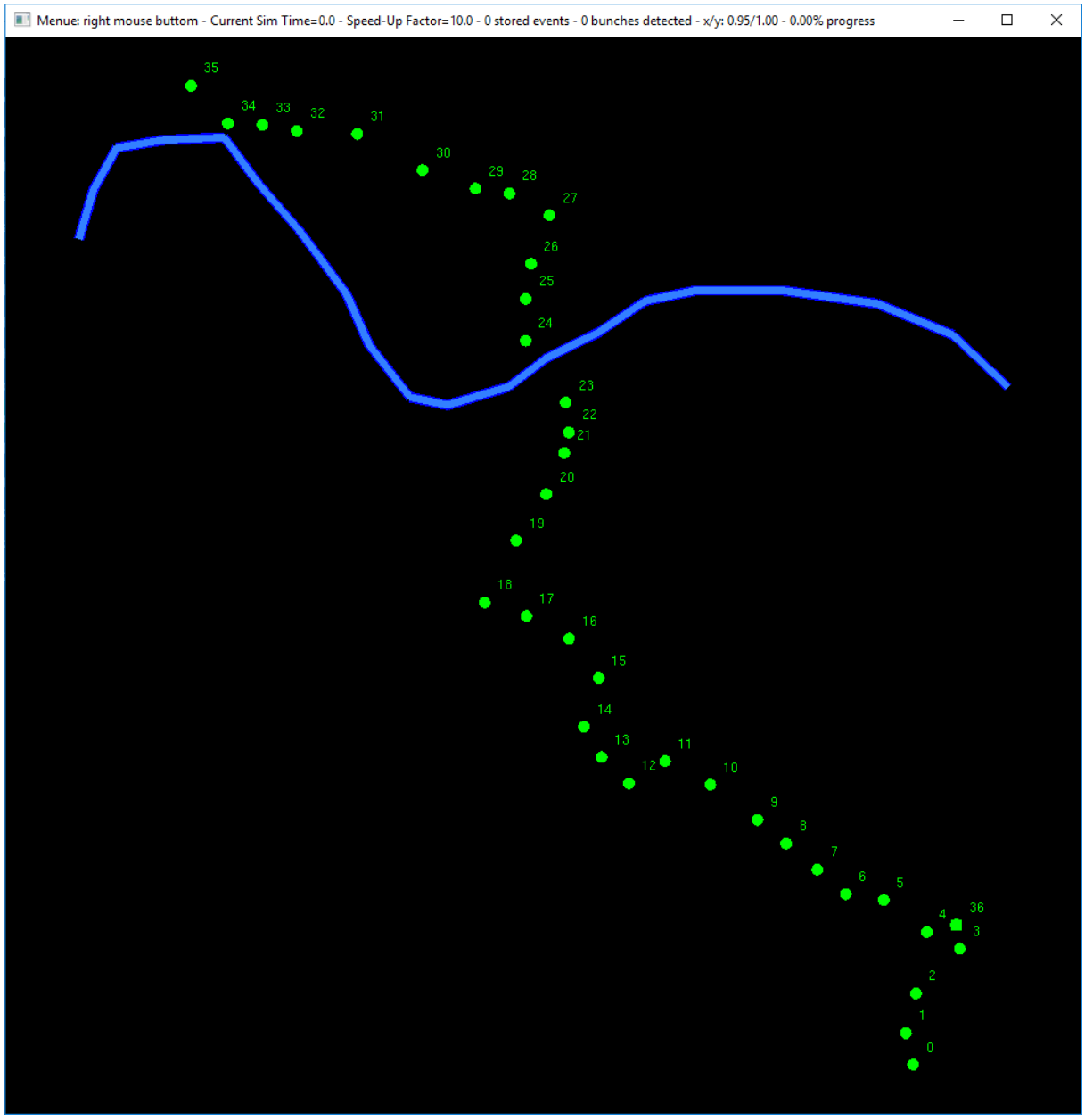

Abbildung 5: Haltestellen entlang der Streckenführung der Linie 13. Die quadratische Haltestelle (36) stellt den Betriebshalt am Betriebshof Reick dar.

Es besteht die in B-u-S-Sim Möglichkeit, reguläre, d.h. im Linienverkehr benutzte Haltestellen von anderen Haltestellen (z.B. Betriebshöfe oder Betriebshalte) zu unterscheiden. Dafür muss das Attribut `type` des ***BUSSIM_POINT***-Objekts `PT` passend gesetzt werden. Hierfür gibt es zwei Konstante Wer-

te: BUSSIM_NODE_LINE deklariert eine reguläre Haltestelle, durch BUSSIM_NODE_SERVICE wird ein Betriebshalte ausserhalb eines Linienverlauf gekennzeichnet. Standardmäßig wird dieses Attribut auf den Wert BUSSIM_NODE_LINE gesetzt.
Wir gehen nun vereinfachend davon aus, dass die gesamte Flotte der Fahrzeuge, die auf der Linie 13 verkehren sollen, im Betriebshof Dresden-Reick beheimatet ist. Daher muss jedes Fahrzeug von dort ausrücken und dorthin einrücken.
Der Betriebshof besitzt die Geo-Koordinaten (13.80377;51.01093). Wir ergänzen unsere Liste der netzwerkknoten um den 37. Knoten PT[36]. Dazu fügen wir die Zeilen

```
1 this->PT[36].world_longitude=13.80377;
2 this->PT[36].world_latitude=51.01093;
```

zur Funktion specify_nodes_longlat hinzu. Anschließend definieren wir dieses Halt als Betriebshalt durch den Befehl this->PT[36].type = BUSSIM_NODE_SERVICE;. Bevor wir das Simulationsprogramm erstellen, müssen wir den zweiten Parameter des Netzwerk-Konstruktors in der Datei main.cpp modifizieren zu , um die 37 Haltestellen (36 Verkehrshalte und ein Betriebshof) einrichten zu können.
Nun übersetzen wir das erweiterte Simulationsprogramm und starten es über die Kommandozeile mit dem Befehl bus_sum.exe. Nach dem Start des Programms klicken Sie mit der Maus in das SImulationsfenster, um diesem Fenster den Fokus zu geben. Drücken Sie nun die Tastenkombination <SHIFT>+<H>, um die Haltestellen alle einzublenden. Angezeigt wird nun das SImulationsfenster mit der schematischen Darstellung der Elbe und den grünen Kreisen, die die hinterlegten Haltestellen repräsentieren (Abbildung 5).

5.3.3 Streckenabschnitte

Sobald die Haltestellen eingerichtet sind, müssen diese durch Streckenabschnitte verbunden werden. Hierfür steht das Array ARC zur Verfügung. Abgespeichert werden können ARCS Streckenabschnitte, die jeweils in einem Objekt des Typs ***BUSSIM_ARC*** im Netzwerk-Objekt NET gespeichert werden: ARC[0],ARC[1],...,ARC[ARCS-1] lauten die Variablennamen der gespeicherten Streckenabschnitte. Da wir je zwei benachbarte Haltestellen mit einem Pfeil verbinden müssen, benötigen wir bei 36 Verkehrshalten 2·(36-1)=70 Streckenabschnitte. Zusätzlich muss aber auch der Betriebshof (Stop 36) eingebunden werden, da ja die Fahrzeuge von dort ausrücken bzw. einrücken. Hierzu müssen wir Verbindungen zwischen den Knoten 3 (Albert-Wolf-Platz) sowie und 36 sowie 4 (Trattendorfer Straße) und 36 einrichten. Somit müssen zusätzlich noch vier Streckenabschnitte für die Einbindung des Betriebshofs hinzugenommen werden, so dass das Netzwerk insgesamt 70+4=74 Streckenabschnitte umfasst.
Nachdem nun feststeht, wie viele TOIs (1), wieviele Haltestellen (37) und wieviele Streckenabschnitte (74) benötigt werden, kann das leere Netzwerk in der Datei main.ccp eingerichtet werden. Dies geschieht mit dem Befehl class BUSSIM_NETWORK NET(1,37,70,0,0);. Das erste Argument spezifiziert die Anzahl der abzuspeichernden TOIs, das zweite Argument legt die Anzahl abzuspeichernder Haltestellen (POINTS) fest und das dritte die Anzahl der benötigten Streckenabschnitte (ARCS). Die Anzahl der Linienverläufe (LINES) wird später über das vierte Argument und die Anzahl der verfügbaren Fahrzeuge (VEHICLES) über das fünfte Argument festgelegt. Wir weisen den letzten beiden Argumenten zunächst den Wert 0 zu.
Streckenabschnitte werden in Objekten des Typs ***BUSSIM_ARC*** abgelegt. Tabelle 6 zeigt die verfügbaren Attribute. Die Streckenabschnitte müssen mit einer eindeutigen, fortlaufenden und lückenlos vergebenen ID versehen werden. Die Starthaltestelle (orig_node) sowie die Zielhaltestelle (dest_node) werden durch die Angabe der zugehörigen Knoten-ID (z.B: PT[0].ID) codiert. Nachdem diese Daten durch den Benutzer in der Funktion void BUSSIM_NETWORK::specify_arcs(void) hinterlegt wurden, berechnet B-u-S-Sim die Länge des Pfeils (length_screen) und kopiert die Start- und Zielkoordinaten für den Ausgabebildschirm. Zu jedem Pfeil ist die Liste der Linienverläufe abgespeichert, die den gerade betrachteten Streckenabschnitt nutzt. Auf diese Liste kann über die Verweise first_line bzw. last_line zugegriffen werden.

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Streckenabschnitts
orig_node	int	ID der Start-Haltestelle
dest_node	int	ID der Ziel-Haltestelle
length_screen	double	Luftlinienentfernung (in km) zwischen Start- und Ziel-Haltestelle
orig_screen_x	double	x-Kooridnate auf dem Bildschirm der Start-Haltestelle
orig_screen_y	double	y-Kooridnate auf dem Bildschirm der Start-Haltestelle
dest_screen_x	double	x-Kooridnate auf dem Bildschirm der Ziel-Haltestelle
dest_screen_y	double	y-Kooridnate auf dem Bildschirm der Ziel-Haltestelle
lines_on_arc	int	Anzahl der Linienverläufe, die diesen Streckenabschnitt nutzen (diese Information wird für die Darstellung von Pfeilen benötigt)
first_line	*class BUSSIM_ARC	Verweis auf den erste abgespeicherten Linienverlauf, der diesen Pfeil nutzt
last_line	*class BUSSIM_ARC	Verweis auf den letzten abgespeicherten Linienverlauf, der diesen Pfeil nutzt

Tabelle 6: Attribute für das Objekt `BUSSIM_ARC`

$$\texttt{this}\rightarrow\texttt{ARC[0].orig\_node} = 1; \quad (1)$$

$$\texttt{this}\rightarrow\texttt{ARC[0].dest\_node} = 2; \quad (2)$$

$$\texttt{this}\rightarrow\texttt{ARC[0].ID} = 0; \quad (3)$$

Mit den Befehlen (1)-(2) wird die Verbindung (der „Pfeil“) $(1; 2)$ erzeugt und der Variablen `ARC[0]` zugewiesen. Diesem Streckenabschnitt wird abschließend der Attribut-Wert `ID`=0 zugewiesen (3). Die übrigen 73 ***ARC***-Variablen `this`→`ARC[1]`,...,`this`→`ARC[69]` werden analog spezifiziert. Der Quellcode befindet sich in der Datei `bussim_network.cpp` in der Funktion `void BUSSIM_NETWORK::specify_arcs(void)` (vgl. Abbildung 6).
Initialiseren Sie das Netzwerk in der Datei `main.cpp` nach dem Einfügen des Quellcodes (vgl. `bussim_network.cpp` mit dem Befehl `class BUSSIM_NETWORK NET(1,37,74,0,0);` und erstellen Sie erneut das ausführbare Programm in *`Code::Blocks`* . Starten Sie anschließend das Programm erneut und drücken Sie die Taste <i>. Nun werden die hinterlegten Streckenabschnitte angezeigt (Abbildung 7). Ein Drücken der Tastenkombination `UMSCHALT+<S>` blendet auch die Haltestellen ein. Es erscheint der in Abbildung 7 gezeigte Streckenverlauf bestehend aus Knoten (Haltestellen) und Pfeilen (Streckenverbindungen) in beide Richtungen des Streckenverlaufs (nicht des Linienverlaufs). Ebenso ist schematisch die Einbindung des Knotens des Betriebshofs (36) unten rechts zu erkennen. Wir können uns die Pfeile als vorhandene Schieneninfrastruktur vorstellen.

