Abteilung: Höhere Lehranstalt für Informatik

Diplomarbeit

Höhere Abteilung für Informatik

Thema:	Tours - Tourenplanungssimulator
eingereicht von:	Milena Lengauer <milena.lengauer@gmail.com> Martin Buchberger <martin99.buchberger@gmail.com>
eingereicht am:	05. April 2018
Betreuer:	Prof. DI Dr. Michael Buchberger
In Zusammenarbeit mit:	FH Steyr FH-Prof. Dr. Efrem Lengauer

Eidesstattliche Erklärung

Die unterfertigten Kandidaten / Kandidatinnen haben gemäß § 34 (3) SchUG in Verbindung mit § 22 (1) Zi. 3 lit. b der Verordnung über die abschließenden Prüfungen in den berufsbildenden mittleren und höheren Schulen, BGBl. II Nr. 70 vom 24.02.2000 (Prüfungsordnung BMHS), die Ausarbeitung einer Diplomarbeit mit der umseitig angeführten Aufgabenstellung gewählt.

Die Kandidaten / Kandidatinnen nehmen zur Kenntnis, dass die Diplomarbeit in eigenständiger Weise und außerhalb des Unterrichtes zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichtes mit einbezogen werden können.

Die Abgabe der Diplomarbeit hat bis spätestens 05.04.2018 beim zuständigen Betreuer / der zuständigen Betreuerin zu erfolgen.

Die Kandidaten / Kandidatinnen nehmen weiters zur Kenntnis, dass gemäß § 9 (6) der Prüfungsordnung BMHS nur der Schulleiter bis spätestens Ende des vorletzten Semesters den Abbruch einer Diplomarbeit anordnen kann, wenn diese aus nicht beim Prüfungskandidaten (bei den Prüfungskandidaten) gelegenen Gründen nicht fertiggestellt werden kann.

Kandidaten / Kandidatinnen inkl. Unterschrift:

Ort, Datum	Milena Lengauer
Ort, Datum	Martin Buchberger

Danksagung

Wir bedanken uns bei unserem Betreuer Prof. DI Dr. Michael Buchberger sowohl für die technische Unterstützung als auch für die Hilfestellungen und Tipps beim Verfassen der Arbeit.
Weiters möchten wir uns gerne bei unserem Auftraggeber, FH-Prof. Dr. Efrem Lengauer, für die gute Zusammenarbeit und raschen Rückmeldungen bei Fragen herzlich bedanken.
Auch den berufsbegleitenden Studenten des zweiten Semesters des Studiengangs Internationales Logistik-Management möchten für das Testen unseres Programms und die hilfreichen Rückmeldungen und Verbesserungsvorschläge unseren Dank aussprechen.
Abschließend wollen wir uns noch ganz herzlich bei unseren Eltern für das Korrekturlesen der Arbeit bedanken.

Kurzbeschreibung

Aufgabenstellung
An der FH Steyr wurde im Studiengang Internationales Logistik-Management ein Programm eingesetzt, mit dem die Studenten versuchen sollten, ein Tourenplanungsproblem möglichst effizient zu lösen. Da dieses Programm jedoch schon sehr veraltet war, wurde als Aufgabenstellung für diese Diplomarbeit definiert, eine neue Software für diesen Zweck zu entwickeln. Dazu sollten zwei Programme implementiert werden, ein Szenariengenerator zur Erstellung und Verwaltung von Tourenplanungsproblemen sowie eine Spiel-Applikation, mit der die Studenten die erstellten Probleme selbst lösen können.
Ziel war es, eine Möglichkeit zu bieten, die in der Lehrveranstaltung theoretisch gelernten Inhalte praktisch zu üben und verschiedene Algorithmen anzuwenden. Des Weiteren war gefordert, dass der Lehrveranstaltungsleiter durch den Szenariengenerator mit nur wenig Aufwand eine große Anzahl verschiedener Übungsaufgaben für die Studenten generieren kann.

Realisierung
Um die Anforderungen erfüllen und eine moderne Lösung bereitstellen zu können, wurden beide Programmteile mit Technologien von Microsoft verwirklicht. Die Oberfläche wurde mit WPF gestaltet, als Datenbank wird ein Microsoft SQL Server verwendet und die Kommunikation zwischen den Programmen und der Datenbank findet über einen WCF-Dienst statt.

Ergebnis
Das Ergebnis bilden zwei Programmteile, ein Szenariengenerator und eine Spiel-Applikation.
Mit dem Szenariengenerator können durch Eingabe verschiedener Parameter kapazitierte Tourenplanungsprobleme erstellt werden. Lösungen für die generierten Probleme werden anschließend anhand verschiedener Algorithmen, welche in Form einer Algorithmen-Bibliothek bereitgestellt werden, berechnet. Außerdem kann der Vortragende mit nur einmaliger Parametereingabe eine Reihe von zufälligen Planungsproblemen erzeugen und dazu Angaben für Studenten in Excel-Dateien ausgeben. Die generierten Planungsprobleme dienen jedoch nicht nur als Übungsaufgaben, sondern können auch in einer Spiel-Applikation von den Studenten selbst gelöst werden. Die Szenarien können dafür sowohl über Dateien als auch über die Datenbank verteilt werden. Dabei ist es ebenfalls möglich, als Spielleiter mit dem Szenariengenerator ein Gruppen-Spiel zu starten. In diesem Fall wird das Szenario in die Datenbank geladen. Von dort können die Studenten unter Angabe eines Spielcodes und des eigenen Namens die Angabe herunterladen und ihre Lösung anschließend abspeichern. Dadurch können sie gegeneinander antreten und am Ende kann der Spielleiter mit dem Szenariengenerator den Studenten mit dem kostengünstigsten Ergebnis ermitteln. Zusätzlich ist es möglich, die Lösungen der Spielteilnehmer mit den automatisch generierten Algorithmen-Lösungen zu vergleichen.

Abstract

Problem definition
Within the International Logistics Management curriculum at the University of Applied Sciences in Steyr, a software for solving capacitated vehicle routing problems as efficiently as possible was used. Since this application is outdated, the goal of this thesis was to create a new software-product consisting of two parts for this purpose. A scenario generator for creating and managing vehicle routing problems, as well as a game-application that allows students to solve the problems themselves.
The aim was to provide the opportunity to practice the theoretical content of the course and to apply different algorithms. Furthermore, the lecturer should be able to create a large number of different scenarios for students with only little effort.

Implementation
In order to be able to meet the requirements and provide a modern solution, both parts of the software were implemented using technologies of Microsoft. The interface was designed with WPF, a Microsoft SQL Server is used as the database. Communication between the applications takes place via a WCF-Service.

Results
The result consists of two parts, a scenario generator and a game-application.
The scenario generator can be used to create capacitated vehicle routing problems by entering various parameters. Solutions for the generated problems can then be calculated using a set of algorithms provided by an algorithm library.
In addition, the lecturer can easily create a large number of vehicle routing problems automatically by entering only a few parameters. After that, the program can export specifications of the scenarios into Excel files. These files can then be handed to the students as an exam or as homework.
Using the game-application, students can find their own solutions to the generated vehicle routing problems by trying to be as cost-effective as possible. The scenarios can be distributed between the scenario generator and the game via files or database.
Furthermore, it is possible for the lecturer to start a group game using the scenario generator. In this case, the scenario is loaded into the database, and from there, students can download the task by entering a game code and their name. After the students have found a good solution to the scenario, they can hand in their results to the database. This allows them to compete against each other and in the end, one student with the most cost-effective solution can be determined by the game administrator using the scenario generator. In addition, it is also possible to compare solutions of students with automatically generated algorithm-solutions.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Ausgangslage

An der Fachhochschule Steyr wird im Studiengang Internationales Logistik-Management (ILM) im Rahmen der Lehrveranstaltung „Distributionslogistik“ eine Spiel-Applikation eingesetzt, mit der die Studenten versuchen, ein kapazitiertes Tourenplanungsproblem selbst zu lösen. Da dieses Programm jedoch nicht mehr den technischen und didaktischen Anforderungen genügt, ist es Aufgabe dieser Diplomarbeit, eine neue Software mit zusätzlichen Funktionen zu implementieren.

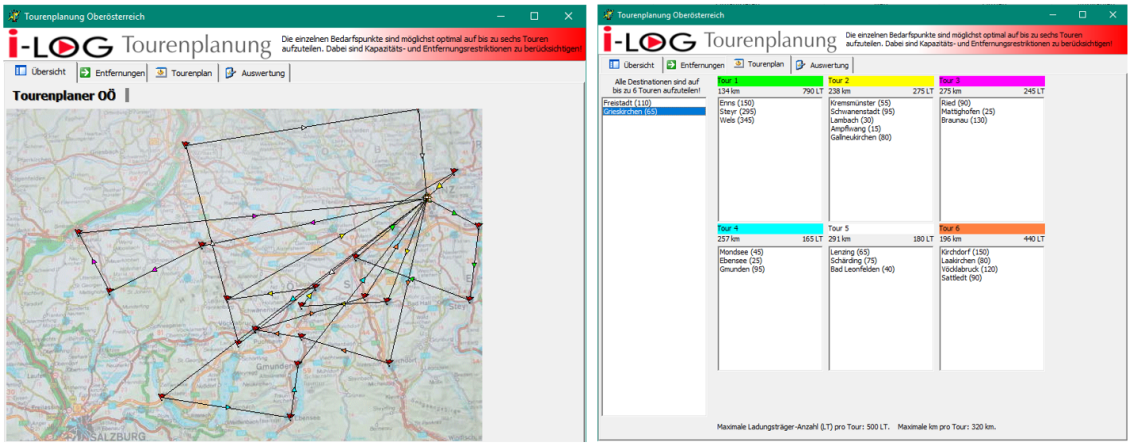

Abbildung 1.1: Bisher eingesetzte Software

Abbildung 1.1 zeigt die Benutzeroberfläche der bisher eingesetzten Spiel-Applikation. Ziel des Spiels ist es, das dargestellte Tourenplanungsproblem möglichst kostengünstig unter Beachtung sämtlicher Kapazitätsrestriktionen zu lösen. Dazu müssen die Studenten die zu verplanenden Orte LKWs zuweisen. Dabei wird eine Höchstanzahl an möglichen Touren vorgegeben und es ist nicht möglich, zusätzliche Touren zu starten. Der Verlauf der Routen wird, wie ebenfalls in Abbildung 1.1 gezeigt, in einer zweiten Ansicht in einem statischen Bild eingezeichnet. Dieses Bild ist sehr unübersichtlich, da die Namen der Orte nicht angezeigt werden und Zoomen und Verschieben der Ansicht nicht möglich sind.
Ein weiterer Nachteil der bisher verwendeten Software ist, dass nur ein einziges Tourenplanungsproblem gespielt werden kann und es keine Möglichkeit gibt, neue Planungsszenarien bereitzu-

stellen. Des Weiteren können erstellte Lösungen nicht abgespeichert bzw. dem Vortragenden der Lehrveranstaltung abgegeben werden. Auch besteht keine Möglichkeit, Planungsalgorithmen in der Software anzuwenden.

Derzeit müssen Planungsszenarien für Übungsaufgaben oder Tests, in denen die erlernten Algorithmen abgeprüft werden, vom Vortragenden selbst ausgearbeitet werden. Dies ist sehr zeitaufwendig, da eine Entfernungsmatrix erstellt und die Algorithmen durchgerechnet werden müssen. Mit der bisher eingesetzten Software gibt es keine Möglichkeit, diesen Prozess zu unterstützen.

1.2 Ziele

Um den Umfang der im Rahmen dieser Diplomarbeit zu entwickelnden Software festzulegen, wurden gemeinsam mit dem Auftraggeber folgende Ziele und Anforderungen an das Produkt definiert.

1.2.1 Produktziele

Ziel ist es, im Rahmen dieser Diplomarbeit ein aus zwei Teilbereichen bestehendes Programm zu entwickeln, einen Szenariengenerator und eine Spiel-Applikation. Zwischen diesen beiden Anwendungen muss eine Verbindung bestehen, über die Daten ausgetauscht werden können.

Szenariengenerator
Die Hauptanforderung an den Szenariengenerator ist die Erstellung von Tourenplanungsproblemen. Dabei soll es die Möglichkeit geben, Planungsprobleme auch automatisch unter Angabe nur weniger Rahmenparameter, um beispielsweise die Größe der Szenarien einschränken zu können, zu generieren. Auf die erstellten Tourenplanungsprobleme sollen zur Lösung die in der Lehrveranstaltung behandelten Näherungsalgorithmen zur Tourenplanung angewandt werden können. Eine weitere Anforderung ist die Möglichkeit, mit nur einmaliger Parametereingabe eine Reihe von zufälligen Planungsproblemen automatisch generieren zu lassen. Zu diesen Szenarien soll dann eine Angabe für die händische Berechnung der Touren für die Studenten in einer Datei ausgegeben werden.
Die erstellten Szenarien sollen im Anschluss in die Spiel-Applikation exportiert werden können.

Spiel-Applikation
In der Spiel-Applikation kann man die vom Spielleiter erstellten Szenarien importieren. Die im Szenario zu beliefernden Orte sollen dann von den Studenten Touren zugeordnet werden können. Somit ist es Ziel des Spiels, einen insgesamt möglichst kostengünstigen und gültigen Tourenplan zu erstellen. Des Weiteren muss es möglich sein, die geplanten Routen in einer Kartenansicht darstellen zu lassen. Nach der Verplanung aller Orte soll die Lösung dem Vortragenden abgegeben werden können. Dieser muss dann die Möglichkeit haben, die abgegebenen Lösungen im Szenariengerator anzuzeigen. Eine weitere Anforderung ist, als Student an einem Gruppenspiel teilnehmen zu können. Dabei versuchen alle, ein vom Spielleiter festgelegtes Szenario zu lösen und am Ende kann der Spieler mit der besten Lösung ermittelt werden.

1.2.2 Ziele des Auftraggebers

Das Ziel des Auftraggebers ist es, ein neues Programm für die Logistik-Lehrveranstaltung zu erhalten, mit dem die Studenten den Lehrstoff zum Thema Tourenplanung üben und praktisch anwenden können.

Im Unterricht werden zu diesem Thema einige einfache Näherungsalgorithmen behandelt. Daher liegt besonderes Augenmerk auf der einfachen und raschen Erstellung verschiedener Szenarien, wodurch die Studenten die theoretisch erlernten Algorithmen selbst in Übungsbeispielen anwenden können. Da die Erstellung mehrerer Szenarien mithilfe des Szenariengenerators völlig automatisch erfolgen kann, stellt dies für den Vortragenden eine enorme Zeitersparnis bei der Erstellung von Übungsangaben dar.

Wichtig ist hier auch, dass es sich bei dieser Diplomarbeit nicht um ein Produkt handelt, das in der Praxis zur Tourenplanung eingesetzt wird. Viel mehr geht es um den didaktischen Nutzen für die Studenten, denn das Spiel bietet einen einfachen Einstieg in die Materie.

1.3 Aufgabenerfüllung und Ergebnisse

Um die definierten Aufgaben zu erfüllen, wurden in allen Bereichen Technologien von Microsoft verwendet. Dies umfasst die Programmiersprache C# mit dem Grafik-Programmierkonzept WPF, WCF als Datenbank-Dienst, Visual Studio 2015 als Entwicklungsumgebung, Microsoft SQL Server als Datenbank und Bing Maps zur Anzeige des Kartenmaterials.

Das Endprodukt besteht aus folgenden Teilen:

- Szenariengenerator (WPF Applikation)
- Algorithmen-Bibliothek (C# Klassen-Bibliothek)
- Planungsspiel (WPF Applikation)
- Dienst für den Datenbankzugriff (WCF)
- Datenbank (MS SQL Server)

1.3.1 Szenariengenerator

Für die Erstellung von Szenarien wurden zwei verschiedene Möglichkeiten entwickelt: Zum einen können Szenarien völlig automatisch generiert werden, indem zuvor die Größe, also die Anzahl der Orte, und die Kapazitätsrestriktionen festgelegt werden. Vom Programm werden dann zufällig Orte und die zu liefernde Anzahl an Ladungsträgern gewählt. Zum anderen kann man Szenarien aber auch manuell erstellen. Dazu müssen die gewünschten Orte und die dazugehörigen Ladungsträger selbst ausgewählt werden.
Das Programm erstellt Angaben für die Studenten zu den generierten Szenarien, indem es die nötigen Informationen in eine Excel-Datei ausgibt.

1.3.2 Algorithmenbibliothek

Um für die generierten Szenarien automatisch Lösungen berechnen zu können, werden die Näherungsalgorithmen in Form einer Klassenbibliothek bereitgestellt. Diese umfasst jene Algorithmen, auf die in Kapitel 2.2 genauer eingegangen wird. Zusätzlich wird für eine annähernd optimale Lösung für die Szenarien auch die Tabu Search (siehe Kapitel 2.3.1) aus der Open-Source Software HeuristicLab verwendet.

1.3.3 Spiel

Für die Verplanung der Orte gibt es zwei Ansichten. Eine Planungsansicht zeigt die Touren in Form von Listen und eine Kartenansicht stellt die geplanten Touren grafisch dar. Um die Orte den einzelnen Touren zuordnen und in eine günstige Reihenfolge bringen zu können, wird für die Listen Drag-and-Drop unterstützt. Durch die laufende Berechnung der zu fahrenden Kilometer und einzuladenden Ladungsträger pro Tour erfolgt eine ständige Überprüfung, ob die Kapazitätsrestriktionen eingehalten werden und die Planung somit gültig ist. Damit das Spiel auch ohne Internetverbindung funktioniert, gibt es die Möglichkeit, anstatt der Bing Maps Karte ein statisches Bild als Kartenmaterial anzuzeigen.

1.3.4 Schnittstelle Spiel - Szenariengenerator

Die Kommunikation zwischen den beiden Programmen kann auf zwei Arten erfolgen, entweder über den Austausch von JSON-Dateien oder über die Datenbank, auf die von beiden Programmen über einen WCF-Dienst zugegriffen werden kann. Die Speicherung von Szenarien in der Datenbank ermöglicht die Realisierung eines Gruppenspiels. Dazu kann der Spielleiter ein mit dem Szenariengenerator erstelltes Planungsproblem hochladen, die Laufzeit des Spiels festlegen und einen Spielcode, über den die Studenten auf das Szenario zugreifen können, erstellen. Die Studenten können dieses Szenario in der Spiel-Applikation öffnen, es lösen und ihr Ergebnis anschließend wieder in die Datenbank hochladen. Von dort kann der Spielleiter über den Szenariengenerator die Lösungen der Studenten einsehen und den besten Spieler feststellen.

1.4 Vergleich mit der ursprünglichen Version der Software

In Tabelle 1.1 werden die wichtigsten Unterschiede zwischen dem bisher eingesetzten Programm und der in dieser Diplomarbeit entwickelten Software gezeigt.

Mängel der früheren Software	**Behebung der Probleme in der Diplomarbeit**
Im Spiel wird nur ein einziges Planungsszenario abgebildet. Es gibt keine Möglichkeit, neue Szenarien zu integrieren.	Es wurden zwei eigene Programmteile implementiert, ein Szenariengenerator und eine Spiel-Applikation. Mit dem Szenariengenerator lassen sich neue Tourenplanungsprobleme, die dann in der Spiel-Applikation gelöst werden können, erstellen.
Mit dem Spiel erstellte Lösungen können nicht abgespeichert bzw. abgegeben werden.	Durch einheitliche Schnittstellen ist es möglich, sowohl in der Spiel-Applikation, als auch im Szenariengenerator erstellte Tourenpläne bzw. die über Näherungsalgorithmen berechneten Pläne zu speichern, wieder zu laden und anzuzeigen. Weiters können Studenten erstellte Lösungen entweder über die Datenbank oder eine Lösungsdatei dem Vortragenden abgeben.

Das Programm kann keine Lösungen für ein Szenario anhand von Näherungsverfahren berechnen.	Diese Funktion wurde durch die Implementierung einer Algorithmen-Bibliothek hinzugefügt. Teil dieser Bibliothek sind folgende Näherungsverfahren: • Savings Algorithmus (mit und ohne Kapazitäten) • Nearest Neighbour Algorithmus (in zwei Varianten) • Sweep Algorithmus • Tabu Search
Planungsprobleme für Übungen müssen händisch erstellt werden.	Mit dem Szenariengenerator können durch Angabe nur weniger Parameter mehrere Planungsszenarien auf einmal erstellt werden. Weiters können für generierte Szenarien Angaben in Form von Excel-Dateien für Studenten abgespeichert werden.
Als Kartenansicht wird ein statisches Bild verwendet.	Die Kartenansicht wird nun über eine Bing Maps Karte verwirklicht. In dieser sind Zoomen und Verschieben der Anischt möglich, wodurch eine bessere Orientierung für Spieler erreicht wird.
Es sind nur Einzelspiele möglich.	Durch die Verbindung der beiden Programmteile über eine Datenbank kann mit dem Szenariengenerator ein Gruppenspiel gestartet werden, bei dem Studenten gegeneinander antreten können und am Ende der beste Spieler ermittelt wird.

Tabelle 1.1: Vergleich der Diplomarbeit mit der früheren Software

Kapitel 2

Theoretische Grundlagen

2.1 Grundlagen zur Tourenplanung

Da im Rahmen dieser Diplomarbeit ein Szenariengenerator und eine Spiel-Applikation entwickelt wurden, die im Bereich des speditionellen Sammelgutverkehrs angesiedelt sind, werden in diesem Kapitel zunächst wichtige theoretische Grundlagen erklärt. Es wird zuerest auf den Sammelgutverkehr und anschließend auf die für diese Arbeit relevanten Arten der Tourenplanung sowie auf deren Lösungsmethoden eingegangen.

2.1.1 Sammelgutverkehr

Der Sammelgutverkehr wird als gebrochener Verkehr bezeichnet, was bedeutet, dass Ware nicht ohne Stopp von A nach B transportiert, sondern üblicherweise zweimal an Umschlagspunkten „gebrochen“ wird. Der Ablauf dazu wird in der Abbildung 2.1 dargestellt und setzt sich aus folgenden Schritten zusammen:

1. Einsammeln der Ware (Vorlauf)
2. Umschlagen der Ware an der Quellniederlassung
3. gebündelter Transport mehrerer Sendungen zu einer Zielniederlassung (Hauptlauf)
4. Ausliefern der Ware an die Empfänger (Nachlauf)

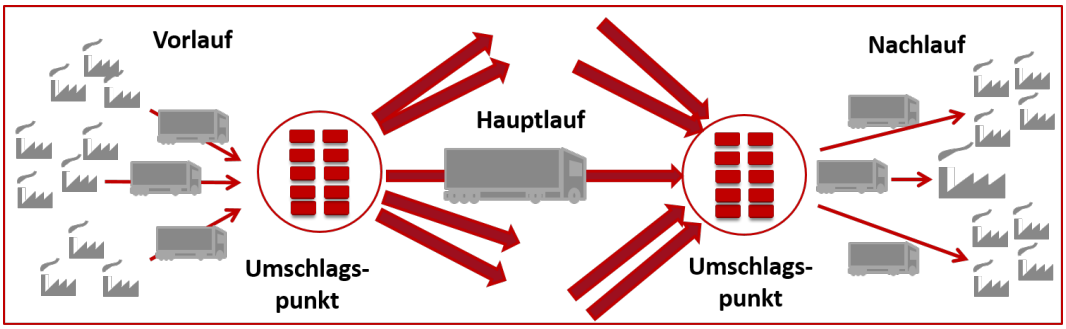

Abbildung 2.1: Sammelgutverkehr; angelehnt an [Bre15]

Der Vorlauf und der Nachlauf werden meist unter dem Begriff Nebenlauf zusammengefasst. Genau in diesem Nebenlauf braucht man in der Spedition die Tourenplanung [Bre15].

Für die weiteren Ausführungen ist es nicht relevant, ob es sich um ein Einsammelproblem (Vorlauf) oder ein Auslieferproblem (Nachlauf) handelt. Kombinierte Probleme (Pick-up-and-Delivery) werden in dieser Diplomarbeit nicht behandelt.

2.1.2 Klassifikation von Tourenplanungsproblemen

Im Standardproblem der Tourenplanung wird von einer bestimmten Anzahl an Orten eine bestimmte Menge an Fracht eingesammelt (bzw. ausgeliefert) und zu einem Sammelpunkt (Depot) gebracht. Die Lage der Orte, die Entfernungen zwischen den Orten sowie die maximale Ladekapazität der Fahrzeuge sind bekannt. Distanzen sind üblicherweise symmetrisch, das heißt die Entfernung von A nach B ist identisch mit der Entfernung von B nach A. Die entsprechenden Distanzen werden dazu in einer Distanzmatrix dargestellt.
Ziel der Tourenplanung ist es, die Kosten für die gefahrenen Kilometer zu minimieren. Dazu sind üblicherweise Restriktionen betreffend Ladekapazitäten und in Einzelfällen auch betreffend maximaler Entfernungen (gesetzlich limitierte Lenkzeiten) zu berücksichtigen. Dieses Standardproblem der Tourenplanung kann in verschiedene Arten gegliedert werden (vgl. [Vah05], [Ohr08]).

Ein-Depot- oder Mehr-Depot-Probleme
Bei einem Ein-Depot-Problem startet und endet jede Tour am selben Depot, während sich bei einem Mehr-Depot-Problem Quell- und Ziel-Depot an verschiedenen Orten befinden.

Knoten- oder Kantenorientiert
Bei knotenorientierten Tourenplanungsproblemen geht es darum, bestimmte Orte (= Knoten) zu bedienen. Bei kantenorientierten Problemen hingegen müssen bestimmte Strecken (= Kanten) befahren werden (z.B. Schneeräumung).

Mit oder ohne Kapazitätsrestriktionen
Bei Tourenplanungsproblemen ohne Kapazitätsrestriktionen ergibt sich lediglich eine einzige Tour. Das Ergebnis ist daher vergleichbar mit einem Travelling Salesman Problem (TSP). Bei diesem Problem geht es darum, Kunden in der Reihenfolge zu besuchen, sodass die Länge der Gesamtstrecke minimal ist [Vah05]. Aufgabe ist es dabei also, das Reihenfolgeproblem zu lösen, das im Kapitel 2.1.3 näher erklärt wird.
Handelt es sich um ein Planungsproblem mit Kapazitäten, können aufgrund der Restriktionen üblicherweise nicht alle Knoten in einer Tour bedient werden.

Statisch oder Dynamisch
Bei statischen Tourenplanungsproblemen sind sämtliche Informationen (Orte, Anzahl der Ladungsträger, verwendete Fahrzeuge, etc.) von vornherein bekannt. Bei dynamischen Tourenplanungsproblemen wird eine Tour gestartet und während der Abarbeitung können der Tour noch neue Orte hinzugefügt oder die Anzahl der Ladungsträger für einen Kunden geändert werden.

Mit und ohne Zeitfenster
Mit dem Begriff „Zeitfenster“ ist gemeint, dass ein bestimmter Ort nur in bestimmten Zeitintervallen bedient werden kann (z.B. bei Auslieferungen in Fußgängerzonen).

Entsprechend der oben dargestellten Klassifizierungen wird in dieser Diplomarbeit ein statisches Ein-Depot-Problem sowohl mit als auch ohne Kapazitätsbeschränkungen, ohne Zeitfenster behandelt.

2.1.3 Lösungsansätze

Das Tourenplanungsproblem besteht aus zwei Teilproblemen, dem Zuordnungsproblem und dem Reihenfolgeproblem.

Zuordnungsproblem
Bei einem Zuordnungsproblem, auch Clustering genannt, geht es um die Festlegung der Bestandteile einer Tour. Einer Tour können solange Orte hinzugefügt werden, bis die Kapazitätsbeschränkungen erreicht sind. Die Reihenfolge spielt hier noch keine Rolle. Daraus ergibt sich, dass bei der Tourenplanung ohne Kapazitäten kein Zuordnungsproblem besteht, da sämtliche Aufträge Bestandteil der selben Tour sind. In diesem Fall ist lediglich das Reihenfolgeproblem zu lösen.

Reihenfolgeproblem
Im Reihenfolgeproblem, auch Routing genannt, wird festgelegt, wie die Orte einer Tour nacheinander angefahren werden. Jede Route startet und endet dabei beim Depot.

Eine mögliche Einteilung von Lösungsverfahren wäre dahingehend, in welcher Form Zuordnungs- und Reihenfolgeproblem gelöst werden. Dabei kommen sogenannte sequentielle Verfahren (Cluster-First-Route-Second bzw. Route-First-Cluster-Second) und simultane Verfahren, die das Zuordnungs- und Reihenfolgeproblem gleichzeitig lösen, zur Anwendung [Dom97].

Bei der Lösung eines Tourenplanungsproblems ist es noch wichtig, folgende Begriffe zu unterscheiden [ZB07]:

- Tour: Eine Tour ist die Menge der Knoten, die in einer einzigen Rundreise bedient werden (unabhängig von der Reihenfolge).
- Route: Eine Route basiert auf einer definierten Tour und legt fest, in welcher Reihenfolge die zugewiesenen Knoten bedient werden, um beispielsweise die Entfernungen zu minimieren.
- Tourenplan: Ein Tourenplan ist eine Menge aus Routen.
- Depot: Das Depot ist der Ort, an dem die Sammel- bzw. Auslieferfahrten starten und enden.

Bei Lösungsverfahren für die Tourenplanung wird zwischen exakten und näherungsweisen Verfahren unterschieden. Exakte Verfahren garantieren, die optimale Lösung für das Problem zu finden. Der große Nachteil dabei ist jedoch, dass mit der Komplexität des Planungsproblems die Berechnungszeit stark zunimmt und zu dem Problem ab einer gewissen Größenordnung (mit den derzeitigen Rechenressourcen) in endlicher Zeit keine Lösung mehr gefunden werden kann. Eine Möglichkeit zur exakten Lösung eines Tourenplanungsproblems ist die vollständige Enumeration. Dabei wird jede mögliche Lösung berechnet und überprüft. Die Anzahl der möglichen Routen kann dabei, wie in Formel 2.1 gezeigt, ermittelt werden [ZB07].

$$\sum_{i=1}^{R} \frac{N!}{(N-i)!} \tag{2.1}$$

N ist in Formel 2.1 die Anzahl der zu beliefernden Kunden und R die maximale Anzahl von Kunden pro Route. Angenommen es müssen pro Tag 100 Kunden beliefert werden und pro

Route können maximal 25 Kunden besucht werden, ergeben sich rund $4*10^{48}$ mögliche Routen. Aufgrund der Komplexität von Tourenplanungsproblemen werden in der Praxis üblicherweise Näherungsverfahren zur Lösung verwendet. Diese können eine möglichst gute Lösung in angemessener Zeit finden. Dazu gibt es einige einfache Algorithmen (Nearest Neighbour Algorithmus, Savings Algorithmus, Sweep Algorithmus), die im Abschnitt 2.2 dargestellt werden, und mit dem Szenariengenerator (siehe Kapitel 4.1) verwendet werden können. Der Vorteil dieser einfachen Verfahren ist, dass man sie auch selbst berechnen und nachvollziehen kann. Dies hat zwar wenig praktischen Wert, jedoch sehr wohl didaktischen in Lehrveranstaltungen.
Tourenplanungsprobleme in einer praktischen Größenordnung werden meist mit metaheuristischen Verfahren, die in Abschnitt 2.3.1 noch genauer erklärt werden, gelöst. Diese garantieren zwar ebenfalls keine optimale Lösung, finden jedoch eine möglichst gute Lösung in kurzer Zeit. Beispiele hierfür sind verschiedenste Arten von Genetischen Algorithmen oder Nachbarschaftsverfahren, wie beispielsweise Tabu Search und Simulated Annealing.
Von den genannten Verfahren wird in dieser Arbeit der Tabu Search Algorithmus näher beschrieben.

2.2 Implementierte Lösungsverfahren

Die in diesem Kapitel beschriebenen Lösungsverfahren wurden im Rahmen dieser Diplomarbeit in der Algorithmen-Bibliokthek (siehe Kapitel 5.4) des Szenariengenerators implementiert (vgl. [Vah05]).

2.2.1 Nearest Neighbour Algorithmus

Der Nearest Neighbour Algorithmus ist dazu geeignet, das Travelling Salesman Problem zu lösen. Dieses ist dadurch charakterisiert, dass es sich um ein reines Reihenfolgeproblem handelt, also keine Kapazitätsrestriktionen aufweist (siehe Kapitel 2.1.3). Beginnend beim Depot (Startort) wird jener Knoten gewählt, der die geringste Distanz zum Depot aufweist (nächster Nachbar). Weisen zwei Knoten dieselbe Distanz auf, wird ein beliebiger gewählt. Von diesem Knoten aus wählt man wiederum den nächsten Nachbarn. Dies wird solange durchgeführt, bis sämtliche zu bedienenden Knoten Bestandteil der Route sind. Nach Verplanung aller Knoten wird zurück zum Depot gefahren.
Um dieses Verfahren auch für die Tourenplanung zu verwenden, wurden für diese Arbeit zwei Varianten des Verfahrens definiert, bei denen Kapazitäten berücksichtigt werden.

Variante 1
Vor dem Hinzufügen eines Knotens wird überprüft, ob dadurch die Kapazitätsrestriktionen verletzt werden. Ist dies der Fall, wird dieser Knoten nicht mehr hinzugefügt und zurück zum Depot gefahren. Die Route ist damit fertig definiert und es wird eine neue Tour vom Depot aus begonnen, solange bis alle Knoten Bestandteil einer Tour sind.

Variante 2
Auch hier überprüft man vor dem Hinzufügen des nächsten Knotens die Kapazitätsrestriktionen. Werden diese mit dem Anhängen des Knotens überschritten, so wird der nächstbeste Knoten herangezogen und die Kapazitäten überprüft, solange bis ein Konten gefunden wurde, der noch zur Tour hinzugefügt werden kann. Findet sich kein Knoten mehr, fährt man zum Depot zurück und startet eine neue Tour.

Sowohl Variante 1 als auch Variante 2 sind simultane Verfahren, bei denen Zuordnungs- und Reihenfolgeproblem „gleichzeitig“ gelöst werden (siehe Kapitel 2.1.3).

