TECHNISCHE UNIVERSITÄT DRESDEN

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

BuS-Sim - Eine C++ - Bibliothek zur Simulation von Fahrzeugbewegungen in ÖPV-Netzwerken

Jörn Schönberger

Dresden, 26. November 2020

Inhaltsverzeichnis

1 Änderungen / Updates der aktuellen Version gegenüber Vorversionen

- Version 1.04 (vom 15.11.2020)
  - Es können nun die Kennzahlen zum modellierten Netzwerk (Anzahl Haltestellen, Anzahl Streckenabschnitte, Anzahl Linien, Anzahl Fahrzeuge) in der Titelleiste des Simulationsfensters angezeigt werden. Hierfür muss durch Drücken der Taste <N> die Titelleisten-Anzeige gewechselt werden. Durch erneutes Betätigen der Taste <N> wird die Anzeige des Simulationsfortschritts wieder eingeblendet. Analog kann der Wechsel der in der Titelleiste angezeigten Informationen über das Maustasten-Menü angezeigt werden.
  - Es ist nun möglich, nur ausgewählte Fahrzeuge in der Simulation anzuzeigen. Soll nur ein Teil der verfügbaren Fahrzeuge angezeigt werden, so muss dies im Quellcode hinterlegt werden. Wie dies genau funktioniert ist in Abschnitt 5.4 beschrieben und an einem Beispiel demonstriert.
  - Es können nun zwei Arten von Haltestellen (reguläre Verkehrshalte und Betriebshalte) unterschieden werden. Wie dies funktioniert, steht in Abschnitt 5.2.2.
  - Es können nun zwei Arten von Linienverläufen (reguläre Verkehrslinienverläufe sowie Einrück-/Ausrücklinien) unterschieden werden. Die notwendigen Schritte hierzu stehen in Abschnitt 5.3.
  - Streckenabschnitte, die in keinem regulären Verkehrslinienverlauf einhalten sind, werden im Liniennetz trotzdem angezeigt (als sehr dünne Linie). Sie können insb. für Einrück- bzw. Ausrücklinienverläufe berücksichtigt werden.
  - Für jedes Fahrzeug muss nun ein individueller Umlaufplan hinterlegt werden. Dieser setzt sich zusammen aus einer Seuqenz von unmittelbar nacheinander abzufahrenden Linienverläufen und kann auch Einrücke- und Ausrückelinienverläufe enthalten, so dass Fahrzeugumläufe realitätsgetreu an einem Betriebshof beginnen bzw. enden. Details zur Umlaufplan-Einrichtung stehen in Abschnitt 5.5.
- Version 1.03 (vom 23.10.2020)
  - Es gibt nun ein zweites Menü, das durch Klicken der mittleren Maustaste angezeigt wird. Über dieses Menü können einzelne oder alle Linienverläufe inkl. aktuellem Fahrzeugverkehr ausgeblendet bzw. eingeblendet werden. Das Resultat der Auswahl wird nach Anzeige des nächsten Zeitschritts berücksichtigt, d.h. im manuellen Modus erst nach erneutem Betätigen der <+>-Taste.
- Version 1.02 (vom 23.10.2020)
  - Änderung der Anzeige von Linienverläufen, die einen Pfeil gemeinsam durchlaufen. Anstelle der Darstellung nebeneinander werden diese nun übereinander angezeigt. Durch Farbwechsel in der Pfeilfarbe wird der gemeinsame Linienverlauf angezeigt („gestreifte Pfeile“).
  - verschiedene Informationen über den Simulationsablauf stehen nun in der Fensterkopfzeile.
  - Das am Bildschirm ausgegebene Menü ist verschwunden. Stattdessen öffnet sich ein Menü, wenn die rechte Maustaste betätigt wird. Die Hotkeys für die Befehle sind dort auch hinterlegt.
  - Es gibt nun die Möglichkeit, in das Netzwerk hinein zu zoomen (Tasten <7> und <8>). Mit <5> wird die Ausgangsanzeige wieder hergestellt.
  - Es ist möglich, den angezeigten Netzwerkausschnitt nach links (<1>), oben (<2>), unten (<4>) oder rechts (<3>) zu verschieben. Mit <5> wird die Ausgangsanzeige wieder hergestellt.

2 Was ist BuS-Sim ?

BuS-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 BuS-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 BuS-Sim verinnerlicht werden.

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

BuS-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.
BuS-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 BuS-Sim aber auch zur Erstellung von anschaulichen Beispielen, so dass insb. Lehrende mit verkehrswissenschaftlichen Interessen BuS-Sim Sinn stiftend nutzen können.

3 Installation

3.1 Überprüfung und ggf. Aktualisierung der Grafiktreiber

BuS-Sim nutzt OpenGL (`https://www.opengl.org/`) zur Erzeugung von Grafiken. 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

BuS-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`

BuS-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 BuS-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 dazu 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 und 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.
- 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. BuS-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 BuS-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.

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

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 dazu 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 und kopieren Sie die vier `.h`-Dateien dort hinein.