$$\text{class BUSSIM\_ NETWORK NET}(\cdot,\cdot,\cdot,\cdot,\cdot); \quad (4)$$

Damit sind alle Arbeiten an der Infrastruktur abgeschlossen. Falls weitere TOIs, Haltestellen und/oder Streckenabschnitte hinzugefügt werden sollen, so muss zunächst die Anzahl der zusätzlichen TOIs, der

```
1  void BUSSIM_NETWORK::specify_arcs(void)
2  {
3      // type in the arc specifications here
4      this->ARC[0].orig_node = 0; this->ARC[0].dest_node = 1; this->ARC[0].ID
           = 0;
5      this->ARC[1].orig_node = 1; this->ARC[1].dest_node = 2; this->ARC[1].ID
           = 1;
6      this->ARC[2].orig_node = 2; this->ARC[2].dest_node = 3; this->ARC[2].ID
           = 2;
7      this->ARC[3].orig_node = 3; this->ARC[3].dest_node = 4; this->ARC[3].ID
           = 3;
8      this->ARC[4].orig_node = 4; this->ARC[4].dest_node = 5; this->ARC[4].ID
           = 4;
9      this->ARC[5].orig_node = 5; this->ARC[5].dest_node = 6; this->ARC[5].ID
           = 5;
10     this->ARC[6].orig_node = 6; this->ARC[6].dest_node = 7; this->ARC[6].ID
           = 6;
11     this->ARC[7].orig_node = 7; this->ARC[7].dest_node = 8; this->ARC[7].ID
           = 7;
12     this->ARC[8].orig_node = 8; this->ARC[8].dest_node = 9; this->ARC[8].ID
           = 8;
13     this->ARC[9].orig_node = 9; this->ARC[9].dest_node = 10; this->ARC[9].
          ID = 9;
14     this->ARC[10].orig_node = 10; this->ARC[10].dest_node = 11; this->ARC
           [10].ID = 10;
15     this->ARC[11].orig_node = 11; this->ARC[11].dest_node = 12; this->ARC
           [11].ID = 11;
16     this->ARC[12].orig_node = 12; this->ARC[12].dest_node = 13; this->ARC
           [12].ID = 12;
17     this->ARC[13].orig_node = 13; this->ARC[13].dest_node = 14; this->ARC
           [13].ID = 13;
18     this->ARC[14].orig_node = 14; this->ARC[14].dest_node = 15; this->ARC
           [14].ID = 14;
19     this->ARC[15].orig_node = 15; this->ARC[15].dest_node = 16; this->ARC
           [15].ID = 15;
20     this->ARC[16].orig_node = 16; this->ARC[16].dest_node = 17; this->ARC
           [16].ID = 16;
21     this->ARC[17].orig_node = 17; this->ARC[17].dest_node = 18; this->ARC
           [17].ID = 17;
22     this->ARC[18].orig_node = 18; this->ARC[18].dest_node = 19; this->ARC
           [18].ID = 18;
23     this->ARC[19].orig_node = 19; this->ARC[19].dest_node = 20; this->ARC
           [19].ID = 19;
24     this->ARC[20].orig_node = 20; this->ARC[20].dest_node = 21; this->ARC
           [20].ID = 20;
25     this->ARC[21].orig_node = 21; this->ARC[21].dest_node = 22; this->ARC
           [21].ID = 21;
26     this->ARC[22].orig_node = 22; this->ARC[22].dest_node = 23; this->ARC
           [22].ID = 22;
27     this->ARC[23].orig_node = 23; this->ARC[23].dest_node = 24; this->ARC
           [23].ID = 23;
28     this->ARC[24].orig_node = 24; this->ARC[24].dest_node = 25; this->ARC
           [24].ID = 24;
```

Abbildung 6: Quellcode zur Spezifikation der Streckenabschnitte

```
29      this->ARC[25].orig_node = 25; this->ARC[25].dest_node = 26; this->ARC
            [25].ID = 25;
30      this->ARC[26].orig_node = 26; this->ARC[26].dest_node = 27; this->ARC
            [26].ID = 26;
31      this->ARC[27].orig_node = 27; this->ARC[27].dest_node = 28; this->ARC
            [27].ID = 27;
32      this->ARC[28].orig_node = 28; this->ARC[28].dest_node = 29; this->ARC
            [28].ID = 28;
33      this->ARC[29].orig_node = 29; this->ARC[29].dest_node = 30; this->ARC
            [29].ID = 29;
34      this->ARC[30].orig_node = 30; this->ARC[30].dest_node = 31; this->ARC
            [30].ID = 30;
35      this->ARC[31].orig_node = 31; this->ARC[31].dest_node = 32; this->ARC
            [31].ID = 31;
36      this->ARC[32].orig_node = 32; this->ARC[32].dest_node = 33; this->ARC
            [32].ID = 32;
37      this->ARC[33].orig_node = 33; this->ARC[33].dest_node = 34; this->ARC
            [33].ID = 33;
38      this->ARC[34].orig_node = 34; this->ARC[34].dest_node = 35; this->ARC
            [34].ID = 34;
39
40      this->ARC[35].orig_node = 35; this->ARC[35].dest_node = 34; this->ARC
            [35].ID = 35;
41      this->ARC[36].orig_node = 34; this->ARC[36].dest_node = 33; this->ARC
            [36].ID = 36;
42      this->ARC[37].orig_node = 33; this->ARC[37].dest_node = 32; this->ARC
            [37].ID = 37;
43      this->ARC[38].orig_node = 32; this->ARC[38].dest_node = 31; this->ARC
            [38].ID = 38;
44      this->ARC[39].orig_node = 31; this->ARC[39].dest_node = 30; this->ARC
            [39].ID = 39;
45      this->ARC[40].orig_node = 30; this->ARC[40].dest_node = 29; this->ARC
            [40].ID = 40;
46      this->ARC[41].orig_node = 29; this->ARC[41].dest_node = 28; this->ARC
            [41].ID = 41;
47      this->ARC[42].orig_node = 28; this->ARC[42].dest_node = 27; this->ARC
            [42].ID = 42;
48      this->ARC[43].orig_node = 27; this->ARC[43].dest_node = 26; this->ARC
            [43].ID = 43;
49      this->ARC[44].orig_node = 26; this->ARC[44].dest_node = 25; this->ARC
            [44].ID = 44;
50      this->ARC[45].orig_node = 25; this->ARC[45].dest_node = 24; this->ARC
            [45].ID = 45;
51      this->ARC[46].orig_node = 24; this->ARC[46].dest_node = 23; this->ARC
            [46].ID = 46;
52      this->ARC[47].orig_node = 23; this->ARC[47].dest_node = 22; this->ARC
            [47].ID = 47;
53      this->ARC[48].orig_node = 22; this->ARC[48].dest_node = 21; this->ARC
            [48].ID = 48;
54      this->ARC[49].orig_node = 21; this->ARC[49].dest_node = 20; this->ARC
            [49].ID = 49;
55      this->ARC[50].orig_node = 20; this->ARC[50].dest_node = 19; this->ARC
            [50].ID = 50;
56      this->ARC[51].orig_node = 19; this->ARC[51].dest_node = 18; this->ARC
            [51].ID = 51;
```

Abbildung 6: Quellcode zur Spezifikation der Streckenabschnitte

```
57      this->ARC[52].orig_node = 18; this->ARC[52].dest_node = 17; this->ARC
            [52].ID = 52;
58      this->ARC[53].orig_node = 17; this->ARC[53].dest_node = 16; this->ARC
            [53].ID = 53;
59      this->ARC[54].orig_node = 16; this->ARC[54].dest_node = 15; this->ARC
            [54].ID = 54;
60      this->ARC[55].orig_node = 15; this->ARC[55].dest_node = 14; this->ARC
            [55].ID = 55;
61      this->ARC[56].orig_node = 14; this->ARC[56].dest_node = 13; this->ARC
            [56].ID = 56;
62      this->ARC[57].orig_node = 13; this->ARC[57].dest_node = 12; this->ARC
            [57].ID = 57;
63      this->ARC[58].orig_node = 12; this->ARC[58].dest_node = 11; this->ARC
            [58].ID = 58;
64      this->ARC[59].orig_node = 11; this->ARC[59].dest_node = 10; this->ARC
            [59].ID = 59;
65      this->ARC[60].orig_node = 10; this->ARC[60].dest_node = 9; this->ARC
            [60].ID = 60;
66      this->ARC[61].orig_node = 9; this->ARC[61].dest_node = 8; this->ARC
            [61].ID = 61;
67      this->ARC[62].orig_node = 8; this->ARC[62].dest_node = 7; this->ARC
            [62].ID = 62;
68      this->ARC[63].orig_node = 7; this->ARC[63].dest_node = 6; this->ARC
            [63].ID = 63;
69      this->ARC[64].orig_node = 6; this->ARC[64].dest_node = 5; this->ARC
            [64].ID = 64;
70      this->ARC[65].orig_node = 5; this->ARC[65].dest_node = 4; this->ARC
            [65].ID = 65;
71      this->ARC[66].orig_node = 4; this->ARC[66].dest_node = 3; this->ARC
            [66].ID = 66;
72      this->ARC[67].orig_node = 3; this->ARC[67].dest_node = 2; this->ARC
            [67].ID = 67;
73      this->ARC[68].orig_node = 2; this->ARC[68].dest_node = 1; this->ARC
            [68].ID = 68;
74      this->ARC[69].orig_node = 1; this->ARC[69].dest_node = 0; this->ARC
            [69].ID = 69;
75
76      // Einbindung des Betriebshofs
77      this->ARC[70].orig_node = 36; this->ARC[70].dest_node = 3; this->ARC
            [70].ID = 70;
78      this->ARC[71].orig_node = 3; this->ARC[71].dest_node = 36; this->ARC
            [71].ID = 71;
79      this->ARC[72].orig_node = 36; this->ARC[72].dest_node = 4; this->ARC
            [72].ID = 72;
80      this->ARC[73].orig_node = 4; this->ARC[73].dest_node = 36; this->ARC
            [73].ID = 73;
81
82      // arc specification ends here. we continue and calculate the length of
             the arcs
83      for(int a=0 ; a < this->ARCS ; a++)
84          this->ARC[a].update_length();
85  }
```