2.2.2 Sweep Algorithmus

Dieser Algorithmus gehört zu der Kategorie der Cluster-First-Route-Second Verfahren. In einem ersten Schritt werden hier die zu verplanenden Orte in einem Koordinatensystem mit dem Ursprung im Depot erfasst und entsprechend ihrer daraus resultierenden Polarwinkel in einer Liste aufsteigend sortiert.
Im zweiten Schritt erfolgt das Clustering. Beginnend bei jenem Ort mit dem niedrigsten Polarwinkel werden der sortierten Liste folgend solange Orte zu einer Tour hinzugefügt, bis die Kapazitätsbeschränkungen nicht mehr erfüllt sind und eine neue Tour begonnen werden muss. Sind sämtliche Orte einer Tour zugeordnet, wird innerhalb der Touren das Reihenfolgeproblem mit dem Nearest Neighbour Algorithmus gelöst. Damit erhält man einen Tourenplan mit einer bestimmten Gesamtdistanz.
Der große Vorteil des Sweep Verfahrens ist, dass es hier noch nicht zu Ende ist, sondern eine Vielzahl verschiedener Tourenpläne generiert wird. Denn nun beginnt der selbe Prozess wieder von Neuem mit dem Unterschied, dass nun mit dem zweiten Knoten in der sortierten Polarwinkelliste begonnen wird. Am Ende erhält man einen zweiten Tourenplan mit einer neuen Gesamtdistanz. Dieser Prozess wird solange wiederholt bis jeder Knoten einmal am Anfang dieses Zuordnungsprozesses war. Bei 15 Orten ergeben sich dadurch 15 verschieden Tourenpläne. Von diesen wird jener ausgewählt, der die geringste Gesamtdistanz aufweist.

```
1  ListOfCustomers <- getPolarAngelsForCustomers()
2  ListOfCustomers <- sortByPolarAngels()
3  ListOfTourplans <-[]
4  currentCust <- 0
5  for i:=0 to ListOfCustomers.size do
6         Tourplan <- new Tourplan
7         currentCust <- i
8         Route <- new Route
9         for j:=0 to ListOfCustomers.size do
10                if currentCust >= ListOfCustomers.size then
11                       currentCust = 0
12
13                if stillEnoughCapacity(Route, ListOfCustomers[currentCust]) then
14                       addCustomerToRoute(Route, ListOfCustomers[currentCust]
15                else
16                       Route <- doNearestNeighbour(Route)
17                       addRouteToTourplan(Tourplan, Route)
18                       Route <- new Route
19                       addCustomerToRoute(Route, ListOfCustomers[startAt]
20                currentCust ++
21
22        addTourplanToList(ListOfTourplans, Tourplan)
23
24 BestTourplan <- getBest(ListOfTourplans)
25 return BestTourplan
```

Listing 2.1: Ablauf des Sweep Algorithmus [Vah05]

Der Ablauf des Sweep Verfahrens ist in Codebeispiel 2.1 veranschaulicht. In Zeile 1 und 2 werden Methoden aufgerufen, mit denen zu den einzelnen Orten Polarwinkel berechnet und danach sortiert werden. In die Liste *ListOfTourplans* werden anschließend die errechneten Tourenpläne gespeichert. Die Variable *currentCust* bekommt zunächst den Wert 0 zugewiesen und steht für den Index des aktuell zu bearbeitenden Orts. In jeder Iteration der äußeren *for*-Schleife wird ein neuer Tourenplan erstellt und die Variable *currentCust* auf *i* gesetzt. Dadurch wird jeder Clustering-Vorgang mit einem anderen Ort begonnen. Die innere *for*-Schleife erstellt die Routen für den jeweiligen Tourenplan. Ist die Variable *currentCust* am Ende der Liste *ListOfCustomers* angelangt wird sie wieder auf 0 gesetzt. Mit der Bedingung in Zeile 13 wird geprüft, ob der aktuelle Ort noch zur derzeit offenen Route hinzugefügt werden kann. Ist dies der Fall, wird der Ort angehängt. Kann der Ort aufgrund der Kapazitätsrestriktionen nicht mehr zur Route hinzugefügt werden, so wird diese Tour abgeschlossen. Dies schließt das Lösen des Reihefolgeproblems mit dem Neares Neighbour Algorithmus und das Hinzufügen der Route zum aktuellen Tourenplan ein. Danach wird in Zeile 18 eine neue Route eröffnet und der aktuelle Ort hinzugefügt. Am Ende der inneren Schleife wird der fertige Tourenplan in die Liste aller Tourenpläne gespeichert. Nach dem Abschluss der äußeren Schleife wird aus der Liste *ListOfTourplans* der kostengünstigste Tourenplan mit der Methode *getBest* ausgewählt.

2.2.3 Savings Algorithmus

Der Savings Algorithmus betrachtet das Zuordnungs- und Reihenfolgeproblem simultan. Im ersten Schritt dieses Verfahrens werden sogenannte Pendeltouren gebildet. Das bedeutet, dass jeder Ort in einer separaten Tour angefahren wird. Damit erhält man so viele Touren wie man Orte hat.
Es gibt zwei Gründe, warum man diesen Schritt durchführt: Erstens wird überprüft, ob dieser Tourenplan zulässig ist, denn wenn mindestens ein Ort so nicht bedient werden kann, weil eine Kapazitätsrestriktion verletzt ist, liegt ein unzulässiges Planungsproblem vor. Zweitens ist dies ein erster möglicher Tourenplan, der als Grundlage für das weitere Verbesserungsverfahren gilt. Die Grundidee hier ist es, durch das Aneinanderreihen von Knoten, Entfernungen gegenüber den Pendeltouren einzusparen. Dabei werden, wie in Abbildung 2.2 dargestellt, immer ein Anfangs- und ein Endknoten von zwei Routen miteinander verknüpft. Um diese Einsparungen (Savings) zu berechnen, vergleicht man stets zwei Knoten miteinander.

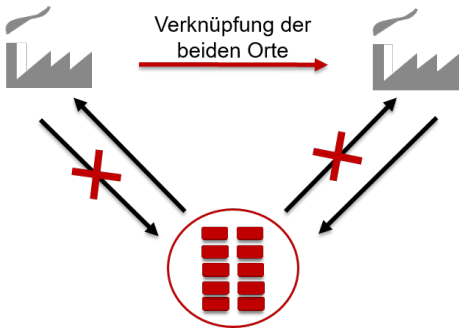

Abbildung 2.2: Einsparung durch das Verknüpfen zweier Orte

Bei zwei Pendeltouren entstehen insgesamt durch die Hin- und Rückfahrt vier Strecken. Verbindet man diese beiden Knoten zu einer Route, fällt bei einem Knoten die Rückfahrt weg und bei dem anderen Knoten die Hinfahrt. Jedoch kommt eine zusätzliche Fahrt hinzu, nämlich jene

zwischen diesen beiden Knoten. Gesamt-Savings-Werte ermitteln sich daher durch die Summe der beiden weggefallenen Strecken abzüglich der zusätzlichen Fahrt zwischen den beiden Knoten. Daraus lässt sich sehr gut erkennen, dass ein Savings-Wert dann besonders hoch ist, wenn die beiden zu verbindenden Knoten nahe beisammen und beide weit vom Depot entfernt sind. Diese Savings-Werte müssen für alle möglichen Verknüpfungen berechnet und in einer Liste absteigend sortiert werden. Bei n zu bedienenden Kunden ergibt sich daher eine Liste der Länge n^2-n.

Nun beginnt der eigentliche Prozess der Tourenplanung. Beginnend mit der Verknüpfung mit der höchsten Ersparnis wird geprüft, ob diese Verknüpfung möglich ist. Möglich ist eine Verknüpfung dann, wenn gegen keine der folgenden Bedingungen verstoßen wird:

- Die Kapazitätrestriktionen dürfen nicht überschritten werden.
- Es können nur Start- und Endknoten verknüpft werden.

Ist eine Verknüpfung zulässig, wird sie durchgeführt und mit dem nächsten Saving-Wert fortgefahren. Der Prozess des Verknüpfens wird solange fortgesetzt bis aufgrund der beiden oben genannten Bedingungen keine Verknüpfung mehr möglich ist bzw. das Ende der Savingsliste erreicht ist.

```
1  Routes <- generate initial routes (*depot-customer-depot*)
2  SavingsList <- calculate Savings
3  SavingsList <- sort(SavingsList)
4
5  for Saving in SavingsList do
6         if Saving.Customer1 is at end of Route in Routes then
7                 Route1 <- get route containing Saving.Customer1
8             if Saving.Customer2 is at end of Route in Routes then
9                 Route2 <- get route containing Saving.Customer2
10
11                JoinedRoute <- join Route1 and Route2
12                if fulfilsCapacityRestrictions(JoinedRoute)
13                        Routes <- delete(Roue1, Routes)
14                        Routes <- delete(Route2, Routes)
15                        Routes <- add(JoinedRoute, Routes)
16
17 return Routes
```

Listing 2.2: Ablauf des Savings Algorithmus [Vah05]

Der Pseudocode 2.2 zeigt den Ablauf des Savings Verfahrens. Dabei werden zunächst die Pendeltouren gebildet und anschließend die Savings-Werte berechnet und sortiert. Mit der *for*-Schleife wird die Liste der Savings-Objekte durchlaufen. Die Bedingung in Zeile 6 prüft ob es in der Liste von Routen eine Route gibt, die als Start - oder Zielort (bzw. zweiten oder vorletzten Ort, wenn das Depot gespeichert wird) Kunde 1 des Savings-Objekts enthält. Ist dies der Fall, wird die Route im Objekt *Route1* gespeichert. Die nächste Bedingung führt die selbe Überprüfung für den zweiten Kunden des Savings-Objekts durch und speichert, wenn sie zutrifft, ebenfalls die betroffene Route zwischen. In Zeile 11 werden die beiden gefundenen Routen im Objekt *Joined-Route* zusammengefügt. Wenn die neu entstandene Route die Kapazitätsrestriktionen erfüllt, so werden *Route1* und *Route2* aus der Liste von Routen entfernt und die neue zusammengefügte Route hinzugefügt, ansonsten wird sofort mit dem nächsten Savings-Objekt fortgefahren.

2.3 Verfahren im HeuristicLab

Die im Kapitel 2.2 beschriebenen Verfahren eignen sich sehr gut dazu, algorithmisches Denken im Rahmen einer Logistik-Lehrveranstaltungen zu trainieren. Lösungen, die nahe am Optimum liegen, werden jedoch nur in Ausnahmefällen erreicht. Um Benchmark-Lösungen für generierte Szenarien zu erhalten, sind daher weiter entwickelte Nährerungsverfahren notwendig. Die Open Source Software HeuristicLab bietet die Möglichkeit, Tourenplanungsprobleme mit modernen Metaheuristiken zu lösen.

2.3.1 Metaheuristiken

Heuristische Methoden werden für das Lösen komplexer Problemstellungen angewandt. Dabei garantieren Heuristiken nicht, die optimale Lösung für dieses Optimierungsproblem zu finden, sondern sie gehen einen Kompromiss ein. Sie versuchen mit guter Laufzeit eine gute Lösung für das Problem zu finden, diese muss dabei aber nicht die beste sein. Metaheuristiken sind Heuristiken, die nicht problemspezifisch sind. Dies bedeutet, dass Metaheuristiken eine allgemeingültige Folge von Schritten vorgeben, die auf Optimierungsprobleme unterschiedlicher Art angewandt werden kann.
Die Vorgehensweise ist bei den meisten Metaheuristiken ähnlich und hat folgenden Ablauf:

1. initialization procedure: Es werden eine oder mehrere Startlösungen bereitgestellt.
2. assessment procedure: Die Qualität einer bereitgestellten möglichen Lösung wird beurteilt.
3. modification procedure: Diese mögliche Lösung wird kopiert und die Kopie wird zufällig leicht verändert.
4. selection procedure: Dabei werden die Lösungen miteinander verglichen und es wird entschieden, welche möglichen Lösungen behalten und welche verworfen werden sollen.

Im Buch „Essentials of Metaheuristics“ von Sean Luke [Luk15] erfolgt hierzu ein sehr anschaulicher Vergleich. Das vorliegende Optimierungsproblem ist eine „Black Box“. Diese „Black Box“ hat genau eine Eingabe-Öffnung und eine Ausgabe-Öffnung. In die Eingabe-Öffnung kann die Heuristik mögliche Lösungen schicken. Anschließend drückt sie den Start-Knopf. Durch die Ausgabe-Öffnung bekommt sie dann eine Beurteilung der Lösung zurück. Anschließend muss die Heuristik entscheiden, was sie mit dieser Information tut [Luk15].

Eine sehr einfache Technik, mit der die oben genannten Schritte gut veranschaulicht werden können, wäre hier der Hill-Climbing Algorithmus. Mit Pseudocodebeispiel 2.3 wird diese Heuristik beschrieben.

```
1 S <- some initial candidate solution (*initialization procedure*)
2 repeat
3   R <- Tweak(Copy(S)) (*modification procedure*)
4   if(Quality(R) > Quality(S)) then
5         S <- R (*assessment and selection procedure*)
6 until S is the ideal solution or no time left
7 return S
```

Listing 2.3: Hill-Climbing [Luk15]

In Zeile 1 des Codebeispiels 2.3 wird eine Startlösung definiert. Anschließend wird diese Lösung einer leichten, zufälligen Änderung unterzogen. Die Bedingung in Zeile 4 prüft, ob die Qualität der neuen Lösung besser ist. Ist dies der Fall, wird in der Variable S die neue Lösung gespeichert. Die Schritte von Zeile 3 bis 5 werden solange wiederholt, bis die ideale Lösung gefunden wurde oder die Zeit abgelaufen ist.

Metaheuristiken können in individuenbasierte Metaheuristiken und populationsbasierte Metaheuristiken gegliedert werden.
Individuenbasierte Heuristiken arbeiten so, dass Lösungen immer nur einzeln, nacheinander erzeugt werden. Ein Beispiel für diese Art der Heuristik ist der Tabu Search Algorithmus, der im Rahmen dieser Diplomarbeit verwendet wurde.
Populationsbasierte Heuristiken arbeiten gleichzeitig mit einer Menge von Lösungen, welche Population genannt wird. Beispiele für diese Art sind Evolutionäre Algorithmen, wie Particle Swarm Optimization oder Ant Colony Optimization [ZB07]. Auf diese Verfahren wird in dieser Arbeit jedoch nicht näher eingegangen.

Tabu Search
Der Tabu Search Algorithmus gehört zur Kategorie der individuenbasierten Metaheuristiken und wird im Rahmen dieser Diplomarbeit mit HeuristicLab verwendet.
Ziel dabei ist es, anhand einer Ausgangslösung immer wieder neue Verbesserungsmöglichkeiten zu finden und schließlich eine (annähernd) optimale Lösung zu berechnen. Die verschiedenen „durchprobierten“ Lösungen werden in sogenannten „Tabulisten“ gespeichert. Wenn nun vom Algorithmus eine Lösung erzeugt wird, die bereits in der Tabuliste enthalten ist, wird sie verworfen und eine neue erzeugt.
In dieser Tabuliste (es kann auch mehrere Tabulisten geben) wird jedoch nicht jedes Mal die gesamte Lösung gespeichert, sondern lediglich die Veränderung der Lösung. Daher kommt auch der Name dieses Verfahrens, denn die Einträge in der Liste sind fortan „tabu“.
Ein Beispiel für eine solche abgespeicherte Veränderung könnte im Fall der Tourenplanung das Verlagern eines Ortes in eine andere Tour sein. Wird der Kunde K von der Tour T1 in die Tour T2 verlagert, muss verhindert werden, dass dieser Schritt vom Algorithmus wieder rückgängig gemacht werden kann. Daher wird folgender Eintrag in der Tabuliste gespeichert: (K, T1). Damit weiß das Verfahren, dass es diesen Schritt im Moment nicht rückgängig machen darf.

Tabulisten arbeiten üblicherweise nach dem FIFO-Prinzip (first in, first out). Ist die Liste voll, wird der älteste Eintrag entfernt und durch den neuen ersetzt. Somit ist zu erkennen, dass die Länge der Tabuliste wichtigen Einfluss auf das Verfahren hat. Verwendet man eine kürzere Liste, wird ein kleinerer Teil des Lösungsraums besucht, da es dadurch früher möglich ist, Änderungen, die in einem früheren Schritt durchgeführt wurden und daher in der Tabuliste eingetragen sind, wieder rückgängig zu machen. Man spricht dabei vom Intensivieren der Suche.
Wählt man eine längere Liste, so wird die Suche diversifiziert. Das bedeutet, dass größere Teile des Lösungsraums erforscht werden. Oft wird die Länge der Tabuliste sogar noch während der Suche angepasst.
Weiters können sogenannte Aspirationskriterien definiert werden, dies sind Ausnahmen, bei denen Lösungen verwendet werden können, obwohl sie bereits in der Tabuliste enthalten sind (vgl. [ZB07], [Luk15]).

Im Codebeispiel 2.4 wird der Ablauf des Tabu Search Verfahrens in Form eines Pseudocodes gezeigt.

```
1  l <- maximum tabu list length
2
3  S0 <- some initial candidate solution
4  Best <- S0
5  L <- [] (*tabu list of maximum length l*)
6  repeat
7          CandidateList <- []
8          Neighbourhood <- getNeighbours(Best)
9          for Candidate in Neighbourhood do
10                 if not containsAnyFeatures(Candidate, L) then
11                         CandidateList <- Candidate
12
13         BestCandidate <- getBestCandidate(CandidateList)
14         if Quality(BestCandidate) > Quality(Best) then
15                 L <- featureDifferences(BestCandidate, Best)
16                 Best <- BestCandidate
17                 while L.size > l do
18                         removeOldestElement(L)
19
20 until Best ist the ideal solution or no time left
21 return Best
```

Listing 2.4: Ablauf der Tabu Search [Luk15]

In Zeile 1 des Pseudocodebeispiels 2.4 wird in der Variable l die maximale Länge für die Tabuliste gespeichert. Anschließend legt das Programm eine Startlösung fest, die zunächst als die beste Lösung in der Variable *Best* gespeichert wird. Anschließend wird in der Schleife die Liste *CandidateList* angelegt. In Zeile 8 wird eine Methode aufgerufen, die die Nachbarschaft der aktuell besten Lösung absucht, also Lösungen mit kleinen Abänderungen findet. Diese Liste von Lösungen wird mit der Schleife in Zeile 9 durchlaufen, wobei überprüft wird, ob ein Merkmal der Kandiaten-Lösung mit einem Merkmal der Tabuliste übereinstimmt. Ist dies nicht der Fall, kann der Lösungskandidat zur Liste *CandidateList* hinzugefügt werden. In Zeile 13 wird aus dieser Liste von Kandidaten die beste Lösung ausgewählt. Ist deren Qualität besser als die Qualität der aktuell besten Lösung, so wird die Kandidaten-Lösung als neue beste Lösung festgelegt und die vorgenommene Veränderung in der Tabuliste gespeichert. Anschließend wird noch geprüft, ob die Tabuliste damit zu lange ist. Ist dies der Fall, muss das älteste Element entfernt werden. Dieser Vorgang wird solange durchgeführt, bis die ideale Lösung gefunden oder das Ende der Zeit erreicht wurde.

2.3.2 HeuristicLab

Der Tabu Search Algorithmus wurde im Rahmen dieser Arbeit nicht selbst implementiert, sondern über HeuristicLab in das Programm eingebunden. HeuristicLab ist eine Open Source Software für Heuristische und Evolutionäre Algorithmen, die seit 2002 von Mitgliedern des Heuristic and Evolutionary Algorithms Laboratory (HEAL) entwickelt wird [heu].
Die Software unterstützt eine Vielzahl dieser Algorithmen (z.B. Tabu Search, Particle Swarm Optimization, Genetic Algorithm, Scatter Search, Simulated Annealing, usw.) und bietet eine Reihe von Klassen zur Darstellung verschiedener Optimierungsprobleme (z.B. Travelling Salesman, Vehicle Routing, Regression, Job Shop Scheduling, Knapsack, Bin Packing, usw.). Im

Rahmen dieser Arbeit wurden dazu die Klassenbibliotheken zur Berechnung eines Tabu Search Algorithmus und zur Darstellung eines kapazitierten Tourenplanungsproblems (Capacitated Vehicle Routing Problem) aus dem HeuristicLab in das Programm eingebunden.

Die Software HeuristicLab bietet außerdem eine grafische Benutzeroberfläche, mit der die verfügbaren Optimierungsprobleme mit den in HeuristicLab implementierten Algorithmen gelöst und visualisiert werden können. Des Weiteren kann man über die grafische Benutzeroberfläche auch Anpassungen an den Algorithmen und Problemen vornehmen. Als Beispiel zeigt Abbildung 2.3 die Ausführung des Tabu Search Verfahrens zur Lösung eines Vehicle Routing Problems.

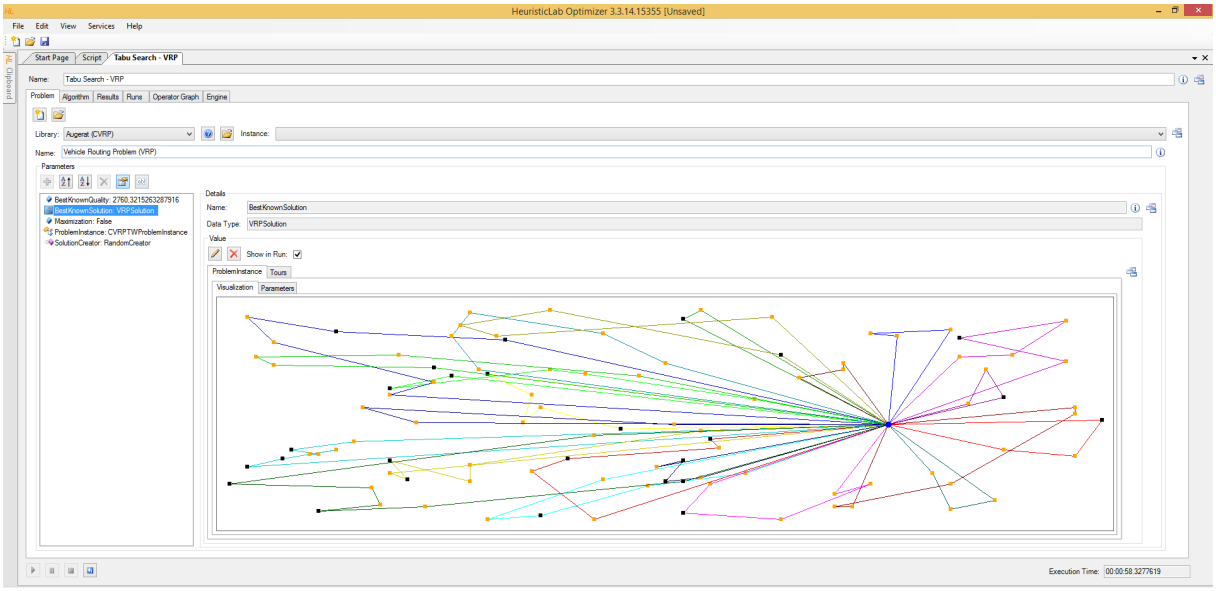

Abbildung 2.3: GUI der HeuristicLab-Anwendung

Neben der Anpassung der Algorithmen über die Benutzeroberfläche ist es auch möglich, auf Programmcode-Ebene bestimmte Funktionalitäten hinzuzufügen. Weiters kann der Plugin-Mechanismus verwendet werden, um als Entwickler eigene Algorithmen oder Probleme hinzuzufügen (vgl. [heu], [ABD+14]).

Kapitel 3

Technische Grundlagen

Im Zuge dieser Diplomarbeit wurde eine Vielzahl unterschiedlicher Technologien eingesetzt. In diesem Kapitel wird auf die Grundlagen der wichtigsten davon eingegangen.

3.1 IIS

Internet Information Services ist ein Webserver für Windows-Betriebssysteme von Microsoft. Er dient dazu, HTML-Seiten und Dateien online zugänglich zu machen und einen sicheren Zugriff auf Webdienste zu gewährleisten. Dazu verwendet er die Standardzugriffsprotokolle HTTP oder HTTPS bzw. FTP oder FTPS für Dateiübertragungen. Weiters werden auch noch einige andere Protokolle, wie beispielsweise SMTP für Mailserver, unterstützt [BDLW17].

Ein wesentlicher Bestandteil bei der Nutzung von IIS ist die Authentifizierung. Hier wird festgestellt, ob ein bestimmter Benutzer die natürliche Person ist, als die er sich ausgibt. Des Weiteren erfolgt eine Autorisierung, bei der geprüft wird, ob der Nutzer die Rechte hat, um auf eine angefragte Ressource zuzugreifen. Normalerweise geschieht eine Authentifizierung über eine Eingabe von Benutzername und Passwort, allerdings gibt es auch andere Methoden, um die Identität eines Benutzers sicherstellen zu können. IIS bietet dazu unterschiedliche integrierte Möglichkeiten, die in diesem Kapitel vorgestellt werden (vgl. [AP16c]).

Anonymous Authentication
Eine anonyme Authentifizierung bestimmt, wie IIS mit unbekannten Nutzern umgehen soll. Ist diese Authentifizierungsmethode aktiviert, wird allen Benutzern Zugang zu den öffentlichen Bereichen der Anwendung gegeben, ohne dass sie Nutzername und Passwort eingeben oder ihre Identität auf andere Art bestätigen müssen. Dazu wird der Standardbenutzer IUSR verwendet. Da die anonyme Authentifizierung jedem Benutzer Zugang zur Anwendung gibt, ist sie nicht als sicher zu erachten und muss zur Verwendung einer anderen Authentifizierungsmethode deaktiviert werden.

Basic Authentication
Bei der Verwendung von Basic Authentication müssen Nutzername und Passwort eingeben werden, um auf die geschützte Anwendung zugreifen zu können. Diese Daten werden nach der Eingabe gemäß dem in Abbildung 3.1 dargestellten Protokoll unverschlüsselt über das Netzwerk übertragen und gegen die Windows-Benutzer in der jeweiligen Domäne geprüft. Um diese Authentifizierungsmethode sicher zu machen, muss SSL aktiviert werden, damit eine verschlüsselte Übertragung der Daten erfolgt.

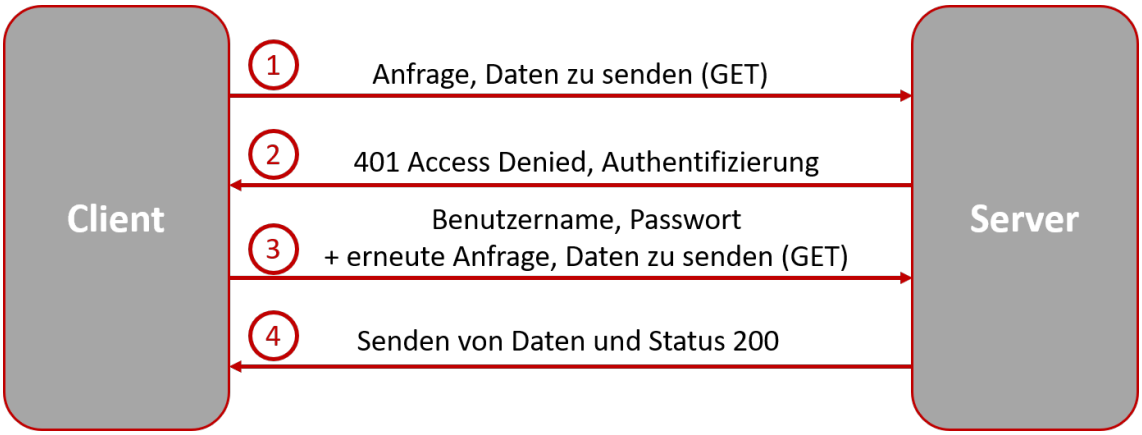

Abbildung 3.1: Ablauf der Basic Authentication; angelehnt an [bil]

Digest Authentication
Um die Sicherheitsprobleme von Basic Authentication zu umgehen, wurde Digest Authentication entwickelt. Dabei wird zuerst vom Server eine Zufallszahl an den Client gesendet. Dieser kombiniert die erhaltene Zahl dann mit dem vom Nutzer eingegeben Passwort und berechnet aus diesem Wert einen Hash, der dann gemeinsam mit dem Nutzernamen über das Netzwerk gesendet wird. Der Server führt die gleiche Berechnung mit dem zum entsprechenden Benutzer gespeicherten Passwort durch und kann somit durch einen Vergleich der beiden Ergebnisse den Nutzer authentifizieren.

Windows Authentication
Windows Authentication kann verwendet werden, wenn alle Benutzerdaten als Windows Accounts im gleichen Netzwerk wie IIS gespeichert sind und der Zugriff auf die Anwendung ebenfalls aus dem gleichen Netzwerk kommt. Die Windows-Anmeldedaten des zugreifenden Nutzers werden verschlüsselt übertragen und die Schlüsselvergabe erfolgt standardmäßig über das Protokoll Kerberos. Sind alle Bedingungen für das Einsetzen dieser Authentifizierungsmethode erfüllt, ist dies meist die Methode erster Wahl, da sich Benutzer nicht selbst authentifizieren müssen, aber dennoch eine hohe Sicherheit gewährleistet werden kann.

Neben den soeben vorgestellten Methoden zur Authentifizierung von Benutzern gibt es noch weitere Möglichkeiten:

- Client Certificate Mapping Authentication mit einem Active Directory (vgl. [AP16a])
- IIS Client Certificate Mapping Authentication (vgl. [AP16b])
- Weitere von Drittanbietern bereitgestellte Authentifizierungsmethoden

Durch Einstellen einer Impersonifizierung kann bestimmt werden, mit welchem Benutzerkonto ein authentifizierter Nutzer am Server auftreten soll. Ist beispielsweise Windows Authentifizierung eingestellt, tritt der Benutzer im Normalfall auch am Server unter seinem tatsächlichen Nutzeraccount auf. Wird jedoch auch eine Impersonifizierung eingestellt, kann festgelegt werden, dass jeder authentifizierte Benutzer am Server einen einheitlichen Benutzeraccount mit fixen Berechtigungen zugewiesen bekommt.

3.2 .NET Framework

.NET ist ein Programmierkonzept der Firma Microsoft, das die Entwicklung serviceorientierter Programme für das Betriebssystem Microsoft Windows ermöglicht. Durch den Einsatz von .NET können global verteilte Anwendungen implementiert werden, die mittels XML-Dateien kommunizieren [dot06].
Es gibt eine Vielzahl an Programmiersprachen, die mit der Laufzeitumgebung *Common Language Runtime (CLR)*, die .NET zur Verfügung stellt, kompatibel sind. Einige Beispiele dazu sind:

- C# von Microsoft
- C++ von Microsoft
- F# von Microsoft
- Visual Basic von Microsoft
- J#, eine Java-Implementierung von Microsoft
- JScript .NET, eine JavaScript-Implementierung von Microsoft

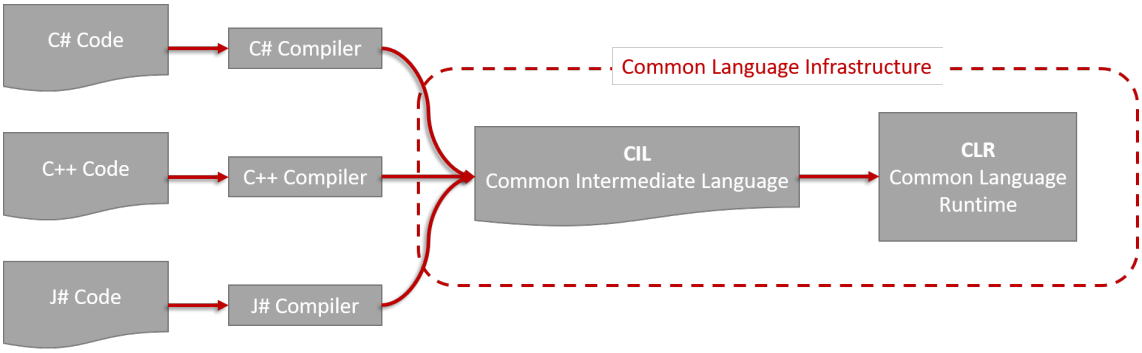

Abbildung 3.2: Common Language Infrastructure; angeleht an [Pii08]

Abbildung 3.2 zeigt, dass der Code einer .NET-kompatiblen Sprache zuerst vom dazugehörigen Compiler in die Sprache *Common Intermediate Language* (CIL) umgewandelt wird, die dann von der Laufzeitumgebung *Common Language Runtime* (CLR) gelesen und weiterverarbeitet werden kann. Bei dieser Weiterverarbeitung gibt es zwei Möglichkeiten. Entweder wird die CIL-Sprache in eine Maschinensprache übersetzt, die von der CPU direkt ausgeführt werden kann oder es findet eine Ausführung in einer virtuellen Maschine statt, was der häufigste Fall ist.
Neben der gemeinsamen Sprache spezifiziert CIL beispielsweise auch die verwendeten Datentypen und das Vorgehen beim Auftreten eines Fehlers [Rou07].

3.2.1 WCF Data Service

Data Service Dienste stellen Datensätze aus unterschiedlichen Quellen auf abstrakte Art dar. Sie regulieren und steuern dabei den Zugriff auf Datenquellen, was einerseits dem Komfort, aber auch der Sicherheit dient. Durch Einsatz von WCF ist es möglich, solche Data Service Dienste über das Internet bereitzustellen. Dazu wird das OData-Protokoll verwendet (Open Data Protocol). Dies ist eine auf REST basierende Möglichkeit, Abfragen zu senden. REST ist die

Abkürzung für Representational State Transfer und stellt eine Alternative zu Protokollen wie SOAP dar. Dabei ist REST zustandslos, was bedeutet, dass alle für die Bearbeitung nötigen Informationen in jeder Anfrage direkt mitgesendet werden. Es bietet eine einheitliche Schnittstelle zu angebotenen Ressourcen und kommuniziert dabei über die Protokolle HTTP sowie HTTPS. Dies ermöglicht unter anderem auch den Austausch von JSON- und XML-Dateien. Zusätzlich zu den Daten selbst werden durch OData auch Informationen und Metadaten zum Datenmodell zur Verfügung gestellt.