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 BuS-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 BuS-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

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

- `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

BuS-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 BuS-Sim

Nachfolgend werden einige Konzepte beschrieben, die bei der Benutzung von BuS-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 räumlich dargestellt wird. Die Simulation läuft zeitgesteuert ab. 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 um eine Zeiteinheit erhöht. Mit fortschreitender Simulationszeit bewegen sich die Fahrzeugsymbole entlang der hinterlegten Linienverläufe durch das Netzwerk (Abbildung 1).

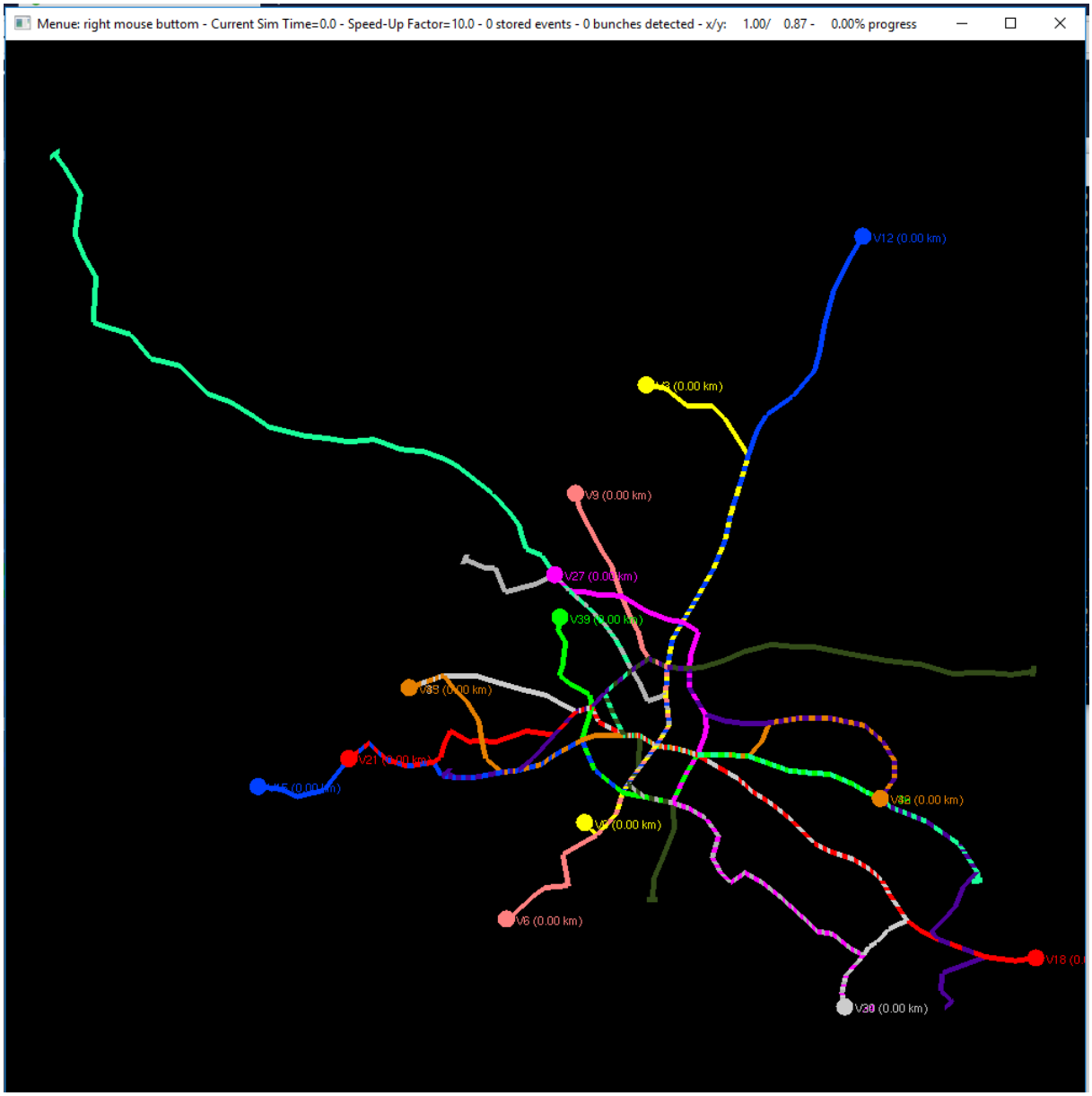

Abbildung 1: Das Simulationsfenster von BuS-Sim

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 5.4).

4.2 Zeitmodell

Eine Zeiteinheit in BuS-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 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.
BuS-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. BuS-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 BuS-Sim

Die überwachten Events in BuS-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 BuS-Sim und die vorbereitenden Schritte, die zur Durchführung eines Simulationslaufs notwendig sind. Anhand der DVB-Straßenbahnlinie 13 wird die Konstruktion des Linienverlaufs erklärt und die Spezifikation der Fahrzeuge und deren Aktivitäten beschrieben.

5.1 Basis-Objekte in BuS-Sim

Objekt	Beschreibung
BUSSIM_POINT	Repräsentation einer Haltestelle (inkl. Endhaltestellen oder Betriebshalte / Depots)
BUSSIM_ARC	Eine Verbindung von zwei benachbarten Haltestellen
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: BuS-Sim -Objekte

BuS-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. Einrichten der Haltestellen, d.h. Spezifikationen der ***BUSSIM_POINT***-Objekte.