Abbildung 6: Quellcode zur Spezifikation der Streckenabschnitte

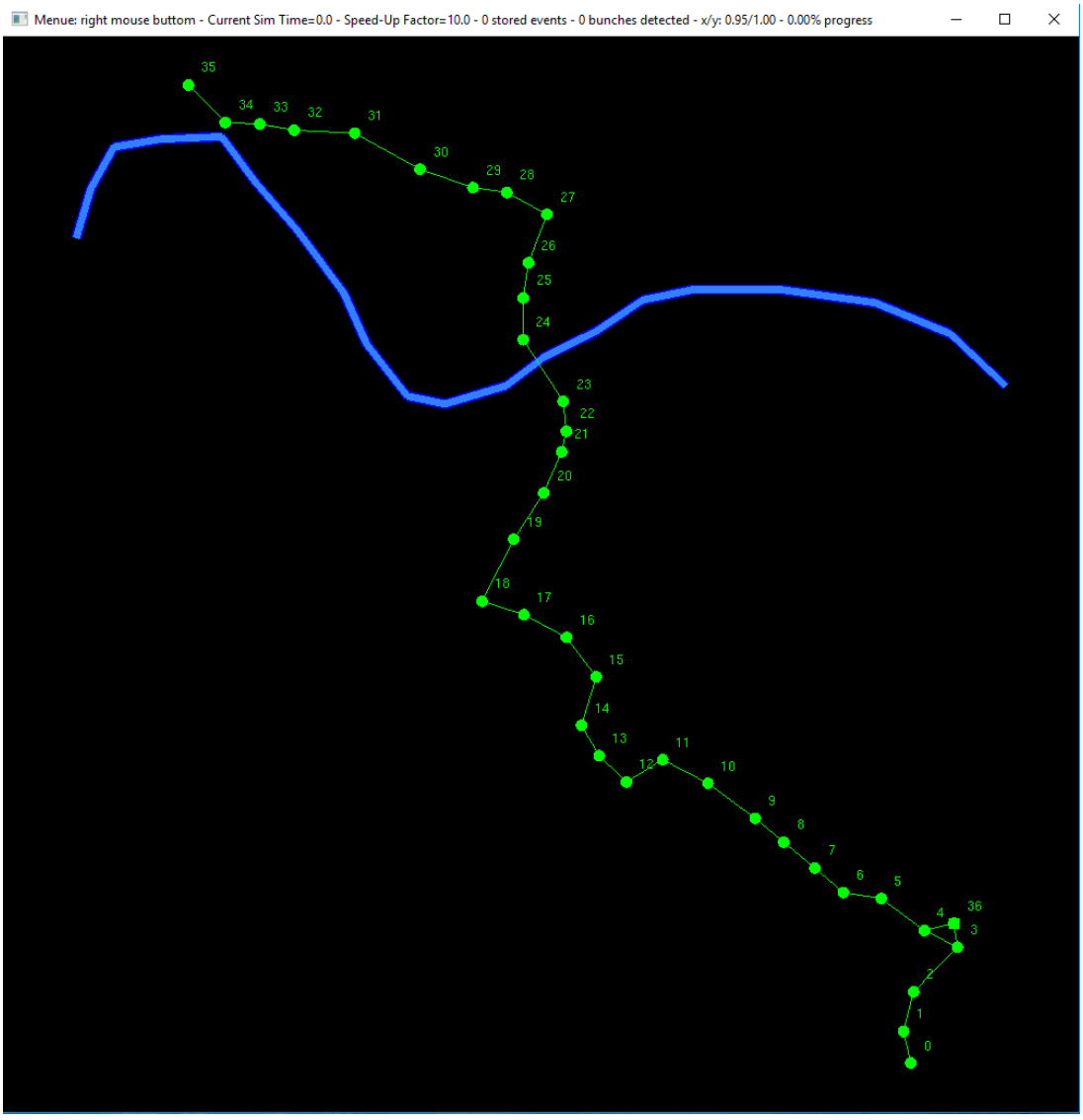

Abbildung 7: Infrastruktur der Linie 13 inkl. Anbindung an das Depot in Reick

Haltestellen sowie der zusätzlichen Streckenabschnitte ermittelt werden.
Anschließend sind in der Datei `main.cpp` wiederum die Argumente des Netzwerk-Konstruktors (4) anzupassen und in den Funktionen `BUSSIM_NETWORK::specify_nodes(void).` bzw `BUSSIM_NETWORK::specify_arcs(void)` die Haltestellenliste bzw. die Streckenabschnittsliste zu ergänzen. Ganz wichtig ist es, dass in diesen beiden Listen keine Duplikate auftreten. Wird beispielsweise ein Streckenabschnitt (Pfeil) von mehreren Linienverläufen genutzt, so darf er nur einmal als ***BUSSIM_ARC***-Objekt `ARC` angelegt sein. Genauso darf eine Haltestelle, die von mehreren Linien angefahren wird, nur einmal in der Liste `PT` auftauchen.

5.4 Einrichten von Linienverläufen

Wir haben bisher die Landschaft und die Infrastruktur hinterlegt, in der Straßenbahnen fahren können. Im nächsten Schritt müssen wir nun den Linienverlauf (der Straßenbahnlinie 13) innerhalb der konfigurierten Infrastruktur festlegen. In B-u-S-Sim wird ein Linienverlauf in einem Datenobjekt des Typs ***BUSSIM_LINE*** gespeichert. Wichtig ist, dass hierbei im Gegensatz zum Alltags-Sprachgebrauch, ein Linienverlauf lediglich von einem Knoten i^{start} (Starthaltestelle) zu einem anderen Knoten i^{ziel} (Endhaltestelle) führt und diese beiden Knoten durch eine endliche Sequenz von auf aufeinanderfolgenden Streckenabschnitten (dargestellt durch `ARC[i]`-Variablen) verbunden sind. Der Pfeil $(k; l)$ folgt dem Pfeil $(i; j)$ genau dann, wenn $k = j$ ist. Um in B-u-S-Sim nun einen bidirektionalen Linienverlauf zu realisieren, müssen wir ein zweites Linienverlaufs-Objekt des Typs ***BUSSIM_LINE*** nutzen, in dem wir die endliche Sequenz der Pfeile abspeichern, die Knoten i^{ziel} mit i^{start} verbindet.
Im Fall der Linie 13 müssen wir somit zunächst das Linienverlaufs-Objekt `this→LINE[0]` erzeugen und dort die Sequenz der Objekte `this→ARC[0]`, `this→ARC[1]`,...,`this→LINE[35]` als Linienverlauf hinterlegen. Das Linienverlaufs-Objekt `this→LINE[0]` stellt somit den Linienverlauf von „Prohlis Gleisschleife“ nach „Mickten“ dar.
Für den Linienverlauf der Linie 13 von „Mickten“ nach „Prohlis Gleisschleife“ nutzen wir ein zweites Linienverlaufs-Objekt `this→LINE[1]` und fügen diesem nacheinander die Sequenz der 35 Streckenabschnitts-Objekte `this→ARC[36]`, `this→ARC[37]`,..., `this→ LINE[69]` hinzu. Mit diesen beiden Linienverlaufs-Objekten `this→LINE[0]` und Linienverlaufs-Objekt `this→LINE[1]` können wir nun den typischen Pendelverkehr auf einer Straßenbahn-Linie zwischen einem gegebenen Paar von Wendepunkten darstellen.
Die vorhandenen Attribute eines ***BUSSIM_LINE***-Objekts sind in Tabelle 7 zusammengefasst und erläutert. In der Funktion `BUSSIM_NETWORK::specify_lines()` muss nun für jedes Linienverlaufsobjekt die Konfiguration vorgenommen und die Sequenz der Streckenabschnitte hinterlegt werden.
Die Abbildung 7 zeigt den Quellcode zur Spezifikation der beiden Richtungs-Linienverläufe der Straßenbahnlinie 13 in Dresden. Zunächst wird der Linienverlauf `LINE[0]` (Prohlis nach Mickten) konfiguriert (6). Die eindeutige ID-Nummer des Linienverlaufs wird auf den Wert 0 gesetzt. Wir werden das Bündel der beiden Richtungs-Linienverläufe der LInie 13 dadurch gruppieren, dass wir für beide den Wert des Attribute `SERVICE` auf 0 setzen (zweiter Eingabewert). Anschließend wird die Anzeigefarbe für diesen Linienverlauf auf gelb gesetzt (3.-5. Eingabewert). Der 6. Eingabewert stellt eine verbale Beschreibung des Linienverlaufs dar. In den Zeilen 7-41 werden sukzessive die 35 nacheinander zu durchfahrenen Streckenabschnitte in der vorgegebenen Reihenfolge dem Linienverlauf `LINE[0]` hinzugefügt.
In den Zeilen 43 - 79 werden identische Konfigurationsschritte für den Linienverlauf `LINE[1]` durchgeführt. Zu beachten ist, dass hierfür der eindeutige ID-Wert in Zeile 43 auf 1 gesetzt wird.
Es ist sinnvoll, für die beiden Linienverläufe, die eine Rundreise bilden, jeweils die gleiche Farbe zu verwenden.

$$class\ BUSSIM\_NETWORK\ NET(1, 37, 74, 2, 0); \tag{5}$$

In der Datei `main.cpp` muss nun noch das dritte Argument des Netzwerk-Konstruktors auf die Anzahl der insgesamt vorhandenen Linien gesetzt werden. In unserem Fall müssen wir also den Konstruktor wie

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Linienverlaufs
SERVICE	int	frei belegbarer Wert zur Bündlung von Linienverläufen
ARCS	int	Anzahl der Streckenabschnitte, die für diese Linie gespeichert sind
USED_ARC	class BUSSIM_ARC[ARCS]	Array der ***BUSSIM_ARC***-Objekte, die zu diesem Linienverlauf gehören. Sie werden in der gespeicherten Sequenz durchfahren, d.h. zunächst `USED_ARC[0]` gefolgt von `USED_ARC[1]`, usw.
x_offset	double	horizontaler Abstand vom Infrastruktur-Pfeil bei der Ausgabe am Bildschirm, um Überlappungen auf gemeinsam genutzten Streckenabschnitten zu vermeiden
y_offset	double	horizontaler Abstand vom Infrastruktur-Pfeil bei der Ausgabe am Bildschirm, um Überlappungen auf gemeinsam genutzten Streckenabschnitten zu vermeiden
RED	double	Rot-Anteil dieses Linienverlaufs bei der Darstellung am Bildschirm (Werte zwischen 0 und 1 erlaubt)
GREEN	double	Grün-Anteil dieses Linienverlaufs bei der Darstellung am Bildschirm(Werte zwischen 0 und 1 erlaubt)
BLUE	double	Blau-Anteil dieses Linienverlaufs bei der Darstellung am Bildschirm(Werte zwischen 0 und 1 erlaubt)
LineName	char[256]	Zeichenkette zur verbalen Beschreibung der Linie, die im Simulationsfenster in der Linien-Agenda angezeigt wird
displayed	int	=1, falls der Linienverlauf angezeigt werden soll, sonst=0
type	int	=BUSSIM_REGULAR für einen regulären Linienverlauf, =BUSSIM_EXTRALINE sonst.