Durch die Erreichbarkeit eines WCF Dienstes über eine eigene URI-Adresse, ist es möglich, die HTTP-Befehle GET, PUT, POST und DELETE auszuführen. Da dadurch der Zugriff auf die Daten von der physischen Datenhaltung losgelöst ist, entsteht eine Speicherunabhängigkeit, dank der es möglich ist, Änderungen an der Speicherstruktur durchzuführen, ohne den Betrieb der Benutzer-Applikationen zu gefährden. [WJHL17]

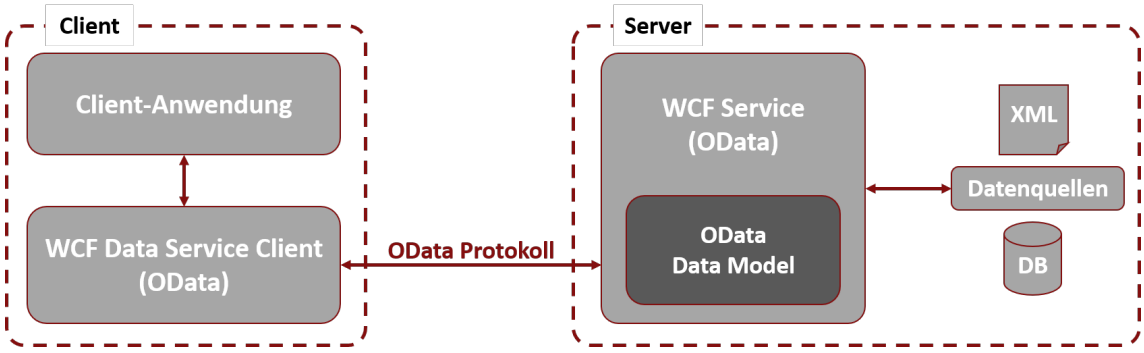

Abbildung 3.3: WCF Data Service (OData); angeleht an [Sar14]

Abbildung 3.3 gibt einen Überblick über die Architektur einer WCF Data Service Applikation. Die Client-Anwendung greift dabei auf eine WCF Data Service Client Bibliothek zu, die OData unterstützt und über dieses Protokoll mit einem bereitgestellten WCF Dienst kommuniziert. Dieser beinhaltet ein OData Datenmodell, das Daten aus beliebigen Datenquellen wie beispielsweise einer relationalen Datenbank oder einer XML-Datei abstrahiert und zugreifbar macht. Über die standardmäßigen HTTP-Befehle kann nun eine Kommunikation zwischen Client und Service stattfinden. Je nach Konfiguration des Dienstes und der Rechtevergabe ist es für die Client-Anwendung dabei möglich, Daten zu lesen, zu schreiben, zu ändern und zu löschen.

In Codebeispiel 3.1 ist ein HTTP GET Befehl, mit dem auf einen WCF-Dienst zugegriffen wird, zu sehen.

```
1 https://tourenda.htl-perg.ac.at/ToursWCF/ToursService.svc/Game?$filter=
2   key eq '1234'
```

Listing 3.1: HTTP GET

Dieser GET Befehl liefert alle Einträge aus der Tabelle Game, bei denen das Filterkriterium zutrifft. In diesem Fall sind dies alle Einträge, bei denen das Attribut *key* 1234 ist.

Neben der in Codebeispiel 3.1 in Zeile 2 gezeigten *equals*-Bedingung gibt es noch folgende andere Bedingungen, die zum Selektieren der gewünschten Werte verwendet werden können [Rui08]:

- ne: „nicht gleich“
- lt: „keiner als“
- le: „kleiner gleich“
- gt: „größer als“
- ge: „größer gleich“

3.2.2 LINQ

LINQ ist die Kurzform für Language Integrated Query und stellt eine einheitliche Methode für einen Zugriff auf Daten aus unterschiedlichen Datenquellen im .NET-Framework zur Verfügung. Dabei gibt es mehrere verschiedene Möglichkeiten, LINQ einzusetzen, die sich in der Herkunft der Daten unterscheiden (vgl. [WWB+17], [WyI+17]):

- Mit *LINQ to XML* können Daten aus XML-Dateien abgefragt werden
- *LINQ to SQL* bietet die Möglichkeit, Daten aus einer relationalen Datenbank zu selektieren
- *LINQ to Object* ermöglicht die Abfrage von Daten aus Objekten
- Mit *LINQ to Entities* kann auf Daten aus einem Entity Data Modell zugegriffen werden

Weiters gibt es auch die Möglichkeit, auf Daten aus SharePoint oder .NET-Datasets zuzugreifen.

```
1 IEnumerable<RouteLocation> iE = (
2   from routeLocation in this.Entities.RouteLocation
3   where routeLocation.routeID == route.id
4   orderby routeLocation.order ascending
5   select routeLocation
6 );
```

Listing 3.2: LINQ Abfrage (C#)

Codebeispiel 3.2 zeigt den typischen Aufbau einer LINQ-Abfrage in C#. Aufgabe der gezeigten Abfrage ist es, eine Liste vom Typ *IEnumerable* mit jenen RouteLocation-Objekten zu liefern, bei denen das Attribut *routeID* gleich der ID der an die Methode übergebenen Route ist. Dazu wird eine LINQ-Abfrage ausgeführt, deren Datenquelle eine Entität aus dem Entity Data Modell ist. Diese wird in Zeile 2 gesetzt. Mit der *where*-Klausel in Zeile 3 wird angegeben, dass nur jene Elemente selektiert werden sollen, bei denen das Attribut *routeID* gleich der ID der übergebenen Route ist. Durch die *orderby*-Klausel in Zeile 4 wird festgelegt, dass eine aufsteigende Sortierung nach dem Attribut *order* erfolgen soll. In Zeile 5 befindet sich der *select*-Teil der Abfrage, in dem angegeben wird, was als Ergebnis zurückgeliefert werden soll.

Tabelle 3.1 veranschaulicht die wichtigsten Operatoren von LINQ in C#.

Operator	**Beschreibung**
from	Hier wird eine Variable definiert, mit der auf ein Element der angegebenen Datenquelle zugegriffen werden kann.
where	Der *where*-Operator ist ein Filter-Operator, mit dem festgelegt werden kann, welche Eigenschaften die zu liefernden Datensätze haben sollen.
orderby	Hiermit kann festgelegt werden, in welcher Reihenfolge die selektierten Datensätze zurückgegeben werden sollen. Die Angabe von *ascending* oder *descending* legt fest, ob die Werte auf- oder absteigend sortiert werden sollen. Durch die Angabe bestimmter durch Komma getrennter Attribute erfolgt eine Sortierung nach mehreren Sortierkriterien.
distinct	Durch die Angabe von *distinct* können Duplikate im Ergebnis entfernt werden. Hierbei ist jedoch darauf zu achten, dass lediglich die Objektreferenzen, nicht aber der Inhalt der Objekte verglichen wird. Kommt beispielsweise ein Objekt mehrfach vor, wobei jedes Objekt eine eigene Objektreferenz hat, wird es im Ergebnis auch mehrfach vorkommen.
contains	Durch den Einsatz von *contains* kann überprüft werden, ob eine Sequenz einen bestimmten Wert enthält.
join	Beim *join*-Operator gibt es zwei Möglichkeiten, ihn einzusetzen: • *inner join*: Hierbei werden nur die Elemente der äußeren Datenquelle zurückgeliefert, zu denen in der inneren Datenquelle ein passendes Element gefunden wurde. • *group join*: Dieser Aufruf liefert alle Elemente der äußeren Datenquelle zurück. Wird in der inneren Datenquelle ein dazu passendes Element gefunden, wird dieses mit dem jeweiligen Element der äußeren Datenquelle verbunden und ebenfalls zurückgegeben.

Tabelle 3.1: Operatoren der LINQ Abfragesprache [All08]

3.3 APIs

API ist die Abkürzung für „Application Programming Interface“ und ist eine Programmierschnittstelle, über die Anwendungen auf Dienste einer Applikation oder des Betriebssystems zugreifen können. Dabei gibt es drei Grundtypen von Programmierschnittstellen, die sich durch die Art des Aufrufs sowie die zur Verfügung gestellten Funktionen unterscheiden (vgl. [RNL17]):

- Lokale Programmierschnittstellen: Diese dienen dazu, auf Funktionen des Betriebssystems oder Daten aus einer Back-End-Schicht zugreifen zu können. So fallen unter Windows die Programmierschnittstellen für .NET von Microsoft oder Schnittstellen für den Zugriff auf Datenbanken in diesen Bereich.
- Webbasierte Programmierschnittstellen: Der Zugriff auf sie erfolgt über das HTTP Protokoll. Oft werden diese Schnittstellen auch REST oder RESTful Web Applications genannt. Webbasierte Programmierschnittstellen werden in vielen Fällen verwendet, damit unterschiedliche Applikationen für verschiedene Plattformen auf den gleichen Datenbestand zugreifen können. Ein Beispiel für diese Art von Schnittellen wären die Bing Maps REST Dienste „Locations API“ und „Routes API“.
- Programm-Schnittstelle: Durch einen „remote-procedure-call“-Aufruf kann eine Programm-Schnittstelle von der aufrufenden Software so behandelt werden, als wäre sie ein in diese integrierter Bestandteil.

3.4 Bing Maps

Bing Maps ist ein Kartendienst der Firma Microsoft, über den diverse Luftaufnahmen und Satellitenbilder angezeigt werden können. Dazu umfasst er eine Vielzahl an REST-Diensten und APIs. In dieser Arbeit wird davon das API „Bing Maps WPF Control“ verwendet, was es erlaubt, in einer WPF Applikation eine Kartenansicht darzustellen. Beispielsweise können in dieser dann Orte und Routen eingezeichnet werden.
Des Weiteren kommen zwei der Bing Maps REST Dienste in dieser Arbeit zum Einsatz, der Locations API sowie der Routes API Dienst.

3.4.1 Locations API

Mit dem Locations API Dienst können zu Orten Informationen, wie zum Beispiel Längen- und Breitengrad, abgefragt werden. Dazu müssen in einem HTTP GET Befehl folgende Parameter angegeben werden (vgl. [bina]):

- *query*: Dabei handelt es sich um eine Zeichenkette, welche die bekannten Informationen zu einem Ort enthält. Beispielsweise können Postleitzahl und Ortsname angegeben werden.
- *includeNeighborhood*: Dieser Parameter ist optional und gibt an, ob in der Antwort auch Daten zur Nachbarschaft des angegebenen Ortes zurückgegeben werden sollen. Standardmäßig ist dieser Wert auf 0 gesetzt, was bedeutet, dass diese Informationen nicht Teil der Antwort sind.
- *include*: Mit diesem optionalen Parameter kann eine Rückgabe von zusätzlichen Informationen verlangt werden. Zum Beispiel gibt es hier die Option *ciso2*, mit der auch ein ISO Ländercode zurückgegeben wird.
- *maxResult*: Dabei handelt es sich um einen optionalen Parameter, der die Menge an zurückgelieferten Ergebnissen einschränkt.

In Codebeispiel 3.3 ist ein Beispielaufruf dieses REST Dienstes zu sehen.

```
1 http://dev.virtualearth.net/REST/v1/Locations?query=HTL%20Perg&
2    includeNeighborhood=1&include=ciso2&maxResultes=10&key=key
```

Listing 3.3: Locations API

Des Weiteren ist hier zu bemerken, dass in allen URL-Adressen die Leerzeichen mit *%20* ersetzt werden müssen [bina].

3.4.2 Routes API

Der zweite Bing Maps REST Dienst, der verwendet wird, ist der Routes API Dienst. Dieser ermöglicht es, für eine Reihe von Start- und Zielorten eine Distanzmatrix zu berechnen. Die URL-Adresse setzt sich dabei aus folgenden Parametern zusammen (vgl. [binb]):

- *origins*: Hier wird eine Reihe von Ausgangsorten in folgender Form angegeben:

  `Breitengrad1,Längengrad1;Breitengrad2,Längengrad2;...`

- *destinations*: Hier werden die Zielorte in der selben Form wie die Ausgangsorte angegeben.
- *travelMode*: Dieser Parameter kann entweder „driving“, „walking“ oder „transit“ sein und bestimmt, welche Verbindungen in die Berechnung mit einbezogen werden sollen und mit welchen durchschnittlichen Reisegeschwindigkeiten die Berechnung der Dauer erfolgen soll.
- *startTime*: Wenn als *travelMode* „driving“ angegben ist, kann mit *startTime* festgelegt werden, zu welcher Uhrzeit die Fahrt beginnen soll, damit relevante Verkehrsinformationen in die Berechnung miteinbezogen werden können.
- *endTime*: Soll nicht nur ein Wert sondern ein gesamtes Histogramm an Werten berechnet werden, kann hier eine Endzeit für die Distanzmatrix festgelegt werden. Der Zeitraum zwischen Start- und Endzeit darf maximal 24 Stunden betragen.
- *resolution*: Dieser Wert gibt die Anzahl der zu berechnenden Intervalle in 15-Minuten-Schritten an. Dies ist dann relevant, wenn ein Histogramm an Ergebnissen ermittelt werden soll. Der Wert kann dabei zwischen 1 und 4 betragen, wobei 1 bedeutet, dass das Intervall 15 Minuten beträgt und ein Wert von 4 bedeutet ein Intervall von einer Stunde.
- *distanceUnit*: Hier kann gewählt werden, ob die Berechnung der Distanz in Kilometern oder Meilen erfolgen soll.
- *timeUnit*: Hier kann gewählt werden, ob die Berechnung der Dauer in Minuten oder Sekunden erfolgen soll.

Der Aufruf in Codebeispiel 3.4 zeigt die Verwendung dieses Dienstes.

```
1 https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?
2    origins=lat0,long0;lat1,long1;lat2,long2&
3    destinations=lat0,long0;lat1,long1;lat2,long2&
4    travelMode=driving&startTime=2018-02-24T08:00:00-02:00&key=key
```

Listing 3.4: Routes API

In der ersten Zeile wird die URL-Adresse für den gewünschten Dienst aufgerufen. In den Zeilen 2 und 3 werden die Start- und Zielorte festgelegt, während in Zeile 4 angegeben wird, was als *travelMode* und Startzeit für das Erstellen der Distanzmatrix verwendet werden soll.
Neben der Möglichkeit eines Aufrufs durch HTTP GET kann auch ein HTTP POST Befehl abgesetzt werden. Dabei werden die nötigen Parameter in Form von JSON Objekten im POST Header mitübergeben.
Als Antwort kommt auf sämtliche Anfragen an die REST Dienste ein JSON Objekt zurück, das neben den angefragten Daten beispielsweise auch Statusinformation enthält [binb].

3.5 JSON

Das Datenformat „JavaScript Object Notation“ dient dazu, Daten in Textform zwischen Applikationen auszutauschen. Trotz der Tatsache, dass ein JSON-Dokument immer aus gültigem JavaScript-Code besteht, ist es in beinahe jeder Programmiersprache möglich, JSON zu parsen, weshalb es ein beliebtes Medium zur Serialisierung und zum Transport von Informationen zwischen Server und Client ist [JSOb].
Um dies zu ermöglichen, gibt es zwei definierte Strukturtypen, die von fast allen modernen Programmiersprachen unterstützt werden, nämlich Paare mit je einem Namen und einem Wert, die *records* genannt werden sowie geordnete Wertelisten.
Für die Umsetzung dieser Strukturen werden in JSON folgende Elemente verwendet (vgl. [JSOa]):

- Objekt: Ein Objekt besteht in JSON aus einer Menge von Paaren mit jeweils einem Namen und einem Wert, wobei es immer mit einer geschwungenen Klammer beginnt und endet. Die innerhalb des Objekt durch Beistriche getrennten Wertepaare haben folgenden Aufbau: *Name:Wert*

- Array: Im Unterschied zu Objekten beginnen und enden Arrays jeweils mit einer eckigen Klammer und die enthaltene Liste von Werten ist geordnet. Bei den in der Auflistung enthaltenen Elemente kann es sich entweder um ein Array, ein Objekt, einen String, eine Zahl oder einen einfachen Ausdruck wie *true*, *false* oder *null* handeln.

- String: Eine durch Hochkommata gekennzeichnete Zeichenkette kann beliebig viele Unicode-Zeichen enthalten. Ähnlich wie bei modernen Programmiersprachen gibt es auch Escape-Sequenzen, wie beispielsweise „\n“ für eine neue Zeile oder „\t“ für einen Tabulator.

- Zahl: Hexadezimale und oktale Zahlen können in JSON nicht verwendet werden. Bis auf diese Ausnahme ist eine Verwendung von Zahlen aber gleich der Verwendung in modernen Programmiersprachen wie C#.

Kapitel 4

Benutzung

Dieser Abschnitt dient als Benutzungsanleitung für das im Rahmen dieser Arbeit erstellte Programm. Im ersten Teil wird die Bedienung des Szenariengenerators und im zweiten Teil die der Spiel-Applikation erläutert.

4.1 Szenariengenerator

Beim Start des Szenariengenerators sieht ein Benutzer zuerst ein Hauptmenü mit vier verschiedenen Optionen. Er kann dabei, wie in Abbildung 4.1 zu sehen ist, zwischen den Aktionen „Neues Szenario generieren“, „Szenario aus Datei laden“, „Online-Szenario laden“ und „Laufende Spiele verwalten“ wählen. Diese vier Menüpunkte werden in diesem Kapitel nun genauer erklärt.

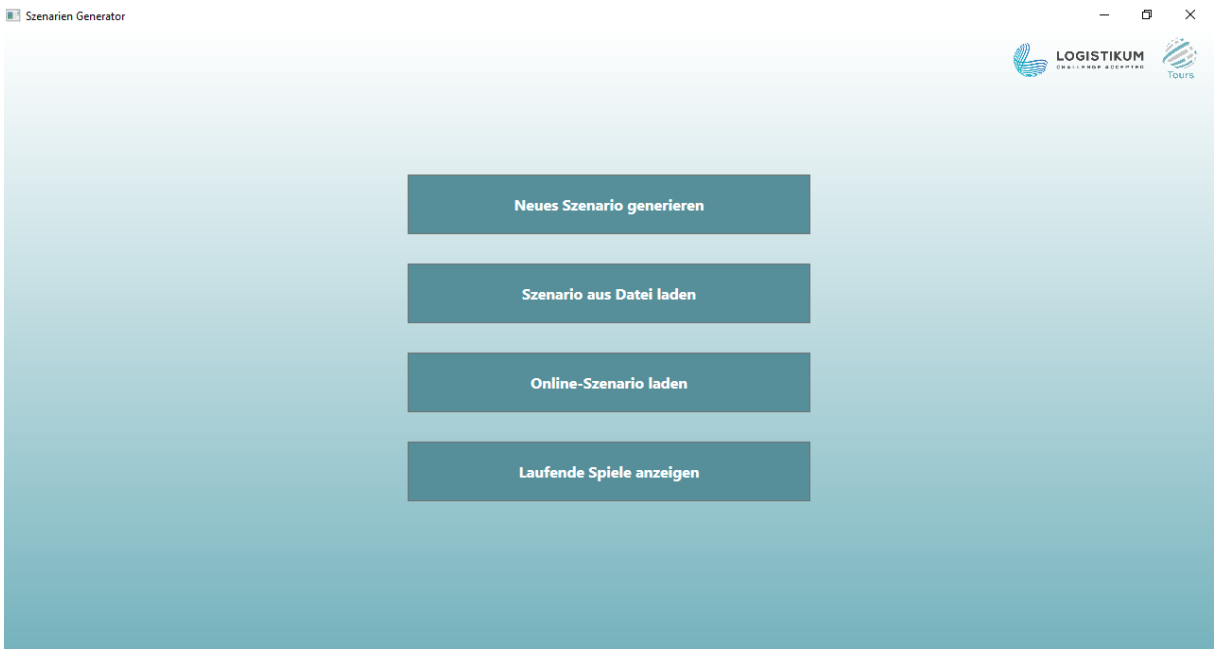

Abbildung 4.1: Hauptmenü

4.1.1 Ein neues Szenario erstellen

Wählt ein Benutzer die Option „Neues Szenario generieren“, so kann er sich in der darauffolgenden Ansicht wiederum entscheiden, in welchem Modus er das Szenario erstellen möchte. Hierzu gibt es zwei Möglichkeiten, eine automatische und eine manuelle.

Automatisch

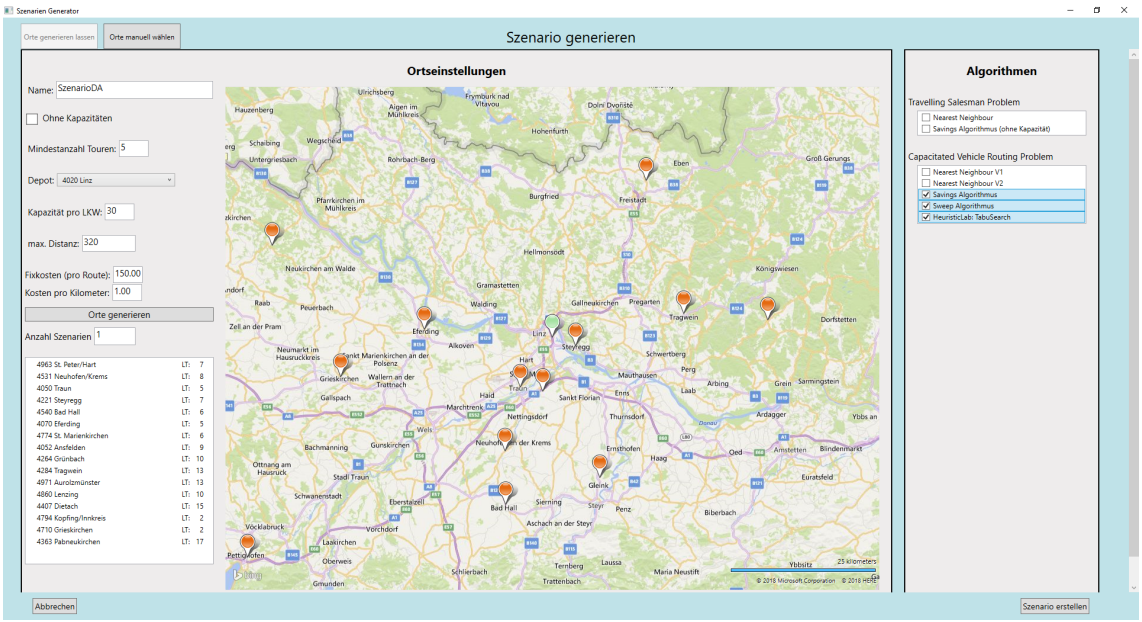

Abbildung 4.2: Szenario automatisch generieren

Hier wird das Szenario unter Eingabe einiger weniger Rahmenparameter automatisch generiert. Die Oberfläche dazu ist in Abbildung 4.2 zu sehen. Zunächst muss der Benutzer für das Szenario einen Namen vergeben und das Depot aus dem Drop-down-Menü wählen. Optional können auch fixe und variable Kosten festgelegt werden. Anschließend gibt es die Wahlmöglichkeit, ein kapazitiertes Szenario oder eines ohne Kapazitätsrestriktionen zu erstellen.
Für ein kapazitiertes Szenario muss der Benutzer die maximale Ladungsträgeranzahl und die maximale Distanz pro Tour festlegen. Um eine ungefähre Größe für das Planungsproblem anzugeben, muss die Mindestanzahl an zu entstehenden Touren eingegeben werden. Für ein Szenario ohne Kapazitäten wird die Größe über die Angabe der Ortsanzahl festgelegt. Der Grund, warum die Größeneinschränkung beim kapazitierten Szenario nicht direkt über die Anzahl der Orte erfolgen kann, ist folgender: Würden die Orte einfach zufällig gewählt werden, könnte es sein, dass in einem Szenario nur Orte vorkommen, die sehr nahe beim Depot sind, und in einem anderen Szenario Orte, die sich sehr weit voneinander und vom Depot entfernt befinden. Dies würde dazu führen, dass sich aufgrund der Entfernungsrestriktion aus dem ersten Szenario nur sehr wenige Touren, aus dem zweiten aber sehr viele ergeben. Aus diesem Grund stellt die Anzahl der Orte bei einem kapazitierten Problem keine geeignete Größeneinschränkung dar.

Anschließend kann noch angegeben werden, wie viele verschiedene Szenarien auf Basis der soeben eingegebenen Eigenschaften für das Planungsproblem erstellt werden sollen. Um das Szenario/die Szenarien vom Programm nun automatisch generieren zu lassen, wählt man „Orte generieren“.

Wenn man nur ein einzelnes Tourenplanungsproblem erstellen lässt, werden die vom Programm zufällig gewählten Orte und die dazugehörigen Ladungsträger in einer Liste und in der Kartenansicht dargestellt. Bei mehreren Szenarien erscheint ein Hinweis, dass die gewünschte Anzahl an Planungsproblemen generiert wurde.
Der entwickelte Algorithmus für die automatische Auswahl der Orte und Ladungsträger wird in Kapitel 6.5 näher erklärt.

Manuell
Im manuellen Modus können die Orte und die zu liefernden Ladungträger pro Ort vom Benutzer selbst gewählt werden. Die Benutzeroberfläche dafür ist in Abbildung 4.3 zu sehen. Des Weiteren müssen auch hier ein Name, das Depot und optional die Kosten für das Szenario angegeben werden. Außerdem gibt es hier ebenfalls die Möglichkeit, zwischen einem kapazitierten und einem Szenario ohne Kapazitäten zu wählen.

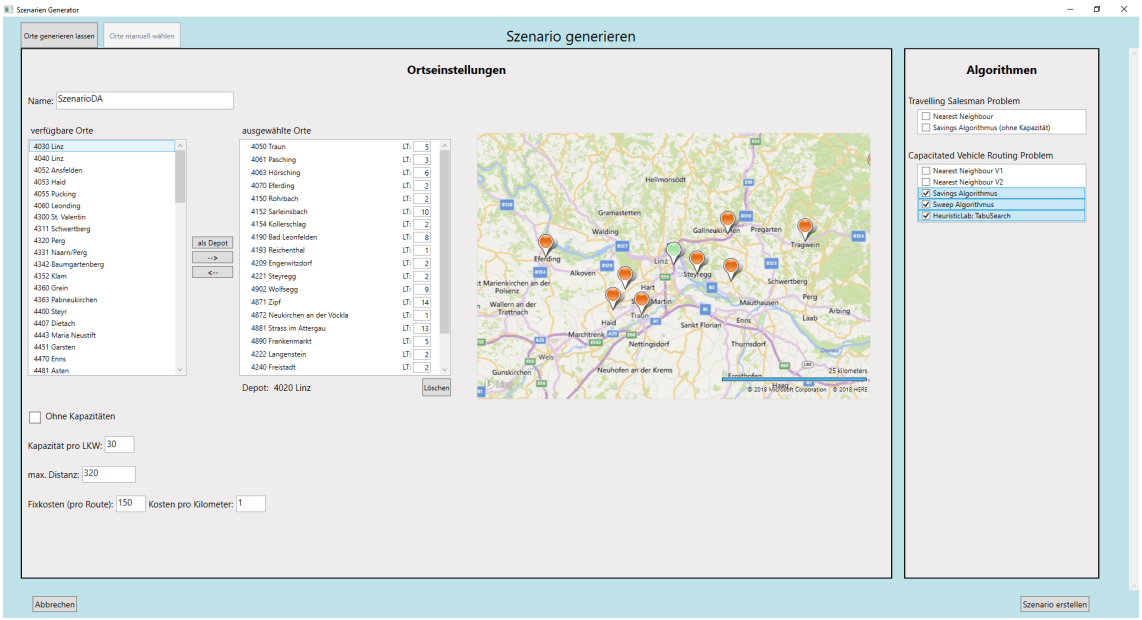

Abbildung 4.3: Szenario manuell generieren

Ist das Szenario fertig erstellt, können sowohl im manuellen als auch im automatischen Modus die Algorithmen, mit denen man das Tourenplanungsproblem lösen möchte, gewählt werden. Um die Lösungen zu berechnen und zu einer Szenario-Übersicht zu gelangen, in der weitere Optionen ausgewählt werden können, muss man die Option „Szenario erstellen“ wählen.

4.1.2 Szenario-Übersicht

Die Szenario-Übersicht gibt sämtliche Informationen zu einem erstellten Szenario wieder und ist in Abbildung 4.4 zu sehen. Es werden hier die zu beliefernden Orte und deren Ladungsträger, das Depot, die Kapazitätsrestriktionen sowie die zum Szenario erstellten Lösungen angezeigt. Bei den Lösungen kann es sich sowohl um Algorithmen-Lösungen als auch um Lösungen, die von Studenten mit der Spiel-Applikation erstellt wurden, handeln. Da es in der Spiel-Applikation möglich ist, Lösungen zwischenzuspeichern, die noch keinen gültigen Tourenplan darstellen, wird dies in der Lösungsübersicht durch graue Markierung der entsprechenden Routen angezeigt.

Zur Szenario-Übersicht gelangt man im Programm auf unterschiedlichen Wegen. Wie bereits erwähnt, zeigt sich diese Sicht nach dem Erstellen eines neuen einzelnen Szenarios. Sie wird aber auch angezeigt, wenn der Benutzer ein bereits vorhandenes Szenario aus einer Datei oder von der Datenbank öffnet.
Lädt der Benutzer das Szenario aus einer Datei, so öffnet sich zuvor der Datei-Explorer und man kann dort eine Datei des Formats „*.szenario“ auswählen.
Wird das Szenario von der Datenbank geladen, öffnet sich zunächst ein zusätzliches Fenster, in dem die gespeicherten Szenarien angezeigt werden und der Benutzer eines auswählen kann.

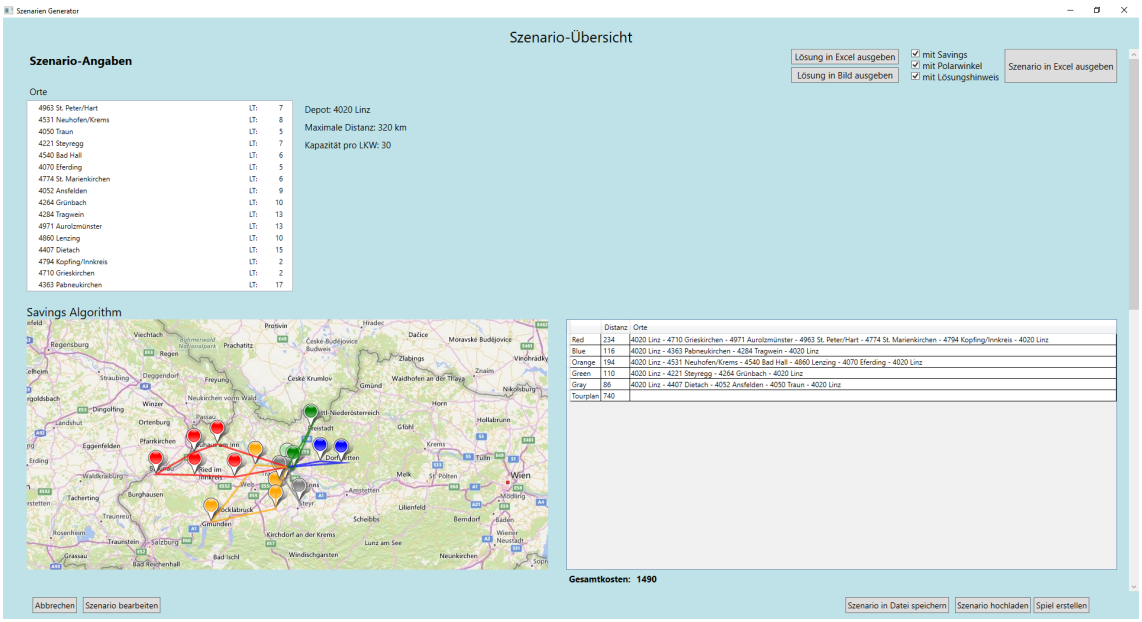

Abbildung 4.4: Übersicht über das erstellte Szenario

Die Szenario-Übersicht bietet eine Vielzahl an Funktionen. Über die Schaltfläche „Szenario bearbeiten“ gelangt man zur in Kapitel 4.1.1 beschriebenen Ansicht „Szenario manuell generieren“, wo der Benutzer Änderungen am Tourenplanungsproblem vornehmen und die Algorithmen-Lösungen neu berechnen lassen kann.
Des Weiteren gibt es die Möglichkeit, das Planungsproblem in einer Szenario-Datei zu speichern. Das so gespeicherte Planungsproblem kann dann in der Spiel-Applikation geöffnet und gelöst werden. Besteht eine Verbindung zum Server, gibt es außerdem die Option, das Szenario in der Datenbank zu speichern.

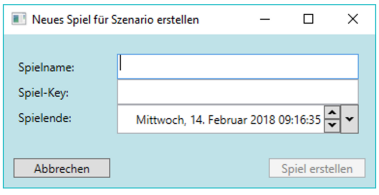

Abbildung 4.5: Neues Spiel

Zusätzlich kann aus dem Tourenplanungsproblem ein Gruppen-Spiel erstellt werden, an dem über die Spiel-Applikation teilgenommen werden kann. Wählt der Benutzer die „Spiel erstellen“-Schaltfläche, so öffnet sich das in Abbildung 4.5 gezeigte Fenster. Darin muss der Anwender für das neue Spiel einen Namen, einen eindeutigen Spielcode und die Spiellaufzeit festlegen.

Um für ein generiertes Szenario eine übersichtliche Angabe für die Studenten zum Lösen der Tourenplanungsprobleme mit den gelernten Algorithmen zu erstellen, wählt man die Schaltfläche „Szenario in Excel ausgeben“. In der Excel-Datei werden dann zwei Registerkarten erstellt. In der ersten befinden sich allgemeine Informationen zum Szenario, wie die Orte, deren Ladungsträger und Kapazitäten. In der zweiten Registerkarte wird die Distanzmatrix dargestellt. Zu dieser Schaltfläche gehören außerdem die drei Kontrollkästchen. Sind diese ausgewählt, so werden in der Excel-Datei zusätzlich noch Registerkarten mit den Savings-Werten, den Polarwinkeln für den Sweep Algorithmus und mit Lösungshinweisen angelegt. Die Lösungshinweise sind die Gesamtlängen der einzelnen Touren, die sich bei einem bestimmten Algorithmus ergeben sollen. Mit der Option „Lösung in Excel ausgeben“ kann zum Szenario eine Lösung erstellt werden. Eine weitere Möglichkeit zum Speichern einer Lösung bietet die Schaltfläche „Lösung in Bild ausgeben“. Dabei werden die einzelnen Lösungen in Form einer PNG-Datei abgespeichert, in der auch die Kartenansicht ersichtlich ist.

4.1.3 Automatisches Generieren mehrerer Szenarien

Wie in Kapitel 4.1.1 beschrieben, kann in der Ansicht „Szenario automatisch generieren“ gewählt werden, wie viele Tourenplanungsprobleme man auf Basis der angegebenen Rahmenparameter erstellen möchte. Wählt man dabei eine Anzahl größer eins, wird man nach dem Erstellen der Szenarien und der gewünschten Lösungen nicht zur in Kapitel 4.1.2 beschriebenen Szenario-Übersicht weitergeleitet, sondern man kommt zu einer eigenen Ansicht für mehrere Tourenplanungsprobleme, die in Abbildung 4.6 dargestellt ist.