2. Einrichten der Streckenabschnitte, d.h. Spezifikationen der ***BUSSIM_ARC***-Objekte.

3. Definition der Linienverläufe, der ***BUSSIM_LINE***-Objekte.
4. Konfiguration der Fahrzeuge (***BUSSIM_VEHICLE***-Objekte) und
5. Festlegung der zugehörigen Fahrten im Netzwerk („Umlaufplan“) in einem ***BUSSIM_ROTATION***-Objekt.

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

5.2 Einrichten der Infrastruktur

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

5.2.1 Netzwerk-Objekt

Jedes BuS-Sim - Projekt beinhaltet genau ein Netzwerk-Objekt von Typ ***BUSSIM_NETWORK*** mit dem Namen `NET`. Dieses enthält u.a. 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.
Zunächst ist eine sich ungefähr in der geografischen Mitte des abzubildenden Netzwerks befindliche Haltestelle zu spezifizieren. Diese muss nicht in der ersten einzurichtenden Linien enthalten sein. Sie bestimmt die Mitte der Bildschirmausgabe. Wir bezeichnen Sie als Referenzhaltestelle. Diese Referenzhaltestelle wird als `PT[0]` abgespeichert. Anschließend werden die Haltestellen der Linien 13 abgespeichert. In unserem Fall sind dies 36 Haltestellen (vgl. Anhang A), die in `PT[1]`, `PT[2]`, ..., `PT[36]` gepeichert werden sollen. Somit muss das Attribut `POINTS` den Wert 37 annehmen.
Falls die Referenzhaltestelle schon im Linienverlauf der einzurichtenden Linie enthalten ist (wenn also beispielsweise die DVB-Linie 3 zuerst eingerichtet wird), dann muss diese nicht zweimal abgespeichert werden.
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.
Nachdem nun feststeht, wie viele Haltestellen (37) und wieviele Streckenabschnitte (70) benötigt werden, kann das leere Netzwerk in der Datei `main.ccp` eingerichtet werden. Dies geschieht mit dem Befehl `class BUSSIM_NETWORK NET(37,70,0,0);`. Das erste Argument spezifiziert die Anzahl abzuspeichernder Haltestellen (`POINTS`), das zweite die Anzahl der benötigten Streckenabschnitte (`ARCS`). Die Anzahl der Linienverläufe (`LINES`) wird später über das dritte Argument und die Anzahl der verfügbaren Fahrzeuge (`VEHICLES`) über das vierte Argument festgelegt. Wir weisen den letzten beiden Argumenten zunächst den Wert 0 zu.

5.2.2 Haltestellen

Die Haltestellen werden als Bestandteil des Netzwerks gespeichert. Hierzu wird die zum Netzwerk-Objekt gehörende Funktion `BUSSIM_NETWORK::specify_nodes` genutzt. Der zu ergänzende Quellcode befindet sich in der Datei `bussim_network.cpp`.
Wir beginnen mit der Einrichtung der Referenzhaltestelle „Hauptbahnhof“. Diese Haltestelle erhält die (x;y)-Koordinaten (0;0), d.h. wir fügen die Befehle

```
this→PT[0].world_x=0;                                    (1)
```

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Knotens
world_x	double	x-Koordinate des Knotens
world_y	double	y-Koordinate des Knotens
screen_x	double	x-Koordinate des Knotens am Bildschirm (wird durch BuS-Sim berechnet)
screen_y	double	y-Koordinate des Knotens am Bildschirm (wird durch BuS-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 3: Attribute für das Objekt `BUSSIM_POINT`

und

```
this→PT[0].world_y=0;
```
(2)

zu Beginn der Funktion ein.
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. 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 $d_y(H_1; H_2) := 111km \cdot (b_1 - b_2)$.

```
this→PT[1].world_x=4.6384;
```
(3)

```
this→PT[1].world_y=-4.4708;
```
(4)

Wir können somit die (x;y)-Koordinaten einer Haltestelle relativ zur Referenzhaltestelle „Hauptbahnhof“ wie folgt bestimmen. Die Haltestelle „Hauptbahnhof“ besitzt die Geo-Koordinaten $(13.7333927; 51.0395826)$ und die Endhaltestelle „Prohlis Gleisschleife“ besitzt die Geo-Koordinaten $(13.7987202; 50.9993055)$. Die x-Differenz in Kilometern beträgt nach dem soeben Gesagten $71km \cdot (13.7987202 - 13.7333927) = 4.6384km$. Die y-Differenz in Kilometern beträgt nach dem soeben Gesagten $111km \cdot (50.9993055 - 51.0395826) = -4.4708km$. Somit erhält die Haltestelle „Prohlis Gleisschleife“ die kartesischen Koordinaten $(4.6384km; -4.4708km)$. Wir richten diese Haltestelle mit dem Index 1 durch

die beiden Befehle (3) und (4) ein. Analog verfahren wir für die übrigen 35 Haltestellen, die durch die Linie 13 bedient werden.
In einer Tabellenkalkulation kann diese Rechnung effizient und schnell für eine komplette Haltestellenliste erstellt werden. Ebenso kann durch die `VERKNÜPFEN`-Funktion sehr effizient und fehlerfrei der entsprechende Quellcode in Excel generiert werden. Dieser kann dann über die Zwischenablage als Text in die Funktion `BUSSIM_NETWORK::specify_nodes` eingefügt werden.