Tabelle 7: Attribute für das Objekt `BUSSIM_LINE`

```
1  void BUSSIM_NETWORK::specify\_lines(void)
2  {
3      // type in your line specifications here
4
5      // Linienverlaufskodierung Linie 13 von Prohlis nach Mickten
6      this->LINE[0].configure(0,13,1.0,1.0,0.0,"Linie 13: Prohlis->Mickten");
7      this->LINE[0].append_arc(this->ARC[0]);
8      this->LINE[0].append_arc(this->ARC[1]);
9      this->LINE[0].append_arc(this->ARC[2]);
10     this->LINE[0].append_arc(this->ARC[3]);
11     this->LINE[0].append_arc(this->ARC[4]);
12     this->LINE[0].append_arc(this->ARC[5]);
13     this->LINE[0].append_arc(this->ARC[6]);
14     this->LINE[0].append_arc(this->ARC[7]);
15     this->LINE[0].append_arc(this->ARC[8]);
16     this->LINE[0].append_arc(this->ARC[9]);
17     this->LINE[0].append_arc(this->ARC[10]);
18     this->LINE[0].append_arc(this->ARC[11]);
19     this->LINE[0].append_arc(this->ARC[12]);
20     this->LINE[0].append_arc(this->ARC[13]);
21     this->LINE[0].append_arc(this->ARC[14]);
22     this->LINE[0].append_arc(this->ARC[15]);
23     this->LINE[0].append_arc(this->ARC[16]);
24     this->LINE[0].append_arc(this->ARC[17]);
25     this->LINE[0].append_arc(this->ARC[18]);
26     this->LINE[0].append_arc(this->ARC[19]);
27     this->LINE[0].append_arc(this->ARC[20]);
28     this->LINE[0].append_arc(this->ARC[21]);
29     this->LINE[0].append_arc(this->ARC[22]);
30     this->LINE[0].append_arc(this->ARC[23]);
31     this->LINE[0].append_arc(this->ARC[24]);
32     this->LINE[0].append_arc(this->ARC[25]);
33     this->LINE[0].append_arc(this->ARC[26]);
34     this->LINE[0].append_arc(this->ARC[27]);
35     this->LINE[0].append_arc(this->ARC[28]);
36     this->LINE[0].append_arc(this->ARC[29]);
37     this->LINE[0].append_arc(this->ARC[30]);
38     this->LINE[0].append_arc(this->ARC[31]);
39     this->LINE[0].append_arc(this->ARC[32]);
40     this->LINE[0].append_arc(this->ARC[33]);
41     this->LINE[0].append_arc(this->ARC[34]);
```

```
42      // Linienverlaufskodierung Linie 13 von Mickten nach Prohlis
43      this->LINE[0].configure(1,13,1.0,1.0,0.0,"Linie 13: Mickten->Prohlis");
44
45      this->LINE[1].append_arc(this->ARC[35]);
46      this->LINE[1].append_arc(this->ARC[36]);
47      this->LINE[1].append_arc(this->ARC[37]);
48      this->LINE[1].append_arc(this->ARC[38]);
49      this->LINE[1].append_arc(this->ARC[39]);
50      this->LINE[1].append_arc(this->ARC[40]);
51      this->LINE[1].append_arc(this->ARC[41]);
52      this->LINE[1].append_arc(this->ARC[42]);
53      this->LINE[1].append_arc(this->ARC[43]);
54      this->LINE[1].append_arc(this->ARC[44]);
55      this->LINE[1].append_arc(this->ARC[45]);
56      this->LINE[1].append_arc(this->ARC[46]);
57      this->LINE[1].append_arc(this->ARC[47]);
58      this->LINE[1].append_arc(this->ARC[48]);
59      this->LINE[1].append_arc(this->ARC[49]);
60      this->LINE[1].append_arc(this->ARC[50]);
61      this->LINE[1].append_arc(this->ARC[51]);
62      this->LINE[1].append_arc(this->ARC[52]);
63      this->LINE[1].append_arc(this->ARC[53]);
64      this->LINE[1].append_arc(this->ARC[54]);
65      this->LINE[1].append_arc(this->ARC[55]);
66      this->LINE[1].append_arc(this->ARC[56]);
67      this->LINE[1].append_arc(this->ARC[57]);
68      this->LINE[1].append_arc(this->ARC[58]);
69      this->LINE[1].append_arc(this->ARC[59]);
70      this->LINE[1].append_arc(this->ARC[60]);
71      this->LINE[1].append_arc(this->ARC[61]);
72      this->LINE[1].append_arc(this->ARC[62]);
73      this->LINE[1].append_arc(this->ARC[63]);
74      this->LINE[1].append_arc(this->ARC[64]);
75      this->LINE[1].append_arc(this->ARC[65]);
76      this->LINE[1].append_arc(this->ARC[66]);
77      this->LINE[1].append_arc(this->ARC[67]);
78      this->LINE[1].append_arc(this->ARC[68]);
79      this->LINE[1].append_arc(this->ARC[69]);
80  }
```

Abbildung 7: Code in der Funktion `BUSSIM_NETWORK::specify_lines()` zur Linienverlaufsspezifikation

in (5) gezeigt modifizieren, damit wir die beiden Linienverläufe für die Linie 13 einrichten können.

Abbildung 8 zeigt den fertig codierten Verlauf der beiden Richtungs-Linienverläufe von Prohlis nach Mickten und von Mickten nach Prohlis dargestellt als gelbe Linienzüge. Die oben links angezeigte Agenda der Linienverläufe wird mit der Tastenkombination <SHIFT>+<A> ein- bzw. ausgeblendet.

Die beiden Richtungs-Linienverläufe einer Straßenbahn-Verbindung (hier der Linie 13) werden übereinander gedruckt. Dies ist gerade bei der Konstruktion von Linienverläufen bzw. bei der Kontrolle der Linienverläufe unübersichtlich, da nicht erkennbar ist, ob beide Linienverläufe vollständig und korrekt eingerichtet sind. Um hier erkennen zu können, ob beide Linienverläufe korrekt eingerichtet sind, können sog. Richtungspfeile, die in Fahrrichtung rechts vom eigentlichen Linienverlauf abgedruckt werden, aktiviert werden. Die Pfeilrichtung gibt die jeweilige Fahrtrichtung an (Abbildung 9). Die Aktivierung bzw. Deaktivierung erfolgt durch das Betätigen der Tastenkombination <SHIFT>+<D> oder die Auswahl des entsprechenden Menü-Eintrags. Die Richtungspfeile entlang eines Linienverlaufs können nur dann angezeigt werden, wenn der Linienverlauf auch angezeigt wird.

Die beiden eingerichteten Linienverläufe können nun Fahrzeuge nach einem vorgegebenen Fahrplan durch Fahrzeuge bedient werden. Dies klappt aber nur, wenn die Fahrzeuge zu Beginn eines Tages an der jeweiligen Start-Haltestelle bereitstehen. Im Fall der Linie 13 sind dies die Haltestellen *M*ickten (35) bzw. *G*leisschleife Prohlis (0). Da die Fahrzeuge aber alle über Nacht im Betriebshof Reick (36) stehen, müssen diese Fahrzeuge irgendwie von dort zu einer der Starthaltestelle fahren. Diese Ausrück- bzw. Einrückfahrten werden durch zusätzliche Linienverläufe beschrieben. Zur Demonstration nehmen wir an, dass die Fahrzeuge alle vom Betriebshof Reick zur Haltstelle *G*leisschleife Prohlis ausrücken und von dort auch am Ende eines Betriebstages wieder einrücken.

Betrachten wir zunächst das Ausrücken. Hierzu müssen wir einen Linienverlauf festlegen, der nacheinander die Streckenabschnitte von 36 nach 3, von 3 nach 2, von 2 nach 1 und von 1 nach 0 enthält. Dafür müssen nacheinader die Pfeile 70, 67, 68 und 69 durchfahren werden. Analog dazu können wir die Einrück-Fahrt durch die Haltestellen-Sequenz 0, 1, 2, 3, 36 mit der Pfeilsequenz 0, 1, 2, 71 beschreiben.

Der Quellcode der beiden zusäztzlichen Linienverläufe ist in Abbildung 10 zu sehen. Die Ausrückfahrt hat die ID 2, die Einrückfahrt wird durch die ID 3 gekennzeichnet. Beide Linienverläufe sind mit der Farbe Rot gekennzeichnet.

Abbildung 11 zeigt den unteren rechten Ausschnitt des Simulationsfensters. Zu erkennen sind die in rot dargestellten zusätzlichen beiden Linienverläufe.

Der Einsatz eines bestimmten Fahrzeugs kann nun wie folgt beschrieben werden. Das Fahrzeug bedient zunächst einmal die Linienverlauf 3 zum Ausrücken aus dem Betriebshof zur Gleisschleife in Prohlis. Anschließend bedient es abwechselnd die Linienverläufe 0 (von Prohlis nach Mickten) und 1 (von Mickten nach Prohlis). Ist die letzte Fahrt nach Prohlis beendet, so bedient das Fahrzeug einmalig den Linienverlauf 2, um von Prohlis in den Betriebshof einzurücken.

Analog zum vorherigen Paar von Ausrück- und Einrücklinienverläufen können wir ein weiteres Paar von Linienverläufen bestimmen, die das Ausrücken vom Betriebshof Reick nach Mickten bzw. das EInrücken von Mickten in den Betriebshof Reick ermöglichen (Abbildung 12).

Wir haben also insgesamt 6 Linienverläufe festlegen müssen, um eine realitätsnahme Abbildung von FAhrzeugbewegungen entlang der Linie 13 (in beide Richtungen) vorzubereiten. Kommen weitere Linienverläufe hinzu, so kann die Simulationsdarstellung schnell unübersichtlich werden. In B-u-S-Sim besteht nun die möglichkeit, die EIn- und Ausrückfahrten von der Bildschirmanzeige auszuschliessen. Hierzu muss der Wert des Attributes `type` eines Linienobjekts auf den Wert `BUSSIM_EXTRALINE` gesetzt werden. Dies sollte am Ende der Methode `BUSSIM_NETWORK::specify_lines` geschehen (Abbildung 13. Anschließend werden die Ein- und Ausrückfahrten in der Simulation noch berücksichtigt aber die Linienverläufe werden nicht mehr am Bildschirm eingeblendet, obwohl Fahrzeuge entlang dieser Linienverläufe geführt werden und auch angezeigt werden.