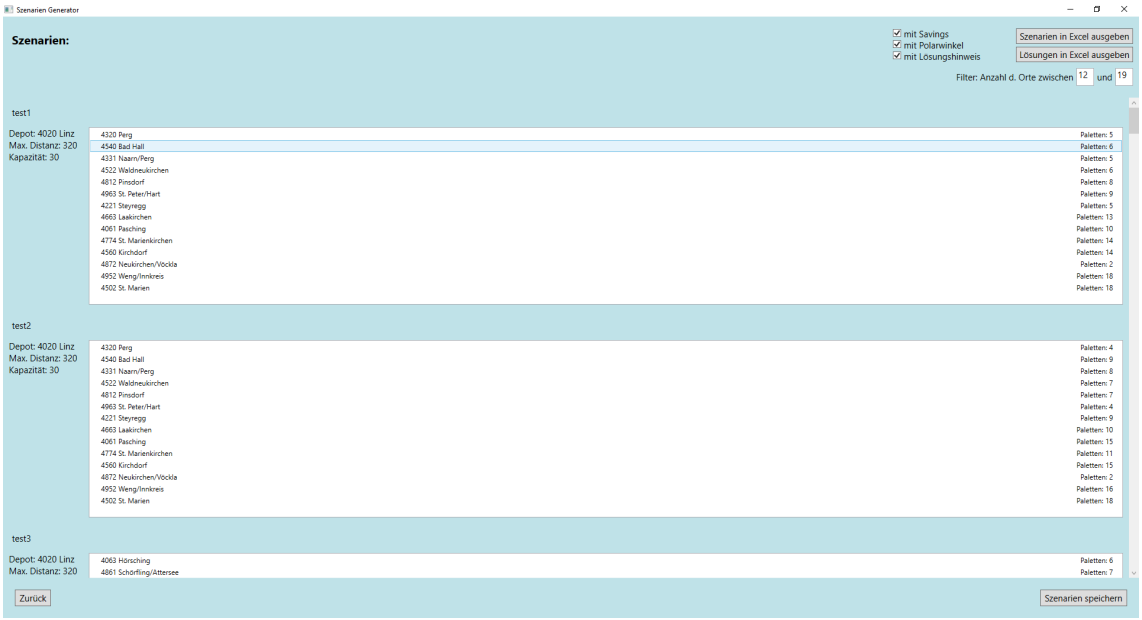

Abbildung 4.6: Übersicht über mehrere Szenarien

Eine wichtige Funktion dieser Seite ist der Filter. Wenn kapazitierte Tourenplanungsprobleme mit dem Szenariengenerator erstellt werden, wird die Größe des Szenarios über die minimale Anzahl an zu entstehenden Touren und nicht über eine fixe Anzahl an Orten vom Benutzer angegeben (siehe Kapitel 4.1.1). Daraus ergibt sich, dass die generierten Planungsprobleme eine unterschiedliche Anzahl an zu verplanenden Orten aufweisen. Mit dem Filter können diese Ab-

weichungen eingeschränkt werden, indem nur die Szenarien gewählt werden, deren Ortsanzahl sich zwischen den vom Filter definierten Werten befindet.
Die übrigen Funktionen dieser Seite sind sehr ähnlich zu denen der Szenario-Übersicht für ein einzelnes Planungsproblem (siehe Kapitel 4.1.2).
Mit der Schaltfläche „Szenarien in Excel ausgeben“ wird für jedes erstellte Szenario, das sich im Filter befindet, eine eigene Angabe in eine Excel-Datei ausgegeben. Dazu muss der Benutzer zuvor einen Ordner wählen, in den er die zu erstellenden Angaben speichern möchte. Über die Schaltfläche „Lösungen in Excel ausgeben“ wird eine Excel-Datei erstellt, in der die Lösungen für die Tourenplanungsprobleme übersichtlich aufgelistet werden. Dabei gibt es für jedes Szenario eine eigene Registerkarte. Die Schaltfläche „Szenarien speichern“ speichert die erstellten Planungsprobleme in Dateien im Szenario-Format ab, somit können diese in der Spiel-Applikation importiert werden.

4.1.4 Laufende Spiele verwalten

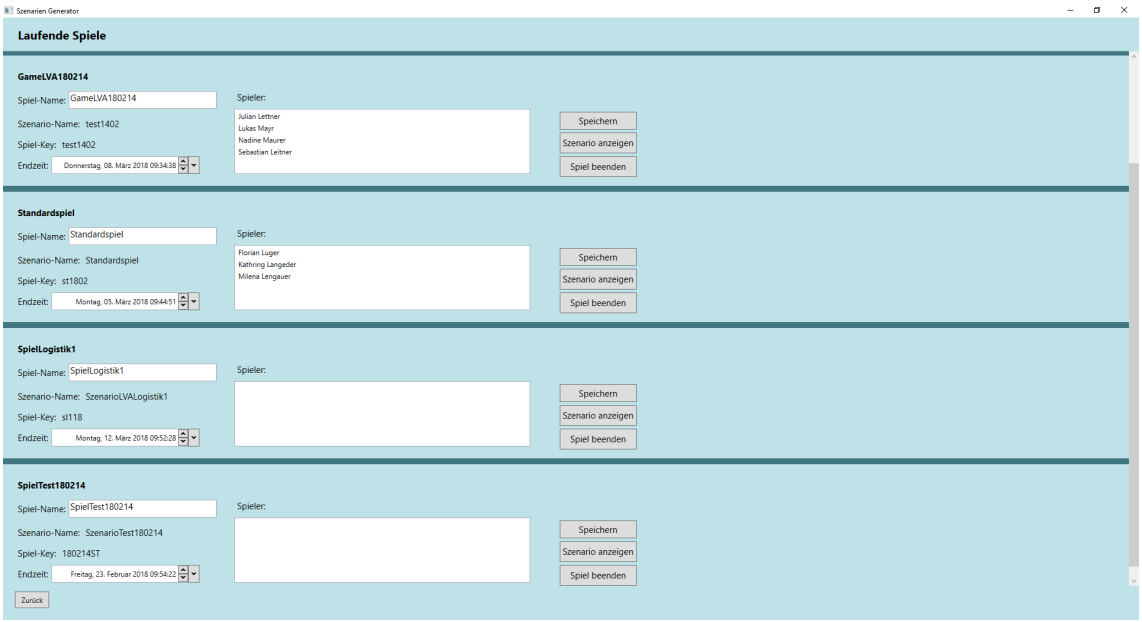

Abbildung 4.7: Ansicht zu den laufenden Spielen

Die in Abbildung 4.7 dargestellte Ansicht bietet eine Übersicht über die laufenden Spiele, also jene die noch nicht ihren Endzeitpunkt erreicht haben. Des Weiteren können hier der Spielname und die Laufzeit angepasst werden. Mit der Option „Spiel beenden“ kann der Spielleiter den Endzeitpunkt des Spiels auf die aktuelle Zeit setzen und somit das Spiel sofort beenden.
Betätigt der Benutzer die Schaltfläche „Szenario anzeigen“, wird er zur Szenario-Übersicht, die im Kapitel 4.1.2 beschrieben wurde, umgeleitet und kann hier Details zum Szenario sehen.

4.2 Spiel

Nach dem Starten einer Spiel-Applikation wird ein Benutzer gefragt, ob er eine Verbindung zum Server aufbauen will. Wählt er „ja“ und ist der Verbindungsversuch erfolgreich, stehen alle Funktionen der Software zur Verfügung. Wählt er „nein“ oder schlägt der Verbindungsversuch zum Beispiel aufgrund einer fehlenden Internetverbindung fehl, stehen keine Onlinedienste zur Verfügung. Nun hat ein Benutzer im Menü, das in Abbildung 4.8 gezeigt wird, die Möglichkeit, auf drei Arten ein Spiel zu starten. Er kann entweder einem vom Spielleiter gestarteten Gruppenspiel beitreten, indem er „Spiel beitreten“ auswählt, mit „Spiel laden“ ein Szenario aus einer Datei starten oder mit „Standardspiel“ ein fix in das Programm integriertes Standardszenario beginnen. Ist keine Internetverbindung verfügbar, steht die Option „Spiel beitreten“ nicht zur Verfügung. Über die Schaltfläche „Verbindung herstellen“ ist es in diesem Fall möglich, einen neuen Verbindungsversuch zu unternehmen.

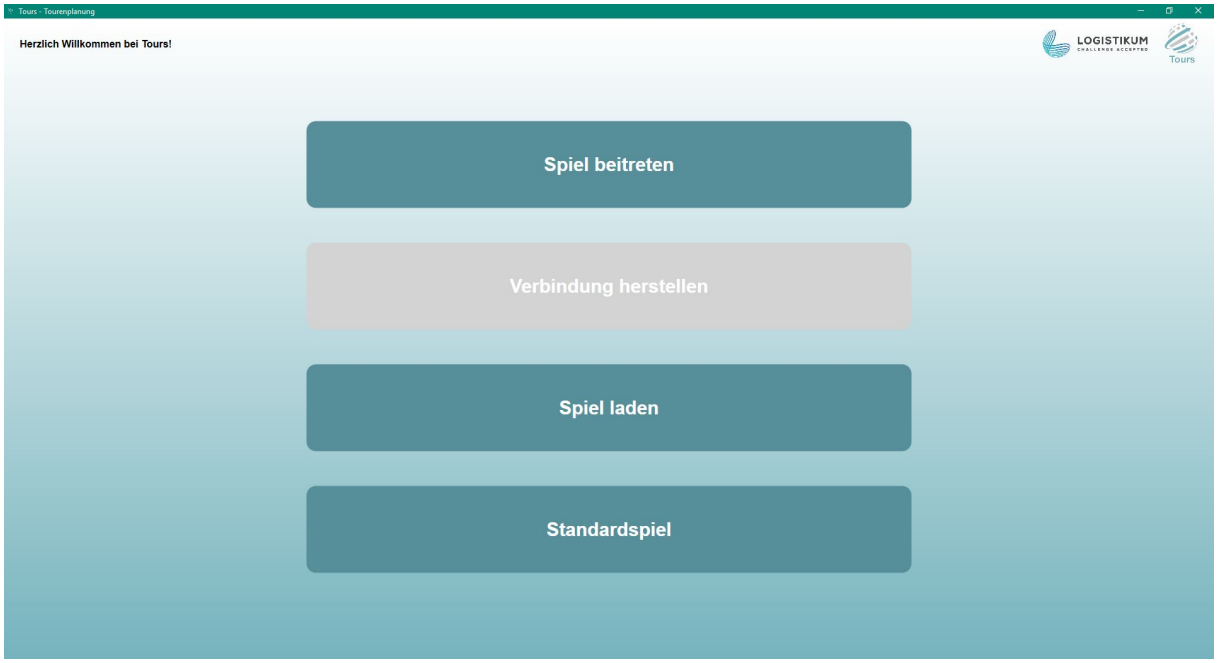

Abbildung 4.8: Hauptmenü

4.2.1 Spiel beitreten

Entscheidet sich ein Benutzer dafür, einem Gruppenspiel beizutreten, muss er in der Ansicht „Spiel beitreten“, die in Abbildung 4.9 zu sehen ist, zuerst einen vom Spielleiter erhaltenen Gamecode angeben, welcher auf seine Gültigkeit überprüft wird. Schlägt die Überprüfung fehl, kann der Nutzer einen anderen Code eingeben und es erneut versuchen. Nach erfolgreicher Prüfung kann ein Spielername angegeben und die Schaltfläche „Spiel beitreten“ gewählt werden. Existiert im geladenen Spiel noch kein Spieler mit dem angegebenen Namen, wird ein neuer angelegt und ein neues leeres Szenario geladen. Existiert jedoch bereits ein Spieler mit diesem Namen, wird der Benutzer aufgefordert, einen anderen Spielernamen zu wählen. Alternativ dazu ist es möglich, eine bereits abgegebene Lösung erneut zu laden und direkt beim gespeicherten Stand weiterzuspielen.

Abbildung 4.9: Ansicht zum Beitreten zu einem Gruppenspiel

4.2.2 Spiel laden

Wählt ein Benutzer im Hauptmenü die Option „Spiel laden“, öffnet sich ein Explorer-Fenster, in dem gespeicherte Dateien vom Typ *.szenario* gewählt werden können. Diese können entweder mit Hilfe eines Szenariengenerators erstellt (siehe Kapitel 4.1.1) oder durch das Speichern eines geöffneten Spiels in eine Datei generiert worden sein.
Hat sich ein Benutzer für eine Datei entschieden, wird das darin enthaltene Szenario geladen. Handelt es sich um eine Datei ohne vorhandener Benutzerlösung, erscheint ein gänzlich ungelöstes Szenario. Ist in einer Datei hingegen bereits eine vollständige oder teilweise Lösung vorhanden, so wird diese geladen und in der Spielansicht angezeigt.

4.2.3 Standardspiel

Die letzte Möglichkeit, ein Spiel zu starten, ist ein Standardspiel. Dabei handelt es sich um ein vorgefertigtes Szenario mittleren Schwierigkeitsgrades, das schnell und ohne einem Onlinespiel beitreten oder eine Szenario-Datei suchen zu müssen, gespielt werden kann. Wählt ein Benutzer das Standardspiel, wird sofort das festgelegte Standardszenario in der Spielansicht geöffnet (siehe Kapitel 4.2.4).

4.2.4 Spielansicht

Die Spielansicht ist der Hauptbildschirm der Applikation und unterteilt sich in zwei große Ansichten, eine Planungs-, sowie eine Kartenansicht.

Planungsansicht

Die in Abbildung 4.10 dargestellte Planungsansicht ist die Startansicht und zugleich auch die Bildschirmmaske, die es einem Benutzer erlaubt, Eingaben zu tätigen. In der oberen Leiste

befinden sich diverse Informationen, die einen Benutzer bei der Planung seines Tourenplans unterstützen. Sie umfassen die folgenden Angaben:

- Depot: Dies ist der Ort, von dem aus jede Tour startet und an dem auch jede wieder endet. Das Depot wird automatisch an den Anfang und das Ende jeder Tour gestellt.
- Maximale Tourlänge: Diese Distanz darf von keiner Tour überschritten werden, um eine gültige Lösung zu erstellen.
- Kapazität: Jede Tour kann lediglich eine gewisse Anzahl an Ladungsträgern transportieren. Diese darf von keiner Tour überschritten werden, um eine gültige Lösung zu erhalten.
- Kosten pro Tour in Euro: Jede Tour hat gewisse Fixkosten, die anfallen, sobald mindestens ein Ort beliefert wird. Eine leere Tour verursacht keine Fixkosten.
- Kosten pro Kilometer in Euro: Dieser Betrag wird mit der Länge jeder Tour in Kilometern multipliziert und zu den Fixkosten addiert. Dadurch ergeben sich letztendlich Gesamtkosten.

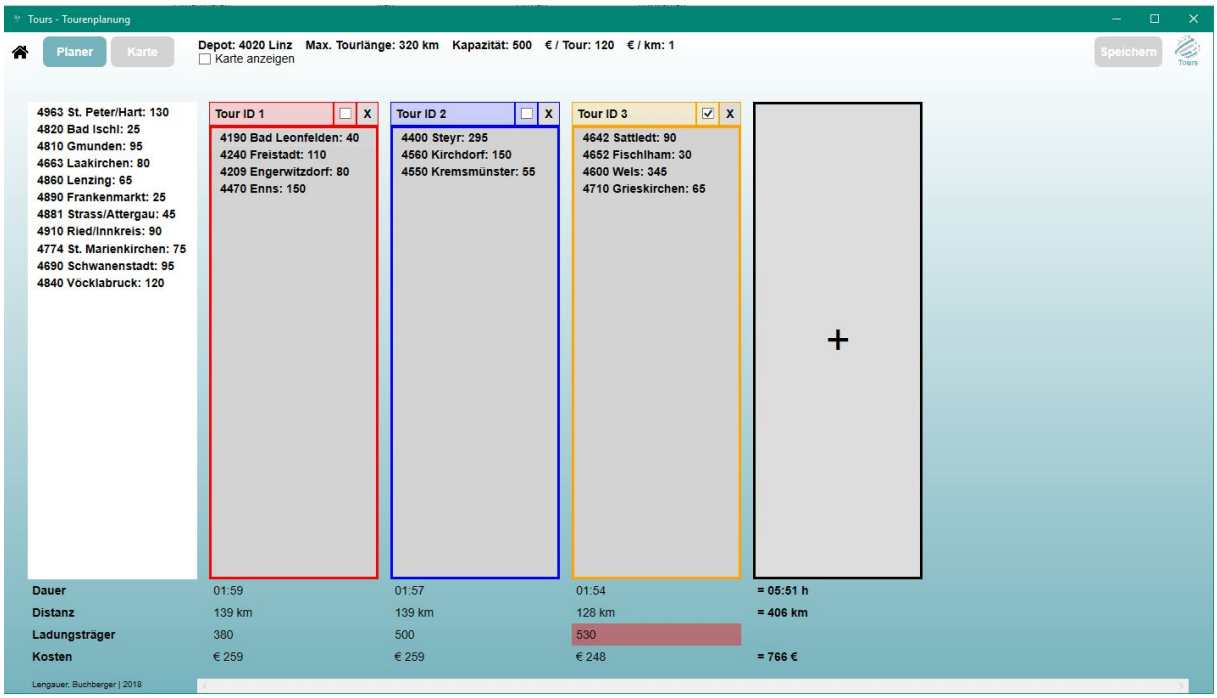

Abbildung 4.10: Planungsansicht mit einer ungültigen Tour

Im zentralen Bereich der Planungsansicht befinden sich anfangs zwei Boxen, sowie eine große Plus-Schaltfläche. Die linke - weiße - Box zeigt die Namen aller zu beliefernden Orte und die jeweils zu liefernde Anzahl an Ladungsträgern. Die zweite Box ist bereits eine Tour, der mittels Drag-and-Drop Orte aus der linken Box zugeordnet werden können. Des Weiteren ist es möglich, die zugeordneten Orte innerhalb der Box ebenfalls mittels Drag-and-Drop in eine günstige Reihenfolge zu bringen. Unter der Touren-Box befinden sich die Kennzahlen Dauer, Distanz und Ladungsträger, sowie die aus diesen Daten resultierenden Kosten. Wird von einer Tour die maximal erlaubte Distanz oder Höchstanzahl transportierter Ladungsträger überschritten, wird der

jeweilige Wert rot eingezeichnet.
Durch Klicken auf die Plus-Schaltfläche kann eine zusätzliche Tour hinzugefügt werden. Diese wird zwischen der letzten Tour und der Plus-Schaltfläche in einer neuen Farbe eingezeichnet. Möchte man eine Tour wieder löschen, ist dies über das X-Symbol in der rechten oberen Ecke der jeweiligen Tour möglich.
Die gesamte Dauer und Distanz der Tourenplanung sowie die Gesamtkosten sind jederzeit in den entsprechenden Feldern unter der Plus-Schaltfläche ersichtlich.

Kartenansicht
Die in Abbildung 4.11 gezeigte Kartenansicht stellt einen erstellten Tourenplan grafisch dar. Dazu werden alle noch zu beliefernden Orte auf einer Straßenkarte grau eingezeichnet und die Orte, die bereits einer Tour zugeordnet sind werden mit der entsprechenden Farbe markiert. Auch werden die zugeteilten Orte mit einer Linie in der jeweiligen Farbe verbunden, um die Sortierung überprüfen zu können. Fährt ein Benutzer mit der Maus über einen Ort, werden ihm die Entfernungen von diesem Ort zu jedem anderen als Auszug aus der Distanzmatrix angezeigt. Über die Schaltfläche „zurück zum Depot“ zoomt die Kartenansicht wieder zum Ausgangspunkt zurück, um die Orientierung zu erleichtern.

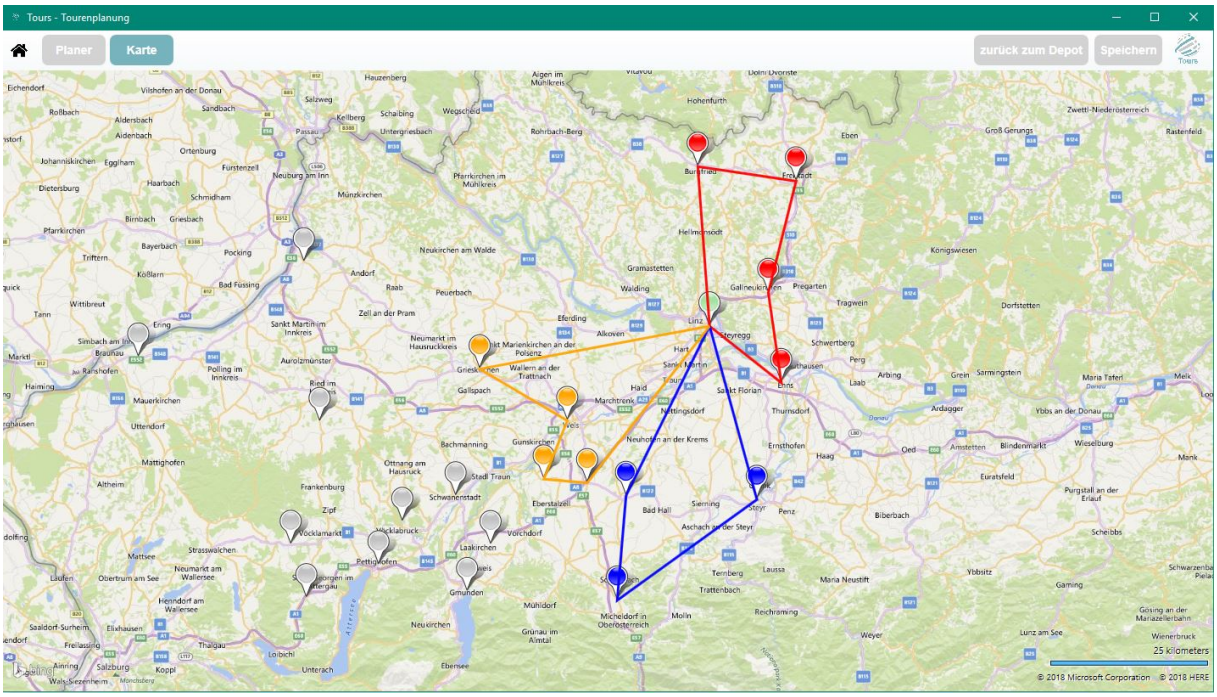

Abbildung 4.11: Kartenansicht

Eine weitere Möglichkeit zur Anzeige einer Kartenansicht stellt die in Abbildung 4.12 gezeigt Minikarte in der Planungsansicht dar. Sofern eine Internetverbindung besteht, kann durch Aktivieren des Kontrollkästchens „Karte anzeigen“ eine kleine Kartenansicht im rechten Bereich der Planungsansicht eingeblendet werden. In dieser sind alle noch nicht zugeteilten Orte eingezeichnet und man kann einzelne Touren einblenden, indem man das entsprechende Kontrollkästchen neben der Bezeichnung der jeweiligen Tour aktiviert. Weiß ein Benutzer nicht, wo ein bestimmter

Ort auf der Karte zu finden ist, kann er ihn im Planer auswählen, woraufhin er in der Minikarte mit einem weißen Pin hervorgehoben wird.

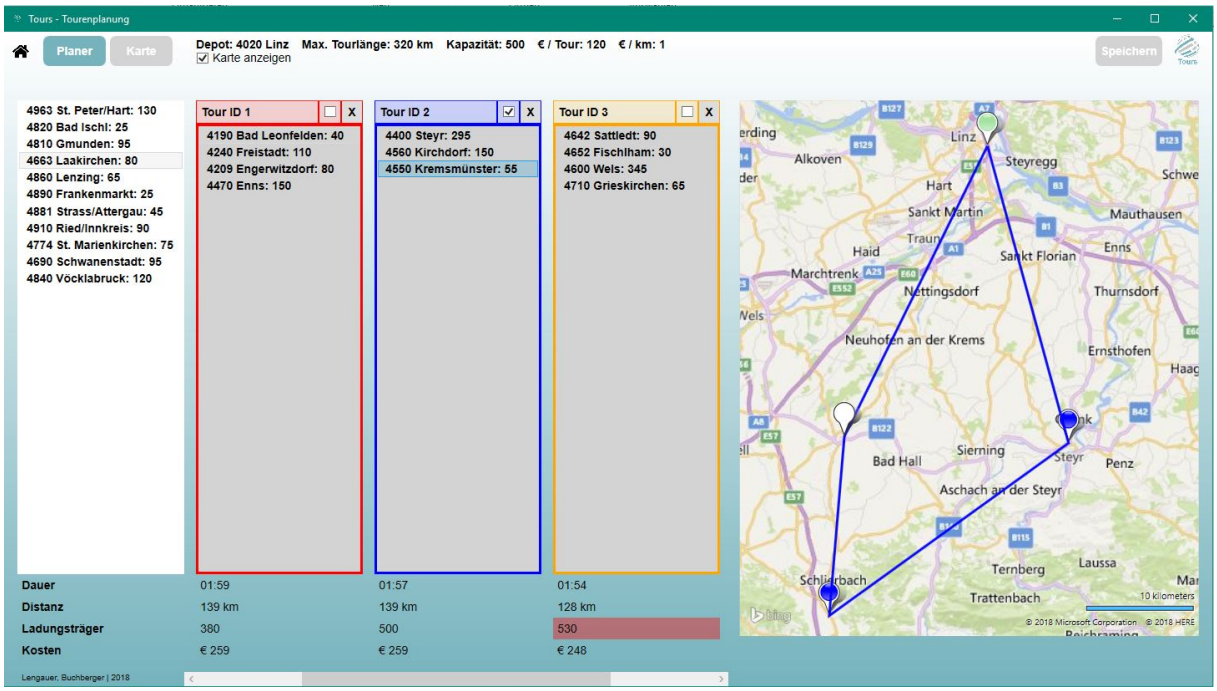

Abbildung 4.12: Planungsansicht mit eingeblendeter Minikarte und hervorgehobenem Ort

Möchte ein Benutzer seine Lösung oder sein Zwischenergebnis speichern, hat er jederzeit die Möglichkeit, die Schaltfläche „Speichern“ zu wählen. Wurde zu Beginn im Menü „Spiel laden“ oder „Standardspiel“ gewählt, kann nun eine Szenario-Datei gespeichert werden, die später über „Spiel laden“ wieder geöffnet werden kann. Diese Datei kann auch im Szenariengenerator in der Szenario-Übersicht geöffnet werden (siehe Kapitel 4.1.2). In dieser Sicht wird dann die Lösung des Benutzers mit sämtlichen Touren angezeigt und es ist auch ersichtlich, welche Touren ungültig sind, da sie entweder eine zu hohe Distanz aufweisen oder eine zu hohe Anzahl an Ladungsträgern transportieren.
Wurde zu Beginn „Spiel beitreten“ gewählt, besteht zusätzlich die Möglichkeit, das Ergebnis online abzugeben. Entscheidet sich ein Benutzer für diese Option, wird überprüft, ob die vom Spielleiter für das jeweilige Spiel festgelegte Abgabezeit noch nicht überschritten wurde. Im Fall einer Zeitüberschreitung, ist die Online-Abgabe nicht mehr möglich und es kann nur noch in einer lokalen Datei gespeichert werden. Falls die Abgabezeit jedoch noch nicht überschritten wurde, wird die gesamte Lösung in der Datenbank gespeichert. Wurde vom selben Benutzer zum gleichen Spiel bereits eine Lösung abgegeben, gibt es zwei Möglichkeiten zur Auswahl. Dabei kann sich ein Benutzer dafür entscheiden, die vorherige Abgabe zu überschreiben oder den Speichervorgang abzubrechen und doch nur in eine lokale Datei zu speichern. Alle online abgegebenen Lösungen können vom Spielleiter mit einem Szenariengenerator eingesehen werden (siehe Kapitel 4.1.2). Über die Schaltfläche mit dem Haus-Symbol kann jederzeit in das Hauptmenü zurückgekehrt werden.

Kapitel 5

Programmstruktur

Das Softwaresystem, das im Rahmen dieser Diplomarbeit entwickelt wurde, besteht aus zwei Programmteilen, einem Szenariengenerator, der auf eine Algorithmen-Bibliothek zugreift, und einer Spiel-Applikation. Die Kommunikation zwischen diesen beiden Komponenten erfolgt entweder über den Austausch von Dateien oder über die Datenbank.
Die Architektur und Kommunikation zwischen den einzelnen Teilen ist in Abbildung 5.1 dargestellt und wird in diesem Kapitel genauer beschrieben.

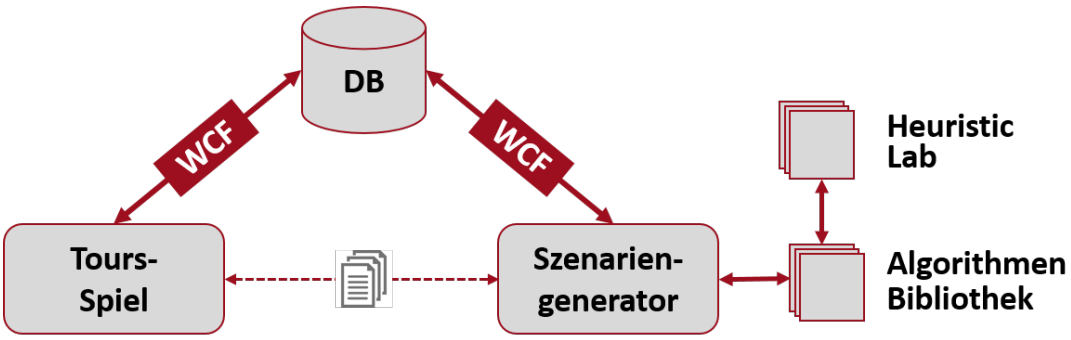

Abbildung 5.1: Programmarchitektur

5.1 Datenbank

Als Datenbank wird ein Microsoft SQL Server 2016 verwendet. Dieser läuft auf einem Windows Server 2012 R2. Als Authentifizierung für die Datenbank wird die Windows Authentication verwendet.
Die Datenbank dient als Datenbasis für die Spiel-Applikation und den Szenariengenerator. Es werden dort die möglichen Orte und die dazugehörigen Distanzen, die mit dem Szenariengenerator erstellten Szenarien und Spiele, die an den Spielen teilnehmenden Benutzer sowie Lösungen zu Szenarien gespeichert. In Abbildung 5.2 wird das Datenmodell dazu dargestellt.

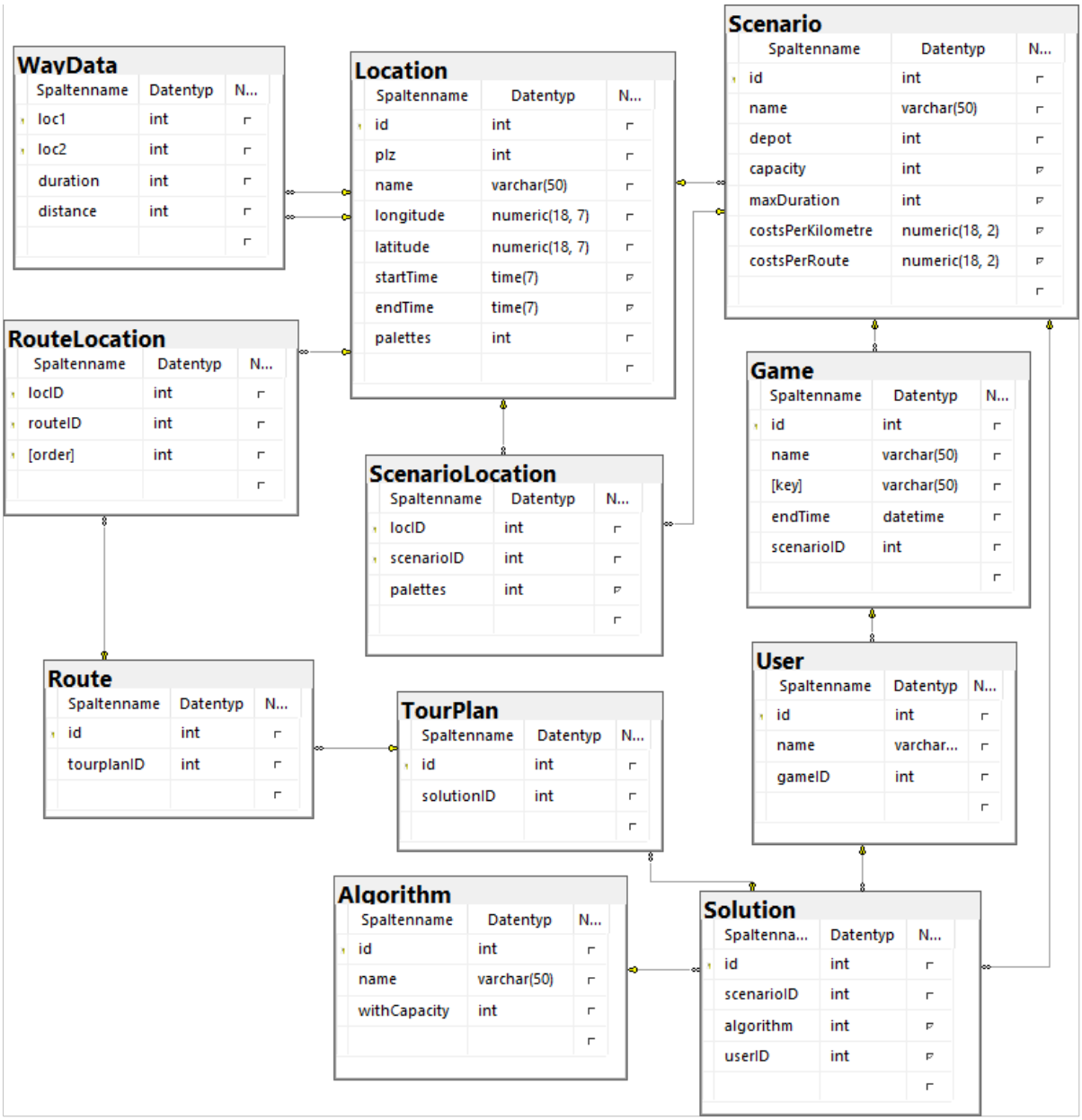

Abbildung 5.2: Datenmodell

5.1.1 Location

In der Tabelle *Location* werden die Orte gespeichert, mit denen im Szenariengenerator Szenarien erstellt werden können. Dazu wurden am Beginn dieser Diplomarbeit gemeinsam mit dem Auftraggeber 100 mögliche Orte definiert. Zu jedem Ort werden folgende Attribute abgespeichert:

- Name und Postleitzahl des Ortes
- Längen- und Breitengrad: Diese Daten wurden über die Micrsoft Bing Maps Locations API bezogen (siehe Kapitel 6.3.3). Im Programm werden diese Daten zur Darstellung der Orte

in der Kartenansicht und in der Algorithmen-Bibliothek zur Berechnung der Polarwinkel für den Sweep Algorithmus benötigt.