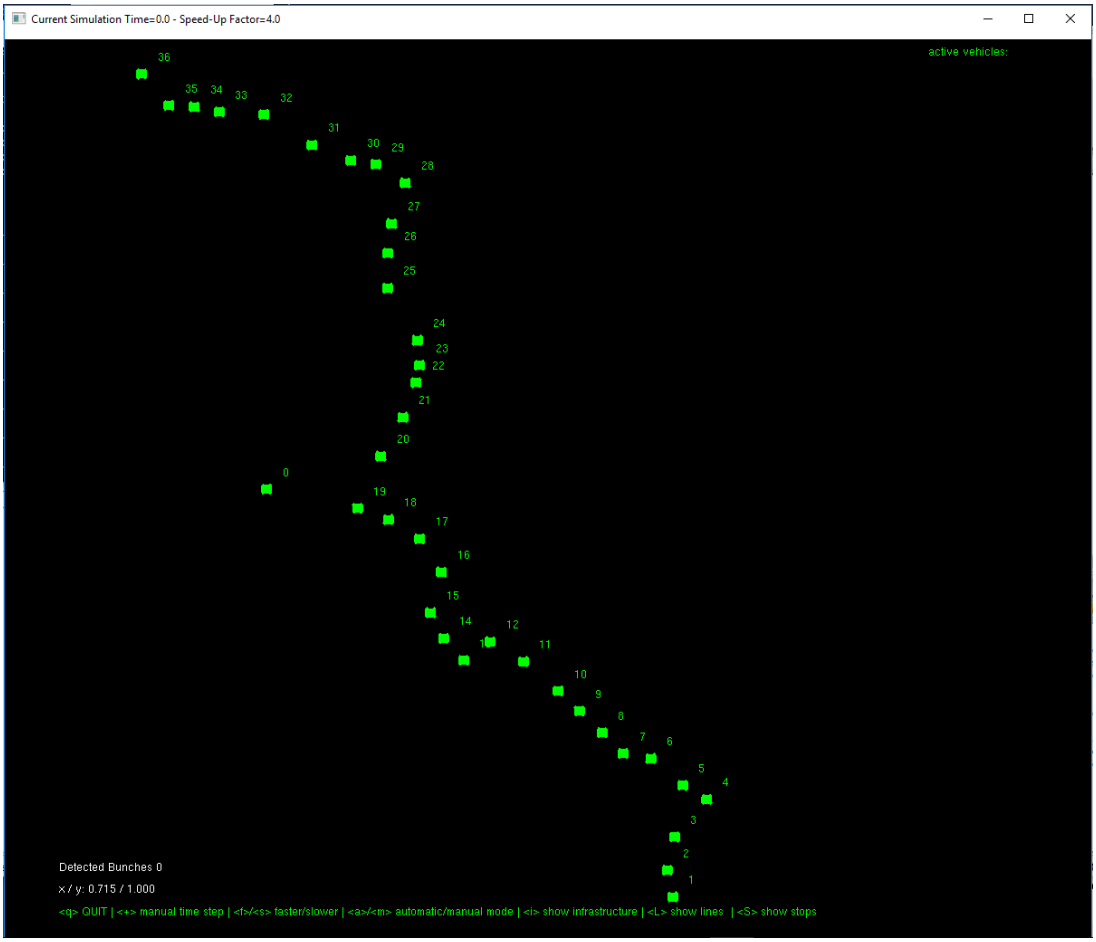

Abbildung 2: Referenzhaltestelle (0) und Haltestellen entlang der Streckenführung der Linie 13 (1-36)

Es besteht die in BuS-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 Werte: `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.

5.2.3 Anzeige des Netzwerks

Initialisieren Sie das Netzwerk in der Datei `main.cpp` nach dem Einfügen der Quellcode-Fragmente (vgl. `bussim_network.cpp` mit dem Befehl `class BUSSIM_NETWORK NET(37,0,0,0);` und erstellen Sie das ausführbare Programm in *`Code::Blocks`* . Starten Sie anschließend das Programm, Klicken Sie mit dem Mauszeiger in das Simulationsfenster und drücken Sie die Taste <S>. Nun werden die hinterlegten Haltestellen angezeigt (Abbildung 2). Ein erneutes Drücken der Taste <S> blendet die Haltestellen wieder aus.

5.2.4 Streckenabschnitte

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 4: Attribute für das Objekt `BUSSIM_ARC`

Streckenabschnitte werden in Objekten des Typs ***BUSSIM_ARC*** abgelegt. Tabelle 4 zeigt die verfügbaren Attribute. Die Streckenabschnitte müssen mit einer eindeutigen, fortlaufenden und lücknlos 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 BuS-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.

$$\texttt{this}\rightarrow\texttt{ARC[0].orig\_node} = 1; \tag{5}$$

$$\texttt{this}\rightarrow\texttt{ARC[0].dest\_node} = 2; \tag{6}$$

$$\texttt{this}\rightarrow\texttt{ARC[0].ID} = 0; \tag{7}$$

Mit den Befehlen (5)-(6) wird die Verbindung (der „Pfeil“) $(1; 2)$ erzeugt und der Variablen `ARC[0]` zugewiesen. Diesem Streckenabschnitt wird abschließend der Attribut-Wert `ID`=0 zugewiesen (7). Die übrigen 69 ***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)`.
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(37,70,0,0);` 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 (Ab-

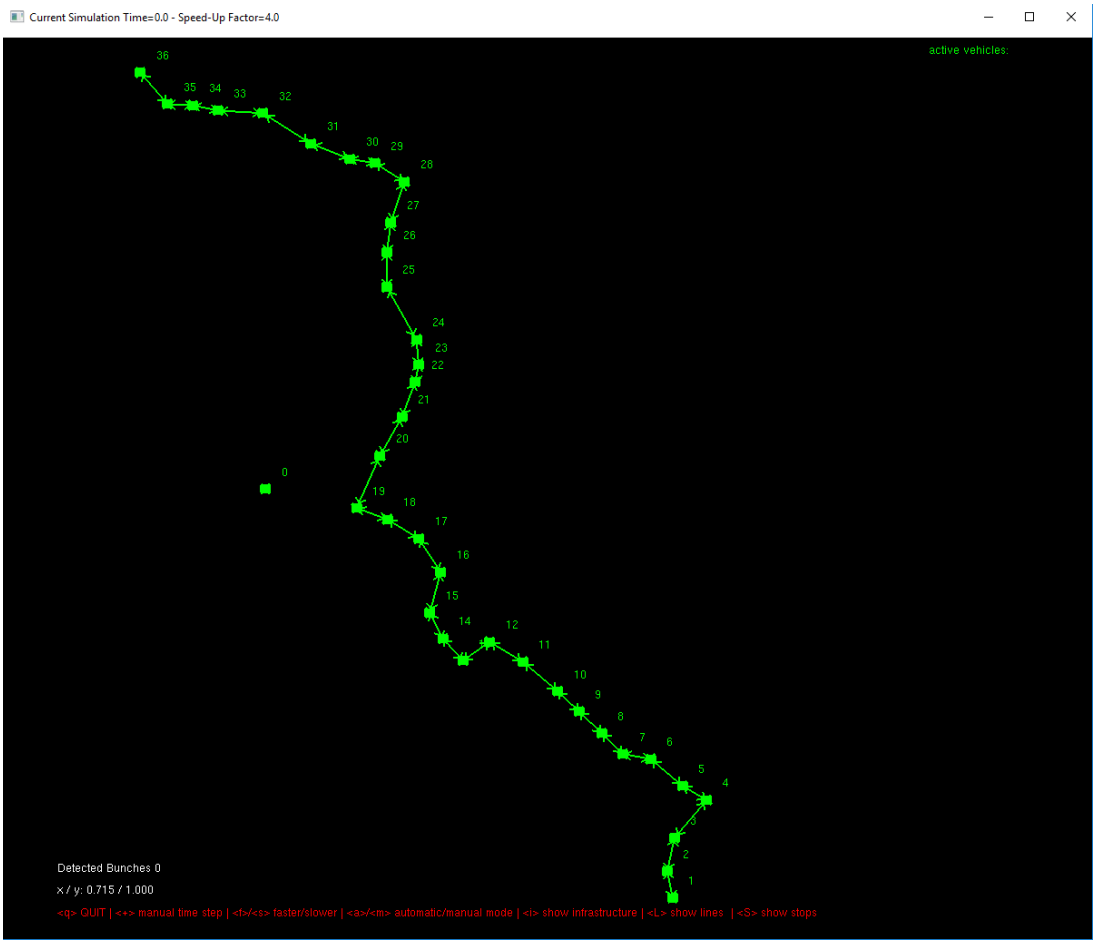

Abbildung 3: Streckenverlauf der Linie 13

bildung 3). Ein Drücken der Tastenkombination UMSCHALT+<S> blendet auch die Haltestellen ein. Es erscheint der in Abbildung 3 gezeigte Streckenverlauf bestehend aus Knoten (Haltestellen) und Pfeilen (Streckenverbindungen) in beide Richtungen.

$$\text{class BUSSIM\_ NETWORK NET}(\cdot,\cdot,\cdot,\cdot); \tag{8}$$

Damit sind alle Arbeiten an der Infrastruktur abgeschlossen. Falls weitere Haltestellen und/oder Streckenabschnitte hinzugefügt werden sollen, so muss zunächst die Anzahl der zusätzlichen Haltestellen sowie der zusätzlichen Streckenabschnitte ermittelt werden. Anschließend sind in der Datei `main.cpp` die Argumente des Netzwerk-Konstruktors (8) 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.

5.3 Einrichten von Linienverläufen

Wie haben bisher Infrastruktur hinterlegt, in der Straßenbahnen fahren können. Im nächsten Schritt müssen wir nun den Linienverlauf der Straßenbahnlinie 13 innerhalb der konfigurierten Infrastruktur festlegen. In BuS-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 BuS-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 das 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.

$$class\ BUSSIM\_NETWORK\ NET(37, 70, 2, 0); \tag{9}$$

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 in (9) gezeigt modifizieren, damit wir die beiden Linienverläufe für die Linie 13 einrichten können.
Die vorhandenen Attribute eines ***BUSSIM_LINE***-Objekts sind in Tabelle 5 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 3 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

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 5: 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 3: Code in der Funktion `BUSSIM_NETWORK::specify_lines()` zur Linienverlaufsspezifikation

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.

5.4 Spezifikation der Fahrzeuge

Ein Fahrzeug wird in BuS-Sim durch ein ***BUSSIM_VEHICLE***-Objekt repräsentiert. Zunächst müssen wir uns überlegen, wieviele Fahrzeuge wir nutzen möchten und diese Zahl als viertes Argument in den Infrastruktur-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.

$$class\ BUSSIM\_NETWORK\ NET(37, 70, 2, 10); \tag{10}$$

Der Netzwerk-Konstruktor muss wie in (10) angepasst werden. Damit werden die 10 Fahrzeuge als vierter 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().

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

Abbildung 4 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.
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-

```
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 visisble again
12     this->VEHICLE[6].VISIBLE = 1;
13     this->VEHICLE[8].VISIBLE = 1;
14 }
```

Abbildung 5: Code in der Funktion `BUSSIM_NETWORK::set_visibility()` zur Auswahl der angezeigten Fahrzeuge

Klasse. Ein Beispiel ist in Abb. 5 zu finden. Es sollen nur die Fahrzeuge 6 und 8 angezeigt werden. Daher werden zunächst alle Fahrzeuge unsichtbar (Zeilen 6-9). Anschließend werden ausschließlich die beiden gewünschten Fahrzeuge sichtbar gemacht (Zeilen 12-13).

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

5.5 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 5.6 beschreibt, wie die Abarbeitung eines Umlaufplans zeitgesteuert gestartet wird.

In BuS-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 BuS-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. 6 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[0]` bedient und anschließend auf den Linienverlauf `LINE[1]` wechselt. Abschließend muss dieses Fahrzeug erneut den Linienverlauf `LINE[0]` abfahren. Anschließend bleibt es am Zielort dieses Linienverlaufs stehen und wird dort deaktiviert

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

Abbildung 6: Code in der Funktion `BUSSIM_NETWORK::specify_rotations()` zum Aufbau von Umlaufplänen für Fahrzeuge

5.6 Fahrplan

BuS-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).

```
1  void BUSSIM_NETWORK::start_rotations(void)
2  {
3      if( (this->CURRENT_TIME >= 0) && (this->CURRENT_TIME < 0.5) )
4      {
5          this->VEHICLE[0].activate(this);
6      }
7      if( (this->CURRENT_TIME >= 10) && (this->CURRENT_TIME < 10.5) )
8      {
9          this->VEHICLE[1].activate(this);
10     }
11     if( (this->CURRENT_TIME >= 20) && (this->CURRENT_TIME < 20.5) )
12     {
13         this->VEHICLE[2].activate(this);
14     }
15         if( (this->CURRENT_TIME >= 0) && (this->CURRENT_TIME < 0.5) )
16     {
17         this->VEHICLE[3].activate(this);
18     }
19     if( (this->CURRENT_TIME >= 10) && (this->CURRENT_TIME < 10.5) )
20     {
21         this->VEHICLE[4].activate(this);
22     }
23     if( (this->CURRENT_TIME >= 20) && (this->CURRENT_TIME < 20.5) )
24     {
25         this->VEHICLE[5].activate(this);
26     }
27 }
```

Abbildung 7: Code in der Funktion `BUSSIM_NETWORK::start_rotations()` für die zeitgesteuerte FAhrzeug-Aktivierung

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. Ein Umlaufplan kann nur einmal aktiviert werden
Im vorliegenden Beispiel sollen 6 Fahrzeuge starten. Abbildung 7 zeigt den zugehörigen Quellcode an. Fahrzeug `VEHICLE[0]` startet zum Zeitpunkt 0. Zum Zeitpunkt 10 folgt das Fahrzeug `VEHICLE[1)`, usw.
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 8). 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

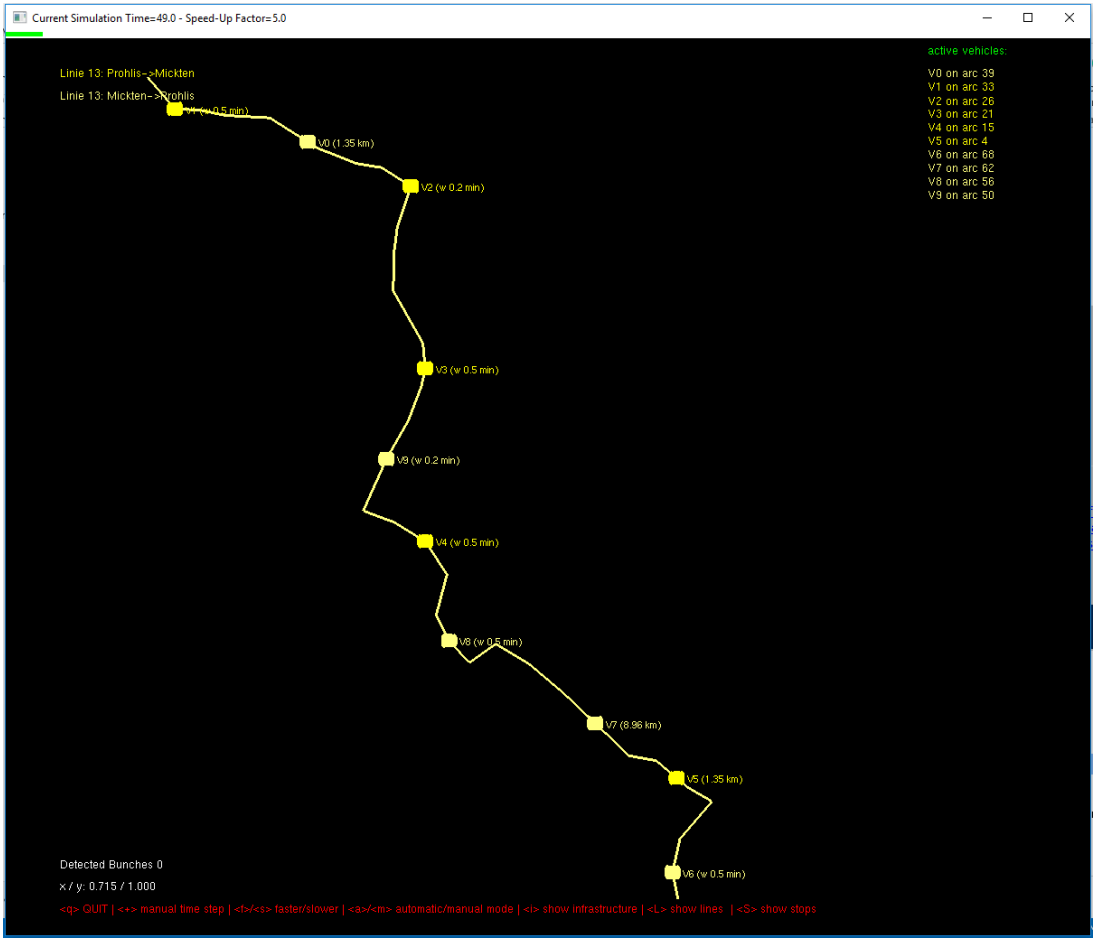

Abbildung 8: Streckenverlauf der Linie 13 mit aktivierten Fahrzeugen

Fahrzeug auf einem Service-Linienverlauf (zum Ausrücken, Einrücken oder Repositionieren), dann wird zusätzlich ein (S) angezeigt.

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)?“

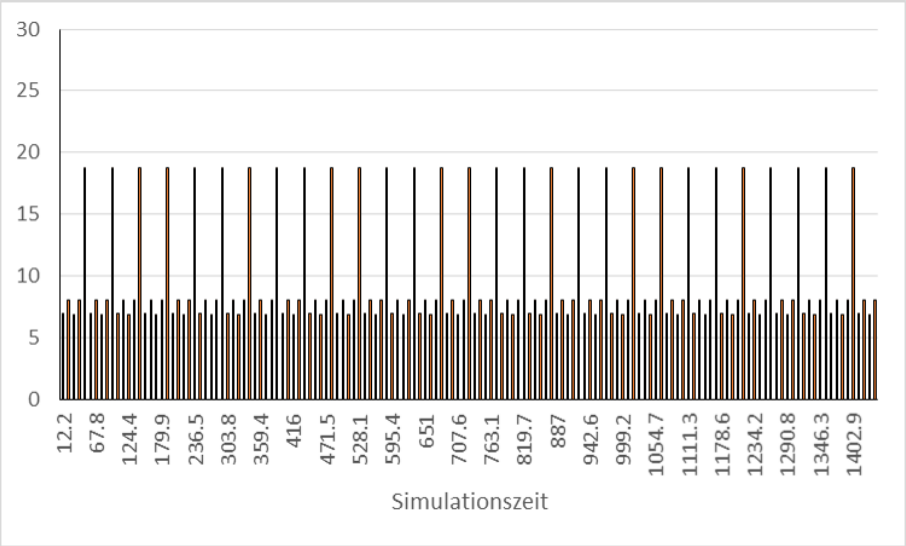

Abbildung 9: Entwicklung der Zeit zwischen zwei aufeinanderfolgenden Abfahrten auf dem Linienverlauf 0 an der Haltestelle 10 bei deterministischer Haltestellen-Aufenthaltszeit

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 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 9 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.
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 übersetzez 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 10. 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

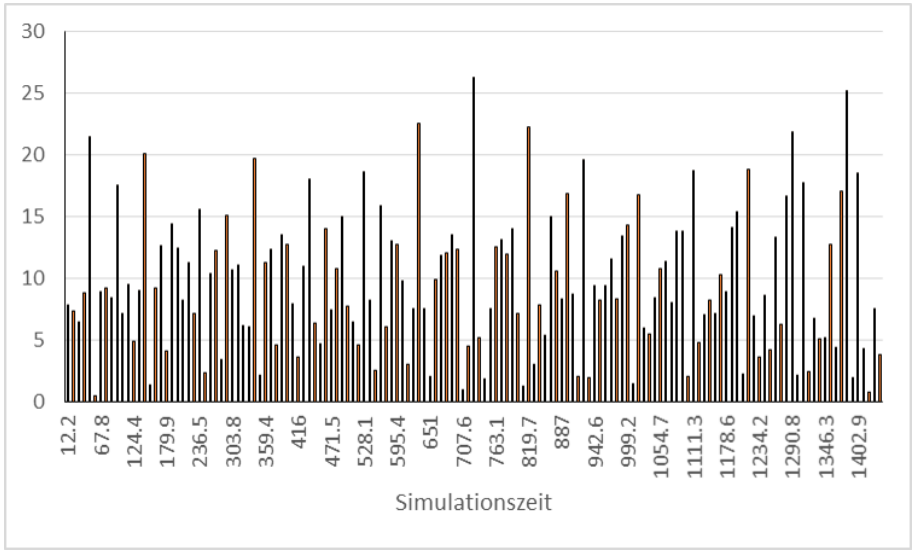

Abbildung 10: Entwicklung der Zeit zwischen zwei aufeinanderfolgenden Abfahrten auf dem Linienverlauf 0 an der Haltestelle 10 bei stochastischer Haltestellen-Aufenthaltszeit

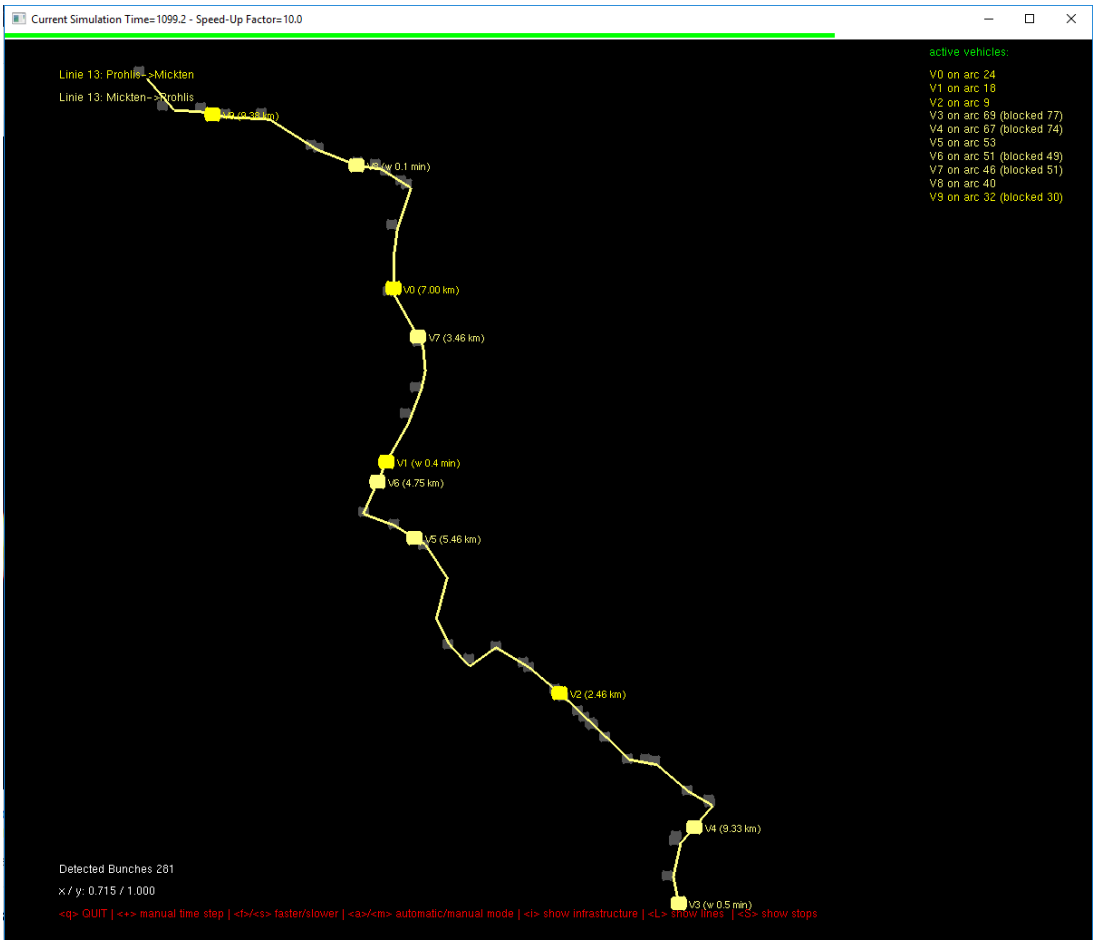

Abbildung 11: Durch die unregelmäßige Taktfolge induzierte Bunchpoints (Fahrzeuge blockieren sich gegenseitig

in Abbildung 11 durch die grauen Punkte entlang des Linienverlaufs gekennzeichnet („Bunchpoints“).

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)