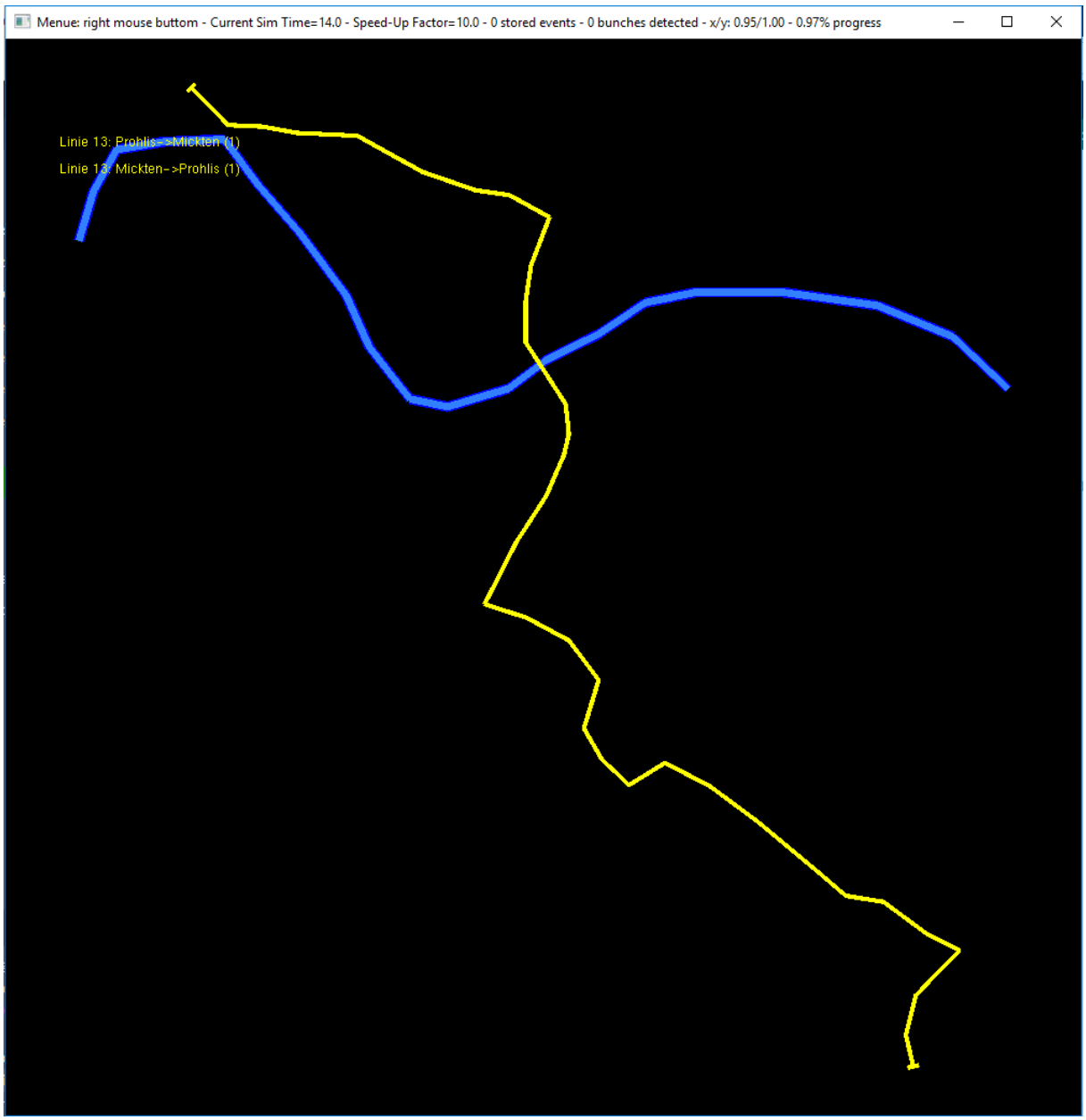

Abbildung 8: Simulationsfenster mit spezifizierten Linienverläufen der Line 13

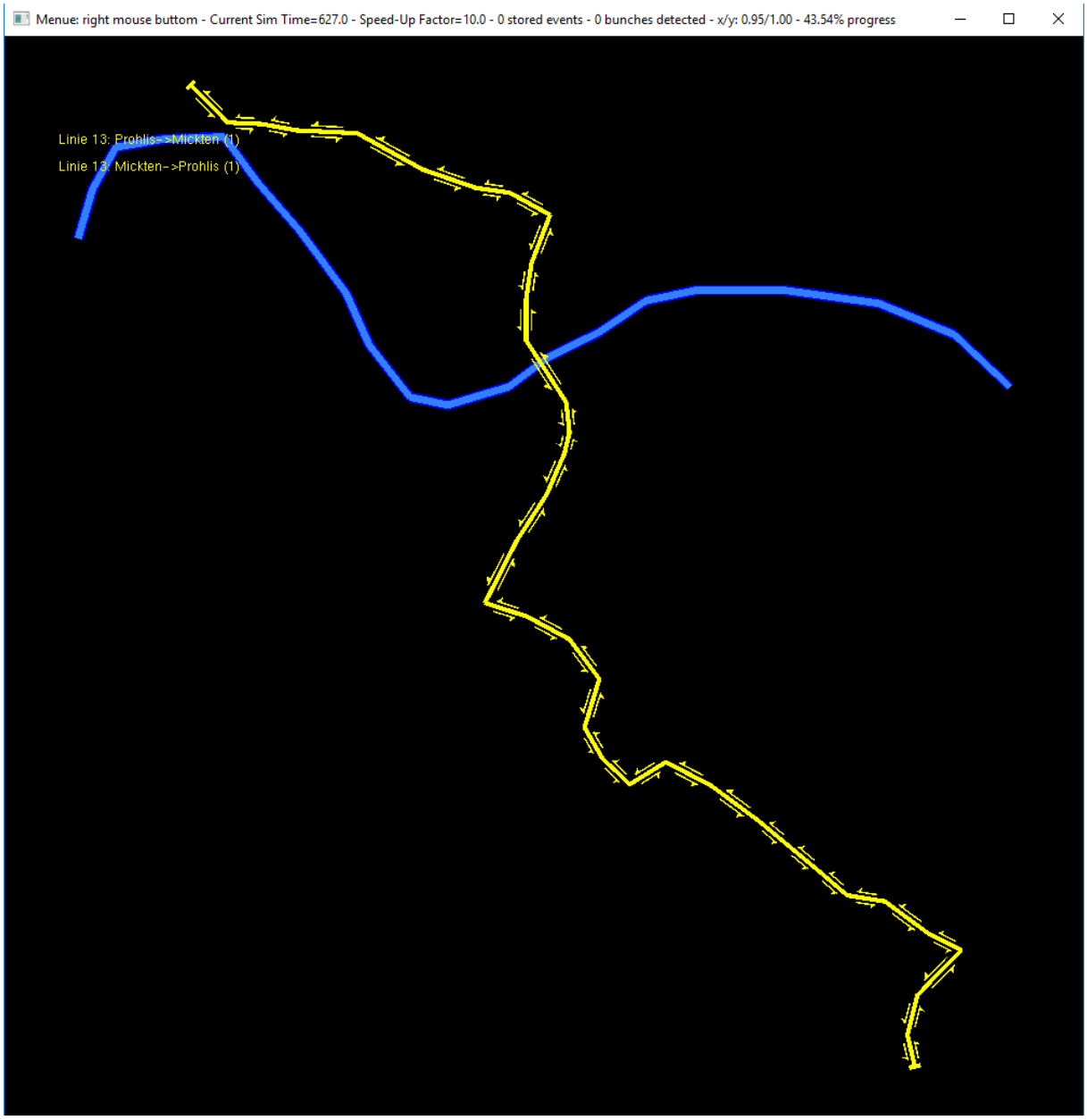

Abbildung 9: Simulationsfenster mit spezifizierten Linienverläufen der Line 13 mit aktivierten Richtungspfeilen

```
1          // Linienverlaufskodierung Linie 13 von Btf Reick nach Prohlis
2      this->LINE[2].configure(2,13,1.0,0.0,0.0,"Linie 13: Btf Reick->Prohlis"
           );
3      this->LINE[2].append_arc(this->ARC[70]);
4      this->LINE[2].append_arc(this->ARC[67]);
5      this->LINE[2].append_arc(this->ARC[68]);
6      this->LINE[2].append_arc(this->ARC[69]);
7
8      // Linienverlaufskodierung Linie 13 von Prohlis nach Btf Reick
9      this->LINE[3].configure(3,13,1.0,0.0,0.0,"Linie 13: Reick-> Btf Prohlis
           ");
10     this->LINE[3].append_arc(this->ARC[0]);
11     this->LINE[3].append_arc(this->ARC[1]);
12     this->LINE[3].append_arc(this->ARC[2]);
13     this->LINE[3].append_arc(this->ARC[71]);
```

Abbildung 10: Quellcode für die Spezifikation des Ein- und des Ausrückslinienverlaufs

5.5 Spezifikation der Fahrzeuge

Ein Fahrzeug wird in B-u-S-Sim durch ein ***BUSSIM_VEHICLE***-Objekt repräsentiert. Zunächst müssen wir uns überlegen, wieviele Fahrzeuge wir nutzen möchten und diese Zahl als fünftes und letztes Argument in den Netzwerk-Konstruktor in der Datei main.cpp eintragen. Der Name des Arrays, in dem die Fahrzeuge im ***BUSSIM_NETWORK***-Objekt gespeichert werden, ist ***VEHICLE***. Die Anzahl der gespeicherten Fahrzeuge ist im ***BUSSIM_NETWORK***-Objekt im Attribut `VEHICLES` gespeichert. Im konkreten Fall wollen wir 10 identische Straßenbahnen nutzen, die alle mit einer Durchschnittsgeschwindigkeit von 25km unterwegs sein sollen.

$$class\ BUSSIM\_NETWORK\ NET(1, 37, 74, 6, 10); \tag{6}$$

Der Netzwerk-Konstruktor muss wie in (6) angepasst werden. Damit werden die 10 Fahrzeuge als fünfter Parameter dem Netzwerk bekanntgegeben. Anschließend müssen die wesentlichen Eigenschaften (ID und Geschwindigkeit) für jedes Fahrzeug einzeln festgelegt werden. Der Ort zur Spezifikation der Fahrzeug-Eigenschaften ist die zum ***BUSSIM_NETWORK***-Objekt gehörende in der Source-Code-Datei `bussim_network.cpp` definierte Methode `BUSSIM_NETWORK::specify_vehicles()`.

Abbildung 14 zeigt den kompletten Quellcode, mit dem die 10 Fahrzeuge `this→VEHICLE[0]`, `this→VEHICLE[0]`,..., `this→VEHICLE[9]` spezifiziert werden. Der erste Parameter legt die fortlaufende ID eines Fahrzeugs fest. Der zweite Parameter spezifiziert die Geschwindigkeit des Fahrzeugs in km/h. Der dritte Parameter `this` bezeichnet das aktuelle Netzwerk, zu dem wir die Fahrzeuge hinzufügen.

Es ist möglich, sich nur ausgewählte Fahrzeuge anzusehen. Ob ein Fahrzeug in der Simulation angezeigt wird oder nicht, hängt vom Wert des Attributs `VEHICLE.VISIBLE` ab. Standardmäßig ist diesem Attribut der Wert 1 zugeordnet und das Fahrzeug wird in der Simulation angezeigt. Möchte man dieses Fahrzeug nicht anzeigen, so geschieht dies durch Zuweisung `VEHICLE.VISIBLE`=0. Die Setzung dieses Attributs geschieht in der Funktion `BUSSIM_NETWORK::set_visibility()` der Netzwerk-Klasse. Ein Beispiel ist in Abb. 15 zu finden. Es sollen nur die Fahrzeuge 6 und 8 angezeigt werden. Daher werden zunächst alle Fahrzeuge als unsichtbar deklariert (Zeilen 6-9). Anschließend werden ausschließlich die beiden gewünschten Fahrzeuge sichtbar gemacht (Zeilen 12-13).

In dieser Funktion können auch die Werte für das Attribute `LINE`displayed gesetzt werden. Mit diesem Attribut wird die Anzeige von Linienverläufen in der Simulation gesteuert.