- Paletten: Pro Ort wird ein Standardwert für die Anzahl der zu liefernden Ladungsträger gespeichert. Dieser wird beim manuellen Generieren eines Szenarios vorgeschlagen, kann aber durch den Benutzer beliebig geändert werden. Der geänderte oder auch übernommene Wert wird dann im Feld *palettes* der Tabelle *ScenarioLocation* für das entsprechende Szenario gespeichert.

- Start- und Endzeit: Diese Felder haben den Zweck, für den jeweiligen Ort das Zeitfenster, in dem eine Zulieferung stattfinden kann, anzugeben. Da die kapazitierte Tourenplanung mit Zeitfenstern jedoch nicht Teil dieser Arbeit ist, sind sie derzeit standardmäßig für jeden Ort auf 00:00 und 24:00 Uhr gesetzt, also auf einen ganzen Tag, was einem Planungsproblem ohne Zeitfenstern entspricht.

5.1.2 WayData

In der Tabelle *WayData* werden die Distanzen zwischen den einzelnen Orten gespeichert. Diese Daten sind für die Berechnung von Distanzmatrizen zu Szenarien nötig. Dabei wird davon ausgegangen, dass es sich um symmetrische Matrizen handelt (siehe Kapitel 2.1.2). Insgesamt ergibt sich dadurch für die festgelegten 100 Orte eine Anzahl von 4.950 *WayData*-Einträgen.
Die Tabelle *WayData* umfasst folgende Attribute:

- Location 1 und Location 2: referenzieren je einen Eintrag der Tabelle *Location*

- Distanz und Dauer: In diesen Attributen werden die zwischen den beiden Orten liegenden Kilometer sowie die dafür benötigte Fahrzeit gespeichert. Diese Daten wurden mit der Microsoft Bing Maps Routes API berechnet (siehe Kapitel 6.3.3).

5.1.3 Scenario

In der Tabelle *Scenario* werden die generierten Szenarien mit folgenden Attributen gespeichert:

- Name: Im Szenariengenerator wird ein Name für das Szenario festgelegt.

- Depot: Dieses Attribut ist eine Referenz auf einen Eintrag der Tabelle *Location*. Das Depot ist der Ort, bei dem jede Tour startet und endet.

- Kapazität und maximale Distanz: Diese Attribute werden für kapazitierte Tourenplanungsprobleme benötigt. Das Attribut Kapazität legt dabei die maximale Anzahl an Ladungsträgern fest, die pro Tour ausgeliefert werden kann. Mit der maximalen Distanz schafft man eine Einschränkung der Länge einer Tour.

- Kosten pro Route und Kosten pro Kilometer: Diese Werte benötigt man zur Berechnung der Gesamtkosten eines Tourenplans. Jede begonnene Tour hat dabei Fixkosten. Hinzu kommen die variablen Kosten pro Kilometer.

5.1.4 ScenarioLocation

Die Tabelle *ScenarioLocation* stellt die Assoziativtabelle zwischen *Location* und *Scenario* dar. Die Tabelle hat einen zusammengesetzten Primärschlüssel, der aus einer Referenz auf einen *Location*-Eintrag und einer Referenz auf einen *Scenario*-Eintrag besteht. Damit wird verhindert, dass in einem Szenario Orte mehrfach vorkommen können. Zusätzlich dazu wird hier noch die Zahl

der im betroffenen Szenario zum jeweiligen Ort zu liefernden Ladungsträger gespeichert. Dieser Wert wird beim manuellen Erstellen des Szenarios vom Benutzer selbst festgelegt und bei der automatischen Szenariogenerierung zufällig nach dem in Kapitel 6.5 beschriebenen Algorithmus gewählt.

5.1.5 Game

Diese Tabelle enthält einen Eintrag für jedes Gruppen-Spiel, das im Szenariengenerator gestartet wird. Dafür besteht sie aus folgenden Attributen:

- Name: Jedem Spiel muss im Szenariengenerator ein Name gegeben werden. Dieser dient zur Identifikation eines Spiels in der Ansicht der aktuell laufenden Spiele (siehe Kapitel 4.1.4).
- Spielcode (Key): Dieser stellt einen eindeutigen Schlüssel für die Tabelle *Game* dar und wird in der Spiel-Applikation benötigt, um einem Gruppen-Spiel beizutreten. Die Spalte wurde in der Datenbank als *UNIQUE* definiert.
- Endzeit: Dieses Attribut definiert die Zeit, bis zu der Benutzer über die Spiel-Applikation dem Gruppenspiel beitreten und Lösungen für das Szenario abgeben können.
- Szenario: Jedes Spiel referenziert ein Szenario, wobei pro Szenario auch mehrere unterschiedliche Spiele gestartet werden können. Es ist nicht möglich, in der Spiel-Applikation ein Szenario aus der Datenbank direkt zu öffnen, sondern es muss immer auf einen *Game*-Eintrag zugegriffen werden.

5.1.6 User

Wenn ein Benutzer einem Gruppen-Spiel beitritt, wird in der Tabelle *User* ein neuer Eintrag für ihn angelegt. Die Tabelle *User* dient dazu, die zu einem Spiel abgegebenen Lösungen den Benutzern zuzuordnen. Mit dem Attribut *gameID* wird der *Game*-Eintrag des Spiels, an dem der Benutzer teilnimmt, referenziert. Die Spalten *name* und *gameID* zusammen sind in der Datenbank als *UNIQUE* definiert, was gewährleistet, dass die Benutzernamen pro Spiel eindeutig sind.

5.1.7 Algorithm

In dieser Tabelle werden jene Algorithmen gespeichert, mit denen Lösungen zu Szenarien berechnet werden können. Dazu gibt es die Unterscheidung zwischen kapazitierten Tourenplanungsproblemen und Problemen ohne Kapazitäten.

5.1.8 Solution

Diese Tabelle enthält alle gespeicherten Lösungen zu verschiedenen Szenarien. Dabei wird zwischen Lösungen, die von Algorithmen berechnet wurden und Lösungen von Benutzern unterschieden. Folgende Attribute werden hier benötigt:

- Szenario-ID: Referenz auf das Szenario, zu dem die jeweilige Lösung gehört
- Algorithmus: Falls die Lösung von einem Algorithmus berechnet wurde, befindet sich hier der Verweis auf den entsprechenden Eintrag in der *Algorithm*-Tabelle. Handelt es sich um eine Benutzerlösung, ist dieser Eintrag *null*.

- User: Stammt die Lösung von einem Benutzer, wird hier auf den jeweiligen Eintrag der *User*-Tabelle verwiesen, wenn dem nicht so ist, ist der Eintrag *null*.

Über die Einträge *scenarioID*, *algorithm* und *userID* ist ein UNIQUE-Schlüssel gelegt, der dafür sorgt, dass weder von einem Benutzer noch zu einem Algorithmus mehrere Lösungen für ein Szenario gespeichert werden können.

Die Tabelle *Solution* ist zusätzlich zur *TourPlan*-Tabelle nötig, weil es sein kann, dass bei der Berechnung einer Lösung mit einem Algorithmus zwei oder mehrere Tourenpläne ein gleich gutes Ergebnis liefern.

5.1.9 TourPlan

Die Tabelle *TourPlan* enthält eine Referenz auf einen *Solution*-Eintrag und befindet sich zwischen den Tabellen *Route* und *Solution*.

5.1.10 Route

Diese Tabelle enthält eine Referenz auf einen Eintrag der Tabelle *TourPlan*.

5.1.11 RouteLocation

Die Tabelle *RouteLocation* definiert, welche Orte aus der Tabelle *Location* zu welchen Routen gehören. Dafür enthält sie folgende Attribute:

- Location: referenziert einen Eintrag der Tabelle *Location*
- Route: definiert, zu welcher Route der jeweilige Eintrag gehört
- Order: Dieses Attribut legt fest, in welcher Reihenfolge die zu einer Tour gehörenden Orte angefahren werden müssen.

Zwischen den Tabellen *Route* und *Location* besteht eine m:n-Verbindung, die mit dieser Tabelle aufgelöst wird.

5.1.12 Speicherung von Lösungen

In diesem Abschnitt wird zum besseren Verständnis noch einmal die Speicherung von Lösungen in den einzelnen Tabellen näher erklärt.
Jede Lösung besteht aus einem oder mehreren Tourenplänen. Mehrere Tourenpläne pro Lösung entstehen, wenn der angewandte Lösungsalgorithmus Pläne berechnet, die in Bezug auf ihre Gesamtkosten gleich gut sind. Jeder Tourenplan besteht aus einer oder mehreren Routen und jede Route besteht aus einer Aneinanderreihung von Orten in einer bestimmten Reihenfolge. Zu Beginn und am Ende jeder Route steht das im Szenario definierte Depot.
In Abbildung 5.3 wird die Speicherung von Lösungen übersichtlich dargestellt.

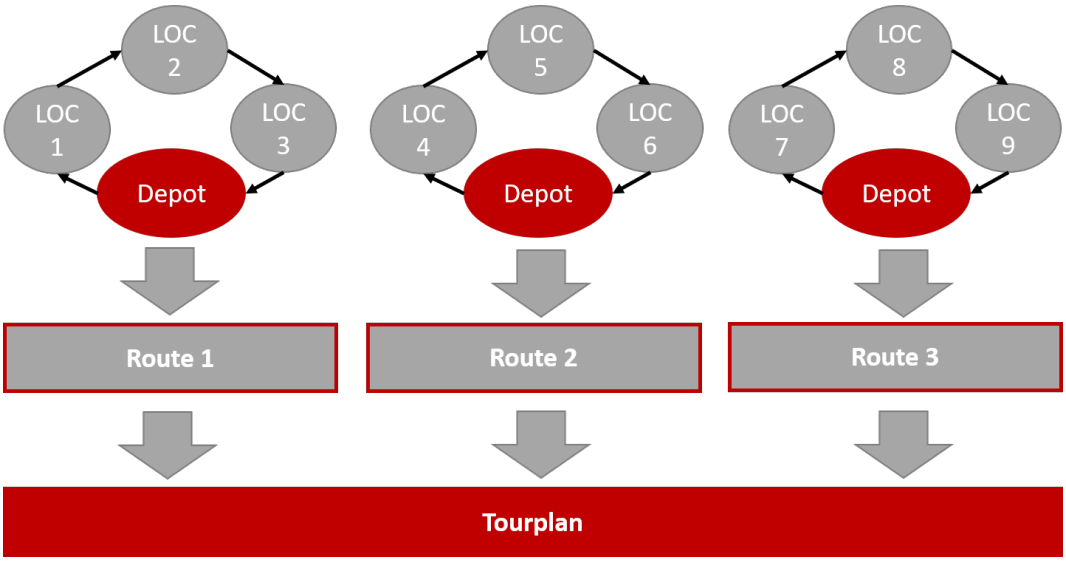

Abbildung 5.3: Speicherung von Lösungen

5.2 WCF-Service

Über einen WCF-Dienst wird der Zugriff auf die Datenbasis für beide Programme ermöglicht. Das Hosting des Dienstes übernimmt ein IIS-Webserver, wobei als Zugriffsprotokoll HTTPS verwendet wird. Das dazu benötigte SSL-Zertifikat stammt vom Anbieter „Let's Encrypt“. Mit der kostenlosen Software „Certify the web“ ist es möglich, diese Zertifikate zu installieren und bei Ablauf automatisch zu erneuern.

Für die Authentifizierung wird im IIS die Basic Authentication verwendet. Bei dieser Art der Authentifizierung werden die Anmeldedaten direkt im HTTP-Authorization-Header in der Form “username:password“ unverschlüsselt bei jeder Anfrage mitgesendet. Durch die Verschlüsselung mit SSL kann die Basic Authentifizierung jedoch trotzdem sicher verwendet werden.

5.3 Szenariengenerator

Der Szenariengenerator ist eine WPF-Applikation, mit der Tourenplanungsprobleme für die Spiel-Applikation erstellt werden können. Der Austausch von Szenarien erfolgt dabei entweder über einen Dateiexport oder über die gemeinsame Datenbank. Damit im Szenariengenerator Lösungen für die erstellten Tourenplanungsprobleme berechnet werden können, enthält die Applikation eine Referenz auf die Algorithmen-Bibliothek.
In Abbildung 5.4 sind die wichtigsten Klassen des Szenariengenerators sowie deren Methoden und Attribute in Form eines UML Klassendiagramms dargestellt.

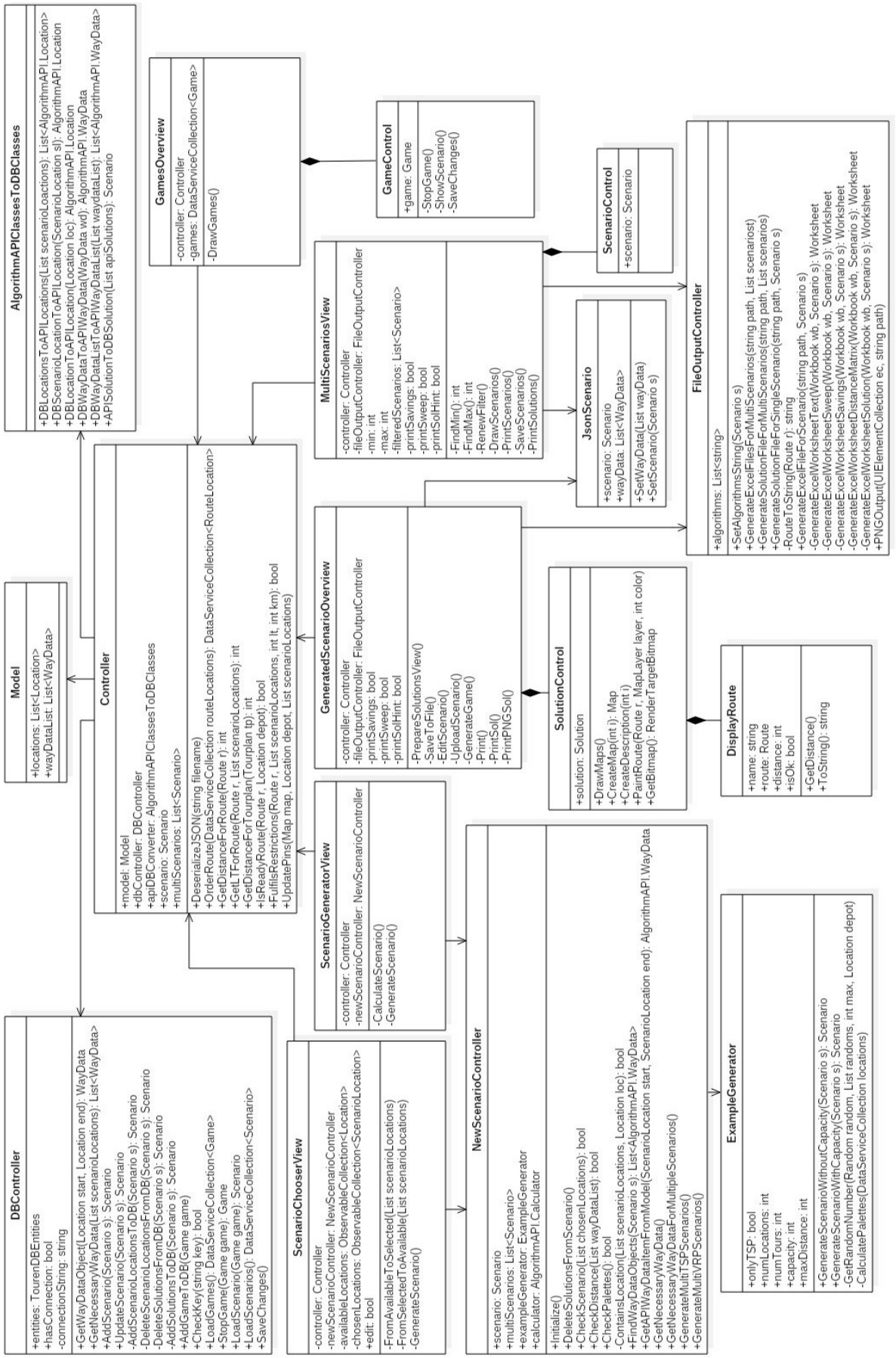

Abbildung 5.4: Klassendiagramm des Szenariengenerators

5.3.1 Programm-Logik

In diesem Abschnitt werden jene Klassen beschrieben, die die Programm-Logik bereitstellen.

Controller
Von der Klasse *Controller* wird bei Programmstart einmalig ein Objekt angelegt. Sie stellt wichtige Methoden, die in der gesamten Anwendung immer wieder benötigt werden, bereit. Ihre Attribute sind das aktuelle Szenario bzw. die aktuelle Liste der Szenarien und ein Objekt der Klasse *Model*. Des Weiteren hält er Referenzen auf die Klassen *DBController* und *AlgorithmAPIClassesToDBClasses*.

Model
Im *Model*-Objekt werden sämtliche verfügbaren Orte und die für das Szenario benötigten *WayData*-Objekte gespeichert. Die verfügbaren Orte werden beim Start der Anwendung geladen. Die *WayData*-Liste wird mit den nötigen Objekten befüllt, sobald ein Szenario neu erstellt oder ein bereits bestehendes geladen wurde.

DBController
Diese Controller-Klasse verwaltet den Zugriff auf die Datenbank über den WCF-Dienst und baut nach dem Start der Applikation die Verbindung auf. Hier wird außerdem gespeichert, ob der WCF-Dienst derzeit erreichbar ist. Schlägt ein Zugriff während der Ausführung des Programms fehl, so wird das Programm im Offline-Modus fortgesetzt.

AlgorithmAPIClassesToDBClasses
Hier werden die Klassen der Algorithmen-Bibliothek in die Klassen, die vom WCF-Dienst bereitgestellt werden, umgewandelt.

NewScenarioController
Diese Klasse verwaltet das manuelle oder auch automatische Erstellen neuer Szenarien. Sie speichert die noch unfertigen Szenarien bzw. Szenario-Listen zwischen und übergibt sie dem *Controller*-Objekt sobald sie fertig erstellt wurden. Neben dem Szenario und der Szenario-Liste enthält sie außerdem ein Objekt der Klasse *ExampleGenerator*. Die *ExampleGenerator*-Klasse kommt beim automatischen Generieren eines Szenarios zum Einsatz. Sie hat die Aufgabe, die Orte und deren Ladungsträger unter Beachtung der eingegebenen Parameter für das Szenario zufällig auszuwählen. Ein weiteres Attribut der *NewScenarioController*-Klasse ist ein Objekt der Klasse *AlgorithmAPI.Calculator*. Dieses Objekt stellt die Schnittstelle zur Algorithmen-Bibliothek dar, über die Lösungen für ein Szenario berechnet werden.

FileOutputController
Diese Controller-Klasse bietet Methoden, um Szenarien bzw. Lösungen zu Szenarien in lesbarer Form auszugeben und abzuspeichern. Außerdem können mit dieser Klasse Angaben zu einem Szenario für Studenten erstellt werden. Es gibt hier die Möglichkeit einer Ausgabe in eine Excel-Datei oder einer Speicherung der Szenario-Lösungen mit der Kartenansicht in ein Bild.

JsonScenario
Diese Klasse wird für das Speichern eines Szenarios in ein File im JSON-Format benötigt. Mit dieser Klasse wird ein einheitliches Format für die JSON-Datei festgelegt und somit eine Schnittstelle zwischen der Spiel-Applikation, die diese Klasse ebenfalls implementiert, und dem Szenariengenerator geschaffen.

5.3.2 Programm-Sichten

Die folgenden Klassen sind abgeleitet von der Klasse *System.Windows.Control.Page* und stellen die wichtigsten Sichten des Szenariengenerators dar. Sie alle enthalten eine Referenz auf das *Controller*-Objekt.

HomeView
Diese Seite ist die Startseite der Anwendung. Sie enthält Referenzen auf sämtliche anderen Sichten.

ScenarioChooserView
Mit dieser Sicht wird das manuelle Erstellen eines Szenarios abgebildet. Die Klasse enthält dazu eine Referenz auf ein *NewScenarioController*-Objekt, das diesen Prozess verwaltet.

ScenarioGeneratorView
Diese Sicht ermöglicht das automatische Generieren eines oder mehrere Szenarien. Auch hier besteht dazu die Referenz auf ein *NewScenarioController*-Objekt.

GeneratedScenarioOverview
Hier erhält der Nutzer eine Übersicht über ein neu erstelltes oder ein aus einer Datei bzw. der Datenbank geladenes Szenario. Die Lösungen zu diesem Szenario werden dabei mit dem selbst definierten User-Control-Element *SolutionControl* dargestellt.

MultiScenariosView
Diese Sicht zeigt einen Überblick über eine Reihe von neu erstellten Szenarien mit den selben Rahmenparametern. Die Szenarien werden hier über das User-Control-Element *ScenarioControl* abgebildet.

GamesOverview
Hier werden sämtliche laufenden Spiele geladen und angezeigt. Des Weiteren können Bearbeitungen durchgeführt werden. Die einzelnen Spiele werden dabei mit dem User-Control-Element *GameControl* dargestellt.

Wie der Umgang mit den User-Control-Elementen im Programm erfolgt, wird im Kapitel 6.6.5 näher beschrieben.

5.4 Algorithmen-Bibliothek

In der Algorithmen-Bibliothek wurden der Savings-Algorithmus mit und ohne Kapazitäten, der Sweep Algorithmus, der Nearest Neighbour Algorithmus und zwei Abwandlungen des Nearest Neighbour Verfahrens, die Kapazitäten berücksichtigen, selbst implementiert.
Die Tabu Search wurde über entsprechende Klassenbibliotheken aus HeuristicLab eingebunden. Details und Codeauszüge zur Durchführung der Tabu Search über das HeuristicLab befinden sich im Kapitel 6.2.2.
Abbildung 5.5 zeigt den Aufbau der Algorithmen-Bibliothek in Form eines UML Klassendiagramms.

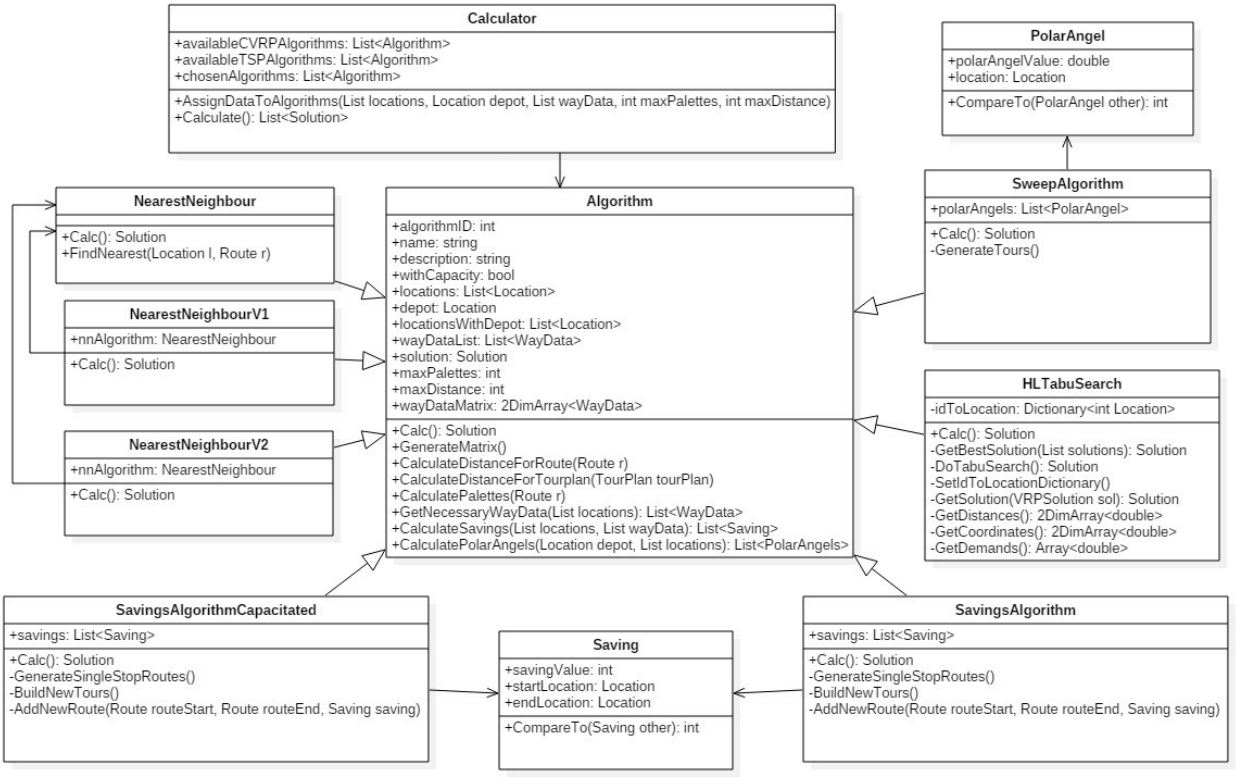

Abbildung 5.5: Klassendiagramm der Algorithmen-Bibliothek

Algorithm ist eine abstrakte Klasse und enthält die abstrakte Methode *Calc*. Die Klassen für die einzelnen Lösungsalgorithmen leiten von *Algorithm* ab und implementieren die entsprechende Logik des Verfahrens in der *Calc*-Methode, die ein Objekt der Klasse *Solution* zurückgibt.
Die Schnittstelle, über die andere Applikationen auf die Bibliothek zugreifen können, ist die Klasse *Calculator*. Sie enthält in den beiden Listen *availableCVRPAlgorithms* und *availableTSPAlgorithms* *Algorithm*-Objekte der verfügbaren Lösungsverfahren. In der Liste *chosenAlgorithms* kann die auf die Bibliothek zugreifende Applikation jene Algorithmen speichern, die sie auf ein bestimmtes Tourenplanungsproblem anwenden möchte. Um die Berechnung durchzuführen, müssen zuerst über die Methode *AssignDataToAlgorithms* die nötigen Daten für das Tourenplanungsproblem zugewiesen werden. Anschließend kann die Methode *Calculate* aufgerufen werden. Diese ruft für sämtliche *Algorithm*-Objekte, die sich in der Liste *chosenAlgorithms* befinden, die *Calc*-Methode auf und gibt am Ende eine Liste von Lösungen zurück. Beispielaufrufe dazu sind im Kapitel 6.2.1 zu finden.

Um die Unabhängigkeit und Wiederverwendbarkeit der Algorithmen-Bibliothek zu gewährleisten, wurden hier zur Repräsentation einer Lösung für ein Tourenplanungsproblem nicht die Klassen, die über das Entity Data Modell im WCF-Dienst erstellt wurden, verwendet, sondern eine eigene Klassenstruktur implementiert. Diese ist im UML Diagramm in Abbildung 5.6 dargestellt. Die Logik dahinter wurde bereits in Kapitel 5.1.12 beschrieben.

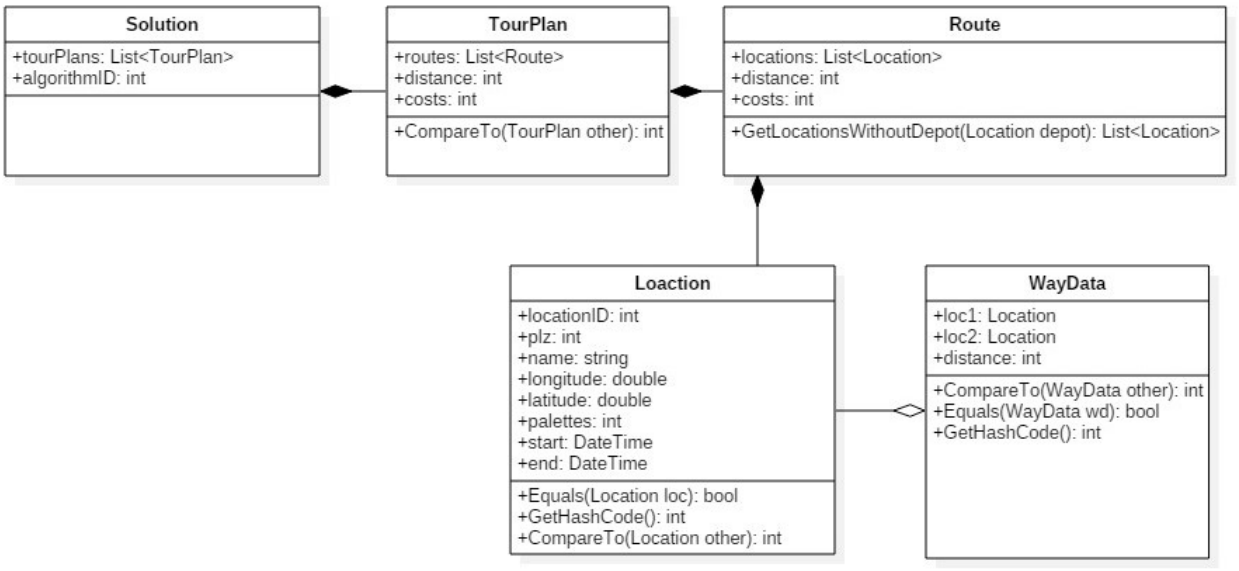

Abbildung 5.6: Klassendiagramm der Algorithmen-Bibliothek zur Speicherung von Lösungen

5.5 Spiel-Applikation

Wie der Szenariengenerator ist auch die Spiel-Applikation eine WPF Applikation. Sie erhält nötige Daten entweder von einer Datenbank oder über Dateien, die durch einen Szenariengenerator erstellt wurden. Abbildung 5.7 zeigt das Klassendiagramm dieser Anwendung.

5.5.1 GamePage

Die *GamePage*-Klasse ist der zentrale Teil des Tours-Spiels, denn sie ist für einen Großteil der Darstellung der Spielansicht verantwortlich. In einer Liste von *TransporterBoxControl*-Objekten, die User-Control-Elemente zur Darstellung und Verwaltung einzelner Touren sind, werden alle Touren-Boxen gespeichert und in einer Liste vom Typ *ObservableCollection* sind alle unverplanten Orte enthalten. Daneben gibt es noch eine Vielzahl weiterer Argumente, die für den reibungslosen Ablauf des Spiels notwendig sind und auf die viele der mit *GamePage* in Verbindung stehenden Klassen zugreifen. Besonders erwähnenswert ist in diesem Zusammenhang das Attribut *mode*, das festlegt, welcher Spielmodus ausgeführt wird. Dabei sind die Modi „Offlinespiel“ bzw. „Onlinespiel“ jeweils mit oder ohne bestehender Lösung möglich. Beim Starten des Spiels ist diese Klasse dafür verantwortlich, die für das Spielen benötigte Ansicht herzustellen, also beispielsweise auch das Laden einer eventuell bestehenden Lösung durch eine der in Abschnitt 5.5.2 beschriebenen *Controller*-Klassen anzustoßen.
Eine weitere wichtige Aufgabe der *GamePage*-Klasse ist es, auf alle Eingaben, die der Benutzer in der Spielansicht tätigt, zu reagieren. Dazu gehören beispielsweise das Umschalten zwischen der Planer- und Kartenansicht, das Ein- und Ausblenden der Minikarte, das Speichern der Lösung oder auch das Verlassen der Spielansicht durch Auswählen der Home-Schaltfläche.

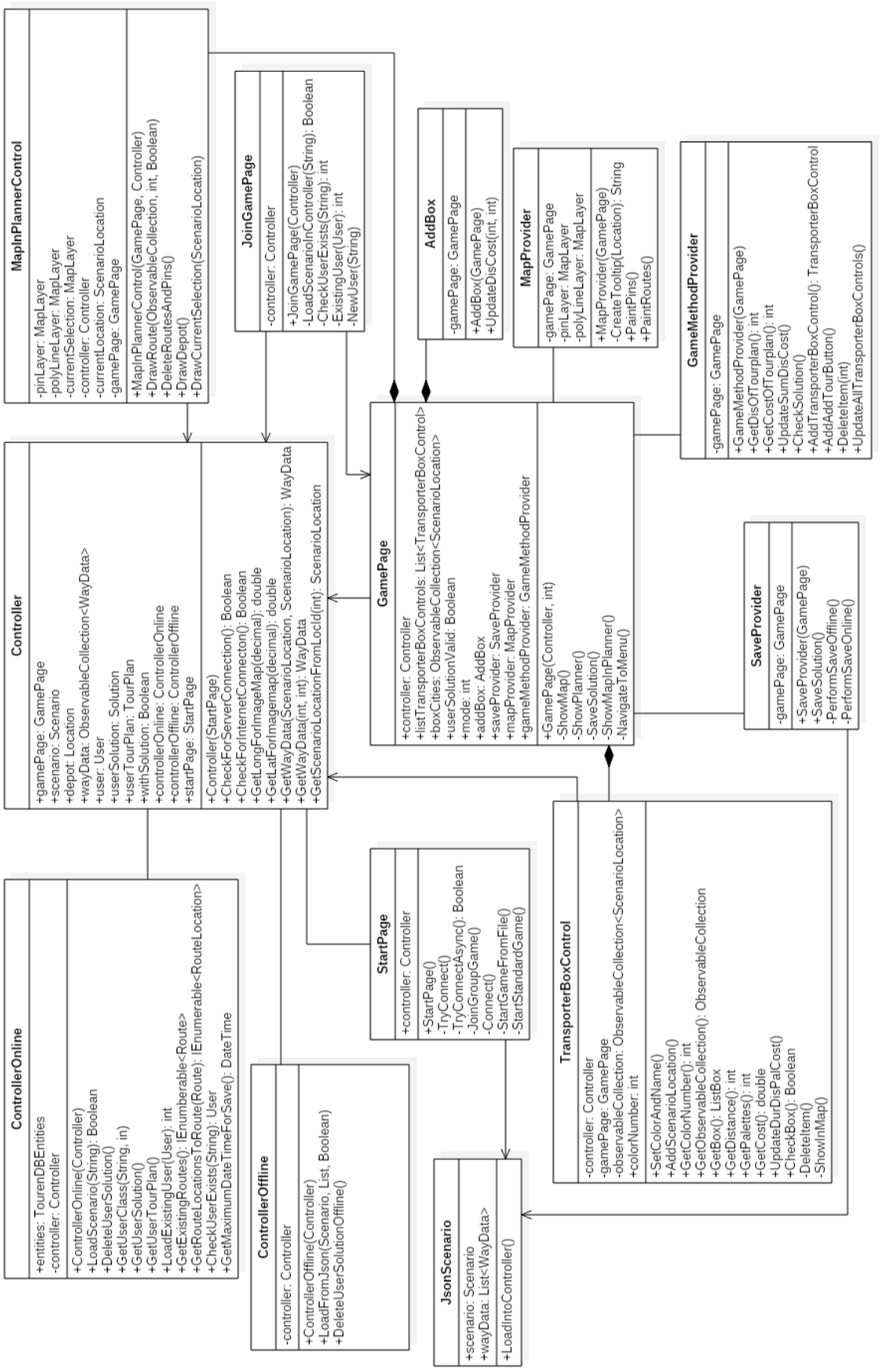

Abbildung 5.7: Klassendiagramm des Tours-Spiels

5.5.2 Controller-Klassen

Die drei Klassen *Controller*, *ControllerOnline* und *ControllerOffline* dienen zur Verwaltung des gesamten für ein Spiel nötigen Datenbestandes. Die *Controller*-Klasse bietet dabei grundlegende Funktionalitäten, die in jedem Fall benötigt werden und Funktionen zur Überprüfung der Internet- bzw. Serververbindung. Die Klasse *ControllerOnline* erweitert dies um Funktionen, die für ein Gruppen-Spiel benötigt werden. So wird hier beispielsweise eine Verbindung zum Server aufgebaut und es werden Daten von der Datenbank heruntergeladen. Auch die Verwaltung eventuell bereits bestehender Benutzerlösungen findet in dieser Klasse statt. Die Klasse *ControllerOffline* stellt diese Funktionen für ein Offline-Spiel zur Verfügung. So findet hier beispielsweise das Laden eines Szenarios und einer eventuell bestehenden Lösung aus den Daten der *JsonScenario*-Klasse, die die Schnittstelle zum Szenariengenerator ist, statt.

