TECHNISCHE UNIVERSITÄT DRESDEN

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

B-u-S-Sim - Simulation von Fahrzeug- und Fahrgastbewegungen in ÖPV-Netzwerken mit C++

Jörn Schönberger

Dresden, 23.11.2022 Version 3.0
29.11.2022 Handbook Version 3.04
b-u-s-sim.de

Inhaltsverzeichnis

1 Änderungen / Updates der aktuellen Version gegenüber Vorversionen

1.1 Lokale Modifications im Handbuch der Software-Distribution 3.0

Version	Beschreibung	Seite
3.01	zusätzlicher Hinweis, dass bei MacOS vorab mit xCode auch die Command Line Tools mit installiert werden (Abschnitt 3.1.2)	8
3.02	zusätzlicher Hinweis, dass der Suchpfad für die Datei `gl.h` unter MacOS in `bussim_global.h` angepasst werden müss(Abschnitt 3.5.3) zusätzlicher Hinweis, dass in xCode die Frameworks *OpenGL* sowie *GLUT* eingebunden werden müssen (Abschnitt 3.5.5)	12 13
3.03	zusätzlicher Hinweis, dass Konfigurationsschritte den tatsächlich vorhandenen Installationspfad benötigen (Abschnitt 3.4.3)	10
3.04	Korrektur der anzupassenden B-u-S-Sim -Header-Datei für die Einbindung der `freeglut`-Header-Datei (Abschnitt 3.5.4)	13

1.2 Simulations-Oberfläche / Simulationssystem

- Keine explizite Unterstützung der 32-Bit-Variante unter Windows mehr.
- Im Simulationsmenü gibt es nun die Option, die Fahrzeug-Labels, die Auskunft über den aktuellen Fahrzeugzustand geben, ein- oder auszuschalten. Hierzu kann die Menü-Option `display vehicle labels` oder die Taste `<v>` verwendet werden.
- Im Simulationsmenü gibt es nun die Option, die Fahrzeuge aus- bzw. einzublenden. Hierzu kann die Menü-Option `show the active vehicles` oder die Taste `<F>` verwendet werden.
- Die aktuelle Netzwerkzeit wird nun im Simulationszeitfenster eingeblendet. Die Position der Uhr kann über einträge in `global.h` frei eingestellt werden. Grundsätzlich startet das erste FAhrzeug zum Zeitpunkt 0, d.h. um 00:00 eines Tages. Um einen realistischeren Betrieb zu simulieren, kann nun ein Zeit-Offset (in Minuten) hinterlegt werden. Hierzu ist die globale Konstante `BUSSIM_MINUTE_OFFSET` in der Datei `global.h` eingeführt worden. Soll der Simulationszeitpunkt 00:00 beispielsweise als 05:30 angezeigt werden, so ist `BUSSIM_MINUTE_OFFSET` in `global.h` auf den Wert `330` zu setzen.
- Das Event-Protokoll wird nun am Ende der Simulation in eine Text-Datei ausgegeben und nicht mehr am Bildschirm ausgegeben.
- Die Simulation protokolliert wichtige Ereignisse in einer Protokoll-Datei. Diese Text-Datei wird beim Beenden des Programm erstellt
- Es wurde eine Anleitung zur Vorbereitung einer MacOS-Umgebung für B-u-S-Sim hinzugefügt.
- Die Status-Anzeige für die Fahrzeuge wurde überarbeitet und erweitert. In einem zweiten Fenster werden diese Informationen nun angezeigt.

1.3 Infrastruktur-Abbildung

- Es gibt nun ein neues Daten-Objekt ***BUSSIM_STOP*** zur Gruppierung von Haltepositionen zu Haltestellen. Dadurch können einzelne Haltepositionen durch Laufwege verknüpft werden. Dies ist eine Vorbereitung für die Nutzung der Nachfrage-Modellierungs-Funktionalitäten, über die B-u-S-Sim ab der Version 3.0 verfügt.
- Es gibt ein neues Infrastruktur-Objekt ***BUSSIM_BLOCK***. In einem ***BUSSIM_BLOCK*** werden ***BUSSIM_ARC***-Objekte zusammengefasst. Auf den Pfeilen, die zu einem Block gehören darf zu

keinem Zeitpunkt mehr als die maximal erlaubte vorgegebene Anzahl von Fahrzeugen verkehren. Mit diesem Infrastruktur-Objekt können z.B. Begegnungsverkehre berücksichtigt werden. Details sind in 6.6 beschrieben.

- Das Objekt ***BUSSIM_POINT*** wurde um das Attribut `INFRASTRUCTURE` ergänzt. Ist der Attributswert =1, so handelt es sich bei diesem Netzwerkknoten **nicht** um eine Haltestelle, sondern um einen Netzwerkknoten, an dem keine Passagiere ein- oder aussteigen. Daher wird an einem derartigen Knoten nicht gehalten (bzw. die Wartezeit eines Fahrzeugs auf Null gesetzt). Mit einem Infrastruktur-Knoten können z.B. Weichen repräsentiert werden. Details zu diesem Feature werden in Abschnitt 6.2 besschrieben.

- Für ein Infrastrukturknoten-Objekt ***BUSSIM_POINT*** gibt es nun ein Attribut `STOPNAME`, um eine verbale Beschreibung dieses Infrastrukturknotens zu ermöglichen. Verwendet wird dieser z.B. für die Angabe zu welchem Zielort ein Fahrzeug gerade fährt. Ebenso kann ein Kurzname vergeben werden, der dann auch sinnvoll am Simulationsbildschirm angezeigt wird.

- Es gibt nun insgesamt vier Verschiedene Typen von ***BUSSIM_POINT***-Objekten. Neben *regulären* Haltepositiosnknoten und *Depot*-Wartepositionen stehen nun noch Weichen-Knoten (*switch*-TYp) und Fahrweg-Detaillierungs-Knoten (*trackpos*-Typ zur Verfügung.

1.4 Fahrzeuge und Fahrzeugumläufe

- zusätzliches Attribut für die Spezifikation der Fahrzeugart (Tram, Bus) zur Steuerung der Fahrzeug-Darstellung

- Umläufe von/bis zur Depottür: automatische Ergänzung von/bis Parkposition

- Zusätzliches Attribut für die maximale-Fahrgastzahl

- Berechnung der Aufenthaltszeit von Fahrzeugen an Haltestellen dynamisiert.

1.5 Fahrgäste

- Nach der Spezifikation aller Knoten und Pfeile erzeugt B-u-S-Sim automatisch weitere Pfeile, die mit dem Wert `BUSSIM_RT_WALK` für das Attribut `ROUTING_TYPE` versehen werden. Diese Pfeile können nur von Fußgängern genutzt werden. Wir verwenden Sie in der Simulation als Gehverbindungen zwischen verschiedenen Haltepositione, die einer Haltestelle zugeordnet sind. Diese Verbindungen werden zur Darstellung von Umstiegen von Fahrgästen an Haltestellen verwendet.

- Es werden verschiedene Typen von optimalen Routen/Pfaden in B-u-S-Sim erzeugt. Für Fahrzeugbewegungen dürfen ausschließlich Fahrbahn-Pfeile verwendet werden. Für Passagier-Reisepfade können auch Fußweg-Abschnitte berücksichtigung werden. Um die zur Pfaderzeugung verwendeten Algorthmen das jeweils korrekte Teilnetzwerk bereitstellen zu können, wird für jeden einzelnen Pfeil dargestellt durch ein ***BUSSIM_ARC*** das Attribut `ROUTING_TYPE` (deklariert als `int`-Variable) genutzt. Über dieses Attribut wird gesteuert, welche Pfeile einer Routing-Routing bereitstellt werden.

- Es gibt nun ein Objekt ***BUSSIM_PAX***. Jede Instanz dieses Objekts repräsentiert einen Fahrgast und für diesen Fahrgast kann ein Reiseweg durch das Netzwerk hinterlegt werden.

- Als wesentliches Werkzeug zur Darstellung von Fahrgastgruppen (wartend und/oder reisend) wurde das Objekt ***BUSSIM_PAX_LIST*** in B-u-S-Sim integriert.

1.6 Verbesserung der Darstellung der Simulation

- In Ergänzung oder als Alternative zu den TOI-Objekten kann nun eine Kartengrafik als Hintergrund des Simulationsfensters angezeigt werden. Hierzu gibt Kapitel 10 Auskunft.

Teil I
Vorbereitungen

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 Vorbereitung des Systems

In diesem Kapitel wird die Erstellung einer ersten Simulation beschrieben. Wir starten ”bei Null“, d.h. wir gehen davon aus, dass Sie bisher noch keine Programmierumgebung für C++ auf Ihrem System installiert haben. Die nachfolgenden Schritte werden in diesem Kapitel sukzessive abgearbeitet.

- Installation der Programmierumgebung (IDE)
- Erstellung eines Programmierprojekts und Import der B-u-S-Sim -Dateien
- Überprüfung und ggf. AKtualisierung der Grafiktreiber
- Installation der benötigten Bibliotheken

3.1 Installation der IDE

3.1.1 Windows-Betriebssysteme: *Code::Blocks*

B-u-S-Sim wurde in der Entwicklungsumgebung *Code::Blocks* programmiert. Die Erstellung bzw. Übersetzung der Simulationssoftware wurde auf einem Windows 10 - System (64Bit) mit zwei verschiedenen Versionen des C++ - Compilers g++ getestet. Es handelt sich 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 grundsätzlich gleich, unterscheiden sich jedoch in Details, so dass nachfolgend die Installation für beide Varianten 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 in3.4.2 für die 32-Bit-Version (mit `mingw32-g++`) bzw. für die 64-Bit-Version (mit `x86_64-w64-mingw32-g++`) in Abschnitt 3.4.3.
Falls Sie bereits *Code::Blocks* mit MinGW installiert haben, dann müssen Sie zunächst herausfinden, welche Compiler-Variante bei Ihnen installiert ist, damit die nachfolgenden Installationsschritte korrekt durchgeführt werden können. 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 `...`.

3.1.2 MacOS-Systeme: Xcode

Code::Blocks funktioniert unter MacOS oftmals nicht korrekt, da aufgrund fehlender aktueller Versionen, ggf. ”zu alte“ Software in *Code::Blocks* unter MacOS verwendet wird. Daher sollten MacOS-Nutzer auf die Alternative IDE **Xcode** zurückgreifen, die von Apple im AppStore kostenlos zur Verfü-

gung gestellt wird[1]. Installieren Sie Xcode vor den weiteren Schritten. Stellen Sie sicher, dass bei der Installation von Xcode die sog. **Command Line Tools** mit installiert werden[2]

3.2 Entpacken des B-u-S-Sim -Zip-Archivs sowie Erstellung eines Projekts in der IDE

Das auf der Website `b-u-s-sim.de` bereitgestellte ZIP-Archiv enthält alle C++ - Quellcode-Dateien, die zur Erstellung der in der Dokumentation beschriebenen Beispielanwendung benötigt werden. Dieses ZIP-Archiv ist zunächst zu entpacken.
Nun muss in der genutzten IDE ein neues Projekt vom Typ Konsolen-Applikation / Terminal Application zu erstellen. In dieses Projekt sind alle `*.cpp`- sowie alle `*.h`-Dateien aus dem ZIP-Archiv zu importieren (vgl. Anhang A).

3.3 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 bzw. Betriebssystem 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.
Nachfolgend beschreiben wir nun, wie Sie Ihren Computer auf die Nutzung der OpenGL-Möglichkeiten vorbereiten.

3.4 Installation der benötigten Bibliotheken: Windows-Systeme

In diesem Unterkapitel beschreiben wir die Installation zusätzlicher Software, die zur Erstellung von B-u-S-Sim benötigt wird, falls Sie einen Computer mit einem Windows-System verwenden.

3.4.1 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. Wir können uns `freeglut` als eine Sammlung von Befehlen vorstellen, die die Nutzung von OpenGL ermöglicht. 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 und starten können, müssen wir die frei und kostenlos verfügbare `freeglut`-Bibliothek auf dem verwendeten 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`

[1] `https://developer.apple.com/de/support/xcode/`
[2] `https://www.freecodecamp.org/news/install-xcode-command-line-tools/`

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.

3.4.2 32-Bit-Variante `mingw32-g++`

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.[3]
- 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.

Kopieren der Bibliotheks-Dateien Wir fahren fort mit den Bibliotheks-Dateien 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. B-u-S-Sim -zip-Archiv bereits eine solche `.dll`-Datei enthält. Diese befindet sich im Unterverzeichnis `\bin\release`.

Integration von `freeglut` in *Code::Blocks* / Bekanntgabe an den Compiler 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

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

`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_static.a`

ein. Damit steht freeglut alle zukünftigen Projekten zur Verfügung.

3.4.3 MS Windows 64-Bit-Variante

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.[4]
- 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.

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. **blue**Da ist zwar ggf. 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.

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

Integration von `freeglut` in *Code::Blocks* / Bekanntgabe an den Compiler 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_static.a`

ein. Damit steht freeglut alle zukünftigen Projekten zur Verfügung.

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.
Damit der geänderte Pfad erkannt wird, starten Sie bitte Ihren Rechner neu. Zumindest müssen Sie aber eine neue Eingabeaufforderung nutzen.

Bei fehlenden `.dll`-Dateien ... Es kann passieren, dass die vom Programm benötigten `.dll`-Dateien nicht gefunden werden. In diesem Fall kommt es beim Start der erzeugten *.exe-Datei zu einer Fehlermeldung. Diese Meldung weißt darauf hin, dass eine benötigte `.dll`-Datei nicht gefunden werden kann. Behoben werden kann diese Fehlermeldung dadurch, dass Sie die entsprechende Datei im Unterverzeichnis `\MinGW\x86_64-w64-mingw32` des *Code::Blocks* -Installationsverzeichnisses suchen und in das Verzeichnis kopieren, in dem die *.exe Datei erzeugt wurde. Ggf. müssen Sie diesen Schritt für mehrere `.dll`-Dateien wiederholen. Bekannt ist dieses Problem für die Dateien `libwinpthread-1.dll`, und .
Eine Alternative zum Suchen und Kopieren der bemängelten bzw. fehlenden `dll`-Dateien ist, auf diese zu verzichten. Dies ist möglich, dann müssen aber alle Informationen, die sonst aus einer `dll`-Datei während der Programm-Ausführung gelesen werden würden bereits bei der Erstellung des Programms, in das Programm integriert werden. Der Wunsch nach dieser sog. **statischen Verlinkung** muss dem Compiler vor Beginn der Erstellung des ausführbaren Programms mitgeteilt werden. Hierfür müssen die sog. Compiler-Optionen des aktuellen Projekts entsprechend eingestellt werden. Dafür sind die folgenden Schritte notwendig.

1. Im Menü *Project* wählen wir die Option *Build options ...*
2. Im sich nun öffnenden Dialogfeld wählen wir oben links zunächst das Projekt `bus_sim`.
3. Nun wählen wir die Karteikarte *Compiler Flags* aus. Hier können wir das Verhalten des Compilers festlegen, in dem die jeweiligen Optionen ausgewählt werden, die beim AUfruf dem Compiler mit auf dem Weg gegeben werden.

4. Wir suchen die beiden Optionen `Static libgcc` und `Static libstdc++` und aktivieren die jeweiligen Auswahlboxen.

5. Abschließend schließen wir den *Build options ...*-Dialog durch Klicken auf den OK-Buttton.

3.4.4 Festlegung des verwendeten C++-Sprachstandards

Unabhängig vom verwendeten Betriebssystem muss nun noch eingestellt werden, welche C++-Sprachversion der Compiler voraussetzen kann. 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.

3.5 Installation der benötigten Bibliotheken: Mac OS-Systeme

In diesem Unterkapitel beschreiben wir die Installation zusätzlicher Software, die zur Erstellung von B-u-S-Sim benötigt wird, falls Sie einen Computer mit einem MacOS-System verwenden..

3.5.1 Getestete Systemumgebung

MacBook Air, M1, 2020, macOS Venture Version 13.0, Xcode 14.0.1

3.5.2 Installation des Package-Managers `homebrew`

Für die nachfolgenden Schritte benötigen Sie ein Terminal, um Befehle an Ihren Computer übergeben zu können.
Ähnlich wie unter Windows-Betriebssystemen muss auch unter MacOS vor der Nutzung von B-u-S-Sim zusätzliche Software ("Bibliotheken") installiert werden, die von B-u-S-Sim erwartet und benötigt werden. Damit diese Installationen möglichst einfach realisiert werden können und nur in Ihrem eigenen Bereich des Systems verbleiben, empfiehlt es sich, zunächst das Tool **homebrew**[5] zu installieren. homebrew ist ein sog. Package-Manager, der zusätzlich benötigte Software ("Pakete") mit wenigen Befehlen aus dem Internet auf den Computer kopiert und dort in die vorhandene Software so integriert, dass diese zusätzliche Software vom Benutzer verwendet werden kann.

3.5.3 Installation der benötigten Packages `xquartz` und `freeglut`

Die erste zusätzlich zu installierende Software ist das Package **xquartz**[6]. Diese Package enthält eine C++ - Bibliothek, mit der Fenster erzeugt und verwaltet werden können.
Nach der erfolgten Installation des xquartz-Packages muss nun noch das Package **freeglut** auf Ihrem Rechner installiert werden[7]. Dieses Package stellt C++-Befehle zur Verfügung, um in den Fenstern, die mit xquartz verwaltet werden, zeichnen zu können.

3.5.4 Anpassen des Pfades für die `freeglut`-Header-Datei

Zunächst müssen wir herausfinden, wo sich die im Projekt einzubindende Header-Datei `freeglut.h` befindet. Dieser Ort hängt u.a. von der Hardware des Geräts sowie der Version des Betriebssystems ab.
Starten Sie für die Lokalisierung der Header-Datei zunächst eine neue Shell bzw. ein Terminalfenster. Tragen Sie hier den Befehl `find / -name freeglut.h` ein. Es werden nun alle Verzeichnis Ihres

[5] `https://brew.sh/index_de`
[6] `https://formulae.brew.sh/cask/xquartz`
[7] `https://formulae.brew.sh/formula/freeglut`

Rechners durchsucht und die Verezichnisse bzw. Pfade angezeigt, die zur benötigten Header-Datei führen, z.B. `/opt/homebrew/include/GL/freeglut.h`. Wir bezeichnen diesen Pfad mit Datei als `HEADERPATH`.
Den gefundenen Pfad müssen wir nun in der Datei `bussim_global.h` eintragen. Öffnen Sie dafür diese Datei in Xcode. Darin kommentieren Sie die beiden Zeilen `#include <GL/gl.h>` und `#include <freeglut/freeglut.h>` aus. Anschliessend fügen Sie in der nächsten Zeile den Eintrag `#include "HEADERPATH"` hinzu, also im Beispiel
`#include /opt/homebrew/include/GL/freegluth.h"`.

3.5.5 Einbinden der externen Bibliotheken in das Projekt

Nun muss dem Compiler noch mitgeteilt werden, dass er beim Erzeugen des ausführbaren Programms verschiedene externe Bibliotheken benutzen kann/soll/muss. Viele dieser Ressourcen sind in zwei sog. Frameworks zusammengefasst. Diese haben die Namen `OpenGL.framework` und `GLUT.framework`. Wie müssen dem Projekt mitteilen, dass es diese beiden Frameworks nutzen muss. Hierfür führen wir folgende Schritte durch:

- Im Projekt-Explorer auf den Ordner des Projekts klicken
- Es öffnet sich rechts das Fenster mit den Projekteigenschaften. Hier das Projekt anklicken und die Build Phases auswählen.
- Öffnen Sie den Bereich *Link Binary with Libraries*.
- Durch Drücken auf das "+Zeichen öffnen SIe einen Diealog in dem Sie die beiden Frameworks auswählen können.

Abschließend müssen wird dem Compiler mitteilen, dass er die im Framework `GLUT` bereitgestellten Befehle berücksichtigen muss (das Framework `OpenGL` wird dann vom Framework `GLUT` eingebunden). Es sind die folgenden Schritte notwendig:

- Im Projekt-Explorer auf den Ordner des Projekts klicken
- Es öffnet sich rechts das Fenster mit den Projekteigenschaften. Hier das Projekt anklicken und die Build Settings auswählen.
- Nun die Option *Other Linker Flags* suchen und dort die Option `-lGLUT` eintragen bzw. hinzufügen (l = "kleines L").

Damit ist die Konfiguration Ihres Rechners für die Nutzung von B-u-S-Sim abgeschlossen. Wir können nun mit der Erstellung des ausführenbaren Beispiels beginnen.

Teil II
Grundlagen

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: Die Fenster

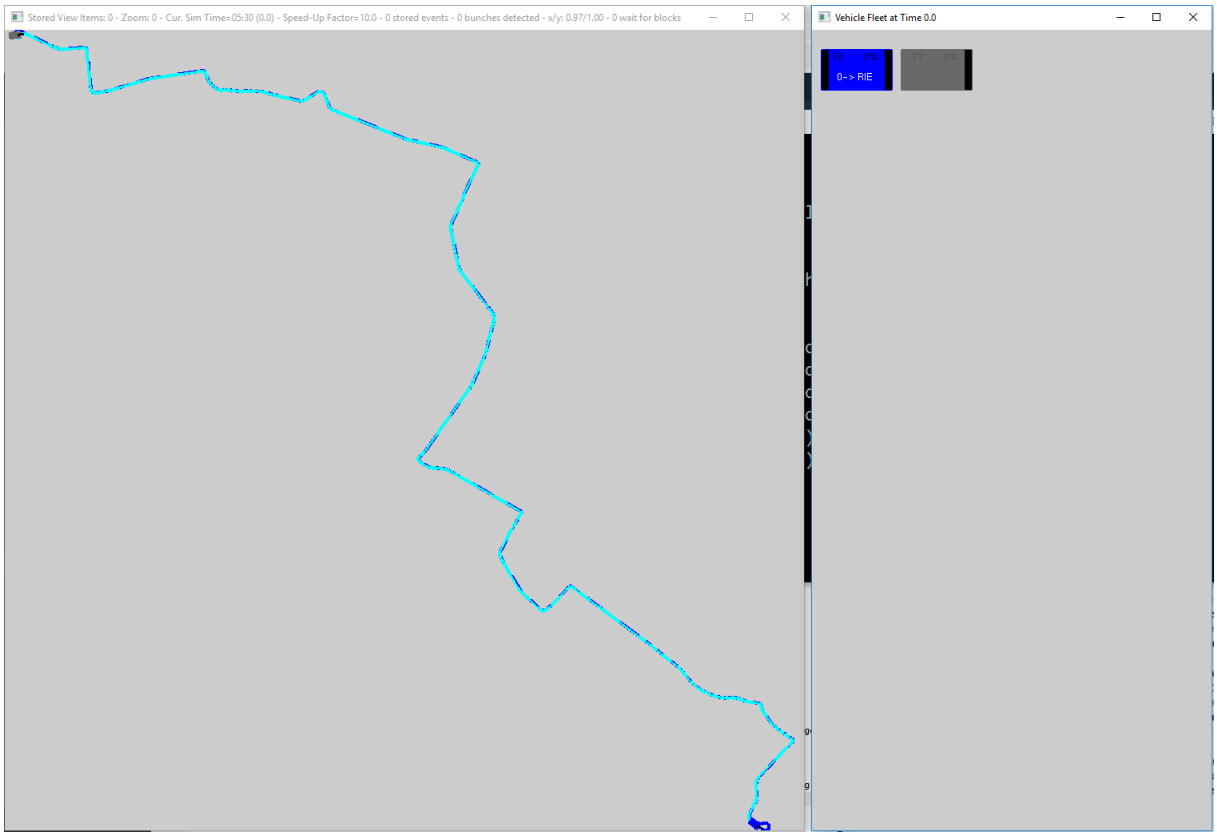

Abbildung 1: Die drei Fenster von B-u-S-Sim

Nach dem Start des Programms über die Kommandozeile öffnen sich zwei Fenster. Im linken Fenster (”Simulationsfenster“) wird eine graphische Darstellung des Netzwerkes angezeigt und ebenso die sich darin bewegenden symbolisierten Fahrzeuge. Das rechte Fenster (”Fahrzeug-Fenster“) enthält Informationen über die vorherigen Fahrzeuge.
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). Dabei nehmen sie ggf. wartende Fahrgäste mit bzw. lassen ankommende Fahrgäste aussteigen.
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.

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 Si-

mulationslaufs (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.h` 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 endet auch, sobald kein Fahrzeug mehr aktiv ist.
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.
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 $a \in \mathcal{A}$ ohne Zwischenhalt zum nächsten Knoten $b \in \mathcal{A}$ 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 $a \in \mathcal{A}$. Dieser wird grundsätzlich automatisch au der geographischen Lage von a und b abgeleitet.
Ein Linienverlauf (eine Linie) ist nun durch eine Sequenz verbundener Pfeile im Graphen $\mathcal{G}$ definiert. Der Linienverlauf gibt den Weg vor, den ein Fahrzeug durch das Netzwerk nehmen muss, um eine Linie zu bedienen. 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 und zeitlich sortiert in eine Protokoll-Textdatei 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
6	BUSSIM_WAIT_FOR_BLOCK	ein Fahrzeug muss auf Einfahrt in einen Pfeil warten, da der Pfeil zu einem Block gehört, dessen Kapazität gerade erschöpft ist

Tabelle 1: Ptozokollierte 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 in eine Protokoll-Datei `protocol_DATUM_UHRZEIT.txt` ausgegeben. Diese Textdatei kann dann einfach in z.B. Excel importiert und dort ausgewertet werden.
Als eines von zwei Event-Arten 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.
Das Event 6 (BUSSIM_WAIT_FOR_BLOCK) tritt immer dann ein, wenn ein Fahrzeug einen Pfeil nicht befahren kann, weil der Pfeil durch ein anderes Fahrzeug genutzt wird. Dieses Event kann nur eintreten, wenn es sogenannte Infrastruktur-Blöcke gibt. Ein Block ist eine Gruppe von Pfeilen und wird gesperrt, sobald mindestens einer der zugehörigen Pfeile durch ein oder mehrere Fahrzeuge belegt ist. Derartige Ereignisse werden gezählt und deren bisherige Anzahl kann in der Kopfzeile des Simulationsfensters abgelesen werden.

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

Objekt	repräsentiert ...
BUSSIM_STOP	... eine Haltestelle (d.h. eine Kollektion von ≥ 1 Knoten des Infrastrukturgraphen)
BUSSIM_POINT	... einen Knoten des Infrastrukturgraphen
BUSSIM_ARC	... eine gerichtete Verbindung von zwei Knoten des Infrastrukturgraphen, entlang deren Fahrzeuge fahren können
BUSSIM_TOI	... geometrische Objekt als wichtige geographische Referenzen in der gezeigten Simulationskarte
BUSSIM_LINE	... einen Streckenverlauf (Linienverlauf) von einem Netzwerk-Knoten (über verschiedene Zwischenknoten) bis zu einer anderen Netzwerk-Knoten
BUSSIM_VEHICLE	... ein Fahrzeug
BUSSIM_EVENT	... ein Ereignis, dass während der Simulation auftritt
BUSSIM_BUNCHPOINT	... einen Ort im Netzwerk, an dem ein Fahrzeug durch ein anderes blockiert wird
BUSSIM_DUTY	... einen Fahrauftrag für ein Fahrzeug (umfasst das Abfahren eines Linienverlaufs)
BUSSIM_ROTATION	... einen Umlaufplan, d.h. eine Liste von nacheinander durch ein spezifiziertes Fahrzeug abzufahrenen Fahraufträgen
BUSSIM_PAX	... einen Fahrgast, mit einem indivudellen Reiseweg durch das Netzwerk
BUSSIM_SCHEDULE	... einen Fahrplan, den ein spezifisches Fahrzeug in der Simulation abarbeiten muss
BUSSIM_NETWORK	... die Gesamtheit aller gespeicherten vorgenannten Objekte und ihrer Instanzen

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

B-u-S-Sim stellt verschiedene C++ - Datenobjekte zur Verfügung, um effizient komplexe Personenverkehrsnetze darzustellen, die die Erfüllung von individuellen Fahrgastwünschen ermöglichen. Diese Objekte sind in Tabelle 2 zusammengestellt. Zu jedem Objekt ***OBJECT*** existiert 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 Objekte spezifiziert werden. Da sie teilweise voneinander abhängig sind, muss hierbei eine vorgegebene Reihenfolge berücksichtigt werden.

1. Festlegung der Haltestellen in Form von ***BUSSIM_STOP***-Instanzen,
2. Einrichten der Netzwerk-Knoten, d.h. Spezifikationen der ***BUSSIM_POINT***-Instanzen,
3. Einrichten der Streckenabschnitte, d.h. Spezifikationen der ***BUSSIM_ARC***-Instanzen,
4. Einrichten von Infrastrukturblöcken, d.h. Spezifikation der ***BUSSIM_BLOCK***-Instanzen,
5. Definition der Linienverläufe, d.h. Spezifikation der ***BUSSIM_LINE***-Instanzen,
6. Konfiguration der Fahrzeuge (***BUSSIM_VEHICLE***-Instanzen),
7. Konfiguration der individuellen Fahrgäste als Instanzen des ***BUSSIM_PAX***-Objekts.

8. Spezifikation wichtiger geographischer Referenzobjekte durch ***BUSSIM_TOI***-Objekte.

9. 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. Dabei ist dem Konstruktor des ***BUSSIM_NETWORK***-Objekts als Parameter die Anzahl der Instanzen der 8 erstgenannten Objekte mitzuteilen.

Teil III
Konfiguration eines Szenarios in B-u-S-Sim

6 Definition des Infrastruktur-Graphens

In diesem Kapitel beschreiben wir die Einrichtung der Infrastruktur, in der sich die Fahrzeuge des ÖPNV bewegen dürfen. Die grundsätzliche Idee besteht darin, die Infrastruktur als mathematischen Graph $\mathcal{G} := (\mathcal{V}; \mathcal{A})$ zu formulieren. Hierbei repräsentieren wir die Knotenmenge $\mathcal{V}$ als eine Liste von Orten, an denen wir verschiedene notwendige Funktionalitäten zur Verkehrsabwicklung hinterlegen. Dies sind zum Beispiel Haltestellen, Haltepositionen, Weichen/Abzweigungen oder Kurven. Die Pfeilmenge $\mathcal{A}$ stellt dann die Fahrstrecken dar, auf denen sich Fahrzeuge ohne von einem Knoten zum nächsten Knoten bewegen können.
B-u-S-Sim stellt drei wesentliche Objekte zur Verfügung, mit denen die Knoten und Pfeile eingerichtet und in der Simulation verwendet werden können. Ein ***BUSSIM_STOP***-Objekt stellt eine Hülle dar, in der eine oder mehrere Haltepositionen von Fahrzeugen zur Aufnahme oder zum Ausstieg von Fahrgästen enthalten sein können. Dementsprechend stellen ***BUSSIM_STOP***-Objekte keinen Netzwerkknoten dar, sondern werden nur zur Gruppierung von Netzwerkknoten verwendet. ***BUSSIM_POINT***-Objekte stehen zur Repräsentation der Netzwerkknoten aus der Knotenmenge $\mathcal{V}$ zur Verfügung. Für die Speicherung von Informationen über Fahrtverbindungen zwischen einem Paar von Netzwerkknoten nutzen wir das ***BUSSIM_ARC***-Objekt.
Wir erklären im weiteren Verlauf dieses Kapitels Schritt-für-Schritt, wie Sie unter Verwendung freizugänglicher Quellen verschiedene Informationen zusammenstellen und konsolidieren, die für die Spezifizierung des Graphen werden, der die reale ÖPNV-Infrastruktur möglichst gut repräsentiert. Wir starten in 6.1 mit der Vorstellung der verschiedenen Typen (Arten) von Netzwerkknoten, die zur Infrastruktur-Modellierung durch B-u-S-Sim zur Verfügung gestellt werden. Anhand der DVB-Straßenbahnlinie 13 demonstrieren wir die notwendigen Schritte der Graphenkonstruktion. Abschnitt 6.2 bildet hierfür den Startpunkt und beschreibt die Einrichtung von Haltestellen als wichtiges Gruppierungs-Werkzeug für Knoten. Die Einrichtung von Haltepositionen, Weichen und ähnlichen relevanten Knoten thematisiert Abschnitt 6.3. Abschnitt 6.4 beschreibt die Einrichtung von Graph-Pfeilen, die Fahrstreckenabschnitte beschreibt. Anschließend erklärt Abschnitt 6.5 die Soezifikation besonderer Infrastruktur-Komponenten wie Gleisschleifen und Überholgleisen. Das Konzept von Infrastrukturblöcken mit denen Kapazitätslimitationen auf Teilen der Infrastrukturen codiert werden, ist Gegenstand des Abschnitts 6.6. Das Aufspüren typischer Modellierungsfehler und deren Behebung thematisiert abschließend Abschnitt 6.7.