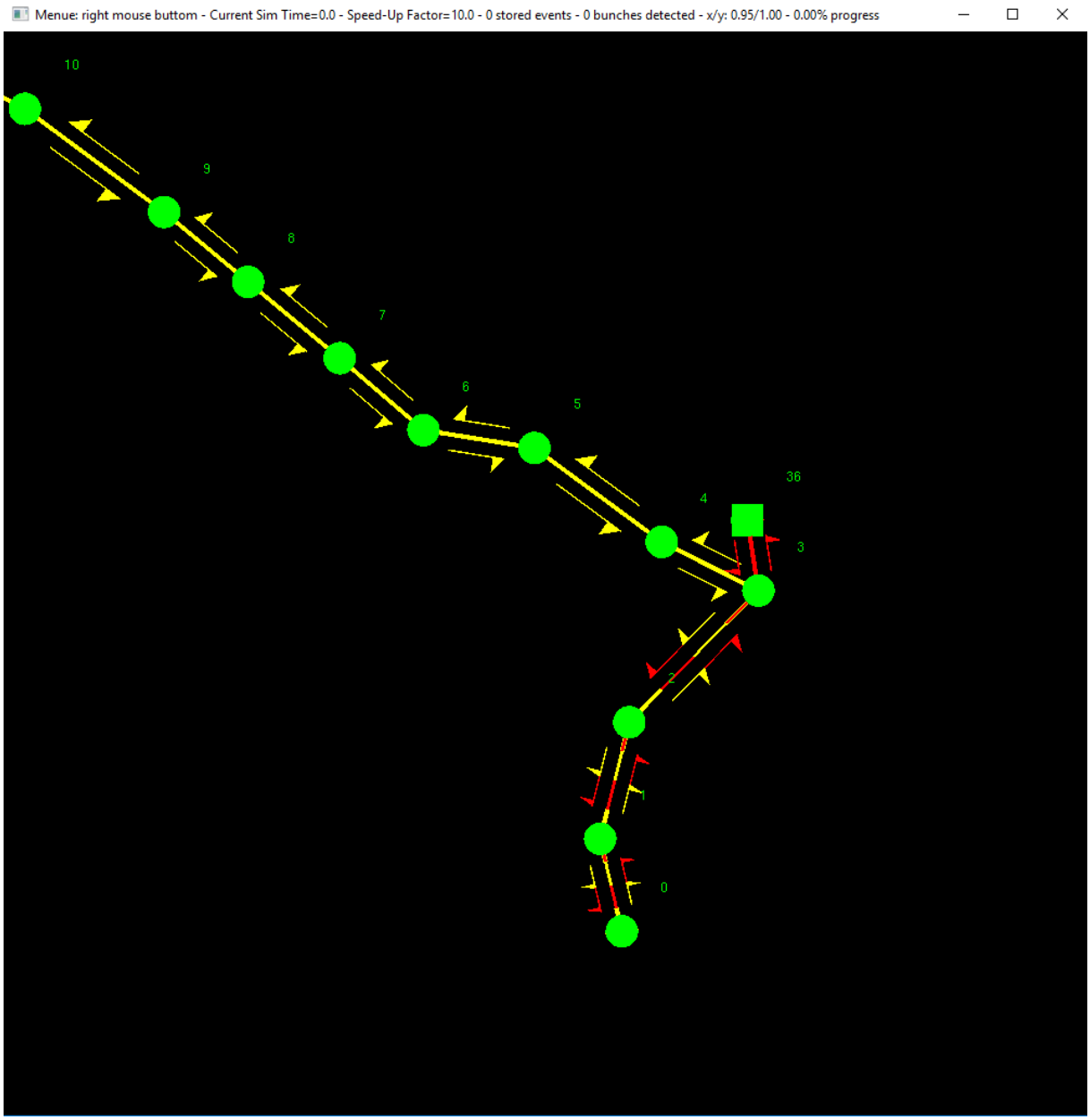

Abbildung 11: Linienverläufe zum Ein- und zum Ausbrücken der Line 13 mit aktivierten Richtungspfeilen

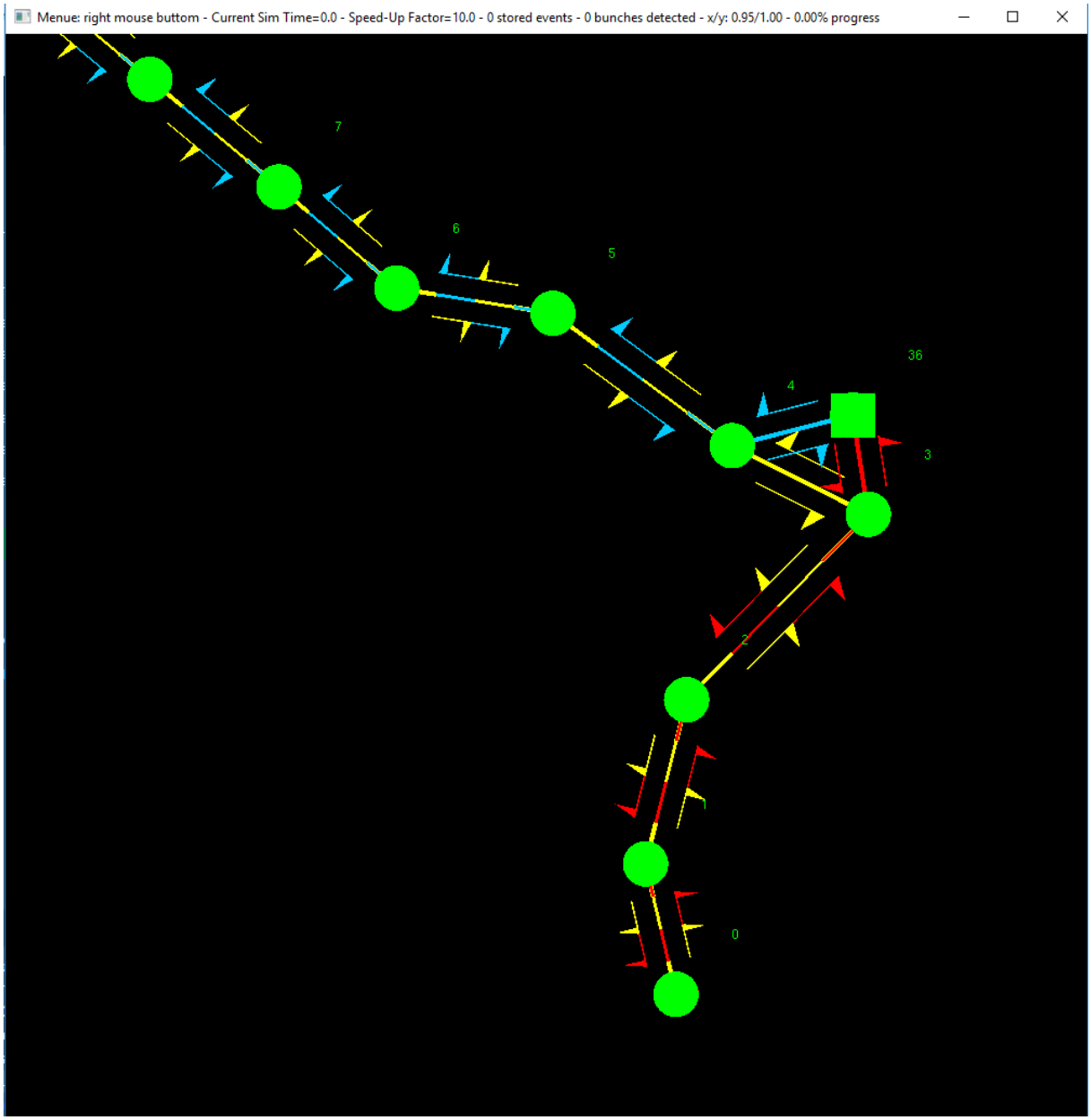

Abbildung 12: Linienverläufe zum Ein- und zum Ausbrücken der Line 13 von / nach Mickten in blauer Farbe

```
1       this->LINE[2].type = BUSSIM_EXTRALINE;
2       this->LINE[3].type = BUSSIM_EXTRALINE;
3       this->LINE[4].type = BUSSIM_EXTRALINE;
4       this->LINE[5].type = BUSSIM_EXTRALINE;
```

Abbildung 13: Deklaration der Linienverläufe 2 bis 5 als Service-Linienverläufe

```
1  void BUSSIM_NETWORK::specify_vehicles(void)
2  {
3      // specify the available vehicles here
4      this->VEHICLE[0].configure(0,25,this);
5      this->VEHICLE[1].configure(1,25,this);
6      this->VEHICLE[2].configure(2,25,this);
7      this->VEHICLE[3].configure(3,25,this);
8      this->VEHICLE[4].configure(4,25,this);
9      this->VEHICLE[5].configure(5,25,this);
10     this->VEHICLE[6].configure(6,25,this);
11     this->VEHICLE[7].configure(7,25,this);
12     this->VEHICLE[8].configure(8,25,this);
13     this->VEHICLE[9].configure(9,25,this);
14 }
```

Abbildung 14: Code in der Funktion `BUSSIM_NETWORK::specify_vehicles()` zur Einrichtung der verfügbaren 10 Fahrzeuge

```
1  void BUSSIM_NETWORK::set_visibility(void)
2  {
3      // here the visibility of the available vehicles can be specified
4
5      // make all vehicles invisible
6      for( int v=0 ; v < this->VEHICLES ; v++ )
7      {
8          this->VEHICLE[v].VISIBLE = 0;
9      }
10
11     // make selected vehicle visible again
12     this->VEHICLE[6].VISIBLE = 1;
13     this->VEHICLE[8].VISIBLE = 1;
14 }
```

Abbildung 15: Code in der Funktion `BUSSIM_NETWORK::set_visibility()` zur Auswahl der angezeigten Fahrzeuge

5.6 Fahrzeug-Umläufe festlegen

Einem Fahrzeug muss als Arbeitsauftrag das komplette Abfahren eines Linienverlaufs zugewiesen werden. Dieser Linienverlauf wird vom Fahrzeug komplett abgefahren. Ist dieser Arbeitsauftrag durchgeführt, kann entweder ein nächster hinterlegter Linienverlauf abgefahren werden oder das Fahrzeug bleibt stehen (Deaktivierung).
Damit ein Fahrzeug einen Arbeitsauftrag abarbeitet, muss dieser in einem ersten Schritt dem Fahrzeug gewiesen werden. In einem zweiten Schritt muss dieser Arbeitsauftrag gestartet werden. Die Umsetzung des erstgenannten Schrittes wird in diesem Abschnitt erklärt. Abschnitt 5.7 beschreibt, wie die Abarbeitung eines Umlaufplans zeitgesteuert gestartet wird.
Sinnigerweise sollte der erste Arbeitsauftrag einen Linienverlauf umfassen, der am Betriebshof startet und der letzte Arbeitsauftrag sollte einen Linienverlauf umfassen, der im Betriebshof endet. Start- und Zielbetriebshof können verschieden sein.
In B-u-S-Sim wird jeder Arbeitsauftrag in einem ***BUSSIM_DUTY***-Objekt gespeichert. Dort ist der abzufahrende Linienverlauf hinterlegt.
Jedem Fahrzeug kann eine Liste von hintereinander abzufahrenden Arbeitsaufträgen zugewiesen werden. Eine derartige Sequenz von Arbeitsaufträgen wird **Fahrzeug-Umlauf** genannt. In B-u-S-Sim wird ein Fahrzeugumlauf in einem ***BUSSIM_ROTATION***-Objekt gespeichert. Zu jedem Fahrzeug ist genau ein Umlauf hinterlegt. Dieser ist im Attribut `rotation` des ***BUSSIM_VEHICLE*** gepeichert. Initial ist der Umlauf leer. Mit der Funktion `BUSSIM_NETWORK::specify_rotations(VEH_ID,LINE_ID)` wird die Linienfahrt mit der ID `LINE_ID` an den vorhandenen (ggf. leeren) Umlaufplan des Fahrzeugs `VEH_ID` angehangen. Somit kann mit Hilfe dieser Funktion auch ein komplexer Umlaufplan, bestehend auf vielen nacheinander abzufahrenden verschiedenen Linienverläufen, definiert werden.
In Abb. 16 ist ein Beispiel-Quellcode für den Aufbau von Umläufplanen für die ersten vier Fahrzeuge hinterlegt. Beispielsweise ist für das Fahrzeug mit der ID 2 vorgesehen, dass es zunächst den Linienverlauf `LINE[2]` bedient (Anrücken vom Betriebshof Reick nach Prohlis) und anschließend auf den Linienverlauf `LINE[0]` wechselt, d.h. von Prohlis nach Mickten reist sowie daran anschließend von Mickten nach Prohlis zurückfährt. Abschließend muss dieses Fahrzeug den Linienverlauf `LINE[3]` abfahren, um in den Betriebshof Prohlis einzurücken. Dort bleibt es stehen und wird deaktiviert.

5.7 Fahrplan

B-u-S-Sim verwendet in der derzeitigen Version ein sehr simples Fahrplan-Konzept. Es basiert darauf, zu einem vorgegebenen Zeitpunkt ein Fahrzeug die Abarbeitung des für dieses Fahrzeug hinterlegten Umlaufplans beginnen zu lassen. Dies wird als Aktivierung eines Fahrzeugs bezeichnet.
Standardmäßig wird für jede Haltestelle die gleiche netzweite Aufenthaltszeit vorgesehen. Diese ist in der globalen Variablen `BUSSIM_STANDARD_WAITING_TIME` gespeichert. Der Wert dieser Konstante kann in der Datei `bussim_global.h` verändert werden und steht standardmäßig auf 0.75 Zeiteinheiten (Simulationsminuten). Ebenfalls ist in der Standardeinstellung vorgesehen, dass jedes Fahrzeug an jeder Haltestelle genau diese Zeit verbringt.
Soll die Aufenthaltszeit variieren, dann kann der Wert der globalen Variables `BUSSIM_DEVIATION_WAITING_TIME` auf einen Wert zwischen 0 und `BUSSIM_STANDARD_WAITING_TIME` in der Datei `bussim_global.h` erhöht werden. Die jeweilige Aufenthaltszeit wird dann gleichverteilt zufällig aus dem Intervall `[BUSSIM_STANDARD_WAITING_TIME-BUSSIM_DEVIATION_WAITING_TIME;BUSSIM_STANDARD_WAITING_TIME+BUSSIM_DEVIATION_WAITING_TIME]` gezogen.
Die Aktivierung der Fahrzeuge wird durch in der Funktion `BUSSIM_NETWORK::start_rotations()` spezifizierte Informationen gesteuert. In Abhängigkeit von der aktuellen Systemzeit wird ein Fahrzeug aktiviert. Aktivierung heißt, dass das Fahrzeug seinen ersten Linienverlauf des Umlaufplans auszuführen beginnt.
Im vorliegenden Beispiel sollen die 10 Fahrzeuge starten. Abbildung 17 zeigt den zugehörigen Quellcode an. Fahrzeug `VEHICLE[0]` startet zum Zeitpunkt 0. Zum Zeitpunkt 5 folgt das Fahrzeug `VEHICLE[1)`,

```
1  void BUSSIM_NETWORK::specify_rotations(void)
2  {
3      // this procedure defines the vehicle rotations
4
5      // Veh 0: Btf Reick -> Prohlis -> Mickten -> Prohlis -> Btf Reick
6      this->append_duty_to_vehicle_rotation(0,2);
7      this->append_duty_to_vehicle_rotation(0,0);
8      this->append_duty_to_vehicle_rotation(0,1);
9      this->append_duty_to_vehicle_rotation(0,3);
10
11     // Veh 1: Btf Reick -> Mickten -> Prohlis -> Prohlis -> Btf Reick
12     this->append_duty_to_vehicle_rotation(1,4);
13     this->append_duty_to_vehicle_rotation(1,1);
14     this->append_duty_to_vehicle_rotation(1,0);
15     this->append_duty_to_vehicle_rotation(1,5);
16
17     // Veh 2: Btf Reick -> Prohlis -> Mickten -> Prohlis -> Btf Reick
18     this->append_duty_to_vehicle_rotation(2,2);
19     this->append_duty_to_vehicle_rotation(2,0);
20     this->append_duty_to_vehicle_rotation(2,1);
21     this->append_duty_to_vehicle_rotation(2,3);
22
23     // Veh 3: Btf Reick -> Mickten -> Prohlis -> Prohlis -> Btf Reick
24     this->append_duty_to_vehicle_rotation(3,4);
25     this->append_duty_to_vehicle_rotation(3,1);
26     this->append_duty_to_vehicle_rotation(3,0);
27     this->append_duty_to_vehicle_rotation(3,5);
28 }
```

Abbildung 16: Code in der Funktion `BUSSIM_NETWORK::specify_rotations()` zum Aufbau von Umlaufplänen für Fahrzeuge

```
1  void BUSSIM_NETWORK::start_rotations(void)
2  {
3          if( (CURRENT_TIME >= 0) && (CURRENT_TIME < 5) )
4          this->VEHICLE[0].activate(this);
5      if( (CURRENT_TIME >= 5) && (CURRENT_TIME < 10) )
6          this->VEHICLE[1].activate(this);
7      if( (CURRENT_TIME >= 10) && (CURRENT_TIME < 15) )
8          this->VEHICLE[2].activate(this);
9      if( (CURRENT_TIME >= 15) && (CURRENT_TIME < 20) )
10         this->VEHICLE[3].activate(this);
11     if( (CURRENT_TIME >= 20) && (CURRENT_TIME < 25) )
12         this->VEHICLE[4].activate(this);
13     if( (CURRENT_TIME >= 25) && (CURRENT_TIME < 30) )
14         this->VEHICLE[5].activate(this);
15     if( (CURRENT_TIME >= 30) && (CURRENT_TIME < 35) )
16         this->VEHICLE[6].activate(this);
17     if( (CURRENT_TIME >= 35) && (CURRENT_TIME < 40) )
18         this->VEHICLE[7].activate(this);
19     if( (CURRENT_TIME >= 40) && (CURRENT_TIME < 45) )
20         this->VEHICLE[8].activate(this);
21     if( (CURRENT_TIME >= 45) && (CURRENT_TIME < 50) )
22         this->VEHICLE[9].activate(this);
23 }
```

Abbildung 17: Code in der Funktion `BUSSIM_NETWORK::start_rotations()` für die zeitgesteuerte Fahrzeug-Aktivierung

usw. Der <-Ausdruck wird benötigt, damit der Umlauf eines Fahrzeugs nach Abarbeitung nicht nochmals gestartet wird.
Wenn Sie nun das Simulationsprogramm übersetzen, starten und die Simulation beginnen, dann sehen Sie, wie sich die Fahrzeuge gemäß dem hinterlegten einfachen Zeitplan entlang der Linienverläufe bewegen (Abbildung 18). Im Netzplan wird die jeweils aktuelle Fahrzeugposition angezeigt. Ist das Fahrzeug auf einem Streckenabschnitt, dann wird die jeweils seit dem Verlassen des Linienverlauf-Startknotens gefahrene Strecke in Kilometern angezeigt. Falls das Fahrzeug an einer Haltestelle steht, dann wird die verbleibende Restwartezeit (in Simulationsminuten) an dieser Haltestelle angezeigt. Befindet sich das Fahrzeug auf einem Service-Linienverlauf (zum Ausrücken, Einrücken oder Repositionieren), dann wird zusätzlich ein (S) angezeigt. Der kleine schwarze Pfeil in einem Fahrzeug gibt die aktuelle Fahrtrichtung des Fahrzeugs an.
Ein Video über den Ablauf der Simulation steht unter der URL `b-u-s-sim.de` zur Verfügung.

6 Ausführen der Simulation und Auswertung - Ein typisches Beispiel

Nach dem wir unser kleines Netzwerk fertig konfiguriert haben, können wir beispielhaft zeigen, was man mit den während des Simulationslaufs gesammelten Informationen alles machen kann. In diesem Zusammenhang wollen wir die folgende Frage beantworten:
„Wie entwickelt sich die Wartezeit auf ein Fahrzeug an der Haltestelle 10, das entlang des Linienverlaufs 0 unterwegs ist über den Simulationstag, wenn die Aufenthaltszeiten der Fahrzeuge an den Haltestellen stochastisch ist (gleichverteilt im Interval von 0.25 bis 1.25 Simulationsminuten)?“
Wir starten zunächst das Simulationsprogramm mit dem Befehl `bus_sim.exe > result.txt`. Nach dem Ende des Simulationslaufs wird dann die Eventliste in die Textdatei `result.txt` geschrieben. Diese Datei kann z.B. in Excel geöffnet werden. Nun werden alle Zeilen herausgefiltert, die in Spalte 2 den Wert 1 enthalten (Departure Event), in der 6. Spalte den Wert 0 (Linienverlauf 0) und in der

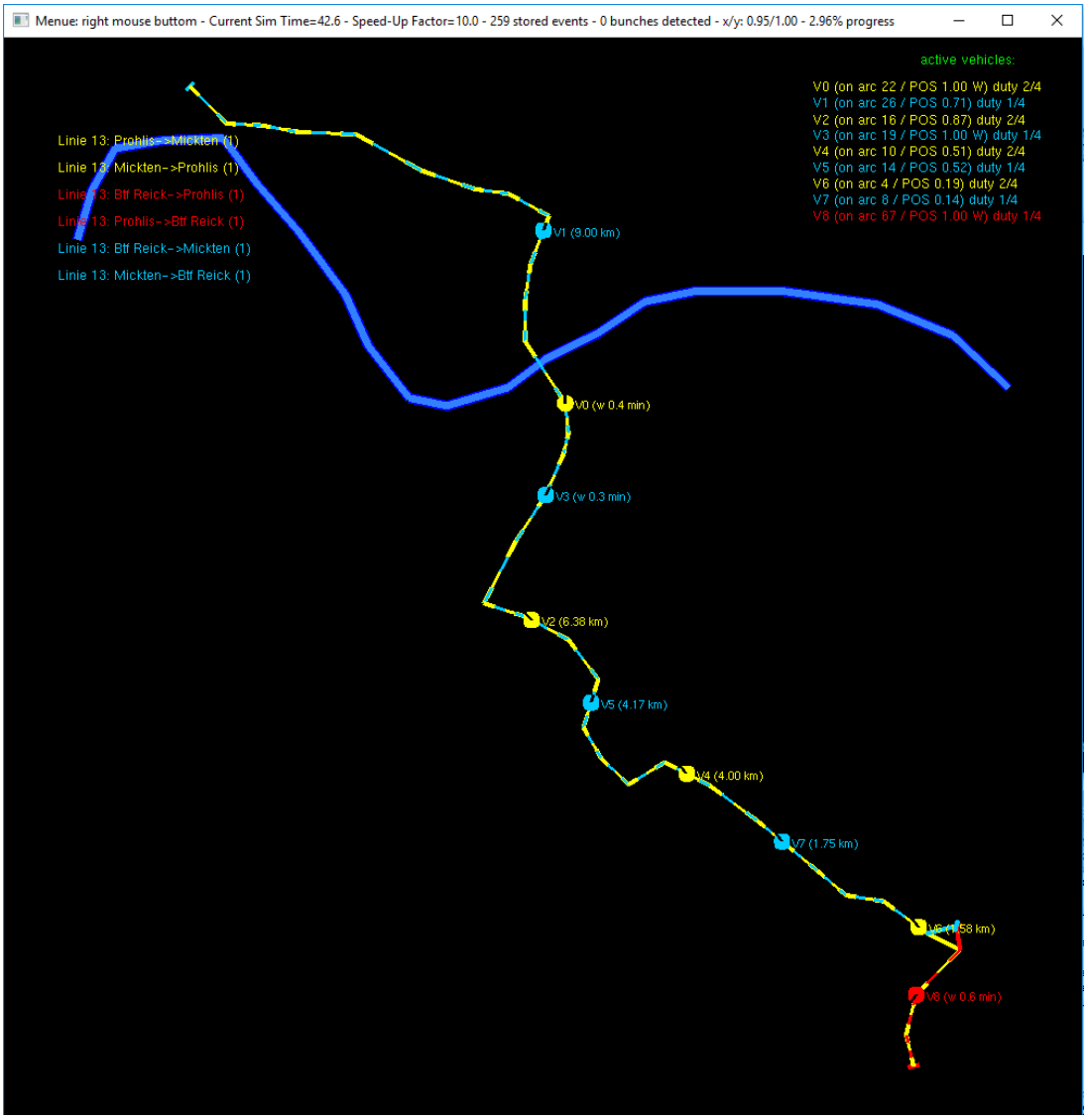

Abbildung 18: Streckenverlauf der Linie 13 mit aktivierten Fahrzeugen

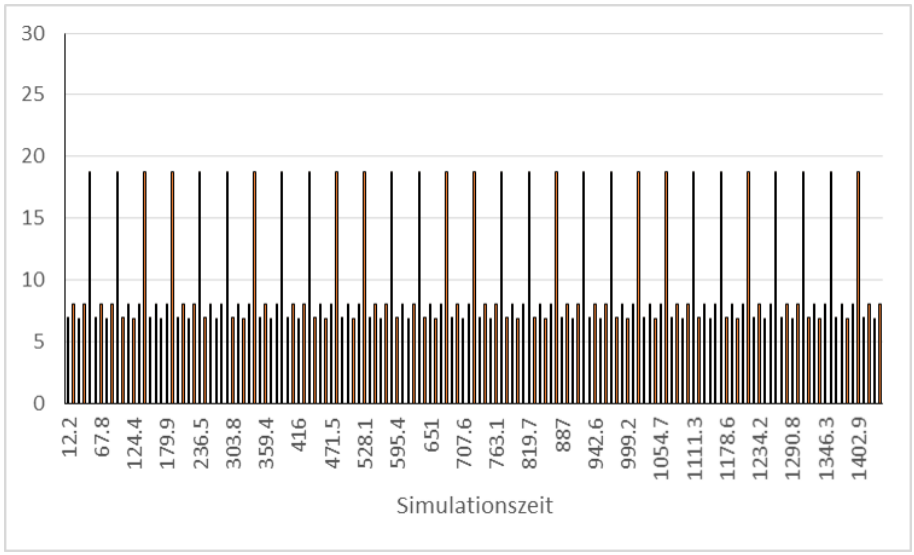

Abbildung 19: Entwicklung der Zeit zwischen zwei aufeinanderfolgenden Abfahrten auf dem Linienverlauf 0 an der Haltestelle 10 bei deterministischer Haltestellen-Aufenthaltszeit

achten Spalte den Wert 10 (für die beobachtete Haltestelle). Die betreffenden Zeilen werden dann aufsteigend nach dem Wert in der ersten Zeile (dem Zeitpunkt des Events) sortiert. Abbildung 19 zeigt den Verlauf der Zeitabstände zwischen zwei aufeinanderfolgenden Abfahrten, abgetragen auf der y-Achse. Zu erkennen ist, dass der vorgesehene Takt von durchschnittlich 7.5 Minuten überwiegend eingehalten wird. Es gibt eine Taktlücke, bei der die Wartezeit auf 18 Minuten steigt, da zu wenige Fahrzeuge auf den Linienverläufen verkehren, um einen durchgehenden identischen Takt zu realisieren.

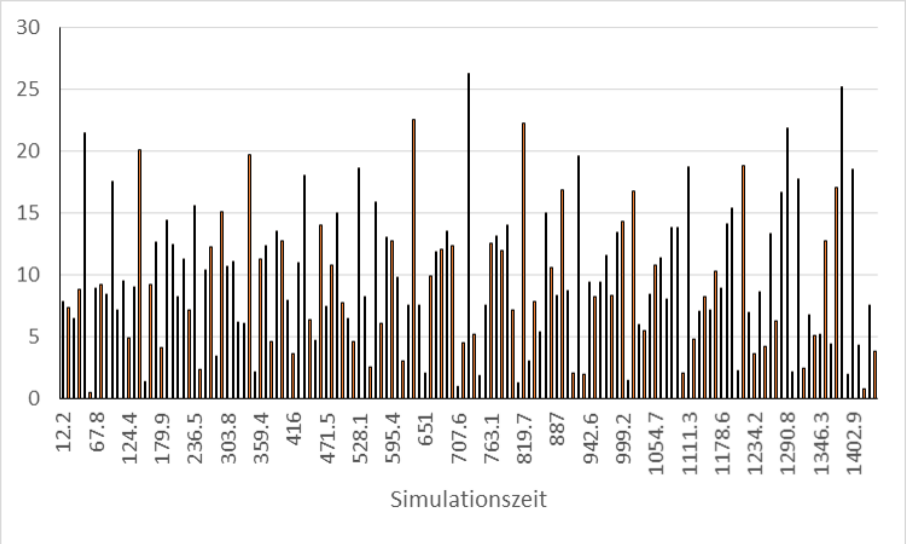

Abbildung 20: Entwicklung der Zeit zwischen zwei aufeinanderfolgenden Abfahrten auf dem Linienverlauf 0 an der Haltestelle 10 bei stochastischer Haltestellen-Aufenthaltszeit

Möchten wir nun stochastische Aufenthaltszeiten simulieren, müssen wir dies zunächst konfigurieren. Hierzu wird in der Datei `bussim_global.h` der Wert der Konstante BUSSIM_DEVIATION_WAITING_TIME auf 0.5 gesetzt. Dadurch wird veranlasst, dass bei jeder Fahrzeugankunft an einer Haltestelle eine zufällige Wartezeit gleichverteilt aus dem Intervall zwischen 0.75-0.5 und 0.75+0.5 Minuten festgelegt wird. Anschließend wird das Projekt neu übersetzt und mit dem o.a. Befehl gestartet. Erneut wird die Eventliste in eine Textdatei ausgegeben. Auf identische Art und Weise erzeugen wir dadurch die Grafik in Abbildung 20. Deutlich ist zu sehen, dass die kleinen Abweichungen von der erwarteten Wartezeit von 0.75 Minuten zu extrem schwankenden Abständen zwischen zwei aufeinanderfolgenden Abfahrten führt. Diese schwanken zwischen 0.4 und 26.3 Minuten.
Durch die resultierende unregelmäßige Taktfolge kommt es im Simulationszeitraum zu diversen Situationen, in den sich die Fahrzeuge gegenseitig blockieren (da sie nicht überholen können/dürfen). Diese sind in Abbildung 21 durch die grauen Punkte entlang des Linienverlaufs gekennzeichnet („Bunchpoints“).

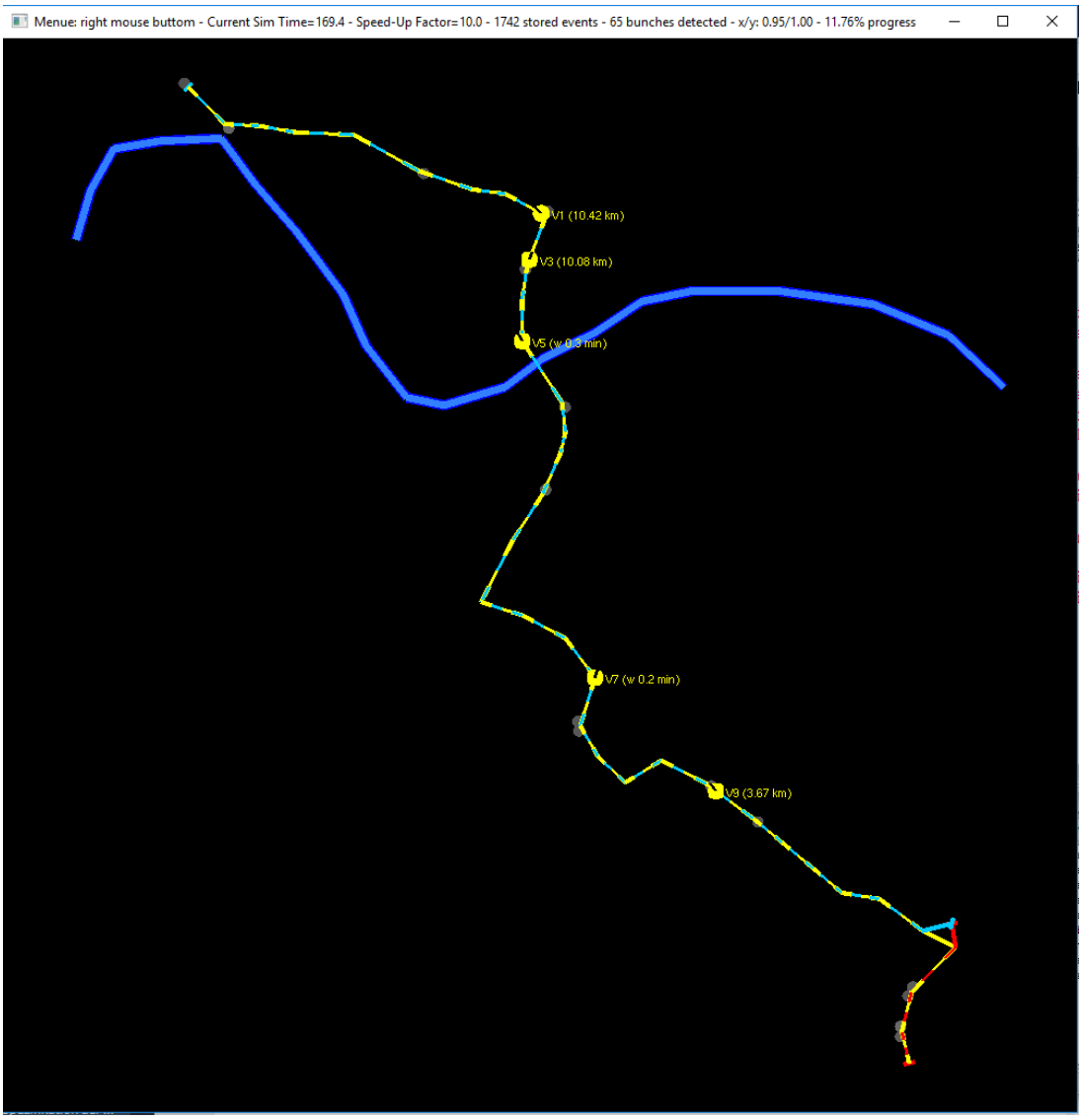

Abbildung 21: Durch die unregelmäßige Taktfolge induzierte Bunchpoints (Fahrzeuge blockieren sich gegenseitig) dargestellt durch die grauen Punkte entlang des Linienverlaufs

A Liste der Haltestellen im Verlauf der Linie 13

Die Geo-Koordinaten der Haltestellen stammen von www.openstreetmap.org

- Prohlis Gleisschleife (13.7987202;50.9993055)
- Georg-Palitzsch-Str. (13.7978505;51.0019235)
- Jacob-Winter-Platz (13.7990222;51.0052175)
- Albert-Wolf-Platz (13.8042152;51.0089377)
- Trattendorfer Straße (13.8003263;51.0103155)
- Altreick (13.7952035;51.0129712)
- Hülßestraße (13.7907414;51.0134771)
- Lohrmannstraße (13.7873736;51.0155091)
- Wieckestraße (13.7836863;51.0176679)
- Otto-Dix-Ring (13.7803055;51.0196443)
- Eugen-Bracht-Str. (13.7747082;51.0225596)
- Cäcilienstraße (13.769357;51.0245206)
- Hugo-Bürkner-Str. (13.7650843;51.0226629)
- Mockritzer Straße (13.7618549;51.0248476)
- Wasaplatz (13.7597683;51.0273837)
- S-Bf. Strehlen (13.7615109;51.0314038)
- Querallee (13.7579935;51.0346843)
- Zoo (13.7529649;51.0365759)
- Lennéplatz (13.7480175;51.0377002)
- Georg-Arnhold-Bad (13.7517587;51.0428411)
- Straßburger Platz (13.7553024;51.0466861)
- St.-Benno-Gymn. (13.7574188;51.0501163)
- Dürerstraße (13.7579853;51.0518303)
- Sachsenallee (13.7576199;51.054302)
- Rosa-Luxemburg-Pl. (13.7528767;51.0594439)
- Bautzn.-/Rothenb.Str. (13.7528809;51.0629002)
- Görlitzer Straße (13.7535012;51.0658187)
- Alaunplatz (13.7556933;51.0698441)
- Bischofsweg (13.7509442;51.0716791)

- Bischofsplatz (13.7469434;51.0720774)
- Friedensstraße (13.7406711;51.0736034)
- Liststraße (13.7329574;51.0766036)
- Bü rgerstraße (13.7257934;51.0768443)
- Rathaus Pieschen (13.7217588;51.0773648)
- Altpieschen (13.7176658;51.0774949)
- Mickten (13.713299;51.0806)