5.5.3 Provider-Klassen

Die drei Provider-Klassen *GameMethodProvider*, *MapProvider* und *SaveProvider* stellen eine Vielzahl an Funktionen zur Verfügung, die von der *GamePage*-Klasse verwendet werden.

GameMethodProvider

Die *GameMethodProvider*-Klasse bietet grundlegende Funktionen zum Verwalten geplanter Touren. Dies sind beispielsweise eine Berechnung der Länge der einzelnen Touren sowie eine Berechnung des Gesamtergebnisses des Tourenplans. Auch ein Hinzufügen einer neuen Tour beziehungsweise ein Löschen einer bestehenden oder ein Überprüfen einer Lösung auf seine Gültigkeit gehören zum Funktionsumfang dieser Klasse.

MapProvider

Die Klasse *MapProvider* dient der Verwaltung und Steuerung der großen Kartenansicht. Dafür bietet sie Funktionen wie ein Erstellen der Tooltip-Texte für die Orte mit einer Anzeige der Entfernungen zu den anderen im Szenario vorkommenden Orten, ein Einzeichnen der unverplanten und verplanten Standorte in der Karte, sowie ein Einzeichnen der geplanten Touren in einer Farbe, die der jeweiligen Tour entspricht.

SaveProvider

Die Klasse *SaveProvider* ist für jeden Speichervorgang verantwortlich. Dazu bietet sie sowohl Funktionen zur Speicherung von vollständigen sowie unvollständigen und ungültigen Ergebnissen in Dateien, als auch Funktionen für eine Speicherung von Ergebnissen in die Datenbank.

5.5.4 User-Controls

Die User-Control-Elemente *MapInPlannerControl*, *AddBox* sowie *TransporterBoxControl* werden in der Spielansicht verwendet und stellen wichtige Teile der Oberfläche bereit. Auch die Verwaltung dieser Oberflächen findet in der jeweiligen Klasse statt. So ist die Klasse *MapInPlannerControl* dafür verantwortlich, eine Minikarte in der Planersansicht darzustellen und zu verwalten. Sie bietet Funktionen zum Einzeichnen einzelner Routen und auch die Möglichkeit, einen aktuell ausgewählten Ort speziell hervorzuheben. Die Klasse *AddBox* stellt eine Schaltfläche zum Hinzufügen neuer Touren sowie eine Übersicht über die insgesamte Distanz und die Kosten der Planung dar. *TransporterBoxControl* dient dazu, die Boxen für einzelne Touren darzustellen und die damit verbundenen Möglichkeiten wie ein Zuordnen der Orte mittels Drag-and-Drop zu ermöglichen. Des Weiteren werden hier auch die Daten zu den einzelnen Touren, also die Distanz, Anzahl der Ladungsträger und Kosten angezeigt.

5.5.5 StartPage und JoinGamePage

Die beiden Klassen *StartPage* und *JoinGamePage* sind von der Klasse *System.Windows.Control.Page* abgeleitet und unter anderem dafür verantwortlich, das Hauptmenü darzustellen und es dem Benutzer zu ermöglichen, einem Gruppenspiel beizutreten. Dazu bietet *JoinGamePage* zum Beispiel eine Funktion an, die zum Beitritt zu einem Gruppenspiel einen neuen Benutzer in der Datenbank anlegt oder dafür sorgt, dass eine bestehende Lösung eines bereits vorhanden Benutzers geladen wird. Die Klasse *StartPage* ist auch dafür verantwortlich, die Daten zu einem Spiel aus einer Datei zu laden und ein Objekt der Klasse *JsonScenario* damit zu befüllen. Aus diesem Objekt können anschließend die notwendigen Attribute gewonnen und in der Klasse *Controller* gespeichert werden, was es der Klasse *ControllerOffline* ermöglicht, das entsprechende Spiel zu laden.

Kapitel 6

Implementierung

6.1 Datenbank

In diesem Abschnitt wird erläutert, wie der WCF-Dienst, mit dem auf die Datenbank zugegriffen wird, erstellt wurde und wie mit Client-Programmen darauf zugegriffen werden kann.

6.1.1 WCF-Dienst

Ein WCF-Dienst ermöglicht und steuert den Zugriff auf eine Datenbank. Um ihn zu erstellen und für Client-Applikationen verfügbar zu machen, sind in Visual Studio 2015 folgende Schritte notwendig:

- Erstellen einer neuen ASP.NET Webanwendung
- Hinzufügen eines ADO.NET Entity Data Modells zum Projekt. Dazu ist es notwendig, auszuwählen, woher die Daten stammen sollen. Kommen die Daten aus einer Datenbank, muss „EF Designer from database“ gewählt werden. Daraufhin müssen die Datenbank und die dazugehörigen Anmeldedaten angegeben werden.
- Um Daten für externe Anwendungen zugänglich zu machen, muss noch ein WCF Data Service Dienst zum Projekt hinzugefügt werden. Dieser erstellt die Service-Klasse, in der festgelegt werden kann, welche Rechte ein auf diesen Dienst zugreifender Benutzer hat. Weiters kann fortan mittels der HTTP Befehle GET, PUT, POST und DELETE auf die Daten zugegriffen werden.

Codebeispiel 6.1 zeigt in Zeile 1 die Einstellung der Zugriffsrechte auf die Tabelle *Location* und in Zeile 2 die Rechte für die Tabelle *Scenario*. Die Berechtigung *EntitySetRights.AllRead* bedeutet dabei, dass nur der Lesezugriff erlaubt ist. Bei *EntitySetRights.All* ist hingegen auch der schreibende Zugriff inklusive der Möglichkeit, bestehende Einträge zu ändern und zu löschen möglich.

```
1 config.SetEntitySetAccessRule("Location", EntitySetRights.AllRead);
2 config.SetEntitySetAccessRule("Scenario", EntitySetRights.All);
```

Listing 6.1: Rechteeinstellungen im WCF Service

6.1.2 Datenbank-Prozeduren

Beim Hinzufügen des Entity Data Modells zum WCF-Dienst auf Basis der Tabellen in der Datenbank, ist es zwar grundsätzlich möglich auch die Datenbankfunktionen und Prozeduren hinzuzufügen, jedoch gibt es beim Erstellen von Skalarfunktionen das Problem, dass der nötige Function Import im Entity Modell nicht erstellt werden kann. Es wird dazu der in Abbildung 6.1 gezeigte Fehler ausgegeben.

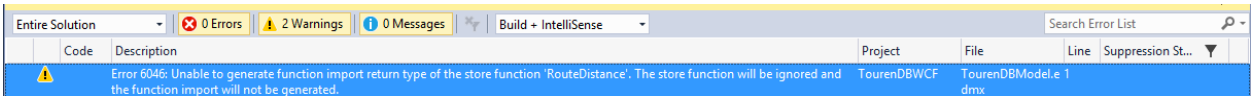

Abbildung 6.1: Fehler beim Einbinden von Skalarfunktionen aus der Datenbank

Des Weiteren ist es auch nicht möglich, den Function Import nachträglich noch hinzuzufügen, da die Datenbankfunktion im entsprechenden Dialog nicht ausgewählt werden kann. Aus diesem Grund muss das für das Entity Modell erstellte EDMX-File manuell verändert werden. Codebeispiel 6.2 zeigt die nötigen Veränderungen der entsprechenden Funktion. Es wurde hier das Attribut *Return-Type* entfernt, *IsComposable* auf *false* gesetzt und die Funktion im Element *CommandText* in Zeile 5 manuell per SQL-Abfrage aufgerufen.

```
1 <Function Name="RouteDistance" Aggregate="false" BuiltIn="false"
2         NiladicFunction="false" IsComposable="false"
3         ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
4                 <CommandText>
5                         SELECT [dbo].[RouteDistance] (@route_id)
6                 </CommandText>
7                 <Parameter Name="route_id" Type="int" Mode="In" />
8 </Function>
```

Listing 6.2: Function Import neu

Nach der Durchführung dieser Änderungen ist das Hinzufügen des Function Imports möglich. Damit über einen HTTP GET Aufruf auf die Datenbankfunktion zugegriffen werden kann, muss in der Service-Klasse des WCF-Dienstes noch die in Codebeispiel 6.3 gezeigte Methode hinzugefügt werden. Das Attribut *[WebGet]* macht die Methode dabei zu einer Service Operation. Dadurch kann der WCF-Dienst Zugriff auf sie über einen HTTP GET Aufruf bieten. In Zeile 4 der Methode wird die aktuelle Datenquelle gespeichert und von dieser dann in Zeile 5 die Datenbankfunktion *RouteDistance* aufgerufen, was entweder einen int-Wert oder *null* zurückliefert.

```
1 [WebGet]
2 public Nullable<int> GetRouteDistance(int routeId)
3 {
4         TourenDBEntities entities = this.CurrentDataSource;
5         return entities.RouteDistance(routeId).First<Nullable<int>>();
6 }
```

Listing 6.3: WCF Service Operation

Der Aufruf der Datenbankfunktionen erfolgt im Client-Programm, indem ein entsprechender HTTP GET Aufruf in ein Objekt des Typs *Uri* gespeichert und dieser dann mit der Methode *Execute* aufgerufen wird (siehe Codebeispiel 6.4).

```
public Nullable<int> GetRouteDistance(int routeId)
{
        Uri uri = new Uri(connectionString + "/GetRouteDistance?routeId="
                          + routeId);
        return this.Entities.Execute<Nullable<int>>(uri).First();
}
```

Listing 6.4: Stored Function Aufruf im Client

6.1.3 Zugriff auf die Datenbank

Sowohl der Szenariengenerator als auch die Spiel-Applikation müssen auf die Datenbank zugreifen, um bei einem Gruppen-Spiel Daten laden und abspeichern zu können. Codebeispiel 6.5 zeigt, wie hierfür die Verbindung zum WCF-Dienst, der Zugriff auf die Datenbank ermöglicht, hergestellt wird. Die dabei in Zeile 1 verwendete Klasse *TourenDBEntities* stammt aus der zuvor zum Projekt hinzugefügten Service-Referenz *ToursService* und dient dem Zugriff auf die einzelnen Tabellen der Datenbank. Die angegebene URI-Adresse führt zum entsprechenden WCF-Dienst, über den auf Daten aus der Datenbank zugegriffen werden kann.

```
public TourenDBEntities entities = new TourenDBEntities(new Uri(@"https://
   tourenda.htl-perg.ac.at/ToursWCF/ToursService.svc/"));
```

Listing 6.5: Zugriff auf den Service

Ein Anwendungsfall hierfür ist das Laden eines Gruppenspiels mit dem zugeordneten Szenario durch einen Benutzer der Spiel-Applikation. Dafür muss zuerst ein vom Spielleiter bekanntgegebener Spielcode eingegeben werden, woraufhin der Code aus Codebeispiel 6.6 ausgeführt wird.

```
//loading the game with this game code
controller.Game = (from g in this.Entities.Game
   where g.key.Equals(gameCode)
   select g).FirstOrDefault();
//checking if a game exists
if (controller.Game == null)
{
   return false;
}
//loading the scenario for the game
controller.Scenario = (from sc in this.Entities.Scenario.Expand
   ("ScenarioLocation/Location").Expand("Location")
   where sc.id == controller.Game.scenarioID
   select sc).FirstOrDefault();
```

Listing 6.6: Laden eines Spiels und des dazugehörigen Szenarios

Die LINQ-Abfrage in den Zeilen 1 bis 4 lädt das Spiel, bei dem das Attribut *key* mit dem angegebenen Spielcode übereinstimmt. Hier kann die Funktion *FirstOrDefault* verwendet werden, da bereits in der Datenbank durch einen eindeutigen Schlüssel festgelegt wird, dass das Attribut *key* in der Tabelle *Game* nicht mehrfach vorkommen darf. Wird kein passender Eintrag in der Datenbank gefunden, so gibt diese Abfrage *null* zurück.

Daraufhin wird in den Zeilen 6 bis 9 überprüft, ob die Abfrage ein Ergebnis geliefert hat. Ist dies nicht der Fall, existiert zu dem eingegebenen Spielcode kein Spiel und der Benutzer wird aufgefordert, einen anderen Code einzugeben. Ist jedoch ein passendes Spiel vorhanden, kann nun durch eine zweite LINQ-Abfrage, die im Codebeispiel in den Zeilen 11 bis 14 abgebildet ist, das dazugehörige Szenario geladen werden.
Dabei wird aus der Klasse *Scenario* jenes Szenario geladen, bei dem das Attribut *id* mit dem Attribut *scenarioID* des zuvor geladenen Spiels übereinstimmt. Auch hier kann wieder die Funktion *FirstOrDefault* verwendet werden, da jedem Spiel lediglich ein Szenario zugeordnet sein kann. Das Objekt *Scenario* enthält unter anderem Referenzen auf ein *Location*-Objekt, das das Depot des Szenarios repräsentiert und auf eine Liste mit *ScenarioLocation*-Objekten. Die *ScenarioLocation*-Objekte enthalten dabei wiederum eine Referenz auf ein *Location*-Objekt. Da die genannten referenzierten Objekte über das *Scenario*-Objekt direkt zugreifbar sein sollen, muss die Abfrage um den in Zeile 12 gezeigten *expand*-Teil erweitert werden. Durch den Einsatz des *expand*-Schlüsselwortes wird Eager Loading für eine Abfrage eingesetzt. Dabei werden neben der abgefragten Entität auch alle im *expand*-Teil angeführten, damit in Verbindung stehenden Entitäten geladen. Dadurch stehen die benötigten Daten von Anfang an zur Verfügung und es muss keine weitere Datenbankabfrage gestartet werden.
Zusätzlich zu dieser Art des Ladens von referenzierten Entitäten, gibt es noch die Möglichkeit, ein Objekt explizit um ein bestimmtes referenziertes Objekt zu erweitern. Ein Beispielaufruf dazu wird in Codeausschnitt 6.7 gezeigt.

```
1 this.Entities.LoadProperty(controller.Scenario, "ScenarioLocation/Location");
2 this.Entities.LoadProperty(controller.Scenario, "Location");
```

Listing 6.7: Explizites Erweitern des Szenarios um bestimmte referenzierte Entitäten

In Zeile 1 wird das Objekt *Scenario* explizit um die Daten aus dem Zugriffspfad */ScenarioLocation/Location* erweitert. Von nun hat man daher von einem *Scenario*-Objekt aus direkt auf die darin enthaltene Liste mit *ScenarioLocation*-Objekten Zugriff. Von diesen kann weiter auf die zugeordneten *Location*-Einträge zugegriffen werden. In Zeile 2 erfolgt der selbe Aufruf noch einmal, nur wird hier um die Daten aus dem Zugriffspfad */Location* erweitert, was bedeutet, dass fortan auch das Depot des Szenarios direkt aus einem *Scenario*-Objekt ausgelesen werden kann.

Des Weiteren ist es mit LINQ auch möglich, mehrere Einträge pro Abfrage zu selektieren. Dies wird in Codebeispiel 6.8 exemplarisch gezeigt.
Es werden dabei alle Routen der Klasse *Route* selektiert, bei denen der Eintrag *tourplanID* dem Attribut *id* des vom Benutzer erstellten Tourenplans enspricht. Wichtig ist, als Typ eine Collection-Klasse zu verwenden, da bei dieser Abfrage auch mehrere Elemente zurückkommen können.
In Zeile 1 des Beispiels wird festgelegt, aus welcher Quelle die zu selektierenden Daten stammen und in welcher Collection-Klasse sie gespeichert werden sollen. Zeile 2 enthält die *where*-Bedingung, die festlegt, dass nur jene Datenbankeinträge mit einer entsprechenden ID selektiert werden sollen. Der *select*-Befehl in Zeile 3 legt schließlich fest, was als Ergebnis zurückgeliefert werden soll.

```
1 IEnumerable<Route> routes = (from route in this.Entities.Route
2    where route.tourplanID == controller.UserTourPlan.id
3    select route);
```

Listing 6.8: Selektieren mehrerer Einträge

6.1.4 Speichern in die Datenbank

Codebeispiel 6.9 zeigt das Speichern einer Lösung zu einem Szenario in die Datenbank.

```
1  foreach (var r in gamePage.LTransporterBoxControls)
2  {
3     if (r.GetObservableCollection().Count != 0)
4     {
5        Route route = new Route();
6        route.tourplanID = gamePage.Controller.UserTourPlan.id;
7        route.TourPlan = gamePage.Controller.UserTourPlan;
8        gamePage.Controller.ControllerOnline.Entities.AddToRoute(route);
9        DataServiceResponse response = gamePage.Controller.ControllerOnline
10          .Entities.SaveChanges();
11       foreach (ChangeOperationResponse changedRoute in response)
12       {
13          EntityDescriptor descriptor = changedRoute.Descriptor
14             as EntityDescriptor;
15          if (descriptor != null)
16          {
17             Route addedRoute = descriptor.Entity as Route;
18             int counter = 0;
19             AppendDepot(addedRoute, counter);
20             counter++;
21             foreach (var routeLocation in r.GetObservableCollection())
22             {
23                AppendLocation(addedRoute, counter, routeLocation);
24                gamePage.Controller.ControllerOnline.Entities.AddToRouteLocation
25                   (routeLocation);
26                counter++;
27             }
28             AppendDepot(addedRoute, counter);
29             gamePage.Controller.ControllerOnline.Entities.SaveChanges();
30          }
31       }
32    }
33 }
```

Listing 6.9: Speichern von Daten in der Datenbank

In der ersten Schleife wird die Liste *LTransporterBoxControls*, welche die einzelnen Tourenboxen enthält, durchlaufen. Die in Zeile 3 verwendete Methode *GetObservableCollection* liefert eine Liste vom Typ *ObservableColletion* mit jenen Orten, die dieser Tour zugeordnet sind. Sind keine Orte in der Liste vorhanden, bedeutet dies, dass die Tour vom Benutzer zwar angelegt wurde, er aber keine Orte in die entsprechende Tourenbox gezogen hat. In diesem Fall muss diese leere Tour auch nicht gespeichert werden. Ist jedoch mindestens ein Ort vorhanden, wird zuerst ein neues Objekt der Klasse *Route* angelegt. Diesem wird nun der durch den Benutzer erstellte Tourenplan sowie dessen *tourplanID* zugeordnet. Mit dem Aufruf von „Entities.AddToRoute(route)“ und „Entities.SaveChanges()“ in den Zeilen 8 und 10 wird die neue Route der Datenbank hinzugefügt. Als Ergebnis des Speicher-Befehls wird ein *DataServiceResponse*-Objekt geliefert, das

alle Änderungen seit dem letzten Speichervorgang enthält. Daher muss es in einer Schleife durchlaufen werden. Nach der Prüfung, ob ein *EntityDescriptor*-Objekt vorhanden ist, kann die durch ihn bereitgestellte Route verwendet werden (Zeile 13-17). Zuerst wird ihr in Zeile 19 das Depot als Startort hinzugefügt und danach wird die Liste von *RouteLocation*-Objekten durchlaufen. Nach dem Hinzufügen aller *Location*-Objekte durch den wiederholten Aufruf der Methode *AppendLocation* und das Speichern der Daten durch den Aufruf in Zeile 24, wird der Route zuletzt noch das Depot als Zielort hinzugefügt. Die erstellten *RouteLocation*-Objekte werden durch den Befehl *SaveChanges* in Zeile 29 in der Datenbank gespeichert.

6.2 Algorithmen-Bibliothek

In diesem Abschnitt werden ausgewählte Implementierungsdetails der Algorithmen-Bibliothek und die Implementierung der Schnittstelle zwischen Szenariengenerator und Algorithmen-Bibliothek gezeigt.

6.2.1 Schnittstelle Szenariengenerator - Algorithmen-Bibliothek

Im Szenariengenerator wird in der Klasse *NewScenarioController* ein neues Objekt des Typs *Calculator* aus der Algorithmen-Bibliothek angelegt. Dieses Objekt wird als Schnittstelle zur Algorithmen-Bibliothek verwendet.

Wird ein Objekt der Klasse *Calculator* angelegt, so befüllt es seine Listen *AvailableTSPAlgorithms* und *AvailableCVRPAlgorithms* mit Objekten jener Klassen, die von *AlgorithmAPI.Algorithm* ableiten. Die Methode dazu ist in Codebeispiel 6.10 dargestellt. Mit dem Aufruf *Assembly.GetAssembly(typeof(Algorithm)).GetTypes()* werden sämtliche Bestandteile der Algorithmen-Bibliothek zurückgegeben. Von diesen werden mit dem LINQ-Schlüsselwort *Where* nur jene selektiert, die eine nicht abstrakte Klasse sind und von *Algorithm* ableiten. In der *foreach*-Schleife wird in Zeile 7 ein Objekt des aktuellen Typs erzeugt. Danach wird es, je nach Art des Algorithmus, in die Liste *AvailableCVRPAlgorithms* oder *AvailableTSPAlgorithms* gespeichert.

```
1  public void GetAvailableAlgorithms()
2  {
3      foreach (Type type in Assembly.GetAssembly(typeof(Algorithm)).GetTypes()
4              .Where(myType => myType.IsClass && !myType.IsAbstract &&
5               myType.IsSubclassOf(typeof(Algorithm))))
6      {
7        Algorithm alg = (Algorithm)Activator.CreateInstance(type);
8        if (alg.WithCapcity)
9          this.AvailableCVRPAlgorithms.Add(alg);
10       else
11         this.AvailableTSPAlgorithms.Add(alg);
12     }
13 }
```

Listing 6.10: Verfügbare Algorithmen anlegen

Damit für ein im Szenariengenerator erstelltes Szenario Lösungen berechnet werden können, müssen zur Liste *ChosenAlgorithms* des *Calculator*-Objekts die gewünschten *Algorihtm*-Objekte hinzugefügt werden.
In Codebeispiel 6.11 wird gezeigt, wie die anschließende Berechnung der Lösungen erfolgt.

Zunächst werden in Zeile 1 bis 11 die nötigen Daten zum Tourenplanungsproblem an die Algorithmen-Bibliothek über die Methode *AssignDataToAlgorithms* übergeben. Dabei handelt es sich um die zu verplanenden Orte, das Depot, die nötigen Entfernungen zwischen den Orten sowie die maximale Kapazität und Distanz pro Route. Bevor die Daten übergeben werden können, müssen sie in die entsprechenden Klassen der Algorithmen-Bibliothek umgewandelt werden, was mithilfe der Methoden in der Klasse *AlgorithmAPIClassesToDBClasses* geschieht. Anschließend erfolgt in Zeile 13 bis 14 die Berechnung der Algorithmen über die Methode *Calculate* des *Calculator*-Objekts. Diese Methode ruft für jeden Algorithmus der Liste *ChosenAlgorithms* die *Calc*-Methode auf und gibt eine Liste von Lösungen zurück.
Am Ende müssen diese Lösungen dann noch über die Klasse *AlgorithmAPIClassesToDBClasses* in die entsprechenden Klassen aus dem WCF-Dienst umgewandelt werden, damit sie schließlich in Zeile 15 dem Szenario-Objekt zugewiesen werden können.

```
1  this.NewScenarioController.Calculator.AssignDataToAlgorithms(
2    this.Controller.AlgorithmAPIClassesToDBClasses.DBLocationsToAPILocations(
3      this.NewScenarioController.Scenario.ScenarioLocation.ToList()
4    ),
5    depot,
6    this.NewScenarioController.FindWayDataObjects(
7      this.NewScenarioController.Scenario
8    ),
9    (int)this.NewScenarioController.Scenario.capacity,
10   (int)this.NewScenarioController.Scenario.maxDuration
11 );
12
13 List<AlgorithmAPI.Solution> solutions =
14        this.NewScenarioController.Calculator.Calculate();
15 this.NewScenarioController.Scenario =
16        this.Controller.AlgorithmAPIClassesToDBClasses.APISolutionToDBSolution(
17               solutions, this.NewScenarioController.Scenario
18        );
```

Listing 6.11: Aufruf der Berechnung durch den Szenariengenerator

6.2.2 Einbinden der Tabu Search aus dem HeuristicLab

In diesem Abschnitt wird erklärt, wie die Tabu Search in die Algorithmen-Bibliothek eingebunden und verwendet wird. Die drei wichtigsten Klassen aus HeuristicLab sind dabei *CVRPTWData*, *VehicleRoutingProblem* und *TabuSearch*.

CVRPTWData
CVRPTW ist die Abkürzung für „Capacitated Vehicle Routing Problem with Time Windows“. In einem Objekt dieser Klasse können die nötigen Daten zu einem kapazitierten Tourenplanungsproblem mit Zeitfenstern gespeichert werden. Diese Klasse erbt von der Klasse *CVRPData*, welche wiederum von *VRPData* erbt. *VRPData* implementiert schließlich das Interface *IVRPData*.

Der Name der Klasse ist hier etwas verwirrend, da in Kapitel 2.1.2 geschrieben wurde, dass im Rahmen dieser Arbeit Planungsprobleme mit Zeitfenstern nicht behandelt werden. Der Grund, warum diese Klasse und nicht einfach die Super-Klasse *CVRPData* verwendet wird ist, dass

mit *CVRPData* nur die Restriktion der Ladungsträger abgebildet werden könnte. Die Klasse *CVRPTWData* hingegen ermöglicht es, auch die Distanzrestriktion umzusetzen, indem das Attribut *DueTimes*, wie in der nachfolgenden Auflistung aller Attribute beschrieben ist, gesetzt wird.

Für ein Objekt der Klasse *CVRPTWData* müssen folgende Attribute festgelegt werden:

- *int Dimensions*: gibt die Zahl der zu beliefernden Kunden plus dem Depot an
- *double[,] Coordinates*: Hier wird ein zweidimensionales Array gespeichert, das aus zwei Spalten und aus einer Reihe für das Depot sowie Reihen für jeden weiteren Ort, besteht. In die erste Spalte kommen die Breitengrade und in die zweite die Längengrade. Wichtig ist hier, dass in der ersten Zeile des Arrays die Koordinaten des Depots stehen. Danach kommen die Koordinaten der Orte in einer festgelegten Reihenfolge. Diese Reihenfolge muss auch beim Setzen der Attribute *Distances* und *Demands* eingehalten werden.
- *double[,] Distances*: In diesem Attribut wird die Distanzmatrix zum Tourenplanungsproblem gespeichert.
- *double[] Demands*: Hier werden die benötigten Ladungsträger der Orte angegeben, wobei der erste Eintrag, also der für das Depot, den Wert null haben muss.
- *double Capacity*: die maximale Anzahl an Ladungsträgern pro Tour
- *double MaximumVehicles*: Gibt es keine Beschränkung der Fahrzeuge, so kann hier einfach die Anzahl der zu beliefernden Orte angegeben werden.
- *double[] DueTimes*: Im Fall dieser Diplomarbeit werden die Werte dieser Liste für jeden Ort auf die im Tourenplanungsproblem festgelegte maximale Distanz gesetzt. Dadurch wird die Restriktion bezüglich der maximalen Kilometer pro Route festgelegt. Auch wenn der Name „Times“ eigentlich aussagt, dass hier Zeiteinheiten gespeichert werden, so ist dies im Grunde nicht zwingend erforderlich. Das einzig Wichtige hier ist, dass die Einheit der Werte in dieser Liste mit der Einheit der Werte in der *Distances*-Matrix übereinstimmt.
- *double[] ServiceTimes*: Mit diesem Array kann für jeden Ort die erforderliche Bedienzeit festgelegt werden. Da dies aber in dieser Arbeit nicht erforderlich ist, werden die Bedienzeiten für jeden Ort auf 0 gesetzt.
- *double[] ReadyTimes*: Mit diesem Array kann für jeden Ort eine Zeit festgelegt werden, ab der er bedient werden kann. Auch dieses Attribut wird in dieser Arbeit nicht benötigt und daher für jeden Ort auf 0 gesetzt.

Die Verwendung der Attribute *DueTimes*, *ServiceTimes* und *ReadyTimes* für die Umsetzung von Zeitfenstern kann man sich am besten anhand eines Zeitstrahls vorstellen. Im folgenden Beispiel wird angenommen, dass Ort 1 von 8 bis 10 Uhr, Ort 2 von 9 bis 10 Uhr und das Depot von 8 bis 17 Uhr angefahren werden können. Für jeden Ort ist außerdem eine Bedienzeit von 0,2 Stunden zu berücksichtigen. Da die minimal vorkommende Uhrzeit bei 8 Uhr liegt, wird dieser Zeitpunkt als Ursprung des Zahlenstrahls angenommen. Um die beschriebenen Zeitfenster umzusetzen, müssten die Attribute folgendermaßen gesetzt werden:

- Depot: ReadyTime=0, ServiceTime=0, DueTime=9 (die Stunden zwischen 8 und 17 Uhr)
- Ort 1: ReadyTime=0, ServiceTime=0.2, DueTime=2

- Ort 2: ReadyTime=1 (da der Ort erst um 9 Uhr, also eine Stunde nach dem Startzeitpunkt, angefahren werden kann), ServiceTime=0.2, DueTime=1

Die Einheit für diese Angaben sind Stunden, daher müsste in diesem Fall auch für die Einheit der Distanzen Stunden gewählt werden.
Im Fall dieser Arbeit wird als Einheit für diese Attribute jedoch nicht Stunden sondern Kilometer gewählt. Durch das Setzen der Attribute *ReadyTimes* auf 0 und *DueTimes* auf die maximale Kilometerzahl, wird erreicht, dass pro Tour maximal diese Distanz zurückgelegt werden kann. In Abbildung 6.2 ist dazu ein Beispiel abgebildet, in dem zwei Orte beliefert werden müssen und die maximale Distanz pro Tour bei 300 Kilometern liegt. Die Fahrt vom Depot zu Ort 1 nimmt 120 Kilometer in Anspruch, diese Fahrt ist also innerhalb der festgelegten 300 Kilometer möglich. Nun wird Ort 2 überprüft. Diesen Ort würde der LKW zum „Zeitpunkt“ 310 Kilometer erreichen. Da jedoch das Attribut *DueTimes* für diesen Ort auf 300 Kilometer gesetzt ist, kann der LKW den Ort2 nicht mehr „rechtzeitig“ anfahren und die Route muss daher abgebrochen werden.

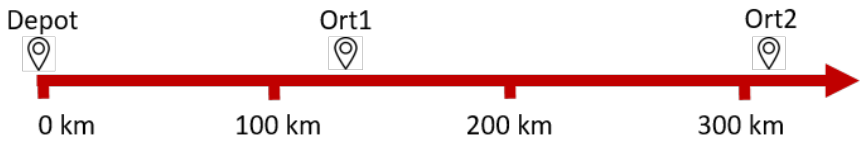

Abbildung 6.2: Verwendung der Zeitfenster-Attribute

VehicleRoutingProblem
Mit dieser Klasse wird jenes Optimierungsproblem repräsentiert, das später mit der Tabu Search gelöst werden soll. Dazu muss vorher noch das Attribut *ProblemInstance* mit den zuvor festgelegten Daten zum Tourenplanungsproblem befüllt werden. Dazu wird die Methode *Load(IVRPData data)* verwendet. Optional kann man noch die Attribute *BestKnownSolution* des Typs *VRPSolution* und *BestKnownQuality* setzen [veh].

TabuSearch
Mit dieser Klasse kann das mit einem Objekt des Typs *VehicleRoutingProblem* definierte Planungsproblem gelöst werden. Der Aufruf wird dabei in Codebeispiel 6.12 gezeigt.

```
1  var maxIterations = 100;
2  var finished = false;
3  do
4  {
5    var ts = new TabuSearch();
6    ts.Problem = vrpProblem;
7    ts.MaximumIterations.Value = maxIterations;
8    ts.MoveGenerator = ts.MoveGeneratorParameter.ValidValues.Single(x =>
9           x is PotvinCustomerRelocationExhaustiveMoveGenerator);
10   ts.Analyzer.Operators.SetItemCheckedState(ts.Analyzer.Operators.Single(x =>
11          x is CapacityRelaxationVRPAnalyzer), true);
12   ts.Engine = new ParallelEngine();
13   ts.Engine.Prepare();
14   ts.Prepare(true);
```

```
15   ts.Start();
16
17   IResult solutionResult;
18   finished = ts.Results.TryGetValue("Best valid VRPSolution",
19           out solutionResult);
20   if (!finished)
21   {
22     maxIterations *= 2;
23     continue;
24   }
25   var solution = (VRPSolution)solutionResult.Value;
26
27   Console.WriteLine("Tours:");
28   Console.WriteLine("{0}", string.Join(Environment.NewLine,
29           solution.Solution.GetTours().Select(x =>
30               string.Join(", ", x.Stops))));
31
32   //method to change the VRP-Solution to a AlgorithmAPI-Solution
33   this.Solution = this.GetSolution(solution);
34
35 } while (!finished);
```

Listing 6.12: Tabu Search

Zunächst wird in Zeile 5 von Codebeispiel 6.12 ein neues Objekt der Klasse *TabuSearch* angelegt und diesem anschließend ein Problem des Typs *VehicleRoutingProblem* zugewiesen. Mit dem Attribut *MaximumIterations* werden die maximalen Iterationen des Tabu Search Verfahrens festgelegt. Über das Attribut *MoveGenerator* wählt der Tabu Search Algorithmus neue Lösungen aus der Nachbarschaft der derzeitigen Lösung aus. Mit dem *Analyzer*-Attribut werden diese Lösungen analysiert und bewertet. Das Attribut *Engine*, das in Zeile 12 ein Objekt der Klasse *ParallelEngine* zugewiesen bekommt, ist für die Durchführung des eigentlichen Rechenprozesses zuständig.
Wie bereits in Kapitel 2 erklärt wurde, gibt es beim Tabu Search Algorithmus die sogenannte Tabuliste, in der die Veränderungen der Lösung gespeichert werden. Mit dem Attribut *TabuTenure* des *TabuSearch*-Objekts kann die Länge dieser Tabuliste festgelegt werden. Standardmäßig liegt sie hier bei einer Länge von 10.
Mit der Methode *Start* wird in Zeile 15 die Berechnung gestartet. In Zeile 18 wird im Attribut *Results* des *TabuSearch*-Objekts versucht, die beste gültige VRPLösung auszulesen. Ist keine gültige Lösung vorhanden, wird die Variable *finished* auf *false* gesetzt, was dazu führt, dass die Anzahl an maximalen Iterationen in Zeile 22 verdoppelt und von neuem gestartet wird. Ist eine gültige Lösung vorhanden, wird diese in das Objekt *solutionResult* gespeichert. In Zeile 25 erfolgt eine Umwandlung dieses Objekts in ein *VRPSolution*-Objekt.
Die *VRPSolution*-Klasse enthält ein Objekt der Klasse *PotvinEncoding*, in der es die Methode *GetTours()* gibt. In einer solchen Tour befindet sich eine Liste mit Stops vom Typ *int*. Bei dem Stop mit der Zahl 1 handelt es sich dabei um jenen Ort, den man beim Definieren der Daten in den Listen *Demands*, *Coordinates* und *Distances* an die erste Stelle nach dem Depot gesetzt hat. Um die Zuordnung dieser int-Werte zu den *Location*-Objekten der Algorithmen-Bibliothek zu ermöglichen, wurde zu Beginn ein Dictionary-Objekt mit einem Schlüssel des Typs *int* und einem Wert des Typs *Location* definiert [tab].