6.1 Arten von Netzwerkknoten

Typ	Beschreibung
regular	beschreibt eine reguläre Haltestelle mit kurzer Aufenthaltszeit zum Ein- und Ausstieg von Fahrgästen
depot	beschreibt ebenfalls eine Haltestelle. An einem Depot können jedoch längere Aufenthaltszeiten für Fahrzeuge eingeplant werden (z.B. Pausen).
switch	beschreibt die Position einer Weiche. Ein Fahrzeug hält hier typischerweise nicht an.
trackpos	beschreibt eine Position, an der ein FAhrzeuge seine Fahrtrichtung etwas ändert. Ein Fahrzeug hält hier typischerweise nicht an.

Tabelle 3: Verschiedene Arten von Netzwerkknoten und ihre Aufgaben bzw. Funktionen

Insgesamt kennt B-u-S-Sim vier verschiedene Arten von Netzwerkknoten. Jede Knotenart stellt bestimmte Funktionalitäten zur Verfügung, die zur Darstellung der Infrastruktur benötigt werden (Tab. 3). Alle vier Knotenarten werden als ***BUSSIM_POINT***-Objekt abgespeichert aber mit verschiedenen Methoden initialisiert, da je nach Knotenart verschiedene Informationen festgelegt werden müssen.
Die offentsichtlichste Verwendung eines Netzwerkknotens ist die Spezifikation eines Ortes, an dem ein

Fahrzeug zum Einsteigen- und/oder Aussteigenlassen von Fahrgästen hält. Entsprechende Funktionalitäten stellen Netzwerkknoten des Typs **regular** dar. Wir bezeichnen derartige Knoten auch als **Haltepositionen** bzw. **Plattformen** bzw. **reguläre Knoten**. Die regulären Knoten bilden die Knotenmenge $\mathcal{A}^{platform}$.

Normalerweise wird eine Halteposition verwendet, um Fahrzeuge für den Fahrgastwechsel möglichst kurz anhalten zu lassen. Es gibt aber auch Situationen, in denen ein Fahrzeug eine längere Zeit halten muss bzw. soll, z.B. damit der Fahrer eine Pause einlegen kann oder damit bis zum Erreichen der Abfahrtzeit gemäß einen Fahrplans gewartet werden kann. Um derartige Haltepositionen eindeutig von regulären Knoten unterscheiden zu können und deren besondere Eigenschaft nutzen zu können, werden diese Knoten als sog. **Depot**-Knoten deklariert und in der Menge $\mathcal{A}^{depot}$ konsolidiert.

Wir verwenden ein ***BUSSIM_POINT***-Objekt aber auch, um Weichen bzw. Abzweigungen in einer Infrastruktur darzustellen, an denen Fahrzeuge nicht halten müssen, die aber den Fahrtweg dieser Fahrzeuge bestimmen bzw. beeinflussen. Wir bezeichnen diese Netzwerkknoten als **switches** und sammeln Sie in der Menge $\mathcal{A}^{switches}$.

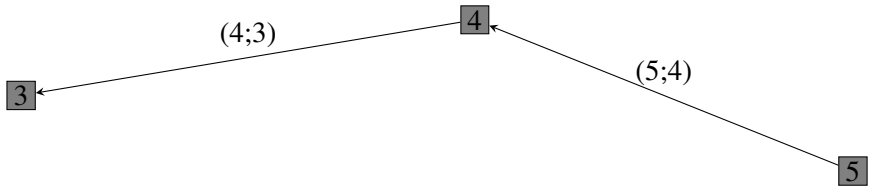

Abbildung 2: Modellierter Streckenabschnitt mit dem ***BUSSIM_POINT***-Objekt 4, der einen *trackpos*-Knoten darstellt.

Die vierte Art eines ***BUSSIM_POINT***-Objekts verwenden wir, um einen realitätsnahen Streckenverlauf zwischen zwei Knoten der ersten drei Arten sicherzustellen. Knoten der vierten Art repräsentieren Orte, an denen ein Fahrzeug seine Fahrtrichtung etwas verändern kann, so dass bespielsweise Kurvenverläufe zwischen zwei Haltestellen möglichst genau modelliert werden können. Abb. 2 zeigt die Verwendung eines solches Knotens mit der Bezeichnung 4. Anstelle einer direkten Luftlinienverbindung vom Knoten 5 zum Knoten 3 können wir einen genaueren Streckenverlauf über den Knoten 4 darstellen. Knoten, die diese Funktion übernehmen, werden als **trackpos**-Knoten bezeichnet. Sie sind ebenfalls ***BUSSIM_POINT***-Objekte. Wir fassen Sie in der Menge $\mathcal{A}^{trackpos}$ zusammen.

Die Knoten-Menge $\mathcal{A}$ des Netzwerks ergibt sich nun aus der Vereinigung der vier vorgenannten Mengen, d.h. $\mathcal{A} := \mathcal{A}^{platform} \bigcup \mathcal{A}^{depot} \bigcup \mathcal{A}^{switch} \bigcup \mathcal{A}^{trackpos}$

Unter Verwendung dieser vier Knotenarten sowie der ***BUSSIM_STOP***-Objekte richten nachfolgend den Teil des Dresdner Straßenbahn-Netzwerks ein, in dem sich die Fahrzeuge der Linie 13 bewegen. Die nachfolgenden Ausführungen erklären einerseits, welche Daten für die Einrichtung dieser Infrastruktur in B-u-S-Sim benötigt werden. Zusätzlich geben wir Hinweise, wie bzw. aus welchen Quellen diese Informationen zu beschaffen sind. Ebenso erklären wir, wie diese Informationen dann in die Simulation gelangen.

6.2 Spezifikation der Haltestellen

Zunächst ist eine Liste der **Haltestellen** zusammenzustellen. An einer Haltestelle können Fahrgäste in ein Fahrzeug einsteigen oder aus einem Fahrzeug aussteigen. Entweder startet oder endet hier die Fahrt eines Fahrgastes oder der Fahrgast wechselt das Fahrzeug und setzt dann seinen Reiseweg fort (Umstieg). Zusätzlich sind Betriebshöfe als Haltestellen zu ermitteln, an denen Fahrzeuge ihre Fahrten beginnen, unterbrechen oder beenden können. Die wesentliche Funktion einer Haltestelle ist es, die anschließend einzurichtenden Haltepositionen zu gruppieren.

6.2.1 Zusammenstellen der Daten

Für die Hinterlegung der Haltestellen-Daten ist in B-u-S-Sim das Objekt ***BUSSIM_STOP*** vorgesehen. Zu jeder Haltestelle (d.h. zu jeder Instanz eines ***BUSSIM_STOP***-Objekts) ist eine fortlaufende eindeutige ID, ein Name sowie ein (betriebsinternes) Kürzel festzulegen und abzuspeichern.

Die Haltestellen und ihre Bezeichnungen können typischerweise den schematischen Linienverlaufsplänen (Übersichtsplänen) entnommen werden, die die Verkehrsbetriebe regelmäßig veröffentlichen. Viele Verkehrsbetriebe stellen ebenfalls ihre internen Haltestellen-Abkürzungen auf ihrer Website zur Verfügung.

Idealerweise erstellen Sie in einer Tabellenkalkulation eine Liste (**Haltestellen-Liste**) der Haltestellen-Namen und der zugehörigen Abkürzungen. Abschließend nummerieren Sie diese Liste fortlaufend mit den Werten 0,1,2,... Anhand dieser Nummer kann später jede Haltestelle eindeutig identifiziert werden (ID). Die lückenlose und bei 0 beginnende Haltestellen-Nummerierung ist wichtig und B-u-S-Sim testet bei Programmstart, ob diese Konvention der Nummernvergabe eingehalten wurde. Falls dies nicht der Fall ist, so startet die Simulation nicht.

Die Liste der Haltestellen der DVB-Linie 13 kann sehr einfach den öffentlich verfügbaren Netz- oder den Aushangfahrplänen entnommen werden. Als Kurzbezeichnungen (Kürzel) verwenden wir die offiziellen Kürzel, die auf der DVB-Website unter der `https://www.dvb.de/de-de/fahrplan/haltestellenauskunft/haltestellenkuerzel` bereitgestellt werden. Die Haltestelle mit der ID 43 werden wir nutzen, um den Betriebshof Reick darzustellen, von dem bzw. zu dem die Straßenbahn-Fahrzeuge der Linie 13 aus- bzw. einrücken. Tabelle 4 stellt die Haltestellen-Informationen zusammen.

6.2.2 Der C++ - Quellcode

Nachdem Sie die Daten über die Haltestellen zusammengestellt haben, müssen diese Daten dem Simulations-Programm bekannt gegeben werden. Hierzu müssen wir C++ - Quellcode hinzufügen. Die Einfügung des Quellcodes muss an der "richtigen" Stelle im vorhandenen Quellcode erfolgen. Die Hinterlegung der Daten zu den ***BUSSIM_STOP***-Objekten erfolgt in der Methode `BUSSIM_NETWORK::specify_stops`, die in der Datei `bussim_stops.cpp` enthalten ist.

Die dem Netzwerk-Konstruktor übergebene Anzahl N^{STOP} von ***BUSSIM_STOP***-Objekten `STOP[0],,...,STOP[`N^{STOP}`-1]` wird beim Programm-Start eingerichtet. Daher müssen für jedes ***BUSSIM_STOP***-Objekt `STOP[i]` lediglich noch die Werte für die Attribute `ID`, `STOP_NAME` und `STOP_SHORT_NAME` festgelegt werden. Hierfür steht die zum ***BUSSIM_STOP***-Objekt gehördende Methode `specify(int ID,string STOP_NAME,string STOP_SHORT_NAME)` zur Verfügung.

Mit dem Befehl `this->STOP[0].specify(0;"Prohlis Gleisschleife";"PRO");` wird das erste Haltestellen-Objekt mit den in Tabelle 4 aufgeführten Attributswerten versehen. Abb. 3 zeigt die vollständige Quellcode-Ergänzung. Dieser ist in den Quellcode der Methode `BUSSIM_NETWORK::specify_stops()` in der Datei `bussim_network.cpp` einzufügen.

Idealerweise erzeugen Sie den Quellcode automatisch. Hierzu können Sie die Daten aus Tab. 4 in eine Excel-Tabelle kopieren und darin zweilenweise den Quellcode mit Hilfe der `VERKETTEN`-Funktion automatisiert zusammenstellen. Dadurch reduziert sich der Tipp-Aufwand und gleichzeitig reduziert sich die Gefahr, dass sich Tippfehler einschleichen.

Sobald wir Veränderungen bei der Anzahl der in einem ***BUSSIM_NETWORK***-Objekt gespeicherten Objekt-Instanzen vornehmen, müssen wir dem Konstruktur der ***BUSSIM_NETWORK***-Instanz die Anzahl der zu speichernden Instanzen mitteilen. Bevor der ergänzte Quellcode des B-u-S-Sim -Projekts übersetzt wird, muss somit die benötigte Anzahl der ***BUSSIM_STOP***-Objekte dem Netzwerk-Konstruktor mitgeteilt werden. In der Datei `main.cpp` ist daher der Befehl `class BUSSIM_NETWORK NET(0,0,0,0,0,0,0,0);` durch den Befehl `class BUSSIM_NETWORK NET(46,0,0,0,0,0,0,0);` zu ersetzen. Anschließend kann das Projekt übersetzt und das erste Programm gestartet werden.

ID	Bezeichnung / Name	Kürzel
0	Prohlis Gleisschleife	PRO
1	Georg-Palitzsch-Straße	GPA
2	Jacob-Winter-Platz	JWP
3	Albert-Wolf-Platz	AWP
4	Trattendorfer Straße	TRS
5	Altreick	ARE
6	Hülßestraße	HUL
7	Lohrmannstraße	LOR
8	Wieckestraße	WIE
9	Otto-Dix-Ring	ODX
10	Eugen-Bracht-Straße	EBR
11	Cäcilienstraße	CAC
12	Hugo-Bürkner-Straße	HBU
13	Mockritzer Straße	MOK
14	Wasaplatz	WAS
15	S-Bf. Strehlen	SSR
16	Querallee	QUE
17	Zoo	ZOO
18	Lennéplatz	LEN
19	Georg-Arnhold-Bad	GEA
20	Straßburger Platz	SBP
21	St.-Benno-Gymnasium	GYM
22	Dürerstraße	DUR
23	Sachsenallee	SAA
24	Rosa-Luxemburg-Platz	RLP
25	Bautzner/Rothenburger Str.	BTZ
26	Görlitzer Straße	GLS
27	Alaunplatz	ALA
28	Bischofsweg	BIW
29	S-Bf. Bischofsplatz	SBI
30	Friedensstraße	FRI
31	Liststraße	LIS
32	Bürgerstraße	BGR
33	Rathaus Pieschen	RPN
34	Altpieschen	API
35	Mickten	MIC
36	Trachauer Straße	TAS
37	Brockwitzer Straße	BRW
38	An der Flutrinne	ADF
39	Sörnewitzer Straße	SRN
40	ElbePark	EPK
41	Washingtonstraße	WSH
42	Kaditz Riegelplatz	RIE
43	Betriebshof Reick	BH0

Tabelle 4: Liste der Haltestellen, die durch die Linie 13 bedient werden

```
1  this->STOP[0].specify(0,"Prohlis Gleisschleife ","PRO");
2  this->STOP[1].specify(1,"Georg-Palitzsch-Straße","GPA");
3  this->STOP[2].specify(2,"Jacob-Winter-Platz","JWP");
4  this->STOP[3].specify(3,"Albert-Wolf-Platz","AWP");
5  this->STOP[4].specify(4,"Trattendorfer Straße","TRS");
6  this->STOP[5].specify(5,"Altreick","ARE");
7  this->STOP[6].specify(6,"Hülßestraße","HUL");
8  this->STOP[7].specify(7,"Lohrmannstraße","LOR");
9  this->STOP[8].specify(8,"Wieckestraße","WIE");
10 this->STOP[9].specify(9,"Otto-Dix-Ring","ODX");
11 this->STOP[10].specify(10,"Eugen-Bracht-Straße","EBR");
12 this->STOP[11].specify(11,"Cäcilienstraße","CAC");
13 this->STOP[12].specify(12,"Hugo-Bürkner-Straße","HBU");
14 this->STOP[13].specify(13,"Mockritzer Straße","MOK");
15 this->STOP[14].specify(14,"Wasaplatz","WAS");
16 this->STOP[15].specify(15,"S-Bf. Strehlen","SSR");
17 this->STOP[16].specify(16,"Querallee","QUE");
18 this->STOP[17].specify(17,"Zoo","ZOO");
19 this->STOP[18].specify(18,"Lenneplatz","LEN");
20 this->STOP[19].specify(19,"Georg-Arnhold-Bad","GEA");
21 this->STOP[20].specify(20,"Straßburger Platz","SBP");
22 this->STOP[21].specify(21,"St.-Benno-Gymnasium","GYM");
23 this->STOP[22].specify(22,"Dürerstraße","DUR");
24 this->STOP[23].specify(23,"Sachsenallee","SAA");
25 this->STOP[24].specify(24,"Rosa-Luxemburg-Platz","RLP");
26 this->STOP[25].specify(25,"Bautzner/Rothenburger Str.","BTZ");
27 this->STOP[26].specify(26,"Görlitzer Straße","GLS");
28 this->STOP[27].specify(27,"Alaunplatz","ALA");
29 this->STOP[28].specify(28,"Bischofsweg","BIW");
30 this->STOP[29].specify(29,"S-Bf. Bischofsplatz","SBI");
31 this->STOP[30].specify(30,"Friedensstraße","FRI");
32 this->STOP[31].specify(31,"Liststraße","LIS");
33 this->STOP[32].specify(34,"Bürgerstraße","BGR");
34 this->STOP[33].specify(35,"Rathaus Pieschen","RPN");
35 this->STOP[34].specify(36,"Altpieschen","API");
36 this->STOP[35].specify(37,"Mickten","MIC");
37 this->STOP[36].specify(38,"Trachauer Straße","TAS");
38 this->STOP[37].specify(39,"Brockwitzer Straße","BRW");
39 this->STOP[38].specify(40,"An der Flutrinne","ADF");
40 this->STOP[39].specify(41,"Sörnewitzer Straße","SRN");
41 this->STOP[40].specify(42,"ElbePark","EPK");
42 this->STOP[41].specify(43,"Washingtonstraße","WSH");
43 this->STOP[42].specify(44,"Kaditz Riegelplatz","RIE");
44 this->STOP[43].specify(45,"Betriebshof Reick ","BH0 ");
```

Abbildung 3: Quelltext-Ergänzung zur Spezifikation der Haltestellen-Objekte

6.3 Einrichten der Netzwerkknoten (*BUSSIM_POINT*-Objekte)

Wie oben beschrieben, gibt es insgesamt vier verschiedene Knotenarten: **regulär**, **depot**, **switch** und **trackpos**. Die Informationen über die Orte, an denen sich diese Knoten befinden, sind zu ermitteln und es sind Namen/Bezeichnungen für jeden einzelnen Knoten festzulegen. Es hat sich als sinnvoll erwiesen, für jeden dieser Knotentypen nacheinander die Informationen zusammenzustellen. Anschließend ist für die gemeinsame Liste der zusätzliche C++-Quellcode zu erstellen und in das bestehende Projekt einzufügen. Hierbei ist zu berücksichtigen, dass es für jeden Knotentyp eine spezielle Spezifikations-Methode gibt, obwohl alle Knoten unabhängig vom Typ als ***BUSSIM_POINT***-Object definiert werden.

6.3.1 Datenquelle Openstreetmap

Die Lage eines Knotens wird durch die Angabe der Geo-Koordinaten Längengrad (x-Koordinate) und Breitengrad (y-Koordinate) bestimmt. Einige Verkehrsunternehmen stellen (auf Anfrage) Listen von Haltestellen etc. inkl. der Geo-Koordinaten zur Verfügung. Im Allgemeinen besteht jedoch kein Zugang zu kompletten und aufbereiteten bzw. weiterverarbeitbaren Listen der benötigten Orte. Daher müssen wir uns überlegen, wie wir möglichst einfach, nachvollzieh- und reproduzierbar die Geokoordinaten der Knoten des zu konstruierenden Netzwerks ermitteln können. Die genaue geographische Lage einer Halteposition kann z.B. aus Online-Karten wie `www.openstreetmap.de` oder `maps.google.com` entnommen werden.

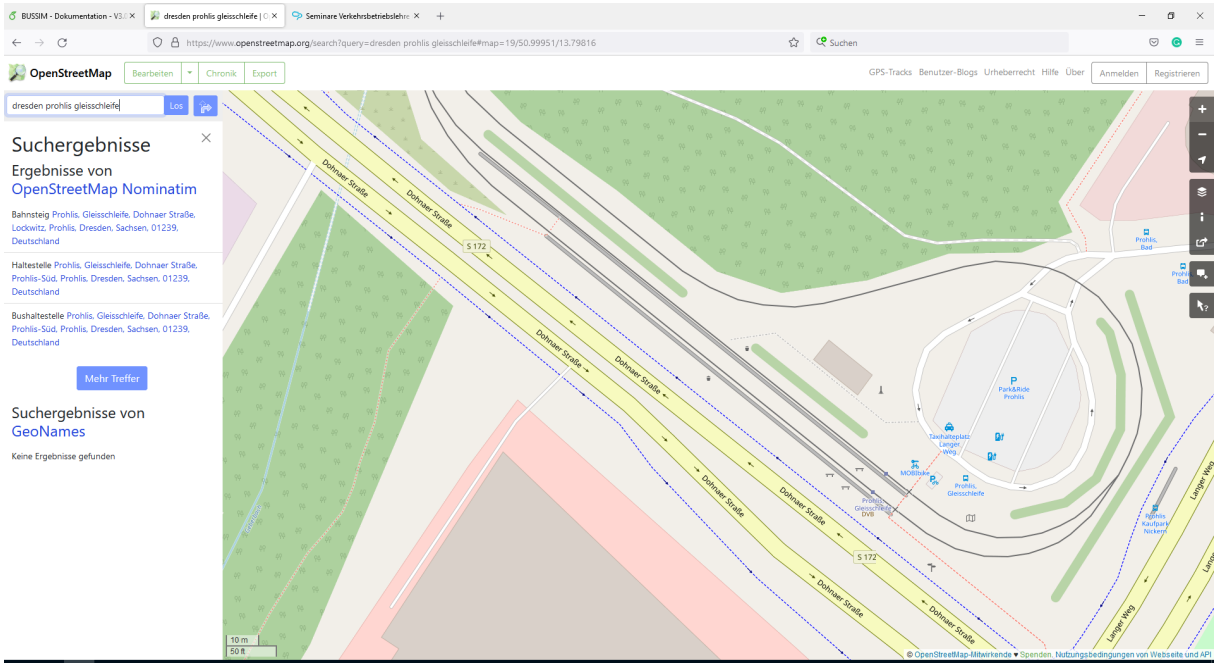

Abbildung 4: Darstellung der Gleisschleife Prohlis auf `www.openstreetmap.de`

Wir werden im weiteren Verlauf dieses Kapitels mit Daten aus dem Projekt openstreetmap arbeiten. Der Grundgedanke dieses Projekts ist es, eine zentrale Datenbank zur Verfügung zu stellen, an der alle interessierten Personen kostenlos relevante Geo-Informationen und Kartendaten ablegen und finden können. Informationen zu Straßenverläufen aber auch zum ÖPNV sind i.A. gut dokumentiert. Abb. 4 zeigt beispielsweise den Kartenausschnitt, in dem die Gleisschleife Prohlis dargestellt ist.
Es gibt verschiedene Möglichkeiten, spezielle Informationen aus Openstreetmap-Karten zu extrahieren. Wir werden die einfachste Variante nutzen. Möchte man beispielsweise die Geokoordinaten eines Ortes aus der dargestellten Karte erfahren, so positioniert man den Mauszeiger an dieser Position, klickt die rechte Maustaste und wählt die Option ”Adresse anzeigen“. Sofort wird die Kombination Breitengrad, Längengrad (y-Koordinate, x-Koordinate) links oben im Fenster angezeigt. Auf diesem Weg finden wir

beispielsweise heraus, dass die Weiche rechts in der Gleisschleife, bei der die beiden Gleise wieder zusammengeführt werden, die Geo-Koordinatenwerte 50.99923 (Breitengrad, y-Koordinate) und 13.79986 (Längengrad, x-Koordinate) besitzt.

6.3.2 Zusammenstellung der Informationen über die Netzwerk-Knoten

Wir beginnen mit der Zusammenstellung einer Liste der **Haltepositionen**. Eine Halteposition (oder eine **Plattform** bzw. ein **regulärer Knoten**) ist ein geographischer Ort, an dem ein Fahrzeug halten kann, um Fahrgäste ein- oder aussteigen zu lassen oder eine Pause einzulegen. Dicht beieinander liegende Haltepositionen bilden typischerweise eine **Haltestelle** und können in einem der soeben eingerichteten ***BUSSIM_STOP***-Objekte zusammengefasst werden. Im Allgemeinen gehören zu einer Haltestelle zwei oder mehr Haltepositionen. Bei Ringverkehren oder an Wendeschleifen kann es aber auch nur eine Halteposition je Haltestelle geben. Um diese Zusammengehörigkeit der Simulation mitzuteilen, müssen die Plattform-Eigenschaften spezifiziert und dann jeder Platform-Knoten einem der bereits eingerichteten ***BUSSIM_STOP***-Objekte zugeordnet werden.
Die Lage einer Halteposition wird durch die Angabe der Geo-Koordinaten Längengrad und Breitengrad bestimmt. Ebenso ist für jede Halteposition die ID der Haltestelle zu notieren, der diese Halteposition zugeordnet ist. Die genaue geographische Lage einer Halteposition kann z.B. aus Online-Karten wie `www.openstreetmap.de` oder `maps.google.com` entnommen werden.

ID	Längengrad	Breitengrad	Stop-ID	Bemerkung
–	13.79881	50.99919	0	Prohlis Gleisschleife / Abfahrt Außengleis
–	13.79783	51.00191	1	Georg-Palitzsch-Straße
–	13.7996	51.00556	2	Jacob-Winter-Platz
–	13.8034	51.00833	3	Albert-Wolf-Platz
–	13.80093	51.00998	4	Trattendorfer Straße
–	13.79565	51.01294	5	Altreick
–	13.79038	51.01367	6	Hülßestraße
–	13.7872	51.01559	7	Lohrmannstraße
–	13.78403	51.01757	8	Wieckestraße
–	13.78017	51.01966	9	Otto-Dix-Ring
–	13.77463	51.02254	10	Eugen-Bracht-Straße
–	13.77051	51.02469	11	Cäcilienstraße
–	13.76508	51.02266	12	Hugo-Bürkner-Straße
–	13.76247	51.02436	13	Mockritzer Straße
–	13.75979	51.02752	14	Wasaplatz
–	13.76147	51.03141	15	S-Bf. Strehlen
–	13.75786	51.03469	16	Querallee
–	13.75253	51.03681	17	Zoo
–	13.74736	51.03806	18	Lennéplatz
–	13.75186	51.04307	19	Georg-Arnhold-Bad
–	13.75461	51.04578	20	Straßburger Platz
–	13.75733	51.05015	21	St.-Benno-Gymnasium
–	13.75792	51.05184	22	Dürerstraße
–	13.75708	51.05489	23	Sachsenallee
–	13.75281	51.05959	24	Rosa-Luxemburg-Platz
–	13.75193	51.06267	25	Bautzner/Rothenburger Str.
–	13.75364	51.0659	26	Görlitzer Straße
–	13.75514	51.07006	27	Alaunplatz
–	13.75111	51.07128	28	Bischofsweg
–	13.74686	51.07203	29	S-Bf. Bischofsplatz

ID	Längengrad	Breitengrad	Stop-ID	Bemerkung
–	13.7406	51.07356	30	Friedensstraße
–	13.73289	51.07689	31	Liststraße
–	13.72572	51.07692	32	Bürgerstraße
–	13.72171	51.07741	33	Rathaus Pieschen
–	13.71765	51.07752	34	Altpieschen
–	13.7135	51.07945	35	Mickten
–	13.70789	51.07889	36	Trachauer Straße
–	13.70246	51.07791	37	Brockwitzer Straße
–	13.69869	51.07726	38	An der Flutrinne
–	13.69719	51.07988	39	Sörnewitzer Straße
–	13.69485	51.08192	40	ElbePark
–	13.69092	51.08239	41	Washingtonstraße
–	13.68699	51.08377	42	Kaditz Riegelplatz, Ankunftsgleis

Tabelle 5: Plattform-Informationen über die Linie 13 (Prohlis in Fahrtrichtung Riegelplatz)

Wir beginnen mit der Zusammenstellung der Haltepositionen (reguläre Knoten). Dafür betrachten wir zunächst eine Fahrt entlang des Verlaufs der Linie 13 von der Endhaltestelle Prohlis zur Enthaltestelle Riegelplatz (Tab. 5). An der ersten Plattform Prohlis Gleisschleife / Abfahrt Außengleis wartet ein Fahrtzeug typischerweise auf den Zeitpunkt, zu dem die nächste Fahrt in Richtung der Endhaltestelle Kaditz / Riegelplatz beginnt. Hier kann ggf. eine längere Wartezeit notwendig sein. Daher soll dieser reguläre Knoten ein Depot-Knoten werden.

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
–	13.79881	50.99919	0	Prohlis Gleisschleife / Abf. Außengleis	
–	13.79984	50.99923	-1	Weiche 1 - Ausfahrt prohlis	W1
–	13.79783	51.00191	1	Georg-Palitzsch-Straße	
–	13.7996	51.00556	2	Jacob-Winter-Platz	
–	13.8034	51.00833	3	Albert-Wolf-Platz	
–	13.8035	51.0084	-1	Weiche 2	W2
–	13.80342	51.00872	-1	Weiche 3	W3
–	13.80252	51.00915	-1	Weiche 4 - Einfahrt BH0	W4
–	13.80171	51.00956	-1	Weiche 5 - Ausfahrt BH0	W5
–	13.80093	51.00998	4	Trattendorfer Straße	
–	13.79565	51.01294	5	Altreick	
–	13.79038	51.01367	6	Hülßestraße	
–	13.7872	51.01559	7	Lohrmannstraße	
–	13.78403	51.01757	8	Wieckestraße	
–	13.78017	51.01966	9	Otto-Dix-Ring	
–	13.77463	51.02254	10	Eugen-Bracht-Straße	
–	13.77051	51.02469	11	Cäcilienstraße	
–	13.76553	51.02236	-1	Weiche 6 - Lockwitzer Straße	W6
–	13.76508	51.02266	12	Hugo-Bürkner-Straße	
–	13.76247	51.02436	13	Mockritzer Straße	
–	13.75979	51.02752	14	Wasaplatz	
–	13.76147	51.03141	15	S-Bf. Strehlen	
–	13.75786	51.03469	16	Querallee	
–	13.75253	51.03681	17	Zoo	
–	13.74852	51.03755	-1	Weiche 7 - Einfahrt Lenneplatz	W7
–	13.74736	51.03806	18	Lennéplatz	

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
–	13.74715	51.03816	-1	Weiche 8 - Ausfahrt Lenneplatz	W8
–	13.74713	51.03862	-1	Weiche 9 - Lennestrasse	W9
–	13.75186	51.04307	19	Georg-Arnhold-Bad	
–	13.75461	51.04578	20	Straßburger Platz	
–	13.75469	51.04585	-1	Weiche 10	W10
–	13.75733	51.05015	21	St.-Benno-Gymnasium	
–	13.75792	51.05184	22	Dürerstraße	
–	13.75839	51.05329	-1	Weiche 11 - Günzstraße	W11
–	13.75801	51.05405	-1	Weiche 12 - Günzplatz	W12
–	13.75708	51.05489	23	Sachsenallee	
–	13.75281	51.05959	24	Rosa-Luxemburg-Platz	
–	13.75276	51.05977	-1	Weiche 13	W13
–	13.75271	51.05994	-1	Weiche 14	W14
–	13.75193	51.06267	25	Bautzner/Rothenburger Str.	
–	13.75192	51.06272	-1	Weiche 15	W15
–	13.7519	51.06275	-1	Weiche 16	W16
–	13.75364	51.0659	26	Görlitzer Straße	
–	13.75378	51.06621	-1	Weiche 17	W17
–	13.75599	51.06961	-1	Weiche 18	W18
–	13.75514	51.07006	27	Alaunplatz	
–	13.75111	51.07128	28	Bischofsweg	
–	13.75035	51.07147	-1	Weiche 19	W19
–	13.74686	51.07203	29	S-Bf. Bischofsplatz	
–	13.7406	51.07356	30	Friedensstraße	
–	13.73459	51.07524	-1	Weiche 20 - Abzweig Fritz-Reuter-Str	W20
–	13.73355	51.07565	-1	Weiche 21	W21
–	13.73289	51.07689	31	Liststraße	
–	13.73284	51.07701	-1	Weiche 22 - Liststr. Ausfahrt	W22
–	13.7319	51.07734	-1	Weiche 23 - Einfahrt Harkortstr.	W23
–	13.72572	51.07692	32	Bürgerstraße	
–	13.72171	51.07741	33	Rathaus Pieschen	
–	13.71765	51.07752	34	Altpieschen	
–	13.71666	51.07757	-1	Weiche 24	W24
–	13.7147	51.07942	-1	Weiche 25 / Abzweig Sternstr.	W25
–	13.7135	51.07945	35	Mickten	
–	13.70789	51.07889	36	Trachauer Straße	
–	13.70246	51.07791	37	Brockwitzer Straße	
–	13.69869	51.07726	38	An der Flutrinne	
–	13.69719	51.07988	39	Sörnewitzer Straße	
–	13.69485	51.08192	40	ElbePark	
–	13.69092	51.08239	41	Washingtonstraße	
–	13.68699	51.08377	42	Kaditz Riegelplatz, Ankunftsgleis	

Tabelle 6: Haltepositionen und Weichen entlang der Linie 13 (Prohlis in Fahrtrichtung Riegelplatz)

Nachdem die Liste der Haltepositionen und die Liste der Depots entlang der Linienführung der Linie 13 von Prohlis nach Kaditz erfolgt ist, werden in einem zweiten Schritt die entlang dieser Route zu durchfahrenden Weichen erfasst. Tab. 6 zeigt die um die 25 vorgefundenen Weichen ergänzte Knotenliste, in der insgesamt 25 Weichen (switch-Knoten) hinzugefügt wurden (in roter Schrift angegeben). Es ist möglich (aber nicht notwendig), einen Weichenknoten einer Haltestelle zuzuordnen. Falls dies gewünscht

ist, dann ist der Wert in der Spalte `Stop-ID` entsprechend auf die ID des zugehörigen ***BUSSIM_STOP***-Objekts zu setzen. Falls keine Zuordnung gewünscht wird, dann ist der Wert -1 zu nutzen. Zusätzlich sollte auch für eine Weiche eine Bezeichnung (5. Spalte) und ein Kurzname definiert werden (6. Spalte), damit man die Weiche später schnell wiederfinden kann.

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
0	13.79881	50.99919	0	Prohlis Gleisschleife / Abf. Außengleis	
1	13.79917	50.99904	-1	Trackpos 1	TP1
2	13.7995	50.99903	-1	Trackpos 2	TP2
3	13.79974	50.99912	-1	Trackpos 3	TP3
4	13.79984	50.99923	-1	Weiche 1 - Ausfahrt prohlis	W1
5	13.79993	50.99936	-1	Trackpos 4	TP4
6	13.79991	50.99959	-1	Trackpos 5	TP5
7	13.79962	50.99978	-1	Trackpos 6	TP6
8	13.79934	50.99981	-1	Trackpos 7	TP7
9	13.79835	50.99968	-1	Trackpos 8	TP8
10	13.79801	50.99978	-1	Trackpos 9	TP9
11	13.79711	51.00025	-1	Trackpos 10	TP10
12	13.79695	51.00043	-1	Trackpos 11	TP11
13	13.797	51.00077	-1	Trackpos 12	TP12
14	13.79783	51.00191	1	Georg-Palitzsch-Straße	
15	13.79801	51.00219	-1	Trackpos 13	TP13
16	13.79812	51.00253	-1	Trackpos 14	TP14
17	13.79799	51.00394	-1	Trackpos 15	TP15
18	13.79811	51.00437	-1	Trackpos 16	TP16
19	13.79855	51.0048	-1	Trackpos 17	TP17
20	13.7996	51.00556	2	Jacob-Winter-Platz	
21	13.8034	51.00833	3	Albert-Wolf-Platz	
22	13.8035	51.0084	-1	Weiche 2	W2
23	13.80356	51.00852	-1	Trackpos 18	TP18
24	13.80354	51.00862	0.1	Trackpos 19	TP19
25	13.80342	51.00872	-1	Weiche 3	W3
26	13.80252	51.00915	-1	Weiche 4 - Einfahrt BH0	W4
27	13.80171	51.00956	-1	Weiche 5 - Ausfahrt BH0	W5
28	13.80093	51.00998	4	Trattendorfer Straße	
29	13.80054	51.01022	-1	Trackpos 20	TP20
30	13.80013	51.01054	-1	Trackpos 21	TP21
31	13.79887	51.01185	-1	Trackpos 22	TP22
32	13.79879	51.01205	-1	Trackpos 23	TP23
33	13.7988	51.01227	-1	Trackpos 24	TP24
34	13.79866	51.01243	-1	Trackpos 25	TP25
35	13.79838	51.01251	-1	Trackpos 26	TP26
36	13.79648	51.01272	-1	Trackpos 27	TP27
37	13.79611	51.0128	-1	Trackpos 28	TP28
38	13.79565	51.01294	5	Altreick	
39	13.79541	51.013	-1	Trackpos 29	TP29
40	13.79496	51.01307	-1	Trackpos 30	TP30
41	13.79423	51.0131	-1	Trackpos 31	TP31
42	13.79334	51.01306	-1	Trackpos 32	TP32
43	13.79258	51.01313	-1	Trackpos 33	TP33
44	13.79141	51.01334	-1	Trackpos 34	TP34

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
45	13.79038	51.01367	6	Hülßestraße	
46	13.78953	51.01398	-1	Trackpos 35	TP35
47	13.7886	51.01444	-1	Trackpos 36	TP36
48	13.78807	51.01483	-1	Trackpos 37	TP37
49	13.7872	51.01559	7	Lohrmannstraße	
50	13.78671	51.01602	-1	Trackpos 38	TP38
51	13.78612	51.0164	-1	Trackpos 39	TP39
52	13.78403	51.01757	8	Wieckestraße	
53	13.78017	51.01966	9	Otto-Dix-Ring	
54	13.77463	51.02254	10	Eugen-Bracht-Straße	
55	13.77051	51.02469	11	Cäcilienstraße	
56	13.77014	51.02487	-1	Trackpos 40	TP40
57	13.77	51.0249	-1	Trackpos 41	TP41
58	13.76983	51.0249	-1	Trackpos 42	TP42
59	13.76961	51.0248	-1	Trackpos 43	TP43
60	13.76704	51.02288	-1	Trackpos 44	TP44
61	13.76602	51.02236	-1	Trackpos 45	TP45
62	13.76602	51.02236	-1	Trackpos 46	TP46
63	13.76578	51.02231	-1	Trackpos 47	TP47
64	13.76564	51.02233	-1	Trackpos 48	TP48
65	13.76553	51.02236	-1	Weiche 6 - Lockwitzer Straße	W6
66	13.76508	51.02266	12	Hugo-Bürkner-Straße	
67	13.76247	51.02436	13	Mockritzer Straße	
68	13.75979	51.02752	14	Wasaplatz	
69	13.75955	51.0278	-1	Trackpos 49	TP49
70	13.75922	51.02834	-1	Trackpos 50	TP50
71	13.75927	51.0285	-1	Trackpos 51	TP51
72	13.76147	51.03141	15	S-Bf. Strehlen	
73	13.7625	51.03262	-1	Trackpos 52	TP52
74	13.76255	51.0327	-1	Trackpos 53	TP53
75	13.76252	51.0328	-1	Trackpos 54	TP54
76	13.76238	51.03288	-1	Trackpos 55	TP55
77	13.75786	51.03469	16	Querallee	
78	13.75253	51.03681	17	Zoo	
79	13.7522	51.03694	-1	Trackpos 56	TP56
80	13.7516	51.03719	-1	Trackpos 57	TP57
81	13.75085	51.03734	-1	Trackpos 58	TP58
82	13.75034	51.0374	-1	Trackpos 59	TP59
83	13.7489	51.03747	-1	Trackpos 60	TP60
84	13.7489	51.03747	-1	Trackpos 61	TP61
85	13.74852	51.03755	-1	Weiche 7 - Einfahrt Lenneplatz	W7
86	13.74736	51.03806	18	Lennéplatz	
87	13.74715	51.03816	-1	Weiche 8 - Ausfahrt Lenneplatz	W8
88	13.74705	51.03822	-1	Trackpos 62	TP62
89	13.74696	51.03833	-1	Trackpos 63	TP63
90	13.74698	51.03848	-1	Trackpos 64	TP64
91	13.74705	51.03856	-1	Trackpos 65	TP65
92	13.74713	51.03862	-1	Weiche 9 - Lennestrasse	W9
93	13.74776	51.039	-1	Trackpos 66	TP66
94	13.74794	51.03916	-1	Trackpos 67	TP67

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
95	13.75186	51.04307	19	Georg-Arnhold-Bad	
96	13.75461	51.04578	20	Straßburger Platz	
97	13.75469	51.04585	-1	Weiche 10	W10
98	13.75504	51.04623	-1	Trackpos 68	TP68
99	13.75522	51.04647	-1	Trackpos 69	TP69
100	13.7568	51.049	-1	Trackpos 70	TP70
101	13.75719	51.04974	-1	Trackpos 71	TP71
102	13.75733	51.05015	21	St.-Benno-Gymnasium	
103	13.75792	51.05184	22	Dürerstraße	
104	13.75807	51.05226	-1	Trackpos 72	TP72
105	13.75829	51.05284	-1	Trackpos 73	TP73
106	13.75836	51.05308	-1	Trackpos 74	TP74
107	13.75839	51.05329	-1	Weiche 11 - Günzstraße	W11
108	13.75834	51.05352	-1	Trackpos 75	TP75
109	13.75827	51.05372	-1	Trackpos 76	TP76
110	13.75815	51.0539	-1	Trackpos 77	TP77
111	13.75801	51.05405	-1	Weiche 12 - Günzplatz	W12
112	13.75708	51.05489	23	Sachsenallee	
113	13.75324	51.05842	-1	Trackpos 78	TP78
114	13.75315	51.05853	-1	Trackpos 79	TP79
115	13.75305	51.05869	-1	Trackpos 80	TP80
116	13.75281	51.05959	24	Rosa-Luxemburg-Platz	
117	13.75276	51.05977	-1	Weiche 13	W13
118	13.75271	51.05994	-1	Weiche 14	W14
119	13.75193	51.06267	25	Bautzner/Rothenburger Str.	
120	13.75192	51.06272	-1	Weiche 15	W15
121	13.7519	51.06275	-1	Weiche 16	W16
122	13.75184	51.06303	-1	Trackpos 81	TP81
123	13.75189	51.06322	-1	Trackpos 82	TP82
124	13.75364	51.0659	26	Görlitzer Straße	
125	13.75376	51.06609	-1	Trackpos 83	TP83
126	13.75378	51.06621	-1	Weiche 17	W17
127	13.75385	51.06646	-1	Trackpos 84	TP84
128	13.75599	51.06961	-1	Weiche 18	W18
129	13.75603	51.06967	-1	Trackpos 85	TP85
130	13.75601	51.06977	-1	Trackpos 86	TP86
131	13.75586	51.06984	-1	Trackpos 87	TP87
132	13.75514	51.07006	27	Alaunplatz	
133	13.75111	51.07128	28	Bischofsweg	
134	13.75083	51.07137	-1	Trackpos 88	TP88
135	13.75052	51.07144	-1	Trackpos 89	TP89
136	13.75035	51.07147	-1	Weiche 19	W19
137	13.74686	51.07203	29	S-Bf. Bischofsplatz	
138	13.74529	51.07227	-1	Trackpos 90	TP90
139	13.74496	51.07234	-1	Trackpos 91	TP91
140	13.7406	51.07356	30	Friedensstraße	
141	13.73459	51.07524	-1	Weiche 20 - Abzweig Fritz-Reuter-Str	W20
142	13.7339	51.07543	-1	Trackpos 92	TP92
143	13.73378	51.07548	-1	Trackpos 93	TP93
144	13.73363	51.07557	-1	Trackpos 94	TP94

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
145	13.73355	51.07565	-1	Weiche 21	W21
146	13.73289	51.07689	31	Liststraße	
147	13.73284	51.07701	-1	Weiche 22 - Liststr. Ausfahrt	W22
148	13.7327	51.07713	-1	Trackpos 95	TP95
149	13.73245	51.07725	-1	Trackpos 96	TP96
150	13.73218	51.07734	-1	Trackpos 97	TP97
151	13.7319	51.07734	-1	Weiche 23 - Einfahrt Harkortstr.	W23
152	13.73175	51.0773	-1	Trackpos 98	TP98
153	13.73155	51.07722	-1	Trackpos 99	TP99
154	13.72961	51.07639	-1	Trackpos 100	TP100
155	13.72952	51.07636	-1	Trackpos 101	TP101
156	13.72931	51.07634	-1	Trackpos 102	TP102
157	13.72912	51.07635	-1	Trackpos 103	TP103
158	13.72572	51.07692	32	Bürgerstraße	
159	13.72504	51.07701	-1	Trackpos 104	TP104
160	13.72346	51.07733	-1	Trackpos 105	TP105
161	13.72316	51.07736	-1	Trackpos 106	TP106
162	13.72171	51.07741	33	Rathaus Pieschen	
163	13.72143	51.07742	-1	Trackpos 107	TP107
164	13.71976	51.07738	-1	Trackpos 108	TP108
165	13.71958	51.07739	-1	Trackpos 109	TP109
166	13.71884	51.07745	-1	Trackpos 110	TP110
167	13.7183	51.07752	-1	Trackpos 111	TP111
168	13.71805	51.07754	-1	Trackpos 112	TP112
169	13.71765	51.07752	34	Altpieschen	
170	13.71707	51.0775	-1	Trackpos 113	TP113
171	13.71692	51.07751	-1	Trackpos 114	TP114
172	13.7168	51.07753	-1	Trackpos 115	TP115
173	13.71666	51.07757	-1	Weiche 24	W24
174	13.71619	51.07779	-1	Trackpos 116	TP116
175	13.71564	51.07813	-1	Trackpos 117	TP117
176	13.71531	51.07845	-1	Trackpos 118	TP118
177	13.7147	51.07942	-1	Weiche 25 / Abzweig Sternstr.	W25
178	13.71455	51.07952	-1	Trackpos 119	TP119
179	13.71449	51.07953	-1	Trackpos 120	TP120
180	13.71435	51.07954	-1	Trackpos 121	TP121
181	13.7135	51.07945	35	Mickten	
182	13.70789	51.07889	36	Trachauer Straße	
183	13.70686	51.0788	-1	Trackpos 122	TP122
184	13.7065	51.07874	-1	Trackpos 123	TP123
185	13.70246	51.07791	37	Brockwitzer Straße	
186	13.70005	51.07741	-1	Trackpos 124	TP124
187	13.69982	51.07737	-1	Trackpos 125	TP125
188	13.69869	51.07726	38	An der Flutrinne	
189	13.69805	51.07722	-1	Trackpos 126	TP126
190	13.69792	51.07723	-1	Trackpos 127	TP127
191	13.69775	51.07728	-1	Trackpos 128	TP128
192	13.69762	51.07737	-1	Trackpos 129	TP129
193	13.69752	51.07754	-1	Trackpos 130	TP130
194	13.69719	51.07988	39	Sörnewitzer Straße	

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
195	13.69689	51.08172	-1	Trackpos 131	TP131
196	13.69685	51.08181	-1	Trackpos 132	TP132
197	13.69676	51.08191	-1	Trackpos 133	TP133
198	13.69662	51.08197	-1	Trackpos 134	TP134
199	13.69646	51.08201	-1	Trackpos 135	TP135
200	13.69634	51.08202	-1	Trackpos 136	TP136
201	13.69485	51.08192	40	ElbePark	
202	13.69343	51.08184	-1	Trackpos 137	TP137
203	13.69315	51.08184	-1	Trackpos 138	TP138
204	13.69275	51.08187	-1	Trackpos 139	TP139
205	13.69233	51.08194	-1	Trackpos 140	TP140
206	13.69202	51.08203	-1	Trackpos 141	TP141
207	13.69092	51.08239	41	Washingtonstraße	
208	13.68699	51.08377	42	Kaditz Riegelplatz, Ankunftsgleis	

Tabelle 7: Haltepositionen, Weichen und Trackpos-Punkte Linie 13 (Prohlis in Fahrtrichtung Riegelplatz)

Nachdem die Liste der ***BUSSIM_POINT***-Objekte um die Weichen ergänzt wurde, muss nun erneut der Streckenverlauf von Prohlis bis zum Riegelplatz analysiert werden, um markante Richtungsänderungen / Kurvenverläufe durch die *trackpos*-***BUSSIM_POINT***-Objekte zu erfassen. Welche zusätzlichen trackpos-Knoten eingerichtet werden, ist jedem selber überlassen. Tab. 7 zeigt die entsprechend ergänzte Liste von ***BUSSIM_POINT***-Objekten. In blauer Farbe sind die zusätzlichen *trackpos*-***BUSSIM_POINT***-Objekte dargestellt. Damit ist die Liste der zur Darstellung des Strecknverlaufs benötigten Graphen-Knoten vollständig und wir können die Instanzen durchgängig und bei 0 beginnend nummerieren (1. Spalte). Insgesamt benötigen wir 209 ***BUSSIM_POINT***-Objekte zur Darstellung.

6.3.3 Quellcode-Ergänzung für die *BUSSIM_POINT*-Spezifikation

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`.
Abb. 5 stellt die Quellcode-Erweiterung vor, mit der die ***BUSSIM_POINT***-Objekte entlang der Strecke von Prohlis nach Kaditz definiert werden. Damit die insgesamt 209 ***BUSSIM_POINT***-Objekte hier befüllt werden können, müssen sie zunächst bei der Einrichtung des übergeordneten Netzwerk-Objekts eingerichtet werden. Hierzu ist erneut eine ANpassung des zugehörigen Konstruktors des ***BUSSIM_NETWORK***-Objekts in `main.cpp` notwendig. Dafür muss der Befehl `class BUSSIM_NETWORK NET(44,0,0,0,0,0,0,0);` zu `class BUSSIM_NETWORK NET(44,209,0,0,0,0,0,0);` geändert werden.
Aus den angegebenen Geo-Koordinaten kann B-u-S-Sim dann aus der paarweisen Differenz von Längen- bzw. Breitengrad zwischen je zwei Knoten-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 $d_y(H_1; H_2) := 111km \cdot (b_1 - b_2)$.
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

```
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   this->PT[0].specify_depot(13.79881,50.99919,0,this);
9   this->PT[1].specify_trackpos(13.79917,50.99904,-1,"Trackpos 1","TP1",this)
        ;
10  this->PT[2].specify_trackpos(13.7995,50.99903,-1,"Trackpos 2","TP2",this);
11  this->PT[3].specify_trackpos(13.79974,50.99912,-1,"Trackpos 3","TP3",this)
        ;
12  this->PT[4].specify_switch(13.79984,50.99923,-1,"Weiche 1 - Ausfahrt
        prohlis","W1",this);
13  this->PT[5].specify_trackpos(13.79993,50.99936,-1,"Trackpos 4","TP4",this)
        ;
14  this->PT[6].specify_trackpos(13.79991,50.99959,-1,"Trackpos 5","TP5",this)
        ;
15  this->PT[7].specify_trackpos(13.79962,50.99978,-1,"Trackpos 6","TP6",this)
        ;
16  this->PT[8].specify_trackpos(13.79934,50.99981,-1,"Trackpos 7","TP7",this)
        ;
17  this->PT[9].specify_trackpos(13.79835,50.99968,-1,"Trackpos 8","TP8",this)
        ;
18  this->PT[10].specify_trackpos(13.79801,50.99978,-1,"Trackpos 9","TP9",this
        );
19  this->PT[11].specify_trackpos(13.79711,51.00025,-1,"Trackpos 10","TP10",
        this);
20  this->PT[12].specify_trackpos(13.79695,51.00043,-1,"Trackpos 11","TP11",
        this);
21  this->PT[13].specify_trackpos(13.797,51.00077,-1,"Trackpos 12","TP12",this
        );
22  this->PT[14].specify_regular(13.79783,51.00191,1,this);
23  this->PT[15].specify_trackpos(13.79801,51.00219,-1,"Trackpos 13","TP13",
        this);
24  this->PT[16].specify_trackpos(13.79812,51.00253,-1,"Trackpos 14","TP14",
        this);
25  this->PT[17].specify_trackpos(13.79799,51.00394,-1,"Trackpos 15","TP15",
        this);
26  this->PT[18].specify_trackpos(13.79811,51.00437,-1,"Trackpos 16","TP16",
        this);
27  this->PT[19].specify_trackpos(13.79855,51.0048,-1,"Trackpos 17","TP17",
        this);
28  this->PT[20].specify_regular(13.7996,51.00556,2,this);
29  this->PT[21].specify_regular(13.8034,51.00833,3,this);
30  this->PT[22].specify_switch(13.8035,51.0084,-1,"Weiche 2 ","W2",this);
31  this->PT[23].specify_trackpos(13.80356,51.00852,-1,"Trackpos 18","TP18",
        this);
32  this->PT[24].specify_trackpos(13.80354,51.00862,0.1,"Trackpos 19","TP19",
        this);
33  this->PT[25].specify_switch(13.80342,51.00872,-1,"Weiche 3","W3",this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
34      this->PT[26].specify_switch(13.80252, 51.00915, -1, "Weiche 4 -
            Einfahrt BH0", "W4", this);
35  this->PT[27].specify_switch(13.80171, 51.00956, -1, "Weiche 5 - Ausfahrt
        BH0", "W5", this);
36  this->PT[28].specify_regular(13.80093, 51.00998, 4, this);
37  this->PT[29].specify_trackpos(13.80054, 51.01022, -1, "Trackpos 20", "
        TP20", this);
38  this->PT[30].specify_trackpos(13.80013, 51.01054, -1, "Trackpos 21", "
        TP21", this);
39  this->PT[31].specify_trackpos(13.79887, 51.01185, -1, "Trackpos 22", "
        TP22", this);
40  this->PT[32].specify_trackpos(13.79879, 51.01205, -1, "Trackpos 23", "
        TP23", this);
41  this->PT[33].specify_trackpos(13.7988, 51.01227, -1, "Trackpos 24", "TP24
        ", this);
42  this->PT[34].specify_trackpos(13.79866, 51.01243, -1, "Trackpos 25", "
        TP25", this);
43  this->PT[35].specify_trackpos(13.79838, 51.01251, -1, "Trackpos 26", "
        TP26", this);
44  this->PT[36].specify_trackpos(13.79648, 51.01272, -1, "Trackpos 27", "
        TP27", this);
45  this->PT[37].specify_trackpos(13.79611, 51.0128, -1, "Trackpos 28", "TP28
        ", this);
46  this->PT[38].specify_regular(13.79565, 51.01294, 5, this);
47  this->PT[39].specify_trackpos(13.79541, 51.013, -1, "Trackpos 29", "TP29"
        , this);
48  this->PT[40].specify_trackpos(13.79496, 51.01307, -1, "Trackpos 30", "
        TP30", this);
49  this->PT[41].specify_trackpos(13.79423, 51.0131, -1, "Trackpos 31", "TP31
        ", this);
50  this->PT[42].specify_trackpos(13.79334, 51.01306, -1, "Trackpos 32", "
        TP32", this);
51  this->PT[43].specify_trackpos(13.79258, 51.01313, -1, "Trackpos 33", "
        TP33", this);
52  this->PT[44].specify_trackpos(13.79141, 51.01334, -1, "Trackpos 34", "
        TP34", this);
53  this->PT[45].specify_regular(13.79038, 51.01367, 6, this);
54  this->PT[46].specify_trackpos(13.78953, 51.01398, -1, "Trackpos 35", "
        TP35", this);
55  this->PT[47].specify_trackpos(13.7886, 51.01444, -1, "Trackpos 36", "TP36
        ", this);
56  this->PT[48].specify_trackpos(13.78807, 51.01483, -1, "Trackpos 37", "
        TP37", this);
57  this->PT[49].specify_regular(13.7872, 51.01559, 7, this);
58  this->PT[50].specify_trackpos(13.78671, 51.01602, -1, "Trackpos 38", "
        TP38", this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
59  this->PT[51].specify_trackpos(13.78612, 51.0164, -1, "Trackpos 39", "TP39
        ", this);
60  this->PT[52].specify_regular(13.78403, 51.01757, 8, this);
61  this->PT[53].specify_regular(13.78017, 51.01966, 9, this);
62  this->PT[54].specify_regular(13.77463, 51.02254, 10, this);
63  this->PT[55].specify_regular(13.77051, 51.02469, 11, this);
64  this->PT[56].specify_trackpos(13.77014, 51.02487, -1, "Trackpos 40", "
        TP40", this);
65  this->PT[57].specify_trackpos(13.77, 51.0249, -1, "Trackpos 41", "TP41",
        this);
66  this->PT[58].specify_trackpos(13.76983, 51.0249, -1, "Trackpos 42", "TP42
        ", this);
67  this->PT[59].specify_trackpos(13.76961, 51.0248, -1, "Trackpos 43", "TP43
        ", this);
68  this->PT[60].specify_trackpos(13.76704, 51.02288, -1, "Trackpos 44", "
        TP44", this);
69  this->PT[61].specify_trackpos(13.76602, 51.02236, -1, "Trackpos 45", "
        TP45", this);
70  this->PT[62].specify_trackpos(13.76602, 51.02236, -1, "Trackpos 46", "
        TP46", this);
71  this->PT[63].specify_trackpos(13.76578, 51.02231, -1, "Trackpos 47", "
        TP47", this);
72  this->PT[64].specify_trackpos(13.76564, 51.02233, -1, "Trackpos 48", "
        TP48", this);
73  this->PT[65].specify_switch(13.76553, 51.02236, -1, "Weiche 6 - Lockwitzer
         Straße", "W6", this);
74  this->PT[66].specify_regular(13.76508, 51.02266, 12, this);
75  this->PT[67].specify_regular(13.76247, 51.02436, 13, this);
76  this->PT[68].specify_regular(13.75979, 51.02752, 14, this);
77  this->PT[69].specify_trackpos(13.75955, 51.0278, -1, "Trackpos 49", "TP49
        ", this);
78  this->PT[70].specify_trackpos(13.75922, 51.02834, -1, "Trackpos 50", "
        TP50", this);
79  this->PT[71].specify_trackpos(13.75927, 51.0285, -1, "Trackpos 51", "TP51
        ", this);
80  this->PT[72].specify_regular(13.76147, 51.03141, 15, this);
81  this->PT[73].specify_trackpos(13.7625, 51.03262, -1, "Trackpos 52", "TP52
        ", this);
82  this->PT[74].specify_trackpos(13.76255, 51.0327, -1, "Trackpos 53", "TP53
        ", this);
83  this->PT[75].specify_trackpos(13.76252, 51.0328, -1, "Trackpos 54", "TP54
        ", this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
84  this->PT[76].specify_trackpos(13.76238, 51.03288, -1, "Trackpos 55", "
        TP55", this);
85  this->PT[77].specify_regular(13.75786, 51.03469, 16, this);
86  this->PT[78].specify_regular(13.75253, 51.03681, 17, this);
87  this->PT[79].specify_trackpos(13.7522, 51.03694, -1, "Trackpos 56", "TP56
        ", this);
88  this->PT[80].specify_trackpos(13.7516, 51.03719, -1, "Trackpos 57", "TP57
        ", this);
89  this->PT[81].specify_trackpos(13.75085, 51.03734, -1, "Trackpos 58", "
        TP58", this);
90  this->PT[82].specify_trackpos(13.75034, 51.0374, -1, "Trackpos 59", "TP59
        ", this);
91  this->PT[83].specify_trackpos(13.7489, 51.03747, -1, "Trackpos 60", "TP60
        ", this);
92  this->PT[84].specify_trackpos(13.7489, 51.03747, -1, "Trackpos 61", "TP61
        ", this);
93  this->PT[85].specify_switch(13.74852, 51.03755, -1, "Weiche 7 - Einfahrt
        Lenneplatz", "W7", this);
94  this->PT[86].specify_regular(13.74736, 51.03806, 18, this);
95  this->PT[87].specify_switch(13.74715, 51.03816, -1, "Weiche 8 - Ausfahrt
        Lenneplatz", "W8", this);
96  this->PT[88].specify_trackpos(13.74705, 51.03822, -1, "Trackpos 62", "
        TP62", this);
97  this->PT[89].specify_trackpos(13.74696, 51.03833, -1, "Trackpos 63", "
        TP63", this);
98  this->PT[90].specify_trackpos(13.74698, 51.03848, -1, "Trackpos 64", "
        TP64", this);
99  this->PT[91].specify_trackpos(13.74705, 51.03856, -1, "Trackpos 65", "
        TP65", this);
100 this->PT[92].specify_switch(13.74713, 51.03862, -1, "Weiche 9 -
        Lennestrasse", "W9", this);
101 this->PT[93].specify_trackpos(13.74776, 51.039, -1, "Trackpos 66", "TP66"
        , this);
102 this->PT[94].specify_trackpos(13.74794, 51.03916, -1, "Trackpos 67", "
        TP67", this);
103 this->PT[95].specify_regular(13.75186, 51.04307, 19, this);
104 this->PT[96].specify_regular(13.75461, 51.04578, 20, this);
105 this->PT[97].specify_switch(13.75469, 51.04585, -1, "Weiche 10", "W10",
        this);
106 this->PT[98].specify_trackpos(13.75504, 51.04623, -1, "Trackpos 68", "
        TP68", this);
107 this->PT[99].specify_trackpos(13.75522, 51.04647, -1, "Trackpos 69", "
        TP69", this);
108 this->PT[100].specify_trackpos(13.7568, 51.049, -1, "Trackpos 70", "TP70"
        , this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
109  this->PT[101].specify_trackpos(13.75719, 51.04974, -1, "Trackpos 71", "
         TP71", this);
110  this->PT[102].specify_regular(13.75733, 51.05015, 21, this);
111  this->PT[103].specify_regular(13.75792, 51.05184, 22, this);
112  this->PT[104].specify_trackpos(13.75807, 51.05226, -1, "Trackpos 72", "
         TP72", this);
113  this->PT[105].specify_trackpos(13.75829, 51.05284, -1, "Trackpos 73", "
         TP73", this);
114  this->PT[106].specify_trackpos(13.75836, 51.05308, -1, "Trackpos 74", "
         TP74", this);
115  this->PT[107].specify_switch(13.75839, 51.05329, -1, "Weiche 11 - Günzstra
         ße", "W11", this);
116  this->PT[108].specify_trackpos(13.75834, 51.05352, -1, "Trackpos 75", "
         TP75", this);
117  this->PT[109].specify_trackpos(13.75827, 51.05372, -1, "Trackpos 76", "
         TP76", this);
118  this->PT[110].specify_trackpos(13.75815, 51.0539, -1, "Trackpos 77", "
         TP77", this);
119  this->PT[111].specify_switch(13.75801, 51.05405, -1, "Weiche 12 - Gü
         nzplatz", "W12", this);
120  this->PT[112].specify_regular(13.75708, 51.05489, 23, this);
121  this->PT[113].specify_trackpos(13.75324, 51.05842, -1, "Trackpos 78", "
         TP78", this);
122  this->PT[114].specify_trackpos(13.75315, 51.05853, -1, "Trackpos 79", "
         TP79", this);
123  this->PT[115].specify_trackpos(13.75305, 51.05869, -1, "Trackpos 80", "
         TP80", this);
124  this->PT[116].specify_regular(13.75281, 51.05959, 24, this);
125  this->PT[117].specify_switch(13.75276, 51.05977, -1, "Weiche 13", "W13",
         this);
126  this->PT[118].specify_switch(13.75271, 51.05994, -1, "Weiche 14", "W14",
         this);
127  this->PT[119].specify_regular(13.75193, 51.06267, 25, this);
128  this->PT[120].specify_switch(13.75192, 51.06272, -1, "Weiche 15", "W15",
         this);
129  this->PT[121].specify_switch(13.7519, 51.06275, -1, "Weiche 16", "W16",
         this);
130  this->PT[122].specify_trackpos(13.75184, 51.06303, -1, "Trackpos 81", "
         TP81", this);
131  this->PT[123].specify_trackpos(13.75189, 51.06322, -1, "Trackpos 82", "
         TP82", this);
132  this->PT[124].specify_regular(13.75364, 51.0659, 26, this);
133  this->PT[125].specify_trackpos(13.75376, 51.06609, -1, "Trackpos 83", "
         TP83", this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
134  this->PT[126].specify_switch(13.75378, 51.06621, -1, "Weiche 17",  "W17",
         this);
135  this->PT[127].specify_trackpos(13.75385, 51.06646, -1, "Trackpos 84",  "
         TP84", this);
136  this->PT[128].specify_switch(13.75599, 51.06961, -1, "Weiche 18",  "W18",
         this);
137  this->PT[129].specify_trackpos(13.75603, 51.06967, -1, "Trackpos 85",  "
         TP85", this);
138  this->PT[130].specify_trackpos(13.75601, 51.06977, -1, "Trackpos 86",  "
         TP86", this);
139  this->PT[131].specify_trackpos(13.75586, 51.06984, -1, "Trackpos 87",  "
         TP87", this);
140  this->PT[132].specify_regular(13.75514, 51.07006, 27,  this);
141  this->PT[133].specify_regular(13.75111, 51.07128, 28,  this);
142  this->PT[134].specify_trackpos(13.75083, 51.07137, -1, "Trackpos 88",  "
         TP88", this);
143  this->PT[135].specify_trackpos(13.75052, 51.07144, -1, "Trackpos 89",  "
         TP89", this);
144  this->PT[136].specify_switch(13.75035, 51.07147, -1, "Weiche 19",  "W19",
         this);
145  this->PT[137].specify_regular(13.74686, 51.07203, 29,  this);
146  this->PT[138].specify_trackpos(13.74529, 51.07227, -1, "Trackpos 90",  "
         TP90", this);
147  this->PT[139].specify_trackpos(13.74496, 51.07234, -1, "Trackpos 91",  "
         TP91", this);
148  this->PT[140].specify_regular(13.7406, 51.07356, 30,  this);
149  this->PT[141].specify_switch(13.73459, 51.07524, -1, "Weiche 20 - Abzweig
         Fritz-Reuter-Str",  "W20", this);
150  this->PT[142].specify_trackpos(13.7339, 51.07543, -1, "Trackpos 92",  "
         TP92", this);
151  this->PT[143].specify_trackpos(13.73378, 51.07548, -1, "Trackpos 93",  "
         TP93", this);
152  this->PT[144].specify_trackpos(13.73363, 51.07557, -1, "Trackpos 94",  "
         TP94", this);
153  this->PT[145].specify_switch(13.73355, 51.07565, -1, "Weiche 21",  "W21",
         this);
154  this->PT[146].specify_regular(13.73289, 51.07689, 31,  this);
155  this->PT[147].specify_switch(13.73284, 51.07701, -1, "Weiche 22 - Liststr.
          Ausfahrt",  "W22", this);
156  this->PT[148].specify_trackpos(13.7327, 51.07713, -1, "Trackpos 95",  "
         TP95", this);
157  this->PT[149].specify_trackpos(13.73245, 51.07725, -1, "Trackpos 96",  "
         TP96", this);
158  this->PT[150].specify_trackpos(13.73218, 51.07734, -1, "Trackpos 97",  "
         TP97", this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
159  this->PT[151].specify_switch(13.7319, 51.07734, -1, "Weiche 23 - Einfahrt
         Harkortstr.", "W23", this);
160  this->PT[152].specify_trackpos(13.73175, 51.0773, -1, "Trackpos 98", "
         TP98", this);
161  this->PT[153].specify_trackpos(13.73155, 51.07722, -1, "Trackpos 99", "
         TP99", this);
162  this->PT[154].specify_trackpos(13.72961, 51.07639, -1, "Trackpos 100", "
         TP100", this);
163  this->PT[155].specify_trackpos(13.72952, 51.07636, -1, "Trackpos 101", "
         TP101", this);
164  this->PT[156].specify_trackpos(13.72931, 51.07634, -1, "Trackpos 102", "
         TP102", this);
165  this->PT[157].specify_trackpos(13.72912, 51.07635, -1, "Trackpos 103", "
         TP103", this);
166  this->PT[158].specify_regular(13.72572, 51.07692, 32, this);
167  this->PT[159].specify_trackpos(13.72504, 51.07701, -1, "Trackpos 104", "
         TP104", this);
168  this->PT[160].specify_trackpos(13.72346, 51.07733, -1, "Trackpos 105", "
         TP105", this);
169  this->PT[161].specify_trackpos(13.72316, 51.07736, -1, "Trackpos 106", "
         TP106", this);
170  this->PT[162].specify_regular(13.72171, 51.07741, 33, this);
171  this->PT[163].specify_trackpos(13.72143, 51.07742, -1, "Trackpos 107", "
         TP107", this);
172  this->PT[164].specify_trackpos(13.71976, 51.07738, -1, "Trackpos 108", "
         TP108", this);
173  this->PT[165].specify_trackpos(13.71958, 51.07739, -1, "Trackpos 109", "
         TP109", this);
174  this->PT[166].specify_trackpos(13.71884, 51.07745, -1, "Trackpos 110", "
         TP110", this);
175  this->PT[167].specify_trackpos(13.7183, 51.07752, -1, "Trackpos 111", "
         TP111", this);
176  this->PT[168].specify_trackpos(13.71805, 51.07754, -1, "Trackpos 112", "
         TP112", this);
177  this->PT[169].specify_regular(13.71765, 51.07752, 34, this);
178  this->PT[170].specify_trackpos(13.71707, 51.0775, -1, "Trackpos 113", "
         TP113", this);
179  this->PT[171].specify_trackpos(13.71692, 51.07751, -1, "Trackpos 114", "
         TP114", this);
180  this->PT[172].specify_trackpos(13.7168, 51.07753, -1, "Trackpos 115", "
         TP115", this);
181  this->PT[173].specify_switch(13.71666, 51.07757, -1, "Weiche 24", "W24",
         this);
182  this->PT[174].specify_trackpos(13.71619, 51.07779, -1, "Trackpos 116", "
         TP116", this);
183  this->PT[175].specify_trackpos(13.71564, 51.07813, -1, "Trackpos 117", "
         TP117", this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
184  this->PT[176].specify_trackpos(13.71531, 51.07845, -1, "Trackpos 118", "
         TP118", this);
185  this->PT[177].specify_switch(13.7147, 51.07942, -1, "Weiche 25 / Abzweig
         Sternstr.", "W25", this);
186  this->PT[178].specify_trackpos(13.71455, 51.07952, -1, "Trackpos 119", "
         TP119", this);
187  this->PT[179].specify_trackpos(13.71449, 51.07953, -1, "Trackpos 120", "
         TP120", this);
188  this->PT[180].specify_trackpos(13.71435, 51.07954, -1, "Trackpos 121", "
         TP121", this);
189  this->PT[181].specify_regular(13.7135, 51.07945, 35, this);
190  this->PT[182].specify_regular(13.70789, 51.07889, 36, this);
191  this->PT[183].specify_trackpos(13.70686, 51.0788, -1, "Trackpos 122", "
         TP122", this);
192  this->PT[184].specify_trackpos(13.7065, 51.07874, -1, "Trackpos 123", "
         TP123", this);
193  this->PT[185].specify_regular(13.70246, 51.07791, 37, this);
194  this->PT[186].specify_trackpos(13.70005, 51.07741, -1, "Trackpos 124", "
         TP124", this);
195  this->PT[187].specify_trackpos(13.69982, 51.07737, -1, "Trackpos 125", "
         TP125", this);
196  this->PT[188].specify_regular(13.69869, 51.07726, 38, this);
197  this->PT[189].specify_trackpos(13.69805, 51.07722, -1, "Trackpos 126", "
         TP126", this);
198  this->PT[190].specify_trackpos(13.69792, 51.07723, -1, "Trackpos 127", "
         TP127", this);
199  this->PT[191].specify_trackpos(13.69775, 51.07728, -1, "Trackpos 128", "
         TP128", this);
200  this->PT[192].specify_trackpos(13.69762, 51.07737, -1, "Trackpos 129", "
         TP129", this);
201  this->PT[193].specify_trackpos(13.69752, 51.07754, -1, "Trackpos 130", "
         TP130", this);
202  this->PT[194].specify_regular(13.69719, 51.07988, 39, this);
203  this->PT[195].specify_trackpos(13.69689, 51.08172, -1, "Trackpos 131", "
         TP131", this);
204  this->PT[196].specify_trackpos(13.69685, 51.08181, -1, "Trackpos 132", "
         TP132", this);
205  this->PT[197].specify_trackpos(13.69676, 51.08191, -1, "Trackpos 133", "
         TP133", this);
206  this->PT[198].specify_trackpos(13.69662, 51.08197, -1, "Trackpos 134", "
         TP134", this);
207  this->PT[199].specify_trackpos(13.69646, 51.08201, -1, "Trackpos 135", "
         TP135", this);
208  this->PT[200].specify_trackpos(13.69634, 51.08202, -1, "Trackpos 136", "
         TP136", this);
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte

```
209   this->PT[201].specify_regular(13.69485, 51.08192, 40,  this);
210   this->PT[202].specify_trackpos(13.69343, 51.08184, -1, "Trackpos 137",  "
          TP137", this);
211   this->PT[203].specify_trackpos(13.69315, 51.08184, -1, "Trackpos 138",  "
          TP138", this);
212   this->PT[204].specify_trackpos(13.69275, 51.08187, -1, "Trackpos 139",  "
          TP139", this);
213   this->PT[205].specify_trackpos(13.69233, 51.08194, -1, "Trackpos 140",  "
          TP140", this);
214   this->PT[206].specify_trackpos(13.69202, 51.08203, -1, "Trackpos 141",  "
          TP141", this);
215   this->PT[207].specify_regular(13.69092, 51.08239, 41,  this);
216   this->PT[208].specify_regular(13.68699, 51.08377, 42,  this);
217
218      // points specification ends here
219 }
```

Abbildung 5: Quellcode zur Spezifikation der ***BUSSIM_POINT***-Objekte für die Strecken von Prohlis zum Riegelplatz

`BUSSIM_LAT_DIST_KM` auf die jeweils für die dargestellte Region eingestellt werden. Standardmäßig werden die vorgenannten Werte verwendet.
Nach dem Compilieren des ergänzten Projekts und dem Start erhalten Sie die in Abb. 6 angezeigte Darstellung, wenn Sie nach dem Start die rechte Maustaste betätigen und im sich öffnenden Menü die Optionen `show platform names`, `show switches and trackpoint names` und `show switches and other trackpoints` auswählen. Angezeigt werden nun alle hinterlegten ***BUSSIM_POINT***-Objekte.
Sie können mit den Tasten 7 und 8 in die Darstellung hinein bzw. heraus zommen. Nach dem Sie in die Darstellung gezoomt haben, können Sie unter Verwendung der Tasten 1-4 den Fokus (also die Lupe) bewegen. Abb. 7 zeigt beispielhaft den Zoom auf den Streckenbereich zwischen den Platformen Liststraße (ID=146) und Bürgerstraße (ID=156). Weichen-Knoten werden durch ein Dreieck-Symbol repräsentiert und Kreise stellen trackpos-Knoten dar.

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
209	13.68724	51.08341	42	Kaditz Riegelplatz, Abf. Außengleis	
210	13.68742	51.08343	-1	Trackpos 142	TP142
211	13.68768	51.08343	-1	Trackpos 143	TP143
212	13.68779	51.08341	-1	Weiche 206	W26
213	13.69121	51.08226	41	Washingtonstraße	
214	13.69201	51.08199	-1	Trackpos 144	TP144
215	13.69235	51.08191	-1	Trackpos 145	TP145
216	13.69276	51.08183	-1	Trackpos 146	TP146
217	13.69316	51.0818	-1	Trackpos 147	TP147
218	13.69344	51.08181	-1	Trackpos 148	TP148
219	13.69579	51.08195	40	ElbePark	
220	13.69634	51.08198	-1	Trackpos 149	TP149
221	13.69646	51.08198	-1	Trackpos 150	TP150
222	13.69657	51.08195	-1	Trackpos 151	TP151
223	13.69669	51.0819	-1	Trackpos 152	TP152
224	13.69679	51.08182	-1	Trackpos 153	TP153
225	13.69683	51.08172	-1	Trackpos 154	TP154

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
226	13.69718	51.0795	39	Sörnewitzer Straße	
227	13.69747	51.07753	-1	Trackpos 155	TP155
228	13.69754	51.07739	-1	Trackpos 156	TP156
229	13.69766	51.07729	-1	Trackpos 157	TP157
230	13.69781	51.07723	-1	Trackpos 158	TP158
231	13.69793	51.07721	-1	Trackpos 159	TP159
232	13.69805	51.0772	-1	Trackpos 160	TP160
233	13.69825	51.0772	-1	Trackpos 161	TP161
234	13.6989	51.07725	38	An der Flutrinne	
235	13.70001	51.07738	-1	Trackpos 162	TP162
236	13.7033	51.07805	37	Brockwitzer Straße	
237	13.70648	51.07871	-1	Trackpos 163	TP163
238	13.70701	51.07878	-1	Trackpos 164	TP164
239	13.70807	51.07888	36	Trachauer Straße	
240	13.7138	51.07944	35	Mickten	
241	13.71431	51.0795	-1	Trackpos 165	TP165
242	13.71446	51.07949	-1	Trackpos 166	TP166
243	13.71454	51.07947	-1	Trackpos 167	TP167
244	13.71462	51.07944	-1	Trackpos 168	TP168
245	13.71468	51.07939	-1	Weiche 27	W27
246	13.71532	51.07839	-1	Trackpos 169	TP169
247	13.71561	51.07811	-1	Trackpos 170	TP170
248	13.71597	51.07788	-1	Trackpos 171	TP171
249	13.71618	51.07776	-1	Trackpos 172	TP172
250	13.71667	51.07754	-1	Weiche 28	W28
251	13.71693	51.07748	-1	Trackpos 173	TP173
252	13.71711	51.07747	-1	Trackpos 174	TP174
253	13.71766	51.07749	34	Altpieschen	
254	13.71806	51.07752	-1	Trackpos 175	TP175
255	13.71885	51.07741	-1	Trackpos 176	TP176
256	13.71976	51.07736	-1	Trackpos 177	TP177
257	13.72119	51.07739	-1	Trackpos 178	TP178
258	13.72176	51.07737	33	Rathaus Pieschen	
259	13.72302	51.07734	-1	Trackpos 179	TP179
260	13.72336	51.07732	-1	Trackpos 180	TP180
261	13.72503	51.07698	-1	Trackpos 181	TP181
262	13.7258	51.07685	32	Bürgerstraße	
263	13.72921	51.07631	-1	Trackpos 182	TP182
264	13.72943	51.07632	-1	Trackpos 183	TP183
265	13.72967	51.07638	-1	Trackpos 184	TP184
266	13.73174	51.07726	-1	Trackpos 185	TP185
267	13.73192	51.07731	-1	Weiche 29 / Harkortstr	W29
268	13.73208	51.07732	-1	Trackpos 186	TP186
269	13.7323	51.07728	-1	Trackpos 187	TP187
270	13.73259	51.07717	-1	Trackpos 188	TP188
271	13.73268	51.07711	-1	Trackpos 189	TP189
272	13.73281	51.07698	-1	Weiche 30 /Harkort/Großenhainer	W30
273	13.73304	51.07647	31	Liststraße	
274	13.73354	51.0756	-1	Weiche 31 / Abzwei in Fritz-Reuter	W31
275	13.73373	51.07548	-1	Trackpos 190	TP190

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
276	13.73383	51.07543	-1	Trackpos 191	TP191
277	13.734	51.07538	-1	Trackpos 192	TP192
278	13.73457	51.07522	-1	Weiche 32	W32
279	13.74116	51.07337	30	Friedensstraße	
280	13.74496	51.07231	-1	Trackpos 193	TP193
281	13.74523	51.07225	-1	Trackpos 194	TP194
282	13.74685	51.072	29	S-Bf. Bischofsplatz	
283	13.75015	51.07146	28	Bischofsweg	
284	13.75021	51.07145	-1	Weiche 33 / Bischofsweg	W33
285	13.7505	51.0714	-1	Trackpos 195	TP195
286	13.75082	51.07134	-1	Trackpos 196	TP196
287	13.75094	51.07131	-1	Trackpos 197	TP197
288	13.75567	51.06987	27	Alaunplatz	
289	13.75582	51.06982	-1	Trackpos 198	TP198
290	13.75595	51.06976	-1	Trackpos 199	TP199
291	13.75599	51.06969	-1	Trackpos 200	TP200
292	13.75351	51.06576	26	Görlitzer Straße	
293	13.75185	51.06324	-1	Trackpos 202	TP202
294	13.75179	51.06305	-1	Trackpos 203	TP203
295	13.75179	51.06294	-1	Trackpos 204	TP204
296	13.75185	51.06276	-1	Weiche 34 / Einf. Von Osten	W34
297	13.75186	51.06272	-1	Weiche 35 / Einf. Von Westen	W35
298	13.75197	51.06233	25	Bautzner/Rothenburger Str.	
299	13.75266	51.05993	-1	Weiche 36	W36
300	13.75271	51.05977	-1	Weiche 37	W37
301	13.75287	51.05921	24	Rosa-Luxemburg-Platz	
302	13.75303	51.05869	-1	Trackpos 205	TP205
303	13.7531	51.05853	-1	Trackpos 206	TP206
304	13.75322	51.05838	-1	Trackpos 207	TP207
305	13.75776	51.05421	23	Sachsenallee	
306	13.75795	51.05403	-1	Weiche 38	W38
307	13.75813	51.05385	-1	Trackpos 208	TP208
308	13.75822	51.05371	-1	Trackpos 209	TP209
309	13.75832	51.05349	-1	Trackpos 210	TP210
310	13.75835	51.0533	-1	Weiche 39	W39
311	13.75832	51.05305	-1	Trackpos 211	TP211
312	13.75829	51.05293	-1	Trackpos 212	TP212
313	13.75787	51.05184	22	Dürerstraße	
314	13.75727	51.05009	21	St.-Benno-Gymnasium	
315	13.75718	51.04984	-1	Trackpos 213	TP213
316	13.7571	51.04963	-1	Trackpos 214	TP214
317	13.75676	51.049	-1	Trackpos 215	TP215
318	13.75515	51.04645	20	Straßburger Platz	
319	13.75501	51.04623	-1	Trackpos 216	TP216
320	13.75471	51.04592	-1	Weiche 40 / Einf. Von SüdOst	W40
321	13.75152	51.04276	19	Georg-Arnhold-Bad	
322	13.74788	51.03915	-1	Trackpos 217	TP217
323	13.74773	51.03903	-1	Trackpos 218	TP218
324	13.7475	51.03888	-1	Trackpos 219	TP219
325	13.74711	51.03865	-1	Weiche 41	W41

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
326	13.74702	51.03858	-1	Trackpos 220	TP220
327	13.74694	51.03849	-1	Trackpos 221	TP221
328	13.7469	51.03837	-1	Trackpos 222	TP222
329	13.74692	51.0383	-1	Trackpos 223	TP223
330	13.747	51.03819	-1	Trackpos 224	TP224
331	13.74707	51.03815	-1	Weiche 42	W42
332	13.74825	51.03762	18	Lennéplatz	
333	13.7485	51.03752	-1	Weiche 43	W43
334	13.74871	51.03746	-1	Trackpos 225	TP225
335	13.74916	51.03743	-1	Trackpos 226	TP226
336	13.75035	51.03737	-1	Trackpos 227	TP227
337	13.75084	51.03732	-1	Trackpos 228	TP228
338	13.75126	51.03724	-1	Trackpos 229	TP229
339	13.7514	51.03721	-1	Trackpos 230	TP230
340	13.75178	51.03708	-1	Trackpos 231	TP231
341	13.75231	51.03685	-1	Trackpos 232	TP232
342	13.75311	51.03654	17	Zoo	
343	13.75832	51.03447	16	Querallee	
344	13.76238	51.03284	-1	Trackpos 233	TP233
345	13.76247	51.03276	-1	Trackpos 234	TP234
346	13.76247	51.0327	-1	Trackpos 235	TP235
347	13.76245	51.03264	-1	Trackpos 236	TP236
348	13.76163	51.03168	-1	Trackpos 237	TP237
349	13.76106	51.03098	15	S-Bf. Strehlen	
350	13.75921	51.0285	-1	Trackpos 238	TP238
351	13.75917	51.02841	-1	Trackpos 239	TP239
352	13.75919	51.0283	-1	Trackpos 240	TP240
353	13.75955	51.02772	-1	Trackpos 241	TP241
354	13.75985	51.02738	14	Wasaplatz	
355	13.76201	51.02471	13	Mockritzer Straße	
356	13.76214	51.02455	-1	Trackpos 242	TP242
357	13.76227	51.02444	-1	Trackpos 243	TP243
358	13.76525	51.0225	12	Hugo-Bürkner-Straße	
359	13.76548	51.02236	-1	Trackpos 244	TP244
360	13.76566	51.0223	-1	Trackpos 245	TP245
361	13.76577	51.02228	-1	Trackpos 246	TP246
362	13.76588	51.02229	-1	Trackpos 247	TP247
363	13.76601	51.02232	-1	Weiche 44	W44
364	13.76937	51.02458	11	Cäcilienstraße	
365	13.76965	51.0248	-1	Trackpos 249	TP249
366	13.76983	51.02486	-1	Trackpos 250	TP250
367	13.77003	51.02486	-1	Trackpos 251	TP251
368	13.77012	51.02484	-1	Trackpos 252	TP252
369	13.7712	51.02427	-1	Trackpos 248	TP248
370	13.77436	51.02264	10	Eugen-Bracht-Straße	
371	13.78043	51.01948	9	Otto-Dix-Ring	
372	13.78197	51.01869	-1	Trackpos 253	TP253
373	13.78378	51.01767	8	Wieckestraße	
374	13.78612	51.01637	-1	Trackpos 254	TP254
375	13.78669	51.01599	-1	Trackpos 255	TP255

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
376	13.78679	51.0159	7	Lohrmannstraße	
377	13.78806	51.0148	-1	Trackpos 256	TP256
378	13.78858	51.01443	-1	Trackpos 257	TP257
379	13.78901	51.01419	-1	Trackpos 258	TP258
380	13.7895	51.01396	-1	Trackpos 259	TP259
381	13.78976	51.01385	-1	Trackpos 260	TP260
382	13.79082	51.0135	6	Hülßestraße	
383	13.79139	51.01332	-1	Trackpos 261	TP261
384	13.79272	51.01309	-1	Trackpos 262	TP262
385	13.79343	51.01304	-1	Trackpos 263	TP263
386	13.79425	51.01308	-1	Trackpos 264	TP264
387	13.79469	51.01307	-1	Trackpos 265	TP265
388	13.79515	51.01302	-1	Trackpos 266	TP266
389	13.79535	51.01298	5	Altreick	
390	13.79545	51.01296	-1	Trackpos 267	TP267
391	13.79626	51.01274	-1	Trackpos 268	TP268
392	13.79649	51.0127	-1	Trackpos 269	TP269
393	13.79849	51.01247	-1	Trackpos 270	TP270
394	13.79862	51.01241	-1	Trackpos 271	TP271
395	13.79875	51.01226	-1	Trackpos 272	TP272
396	13.79874	51.01208	-1	Trackpos 273	TP273
397	13.79878	51.01193	-1	Trackpos 274	TP274
398	13.79883	51.01183	-1	Trackpos 275	TP275
399	13.80018	51.01044	-1	Trackpos 276	TP276
400	13.80043	51.01026	4	Trattendorfer Straße	
401	13.80064	51.01011	-1	Trackpos 277	TP277
402	13.80083	51.00999	-1	Trackpos 278	TP278
403	13.80168	51.00954	-1	Weiche 45 / Abzw. BH0	W45
404	13.80244	51.00915	-1	Weiche 46 / aus BH0	W46
405	13.80334	51.00872	-1	Weiche 47	W47
406	13.80344	51.00865	-1	Trackpos 279	TP279
407	13.8035	51.00854	-1	Trackpos 280	TP280
408	13.80349	51.00848	-1	Trackpos 281	TP281
409	13.80344	51.00841	-1	Weiche 48	W48
410	13.80266	51.00784	3	Albert-Wolf-Platz	
411	13.79908	51.00523	2	Jacob-Winter-Platz	
412	13.79829	51.00465	-1	Trackpos 282	TP282
413	13.79805	51.00438	-1	Trackpos 283	TP283
414	13.79795	51.00416	-1	Trackpos 284	TP284
415	13.79793	51.00404	-1	Trackpos 285	TP285
416	13.79806	51.00253	-1	Trackpos 286	TP286
417	13.79804	51.00237	-1	Trackpos 287	TP287
418	13.798	51.0023	1	Georg-Palitzsch-Straße	
419	13.79796	51.00221	-1	Trackpos 288	TP288
420	13.79695	51.00078	-1	Trackpos 289	TP289
421	13.79689	51.00067	-1	Trackpos 290	TP290
422	13.79685	51.00058	-1	Trackpos 291	TP291
423	13.79685	51.00047	-1	Trackpos 292	TP292
424	13.79686	51.00039	-1	Weiche 49 / Einf. Prohlis	W49
425	13.79688	51.00031	-1	Trackpos 293	TP293

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
426	13.79694	51.00022	-1	Trackpos 294	TP294
427	13.79701	51.00015	-1	Trackpos 295	TP295
428	13.7971	51.00007	-1	Trackpos 296	TP296
429	13.79721	51	-1	Trackpos 297	TP297
430	13.79749	50.99986	0	Prohlis Gleisschleife / Ankunft Außengleis	

Tabelle 8: Haltepositionen, Weichen und Trackpos-Punkte Linie 13 (Riegelplatz in Fahrtrichtung Prohlis)

Analog zum vorherigen Vorgehen können wir nun die Netzwerkknoten einrichten, die für die Konstruktion der Infrastruktur notwendig sind, damit Fahrzeuge von der Endhaltestelle Kaditz / Riegelplatz (Stop-ID 42) zur Gleisschleife Prohlis (Stop-ID 0) fahren können. Hierbei ist eine Besonderheit in der Infrastruktur zu beachten. Zwischen den Haltestellen Görlitzer Straße (Stop-ID 26) und Alaunpark (Stop-ID 27) gibt es einen eingleisigen Abschnitt, auf dem sich Fahrzeuge nicht begegnen dürfen. Für die Streckenführung von Kaditz / Riegelplatz nach Prohlis dürfen somit die beiden Weichen W17 und W18 sowie TP84 nicht nochmal eingerichtet werden. Es ergibt sich somit die in Tab. 8 dargestellte Liste zusätzlich einzurichtender Netzwerkknoten. Die Nummerierung dieser Knoten startet bei 209, da der letzte zuvor eingerichtete Knoten die ID 208 vorweist. Die Quellcode-Ergänzung erfolgt analog zum in Abb. 5 gezeigten Quellcode. Ebenso muss erneut der Ausdruck des Konstruktors des ***BUSSIM_NETWORK***-Objekts in `main.cpp` angepasst werden (von `class BUSSIM_NETWORK NET(44,209,208,0,0,0,0,0);` zu `class BUSSIM_NETWORK NET(44,431,208,0,0,0,0,0);`).

6.4 Einrichten der Streckenabschnitte

Das Simulationssystem bewegt Fahrzeuge von Knoten zu Knoten im Graphen $\mathcal{G}$ nach der jeweils für ein Fahrzeug angegebenen Geschwindigkeit. Damit das System weiß, wo es Fahrzeuge bewegen kann, muss gekennzeichnet werden, ob und wie eine Verbindung ausgehend von einem Knoten i zu einem anderen Knoten j existiert. Hierfür legen wir die **Pfeile** $(i; j)$ des Graphen $\mathcal{G}$ fest. Falls ein solcher Pfeil festgelegt wird, weiß das Simulationsprogramm, dass es ein Fahrzeug von i direkt nach j fahren lassen kann. Aus den Koordinaten der Knoten i und j kann die Streckenlänge bzw. die Fahrdauer zwischen i und j ermittelt werden. Wurde dem Simulationsprogramm kein Pfeil $(i; j)$ bekannt gegeben, so kann kein Fahrzeug von i direkt nach j in Bewegung gesetzt werden. Wir können uns einen Pfeil $(i; j)$ als eine Fahrstreckenverbindung vorstellen, die auf direktem Wege von i nach j (aber nicht von j nach i!) führt. Liegt in der Realität keine geradlinige Verbindung von i nach j vor, so kann durch die Hinzunahme von trackpos-Knoten der Streckenverlauf beliebig genau nachgebildet werden.

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Streckenabschnitts
orig_node	int	ID des Start-Knotens
dest_node	int	ID des Ziel-Knotens

Tabelle 9: Wesentliche Attribute für das Objekt `BUSSIM_ARC`

Streckenabschnitte werden in Instanzen des Objekts ***BUSSIM_ARC*** abgelegt. Tabelle 9 zeigt die wesentlichen Attribute, die durch den Benutzer für die Abbildung der Infrastruktur gesetzt werden müssen. Die Streckenabschnitte müssen mit einer eindeutigen, fortlaufenden und lückenlos vergebenen ID versehen werden (beginnend bei 0). Der Starknoten (orig_node) sowie der Zielknoten (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,

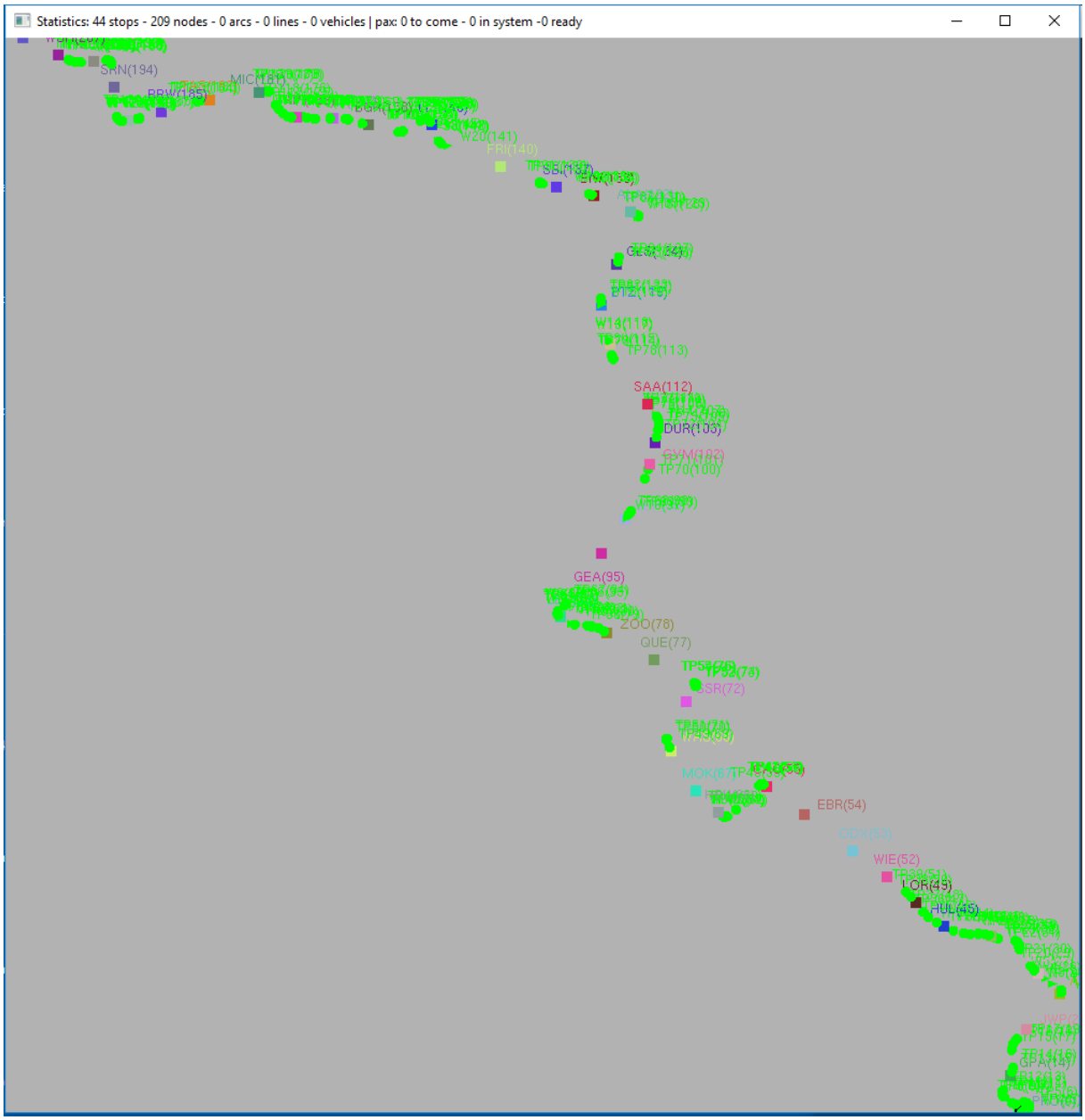

Abbildung 6: ***BUSSIM_POINT***-Objekte entlang der Streckenführung der Linie 13 von Prohlis and Kaditz

Abbildung 7: Zoom in die ***BUSSIM_POINT***-Objekte entlang der Streckenführung der Linie 13 von Prohlis and Kaditz im Bereich Listplatz

berechnet B-u-S-Sim die Länge des Pfeils (`length_screen`) und ermittelt die Start- und Zielkoordinaten für die Darstellung auf dem Ausgabebildschirm.

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

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

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

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).
Eine Analyse der abgespeicherten ***BUSSIM_POINT***-Objekte ergibt, dass wir insgesamt 208 Pfeile einfügen müssen, um eine durchgehende Verbindung vom Start-Depot-Knoten in Prohlis zum Endhalt in Kaditz einzurichten. Die Pfeile sind $(0; 1), (1; 2), (2; 3), \ldots, (207; 208)$. Sie werden von 0 bis 207 bei 0 beginnend mit einer fortlaufenden ID versehen.
Die für die Befüllung der 208 eingerichteten ***BUSSIM_ARC***-Instanzen `ARC[0]`,...,`ARC[207` notwendige Quellcode-Erweiterung ist in Abbildung 8 enthalten. Damit bei der Einrichtung des übergeordneten ***BUSSIM_NET***-Objekts die ausreichende Anzahl von ***BUSSIM_ARC***-Instanzen deklariert wird, muss erneut Befehl zum Aufruf des Konstruktors in der Datei `main.cpp` angepasst werden. Hierfür ersetzen Sie den Ausdruck `class BUSSIM_NETWORK NET(44,209,0,0,0,0,0,0);` durch den Ausdruck `class BUSSIM_NETWORK NET(44,209,208,0,0,0,0,0);`.
Die Einrichtung der Pfeile zwischen den Netzwerkknoten entlang des Streckenverlaufs von Kaditz in Fahrtrichtung Prohlis erfolgt grundsätzlich genauso wie soeben beschrieben. Wir müssen hier allerdings ein besonderes Augenmerk darauf legen, dass es zwischen den Haltepositionen Alaunplatz und Görlitzer Straße den eingleisigen Streckenabschnitt gibt, in der die Knoten aus der Gegenrichtung (von Prohlis nach Kaditz) erneut benutzt werden müssen. Bis zum Knoten mit der ID 291 fügen wir nun die Pfeile zwischen den zusätzlichen für die Fahrstrecke von Kaditz nach Prohlis eingerichteten Knoten mit den IDs 209-291 ein, d.h. die Pfeile $(209; 210), (210; 211), \ldots, (290; 291)$. Der Knoten mit der ID 291 (Kurzname: TP200) ist der letzte Trackpos-Knoten vor der Einfahrtsweiche in den eingleisigen Abschnitt. Wir fahren dann mit der Spezifikation der Pfeile nach der Ausfahrtsweiche aus dem eingleisigen Abschnitt fort. Der erste Knoten nach der Weiche hat die ID 292 (Plattform Görlitzer Straße in Fahrtrichtung Prohlis). Somit müssen wir die Pfeile $(292; 293), (293; 294), \ldots, (429; 430)$ definieren. Hierzu verwenden wir bei weiterführender ID-Nummerierung die `specify_arc`-Methode wie in Abb. 8 und legen die ***BUSSIM_ARC***-Instanzen mit den IDs 208 bis 427 an.
Um den Streckenverlauf von Kaditz nach Prohlis zu komplettieren, müssen wir die Modellierung des eingleisigen Abschnitts zwischen Alaunplatz und Görlitzer STraße genauer betrachten. Abb. 9 stellt den entsprechenden Abschnitt des Graphen dar. In rot sind die IDs der Pfeile abgebildet. Die Pfeilsequenz 125, 126, 127 und 128 beschreibt die Fahrstrecke in Richtung Kaditz (d.h. vom Knoten mit der ID125 bis zum Knoten mit der ID 129). Zunächst wird über die Weiche W17, anschließend über den Knoten TP84 und die Weiche W18 gefahren. Dies vier benötigten Pfeile wurden bereits bei der Spezifikation des Streckenverlaufs von Prohlis nach Kaditz eingerichtet. Die Pfeile erlauben nur eine Durchfahrt in Pfeilrichtung. Damit ein Fahrzeug von TP200 über W18, TP84 und W17 zum Knoten GÖR fahren kann, müssen wir also abschließend noch die vier Pfeile $(291; 128), (128; 127), (127; 126)$ und $(126; 292)$ definieren. Diese erhalten die IDs 428 bis 431. Auch diese vier Pfeile werden mit der Methode `specify_arcs` des ***BUSSIM_NETWORK***-Objekts spezifiziert.
Insgesamt haben wir zur Darstellung der beiden Streckenverläufe zwischen Prohlis und Kaditz 432 Pfeile einrichten müssen. Damit diese bei der Einrichtung des Netzwerk-Objekts zur Verfügung gestellt werden, muss in der Datei `main.cpp` der Ausdruck für den Netzwerk-Konstruktur geändert werden zu `class BUSSIM_NETWORK NET(44,431,432,0,0,0,0,0);`.
Nachdem wir die Knoten und Pfeile der Infrastruktur eingerichtet haben, können wir uns diese nun im Simulationsprogramm ansehen. Hierzu ist ein Übersetzen und der Neustart des Simulationsprogramms notwendig. Abb. 10 zeigt die Infrastruktur. Es sind alle Knoten inkl. IDs angezeigt. Standardmäßig wird

```
1  void BUSSIM_NETWORK::specify_arcs(void)
2  {
3      // type in the arc specifications here
4
5   this->ARC[0].specify(0, 0, 1);
6   this->ARC[1].specify(1, 1, 2);
7   this->ARC[2].specify(2, 2, 3);
8   this->ARC[3].specify(3, 3, 4);
9   this->ARC[4].specify(4, 4, 5);
10  this->ARC[5].specify(5, 5, 6);
11  this->ARC[6].specify(6, 6, 7);
12  this->ARC[7].specify(7, 7, 8);
13  this->ARC[8].specify(8, 8, 9);
14  this->ARC[9].specify(9, 9, 10);
15  this->ARC[10].specify(10, 10, 11);
16  this->ARC[11].specify(11, 11, 12);
17  this->ARC[12].specify(12, 12, 13);
18  this->ARC[13].specify(13, 13, 14);
19  this->ARC[14].specify(14, 14, 15);
20  this->ARC[15].specify(15, 15, 16);
21  this->ARC[16].specify(16, 16, 17);
22  this->ARC[17].specify(17, 17, 18);
23  this->ARC[18].specify(18, 18, 19);
24  this->ARC[19].specify(19, 19, 20);
25  this->ARC[20].specify(20, 20, 21);
26  this->ARC[21].specify(21, 21, 22);
27  this->ARC[22].specify(22, 22, 23);
28  this->ARC[23].specify(23, 23, 24);
29  this->ARC[24].specify(24, 24, 25);
30  this->ARC[25].specify(25, 25, 26);
31  this->ARC[26].specify(26, 26, 27);
32  this->ARC[27].specify(27, 27, 28);
33  this->ARC[28].specify(28, 28, 29);
34  this->ARC[29].specify(29, 29, 30);
35  this->ARC[30].specify(30, 30, 31);
36  this->ARC[31].specify(31, 31, 32);
37  this->ARC[32].specify(32, 32, 33);
38  this->ARC[33].specify(33, 33, 34);
39  this->ARC[34].specify(34, 34, 35);
40  this->ARC[35].specify(35, 35, 36);
41  this->ARC[36].specify(36, 36, 37);
42  this->ARC[37].specify(37, 37, 38);
43  this->ARC[38].specify(38, 38, 39);
44  this->ARC[39].specify(39, 39, 40);
45  this->ARC[40].specify(40, 40, 41);
46  this->ARC[41].specify(41, 41, 42);
47  this->ARC[42].specify(42, 42, 43);
48  this->ARC[43].specify(43, 43, 44);
49  this->ARC[44].specify(44, 44, 45);
50  this->ARC[45].specify(45, 45, 46);
51  this->ARC[46].specify(46, 46, 47);
52  this->ARC[47].specify(47, 47, 48);
53  this->ARC[48].specify(48, 48, 49);
54  this->ARC[49].specify(49, 49, 50);
55  this->ARC[50].specify(50, 50, 51);
```

Abbildung 8: Quellcode zur Spezifikation der Streckenabschnitte

```
56  this->ARC[51].specify(51, 51, 52);
57  this->ARC[52].specify(52, 52, 53);
58  this->ARC[53].specify(53, 53, 54);
59  this->ARC[54].specify(54, 54, 55);
60  this->ARC[55].specify(55, 55, 56);
61  this->ARC[56].specify(56, 56, 57);
62  this->ARC[57].specify(57, 57, 58);
63  this->ARC[58].specify(58, 58, 59);
64  this->ARC[59].specify(59, 59, 60);
65  this->ARC[60].specify(60, 60, 61);
66  this->ARC[61].specify(61, 61, 62);
67  this->ARC[62].specify(62, 62, 63);
68  this->ARC[63].specify(63, 63, 64);
69  this->ARC[64].specify(64, 64, 65);
70  this->ARC[65].specify(65, 65, 66);
71  this->ARC[66].specify(66, 66, 67);
72  this->ARC[67].specify(67, 67, 68);
73  this->ARC[68].specify(68, 68, 69);
74  this->ARC[69].specify(69, 69, 70);
75  this->ARC[70].specify(70, 70, 71);
76  this->ARC[71].specify(71, 71, 72);
77  this->ARC[72].specify(72, 72, 73);
78  this->ARC[73].specify(73, 73, 74);
79  this->ARC[74].specify(74, 74, 75);
80  this->ARC[75].specify(75, 75, 76);
81  this->ARC[76].specify(76, 76, 77);
82  this->ARC[77].specify(77, 77, 78);
83  this->ARC[78].specify(78, 78, 79);
84  this->ARC[79].specify(79, 79, 80);
85  this->ARC[80].specify(80, 80, 81);
86  this->ARC[81].specify(81, 81, 82);
87  this->ARC[82].specify(82, 82, 83);
88  this->ARC[83].specify(83, 83, 84);
89  this->ARC[84].specify(84, 84, 85);
90  this->ARC[85].specify(85, 85, 86);
91  this->ARC[86].specify(86, 86, 87);
92  this->ARC[87].specify(87, 87, 88);
93  this->ARC[88].specify(88, 88, 89);
94  this->ARC[89].specify(89, 89, 90);
95  this->ARC[90].specify(90, 90, 91);
96  this->ARC[91].specify(91, 91, 92);
97  this->ARC[92].specify(92, 92, 93);
98  this->ARC[93].specify(93, 93, 94);
99  this->ARC[94].specify(94, 94, 95);
100 this->ARC[95].specify(95, 95, 96);
101 this->ARC[96].specify(96, 96, 97);
102 this->ARC[97].specify(97, 97, 98);
103 this->ARC[98].specify(98, 98, 99);
104 this->ARC[99].specify(99, 99, 100);
105 this->ARC[100].specify(100, 100, 101);
```

Abbildung 8: Quellcode zur Spezifikation der Streckenabschnitte

```
106   this->ARC[101].specify(101, 101, 102);
107   this->ARC[102].specify(102, 102, 103);
108   this->ARC[103].specify(103, 103, 104);
109   this->ARC[104].specify(104, 104, 105);
110   this->ARC[105].specify(105, 105, 106);
111   this->ARC[106].specify(106, 106, 107);
112   this->ARC[107].specify(107, 107, 108);
113   this->ARC[108].specify(108, 108, 109);
114   this->ARC[109].specify(109, 109, 110);
115   this->ARC[110].specify(110, 110, 111);
116   this->ARC[111].specify(111, 111, 112);
117   this->ARC[112].specify(112, 112, 113);
118   this->ARC[113].specify(113, 113, 114);
119   this->ARC[114].specify(114, 114, 115);
120   this->ARC[115].specify(115, 115, 116);
121   this->ARC[116].specify(116, 116, 117);
122   this->ARC[117].specify(117, 117, 118);
123   this->ARC[118].specify(118, 118, 119);
124   this->ARC[119].specify(119, 119, 120);
125   this->ARC[120].specify(120, 120, 121);
126   this->ARC[121].specify(121, 121, 122);
127   this->ARC[122].specify(122, 122, 123);
128   this->ARC[123].specify(123, 123, 124);
129   this->ARC[124].specify(124, 124, 125);
130   this->ARC[125].specify(125, 125, 126);
131   this->ARC[126].specify(126, 126, 127);
132   this->ARC[127].specify(127, 127, 128);
133   this->ARC[128].specify(128, 128, 129);
134   this->ARC[129].specify(129, 129, 130);
135   this->ARC[130].specify(130, 130, 131);
136   this->ARC[131].specify(131, 131, 132);
137   this->ARC[132].specify(132, 132, 133);
138   this->ARC[133].specify(133, 133, 134);
139   this->ARC[134].specify(134, 134, 135);
140   this->ARC[135].specify(135, 135, 136);
141   this->ARC[136].specify(136, 136, 137);
142   this->ARC[137].specify(137, 137, 138);
143   this->ARC[138].specify(138, 138, 139);
144   this->ARC[139].specify(139, 139, 140);
145   this->ARC[140].specify(140, 140, 141);
146   this->ARC[141].specify(141, 141, 142);
147   this->ARC[142].specify(142, 142, 143);
148   this->ARC[143].specify(143, 143, 144);
149   this->ARC[144].specify(144, 144, 145);
150   this->ARC[145].specify(145, 145, 146);
151   this->ARC[146].specify(146, 146, 147);
152   this->ARC[147].specify(147, 147, 148);
153   this->ARC[148].specify(148, 148, 149);
154   this->ARC[149].specify(149, 149, 150);
155   this->ARC[150].specify(150, 150, 151);
```

Abbildung 8: Quellcode zur Spezifikation der Streckenabschnitte

```
156   this->ARC[151].specify(151, 151, 152);
157   this->ARC[152].specify(152, 152, 153);
158   this->ARC[153].specify(153, 153, 154);
159   this->ARC[154].specify(154, 154, 155);
160   this->ARC[155].specify(155, 155, 156);
161   this->ARC[156].specify(156, 156, 157);
162   this->ARC[157].specify(157, 157, 158);
163   this->ARC[158].specify(158, 158, 159);
164   this->ARC[159].specify(159, 159, 160);
165   this->ARC[160].specify(160, 160, 161);
166   this->ARC[161].specify(161, 161, 162);
167   this->ARC[162].specify(162, 162, 163);
168   this->ARC[163].specify(163, 163, 164);
169   this->ARC[164].specify(164, 164, 165);
170   this->ARC[165].specify(165, 165, 166);
171   this->ARC[166].specify(166, 166, 167);
172   this->ARC[167].specify(167, 167, 168);
173   this->ARC[168].specify(168, 168, 169);
174   this->ARC[169].specify(169, 169, 170);
175   this->ARC[170].specify(170, 170, 171);
176   this->ARC[171].specify(171, 171, 172);
177   this->ARC[172].specify(172, 172, 173);
178   this->ARC[173].specify(173, 173, 174);
179   this->ARC[174].specify(174, 174, 175);
180   this->ARC[175].specify(175, 175, 176);
181   this->ARC[176].specify(176, 176, 177);
182   this->ARC[177].specify(177, 177, 178);
183   this->ARC[178].specify(178, 178, 179);
184   this->ARC[179].specify(179, 179, 180);
185   this->ARC[180].specify(180, 180, 181);
186   this->ARC[181].specify(181, 181, 182);
187   this->ARC[182].specify(182, 182, 183);
188   this->ARC[183].specify(183, 183, 184);
189   this->ARC[184].specify(184, 184, 185);
190   this->ARC[185].specify(185, 185, 186);
191   this->ARC[186].specify(186, 186, 187);
192   this->ARC[187].specify(187, 187, 188);
193   this->ARC[188].specify(188, 188, 189);
194   this->ARC[189].specify(189, 189, 190);
195   this->ARC[190].specify(190, 190, 191);
196   this->ARC[191].specify(191, 191, 192);
197   this->ARC[192].specify(192, 192, 193);
198   this->ARC[193].specify(193, 193, 194);
199   this->ARC[194].specify(194, 194, 195);
200   this->ARC[195].specify(195, 195, 196);
201   this->ARC[196].specify(196, 196, 197);
202   this->ARC[197].specify(197, 197, 198);
203   this->ARC[198].specify(198, 198, 199);
204   this->ARC[199].specify(199, 199, 200);
205   this->ARC[200].specify(200, 200, 201);
```

Abbildung 8: Quellcode zur Spezifikation der Streckenabschnitte

```
206   this->ARC[201].specify(201, 201, 202);
207   this->ARC[202].specify(202, 202, 203);
208   this->ARC[203].specify(203, 203, 204);
209   this->ARC[204].specify(204, 204, 205);
210   this->ARC[205].specify(205, 205, 206);
211   this->ARC[206].specify(206, 206, 207);
212   this->ARC[207].specify(207, 207, 208);
213
214       // arc specification ends here. we continue and calculate the length of
              the arcs
215       for(int a=0 ; a < this->ARCS ; a++)
216           this->ARC[a].update_length(this);
217   }
```

Abbildung 8: Quellcode zur Spezifikation der Streckenabschnitte von Prohlis nach Kaditz

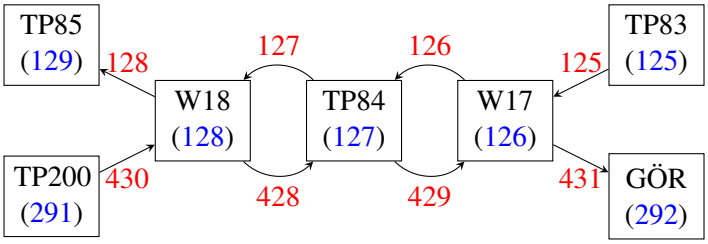

Abbildung 9: Graphen-Modellierung des eingleisigen Streckenabschnitts zwischen Alaunplatz und Görlitzer Straße

die Infrastruktur nicht angezeigt, daher muss dies über das Menü, das sich hinter der rechten Maustaste verbirgt, aktiv eingefordert werden.
Abb. 11 stellt einen Zoom auf den eingleisigen Bereich zwischen den Haltepositionen Görlitzer Straße und Alaunplatz zur Verfügung. Am oberen sowie am unteren Bildrand sind die Weichen zu erkennen, über die die Fahrzeuge in den eingleisigen Abschnitt hinein bzw. aus dem eingleisigen Abschnitt herausgeführt werden.

6.5 Verknüpfen von Streckenverläufen

Im Wesentlichen haben wir nun die Gleisinfrastruktur dem Simulationsprogramm mitgeteilt, die zwischen den Endhaltestellen in Prohlis sowie Riegelplatz vorhanden sind. Zommen wir nun in den Bereich, in dem die Gleisschleife Prohlis liegt, so erkennen wir, dass die Schleife noch nicht geschlossen ist (Abb. 12). Die Strecke aus Kaditz kommend endet im Knoten 430 und die Strecken nach Kaditz beginnt im Knoten 0. Damit Fahrzeuge die Gleisschleife durchfahren können, müssen wir noch diese beiden Punkte durch Pfeile verbinden. Zusätzlich müssen wir auch noch das innere Gleis zwischen der Weiche am Knoten 424 und der Weiche am Knoten 4 einrichten.

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
431	13.7981	50.99955	-1	Trackpoint 298	TP298
432	13.79704	51.00018	-1	Trackpoint 299	TP299
433	13.79749	50.99992	0	Ankunft Innengleis	
434	13.79813	50.99962	-1	Trackpoint 300	TP300
435	13.79888	50.99924	0	Abfahrt Innengleis	
436	13.7991	50.99913	-1	Trackpoint 301	TP301
437	13.79934	50.99908	-1	Trackpoint 302	TP302
438	13.79958	50.99909	-1	Trackpoint 303	TP303

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
439	13.79974	50.99915	-1	Trackpoint 304	TP304

Tabelle 10: zusätzliche Knoten zur Modellierung der Gleisschleife Prohlis

Wir kümmern uns zunächst um die Schließung der Lücke zwischen den Knoten 430 und 0. Beide Knoten stellen Haltepunkte in der Haltestelle mit der ID=0 dar. Am Knoten 430 kommt ein Fahrzeug mit dem Ziel Prohlis an und den Knoten 0 verlässt das Fahrzeug mit der Zielbeschilderung Kaditz. Somit muss zwischen diesen beiden Haltepositionen ein Wechsel der Zielbeschilderung ausgelöst werden. Hierfür muss ein zusätzlicher trackpos-Knoten eingefügt werden (Tab. 10). Dieser erhält die ID=431 zugewiesen. Die zusätzlichen Knoten 432-439 nutzen wir zur Modellierung des Innengleises, wobei Knoten 433 der Ankunftsknoten und Knoten 435 den Abfahrts-Depot-Knoten repräsentiert. Der trackpos-Knoten mit der ID 434 (TP300) wird dann genutzt, um den Wechsel der Zielbeschilderung zu ermöglichen.
Mit den in Abb. 13 gezeigten zusätzlichen ***BUSSIM_ARC***-Objekten werden nun die Lücken in der Gleisinfrastruktur der Gleisschleife Prohlis eingerichtet. Nachdem die Anzahl der benötigten ***BUSSIM_POINT***- und ***BUSSIM_ARC***-Instanzen im Konstruktur des ***BUSSIM_NETOWRK***-Objekts in der `main.cpp`-angepasst wurden und der ergänzte Quellcode erneut übersetzt wurde wird nach dem Programmstart die komplette Gleisschleife angezeigt (Abb. 12).
Analog zum soeben gezeigten Vorgehen können wir nun die Gleisschleife in Kaditz vervollständigen. Dort haben wir bisher nur die Ankunftsplattform (ID=208) und das Abfahrts-Depot-Objekt (ID=209) festgelegt (Abb. 15).

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
440	13.6867	51.08387	-1	Trackpoint 305	TP305
441	13.68646	51.08388	-1	Trackpoint 306	TP306
442	13.68634	51.08384	-1	Trackpoint 307	TP307
443	13.68625	51.08378	-1	Trackpoint 308	TP308
444	13.68619	51.08369	-1	W50 / Einfahrt Kaditz	W50
445	13.68614	51.0836	-1	Trackpoint 309	TP309
446	13.68613	51.08353	-1	Trackpoint 310	TP310
447	13.68615	51.08346	-1	Trackpoint 311	TP311
448	13.68625	51.08339	-1	Trackpoint 312	TP312
449	13.68634	51.08337	-1	Trackpoint 313	TP313
450	13.68647	51.08335	-1	Trackpoint 314	TP314
451	13.68618	51.0836	-1	Trackpoint 315	TP315
452	13.68619	51.08353	-1	Trackpoint 316	TP316
453	13.68625	51.08347	-1	Trackpoint 317	TP317
454	13.68638	51.08343	-1	Trackpoint 318	TP318
455	13.68654	51.08343	-1	Trackpoint 319	TP319
456	13.68723	51.08348	42	Abfahrt Kaditz / Innengleis	
457	13.68736	51.08349	-1	Trackpoint 320	TP320
458	13.68758	51.08348	-1	Trackpoint 321	TP321

Tabelle 11: Zusätzliche Knoten zur Modellierung der Gleisschleife Kaditz

Tab. 11 stellt die zusätzlich für die Modellierung der Gleisschleife Kaditz benötigten Information für die zugehörigen ***BUSSIM_POINT***-Instanzen zusammen. Die Knoten mit den IDs 440 - 450 werden für das Außengleis benötigt. Die Knoten mit den IDs 451 - 458 definieren das Innnengleis.
Nach der Einrichtung der zusätzlichen Knoten können die in Abb. 16 gelisteten zusätzlichen ***BUSSIM_ARC***-Instanzen spezifiziert werden. Die Pfeile mit den IDs 443-454 repräsentieren das Außengleis. Das Innengleis ist durch die Pfeile mit den IDs 455-463 abgebildet.

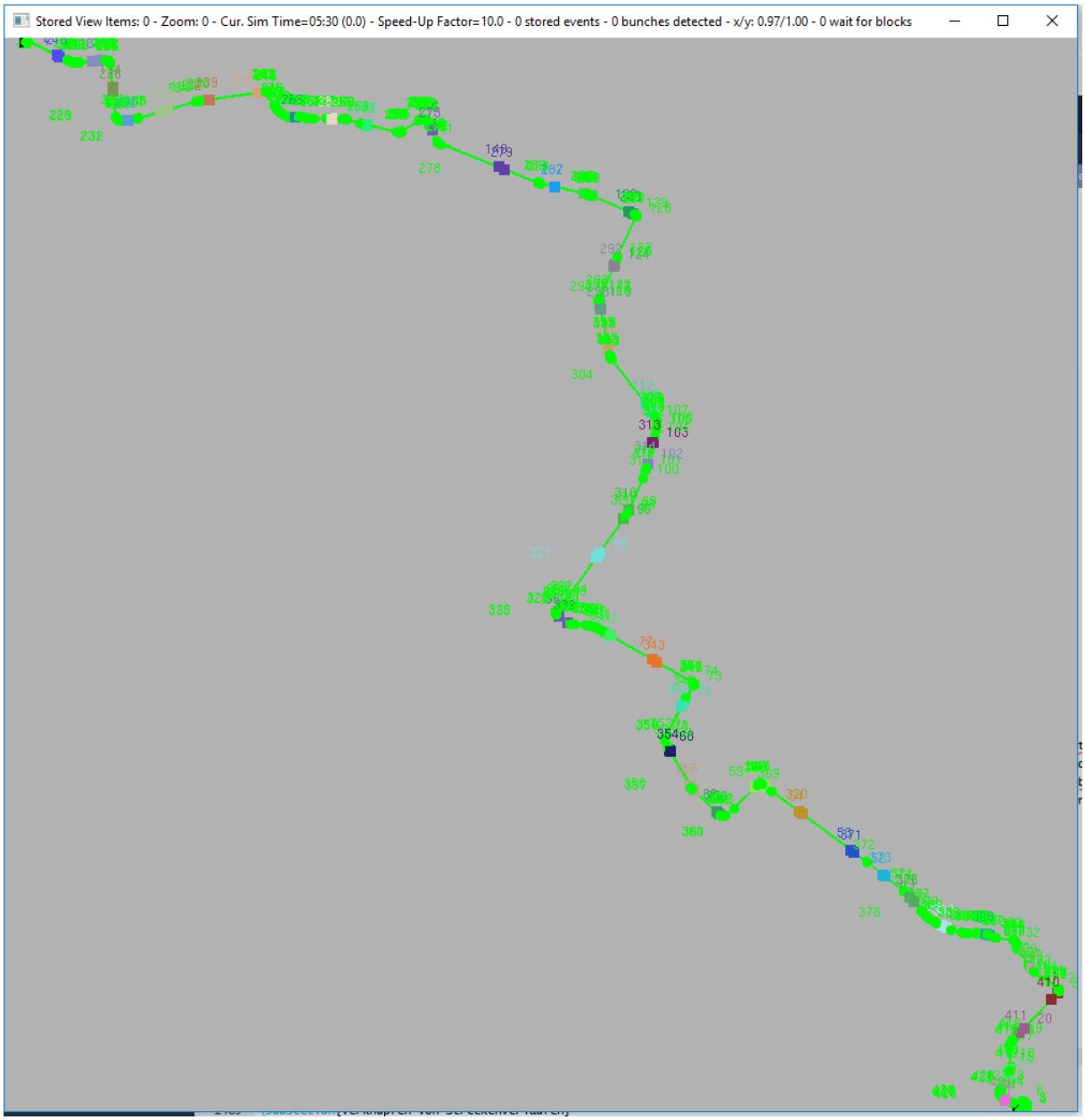

Abbildung 10: Gesamter Streckenverlauf der Linie 13

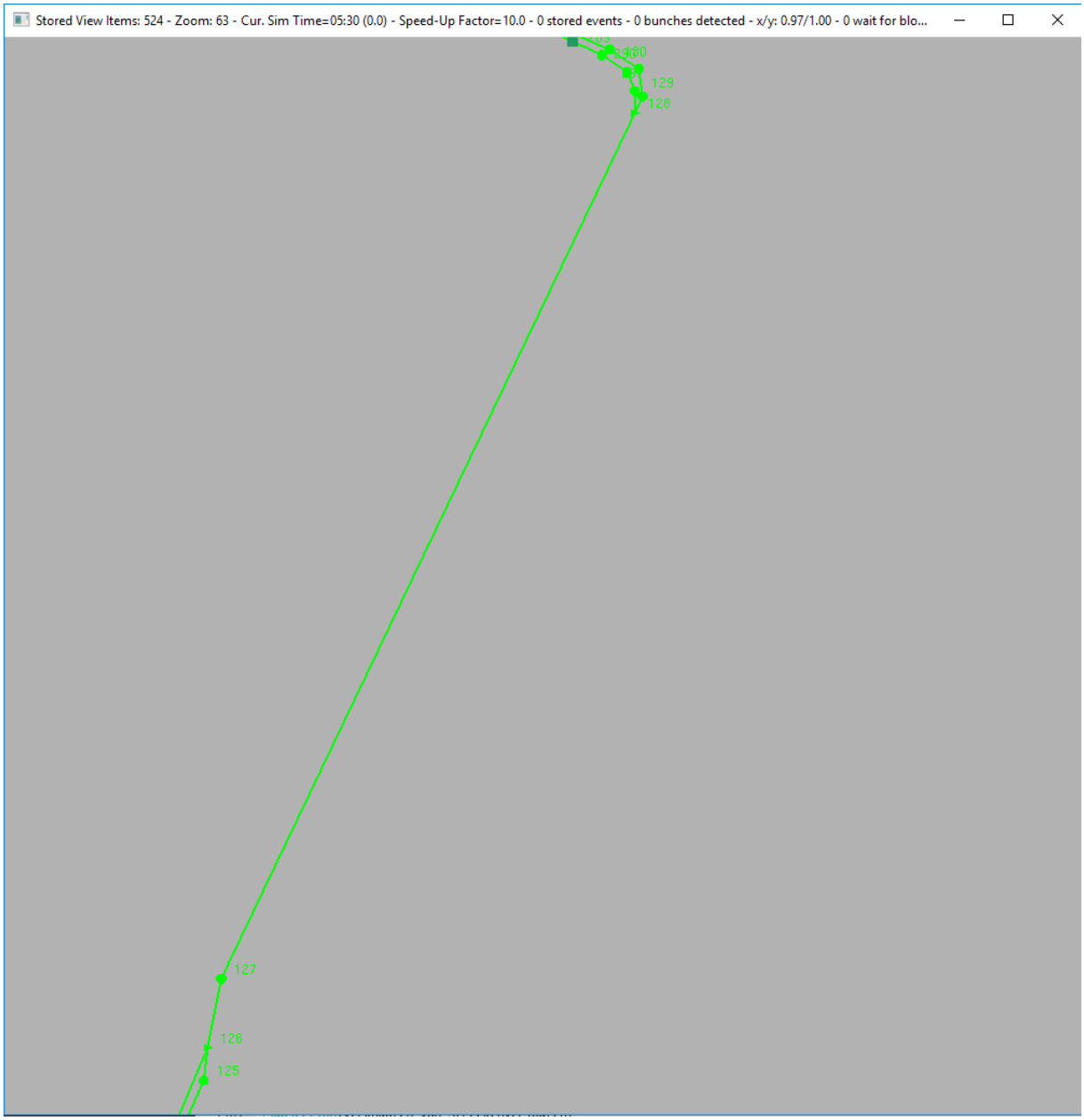

Abbildung 11: Eingleisiger Streckenabschnitt entlang der Linie 13 zwischen Görlitzer Straße und Alaunplatz

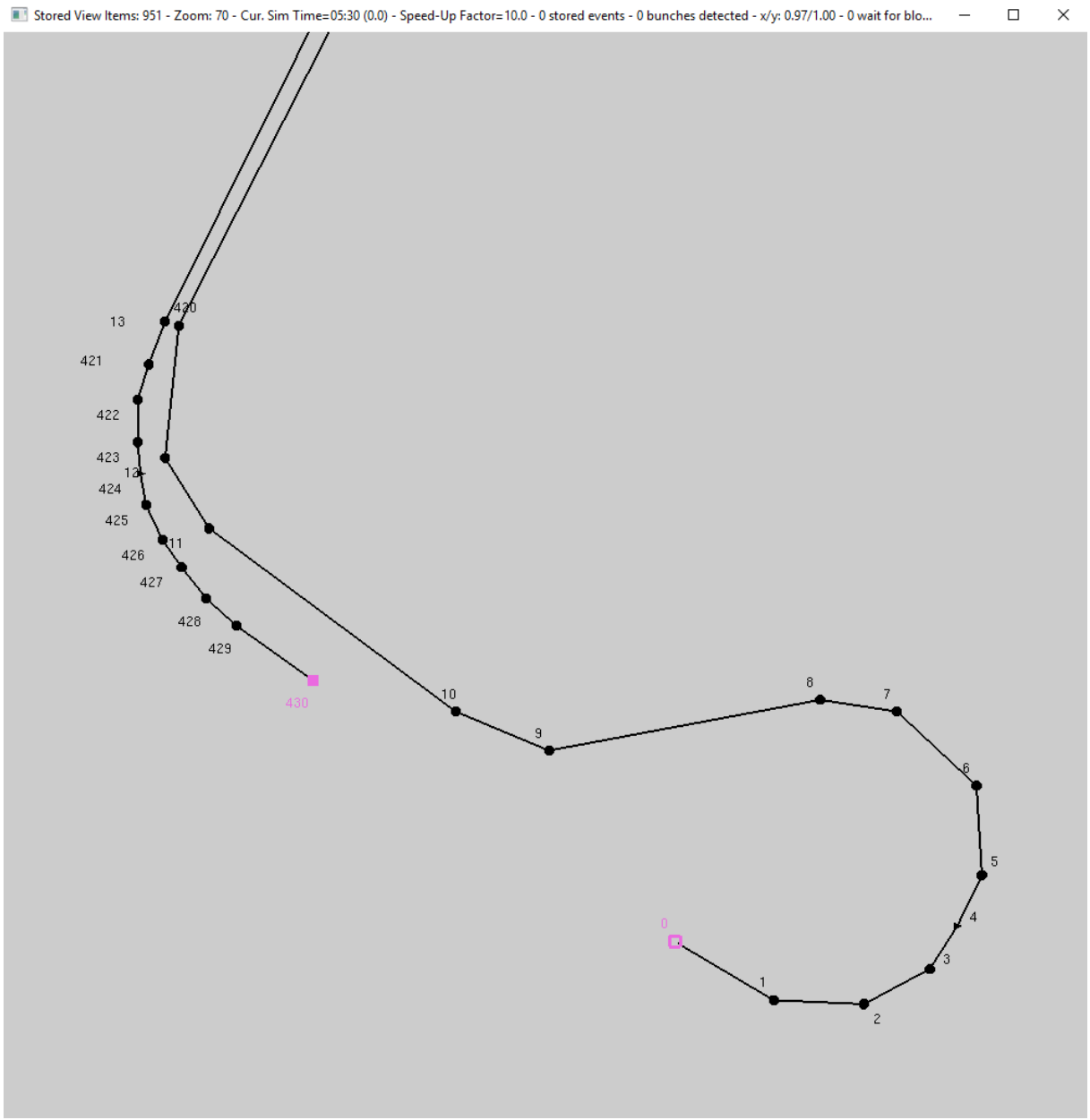

Abbildung 12: Unvollständige Gleisschleife Prohlis

```
206   this->ARC[432].specify(432, 430, 431);
207   this->ARC[433].specify(433, 431, 0);
208   this->ARC[434].specify(434, 424, 432);
209   this->ARC[435].specify(435, 432, 433);
210   this->ARC[436].specify(436, 433, 434);
211   this->ARC[437].specify(437, 434, 435);
212   this->ARC[438].specify(438, 435, 436);
213   this->ARC[439].specify(439, 436, 437);
214   this->ARC[440].specify(440, 437, 438);
215   this->ARC[441].specify(441, 438, 439);
216   this->ARC[442].specify(442, 439, 4);
```

Abbildung 13: Quellcode-Ausschnitt zur Spezifikation der Streckenabschnitte für die Komplettierung der Gleisschleife Prohlis

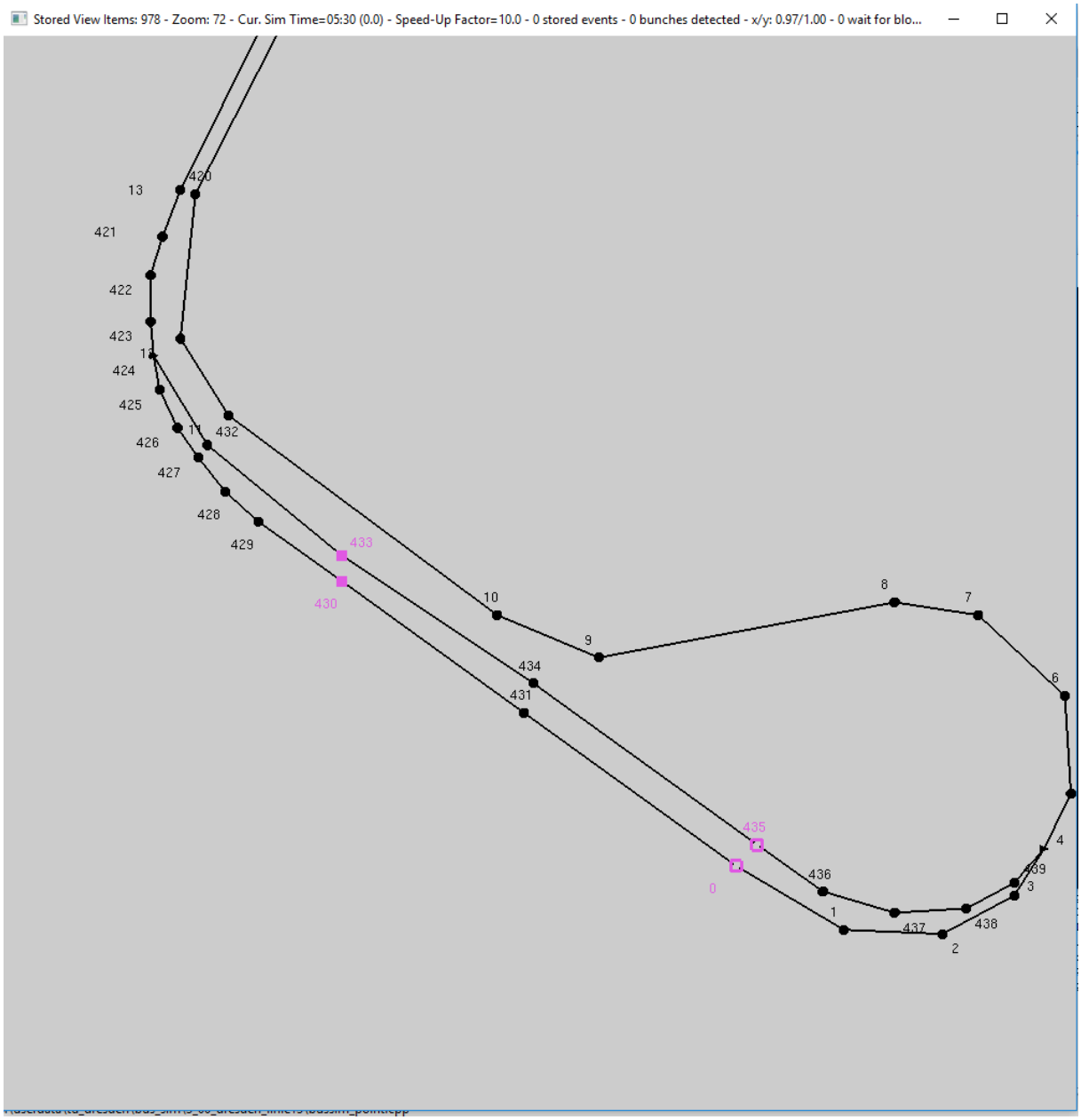

Abbildung 14: Vollständige Gleisschleife Prohlis

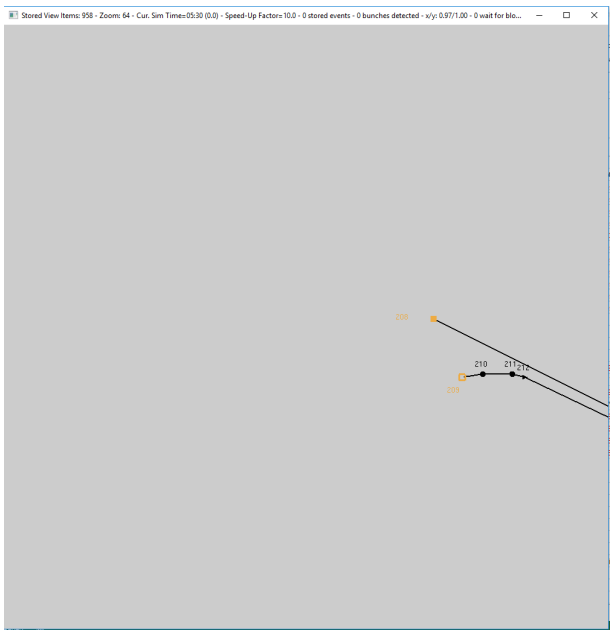

Abbildung 15: Unvollständige Gleisschleife Kaditz / Riegelplatz

```
1   this->ARC[443].specify(443, 208, 440);
2   this->ARC[444].specify(444, 440, 441);
3   this->ARC[445].specify(445, 441, 442);
4   this->ARC[446].specify(446, 442, 443);
5   this->ARC[447].specify(447, 443, 444);
6   this->ARC[448].specify(448, 444, 445);
7   this->ARC[449].specify(449, 445, 446);
8   this->ARC[450].specify(450, 446, 447);
9   this->ARC[451].specify(451, 447, 448);
10  this->ARC[452].specify(452, 448, 449);
11  this->ARC[453].specify(453, 449, 450);
12  this->ARC[454].specify(454, 450, 209);
13
14  this->ARC[455].specify(455, 444, 451);
15  this->ARC[456].specify(456, 451, 452);
16  this->ARC[457].specify(457, 452, 453);
17  this->ARC[458].specify(458, 453, 454);
18  this->ARC[459].specify(459, 454, 455);
19  this->ARC[460].specify(460, 455, 456);
20  this->ARC[461].specify(461, 456, 457);
21  this->ARC[462].specify(462, 457, 458);
22  this->ARC[463].specify(463, 458, 212);
```

Abbildung 16: Quellcode zur Spezifikation der Streckenabschnitte zur Komplettierung der Gleisschleife Kaditz

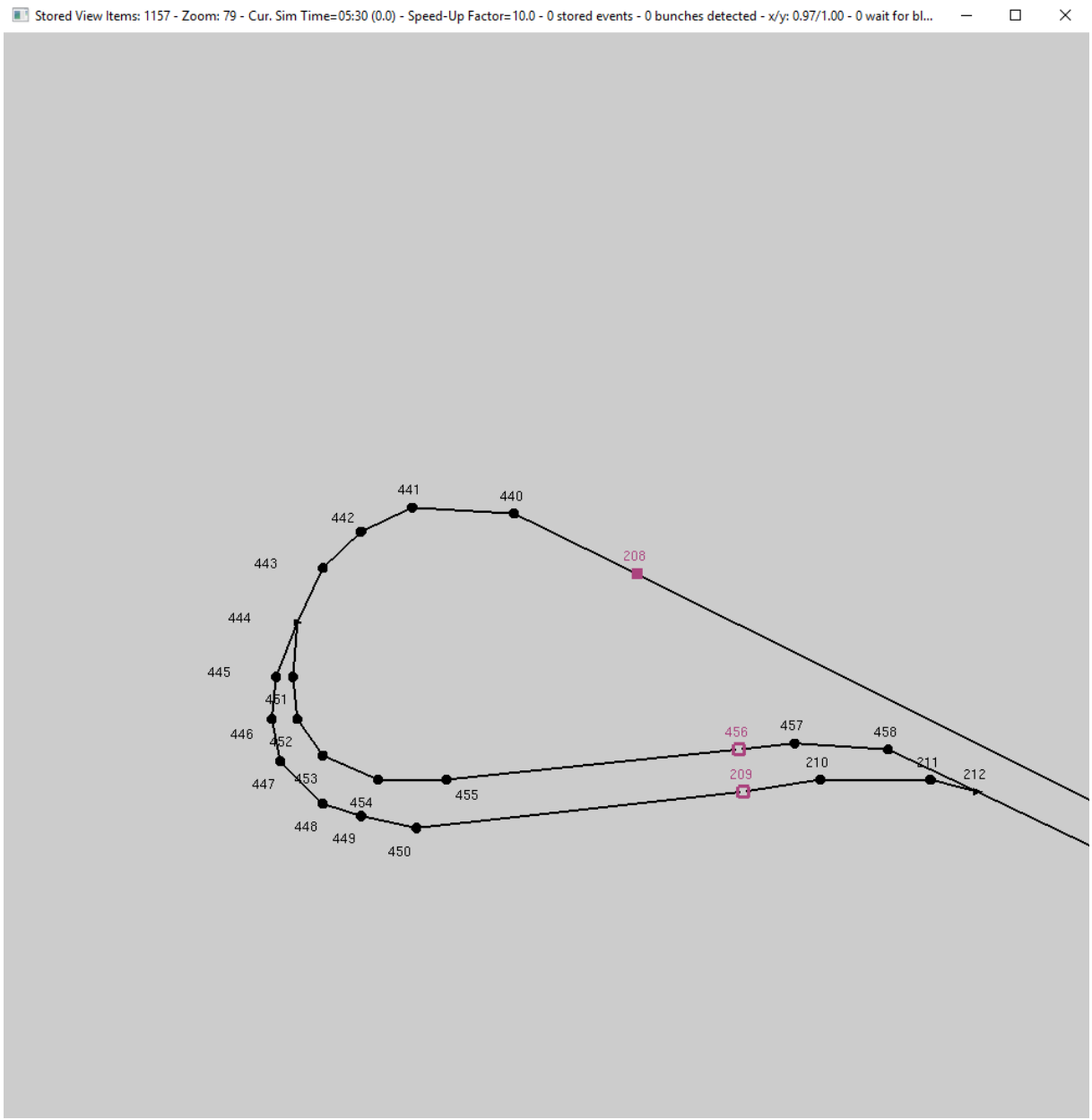

Abbildung 17: Vollständige Gleisschleife Kaditz / Riegelplatz

Nach der erneuten Abpassung des Netzwerk-Konstruktors zu `class BUSSIM_NETWORK NET(44,459,464,0,0,` kann das Projekt erneut übersetzt werden. Die Gleisschleife Kaditz/Riegelplatz ist nun ebenfalls komplett in der Simulation abgebildet (Abb. 17).

6.6 Einrichten von Infrastruktur-Blöcken

Neben den ***BUSSIM_POINT***- und ***BUSSIM_ARC***-Objekten stellen die sog. Infrastruktur-Blöcke eine wichtige Komponente der Infrastruktur dar. Ein Block ist eine Gruppe von Pfeilen. Für einen Block wird eine maximale Zahl von Fahrzeugen festgelegt, die sich gleichzeitig auf den Pfeilen eines Blocks befinden können.
Mit hilfe eines Blocks können dz.B. Abschnitte mit Begegnungsverkehren abgebildet werden, in dem zwei Pfeile (i, j) sowie $(j; i)$ in einem Block gruppiert werden: Fährt auf einem Pfeil ein Fahrzeug von i nach j kann nicht gleichzeitig ein Fahrzeug von j nach i fahren, sofern die maximal zulässige Zahl von Fahrzeugen auf Pfeilen des Blocks auf 1 gesetzt wird. Möchte ein Fahrzeug in einen Block fahren, der schon durch die maximale Anzahl von Fahrzeugen belegt ist, dann muss dieses Fahrzeug warten, bis der Block befahrbar wird. Das betreffende Fahrzeuge wird als Quadrat im Netzwerk dargestellt und wartet. Sobald der Block frei ist, fährt das Fahrzeug in den Block ein.
Im Simulationsfenster werden die vorhandenen Blöcke angezeigt, wenn die Infrastruktur eingeblendet ist (<i>-Taste). Pfeile werden in der Infrastruktur-Ansicht grundsätzlich in der in der Datei `bussim_global.h` bestimmten Farbe angezeigt. Für einen Block kann eine andere Farbe festgelegt werden und die Pfeile eines Blocks werden dann in der Blockfarbe angezeigt statt in Standard-Farbe. Ist die maximale Anzahl der zulässigen Fahrzeuge in einem Block erreicht, so wird der Block durch dickere Pfeile visuell hervorgehoben angezeigt.

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Infrastruktur-Blocks
CAPACITY	int	maximale Anzahl von Fahrzeugen, die sich gleichzeitig in einem Block befinden dürfen
DESC	string	Name / Bezeichnung des Blocks
RED	double	Anteil Rot der Darstellungsfarbe des Blocks
GREEN	double	Anteil Grün der Darstellungsfarbe des Blocks
BLUE	double	Anteil BLAU der Darstellungsfarbe des Blocks

Tabelle 12: Attribute für das Objekt `BUSSIM_BLOCK`

In B-u-S-Sim wird jeder Infrastrukturblock durch eine Instanz des Objekts ***BUSSIM_BLOCK*** repräsentiert. Tabelle 12 listet die für einen Block zu spezifizierenden Attribute auf.
Im Rahmen der im Beispiel betrachteten Linie 13 gibt es zwischen den Haltestellen Görlitzerstraße und Alaunplatz einen eingleisigen Abschnitt. Daher müssen (vereinfachend) die in Abb. 8 gezeigten Pfeile mit den IDs 125-128 sowie 428-431 in einem Block gruppiert werden, damit Frontalkollisionen von sich dort begegnenden Straßenbahn-Fahrzeugen vermieden werden. Der hierfür benötigte C++-Quellcode ist in Abb. 18 dargestellt.
Alle Infrastrukturblöcke werden in der Methode `BUSSIM_NETWORK::specify_blocks()` eingerichtet. Die Spezifikation der Attributswerte für einen Block erfolgt durch den Aufruf der Methode `BUSSIM_BLOCK::specify` unter Angabe der Werte für die Attribute `ID`, `CAPACITY`, `USED_CAP`, `DESC`, `RED`, `GREEN` und `BLUE` als Parameter. Anschließend werden mit Verwendung der Methode `add_arc` des ***BUSSIM_BLOCK***-Objekts die zum Block gehörenden Pfeile sukzessive hinzugefügt.

```
void BUSSIM_NETWORK::specify_blocks(void)
{
    this->BLOCK[0].specify(0,1,"Block Görlitzer Straße",1.0,0.0,0.0);
    this->BLOCK[0].add_arc(&(this->ARC[125]));
    this->BLOCK[0].add_arc(&(this->ARC[126]));
    this->BLOCK[0].add_arc(&(this->ARC[127]));
    this->BLOCK[0].add_arc(&(this->ARC[128]));
    this->BLOCK[0].add_arc(&(this->ARC[428]));
    this->BLOCK[0].add_arc(&(this->ARC[429]));
    this->BLOCK[0].add_arc(&(this->ARC[430]));
    this->BLOCK[0].add_arc(&(this->ARC[431]));
}
```

Abbildung 18: C++ - Code zur Einrichtung eines Infrastrukturblocks

6.7 Überprüfen des Graphen und Berücksichtigung von Ergänzungen

Wie wir gesehen haben, verursacht die realitätsnahe Repräsentation der Infrastruktur als mathematischer Graph einen hohen Arbeitsaufwand. Da oftmals auch viele manuelle Datenerfassungen und -konvertierungen vorgenommen werden müssen, besteht immer die Gefahr, dass sich Fehler z.B. bei der Festlegung der Koordinaten von Knoten ergeben. Da die weiteren Konfigurationsschritte auf die bisher definierten ***BUSSIM_NODES***- und ***BUSSIM_ARC***-Objekte zugreifen, sollten diese zum jetzigen Zeitpunkt auf Plausibilität und Korrektheit überprüft werden.
Notwendig ist eine Sichtinspektion. Hierzu wird die Simulationssoftware gestartet und die Infrastruktur sowie die Knoten inkl. den Labeln eingeblendet. Nun setzen wir den Fokus auf eine Endhaltestelle. In unserem Beispiel ist dies die Gleisschleife Prohlis. Ausgehend von dort fahren wir nun den Verlauf der Infrastruktur mit der Lupe (Fokus) entlang.
Ein erstes Problem taucht am Knoten mit der ID372 (Trackpos 253) auf. In Höhe dieses Knotens macht die Straße, deren Verlauf die Schienen folgen, eine leichte Richtungsänderung. Es ist vergessen worden, einen entsprechenden trackpos-Knoten in Fahrtrichtung Kaitz/Riegelplatz etwas nord-östlich des Knotens 372 einzufügen, so dass es zu einer Überlappung der beiden Richtungsgleise kommt (Abb.20).

ID	Längengrad	Breitengrad	Stop-ID	Bezeichnung	Kurzname
459	13.78199	51.01871	-1	Trackpoint 322	TP322
460	13.76709	51.02287	-1	Trackpoint 323	TP323

Tabelle 13: zusätzliche Knoten, die im Rahmen des finalen Checks notwendig geworden sind

Um das zuvor beschriebene Problem zu beheben, müssen wir zunächst einen zusätzlichen Knoten an passender Position spezifizieren. Dies ist der trackpos-Knoten mit der ID=459 (Tab. 13).
Um diesen zusätzlichen Knoten zwischen den Knoten 52 und 53 in den Graphen einzubinden, müssen wir den Pfeil mit der ID=52 modifizieren. Der Zielknoten ist nun nicht mehr der Knoten 53 sondern der neue Knoten mit der ID 459, d.h. der Pfeile 52 verbindet Knoten 52 mit Knoten 459 (Abb. 20). Um die Verbindung vom neuen Knoten 459 mit dem Knoten 53 zu verbinden, benötigen wir einen zusätzlichen Pfeil. Dieser besitzt im vorliegenden Fall die ID 464, den Startknoten 459 und den Zielknoten 53 (Abb. 20).
Wir haben im Rahmen dieses Korrekturschrittes eine zusätzliche Instanz des ***BUSSIM_POINT***- sowie eine zusätzliche Instanz des ***BUSSIM_ARC***-Objekts spezifiziert. Daher müssen wir deren Anzahl im Konstruktor in der Datei `main.cpp` jeweils um 1 erhöhen (`BUSSIM_NETWORK NET(44,460,465,1,0,0,0,0);`
Abb. 21 zeigt den Graphen nach durchgeführter Korrektur.
Ein identisches Problem entdecken wir am Knoten mit der ID 60 (Trackpos 44). Hier wurde für die

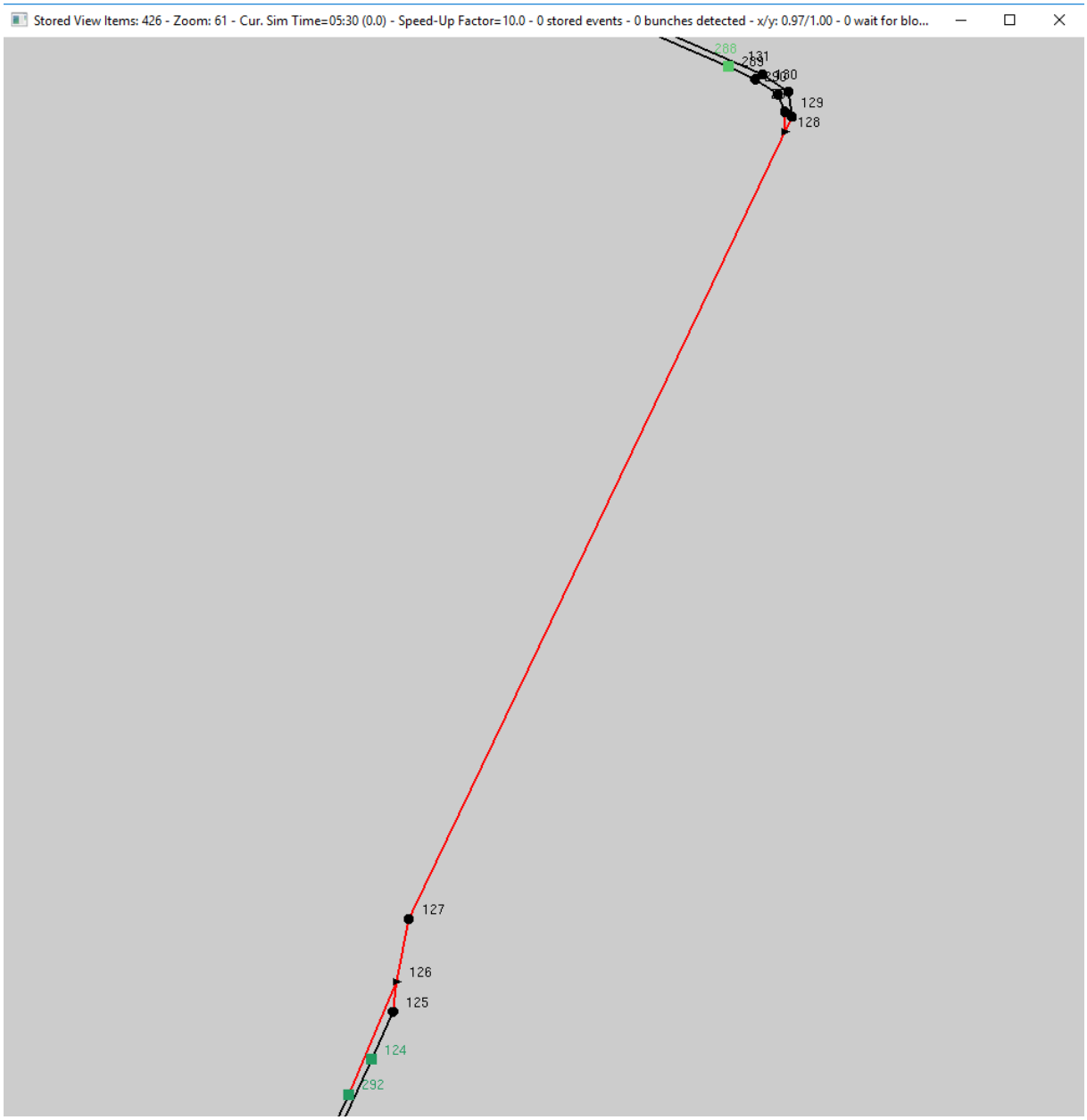

Abbildung 19: Infrastruktur der Linie 13: Infrastrukturblock-Darstellung (in rot) zwischen den Haltepositionen Alaubplatz und Görlitzer Straße

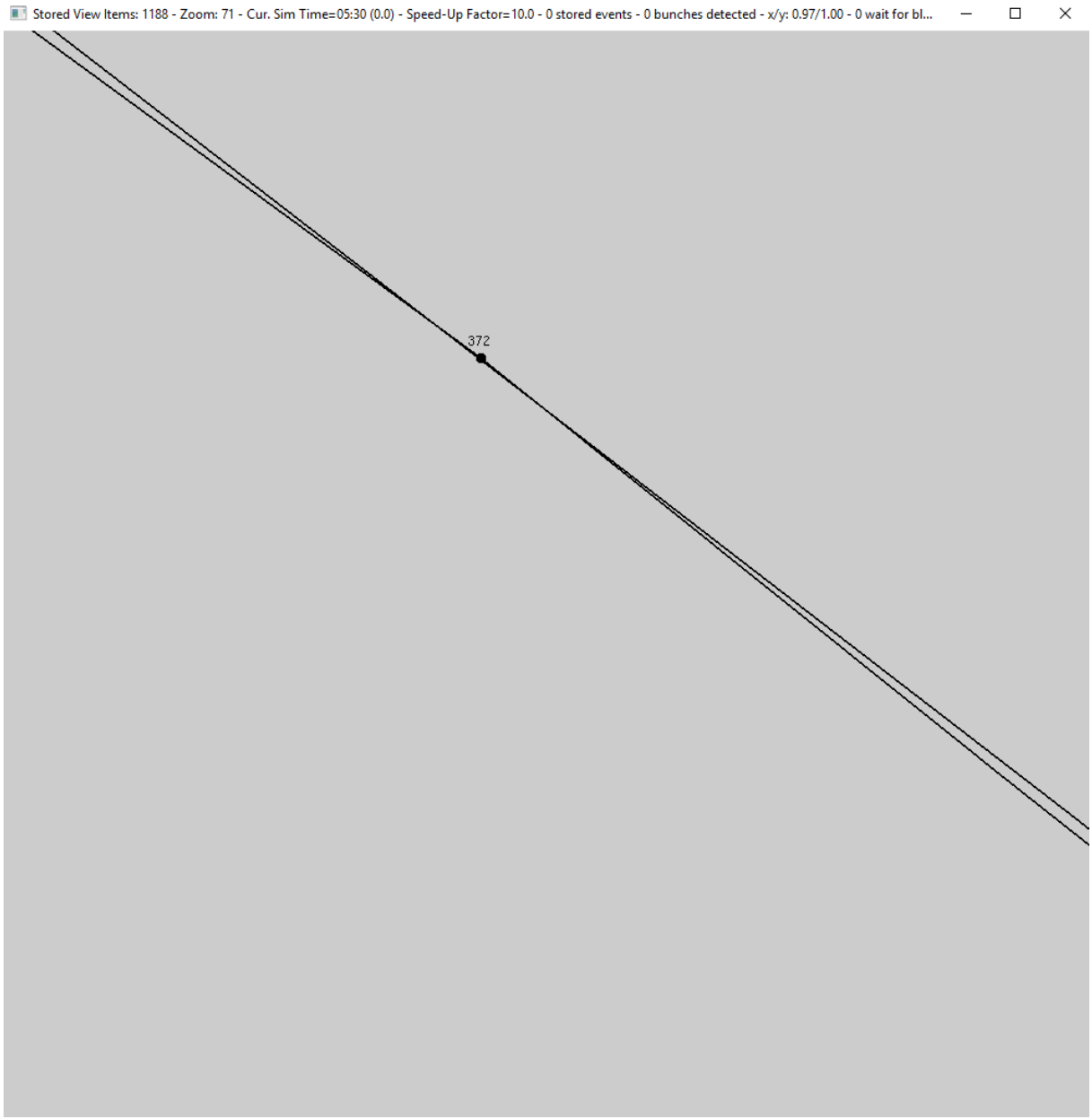

Abbildung 20: Problem am Knoten mit der ID 372 (Trackpos 253)

```
1  this->ARC[52].specify(52, 52, 459);
2  this->ARC[464].specify(464, 459, 53);
3  this->ARC[361].specify(361, 363, 460);
4  this->ARC[465].specify(465, 460, 364);
```

Abbildung 20: Modifizierte und ergänzte ***BUSSIM_ARC***-Instanzen

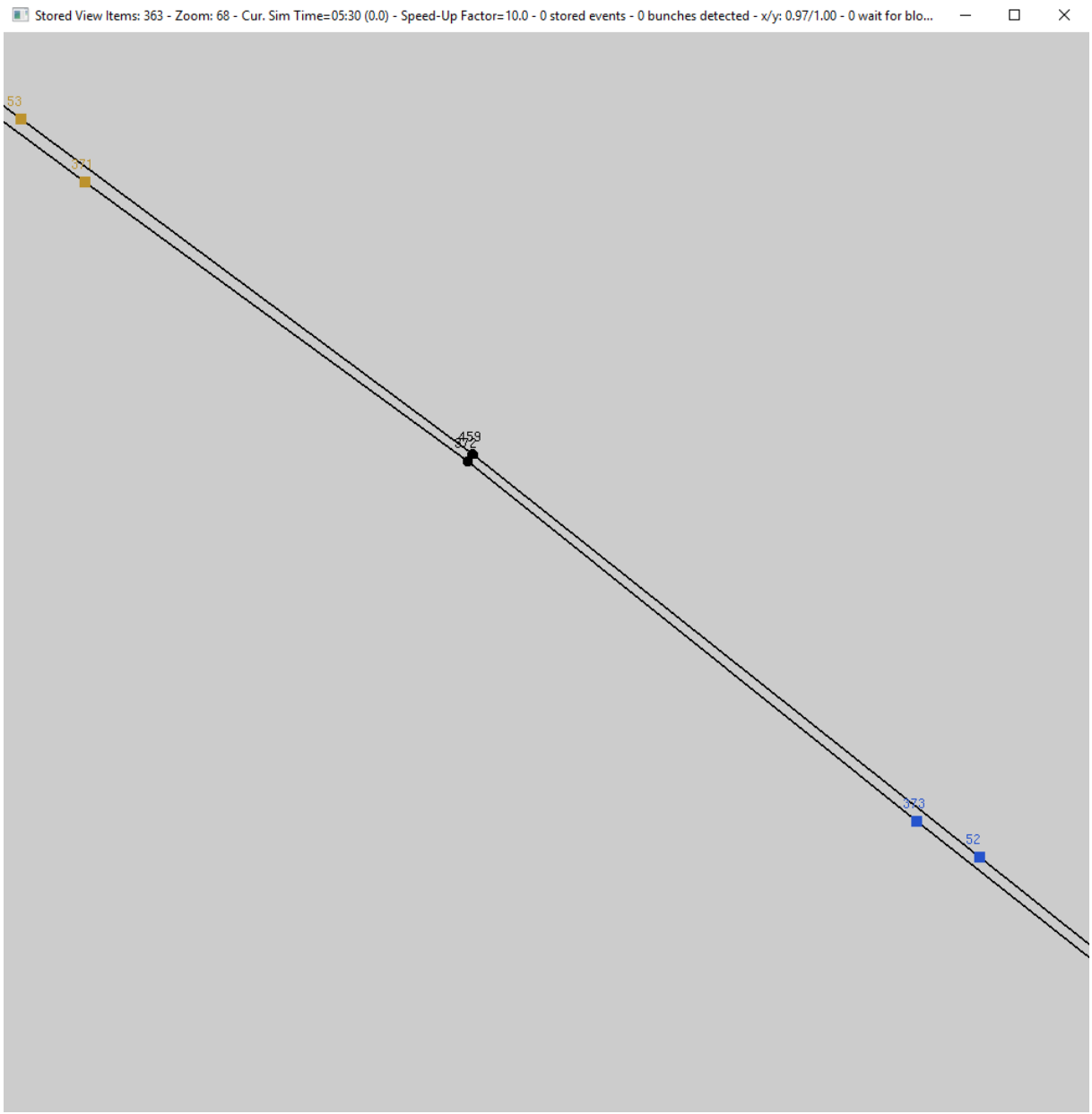

Abbildung 21: Gelöstes Problem am Knoten mit der ID 372 (Trackpos 253)

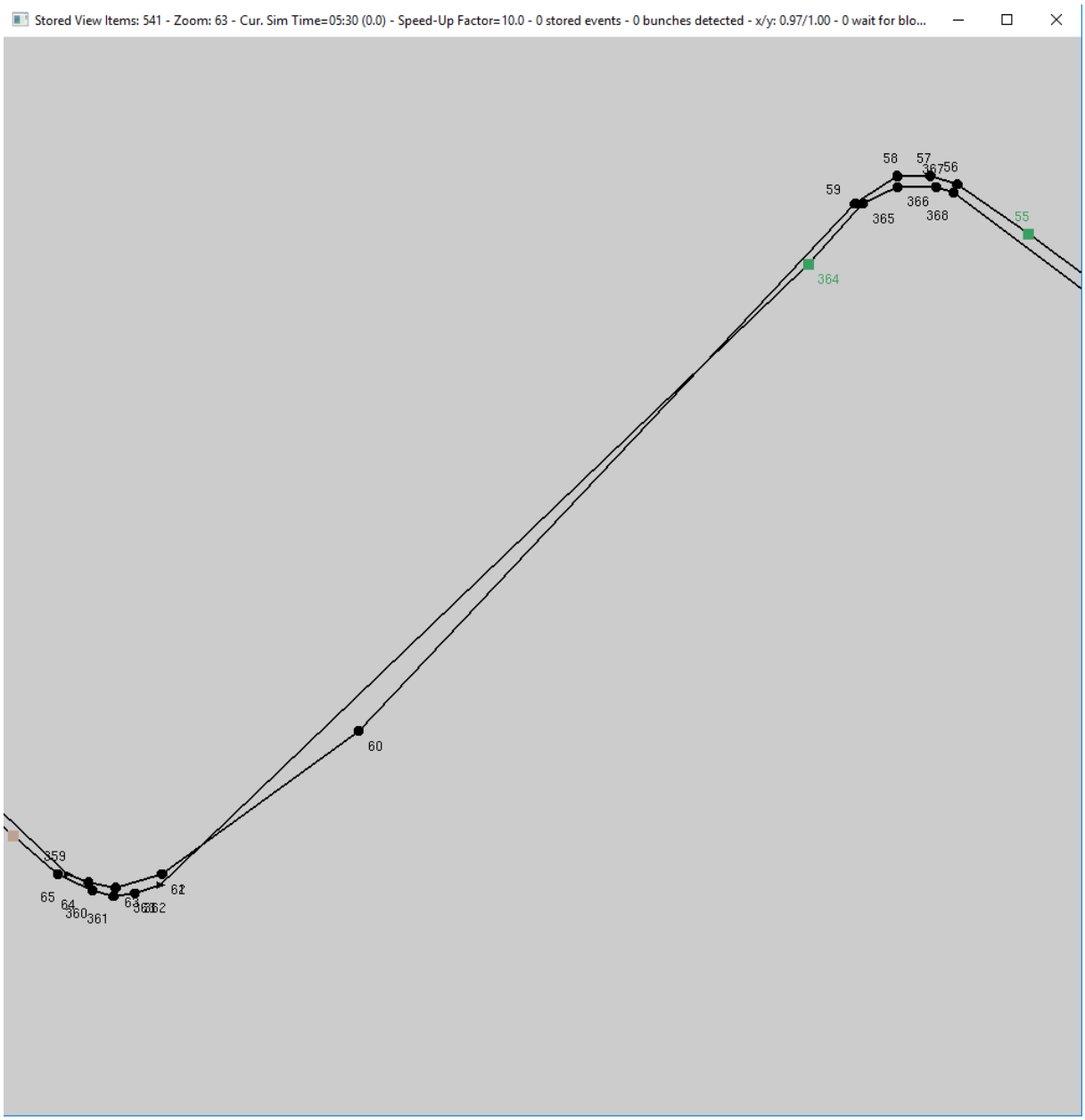

Abbildung 22: Problem am Knoten mit der ID 60 (Trackpos 44)

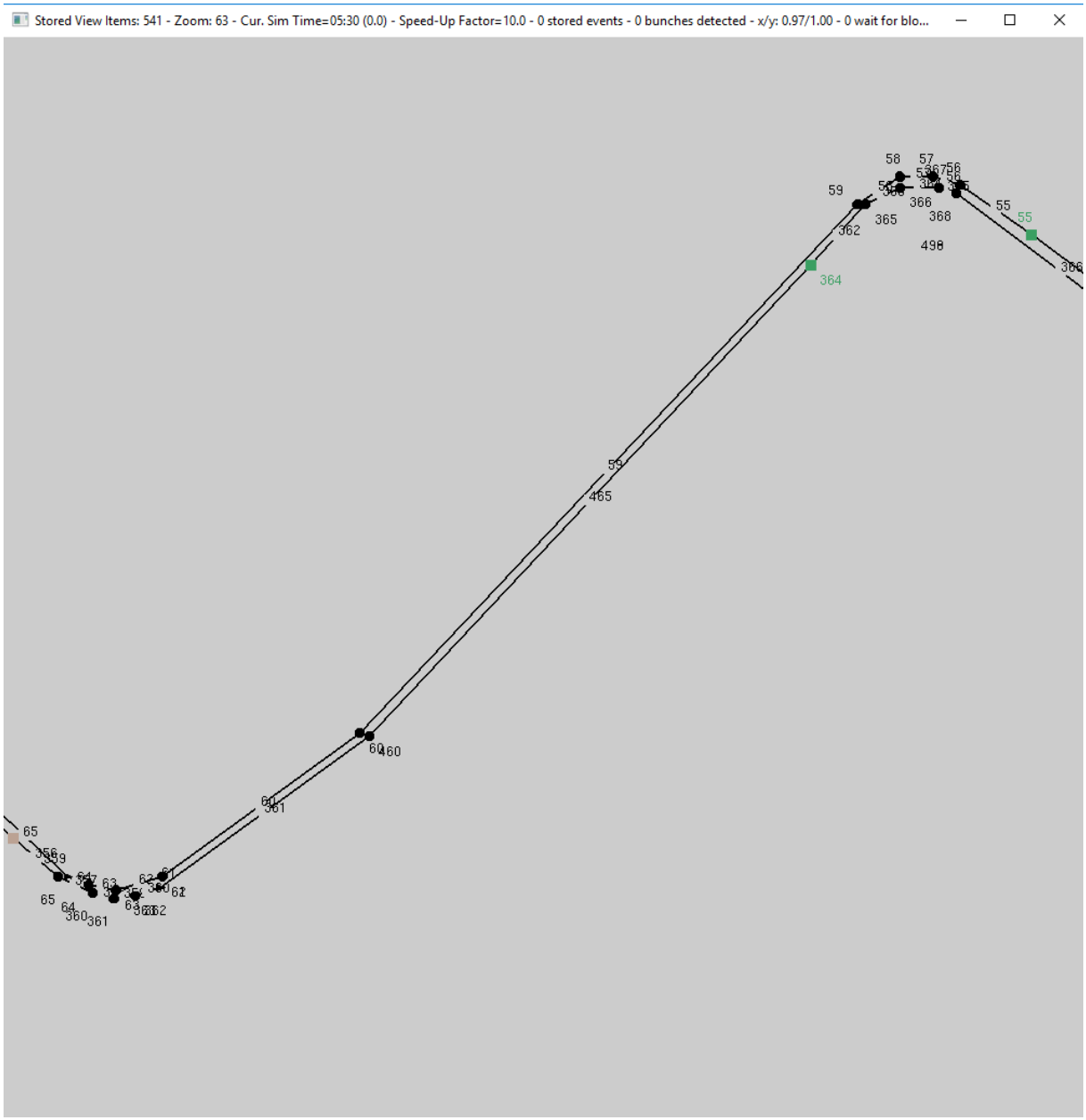

Abbildung 23: Gelöstes Problem am Knoten mit der ID 60 (Trackpos 44)

Richtung von Kaditz nach Prohlis zwischen den Knoten 363 und 364 etwas süd-ostlich vom Knoten 60 ein Trackpos-Knoten vergessen. Dieser wird nun mit der ID=460 der Knoten-Liste hinzugefügt (Tab. 13). Zur Einbindung dieses Knotens muss der bisherige Pfeil mit der ID 361, der die Knoten 363 und 364 verknüpft hat, aufgesplittet werden. Dieser Pfeil startet nun im Knoten 363 und endet im neuen Knoten 460. Neu hinzugefügt werden muss der Pfeil, der dann den Knoten 460 mit dem Knoten 364 verbindet. Dieser Pfeil erhält die ID 465 (Tab. 20). Abschließend ist der Konstruktor des Netzwerk-Objekts zu `BUSSIM_NETWORK NET(44,461,466,1,0,0,0,0);` anzupassen, um den zusätzlichen Knoten und den zusätzlichen Pfeil abspeichern zu können. Damit ist das Problem gelöst (Abb. 23).

7 Einrichten von Linienverläufen

Wir haben bisher die Infrastruktur im Simulationsprogramm hinterlegt, in der Fahrzeuge 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.

Wie wir bereits in Abschnitt 6 erwähnt haben, muss ein Fahrzeug beim Wechsel eines bedienten Linienverlaufs seine Fahrtzielanzeige ändern. Daher muss jeder Linienverlauf an einem `trackpos`-Knoten starten und an einem `trackpos` enden, damit die wartenden Fahrgäste ein korrekt beschildertes Fahrzeug auf sich zufahren sehen. Der Startknoten eines Linienverlaufs sollte zwischen der Ankunfts- und der Abfahrtsplattform der Starthaltestelle liegen. Der letzte Knoten eines Linienverlaufs sollte dann zwischen der Ankunfts- und der ABfahrtsplattform der Zielhaltestelle liegen.

7.1 Konzeptionelle Überlegungen und Datensammlung

Im Fall der Linie 13 müssen wir somit zunächst das Linienverlaufs-Objekt `this→LINE[0]` erzeugen. Als Startknoten nehmen wir den trackpos-Knoten mit der ID 431 (Abb. 14). Als Zielknoten verwenden wir den trackpos-Knoten mit der ID 450 (Abb 17). Dann ist der erste in diesem Linienverlauf zu durchfahrene Abschnitt durch den Pfeil $(431; 0)$ dargestellt, d.h. durch die ***BUSSIM_ARC***-Instanz `this→ARC[433]`. Anschließend sind nacheinander die Instanzen `this→ARC[0]`, ..., `this→ARC[52]`, `this→ARC[464]`, `this→ARC[53]`, ..., `this→ARC[207]` zu durchqueren. Um von der Ausstiegs-Plattform (Knoten mit der ID 208) zum Zielknoten mit der ID 450 zu gelangen, sind nun noch die Pfeile repräsentiert durch die ***BUSSIM_ARC***-Instanzen `this→ARC[443]`, ..., `this→ARC[453]` abzufahren.

Analog konstruieren wir den "Rückweg" von Kaditz/Riegelplatz nach Prohlis. Hierfür müssen wir ein zweite Instanz `this→LINE[1]` des Typs ***BUSSIM_LINE*** nutzen. Als Startknoten nutzen wir den Knoten mit der ID 450 und der Zielknoten soll die ID 431 besitzen. Damit ist die erste auf dem Weg von Kaditz nach Prohlis zu durchfahrene ***BUSSIM_ARC***-Instanz `this→ARC[454]`. Es folgen die Instanzen `this→ARC[208]`, ..., `this→ARC[289]`, `this→ARC[430]`, `this→ARC[428]`, `this→ARC[429]`, `this→ARC[431]`, `this→ARC[290]`, ..., `this→ARC[361]`, `this→ARC[465]`, `this→ARC[362]`, ..., `this→ARC[427]`, `this→ARC[432]`.

7.2 Quellcode-Ergänzungen

Nachdem wir zunächst die Daten zur Spezifikation der beiden Linienverläufe zusammengetragen haben, müssen wir diese im B-u-S-Sim -Projekt hinterlegen. Die zu spezifizierenden Attribute eines ***BUSSIM_LINE***-Objekts stellt Tabelle 14 zusammen. In der Funktion `BUSSIM_NETWORK::specify_lines()` muss nun für jedes Linienverlaufsobjekt die Konfiguration vorgenommen und die Sequenz der Streckenabschnitte hinterlegt werden.

Die Abbildung 24 zeigt den Quellcode zur Spezifikation der beiden Richtungs-Linienverläufe der Straßenbahnlinie 13 in Dresden. Zunächst wird der Linienverlauf `LINE[0]` (Prohlis nach Kaditz) konfiguriert (5). 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 13 setzen (zweiter Parameter). Anschließend wird die Anzeigefarbe

```
1  void BUSSIM_NETWORK::specify_lines(void)
2  {
3
4      // type in your line specifications here
5      this->LINE[0].configure(0,13,0.0,0.0,1.0,"Linie 13: Prohlis->Kaditz");
6       this->LINE[0].append_arc(this->ARC[433]);
7       this->LINE[0].append_arc(this->ARC[0]);
8       this->LINE[0].append_arc(this->ARC[1]);
9      ...
10     this->LINE[0].append_arc(this->ARC[52]);
11      this->LINE[0].append_arc(this->ARC[464]);
12      this->LINE[0].append_arc(this->ARC[53]);
13  ...
14      this->LINE[0].append_arc(this->ARC[207]);
15      this->LINE[0].append_arc(this->ARC[443]);
16     ...
17     this->LINE[0].append_arc(this->ARC[453]);
18
19     this->LINE[1].configure(1,13,0.0,1.0,1.0,"Linie 13: Kaditz->Prohlis");
20      this->LINE[1].append_arc(this->ARC[454]);
21      this->LINE[1].append_arc(this->ARC[208]);
22     ...
23     this->LINE[1].append_arc(this->ARC[431]);
24      this->LINE[1].append_arc(this->ARC[290]);
25     ...
26     this->LINE[1].append_arc(this->ARC[361]);
27      this->LINE[1].append_arc(this->ARC[465]);
28      this->LINE[1].append_arc(this->ARC[362]);
29     ...
30     this->LINE[1].append_arc(this->ARC[427]);
31      this->LINE[1].append_arc(this->ARC[432]);
32
33  }
```

Abbildung 24: Quellcode der Funktion `BUSSIM_NETWORK::specify_lines()` zur Linienverlaufsspezifikation

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Linienverlaufs
SERVICE	int	frei belegbarer Wert zur Bündlung von Linienverläufen
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

Tabelle 14: Festzulegende Attribute für eine Instanz des `BUSSIM_LINE`-Objekts

für diesen Linienverlauf nach RGB-Schema [8] auf blau gesetzt (3.-5. Eingabewert). Der 6. Eingabewert stellt eine verbale Beschreibung des Linienverlaufs dar. In den nachfolgenden Zeilen werden sukzessive die nacheinander zu durchfahrenen Streckenabschnitte in der vorgegebenen, o.a. Reihenfolge dem Linienverlauf `LINE[0]` hinzugefügt. Ab Zeile 19 wird der Linienverlauf `LINE[1]` deklariert. Zu beachten ist, dass hierfür der eindeutige ID-Wert in Zeile 19 auf 1 gesetzt wird (1. Parameterwert) und die Farbe auf hellblau nach RGB-Schema eingestellt wird. Die "..." müssen im Projekt mit den passenden `append_arc`-Befehlen gefüllt werden.

$$class\ BUSSIM\_NETWORK\ NET(44, 461, 466, 1, 2, 0, 0, 0); \tag{4}$$

In der Datei `main.cpp` muss nun noch das fünfte Argument des Netzwerk-Konstruktors auf die Anzahl der insgesamt vorhandenen Linien gesetzt werden. In unserem Fall müssen wir also den Konstruktor wie in (4) gezeigt modifizieren, damit wir die beiden Linienverläufe für die Linie 13 einrichten können.

Abbildung 25 zeigt den fertig codierten Verlauf der beiden Richtungs-Linienverläufe von Prohlis nach Kaditz (blau) und von Kaditz nach Prohlis dargestellt als hellbaluer Linienzug. 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 25). 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 Ergänzung der beiden Linienverläufe zu einem "Rundkurs" wird durch die beiden Darstellungen in Abb. 26 erkennbar. In Prohlis beginnt der in blau dargestellte Linienverlauf, der in Kaditz endet und dort unmittelbar in den hellblauen Linienverlauf übergeht. Dieser endet dann wieder in Prohlis.

[8] `https://www.farb-tabelle.de`

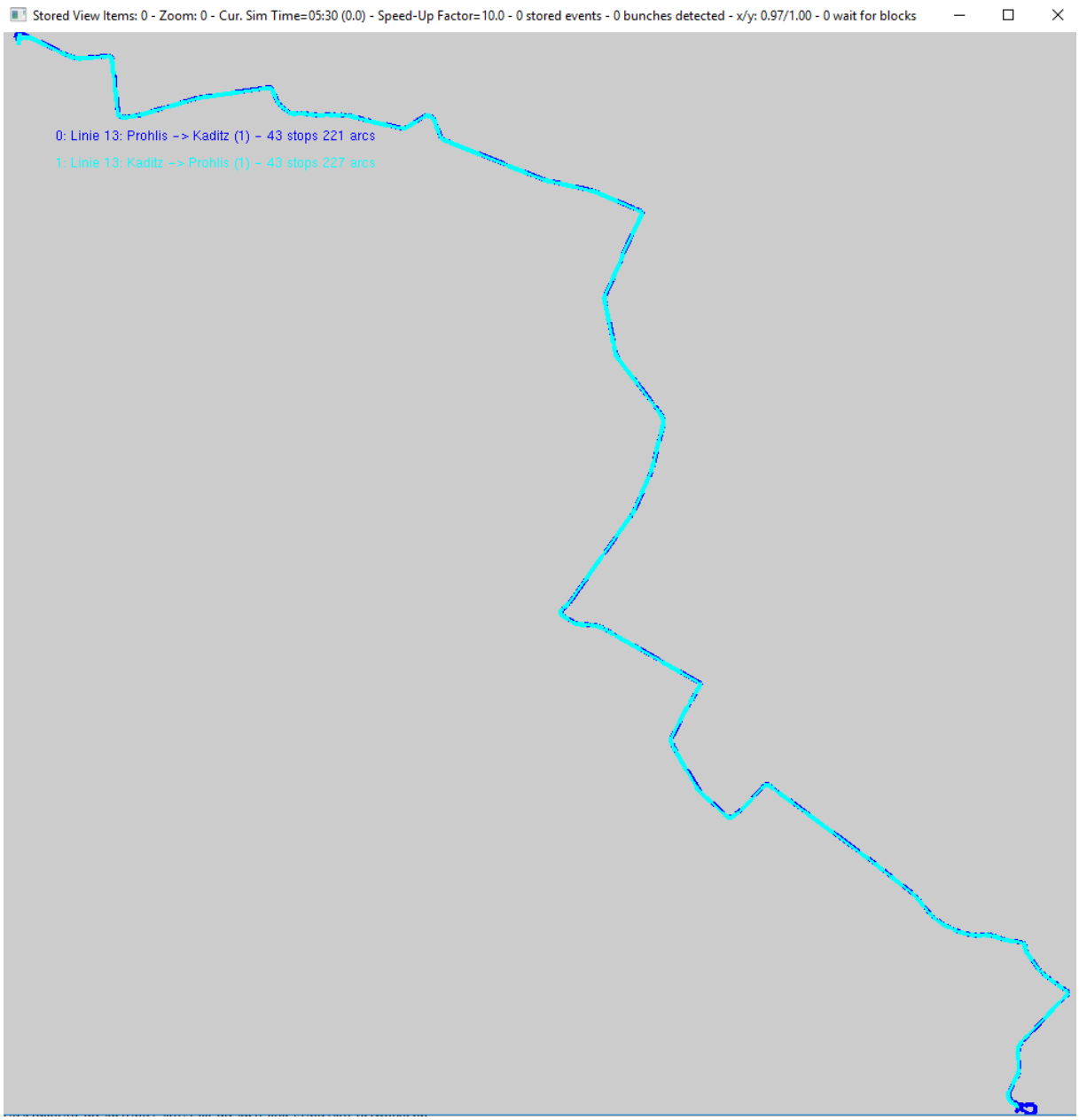

Abbildung 25: Simulationsfenster mit den beiden spezifizierten Linienverläufen der Linie 13

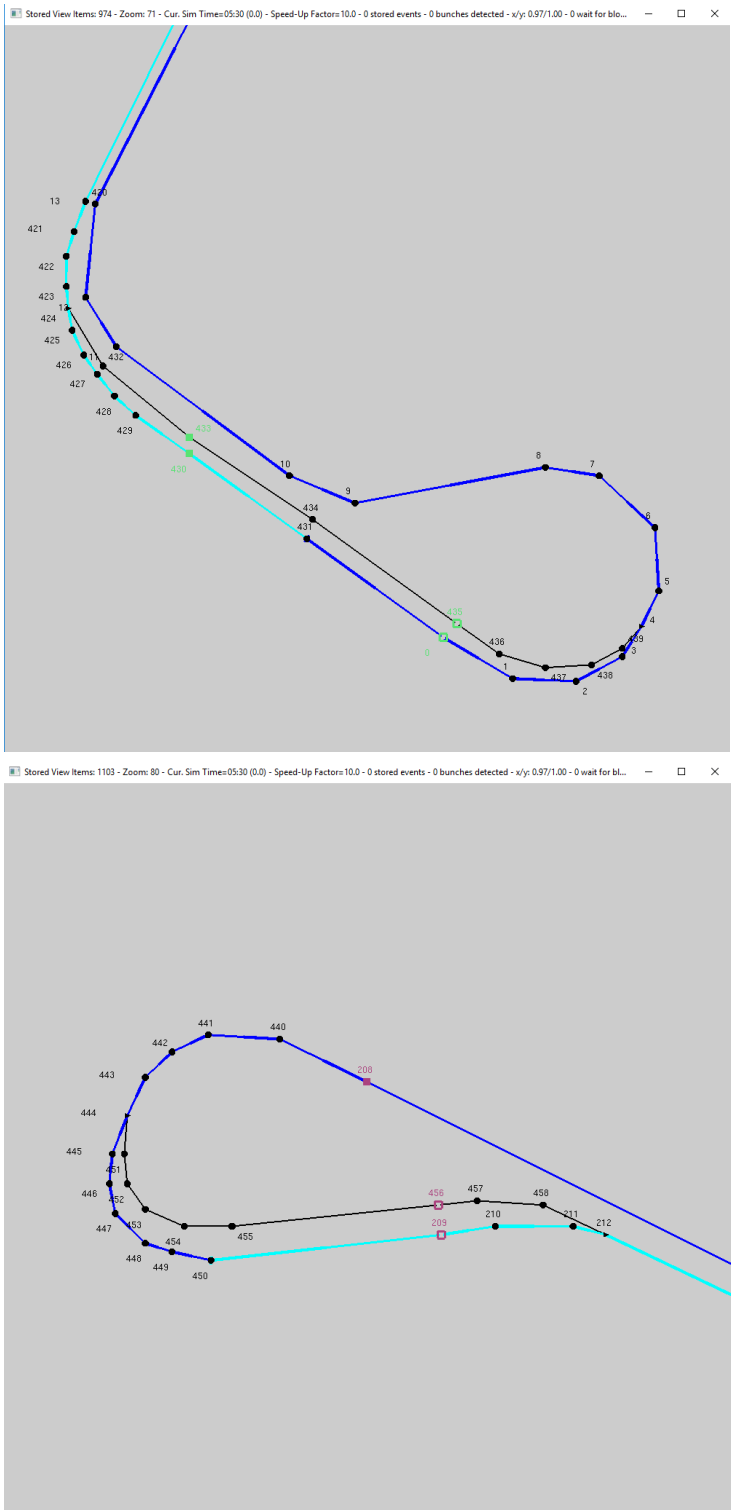

Abbildung 26: Simulationsfenster mit den beiden spezifizierten Linienverläufen der Linie 13 an den Wendeschleifen in Prohlis (oben) und Kaditz (unten)

Abschließend ist zu erwähnen, dass mehrere Linienverläufe sich einen Streckenabschnitt ”teilen“ können.

8 Spezifikation der Fahrzeuge

Wir haben bisher den Infrastrukturgraphen in B-u-S-Sim hinterlegt und darin Linienverläufe spezifiziert, entlang deren Fahrzeuge Fahrgäste befördern sollen. Die Fahrzeuge stellen die mobilen Komponenten im simulierten Netzwerk dar. Sie dienen der Beförderung von Fahrgästen. Um Fahrzeuge in die Simulation sinnstiftend zu integrieren, muss deren Einsatz geplant werden. InB-u-S-Sim sind hierfür drei Schritte notwendig.

- Im Rahmen der **Fahrzeugkonfiguration** werden die wesentlichen Eigenschaften eines Fahrzeugs festgelegt (vgl. 8.1).
- Während der anschließenden **Umlaufplanung** spezifizieren Sie die detaillierten Fahrwege der Fahrzeuge durch die modellierte Infrastruktur (vgl. 8.2).
- Anschließend legen Sie im Zuge der **Fahrplan-Erstellung** fest, wann genau die Fahrzeuge entlang der einzelnen Streckenabschnitte fahren (vgl. 8.3).

Wir demonstrieren in diesem Kapitel anhand eines übersichtlichen Beispiels, wie die genauen Vorbereitungs-Schritte des Fahrzeug-Einsatzes aussehen und wie die einzelnen Schritte ineinander greifen. Ziel ist es, ein einzelnes Straßenbahn-Fahrzeug eine komplette Rundreise entlang der Linie 13 in beiden Richtungen absolvieren zu lassen. Dafür soll das Fahrzeug an der Gleisschleife Prohlis initial positioniert werden. Von dort soll mit Halt an allen Unterwegs-Haltepositionen zur Gleisschleife nach Kaditz gefahren werden. Nach einer kurzen Pause soll dann das Fahrzeug von Prohlis wieder zurück zu seiner Ausgangsposition in der Gleisschleife Prohlis fahren und wieder an allen Unterwegs-Haltepositionen halten. In dieser Situation muss für das Fahrzeug ein Umlauf definiert werden, der zunächst das Durchfahren des Linienverlaufs `Line[0]` erfordert, gefolgt von einem Durchfahren des Linienverlaufs `Line[1]`.

8.1 Konfiguration der Fahrzeuge

$$class\ BUSSIM\_NETWORK\ NET(44, 461, 466, 1, 2, 1, 0, 0); \tag{5}$$

Ein Fahrzeug wird in B-u-S-Sim durch ein ***BUSSIM_VEHICLE***-Instanz repräsentiert. Vor deren Nutzung müssen wir festlegen, wieviele Fahrzeuge wir nutzen möchten und diese Zahl als sechstes Argument in den Netzwerk-Konstruktor in der Datei main.cpp eintragen (5).
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.
Tabelle 15 enthält die Attribute, die für jedes Fahrzeug individuell festgelegt werden müssen. Über die `ID` wird das Fahrzeug identifiziert und die `vehicle_category` legt fest, wie das Fahrzeug in der Simulation angezeigt wird (ausgefülltes Rechteck für eine Tram bzw. nicht-ausgefülltes Rechteck für einen Bus). Die `velocity` bestimmt, wie schnell sich das Fahrzeug durch das Netzwerk im Durchschnitt bewegt (Angabe in km/h). Die `capacity` gibt an, wieviele Fahrgäste Platz im Fahrzeug haben. Diese Information wird u.a. genutzt, um die Aufenthaltszeit eines Fahrzeugs an einer Haltestelle zu bestimmen. Je höher die Anzahl der ein- und aussteigenden fahrgäste an einer Haltestelle im Verhältnis zur Kapazität des Fahrzeugs ist, desto länger ist die Aufenthaltszeit. Die letzten beiden Attribute `ON_ARC`sowie `parc_arc` legen die initiale sowie die gewünschte letzte Fahrzeug-Position im Netzwerk fest. Auf dem initialen Pfeil wird das Fahrzeug auf den Anfang gesetzt und startet von dort seine Fahrt. Das Fahrzeug muss seine Fahrt dann am Ende des Zielpfeils `parc_arc` beenden.
Abbildung 27 zeigt den kompletten Quellcode, mit dem das Fahrzeug `VEHICLE[0]` spezifiziert wird. Der ersten anzugebene Parameter ist die ID des Fahrzeugs (0). Der zweite Parameter identifiziert das Fahrzeug als Straßenbahn (`BUSSIM_VEHCAT_BUS` würde einen Bus spezifizieren). Der dritte Parameter legt die durchschnittlich für das Fahrzeug beobachtete Geschwindigkeit fest. Das Fahrzeug ist für

Attribut	Typ	Bedeutung bzw. Inhalt
ID	int	eindeutiger Schlüssel zur Identifikation eines Fahrzeugs
vehicle_category	int	legt fest, ob es sich um eine Tram (BUSSIM_VEHCAT_TRAM) oder einen Bus (BUSSIM_VEHCAT_BUS) handelt
velocity	double	durchschnittliche Geschwindigkeit des Fahrzeugs in km/h
capacity	int	max. Anzahl der Fahrgäste lt. Fahrzeugschein o.ä.
ON_ARC	BUSSIM_ARC	Streckenabschnitt, auf dem sich das Fahrzeug gerade befindet
parc_arc	BUSSIM_ARC	Streckenabschnitt, an dessen Ende das Fahrzeug sich am Ende der Simulation befinden soll

Tabelle 15: Attribute für das Objekt `BUSSIM_VEHICLE`

```
1 void BUSSIM_NETWORK::specify_vehicles(void)
2 {
3     // specify the available vehicles here
4     this->VEHICLE[0].configure(0,BUSSIM_VEHCAT_TRAM,20,100,this->ARC[433],
          this->ARC[432], this);
5
6 }
```

Abbildung 27: Code in der Funktion `BUSSIM_NETWORK::specify_vehicles()` zur Einrichtung des Fahrzeugs

die Beförderung von bis zu 100 Fahrgästen ausgelegt (vierter Parameter). Zu Beginn der Simulation soll das Fahrzeug am Anfang des Pfeils `ARC[433]` stehen und die finale Position des Fahrzeugs soll das Ende des Pfeils `ARC[432]` sein. Somit startet und endet die Bewegung des Fahzeugs am Knoten mit der ID431. Der siebte und letzte Parameter ist eine Referenz auf die aktuelle ***BUSSIM_NETWORK***-Instanz.

8.2 Festlegung der Fahrzeugumläufe

Ein Fahrzeugumlauf legt fest, welchen Weg ein Fahrzeug (wann) im Netzwerk abfährt (und wann es wo für welche Zeit hält). In B-u-S-Sim wird die Spezifikation der Fahrzeugumläufe in zwei Schritten durchgeführt. Im ersten hier beschriebenen Schritt wird ausschließlich der Weg von der Startposition (Am Beginn des Pfeils `ON_ARC`) durch das Netzwerk zur Endposition (am Ende des Pfeils `parc_arc`) spezifiziert. Dieser ohne Ausführungszeiten konstruierte Fahrzeugweg wird in einem ***BUSSIM_ROTATION***-Objekt gespeichert. Jedes Fahrzeug besitzt genau ein solches Objekt.
Der zu konstruierende Umlauf wird durch eine endliche Sequenz von sog. Aufgaben definiert. Eine Aufgabe $(i; LINE)$ ist dabei eine Kombination von einem Fahrzeug i und einem Linienverlauf $LINE$. Zur Erfüllung (Abarbeitung) dieser Aufgabe muss das Fahrzeug den kompletten Verlauf der Linie abfahren. Aufgaben werden in ***BUSSIM_DUTY***-Objekten abgespeichert. Ist ein Arbeitsauftrag durchgeführt, kann entweder der nächste hinterlegte Linienverlauf abgefahren werden oder das Fahrzeug bleibt stehen (Deaktivierung).
Damit ein Fahrzeug eine Sequenz von Arbeitsaufträgen abarbeiten kann, muss zunächst diese Aufgaben-Sequenz dem Fahrzeug zugewiesen werden. Hierfür wird die Methode `append_duty_to_vehicle_rotation(<i>,<LINE>)` des ***BUSSIM_NETWORK***-Objekts verwendet. Diese Befehle werden in der Methode `BUSSIM_NETWORK::specify_rotations()` gesammelt. Die Reihenfolge der einem Fahrzeug i zugewiesenen Aufträge entspricht der Reihenfolge der Aufrufe der Methode `append_duty_to_vehicle_rotation(<i>,<LINE>)`.

```
1 void BUSSIM_NETWORK::specify_rotations(void)
2 {
3     // Fahrzeug 0 soll entlang LINE[0] von Prohlis nach Kaditz fahren
4     this->append_duty_to_vehicle_rotation(0,0);
5     // Fahrzeug 0 soll entlang LINE[1] von Kaditz nach Prohlis fahren
6     this->append_duty_to_vehicle_rotation(0,1);
7 }
```

Abbildung 28: Konstruktion einen einfachen Umlaufs für das Fahrzeug mit der ID 0.

Abbildung 28 zeigt ein Beispiel-Umlaufplan für das oben spezifizierte Fahrzeug 0. Zunächst soll das Fahrzeug entlang des Linienverlaufs `LINE[0]` von Prohlis nach Kaditz fahren (Zeile 4). Direkt anschließend ist die Rückfahrt von Kaditz nach Prohlis entlang des Linienverlaufs `LINE[1]` vorgesehen (Zeile 6).

8.3 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. Die im Umlaufplan (inkl. der automatischen Ergänzungen) zu findende Sequenz von ***BUSSIM_ARC***-Objekten wird dann sukzessive durchfahren. Somit ist zunächst für jedes Fahrzeug die Aktivierungszeit zu setzen. Diese Aufgabe wird in der Methode `BUSSIM_NETWORK::specify_vehicle_activation_times()` erledigt. Dort wird der

```
1 void BUSSIM_NETWORK::specify_vehicle_activation_times(void)
2 {
3     // set the vehicle activation times
4     this->VEHICLE[0].set_activation_time(0);
5
6 }
```

Abbildung 29: Festlegung der AKtivierungszeit eines Fahrzeugs

Befehl `this->VEHICLE[1].set_activation_time(X);` eingefügt, wobei X für den Aktivierungszeitpunkt steht. Abb. 29 zeigt den entsprechenden Code-Snippet für die AKtivierung des Fahrzeugs 0 zum Zeitpunkt 0 dar. **Hinweis: Da der Start einer Simulation ohne aktivierte Fahrzeuge keinen Sinn ergibt, muss für mindestens ein Fahrzeug die AKtivierungszeit 0 festgelegt werden.**
Für jeden erreichten Knoten gemäß festgelegten Umlauf wird in Abhängigkeit vom Typ dieses Knotens eine Aufenthaltszeit eingeplant. Standardmäßig wird für jede reguläre Halteposition die in der globalen Variablen `BUSSIM_STANDARD_WAITING_TIME` gespeicherte Haltestellenaufenthaltszeit vorgesehen. Der Wert dieser Konstanten kann in der Datei `bussim_global.h` verändert werden und steht standardmäßig auf 0.75 Zeiteinheiten (Simulationsminuten). Lediglich für Depot-Knoten wird eine abweichende Aufenthaltszeit vorgesehen. Diese ist in den Einstellungen in der Datei `global.h` im Ausdruck `BUSSIM_DEPOT_WAITING_TIME` gespeichert und standardmäßig auf 5.0 Zeiteinheiten eingestellt. Wird ein Knoten vom Typ `trackpos` oder `switch` erreicht, wird gar nicht angehalten, sondern einfach in den nächsten Pfeil eingefahren (sofern dies möglich ist).
Für die Erzeugung der Fahrpläne sind keine weiteren Benutzeraktivitäten notwendig.

```
this->PT[431].specify_trackpos(13.7981, 50.99955, 0, "Trackpoint 298", "TP298", this);   (6)
this->PT[450].specify_trackpos(13.68647, 51.08335, 42, "Trackpoint 314", "TP314", this); (7)
```

Damit in der Simulation für das Fahrzeug jeweils die korrekte Zielhaltestelle angezeigt werden kann, müssen wir abschließend noch die Endpunkte der Linienverläufe den Endhaltestellen zuordnen. Hierfür ordnen wir die ***BUSSIM_POINT***-Instanz mit der ID 431 durch Angabe 0 als dritten Spezifikationsparameter die Haltestelle Prohlis Gleisschleife (ID=0) zu (7). Analog dazu ordnen wir die ***BUSSIM_POINT***-Instanz mit der ID 450 der Haltestelle Kaditz / Riegelplatz (ID=42) zu (7)

8.4 Start der Simulation mit einem Fahrzeug

Nach dem alle vorgenannten Anpassungen und Ergänzungen des Quellcodes vorgenommen worden sind, kann das Projekt übersetzt und gestartet werden. Nach dem Start kann das eingerichtete Fahrzeug sowohl im Simulationsfenster als auch im Fahrzeugfenster bestrachtet werden. Im Simulationsfenster ist das Fahrzeug als Straßenbahn-Fahrzeug als blaues Rechteck am Beginn des blauen Linienverlaufs dargestellt. Sobald der Automatik-Modus durch <a> aktiviert wird, zieht das Fahrzeug bis zum Abfahrtpunkt vor und verweilt dort 5 Zeiteinheiten. Anschließend fährt es entlang des Graphen automatisch weiter und hält an allen Unterwegshaltepunkten.
Auch im Fahrzeugfenster ist das Fahrzeug 0 dargestellt. Hier werden in einem Rechteck viele relevante Informationen zum Fahrzeug zusammengestellt präsentiert. Oben links im Rechteck steht die Fahrzeug ID. Das ”T“ links von der Nummer zeigt an, dass es sich bei dem Fahrzeug um eine Straßenbahn handelt. Die Prozentzahl oben rechts im Rechteck stellt die aktuelle Fahrzeug-Auslastung relativ zur bei der Spezifikation angegebenen maximalen Fahrgastzahl dar. In der Mitte des Rechtecks steht links vom →-Symbol die ID des aktuell befahrenen Linienverlaufs. Rechts vom →-Symbol wird das Kürzel der aktuell angesteuerten Endhaltestelle dieses Linienverlaufs angezeigt. Zusätzlich ist das Rechteck in der Farbe des aktuell bedienten Linienverlaufs eingefärbt. Der Fortschrittsbalken an der linken Seite des Rechtecks

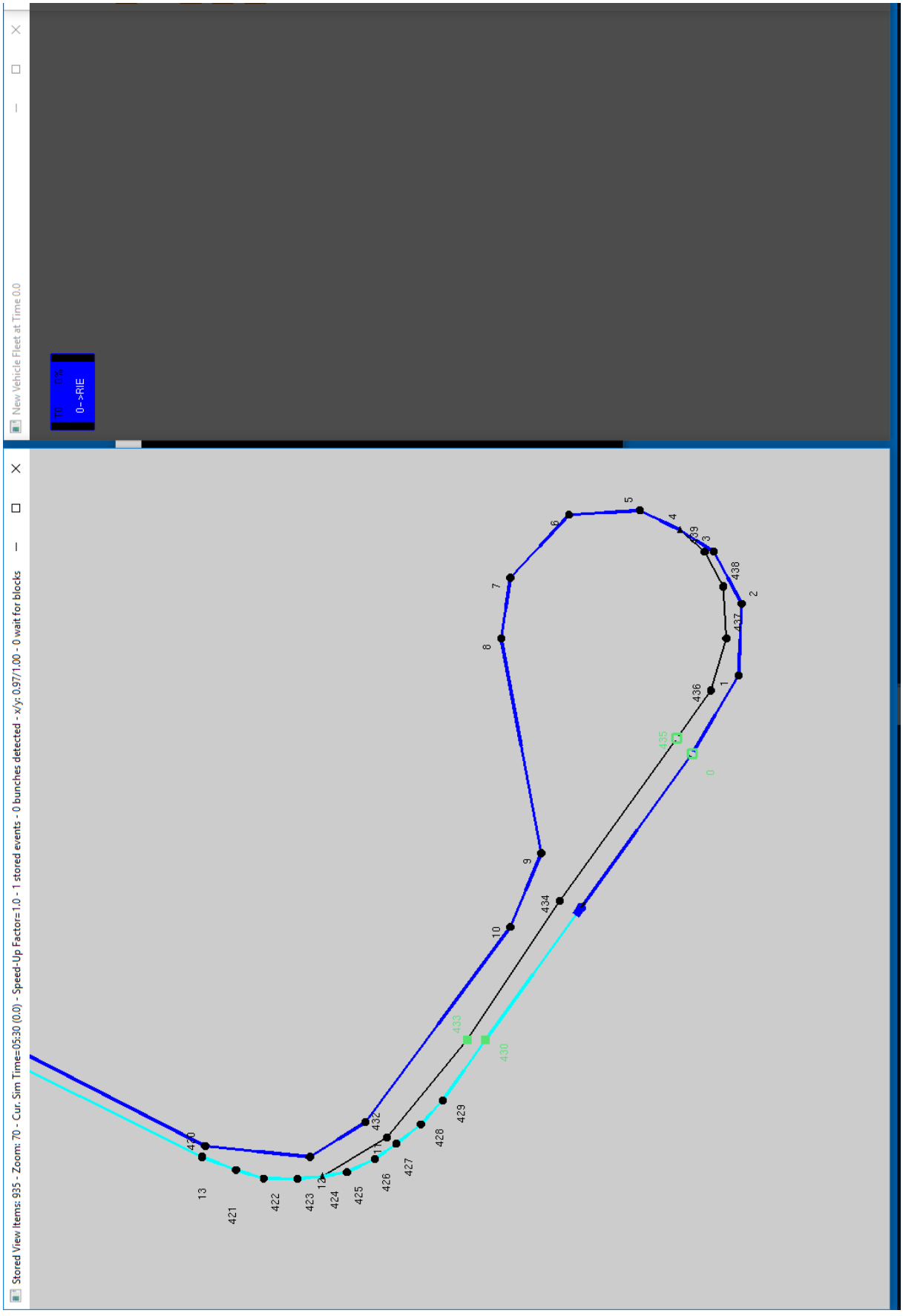

Abbildung 30: Start-Zustand der Simulation nach Einrichtung eines Fahrzeugs mit Aktivierungszeitpunkt 0

```
1       class BUSSIM_NETWORK NET(44,461,466,1,2,2,0,0);
```

Abbildung 31: Konstruktur für das Szenario mit 2 Fahrzeugen

```
1  void BUSSIM_NETWORK::specify_vehicles(void)
2  {
3      // this procedure is the place where the vehicle specifications (ID as
           well as velocity) are specified
4
5      // specify the available vehicles here
6      this->VEHICLE[0].configure(0,BUSSIM_VEHCAT_TRAM,20,100,this->ARC[433],
           this->ARC[432], this);
7      this->VEHICLE[1].configure(1,BUSSIM_VEHCAT_TRAM,20,100,this->ARC[454],
           this->ARC[453], this);
8  }
```

Abbildung 32: Quellcode zur Spezifikation von zwei Fahrzeugen

zeigt die aktuelle Prozentzahl der Abarbeitung des Fahrzeug-Umlaufs an. Der Fortschrittsbalken an der rechten Seite des Rechtecks zeigt die aktuelle Fahrzeugauslastung visuell an (grün, gelb, rot).

8.5 Hinzufügen eines zweiten Fahrzeugs

Wir möchten nun ein zweites Fahrzeug zur Simulation hinzufügen. Es erhält die ID 1. Dieses Fahrzeug fährt zunächst von Kaditz nach Prohlis und anschließend von Prohlis zurück nach Kaditz. Dieses Fahrzeug soll zum Zeitpunkt 33 aktiviert werden. Die sonstigen Eigenschaften sind identisch mit dem Fahrzeug 0. Als Startpfeil ist nun für das zweite Fahrzeug die Pfeil-Instanz `this->ARC[454]` festzulegen und als Zielpfeil die Instanz `this->ARC[453]`.
Abb. 31 zeigt den modifizierten Konstruktor für die ***BUSSIM_NETWORK***-Instanz an. Der Wert "2"der 6. Komponente stellt sicher, dass zwei ***BUSSIM_VEHICLE***-Instanzen in der ***BUSSIM_NETWORK***-Instanz eingerichtet werden.
Der Quellcode der in Abb. 32 gezeigten ergänzten `specify_vehicles`-Methode positioniert das zweite Fahrzeug zu Beginn in der Wendeschleife in Kaditz und legt als Ziel ebenfalls die Wendschleife in Kaditz fest (Zeile 7).
Für das zweite Fahrzeug mit der ID=1 legt die in Abb. 33 gezeigte Methode `specify_rotations` in den Zeilen 5-6 den Umlauf fest. Zunächst ist der Linienverlauf `LINE[0]` abzufahren, anschließend der Linienverlauf `LINE[1]`.
Mit der Instruktion in Zeile 5 in der in Abb. 34 abgebildeten Prozedur `specify_vehicle_activation_times` wird die Aktivierungszeit des zweiten Fahrzeugs auf den Zeitpunkt 32 festgelegt. Bei dieser Aktivierungszeit des zweiten Fahrzeugs würden sich die beiden Fahrzeuge zwischen den Halte-

```
1  void BUSSIM_NETWORK::specify_rotations(void)
2  {
3      this->append_duty_to_vehicle_rotation(0,0);
4      this->append_duty_to_vehicle_rotation(0,1);
5      this->append_duty_to_vehicle_rotation(1,1);
6      this->append_duty_to_vehicle_rotation(1,0);
7  }
```

Abbildung 33: Quellcode zur Spezifikation der Umläufe der beiden Fahrzeuge

```
1 void BUSSIM_NETWORK::specify_vehicle_activation_times(void)
2 {
3     // set the vehicle activation times
4     this->VEHICLE[0].set_activation_time(0);
5     this->VEHICLE[1].set_activation_time(32);
6 }
```

Abbildung 34: Quellcode zur Spezifikation der Fahrzeug-Aktivierungszeiten

stellen Alaunpark und Görlitzer Straße ungefähr zum Zeitpunkt 79 begegnen. Da die Strecke zwischen diesen beiden Haltestellen aber eingleisig ist dieser Abschnitt durch einen Infrastruktur-Block gesichert wird, muss das später den Block erreichende Fahrzeug warten. Die Protokoll-Datei enthält entsprechende Einträge.

Teil IV
Aufhübschen der Simulation

9 Things of Interests

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

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 16 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 16
width	double	die Standardbreite eines offenen Polygonzugs wird reduziert (< 1) oder vergrößert (> 1)

Tabelle 17: 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 17). 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 35.
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 35: Spezifikation der Sequenz der Geo-Koordinaten, die den Verlauf der Elbe im Großraum Dresden beschreiben

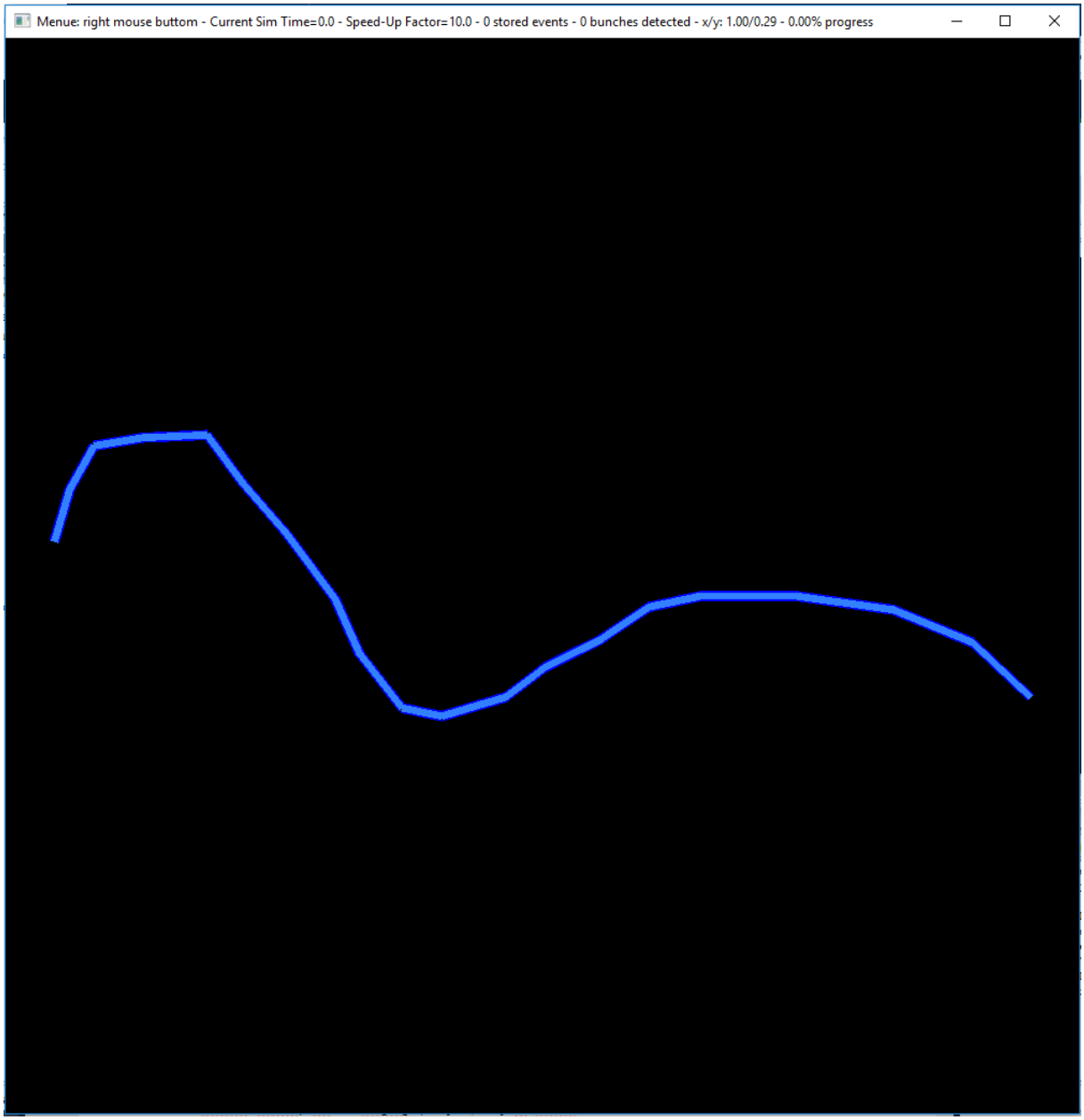

Abbildung 36: Elbe als TOI in der Simulationskarte

bare Programm in `Code::Blocks` . Starten Sie anschließend das erstellte Simulationsprogramm. Klicken Sie anschließend mit dem Mauszeiger in das Simulationsfenster. Abbildung 36 zeigt die graphische Darstellung der Elbe im Simulationsprogramm.

10 Using a Map File as Simulation Window Background

The specification of ***TOI***-objects improve the orientation in the simulation window. However, a detailled representation of a ***TOI***-object is hardly possible. To overcome this shortcoming, we present another approach to close the gap between the abstract network representation of the simulation and a realistic environment. This chapter explains the necessary steps to create a map of the covered simulation area. We follow the core idea to use electronic map data from openstreetmap to create a bitmap-file that fits with the geographic area shown in the simulation.
Section 10.1 guides you through the necessary steps to prepare your B-u-S-Sim -project for the incorporation of the map background. Section 10.2 explains the creation of the background file. Section 10.3 teaches you to integrate the created background file in your B-u-S-Sim -project.

10.1 Preparations

We are in need of an additional external library that provides all instructions to show a graphic file as background since the already incorporated `freeglut` library. We are going to use the library Simple OpenGL Image Library 2 (`SOIL2`).
First, you have to download the source code of `SOIL2` from the website at `https://github.com/SpartanJ/SOIL2`. Open this website and select the option `Download ZIP` by clicking on the green buttom on top at the right side. Save the zip-file under the name `SOIL2-master.zip`.
Second, you need to unpack the downloaded zip-file. The unpacked package contains a subfolder `src/SOIL2`. It is necessary to copy this folder `SOIL2` in the root folder of your B-u-S-Sim -*Code::Blocks* -project. After copying, the B-u-S-Sim -projectfolder contains the subfolder `SOIL2`.
Third, all files in the subfolder `SOIL2` in your *Code::Blocks* -project folder must become a member of your *Code::Blocks* -project. Click the menue item `Project` from *Code::Blocks* 's menue bar and select the option `Add files...`. Navigate into the subfolder `SOIL2` into the *Code::Blocks* 's project folder and select all files there. Proceed and click the button `Open`. This completes the integration of the source files from `SOIL2` into your B-u-S-Sim -project. You can find a subfolder `SOIL2` now in the project browser for both the source files and the header files. They contain the added files. During the next project compilation the required additional commands to show the background file will be created.
Fourth, you have to configure the B-u-S-Sim -code to integrate the additional commands from `SOIL2`. To do this, it is necessary to update one instruction line in the file `bussim_extern.h`. Open this file in *Code::Blocks* and search the line with the source code `// #define BUSSIM_EXTERN_SOIL`. Remove this comment signs `//` and save the modified file `bussim_extern.h`. Now, the compiler-statement `#define BUSSIM_EXTERN_SOIL` is active and ensures that the `SOIL2`-commands are integrated into the `bus_sim.exe` during the next project compilation.
Finally, start the compilation of your B-u-S-Sim -project using the target `release`. Open the command line and change into the project's subfolder `bin/release`. Start `bus_sim.exe`. As soon as the windows have been opened, select the simulation window. An example background map fades in if you press the combination `<SHIFT>+<G>`.

10.2 Creating the Background Map File

You need two ingredients for the creation of a background map file:

- you need the raw data of an electronic map as the base for the creation of a drawn map and
- a tool to draw the map and to create a perfectly fitting map segment as background file.

We use raw data from openstreetmap found at `https://www.openstreetmap.org/`. Regional extracts of these data are regularly processed and prepared for free download at the website at `https://download.geofabrik.de/`. For our example, we need a map segment from the German state of

Saxonia. So, we navigate to `Europe`, to `Germany` and download the corresponding regional map data in `osm.pbf`-format by selecting the file `sachsen-latest.osm.pbf`.
Maperitive is a useful tool to render (draw) electronic map data and to create precisly cut map segments from a larger map. It is available from the website hosted at `http://maperitive.net`. Load the Maperitive program from the aforementioned website and install it on your computer. Now, start Maperitive and open the previously downloaded openstreetmap data file `sachsen-latest.osm.pbf`. Maperitive imports the map data now and renders the graphics. This might last a few minutes depending on the size of the `.osm.pbf`-file. Once the import is done, you see the map on the screen.
Next, you need to identify the exact bounds of the map segment to be extracted. These values form the *bounding box*. We define a bounding box by the specification of the minimal (`WEST`) and the maximal longitude (`EAST`) and the minimal (`SOUTH`) as well as maximal latitude (`NORTH`) in the simulated network. B-u-S-Sim tells you these values on startup and prints them to the command line. In our example, we have `WEST=8.74135`, `EAST=8.8593`, `SOUTH=53.0393`, and `NORTH=53.1222`.
To apply this bounding box in Maperitive, you have to type in the appropriate command in the command prompt box on the bottom of the Maperitive window. Click into grey box and type in the command

```
set-print-bounds-geo 8.74135,53.0393,8.8593,53.1222
```

to inform Maperitive about the limitation of the area of interest. Next, press return and Maperitive indicates the bounding box in the draw map area. Now, zoom into the bounding box by clicking the menue `View` followed by the selection of the `Zoom Bounds` option. You are ready to export the map into bitmap file. To do so, click the menue `Tools` and choose the option `Export to Bitmap`. The generation of the exported `png`-bitmap-file with the default name `output.png` may last some time.

10.3 Using the Background file in the B-u-S-Sim -Scenario

First, rename the generated `png`-file and assign a value to your choice, e.g. `background_dresden.de`. Copy this file into the folder where your `bus_sim.exe` is located. To tell your project to use the file `background_dresden.de` as background, you have to update a line in the file `bussim_global.h` in your *`Code::Blocks`* -project. Open the file `bussim_global.h` and search the line where the value of the constant `BUSSIM_BGFILE` is specified. Update this line to `#define BUSSIM_BGFILE "bremen.png"`.
Compile the project again, start the simulation, select the simulation window and press `<SHIFT>+<G>` to activate the map background. Fig. 37 shows the result.

Abbildung 37: Using the created bitmap-file as background in the simulation window

qq).

A Dateien in der Distribution 3.00

Die aktuelle B-u-S-Sim -Distribution umfasst die nachfolgend aufgelisteten Dateien. Es ist das in dieser Dokumentation beschriebene Szenario der Linie 13 hinterlegt.

- bussim_arc.cpp
- bussim_arc.h
- bussim_block.cpp
- bussim_block.h
- bussim_bunchpoint.cpp
- bussim_bunchpoint.h
- bussim_duty.cpp
- bussim_duty.h
- bussim_event.cpp
- bussim_event.h
- bussim_extern.h
- bussim_global.cpp
- bussim_global.h
- bussim_line.cpp
- bussim_line.h
- bussim_network.cpp
- bussim_network.h
- bussim_pax.cpp
- bussim_pax.h
- bussim_point.cpp
- bussim_point.h
- bussim_rotation.cpp
- bussim_rotation.h
- bussim_schedule.cpp
- bussim_schedule.h
- bussim_stop.cpp
- bussim_stop.h
- bussim_toi.cpp

- bussim_toi.h
- bussim_vehicle.cpp
- bussim_vehicle.h
- main.cpp

Möchten Sie ein leeres Projekt herstellen, so sind zunächst alle Parameter des in `main.cpp` genutzten Konstruktors für die ***BUSSIM_NETWORK***-Instanz `NET` auf 0 zu setzen. Zusätzlich sind die Instruktionen aus den folgenden Funktionen zu löschen:

- `void BUSSIM_NETWORK::specify_stops(void)`
- `void BUSSIM_NETWORK::specify_nodes_longlat(void)`
- `void BUSSIM_NETWORK::specify_arcs(void)`
- `void BUSSIM_NETWORK::specify_lines(void)`
- `void BUSSIM_NETWORK::specify_vehicles(void)`
- `void BUSSIM_NETWORK::specify_rotations(void)`
- `void BUSSIM_NETWORK::specify_vehicle_activation_times(void)`