6.3 Bing Maps

Durch den Einsatz von Bing Maps können in der Spiel-Applikation und im Szenariengenerator Orte in einer Kartenansicht dargestellt werden. Um Bing Maps in den Anwendungen verwenden zu können, muss zuerst ein Bing Maps Entwickleraccount eröffnet und eine Lizenz erworben werden. Da es sich bei dieser Arbeit um eine schulische Tätigkeit handelt und auch die erstellte Software lediglich für Bildungszwecke eingesetzt wird, kann in diesem Fall auf eine kostenlose Bildungslizenz zurückgegriffen werden. Diese ermöglicht bis zu 50.000 Transaktionen innerhalb von jeweils 24 Stunden. Nach Erhalt der Lizenz muss noch das Karten-UI-Element *Bing Maps Windows Presentation Foundation Control* aus dem Microsoft Download Center heruntergeladen und installiert werden. Nach der Installation ist es möglich, Maps-Elemente direkt in Visual Studio zu verwenden. Unter Angabe des erhaltenen Lizenzschlüssels ist es weiters möglich, auf alle Bing Maps REST Dienste zuzugreifen und Berechnungen durchzuführen.

6.3.1 Kartendarstellung mit Bing Maps

Um eine unabhängige Darstellung eingezeichneter Orte und Routen zu ermöglichen, werden für die Map-Elemente zwei unterschiedliche *MapLayer*-Objekte verwendet, deren Definition in Codebeispiel 6.13 gezeigt wird. In den Zeilen 1 und 2 werden die beiden *MapLayer*-Objekte angelegt, wobei eines dafür dient, die Orts-Pins und das andere die Routen einzuzeichnen. In den Zeilen 3 und 4 werden die beiden *MapLayer*-Objekte zum Maps-Element hinzugefügt.

```
1 MapLayer pinLayer = new MapLayer();
2 MapLayer polyLineLayer = new MapLayer();
3 gamePage.MyMap.Children.Add(pinLayer);
4 gamePage.MyMap.Children.Add(polyLineLayer);
```

Listing 6.13: Speichern von Daten in der Datenbank

Codebeispiel 6.14 zeigt, wie bereits verplante Orte in der Kartenansicht eingezeichnet werden. Zuerst wird dabei in den Zeilen 3 und 4 aus einem *TransporterBoxControl*-Objekt die Liste an einzuzeichnenden Orten gewonnen. Die Klasse *TourColor* dient dazu, aus der im *TransporterBoxControl*-Objekt gespeicherten Farbnummer die entsprechende Farbe zu erhalten, in der die Orte eingezeichnet werden sollen.
Dafür wird die vorher erhaltene Liste mit Orten ab Zeile 7 durchlaufen und für jeden vorkommenden Ort ein *Microsoft.Maps.MapControl.WPF.Location*-Objekt angelegt. Dazu werden im Konstruktor Längen- und Breitengrad des jeweiligen Ortes gesetzt. Für jeden der so definierten Orte wird anschließend ein Pushpin-Objekt angelegt, das die Markierung auf der Karte in der entsprechenden Farbe einzeichnet. Weiters wird durch die Methode *createTooltip* jener Text erzeugt, der angezeigt wird, wenn der Benutzer mit der Maus über einen solchen Orts-Pin fährt. Zuletzt wird die somit erzeugte fertige Pushpin-Markierung in Zeile 18 dem *pinLayer*-Objekt hinzugefügt und damit auf der Karte dargestellt.

```
1 foreach (var transporterBox in gamePage.LTransporterBoxControls)
2 {
3    var listBox = transporterBox.GetBox();
4    var listLocations = listBox.ItemsSource.Cast<ScenarioLocation>().ToList();
5    TourColor c = new TourColor(transporterBox.GetColorNumber());
6
7    foreach (var data in listLocations)
8    {
```

```
9       Microsoft.Maps.MapControl.WPF.Location pinLocation =
10         new Microsoft.Maps.MapControl.WPF.Location(
11            Convert.ToDouble(data.Location.latitude),
12            Convert.ToDouble(data.Location.longitude)
13         );
14      Pushpin pin = new Pushpin();
15      pin.Location = pinLocation;
16      pin.ToolTip = CreateTooltip(data.Location);
17      pin.Background = new SolidColorBrush(c.GetColor());
18      pinLayer.Children.Add(pin);
19   }
20 }
```

Listing 6.14: Markieren der verplanten Orte in der Kartenansicht

Ähnlich zum in Codebeispiel 6.14 gezeigten Einzeichnen von Orten erfolgt auch das Zeichnen geplanter Routen.

```
1  MapPolyline polyline = new MapPolyline();
2  TourColor c = new TourColor(transporterBox.GetColorNumber());
3  polyline.Stroke = new SolidColorBrush(c.GetColor());
4  polyline.StrokeThickness = 3;
5  polyline.Opacity = 0.95;
6
7  polyline.Locations = new LocationCollection();
8  foreach (var data in list)
9  {
10    polyline.Locations.Add(new Microsoft.Maps.MapControl.WPF.Location(Convert.
11       ToDouble(data.latitude), Convert.ToDouble(data.longitude)));
12 }
13 polyLineLayer.Children.Add(polyline);
```

Listing 6.15: Zeichnen der geplanten Routen in der Kartenansicht

Codebeispiel 6.15 zeigt, wie die Orte einer Tour auf der Karte verbunden werden. Dazu wird ein *MapPolyline*-Objekt verwendet, dem in Zeile 2 die gleiche Farbe zugewiesen wird wie den Orten der jeweiligen Tour. In den Zeilen 3 bis 5 werden Einstellungen vorgenommen, die das Aussehen der Verbindungslinie festlegen. Um die Eckpunkte der Linie festzulegen, wird ein Objekt vom Typ *Microsoft.Maps.MapControl.WPF.LocationCollection* verwendet. Dieser Liste werden in der Schleife von Zeile 8 bis 12 *Microsoft.Maps.MapControl.WPF.Location*-Objekte mit den Koordinaten der Orte aus dem *TransporterBoxControl*-Objekt hinzugefügt. Danach erfolgt in Zeile 13 das Zeichnen der Tour, indem das erstellte *PolyLine*-Objekt dem *polyLineLayer*-Objekt der Karte hinzugefügt wird.

6.3.2 Kartendarstellung bei fehlender Internetverbindung

Steht keine Internetverbindung zur Verfügung, kann nicht auf die Dienste von Bing Maps zugegriffen und daher auch keine Bing Maps Karte geladen werden. In diesem Fall wird auf ein Bild einer Kartenansicht zurückgegriffen und Orte sowie Verbindungen werden in einem darübergelegten Zeichenbereich eingezeichnet. Dazu muss aus den bekannten Koordinaten für

jeden Ort jeweils ein X- und ein Y-Wert zur Darstellung auf dem Bild berechnet werden. Diese Berechnung ist in Codebeispiel 6.16 ersichtlich.

```
1 public double GetLongForImageMap(decimal coordD)
2 {
3    double coord = Convert.ToDouble(coordD);
4    coord = coord - longitudeLeftTop;
5    coord = coord / longImageWidth; // longImageWidth = rightLong - leftLong
6    return imageWidthPx * coord;
7 }
```

Listing 6.16: Berechnung der Bildpunkte für die Offline-Kartenansicht

Die Methode in Codebeispiel 6.16 ist für die Berechnung eines am Bild einzuzeichnenden Längengrades zuständig. Dazu wird der entsprechende Längengrad an die Methode *GetLongForImageMap* übergeben. Um die weiteren Berechnungen durchführen zu können, muss dieser *decimal*-Wert in Zeile 3 zuerst in einen *double*-Wert konvertiert werden. In Zeile 4 wird von diesem Wert der Längengrad des linken Bildrandes subtrahiert, um die Differenz zwischen dem linken Rand des Bildes und dem gewünschten Punkt zu erhalten. Dieses Ergebnis wird in Zeile 5 wiederum durch die Breite des Bildes in Längengraden dividiert. Das Ergebnis dieser Division ist ein Prozentbetrag, um den vom linken Bildrand eingerückt werden muss, um den korrekten Längengrad einzuzeichnen. Durch die Multiplikation dieses Prozentbetrages mit der Breite des Bildes in Pixeln erhält man die Anzahl an vom linken Bildrand einzurückenden Pixeln.
Aufgrund der geringen flächenmäßigen Größe des auf der in dieser Kartenansicht abgebildeten Gebiets muss in der Berechnung der Punkte eine Verzerrung durch die Erdkrümmung nicht berücksichtigt werden. Bei größeren Kartenansichten kann diese jedoch zu merkbaren Verschiebungen führen und müsste deshalb einberechnet werden.

Codebeispiel 6.17 zeigt, wie in einer Offline-Karte eine Markierung für einen Ort erfolgt. Zuerst wird dabei in Zeile 1 eine kreisrunde Ellipse erstellt, die später den Ort markieren soll. Nach dem Hinzufügen des Hinweistextes in Zeile 4 und dem Einstellen der Farbe der Markierung in Zeile 5 erfolgen Berechnung des Breiten- und Längengrades in den Zeile 6 und 7 bzw. 8 und 9 aufgrund der Koordinaten des jeweiligen Ortes. Ergebnis dieser Berechnungen sind jeweils die Anzahl an Pixeln, die von oben beziehungsweise links eingerückt werden muss, um die Markierung richtig zu platzieren. Aufgrund des Durchmessers der verwendeten Ellipse müssen von beiden Ergebnissen noch jeweils 11 Pixel abgezogen werden, damit die Mitte der Markierung den Ort zeigt. Zuletzt wird die Ellipse in Zeile 10 noch dem über das Bild gelegten Zeichenbereich hinzugefügt.

```
1  Ellipse e = new Ellipse();
2  e.Height = 22;
3  e.Width = 22;
4  e.ToolTip = CreateTooltip(data.Location);
5  e.Fill = new SolidColorBrush(tourColor);
6  Canvas.SetTop(e, gamePage.Controller.GetLatForImgMap(data.Location.latitude)
7     - 11);
8  Canvas.SetLeft(e, gamePage.Controller.GetLongForImgMap(data.Location.longitude)
9     - 11);
10 gamePage.MyCanvas.Children.Add(e);
```

Listing 6.17: Zeichnen eines Ortes in der Offline-Kartenansicht

6.3.3 Orts- und Entfernungsdaten

Mit dem Bing Maps REST Dienst „Locations API“ können Daten zu Orten abgefragt werden (siehe Kapitel 3.4). In dieser Diplomarbeit wurde der Dienst genutzt, um die Breiten- und Längengrade zu Ortsadressen herauszufinden.
Der dazu nötige Aufruf im Programm wird in Codebeispiel 6.18 gezeigt. Die Methode *GetBingFormat*, die in Zeile 4 verwendet wird, gibt eine Zeichenkette bestehend aus Postleitzahl und Name des Ortes zurück und ersetzt dabei Leerzeichen durch „%20“. Die Methode *DoWebRequest* liefert zur festgelegten Adresse eine Antwort in Form einer JSON-Zeichenkette zurück. Die Methode *GetStatus* liest aus dieser Zeichenkette das Attribut *statusDescription* aus und gibt es zurück. Wenn dieser Status „OK“ lautet, werden die nötigen Ortsinformationen, also Längen- und Breitengrad, ausgelesen. Für das Arbeiten mit JSON-Objekten und JSON-Zeichenketten wird das *Nuget*-Paket *NewtonsoftJSON* eingesetzt. Mit diesem kann, wie in Zeile 10 zu sehen ist, eine Zeichenkette im JSON-Format in ein Objekt des Typs *JToken* umgewandelt werden. Um aus diesem *JToken*-Objekt den Längen- und Breitengrad zu erhalten, kann die Methode *SelectToken* unter Angabe des Pfads zum gewünschten Element verwendet werden.

```
1  public string GetLocationData (Location loc)
2  {
3    string address = @"https://dev.virtualearth.net/REST/v1/Locations?query="
4        + loc.GetBingFormat() +"&maxResults=1&key="+APIKey;
5    string responseJSON = DoWebRequest(address);
6    string status = this.GetStatus(responseJSON);
7
8    if (status.Equals("OK"))
9    {
10     JToken result = JToken.Parse(responseJSON);
11     loc.Latitude = Double.Parse(result.SelectToken(
12             "resourceSets[0].resources[0].point.coordinates[0]"
13         ).ToString());
14     loc.Longitude = Double.Parse(result.SelectToken(
15             "resourceSets[0].resources[0].point.coordinates[1]"
16         ).ToString());
17   }
18   return status;
19 }
20 public string GetStatus (string jSONData)
21 {
22   JToken result = JToken.Parse(jSONData);
23   string status = result.SelectToken("statusDescription").ToString();
24   return status;
25 }
```

Listing 6.18: Verwendung des Locations API Dienst

Für das Erstellen der Distanzmatrix zu den gewählten Orten wird der Bing Maps Dienst „Routes API“ verwendet (siehe Kapitel 3.4). Bei diesem Dienst können Start- und Zielorte angegeben werden, zwischen denen Entfernungen berechnet werden sollen.
Das Abfragen von Daten aus diesem REST Dienst funktioniert dabei ähnlich wie mit dem Dienst Locations API (siehe Codebeispiel 6.18).

Da die Anzahl der Anfragen, die mit den Bing Maps REST Diensten durchgeführt werden können, aufgrund der Lizenzbestimmungen beschränkt ist, ist es im Fall des Routes API Dienstes nicht möglich, die Distanzmatrix auf einmal für alle 100 Orte generieren zu lassen. Aus diesem Grund wird pro HTTP GET Abfrage immer nur ein Start- und ein Zielort übergeben und dies für sämtliche mögliche Kombinationen einzeln durchgeführt. Als *travelmode* wird der Modus *driving* festgelegt.

6.4 Dateiausgaben

Um eine Kommunikation zwischen Szenariengenerator und Spiel-Applikation auch offline möglich zu machen, können beide Anwendungen Dateien im JSON-Format speichern und einlesen. Diese Dateien dienen zum Austausch von Szenarien und zur Persistierung ersteller Lösungen. Weiters ist es möglich, mit dem Szenariengenerator ein erstelltes Szenario als Excel-Datei oder PNG-Bild zu exportieren. In diesem Abschnitt werden die technischen Hintergründe hinter diesen Möglichkeiten der Speicherung erklärt.

6.4.1 Speichern einer JSON Datei

In Codebeispiel 6.19 wird die Speicherung eines Szenarios mit allen verfügbaren Lösungen und WayData-Objekten in eine JSON-Datei gezeigt.
Zuerst wird durch den Code in den Zeilen 1 bis 4 ein *SaveFileDialog*-Fenster angezeigt, das es einem Benutzer ermöglicht, einen Speicherort und Namen für die zu erstellende Datei zu wählen. Als Dateiendung ist *.szenario* festgelegt. Hat der Benutzer Pfad und Name gewählt, wird in Zeile 7 ein neues Objekt von Typ *JsonScenario* angelegt, dem dann in Zeile 8 das Szenario mitsamt allen gespeicherten Lösungen und in Zeile 9 eine Liste von *WayData*-Objekten zugewiesen wird. Durch das Aufrufen von „File.WriteAllText“ unter Angabe des Dateinamens und der ins JSON-Format konvertierten Speicherklasse in Zeile 10 wird die Datei im zuvor gewählten Ordner gespeichert.

```
1  SaveFileDialog fileDialog = new SaveFileDialog();
2  fileDialog.Filter = "Szenario|*.szenario";
3  fileDialog.Title = "Szenario mit Lösung speichern";
4  fileDialog.ShowDialog();
5  if (fileDialog.FileName != "")
6  {
7     JsonScenario jsonScenario = new JsonScenario();
8     jsonScenario.Scenario = gamePage.Controller.Scenario;
9     jsonScenario.WayData = gamePage.Controller.WayData.ToList();
10    File.WriteAllText(fileDialog.FileName, JsonConvert
11       .SerializeObject(jsonScenario));
12 }
```

Listing 6.19: Speicherung in Datei

6.4.2 Excel-Dateiausgabe

Die Namensräume *Mirosoft.Office.Interop.Excel* und *Microsoft.Office.Core* enthalten Klassen und Interfaces, die den Zugriff auf Excel von C#-Programmen aus möglich machen. Um eine neue Excel-Datei zu erstellen, werden folgende Interfaces benötigt [msde]:

- *Microsoft.Office.Interop.Excel.Application*: Dieses Interface repräsentiert eine Excel Anwendung. Beispielsweise können mit einem Objekt dieses Typs anwendungsweite Einstellungen für Excel vorgenommen oder Daten wie die aktuelle Registerkarte oder die aktuell markierte Zelle zurückgegeben werden. Mit dem Attribut *Visible* kann entschieden werden, ob die gestartete Excel Anwendung für den Benutzer ein sichtbares Fenster anzeigen soll oder nicht (siehe Codebeispiel 6.20 Zeile 5,6).
- *Microsoft.Office.Interop.Excel.Workbook*: Dieses Interface repräsentiert eine Excel Arbeitsmappe. *Workbook*-Objekte können zu *Application*-Objekten hinzugefügt werden (siehe Codebeispiel 6.20 Zeile 8). Des Weiteren bietet das Interface die Methode *SaveAs*, mit der Arbeitsmappen gespeichert werden können (siehe Codebeispiel 6.20 Zeile 25-40).
- *Microsoft.Office.Interop.Excel.Worksheet*: Dieses Interface steht für eine Registerkarte in einer Arbeitsmappe. *Worksheet*-Objekte können zu Arbeitsmappen mit der Methode *Worksheets.Add* hinzugefügt werden (siehe Codebeispiel 6.20 Zeile 10). Mit einem *Worksheet*-Objekt können Bearbeitungen jeder Art in einer Registerkarte vorgenommen werden, wie beispielsweise das Hinzufügen von Text, Formen, Bildern, etc (siehe Codebeispiel 6.20 Zeile 12-21).

```
1  using Excel = Microsoft.Office.Interop.Excel;
2  using Office = Microsoft.Office.Core;
3  ...
4
5  var excelApp = new Excel.Application();
6  excelApp.Visible = false;
7
8  Excel.Workbook workbook = excelApp.Workbooks.Add();
9
10 Excel.Worksheet workSheet = (Excel.Worksheet)workbook.Worksheets.Add();
11 workSheet.Name = "Angabe";
12
13 //Adding a textbox
14 Excel.Shape textbox = workSheet.Shapes.AddTextbox(
15     Office.MsoTextOrientation.msoTextOrientationHorizontal, 0, 0, 500, 110
16 );
17 textbox.TextFrame.Characters(Type.Missing, Type.Missing).Text = text;
18
19 //Writing Text in a call
20 workSheet.Cells[10, "A"] = "Depot:";
21 workSheet.Cells[10, 2] = scenario.Location.plz + " " + scenario.Location.name;
22
23 workSheet.Columns[1].AutoFit();
24
25 workbook.SaveAs(
26     path + "\\" + scenario.name, //Filename
27     Excel.XlFileFormat.xlWorkbookDefault, //FileFormat
28     Type.Missing, //Password
29     Type.Missing, //WriteResPassword (if password incorrect, read-only-mode)
30     false, //ReadOnlyRecommended
```

```
31      false, //CreateBackup
32      Excel.XlSaveAsAccessMode.xlNoChange, //AccessMode (NoChange takes default)
33      Excel.XlSaveConflictResolution.xlUserResolution, /*ConflictResolution
34                                  (dialog box asks user to resolve conflict)*/
35      false, //AddToMru (when true: adds workbook to list of recently used files)
36      Type.Missing, //TextCodepage
37      Type.Missing, //TextVisualLayout
38      true /*Local (language to be used, true: Excel-language,
39                            false: language of the application)*/
40  );
41
42  workbook.Close();
```

Listing 6.20: Arbeiten mit Excel

6.4.3 Bildausgabe

Im Szenariengenerator ist es möglich, die zu einem Szenario erstellten Lösungen in ein Bild auszugeben. In diesem Bild wird das gesamte User-Control-Element *SolutionControl* mit Kartenansicht und Beschreibung der Routen in der Tabelle abgebildet. Um ein Bild dieses User-Control-Elements zu erzeugen, wird die Klasse *RenderTargetBitmap* verwendet. Mit dieser ist es möglich, Visual-Objekte in eine Bitmap umzuwandeln [msdf]. Diese Umwandlung ist in Codebeispiel 6.21 zu sehen.

```
1   public RenderTargetBitmap GetBitmap()
2   {
3     RenderTargetBitmap renderTargetBitmap = new RenderTargetBitmap(
4         Convert.ToInt32(SystemParameters.PrimaryScreenWidth), //pixelWidth
5         500, //pixelHeight
6         96, //dpiX
7         96, //dpiY
8         PixelFormats.Pbgra32 //PixelFormat
9     );
10    renderTargetBitmap.Render(this.printGrid);
11    return renderTargetBitmap;
12  }
```

Listing 6.21: RenderTargetBitmap-Klasse

Die Methode in Codebeispiel 6.21 wird für jede Lösung im Szenario aufgerufen. In der Methode erfolgt zunächst das Anlegen eines neuen *RenderTargetBitmap*-Objekts. Anschließend wird in Zeile 10 die Methode *Render* aufgerufen, mit der das *Visual*-Objekt *printGrid* gerendert wird. Am Ende gibt die Methode das erstellte *RenderTargetBitmap*-Objekt zurück. Diese dadurch entstandenen Objekte sollen dann in ein PNG-File abgespeichert werden können, wozu folgende Klassen verwendet werden:

- DrawingGroup: enthält eine Liste von *Drawing*-Objekten und fasst diese zusammen [msdb]
- DrawingVisual: wird zum Rendern von Formen, Bildern oder Texten verwendet [msdc]
- DrawingContext: Mit dieser Klasse können Zeichnungsinhalte zu einem *DrawingVisual*-Objekt hinzugefügt werden. Ein Objekt dieser Klasse wird beim Aufruf der Methode

RenderOpen eines *DrawingVisual*-Objekts zurückgegeben. Mit der Methode *DrawDrawing* des *DrawingContext*-Objekts kann das an die Methode mitübergebene *Drawing*-Objekt gezeichnet werden [msda].

In Codebeispiel 6.22 ist die Verwendung dieser Klassen dargestellt. In Zeile 1 wird zunächst ein neues *DrawingGroup*-Objekt erstellt, zu dem in der nachfolgenden *for*-Schleife die *RenderTargetBitmap*-Objekte der einzelnen *SolutionControl*-Elemente hinzugefügt werden. In Zeile 8 wird das *RenderTargetBitmap*-Objekt *combinedImg* erstellt, das letztendlich in die Bild-Datei gespeichert werden soll. Zuvor muss aber noch der nötige Inhalt in das Objekt *combinedImg* gezeichnet werden. Dies geschieht von Zeile 11 bis 14 mit den bereits beschriebenen Klassen *DrawingVisual* und *DrawingContext*. In Zeile 16 wird ein Objekt der Klasse *PngBitmapEncoder* erstellt, mit dem ein Objekt des Typs *RenderTargetBitmap* im PNG-Format codiert und schließlich gespeichert werden kann.

```
1  DrawingGroup drawingGroup = new DrawingGroup();
2  foreach(UIElement element in elementCollection)
3  {
4    SolutionControl sc = element as SolutionControl;
5    RenderTargetBitmap rtb = sc.GetBitmap();
6    drawingGroup.Children.Add(new ImageDrawing(rtb, new Rect(x,y,width,height));
7  }
8  RenderTargetBitmap combinedImg = new RenderTargetBitmap(
9      width, height, 96, 96, PixelFormats.Pbgra32
10 );
11 DrawingVisual drawingVisual = new DrawingVisual();
12 DrawingContext drawingContext = drawingVisual.RenderOpen();
13 drawingContext.DrawDrawing(drawingGroup);
14 combinedImg.Render(drawingVisual);
15
16 PngBitmapEncoder encoder = new PngBitmapEncoder();
17
18 encoder.Frames.Add(BitmapFrame.Create(combinedImg));
19 using (FileStream file = File.Create(path))
20 {
21   encoder.Save(file);
22 }
```

Listing 6.22: Ausgeben einer Bild-Datei

6.5 Szenarien automatisch generieren

Mit dem Szenariengenerator ist es möglich, Tourenplanungsprobleme automatisch generieren zu lassen. Bei einem Szenario ohne Kapazitätsrestriktionen ist dies nicht besonders kompliziert. Hierzu muss der Benutzer einfach die gewünschte Anzahl an Orten angeben und das Programm wählt anschließend zufällig *Location*-Objekte aus der Liste möglicher Orte aus.

Bei der Erstellung eines Planungsproblems mit Kapazitäten ist dies jedoch keine Option. Denn würde man die Orte hier ebenfalls einfach zufällig wählen, könnte es sein, dass einmal Orte gewählt werden, die alle sehr weit voneinander entfernt liegen und ein anderes Mal Orte, die

sich sehr nahe beieinander und nahe beim Depot befinden. Dadurch könnte für zwei Tourenplanungsprobleme mit den selben Rahmenparametern eine völlig unterschiedliche Anzahl an Touren als Lösung entstehen. Die Größeneinschränkung für das Szenario kann hier also nicht über die Ortsanzahl angegeben werden. Aus diesem Grund wird bei der Festlegung der Rahmenparameter ein Wert für die minimal zu entstehenden Touren verlangt.
In Codebeispiel 6.23 ist der Ablauf der zufälligen Szenarioerstellung erklärt. Dabei wird in der *while*-Schleife in Zeile 16 zunächst mit der Methode *GetRandomNumber* eine Zufallszahl erstellt. Mithilfe dieser wird anschließend aus der *Locations*-Liste des *Model*-Objekts jener Ort ausgewählt, dessen Listenindex der Zufallszahl entspricht. In Zeile 21 erfolgt die Berechnung, wie lange der Weg von diesem Ort bis zum Depot ist. Diese Strecke wird in Zeile 25 mit zwei multipliziert, was dann der Länge der Pendeltour zu diesem Ort entspricht, und zum Wert der Variable *currDist* addiert. Anschließend überprüft das Programm, ob die derzeitige Gesamtkilomterzahl, die in *currDist* gespeichert ist, noch kleiner als der Wert der in Zeile 4 berechneten Variable *distance* ist. Ist dem so, wird die in dieser Iteration generierte Zufallszahl zur Liste der bereits verwendeten Zufallszahlen hinzugefügt und eine neue Schleifeniteration begonnen. Ist die Bedingung nicht mehr erfüllt, bekommt die *boolean*-Variable *inDist* den Wert *false*, was dazu führt, dass die Schleife beendet wird.
In der Schleife in Zeile 36 werden die gewählten Orte dann zum Szenario hinzugefügt. Danach wird in Zeile 42 die Methode *CalculatePalettes* aufgerufen.
Diese Methode ist für die Zuweisung der Ladungsträger zu den zuvor gewählten Orten zuständig. Dabei wird folgende prozentuelle Verteilung für die Ladungsträger pro Ort angewandt:

- 10 % der Orte haben 5-10 % der maximalen Kapazität pro Tour
- 50 % der Orte haben 10-30 % der maximalen Kapazität pro Tour
- 30 % der Orte haben 30-50 % der maximalen Kapazität pro Tour
- 10 % der Orte haben 50-60 % der maximalen Kapazität pro Tour

```
1  Random random = new Random();
2  List<int> randoms = new List<int>(); //already used random numbers
3  //minimal number of Tours * maximum of kilometres per tour
4  int distance = this.NumTours * MaxDistance;
5  int currDist = 0;
6  int number = 0;
7  bool inDist = true;
8
9  ScenarioLocation depot = new ScenarioLocation();
10 depot.Location = scenario.Location;
11 depot.palettes = 0;
12
13 while (inDist)
14 {
15   //GetRandomNumber: returns a random number, that hasn't already been used
16   number = (GetRandomNumber(random, randoms, this.Model.Locations.Count,
17       scenario.Location));
18   ScenarioLocation sl = new ScenarioLocation();
19   sl.Location = this.Model.Locations[number];
20
```

```
21    WayData toDepot = Controller.GetWayDataObject(depot, sl);
22
23    if (toDepot != null)
24    {
25      currDist += toDepot.distance * 2;
26      if (currDist < distance)
27      {
28        randoms.Add(number);
29      }else
30      {
31        inDist = false;
32      }
33    }
34  }
35
36  for (int i = 0; i < randoms.Count; i++)
37  {
38    ScenarioLocation sl = new ScenarioLocation();
39    sl.Location = this.Model.Locations[randoms[i]];
40    scenario.ScenarioLocation.Add(sl);
41  }
42  CalculatePalettes(scenario.ScenarioLocation);
```

Listing 6.23: Automatisches Generieren eines Szenarios mit Kapazitäten

6.6 Implementierungdetails in WPF und XAML

In diesem Abschnitt werden interessante Details zum Umgang mit WPF-Applikationen vorgestellt.

6.6.1 Navigation

Beim Start einer WPF-Applikation wird zuerst die Datei *App.xaml* aufgerufen. Diese hat die Aufgabe, den Startvorgang der Applikation zu steuern. Dazu kann entweder mit dem Attribut *StartupUri* direkt das anzuzeigende *Window*-Element angegeben werden, oder es wird das Attribut *Startup* verwendet. Ist Letzteres der Fall, wird eine Methode aufgerufen, in der selbst definiert werden kann, wie der Startvorgang ablaufen soll. Des Weiteren kann die Datei *App.xaml.cs* dazu verwendet werden, globale Variablen zu definieren, da alle Klassen hierauf Zugriff haben.

Im Fall dieser Arbeit lädt die App-Datei ein Fenster des Typs *NavigationWindow*. In diesem kann über das Attribut *Source* festgelegt werden, welche Ansicht initial geladen und damit angezeigt werden soll.
Durch Setzen des Attributs *ShowsNavigationUI* auf *false* kann man weiters festlegen, dass die standardmäßig vorhandenen Schaltflächen „Vorwärts“ und „Zurück“ ausgeblendet werden.

Des Weiteren ist es durch die Implementierung der in Codebeispiel 6.24 gezeigten Methode möglich, einen Benutzer vor dem Schließen der Anwendung zu warnen, da möglicherweise nicht gespeicherte Daten verloren gehen könnten.

```
1  private void NavigationWindowClosing(object sender, System.ComponentModel.
2     CancelEventArgs e)
3  {
4     MessageBoxResult result = MessageBox.Show("Wollen Sie die Anwendung wirklich
5        beenden? Alle nicht gespeicherten Änderungen gehen verloren.",
6        "Beenden", MessageBoxButton.YesNo, MessageBoxImage.Question);
7     if(result == MessageBoxResult.Yes)
8     {
9        e.Cancel = false;
10    }
11    else
12    {
13       e.Cancel = true;
14    }
15 }
```

Listing 6.24: Meldung, wenn der Benutzer die Anwendung schließen möchte

Die Methode *NavigationWindowClosing* wird aufgerufen, sobald ein Benutzer auf die Schließen-Schaltfläche der Anwendung klickt. Durch den Aufruf in den Zeilen 4 bis 6 wird eine Warnmeldung angezeigt, bei der sich ein Benutzer noch einmal entscheiden kann, ob er die Anwendung wirklich schließen will. Entscheidet er sich dafür, mit dem Schließen fortzufahren, wird das *cancel*-Attribut des übergebenen Schließen-Befehls auf *false* gesetzt, ansonsten auf *true*, was den Schließvorgang abbricht.

Alle Sichten, die im Fenster vom Typ *NavigationWindow* angezeigt werden sollen, erben von der Klasse *Page*. Zu den einzelnen Sichten kann mit Hilfe des Attributs *NavigationService* der Klasse *Page* navigiert werden. Codebeispiel 6.25 zeigt einen solchen Navigations-Aufruf. Hier wird ein neues Objekt der Klasse *GamesOverwiew* erzeugt, zu dem anschließend navigiert wird.

```
1  this.NavigationService.Navigate(new GamesOverview());
```

Listing 6.25: Navigations-Aufruf

6.6.2 Asynchrone Bearbeitung zeitaufwendiger Aufgaben

Da beispielsweise eine Überprüfung der Internetverbindung oder das Laden der Daten zu einem Gruppen-Spiel längere Zeit in Anspruch nehmen kann, die Oberfläche währenddessen aber trotzdem bedienbar bleiben soll, muss in diesen Fällen auf eine asynchrone Bearbeitung der Aufgaben gesetzt werden. Asynchrone Vorgänge können mithilfe der Klasse *Task* gestartet werden. Diese ist dann für die zeitaufwendige Aufgabe zuständig, während der Hauptthread weiterhin aktiv bleibt und zum Beispiel auf Benutzereingaben reagieren kann. Codebeispiel 6.26 zeigt die Verwendung der *Task*-Klasse, um ein Spiel zu einem vom Benutzer eingegebenen Spielcode asynchron vom Server herunterzuladen.
Die Methode *ButtonCheckCodeClick* wird aufgerufen, wenn ein Benutzer auf die Schaltfläche „Code überprüfen“ der in Kapitel 4.2.1 beschriebenen Sicht klickt und lädt zu Beginn den vom Benutzer eingegebenen Spielcode aus der entsprechenden Textbox. Dann wird die auf der angezeigten Seite vorhandene Statusleiste auf „Indeterminate“ gesetzt, was bedeutet, dass sie, sobald sie im nächsten Schritt auf „Visible“ gesetzt wird, einen Vorgang unbekannter Dauer anzeigt.

Damit soll einem Benutzer mitgeteilt werden, dass das Programm eine Hintergrundaktivität ausführt und nicht abgestürzt ist. Des weiteren zeigt ein Textfeld diesen Status in Textform an. In Zeile 7 befindet sich der Aufruf, der eine neue Hintergrundaktivität anlegt, in der die Methode *LoadScenarioInController* ausgeführt wird. Das Codewort *await* führt dazu, dass die Ausführung des Programmcodes an dieser Stelle pausiert, bis von der angegebenen Methode ein Rückgabewert geliefert wird. Dies beudetet allerdings nicht, dass das Programm deshalb einfriert, so ist beispielsweise die Statusleiste weiterhin aktiv und es können Events ausgelöst werden, indem ein Benutzer zum Beispiel auf eine andere Schaltfläche klickt.
Die Methode *LoadScenarioInController*, die in der Hintergrundaktivität zur Ausführung kommt, hat die Aufgabe, das Laden des Spiels durch die entsprechende *Controller*-Klasse anzustoßen. Diese versucht nun, ein zum Spielcode passendes Spiel zu finden und zu laden. Ist kein passendes Spiel vorhanden, wird *false* zurückgeliefert, ansonsten *true*. Diese Variable wird auch von der Methode wieder zurückgegeben und der Hintergrundvorgang ist damit beendet. Nun kann auch die Ausführung der im Hauptthread wartenden Methode fortgesetzt werden.

```
1  private async void ButtonCheckCodeClick(object sender, RoutedEventArgs e)
2  {
3     String gameCode = gameCodeBox.Text;
4     progress_bar.IsIndeterminate = true;
5     progress_bar.Visibility = Visibility.Visible;
6     labelSuccess.Content = "Das Spiel wird überprüft und geladen ...";
7     Boolean ok = await Task.Run(() => LoadScenarioInController(gameCode));
8     progress_bar.Visibility = Visibility.Collapsed;
9
10    if (ok)
11    {
12       ...
13    }
14 }
15
16 private Boolean LoadScenarioInController(String gameCode)
17 {
18    Boolean ok = controller.ControllerOnline.LoadScenario(gameCode);
19    return ok;
20 }
```

Listing 6.26: Asynchrones Laden eines Spiels

6.6.3 Drag-and-Drop

Grundsätzlich kann eine Drag-and-Drop Methodik mit den standardmäßig von WPF bereitgestellten Möglichkeiten umgesetzt werden.
Codebeispiel 6.27 zeigt eine Methode, die dafür zuständig ist, das „Anheben“ eines Eintrags aus einem *ListBox*-Element zu ermöglichen. Sie wird aufgerufen, wenn ein Eintrag aus einem *ListBox*-Element mit der linken Maustaste ausgewählt wird und setzt das betroffene *ListBox*-Element als Quelle für den Drag-and-Drop-Vorgang. Durch die Methode *GetDataFromListBox*, die in Zeile 6 aufgerufen wird, kann jenes Element herausgefunden werden, das ausgewählt wurde und dementsprechend angehoben werden soll. Dafür wird der genaue Standort des Mauszeigers innerhalb des *ListBox*-Elements benötigt, der mit dem Aufruf von *e.GetPosition(source)* ermit-

telt wird. Um sicherzugehen, dass wirklich ein Element ausgewählt wurde und nicht in einen leeren Bereich des *ListBox*-Elements geklickt wurde, muss nun noch überprüft werden, ob ein Datensatz vorhanden ist (Zeile 7). Ist dies der Fall, kann der Drag-Vorgang durch den Befehl *DragDrop.DoDragDrop* gestartet werden. Durch Angabe des *DragDropEffects*-Attributs kann festgelegt werden, welche Effekte dem Benutzer beim Anheben und Ablegen eines Elements angezeigt werden. Einer dieser Effekte ist beispielsweise das Aussehen des Mauszeigers. Hierfür gibt es mehrere Möglichkeiten [Dra16]:

- Copy: Ein Element soll kopiert werden und ist danach mehrfach vorhanden.
- Move: Ein Element soll verschoben werden und ist danach nur noch am Ablageort zu finden.
- Scroll: Es kann ein Bildlauf durchgeführt werden, um den Ablageort eines Elements zu finden.
- Link: Am Ablageort wird ein Link zu einem Datensatz aus der Quelle eingefügt.
- None: Ein Element kann nicht abgelegt werden.
- All: kombiniert die Effekte aus *Copy*, *Move* und *Scroll*

```
1  ListBox dragSource = null;
2  private void ListBoxClick(object sender, MouseButtonEventArgs e)
3  {
4     ListBox source = (ListBox)sender;
5     dragSource = source;
6     object data = GetDataFromListBox(source, e.GetPosition(source));
7     if (data != null)
8     {
9        DragDrop.DoDragDrop(source, data, DragDropEffects.All);
10    }
11 }
```

Listing 6.27: Drag-and-Drop: Aufnehmen eines Elements [Pat10]

Um ein angehobenes Element am gewünschten Ort ablegen zu können, kann der Code aus Codebeispiel 6.28 eingesetzt werden.

```
1  private void ListTransporterDrop(object sender, DragEventArgs e)
2  {
3     ListBox dest= (ListBox)sender;
4     object data = e.Data.GetData(typeof(ScenarioLocation));
5     ObservableCollection<ScenarioLocation> switchScenarioLocation =
6        (ObservableCollection<ScenarioLocation>) dragSource.ItemsSource;
7     switchScenarioLocation.Remove((ScenarioLocation)data);
8     switchScenarioLocation = (ObservableCollection<ScenarioLocation>)dest
9        .ItemsSource;
10    switchScenarioLocation.Add((ScenarioLocation)data);
11 }
```

Listing 6.28: Drag-and-Drop: Ablegen eines Elements

Die in Codeausschnitt 6.28 gezeigte Methode *ListTransporterDrop* wird von einem *ListBox*-Element aufgerufen, wenn ein Element hineingezogen wird. Daher wird das aufrufende *ListBox*-Element in Zeile 3 als Ziel festgelegt und aus den mitgegebenen Argumenten kann der abzulegende Datensatz gewonnen werden. Um ein korrektes Bewegen eines Elements umzusetzen, muss es aus dem alten *ListBox*-Element gelöscht und zum neuen hinzugefügt werden. Dies geschieht, indem der betroffene *ScenarioLocation*-Eintrag in Zeile 7 mit *Remove* aus der Liste *ItemsSource* des Quell-ListBox-Elements entfernt wird. Anschließend wir in Zeile 9 aus dem Ziel-ListBox-Element das Attribut *ItemsSource* selektiert und diesem schließlich das bewegte *ScenarioLocation*-Objekt hinzugefügt.

Das in den Codebeispielen 6.27 und 6.28 beschriebene Vorgehen ermöglicht zwar die Umsetzung von Drag-and-Drop, bedeutet allerdings vor allem bei zusätzlichen gewünschten Funktionen einen erheblichen Programmieraufwand. So ist es aufwendig, mittels Drag-and-Drop eine Sortierung innerhalb eines *ListBox*-Elements vorzunehmen oder eine Markierung anzuzeigen, wo ein bestimmtes Element beim Loslassen der Maustaste eingefügt werden wird. Diese Probleme können umgangen werden, indem eine fertige Drag-and-Drop Bibliothek mit zahlreichen zusätzlichen Funktionen eingesetzt wird. In dieser Arbeit wird die Bibliothek „GongSolutions.WPF.DragDrop“ verwendet. Um nach dem Hinzufügen des NuGet-Pakets „gong-wpf-dragdrop“ die Funktionalität verwenden zu können, muss zur jeweiligen XAML-Datei der Namensraum „urn:gong-wpf-dragdrop“ hinzugefügt werden.

Um ein *ListBox*-Element so zu konfigurieren, dass sowohl Elemente davon weggezogen als auch Elemente mittels Drag-and-Drop hinzugefügt werden können, müssen die in Codebeispiel 6.29 gezeigten Attribute in der XAML-Datei zum betroffenen *ListBox*-Element hinzugefügt werden. Durch Einstellen des Attributs *IsDragSource* wird festgelegt, dass Einträge aus dem jeweiligen Element weggezogen werden können. *IsDropTarget* definiert, dass im Element Einträge aus anderen Elementen abgelegt werden können.

```
1 dd:DragDrop.IsDragSource="True" dd:DragDrop.IsDropTarget="True"
```

Listing 6.29: GongSolutions.WPF.DragDrop: Einstellungen in der XAML-Datei

Durch das Verwenden dieser Möglichkeit, Drag-and-Drop zu realisieren, ergeben sich einige zusätzliche Funktionen. So wird hier standardmäßig eine Markierung gesetzt, wo ein Element eingefügt werden wird. Weiters ist das Sortieren innerhalb eines *ListBox*-Elements durch Ziehen der Einträge möglich [gon].

6.6.4 Eigene Styles für WPF Controls

Will man beispielsweise das Aussehen eines Button-Elements anpassen, ist dies in beschränkter Form über die Angabe von Attributen direkt beim jeweiligen Button-Element möglich. Für umfassendere Anpassungen muss jedoch ein eigener Stil für das Element definiert werden.
Codebeispiel 6.30 zeigt den Stil, der für die Menüschaltflächen in der Spiel-Applikation definiert wurde. Dabei wird zu Beginn mit *TargetType* festgelegt, für welches Element der jeweilige Stil angewandt werden soll. Des Weiteren kann dem Stil hier ein Name gegeben werden, um ihn später den einzelnen Elementen zuweisen zu können. Die drei *Setter*-Abschnitte in den Zeilen 3, 4 und 5 definieren das Aussehen der Schaltfläche im Normalzustand. Durch das Festlegen des Attributs *CornerRadius* in Zeile 9 können abgerundete Ecken verwendet werden. Nach dem Einstellen des Aussehens im Normalzustand können Trigger definiert werden. Diese verändern das Aussehen des Elements, sobald ein bestimmtes Ereignis eintritt. Der in den Zeilen 17 bis 19

implementierte Trigger ändert die Hintergrundfarbe einer Schaltfläche, wenn ein Benutzer mit dem Mauszeiger darüber fährt. Der zweite Trigger, der in den Zeilen 20 bis 22 zu sehen ist, legt fest, wie sich das Aussehen einer derartigen Schaltfläche ändert, wenn sie deaktiviert wird.

```
1  <Page.Resources>
2     <Style TargetType="Button" x:Key="menuButton">
3        <Setter Property="Background" Value="#FF568F99"/>
4        <Setter Property="Foreground" Value="White"/>
5        <Setter Property="Template">
6           <Setter.Value>
7              <ControlTemplate TargetType="Button">
8                 <Border CornerRadius="15" Background=
9                    "{TemplateBinding Background}">
10                   <ContentPresenter HorizontalAlignment="Center"
11                      VerticalAlignment="Center"/>
12                </Border>
13             </ControlTemplate>
14          </Setter.Value>
15       </Setter>
16       <Style.Triggers>
17          <Trigger Property="IsMouseOver" Value="True">
18             <Setter Property="Background" Value="BurlyWood"/>
19          </Trigger>
20          <Trigger Property="IsEnabled" Value="False">
21             <Setter Property="Background" Value="LightGray"/>
22          </Trigger>
23       </Style.Triggers>
24    </Style>
25 </Page.Resources>
```

Listing 6.30: Eigens definierter Style für das WPF Button-Control

Um einem *Button*-Element den in Codebeispiel 6.30 definierten Stil zuzuweisen, muss der in Codebeispiel 6.31 gezeigte Code als Attribut zum Element hinzugefügt werden.

```
1  Style="{StaticResource menuButton}"
```

Listing 6.31: Hinzufügen eines Stils zu einem Element

6.6.5 Hinzufügen von User-Controls

Im Szenariengenerator werden für die Game-Ansicht, für die Szenario-Übersicht und die Übersicht bei der Erstellung von mehreren Szenarien eigene User-Control-Elemente verwendet, die in der entsprechenden Sicht in einem Grid-Element angeordnet sind (siehe Kapitel 4.1). Da die Anzahl der anzuzeigenden User-Control-Elemente von der Anzahl der Spiele, der Lösungen bzw. der Szenarien, abhängt, können das Grid-Element und die User-Control-Elemente nicht in der XAML-Datei definiert werden, sondern müssen im Code-Behind-Teil hinzugefügt werden. Als Beispiel dazu wird im Codeausschnitt 6.32 das Hinzufügen der SolutionControl-Elemente zur Szenario-Übersicht gezeigt. In Zeile 1 werden dabei zunächst sämtliche Reihen und damit auch die Inhalte des Grid-Elements entfernt. In der darauffolgenden *for*-Schleife wird für jede Lösung des Sze-

narios eine eigene Zeile im Grid-Element angelegt. Diese Grid-Zeilen werden in der nächsten *for*-Schleife mit einem User-Control-Element des Typs *SolutionControl* befüllt.

```
1  this.gridSolutions.RowDefinitions.Clear();
2
3  //adding a row to the grid for each solution
4  for (int i = 0; i < this.Controller.Scenario.Solution.Count; i++)
5  {
6    this.GridLocationSolutions.RowDefinitions.Add(new RowDefinition());
7  }
8  //filling the rows with the Solution-Controls
9  for (int i = 0; i < this.Controller.Scenario.Solution.Count; i++)
10 {
11   SolutionControl sc = new SolutionControl(
12       this.Controller.Scenario.Solution[i],
13       (double)(this.Controller.Scenario.costsPerRoute),
14       (double)(this.Controller.Scenario.costsPerKilometre)
15   );
16
17   this.GridLocationSolutions.Children.Add(sc);
18   Grid.SetRow(sc, i);
19 }
```

Listing 6.32: Hinzufügen der SolutionControl-Elemente

6.6.6 Editieren eines DataGrids im Code Behind

Im User-Control-Element *SolutionControl* werden die Touren sowohl in einer Karte als auch in einer DataGrid-Tabelle dargestellt. Da es möglich ist, dass Studenten in der Spiel-Applikation Lösungen zu einem Szenario erstellen und speichern, die nicht gültig sind, müssen diese ungültigen Routen in der DataGrid-Tabelle besonders gekennzeichnet werden. Ungültig ist eine Route dann, wenn sie die Kapazitätsrestriktionen verletzt oder am Anfang und Ende der Route nicht das Depot steht. Bei Routen ohne Depot handelt es sich um die in der Spiel-Applikation noch unverplanten Orte.
Die DataGrid-Tabelle im User-Control-Element *SolutionControl* wird im Code-Behind-Teil erstellt. Aus diesem Grund müssen auch die Kennzeichnung von ungültigen Routen und andere DataGrid-Formatierungen im C#-Code und nicht in der XAML-Datei erfolgen. Wie dies funktioniert, wird in den Codeausschnitten 6.33 und 6.34 gezeigt.

DataGrid-Binding

Codebeispiel 6.33 zeigt, wie Data-Binding für die DataGrid-Tabelle umgesetzt wird. Dazu wird in Zeile 2 das Attribut *ItemsSource* für das DataGrid-Element auf die Liste der anzuzeigenden Routen gesetzt. In Zeile 3 und 4 wird definiert, dass die Spalten der Tabelle nicht automatisch sondern manuell erstellt werden sollen und dass das DataGrid-Element vom Benutzer nicht bearbeitet werden kann. Das manuelle Anlegen der Spalten erfolgt ab Zeile 6. Dazu werden zunächst zwei Spalten des Typs *DataGridTextColumn* erzeugt. Über das Attribut *Header* wird die Überschrift für eine Spalte festgelegt und mit *Binding* eine Verknüpfung zu den anzuzeigenden Daten erzeugt. Die dritte Spalte ist vom Typ *DataGridTemplateColumn*. Mit diesem Typ ist es möglich, selbst zu definieren, welche UI-Elemente in einer Zelle dieser Spalte angezeigt werden

sollen. Der Grund, warum hier dieser Typ gewählt wurde, ist, dass in dieser Spalte ein automatischer Zeilenumbruch erfolgen muss und dies nur mit einem TextBlock-Element möglich ist. In Zeile 17 wird dazu ein Objekt des Typs *FrameworkElementFactory* angelegt, das die Erzeugung von Template-Elementen unterstützt [msdd]. Der Typ für dieses Objekt wird auf *TextBlock* gesetzt. Anschließend wird für das *FrameworkElementFactory*-Objekt die Datenverknüpfung erstellt und in Zeile 21 der automatische Zeilenumbruch gesetzt. Von Zeile 23 bis 25 erfolgt die Zuweisung des soeben erstellten Template-Elements zur entsprechenden DataGrid-Spalte. Am Ende werden der DataGrid-Tabelle die drei erstellten Spalten zugewiesen.

```
1  DataGrid descriptionGrid = new DataGrid();
2  descriptionGrid.ItemsSource = displayRoutes;
3  descriptionGrid.AutoGenerateColumns = false;
4  descriptionGrid.IsReadOnly = true;
5
6  DataGridTextColumn cRouteName = new DataGridTextColumn();
7  cRouteName.Binding = new Binding("Name");
8
9  DataGridTextColumn cDistance = new DataGridTextColumn();
10 cDistance.Header = "Distanz";
11 cDistance.Binding = new Binding("dist");
12
13 DataGridTemplateColumn cLocations = new DataGridTemplateColumn();
14 cLocations.Header = "Orte";
15 cLocations.Width = new DataGridLength(1, DataGridLengthUnitType.Star);
16
17 FrameworkElementFactory fact = new FrameworkElementFactory(typeof(TextBlock));
18 Binding b1 = new Binding("Route");
19 fact.SetBinding(TextBlock.TextProperty, b1);
20 //automatic line break
21 fact.SetValue(TextBlock.TextWrappingProperty, TextWrapping.Wrap);
22 //set the template for cLocations-Column
23 DataTemplate cellTemplate1 = new DataTemplate();
24 cellTemplate1.VisualTree = fact;
25 cLocations.CellTemplate = cellTemplate1;
26
27 descriptionGrid.Columns.Add(cRouteName);
28 descriptionGrid.Columns.Add(cDistance);
29 descriptionGrid.Columns.Add(cLocations);
```

Listing 6.33: DataGrid-Binding

DataGrid-Style

Routen, die ungültig sind, sollen in der DataGrid-Tabelle grau hinterlegt werden. Die nötige Implementierung dazu wird in Codebeispiel 6.34 gezeigt. Zunächst wird in Zeile 1 ein neues *Style*-Objekt angelegt und dessen Ziel-Typ dann in Zeile 2 auf den Typ *DataGridRow* gesetzt. Somit kann der Stil auf DataGrid-Zeilen angewandt werden. Anschließend wird ein Objekt des Typs *DataTrigger* angelegt. Für einen solchen Trigger muss eine Bedingung, unter der er ausgeführt werden soll, festgelegt werden [msd16]. In diesem Fall ist die Ausführungsbedingung die, dass der Wert der Variable *isOK* gleich *false* ist. In Zeile 6 wird dann definiert, welche Aktion der Trigger durchführen soll, wenn seine Bedingung erfüllt ist. Die Aktion in diesem Beispiel ist die, dass

der Hintergrund der DataGrid-Zeile hellgrau eingefärbt wird. In Zeile 11 wird dann dem *Style*-Objekt der zuvor definierte Trigger zugewiesen. Abschließend wird für die DataGrid.Tabelle das Attribut *RowStyle* gesetzt.

```
1  Style rowStyle = new Style();
2  rowStyle.TargetType = typeof(DataGridRow);
3  DataTrigger tg = new DataTrigger() {
4      Binding = new Binding("isOK"), Value = false
5  };
6  tg.Setters.Add(new Setter()
7  {
8          Property = DataGridRow.BackgroundProperty,
9          Value = Brushes.LightGray
10 });
11 rowStyle.Triggers.Add(tg);
12
13 descriptionGrid.RowStyle = rowStyle;
```

Listing 6.34: Kennzeichnen der falschen Routen im DataGrid-Element

Kapitel 7

Kritische Beurteilung

7.1 Erster Test an der FH Steyr

Am 14. Februar 2018 fand der erste Live-Test an der FH Steyr in der Lehrveranstaltung „Distributionslogistik“ statt. Zunächst wurde dabei mit einem Szenariengenerator vom Auftraggeber und Vortragenden eine Szenario-Datei mit dem Szenario, das die Studenten dann in der Lehrveranstaltung lösen sollen, erstellt. Zusätzlich wurde bereits im Vorhinein ein eigenes Tourenplanungsproblem und dazu eine Excel-Angabe als Hausübung für jeden Studenten generiert. In der Lehrveranstaltung wurde die Spiel-Applikation zunächst von den Studenten installiert und daraufhin das zuvor vom Professor generierte Szenario in das Spiel geladen. Anschließend gab es 25 Minuten Zeit, um die Problemstellung zu lösen. Während dieses Tests sind keine Probleme beim Spielen aufgetreten, jedoch gab es bei der Installation Probleme aufgrund des Virenscanners. Des Weiteren gab es ein paar Verbesserungsvorschläge von den Studenten, die mit einem Fragebogen erhoben wurden.

Ergebnisse des Tests

Probleme bzw. Vorschläge	**Behebung bzw. Umsetzung**
Da der Herausgeber des Programms zu diesem Zeitpunkt noch kein Zertifikat hatte, wurde bei der Installation die Warnung „Unbekannter Herausgeber“ ausgegeben. Dies erforderte dann eine Freigabe der Installation mit Administratorrechten.	Es wird nun ein Zertifikat der HTL Perg verwendet, das den Herausgeber verifiziert.
Mit dem zu der Zeit verwendeten Installationstool war es nicht möglich, einen anderen Speicherort für das Programm am Computer zu wählen.	Für das Erstellen einer Installationsdatei aus dem Visual Studio Projekt wird nun die Software InnoSetup verwendet. Mit ihr sind viele zusätzliche Einstellungsmöglichkeiten gegeben, wie zum Beispiel eine Auswahl des Speicherorts oder das Erstellen einer Desktop-Verknüpfung.

Ein Vorschlag vieler Studenten war, die Möglichkeit zu haben, auch unvollständig verplante oder ungültige Lösungen zwischenspeichern zu können.	Dieser Vorschlag wurde umgesetzt und es ist nun möglich, eine Lösung jederzeit in eine Szenario-Datei oder auch in die Datenbank zu speichern.
Einige Studenten schlugen vor, die Farbe der ausgewählten Registerkarte zu ändern, da es für sie nicht sofort klar war, wie man von der Planer-Ansicht in die Karten-Ansicht wechselt.	Die Farbgestaltung wurde den Anregungen entsprechend angepasst.
Es wurde bemerkt, dass es einfacher wäre, wenn in der Planer-Ansicht neu hinzugefügte Touren links, also direkt neben den unverplanten Orten, angeordnet werden würden.	Nach einer Besprechung mit dem Auftraggeber wurde entschieden, dass das derzeit eingesetzte Format der Planer-Ansicht jedoch übersichtlicher ist.
Manche Studenten äußerten im Fragebogen den Wunsch, die eigene Planung in Form einer Excel-Datei speichern zu können	Dieser Wunsch hat sich erübrigt, da es nun möglich ist, auch unfertige Lösungen zu speichern.

Tabelle 7.1: Ergebnisse des Tests

7.2 Mögliche Erweiterungen

In diesem Abschnitt werden Erweiterungsmöglichkeiten für die Spiel-Applikation und den Szenariengenerator, die nicht mehr Teil der Diplomarbeit sind, vorgeschlagen.

7.2.1 Echtzeitdaten aus Bing Maps

Derzeit ist in der Datenbank eine fixe Anzahl an Orten und die dazugehörigen Distanzen zwischen ihnen statisch gespeichert. Es gibt also keine Möglichkeit, im Programm neue Orte hinzuzufügen. Des Weiteren werden Fahrzeiten und Entfernungen zwischen den Orten nicht an die zum Zeitpunkt der Programmausführung aktuelle Verkehrssituation, mögliche Änderungen im Straßennetz oder Baustellen angepasst.
Eine mögliche Lösung dazu wäre, Distanzen zwischen den Orten nicht mehr statisch in den Szenario-Dateien oder in der Datenbank zu speichern, sondern jedes Mal, wenn man die Daten im Programm benötigt werden, eine neue Abfrage an den Dienst *Bing Maps Routes API* zu starten. Dies würde jedoch den Nachteil nach sich ziehen, dass aufgrund der dadurch entstehenden großen Menge an Abfragen eine kostenpflichtige Bing Maps Lizenz nötig wäre.
Mit dieser Lizenz wäre es dann auch möglich, beliebige Orte zu verwenden.

7.2.2 Erweiterung der Algorithmen-Bibliothek

Neben den derzeit verwendeten Algorithmen gibt es eine Vielzahl weiterer Möglichkeiten, derartige Tourenplanungsprobleme zu lösen. Diese könnten entweder selbst implementiert oder durch Verwendung weiterer im HeuristicLab implementierter Verfahren eingebunden werden.
Des Weiteren könnten die Tourenplanungsprobleme um Zeitfenster für die zu beliefernden Orte erweitert werden. Dazu müssten Algorithmen eingebunden werden, die die Berechnung von Planungsproblemen mit Zeitfenstern unterstützen. Für diese mögliche Erweiterung sind in der bestehenden Datenbank in der Tabelle *Location* bereits die beiden Felder *startTime* und *endTime* vorhanden.

7.3 Persönliches Resümee

Durch das Verfassen unserer Diplomarbeit haben wir erstmals einen Einblick bekommen, wie wissenschaftlich gearbeitet wird, was uns im späteren Studium oder Berufsleben sicher helfen wird. Des Weiteren haben wir uns durch das Lesen entsprechender Fachliteratur und das Besuchen einer Lehrveranstaltung zu diesem Thema sehr genau mit der Tourenplanung auseinandergesetzt. Die Implementierung des Programms hat es uns ermöglicht, unser Wissen im Umgang mit WPF und WCF stark zu vertiefen. Weiters haben wir gelernt, dass es bei Projekten dieses Umfangs sehr wichtig ist, sich von Anfang an Gedanken über den Aufbau der Programme, das Datenmodell sowie die Projektorganisation zu machen.

Da wir mit der Planung und Implementierung der Arbeit schon in den Sommerferien begonnen haben, konnten wir das Programm bis Weihnachten fertigstellen. Deshalb ist uns noch genügend Zeit für das ausführliche Testen sowie die Umsetzung von Verbesserungsvorschlägen geblieben.
Was uns außerdem bei der Umsetzung unserer Arbeit sehr geholfen hat, waren der ständige Kontakt mit, sowie die Rückmeldungen von unserem Betreuer und unserem Auftraggeber. Auch die Kommunikation zwischen uns beiden hat sehr gut funktioniert. Wir haben uns beide an unsere Abmachungen und Termine gehalten, wodurch Probleme von Anfang an vermieden werden konnten.

Literaturverzeichnis

[ABD+14] Affenzeller, Beham, Dorfer, Kofler, Kommenda, Kronberger, Pitzer, Scheibenpflug, Vonolfen, Wagner, and Winkler. *Architecture and Design of the HeuristicLab Optimization Environment*. Springer Verlag, 2014. 193-257 pp.

[All08] K. Scott Allen. The Standard LINQ Operators, Oktober 2008. URL `https://odetocode.com/articles/739.aspx`. zugegriffen am: 31.01.2018.

[AP16a] Rick Anderson and Samir Patel. Client Certificate Mapping Authentication ¡clientCertificateMappingAuthentication¿, September 2016. URL `https://docs.microsoft.com/en-us/iis/configuration/system.webserver/security/authentication/clientcertificatemappingauthentication`.

[AP16b] Rick Anderson and Samir Patel. IIS Client Certificate Mapping Authentication ¡iisClientCertificateMappingAuthentication¿, September 2016. URL `https://docs.microsoft.com/en-us/iis/configuration/system.webserver/security/authentication/iisclientcertificatemappingauthentication/`. zugegriffen am: 27.02.2018.

[AP16c] Rick Anderson and Samir Patel. Security Authentication ¡authentication¿, September 2016. URL `https://docs.microsoft.com/en-us/iis/configuration/system.webserver/security/authentication/`. zugegriffen am: 31.01.2018.

[BDLW17] Stephen J. Bigelow, Keith Dodge, Bob Lehto, and Mike Weiner. Internet Information Services (IIS), März 2017. URL `http://searchwindowsserver.techtarget.com/definition/IIS`. zugegriffen am: 31.01.2018.

[bil] Basic Authentication. URL `https://technet.microsoft.com/en-us/library/cc961153.aspx`. zugegriffen am: 31.01.2018.

[bina] Locations API. URL `https://msdn.microsoft.com/en-us/library/ff701715.aspx`. zugegriffen am: 22.02.2018.

[binb] Routes API. URL `https://msdn.microsoft.com/en-us/library/ff701705.aspx`. zugegriffen am: 22.02.2018.

[Bre15] Wolf-Rüdiger Bretzke. *Logistische Netzwerke*. Springer Verlag, 2015. 364 - 369 pp.

[Dom97] Wolfgang Domschke. *Logistik: Rundreisen und Touren*. Oldenburg Verlag, 1997. 234 pp.

[dot06] Chapter 1: Introduction to .NET, Mai 2006. URL `https://technet.microsoft.com/en-us/library/bb496996.aspx`. zugegriffen am: 12.01.2018.

[Dra16] DragDropEffects-Enumeration, Oktober 2016. URL https://msdn.microsoft.com/de-de/library/system.windows.forms.dragdropeffects(v=vs.110).aspx. zugegriffen am: 23.02.2018.

[gon] gong-wpf-dragdrop. URL https://github.com/punker76/gong-wpf-dragdrop. zugegriffen am: 23.02.2018.

[heu] HeuristicLab. URL https://dev.heuristiclab.com/trac.fcgi/. zugegriffen am: 02.01.2018.

[JSOa] Einführung in JSON. URL json.org/json-de.html. zugegriffen am: 05.02.2018.

[JSOb] JSON - Introduction. URL https://www.w3schools.com/js/js_json_intro.asp. zugegriffen am: 05.02.2018.

[Luk15] Sean Luke. *Essentials of Metaheuristics*. October 2015. 9-26 pp.

[msda] DrawingContext Class. URL https://msdn.microsoft.com/en-us/library/system.windows.media.drawingcontext(v=vs.110).aspx. zugegriffen am: 22.02.2018.

[msdb] DrawingGroup Class. URL https://msdn.microsoft.com/en-us/library/system.windows.media.drawinggroup(v=vs.110).aspx. zugegriffen am: 22.02.2018.

[msdc] DrawingVisual Class. URL https://msdn.microsoft.com/en-us/library/system.windows.media.drawingvisual(v=vs.110).aspx. zugegriffen am: 22.02.2018.

[msdd] FrameworkElementFactory Class. URL https://msdn.microsoft.com/en-us/library/system.windows.frameworkelementfactory(v=vs.110).aspx. zugegriffen am: 03.03.2018.

[msde] Microsoft.Office.Interop.Excel-Namespace. URL https://msdn.microsoft.com/library/microsoft.office.interop.excel(v=vs.140).aspx. zugegriffen am: 22.02.2018.

[msdf] RenderTargetBitmap-Klasse. URL https://msdn.microsoft.com/de-de/library/system.windows.media.imaging.rendertargetbitmap(v=vs.110).aspx. zugegriffen am: 22.02.2018.

[msd16] DataTrigger-Klasse, Oktober 2016. URL https://msdn.microsoft.com/de-de/library/system.windows.datatrigger(v=vs.110).aspx. zugegriffen am: 03.03.2018.

[Ohr08] Claudius Ohrt. *Tourenplanung im Straßengüterverkehr*. Gabler Verlag, 2008. 30 pp.

[Pat10] Diptimaya Patra. Drag And Drop Item in ListBox in WPF, Jänner 2010. URL http://www.c-sharpcorner.com/uploadfile/dpatra/drag-and-drop-item-in-listbox-in-wpf/. zugegriffen am: 04.10.2018.

[Pii08] Jarkko Piiroinen. Overview of the Common Language Infrastructure.svg, Februar 2008. URL `https://commons.wikimedia.org/wiki/File:Overview_of_the_Common_Language_Infrastructure.svg`. zugegriffen am: 12.01.2018.

[RNL17] Margaret Rouse, Tom Nolle, and Thomas Li. application program interface (API), April 2017. URL `http://searchmicroservices.techtarget.com/definition/application-program-interface-API`. zugegriffen am: 12.01.2018.

[Rou07] Margaret Rouse. Common Language Infrastructure (CLI), Juni 2007. URL `http://searchmicroservices.techtarget.com/definition/Common-Language-Infrastructure-CLI`. zugegriffen am: 18.01.2018.

[Rui08] Marcelo Lopez Ruiz. $filter Query Option in ADO.NET Data Services, Jänner 2008. URL `https://blogs.msdn.microsoft.com/marcelolr/2008/01/11/filter-query-option-in-ado-net-data-services/`. zugegriffen am: 18.02.2018.

[Sar14] Saravanakumar. Parts Of OData, 2014. URL `http://wcftutorial.net/Parts-Of-OData.aspx`. zugegriffen am: 18.02.2018.

[tab] Tabu Search. URL `https://dev.heuristiclab.com/trac.fcgi/wiki/TS`. zugegriffen am: 23.02.2018.

[Vah05] Richard Vahrenkamp. *Logistik, Management und Strategien*. Oldenburg Verlag, 2005. 448-464 pp.

[veh] Vehicle Routing Problem. URL `https://dev.heuristiclab.com/trac.fcgi/wiki/Documentation/Reference/VehicleRoutingProblem`. zugegriffen am: 23.02.2018.

[WJHL17] Maira Wenzel, Mike Jones, Matt Hoffmann, and Luke Latham. WCF Data Services Overview, März 2017. URL `https://docs.microsoft.com/en-us/dotnet/framework/data/wcf/wcf-data-services-overview`. zugegriffen am: 18.02.2018.

[WWB+17] Bill Wagner, Maira Wenzel, Mike B, Luke Latham, and Steve Hoag. Language Integrated Query (LINQ), Februar 2017. URL `https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/`. zugegriffen am: 01.02.2018.

[WyI+17] Maira Wenzel, yishenhjin1413, ItsPugle, guardrex, Mikejo5000, mikeblome, and craigg msft. LINQ to SQL, März 2017. URL `https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/`. zugegriffen am: 01.02.2018.

[ZB07] Günther Zäpfel and Michael Bögl. *Tourenplanung mit Zeitfenstern*. Trauner Verlag, 2007. 5-44 pp.

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis