|  |
| --- |
| Mateusz Ochal |
| TravelConn - bus journey finder |
| A solution to the quickest path problem |
|  |
| **Mateusz Ochal** |
|  |

Table of Contents

[Analysis 3](#_Toc414586190)

[Introduction 3](#_Toc414586191)

[Investigation 3](#_Toc414586192)

[Documents of the current system 4](#_Toc414586193)

[Data in the proposed system 6](#_Toc414586194)

[Data Volumes: 7](#_Toc414586195)

[Limitations of Database Format 8](#_Toc414586196)

[Constraints 8](#_Toc414586197)

[**Software and Hardware constraints** 8](#_Toc414586198)

[**Time constraints** 9](#_Toc414586199)

[**Users and Clients** 9](#_Toc414586200)

[**Benefits of Computerising** 9](#_Toc414586201)

[**Not included in the computerisation** 9](#_Toc414586202)

[Objectives of the new system: 9](#_Toc414586203)

[Possible solutions 10](#_Toc414586204)

[**Storage Solutions:** 10](#_Toc414586205)

[**Program Solutions:** 10](#_Toc414586206)

[**Chosen Solution:** 10](#_Toc414586207)

[Security and Integrity of Data 11](#_Toc414586208)

[Objectives signed by client 12](#_Toc414586209)

[Research Methods 13](#_Toc414586210)

[Early stages and ideas 14](#_Toc414586211)

[Contacting people 15](#_Toc414586212)

[Design 16](#_Toc414586213)

[Introduction 16](#_Toc414586214)

[IPSO chart 16](#_Toc414586215)

[Storage 17](#_Toc414586216)

[Entity Relation Diagram 17](#_Toc414586217)

[Data Volumes 19](#_Toc414586218)

[Background 19](#_Toc414586219)

[Data Dictionary and Data Volumes 19](#_Toc414586220)

[CBusNetwork Class 21](#_Toc414586221)

[Algorithms 24](#_Toc414586222)

[Original Dijkstra’s Algorithm 24](#_Toc414586223)

[Depart after and arrive before algorithm 25](#_Toc414586224)

[Space and time complexity of the algorithm 27](#_Toc414586225)

[System Flow Diagram 28](#_Toc414586226)

[User Interface Design 29](#_Toc414586227)

[Design 1: (Screenshot of a prototype) 29](#_Toc414586228)

[Design 2: Sketch of an improved design 29](#_Toc414586229)

[Interface description and interaction 30](#_Toc414586230)

[Visual Component Class Diagrams 31](#_Toc414586231)

[*Class relationship* 31](#_Toc414586232)

[ResultTable design 33](#_Toc414586233)

[Window Navigation 35](#_Toc414586234)

[Test Plan - Introduction 37](#_Toc414586235)

[Testing Outline 41](#_Toc414586236)

[Test Plan 42](#_Toc414586237)

[Testing 48](#_Toc414586238)

[System Maintenance 59](#_Toc414586239)

[Overview of the new system 59](#_Toc414586240)

[Interface 59](#_Toc414586241)

[MapCtrl - in depth description 59](#_Toc414586242)

[Accessing bus network data 60](#_Toc414586243)

[Automatically generated bus networks 60](#_Toc414586244)

[Important information 60](#_Toc414586245)

[Connecting to a database 60](#_Toc414586246)

[Code overview 61](#_Toc414586247)

[Additional Class Description 61](#_Toc414586248)

[User Controls Classes 62](#_Toc414586249)

[Windows 62](#_Toc414586250)

[MainWindow Class 63](#_Toc414586251)

[Functions and procedures 63](#_Toc414586252)

[Variables 64](#_Toc414586253)

[Class Diagram 64](#_Toc414586254)

[Bus Network Classes 64](#_Toc414586255)

[Trace Tables 65](#_Toc414586256)

[Depart After Algorithm (CBusNetwork) 65](#_Toc414586257)

[Code 72](#_Toc414586258)

[SQL 73](#_Toc414586259)

[Creating tables 73](#_Toc414586260)

[Populating tables 74](#_Toc414586261)

[XAML 75](#_Toc414586262)

[MainWindow 75](#_Toc414586263)

[SplashWindow 78](#_Toc414586264)

[BusStopInfoWindow 78](#_Toc414586265)

[JourneyInfoWindow 79](#_Toc414586266)

[BusStopCtrl 79](#_Toc414586267)

[LinkCtrl 80](#_Toc414586268)

[*MapCtrl* 81](#_Toc414586269)

[ResultTableCtrl 81](#_Toc414586270)

[C# 83](#_Toc414586271)

[BusNetwork Classes 83](#_Toc414586272)

[MainWindow 97](#_Toc414586273)

[SplashWindow 104](#_Toc414586274)

[BusStopInfoWindow 105](#_Toc414586275)

[JourneyInfoWindow 107](#_Toc414586276)

[AboutWindow 108](#_Toc414586277)

[BusStopCtrl 108](#_Toc414586278)

[LinkCtrl 109](#_Toc414586279)

[MapCtrl 111](#_Toc414586280)

[ResultTableCtrl 116](#_Toc414586281)

[User Manual 122](#_Toc414586282)

[TravelConn - User Manual 124](#_Toc414586283)

[Instilalisation from a CD disk 124](#_Toc414586284)

[Data sources 124](#_Toc414586285)

[Database 124](#_Toc414586286)

[Hardware and software configuration 124](#_Toc414586287)

[Main Window 125](#_Toc414586288)

[Basic Lay Out 126](#_Toc414586289)

[Search Panel – Selecting your journey 126](#_Toc414586290)

[Map 127](#_Toc414586291)

[Results List – Viewing the results 127](#_Toc414586292)

[Printing 128](#_Toc414586293)

[Printing - Step by step 128](#_Toc414586294)

[Saving an electronic copy of the document 129](#_Toc414586295)

[Viewing more detail 129](#_Toc414586296)

[Contact Details 131](#_Toc414586297)

[Appraisal 133](#_Toc414586298)

[Meeting the objectives 133](#_Toc414586299)

[General Objectives: 134](#_Toc414586300)

[Extension Objectives: 134](#_Toc414586301)

[Feedback from the client, Joe Johnson 135](#_Toc414586302)

[Response to feedback 136](#_Toc414586303)

[Screenshots 138](#_Toc414586304)

# Analysis

## Introduction

#### Background

The Tourist Information Centre in Cambridge provides information about local accommodation, public transport, conference services, events, UK holiday information, walking tours of Cambridge, maps & guidebooks and much more. They already have a website and software that run in their office that provide general information about local events and information. However they do not have a computerised version for information about public transport. Currently the Tourist Information Centre uses printed timetables provided by Stagecoach to store information about buses. In the past this system was sufficient as Cambridge had very few buses and it was easy find a way around. Over time Cambridge has grown and expanded and now there are over 30 buses coming from and to Cambridge. A few problems have arisen with the current system and Joe Johnson, who works at the information centre, has asked me to make a computerised solution to this problem.

#### Project Definition

Client: The Cambridge Tourist Information Centre

Contact: Joe Johnson

Peas Hill

Cambridge,

CB2 3AD

## Investigation

#### Summary of the current system based on interview from 8th Nov 2014

The problems with the current system lie in finding out the optimal bus journey between bus stops. The tourist office in Cambridge uses printed timetables of buses provided by Stagecoach in Cambridge for the tourists to find how to get from place to place. This requires a lot of time as people have to look up a table. In most cases the bus journey involves at least one bus change. It is not easy to spot which bus stops share bus stops and where it is best to get on and/or off. The tourists often ask for help at the desk, unless the worker has memorised all the timetables the workers usually result to looking at the printed timetables themselves. This takes up a lot of time which could otherwise be spent answering other questions. The printed timetables are often given to tourists for their benefit but this is often unnecessary as only specific bus journeys are required. This system requires a lot of printing and is expensive. The printed timetables only show the name of the bus stops and not the address and although the bus stop name often corresponds to a nearby street name or facility, it is difficult to locate its actual location.

The printed timetables are purchased from by Stagecoach Cambridge for a reasonably small financial cost. The timetables are printed using the information from their database which gets updated at least once a year, so Tourist Information Centre has to request a reprint of their timetables to be kept up to date.

#### Problems with the current system

Here are the gathered problems with the current system:

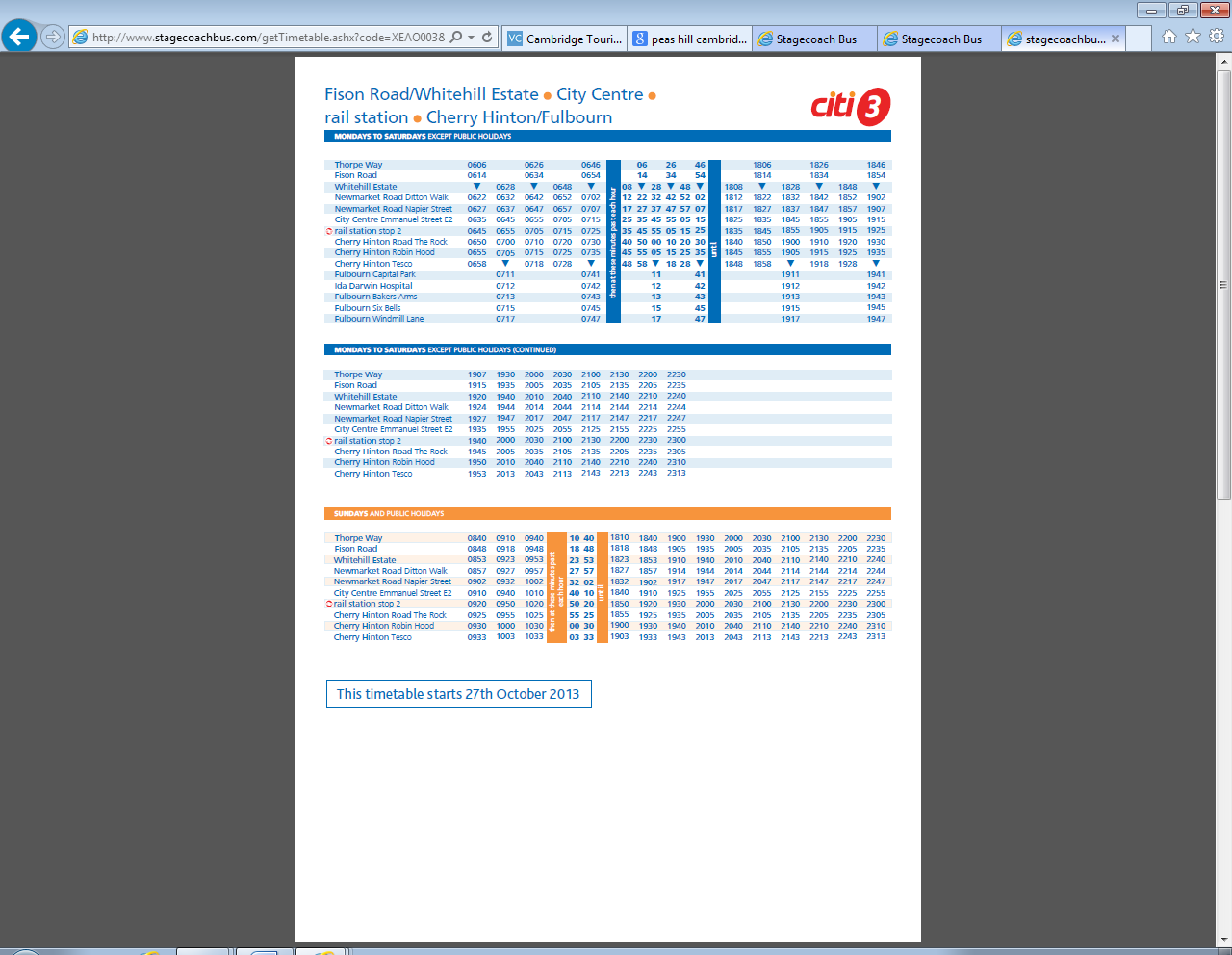
Printed timetables – needed for every bus coming from and to Cambridge, given to each tourist if they would like a copy of a timetable for themselves, costs money, uses a lot of paper and ink, hard to search through the timetable, hard to seek bus stops, time costly, only key bus stops are include in the timetable, reprinting required each time a timetable changes, only bus routes are given and one has to look through several bus routes to plan an appropriate bus journey, hard to find alternative bus journey.

Finding the appropriate bus stop – not all bus stops are included in the timetable, no data about the location of the bus stop apart from its name, no address, hard to locate on a map. Only the names are related to the location.

Input, output, forms and report formats from existing system

## Documents of the current system

#### Example of a timetable for a bus used by the current system:

****

#### Data Flow Diagram of the current system

Printed timetables stacks

P1

Bus journey

The worker from the tourist office/ the tourist looks at the printed timetables

1

Timetable Printing

The timetables are printed using data provided by Stagecoach

2

Destination and source bus stop

Request Timetables of buses

Timetables of buses

Timetables of buses in paper form

Request Timetables of buses

Timetables of buses

## Data in the proposed system

#### Data flow diagram

A postcode/ address of a location to display nearest bus stops

Maps interface

Bus name

Destination and source bus stop

Find Bus stop

Find appropriate bus stop given a location

3

Find bus journey

The tourist office worked looks at the printed timetables

2

Bus route information

General information about the bus

1

Bus stop location

Bus info

Timetable info

Bus stop info

Request

Request

Request

D2

Database timetable and day category

Database bus stops

D3

Database bus routes

D1

Timetable and day category

Bus stops

Bus Routes

Update

Requesting an update for the database

2

A copy of the database bus information

Request

#### Entity Relation Diagram

Here is an entity diagram which shows the basic relationship between the tables in the proposed system. This is only a very basic design for a database to act like a Stagecoach database.

Timetable

BusRoutes

Busstops

DayCategory

## Data Volumes:

#### Background

After doing some preliminary research, I found the following information about the Stagecoach bus network in Cambridge:

* There are 33 stagecoach buses coming to and from Cambridge
* Each bus has a 2 ‘bound’ route
* Each route has about 70 bus stops
* 6 buses are every 10 min, 10 buses are every 20 min and the rest 17 buses on average are every 45 min in a day
* The buses usually run from 6:00 until 21:00

#### Data Dictionary and Data Volumes:

Here is a basic design for the database to get an idea of how much storage would be required for the system.

**Bold** indicates a primary or in the case where there are several primary keys they make up a composite key.

Table name - Buses

|  |  |  |  |
| --- | --- | --- | --- |
| Columns | Type | Size | Number of bytes |
| **BusID** | Integer | Up to 66 | 1 |
| Name | String | 20 chars (ASCII) | 20 |
| Description | String | 200 chars (ASCII) | 200 |
| Number of bytes for each row | | | 221 |
| Total kB | | | 13.3 |

Bus Stops (there are about 2000 bus stops)

|  |  |  |  |
| --- | --- | --- | --- |
| Columns | Type | Size | Number of bytes |
| **Bus Stop ID** | Integer | Up to 2000 | 2 |
| Name | String | 80 chars (ASCII) | 80 |
| Latitude | Decimal | Between -90.0 and 90.0 | 4 |
| Longitude | Decimal | Between -180.0 and 180.0 | 4 |
| Number of bits for each row | | | 90 |
| Total kB | | | 1800 |

Day Categories (Mondays to Saturdays, and Sundays and Bank Holidays)

|  |  |  |  |
| --- | --- | --- | --- |
| Columns | Type | Size | Number of bytes |
| **Day Category ID** | Integer | Up to 9 | 1 |
| Description | String | 30 chars (ASCII) | 30 |
| Number of bits for each row | | | 31 |
| Total kB | | | 0.27 |

Timetable (70 bus stops on average for each bus route, 66 routes, on average 60 bus courses every day for each bus route, 3 day categories, and therefore about 830000 rows of data altogether)

|  |  |  |  |
| --- | --- | --- | --- |
| Columns | Type | Size | Number of bytes |
| **Bus ID** | Integer | Up to 66 | 1 |
| **Bus Course ID** | Integer | Up to 99 | 1 |
| **Bus Stop ID** | Integer | Up to 2000 | 2 |
| Time | Integer | 0000 to 2400 | 2 |
| Day Category | Integer | Up to 9 | 1 |
| Number of bits for each row | | | 7 |
| Total kB | | | 5800 |

All together: about 7600kB, which means that the size of the database would be about 7.6Mbytes.

This is the size of the database alone, the actual size of the objects could be much

### Limitations of Database Format

The timetable is day dependent meaning if there was a ‘night’ bus, which ended its route on a different day it would have to be stored differently. One alternative could store 1 am of the next day as 2500, to show that it exceeds the day by 1 hour.

## Constraints

**Software and Hardware constraints**

The devices must have access to the internet.

**Time constraints**

The final outcome should be completed and ready by 20thMarch.

**Users and Clients**

The users are going to tourists that not necessarily understand English the software should be in simple English for foreign tourists to understand

**Benefits of Computerising**

Allowing bus journeys to be calculated quickly and efficiently

Allowing the tourist office to keep the timetables up to date

Allowing a more user-friendly interface

Allowing an extension to their already existing software and website

Allowing a user to print only what they need and saving paper

**Not included in the computerisation**

This solution would require software to manage the database and sync it with the Stagecoach database.

## Objectives of the new system:

#### Specific Objectives:

1. The program should accept a start and destination bus stop and calculate the shortest journey between them.
2. The calculations should take into account different bus timetables for different days.
3. There should be two algorithm options the user can select from: ‘arrive before’ and ‘depart after’.
4. The user should be able to choose a date from a calendar.
5. The program should display the shortest bus journey.
6. The program should keep track of the previous result of calculated bus journeys showing the time of start of the journey and the time of arrival at the destination, and number of bus changes, total time of journey.
7. Upon selecting a route more details should be shown including: the time and place of the bus changes, the bus name of the buses, time of the journey on each bus and waiting time between bus changes and the total waiting time.
8. The program should produce a document that could be printed or stored if desired. The information should include: the time and place of the bus changes, the bus name of the buses, time of the journey on each bus and waiting time.
9. When selected, the route will be displayed on a map, showing the bus journey including all bus stops and bus changes involved.
10. The map should have basic zoom in and out feature and span.
11. The user should interact with the map and be able to choose start and destination bus stop.
12. The system should be capable of handling about 8Mbytes of data at a time and the bus journeys should be displayed within 2 seconds.

#### General Objectives:

1. The program should be intuitive and easy to use for non-experienced tourists to learn how to use it quickly.

#### Extension Objectives:

1. When clicked on a bus stop on the map the following information could be displayed: the buses going through the bus stop, geographical location.

## Possible solutions

**Storage Solutions:**

Stagecoach does not provide access to their database but is willing to send copies of their timetables in electronic file, which can be easily pulled into a database or text file.

***Text File***- This solution of storing information would have a high to seek time as it would involve looking at about 200000 records

***Local Database*** - a database has existing optimised search algorithms, good for a desktop solution

***Server Based Database*** - hosting required – requires money, internet access – not too much of a problem. This type of solution would

**Program Solutions:**

***A desktop application***- has to be install on every computer, can only be accessed from the tourist office, could use both a local database or from a server, can be operating system specific in certain programming languages, slightly easier to develop

***A phone application*** - nearly everyone if not all have a phone, but operating system specific android, OS and Windows Phone, assumes every tourist has a suitable phone, could use a database but limited processing speed and storage space, a server based database may be required, with this solution printing could be avoided because a phones is portable

*My limitations*: I would need to discover a lot more about making phone apps, I would develop it the apps in C# which would only work on Windows Phones

***A web-based app and a server based database***- can be accessed from anywhere with internet connection, anyone would be able to access the application, cross platform, requires a server based database, accessible from both phone and desktop, calculation would be done on a server.

*My limitations*: I would need to discover a lot more about making web apps. There are different ways of creating web solutions; the most common one is MVC. I would require me to break through this type of style of programming.

**Chosen Solution:**

Desktop application - I choose this solution because I feel most confident in developing programs in this form. Windows Application Forms use XMAL which is a mark-up language so it will be capable of creating a friendly graphical interface.

## Security and Integrity of Data

As there is no confidential information stored about the bus networks within the system, there is no need for any kind of encryption. Users will not have access to the data in the system directly as all the set up would be made by stuff members of the tourist information office or the developer of the application. Therefore restricted access profile is similarly not a concern. Storing the bus network in text files or in a hosted SQL database means that information can be accessed offline as well as online, for more up-to-date information. However there should also be a possibility of connecting to a secure database source.

To protect the integrity of the stored data, all data entry should be controlled by strict validation rules. The data will be validated in an extent before being fully loaded into the program. To help user search of a bus stop name they should be able to select their options from drop-down menus or radio buttons. This minimises free text input which, as well as saving time, also stops typographic errors which may cause the system to crash or work on invalid data to be stored.

# Objectives signed by client

#### Specific Objectives:

1. The program should accept a start and destination bus stop and calculate the shortest journey between them.
2. The calculations should take into account different bus timetables for different days.
3. There should be two algorithm options the user can select from: ‘arrive before’ and ‘depart after’.
4. The user should be able to choose a date from a calendar.
5. The program should display the shortest bus journey.
6. The program should keep track of the previous result of calculated bus journeys showing the time of start of the journey and the time of arrival at the destination, and number of bus changes, total time of journey.
7. Upon selecting a route more details should be shown including: the time and place of the bus changes, the bus name of the buses, time of the journey on each bus and waiting time between bus changes and the total waiting time.
8. The program should produce a document that could be printed or stored if desired. The information should include: the time and place of the bus changes, the bus name of the buses, time of the journey on each bus and waiting time.
9. When selected, the route will be displayed on a map, showing the bus journey including all bus stops and bus changes involved.
10. The map should have basic zoom in and out feature and span.
11. The user should interact with the map and be able to choose start and destination bus stop.
12. The system should be capable of handling about 8Mbytes of data at a time and the bus journeys should be displayed within 2 seconds.

#### General Objectives:

1. The program should be intuitive and easy to use for non-experienced tourists to learn how to use it quickly.

#### Extension Objectives:

1. When clicked on a bus stop on the map the following information could be displayed: the buses going through the bus stop, geographical location.

Signature: Date:

## Research Methods

The methods I used to find out about the problem included:

* interview with my client Joe Johnson (see notes taken during the interview below)
* observations of the current system – I watched how office workers interact with tourists. It usually takes 3-5 min for the tourists to find and talk with the office workers.

#### Evidence of the interview

Notes:

Summary of the Interview:

On Saturday 8th Nov I had an interview with Joe Johnson. It turns out he works for a tourist agency in Cambridge. He complains about the current system and asked me to help to solve it.

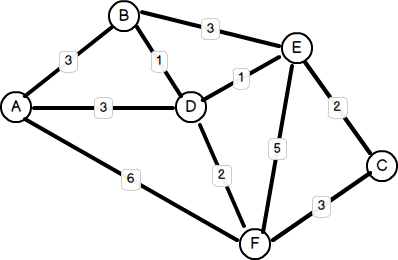
1. What is the current problem?
   1. Currently, the office uses printed timetables for all Stagecoach buses in Cambridge for the tourists to find how to get from place to place. This requires a lot of time. The tourists often ask for help at the desk with trivial problem of what bus will take them to their destination. The workers have to memories where each bus is going and if they don’t know they often have to look at a map themselves. They spend time answering this question which could otherwise be spent answering other questions. In addition, they are unable to answer questions on the time of journey and are unable to say if there is a quicker route. Printing timetables for each tourist who wants one is also expensive and often unnecessary when the tourists just want to know about one bus journey there and back.
2. What problem would you like to solve with the application?
   1. Having computers in the office with an application that would calculate their journey for them would be very useful to the tourists. They would be able to find the bus routes they need without coming up to the help desk or asking the workers.
3. Who is going to use this application?
   1. The application would be used by the tourists or anyone that needs to know how to get from a place to place.
4. When and where is the user going to use this application?
   1. While at the office and it would also be possible to download the application to a desktop so a tourist can look at it in their free time.
5. How is the user to use this application?
   1. The user will enter the start and destination bus stop, the desired time of departure and date.
   2. The bus journey will be calculated based on the information given by the user and based on Stagecoach schedule in Cambridge.
   3. At least three bus routes will be displayed showing the time of start of the journey and the time of arrival at the destination, and number of bus changes, total time of journey.
   4. Upon selecting a route more details will be shown including: the time and place of the bus changes, the bus name/number of the buses, time of the journey on each bus and waiting time.
   5. When selected, the route will be displayed on a map, showing all bus stops and bus changes.

## Early stages and ideas

#### Project Proposal

Visitors in Cambridge complain about finding the bus that would take them to their destination. They often have to spend a lot of time looking for the bus they need to take. They even have to ask the bus drivers about the bus which would take them to their stop. This causes time delays as the drivers have to explain where they would find the desired bus and wastes everybody’s time and is highly inefficient. I have stepped in to offer some help. For my project I will design and program, desktop application that will enable the user to find the desired route. Stagecoach manager expects the program to find the best route between bus stops in an efficient way.

A representation of a graph data structure:

Each stop will be a vertex storing the bus number that stops on the bus stop. There will be time between the nodes. These connections are called edges.

I could use a table to store each of the values, i.e. vertexes v edges. I need to do more research on it though. I need to learn how to implement a graph data structure in C#.

* There are algorithms for finding the shortest path possible like the Dijkstra's Shortest Paths. More algorithms can be found in this link: <http://www.boost.org/doc/libs/1_56_0/libs/graph/doc/index.html>

I need to look at the theory as well and see how much more memory is taken up by adding a node or an edge. On Wikipedia there is a nice table showing how the complexity rises when give n nodes and a edges. <http://en.wikipedia.org/wiki/Graph_(abstract_data_type)>

My application needs to be user friendly enough so that a user can easily navigate through it. It needs to be fairly responsive so that it takes a relatively short amount of time to calculate the optimal.

To expand the program I could introduce more cities because Stagecoach is not only used in Cambridge but other 400 towns and cities in UK. The information about the bus stops and buses in Cambridge can be found on their website (<http://www.stagecoachbus.com/timetable-landing.aspx>)

in a table format which can be easily copied and pasted. The software can also be expanded further giving information about the buses and ticket prices as well as an option of a map a user can click on to choose destinations.

My user/client is Joe Johnson.

## Contacting people

This email was sent to the Stagecoach Company asking them to provide information about bus stops locations and names to test my program with. Unfortunately, they already had Data Sharing agreement between Stagecoach, the council and the Real Time system developer.

*Dear Mateusz,*  
 *I am in receipt of your email regarding the request for our data source for your developing app.   
  
I do apologise for the delay in responding to you directly.   
  
Unfortunately we have a pre agreed Data Sharing agreement between Stagecoach, the council and the Real Time system developer which does not allow for such data to be sent to any other individual.   
  
In relation to bus stop information, the best source of data I would suggest is google maps, as if you zoom in to street level, the bus stops all show up with a blue and white symbol. If you click on that, it will bring up the service number of the buses which serve that stop. This would appear to be the information you are looking for.   
  
I trust this information is of use to you.  
  
Yours sincerely,****Zoë Paget*** *Operations Director  
Stagecoach East*

# Design

## Introduction

This is the design section of the project, where the functionality, classes and processes of the program are described as they were designed.

## IPSO chart

This chart outlines the processes in the program at a basic level, in terms of input/output, processing, and storage. A full detail of these processes is described below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Processing** | **Storage** | **Output** |
| Departure bus stop  Arrival bus stop  Time and date  Arrive before or depart after time option  Bus Change Risk\* | Retrieve appropriate timetable from the database  Calculating the shortest journey time  Drawing interactive map  Create a printable document | Database tables:  *BusStops*  *Buses*  *Links*  *Timetable*  *Day Categories*  *Special Days* | Digital document  Printed document  Interactive map |

\*Bus Change Risk is the time gap in minutes between bus changes.

## Storage

The database structure has changed slightly since it was in the analysis. A table of bus links has been added. This it makes it easier to edit data as only a link has to be changed rather than the whole route when uploading to a database. A table for special days has been added as well, this table will include dates for bank holidays and other days when services run in a different way than to normal.

### Entity Relation Diagram

Here is an entity diagram which shows the basic relationship between the tables in the proposed system. This is only a very basic design for a database to act like a Stagecoach database.

**Buses** - bus networks usually have two routes for one bus name; one is usually the reverse of the other. For the simplicity of the program, the routes are going to be treated as two separate buses. This format does not constraint the use of double route buses however.

**Bus Stops** – usual format for storing information about a bus stop in a database

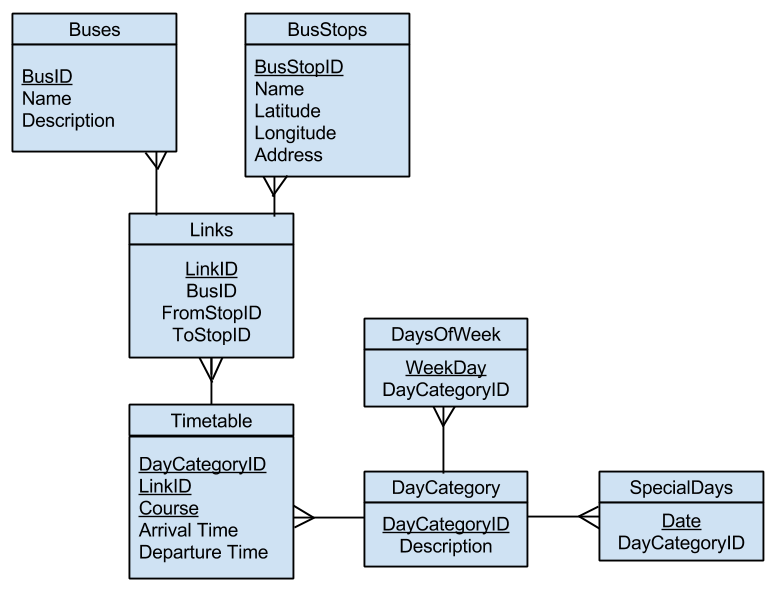
**Links** – (new as from design) – this corresponds to the edge between two bus stops. A link can only reference one bus; so if there are more buses which go through the same edge a new record in the Link table has to be created.

**Special Days** – will refer to special dates that have a non-standard timetable, like bank holidays.

**DaysOfWeek –** will store a day category for day of week.

**Day Category** – will be reference by the Timetable dependent on the days of the week.

**Timetable** – one for the whole bus network

**** We can tell the data has been normalised because there are no many-to-many relationships

## Data Volumes

### Background

This is an improved calculation of the database data. Although data is not going to be inserted into the database using the software validation is also included.

### Data Dictionary and Data Volumes

Here is a basic design for the database to get an idea of how much storage would be required for the system.

**Bold** indicates a primary key, or in the case where there are several primary keys, they make up a composite key.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Buses (about 66 buses) | | | | |
| **Columns** | **Type** | **Size** | **Bytes** | **Validation** |
| **BusID** | Integer | Up to 66 | 1 | Not null |
| Name | String | 20 chars (ASCII) | 20 | String up to 20 chars |
| Description | String | 200 chars (ASCII) | 200 | String up to 200 chars |
| Number of bytes for each row | | | 221 |  |
| Total kB | | | 14.6 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bus Stops (about 2000 bus stops) | | | | |
| Columns | Type | Size | Bytes | Validation |
| **Bus Stop ID** | Integer | Up to 2000 | 2 | Not null |
| Name | String | 80 chars (ASCII) | 80 | String up to 80 chars |
| Latitude | Decimal | Between -90.0 and 90.0 | 4 | Valid latitude number |
| Longitude | Decimal | Between -180.0 and 180.0 | 4 | Valid longitude decimal, up to 6 decimal places |
| Address | String | 200 chars (ASCII) | 80 | Valid address, postcode and street name |
| Number of bits for each row | | | 170 |  |
| Total kB | | | 1800 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Links (about 8000 links) | | | | |
| **Columns** | **Type** | **Size** | **Bytes** | **Validation** |
| **LinkID** | Integer | Up to 5000 | 2 | Not null |
| BusID | Integer | Up to 66 | 1 | Not null |
| BusStopAID | Integer | Up to 1000 | 2 | Not null |
| BusStopBID | Integer | Between -180.0 and 180.0 | 4 | Not null |
| Number of bits for each row | | | 9 |  |
| Total kB | | | 72 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DaysOfWeek (Mondays to Saturdays, Sundays and Bank Holidays) | | | | |
| **Columns** | **Type** | **Size** | **Bytes** | **Validation** |
| DayCategoryID | Integer | Up to 9 | 1 | Not null |
| **Week Day** | String | 30 chars (ASCII) | 20 | Day of week, i.e. Monday |
| Number of bits for each row | | | 21 |  |
| Total kB | | | 0.18 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Special days (about 20 special days in a year) | | | | |
| **Columns** | **Type** | **Size** | **Bytes** | **Validation** |
| **Date** | Date | Variable | 4 | Not null, valid date type format |
| DayCategoryID | Integer | Up to 9 | 1 | Not null |
| Number of bits for each row | | | 5 |  |
| Total kB | | | 0.1 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Day Categories (Mondays to Saturdays, Sundays and Bank Holidays) | | | | |
| **Columns** | **Type** | **Size** | **Bytes** | **Validation** |
| **Day Category ID** | Integer | Up to 9 | 1 | Not null |
| Description | String | 30 chars (ASCII) | 20 | None |
| Number of bits for each row | | | 21 |  |
| Total kB | | | 0.18 |  |

Timetable (70 bus stops on average for each bus route, 66 routes, on average 60 bus courses every day for each bus route, 3 day categories, and therefore about 830000 rows of data altogether)

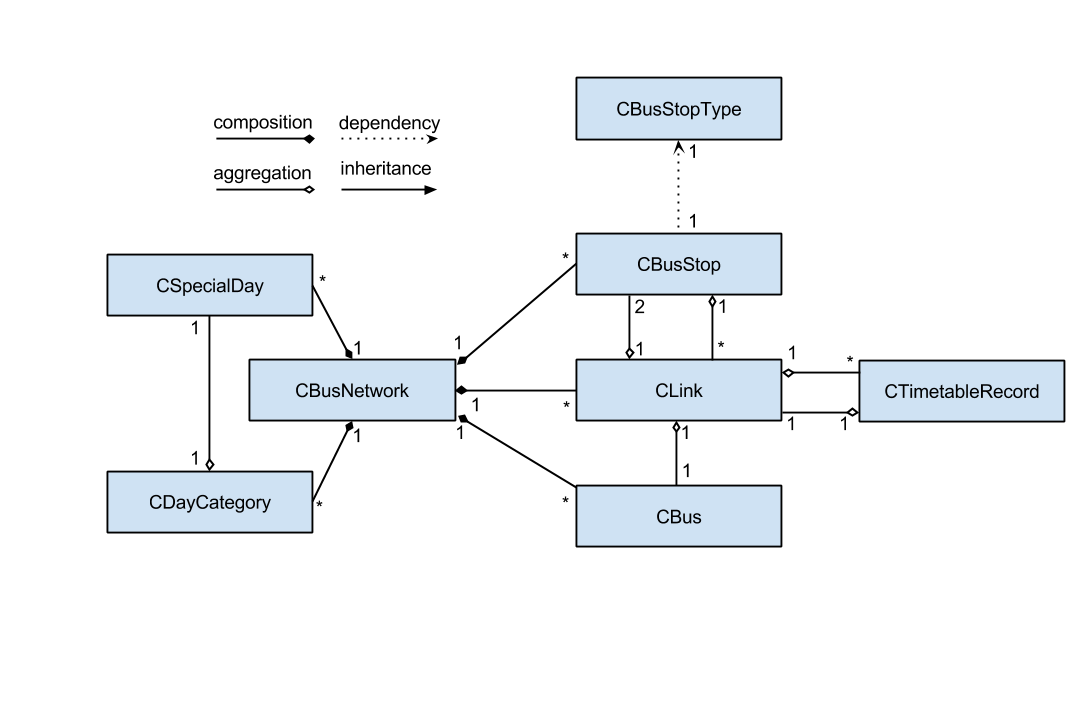
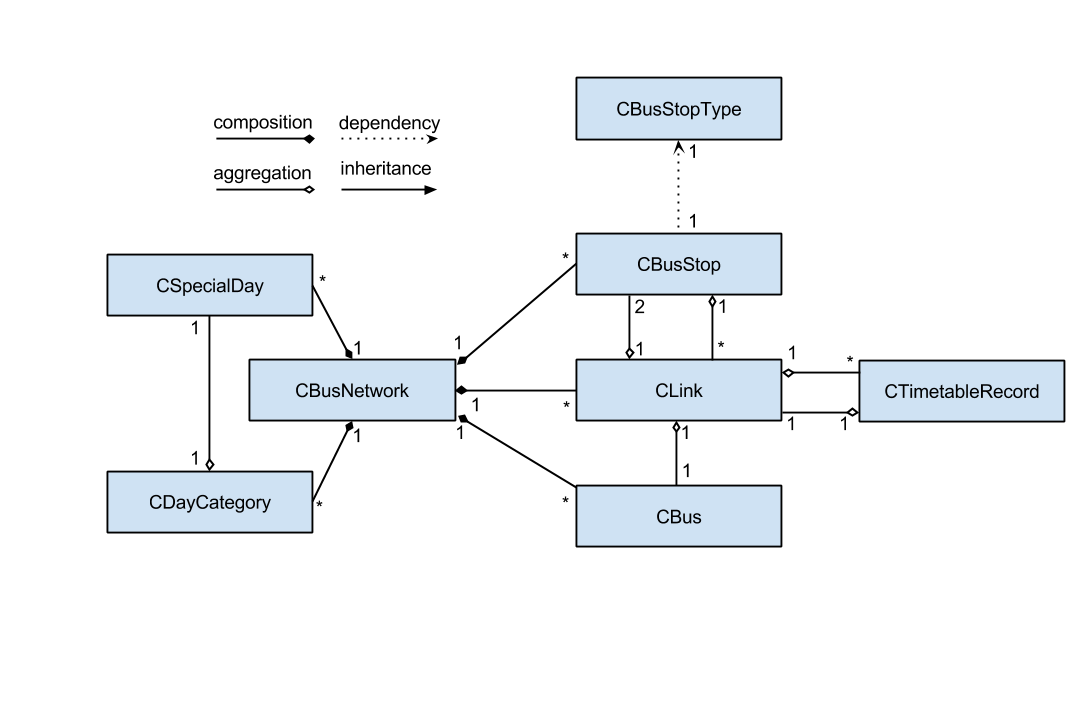
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Timetable | | | | |
| **Columns** | **Type** | **Size** | **Bytes** | **Validation** |
| **DayCategoryID** | Integer | Up to 9 | 1 | Not null |
| **Bus Course** | Integer | Up to 99 | 1 | Not null |
| **Bus Stop ID** | Integer | Up to 2000 | 2 | Not null |
| Time | Integer | 0000 to 2400 | 2 | Not null |
| Day Category | Integer | Up to 9 | 1 | Not null |
| Number of bits for each row | | | 7 |  |
| Total kB | | | 5800 |  |

All together: about 7700kB, which means that the size of the database would be about 7.7Mbytes. This is still very close to the initial estimation of ... Mbytes.

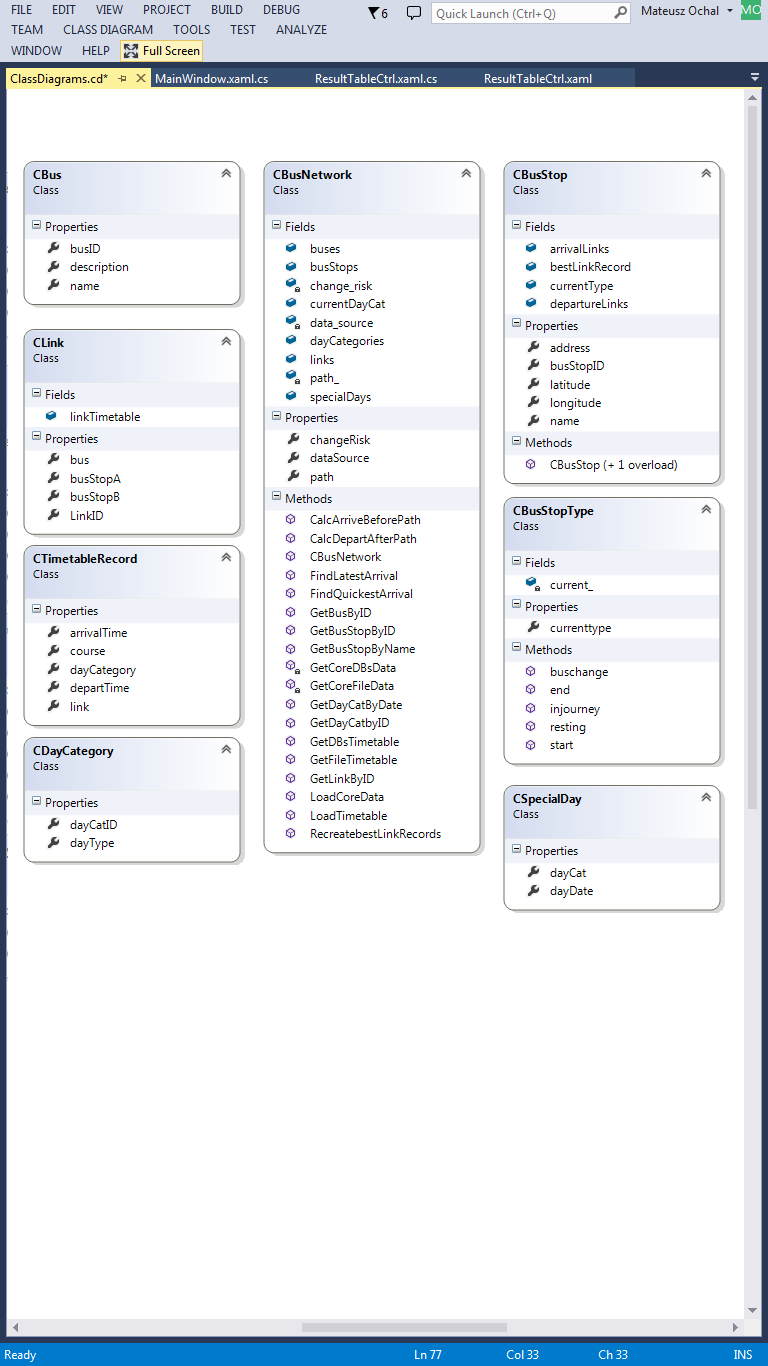
## CBusNetwork Class

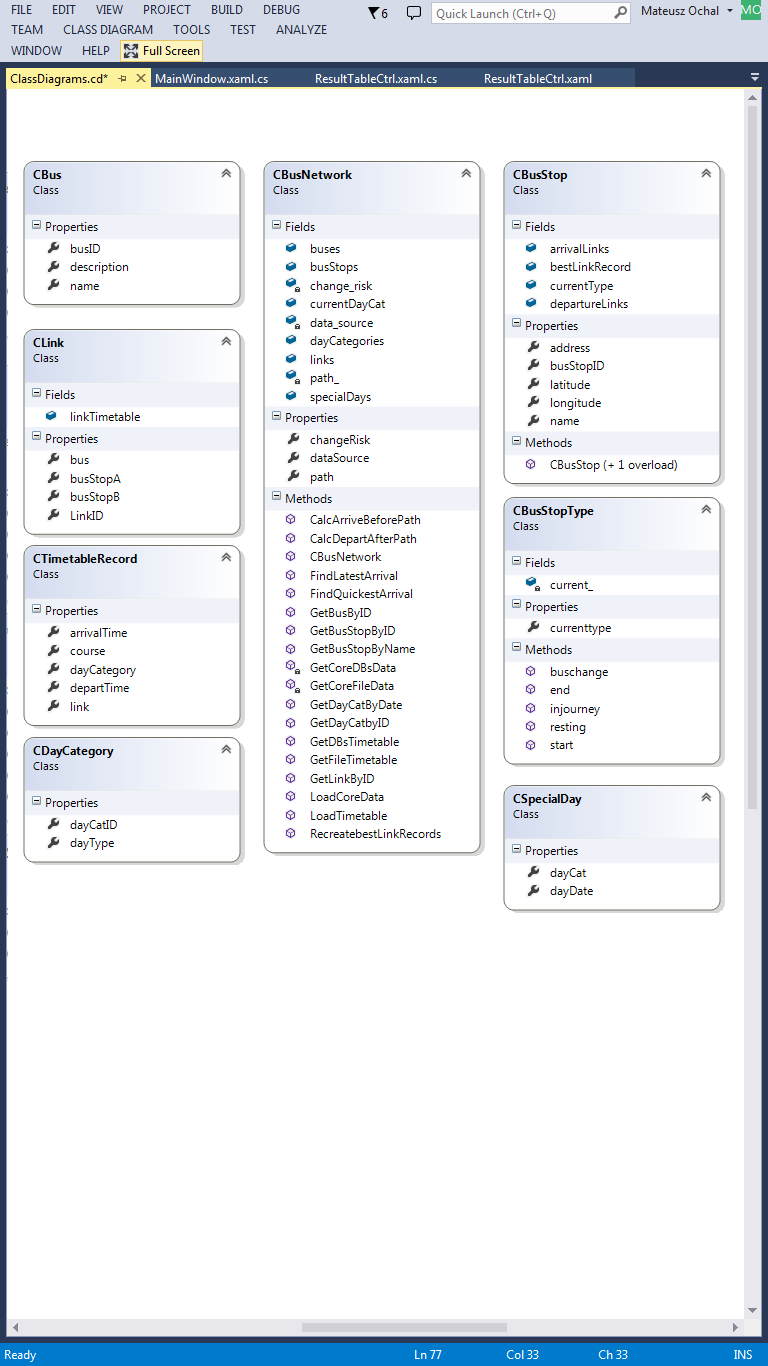
Bus Network Class will be based on the database entities, so that data can be directly loaded into the memory and easy to work with.

Bus Network Class will be designed to be independent of the rest of application. The idea is that this class would only deal with communication to a data source and algorithm calculations and not be dependent on the user interface. This is to make the class reusable with different applications.

Below is an entity diagram showing the relationship between each class. This is a more detailed design compared to the one included in the analysis. The description of what each class contains is described below.  
(NB. The program uses more classes which will be described later under the Design section)

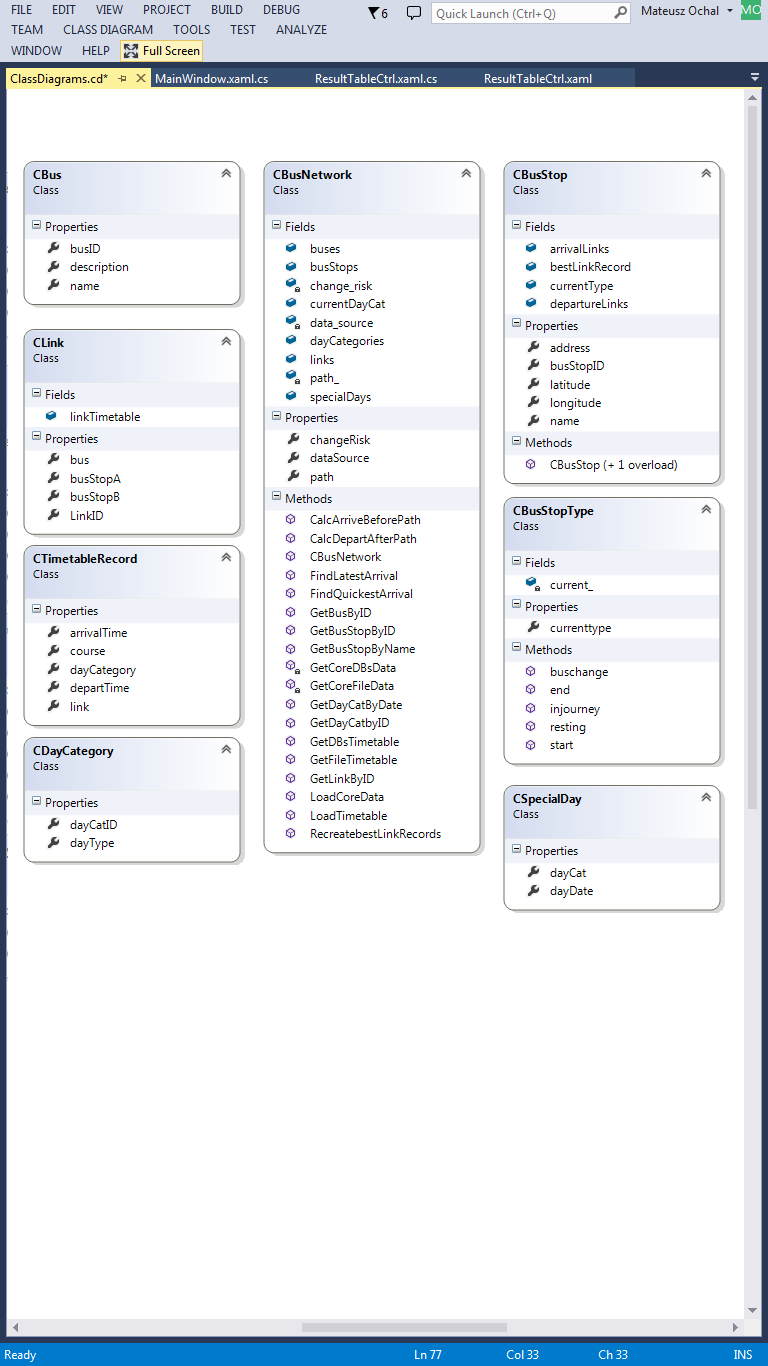
Double linking indicates a ‘recursive’ relationship. For instance an instance of CLink will reference two instances of CBusStop, the two bus stops that link exists between. However an instance of CBusStop will reference multiple instances of CLink, the links coming from and to that bus stop.

**CBusNetwork** – is the main class which will be visible to the rest of the application. It will contain all information about the bus stop, buses, links and their timetables, and special days and day categories. An instance of CBusNetwork class will be able to read data from a data source, like a database or a text file and turn that data into instances of the other objects.

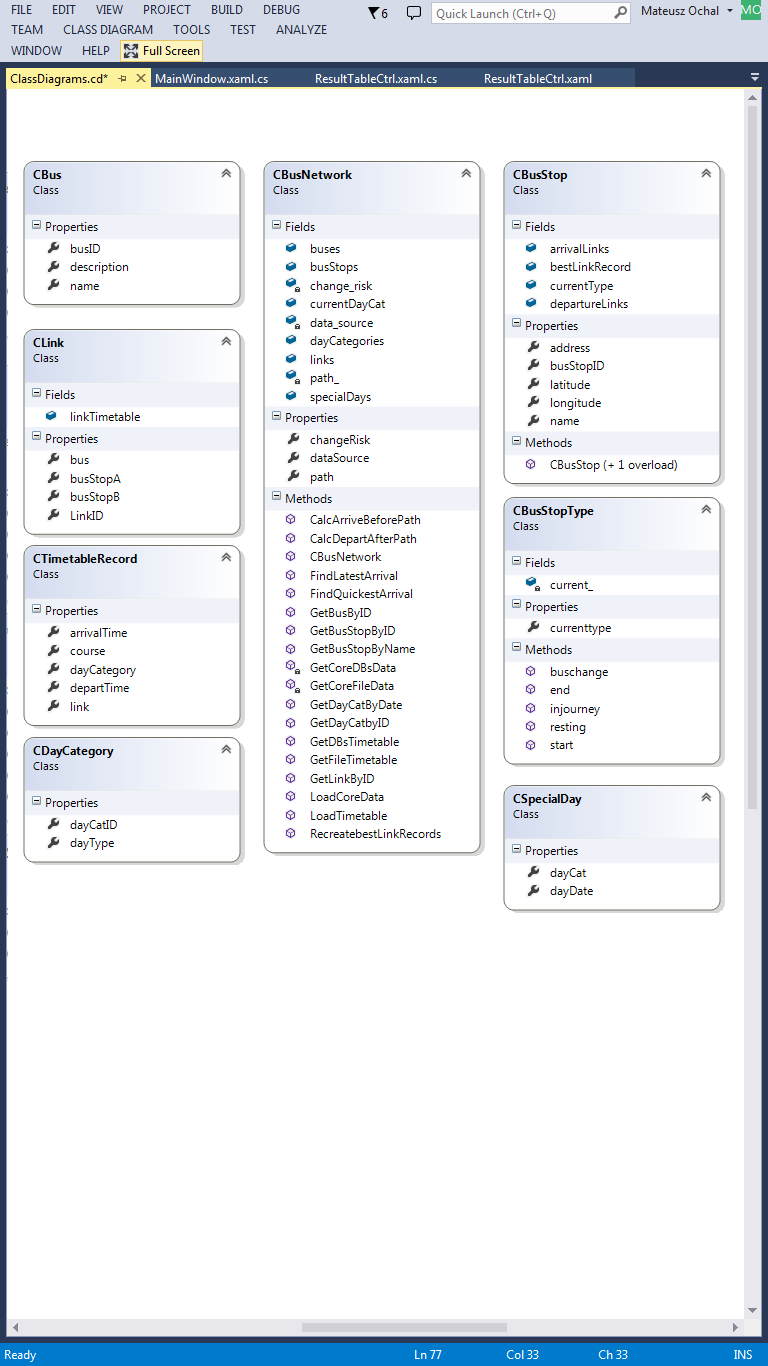
In the class diagram to the left, the fields contained by the class include a list of buses, bus stops, links, day categories and special days. These correspond to the classes in the relationship diagram.

changeRisk – property - is for algorithm use described later.

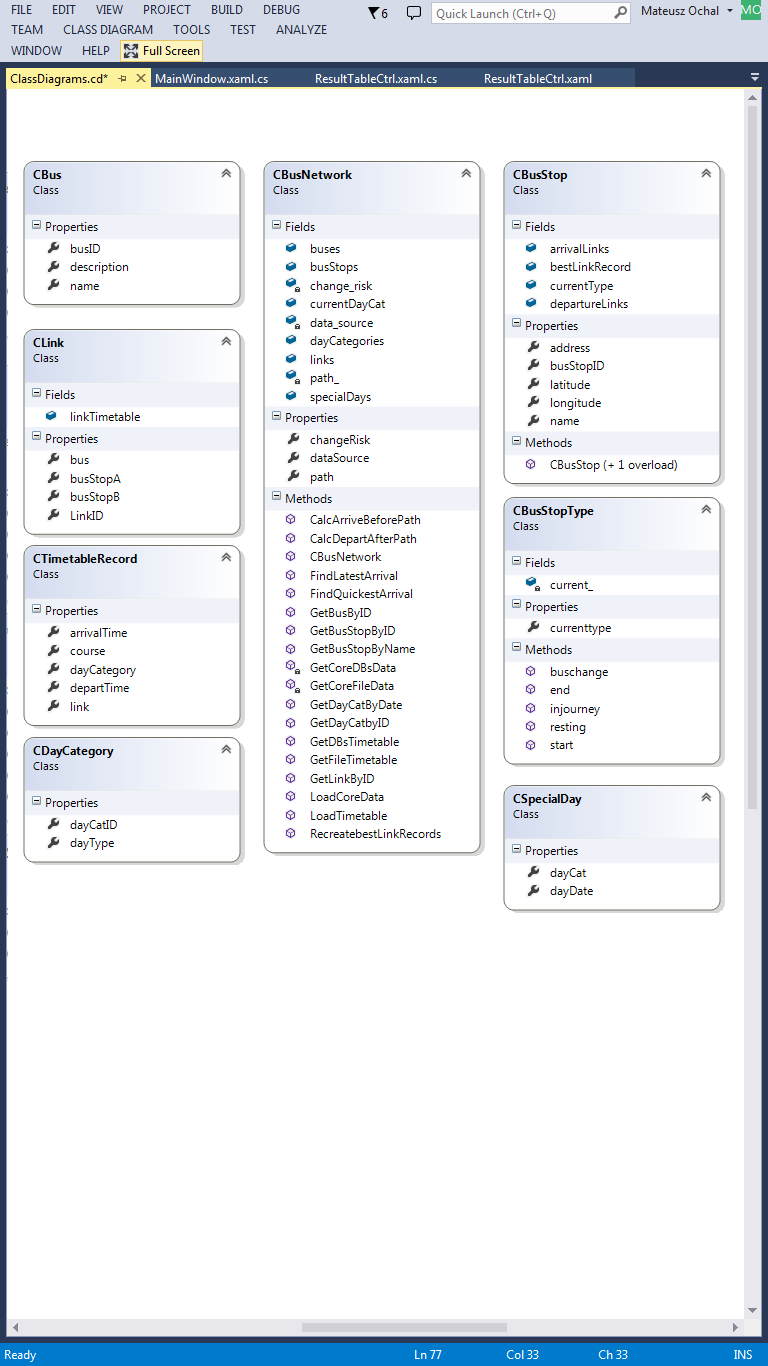
dataSource and path – properties – will store the source and the path to connect to it. For a textfile, path will be the directory. For a database, path will store connection string to the database. A more detailed description of how database connection works will be described in the Systems Maintenance section.



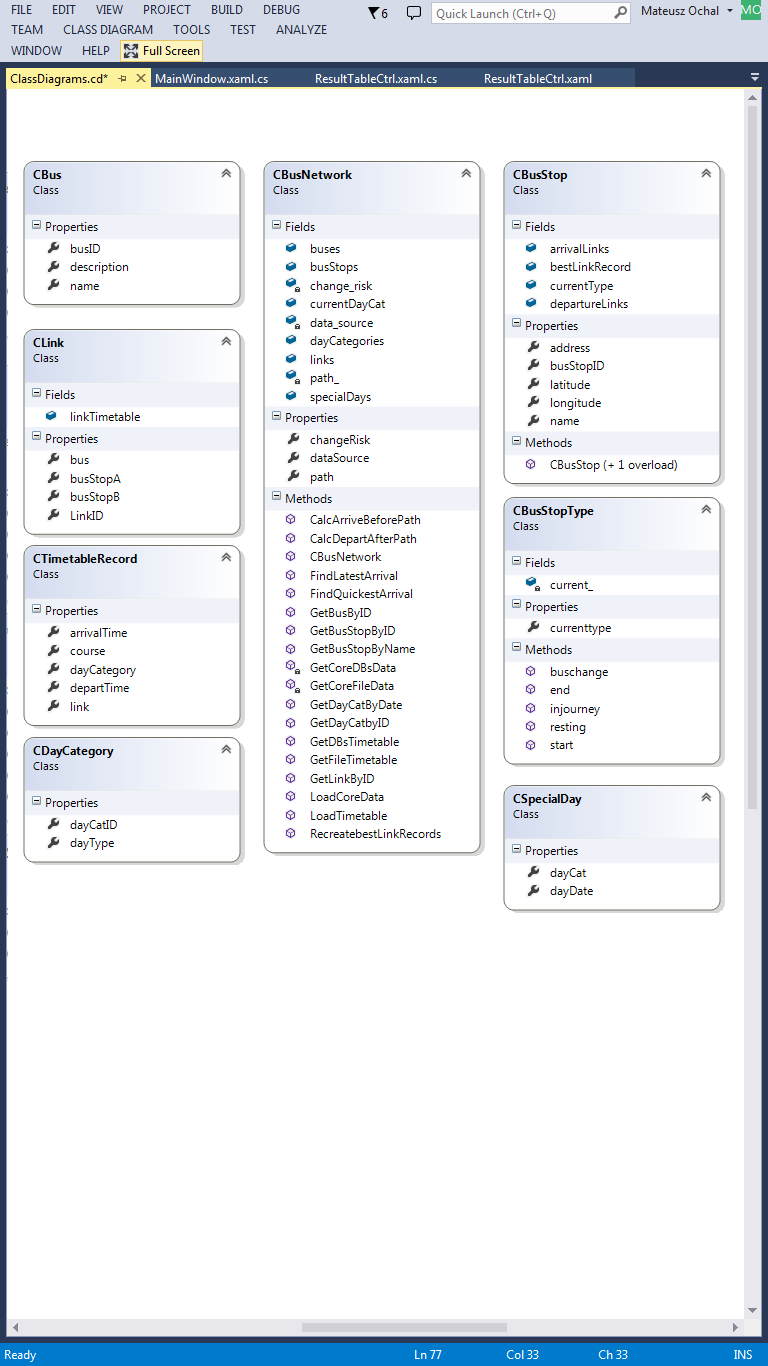
**CBus** – corresponds to the bus table in the database.



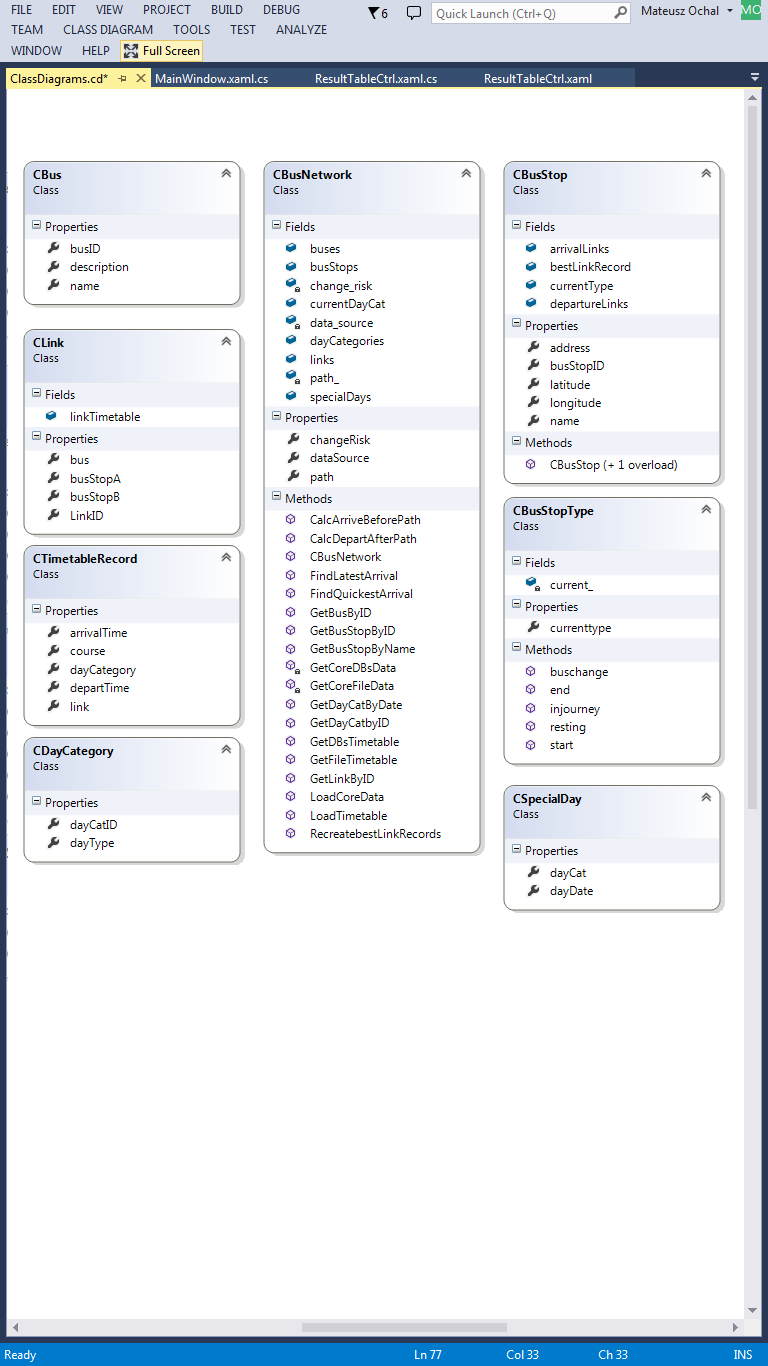
**CBusStop** – corresponds to the bus table in the database. Only the properties of this class represent the columns in the database/ text file. The fields represent extra information about the bus stops, for instance the links going in and out of the bus stop (arrivalLinks/departureLinks).



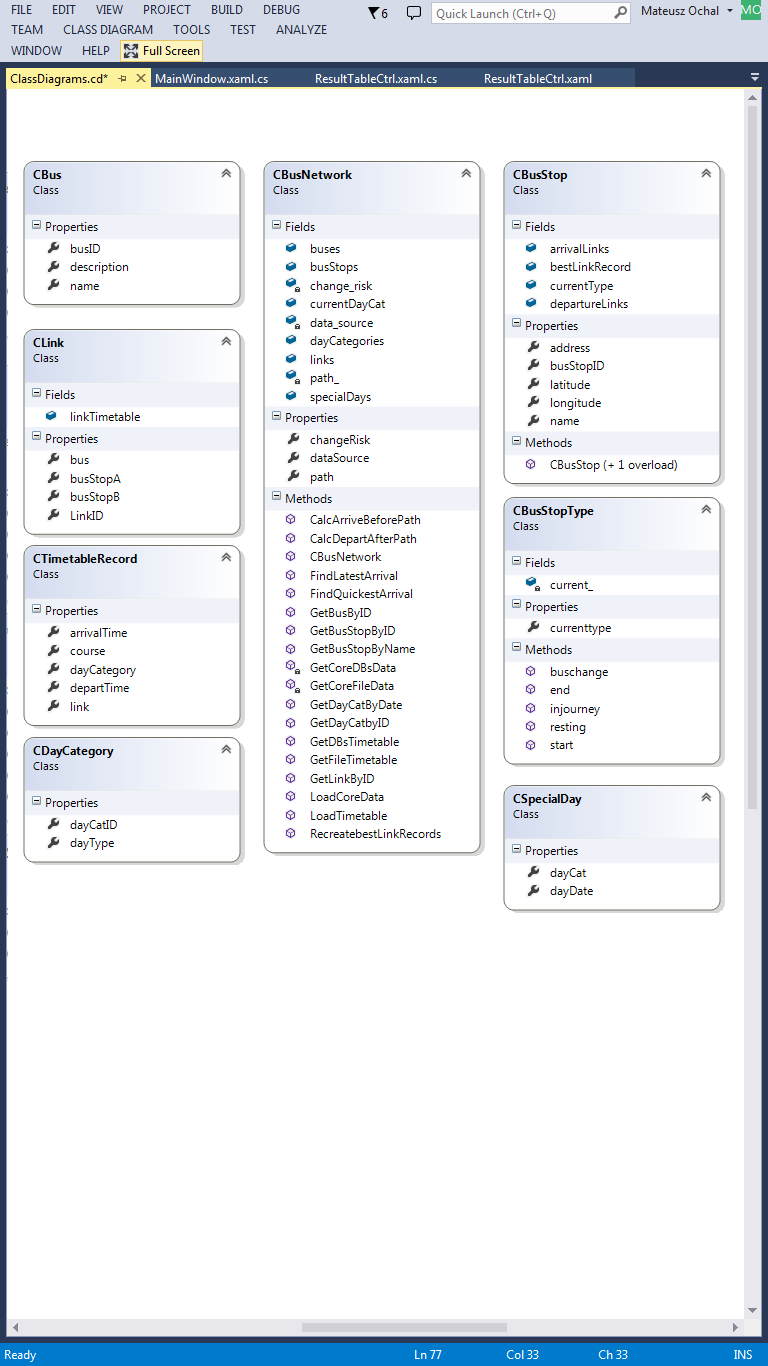
**CLink** - corresponds to Link table in the database. Each Link contains a corresponding timetable.



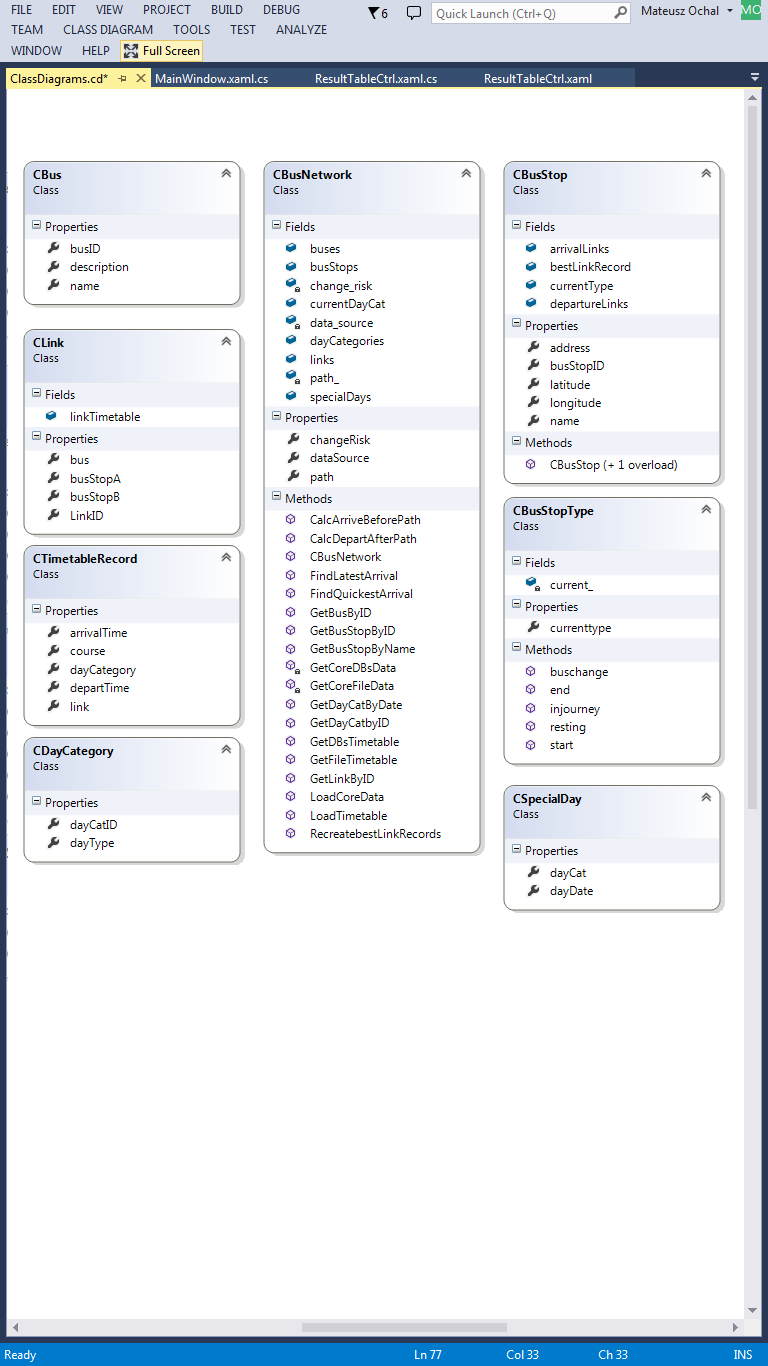
**CTimetableRecord** – is a representation of row/record in the timetable table in the database. It references the link to which it belongs to.



**CDayCategory** corresponds to the records in DayCategories table in the database.



**CSpecialDay** -corresponds to the records in the SpecialDays Table in the database.



This class will identify the type of the bus stop for a give point in time. This class will be used by the visual components to colour the bus stops appropriately. The instance of this class will only operate on strings and not depend on other types, so this makes the CBusNetwork Classes completely independent to the graphical user interface. The CBusStop class is dependent on this class but it not essential for it to work.

## Algorithms

To calculate the quickest journey between two bus stops, the program will use alteration of the well known Dijkstra’s Algorithm which solves shortest path problem for weighted graphs. Firstly I will describe the original algorithm before proceeding to the actual algorithm used by the program to help explain the subtle differences.

### Original Dijkstra’s Algorithm

Original Dijkstra’s Algorithm calculates the shortest distance between two nodes with fixed weight edges. Each node in the graph contains two variables: distance to source and a pointer to previous node that linked with node. A function which calculates the shortest path using Dijksta’s algorithm, takes in two parameters: graph (containing all nodes) and source node. Here is the pseudo code for the algorithm:

**Function Dijkstra(Graph, source):**

*//Initialisation*

*// dist[node] is the distance from source to node*  
**dist[source] ← 0***// dist[source] is the distance from source to source*

*// prev[node] is the previous node that linked with the node*  
**prev[source] ← undefined***// in this case: prev[source] is the previous node that linked with source*  
  
*// For the rest of the nodes in the Graph*  
 **for each vertex v in Graph:  
 if v ≠ source  
 dist[v] ← infinity** *// The distance from source is set to be very large*

**prev[v] ← undefined** *// The previous node is set to be undefined* **end if  
 add v to Unvisited**  
 *// Add all nodes initially to Unvisited (nodes)* **end for**

*//Calculating* **while Unvisited is not empty:** **u ← vertex in Unvisited with min dist[u]** *// Source node in first case*

**remove u from Unvisited**

**for each neighbour v of u:** *// where v is still in Unvisited* **alt ← dist[u] + length(u, v)**   
 *//length is the fixed weight between the nodes  
 // alt is the new length calculated  
 // Comparison Section* **if alt < dist[v]:***// i.e. a shorter path to v has been found* **dist[v] ← alt  
 end if   
 prev[v] ← u  
 end for  
 end while  
 return dist[], prev[]  
end function**

### Depart after and arrive before algorithm

The program will use two algorithms based on Original Dijkstra’s (explained at the beginning of the Algorithm’s section):

‘Depart After’ algorithm – looks for an optimal journey after a given time

‘Arrive Before’ algorithm – looks for an optimal journey before a given time

##### Small analysis of timetabled problem

First of all, here are some problems associated with calculating the shortest path using timetabled graph by the algorithm described earlier:

|  |  |  |
| --- | --- | --- |
| Problem | Description | Addressing the problem |
| Timetabled edges | Edges exist depending on time. A timetable usually consists of a departure time of a bus from a bus stop and an arrival time of the bus to the next bus stop. | The algorithm could be made to compare the arrival time and departure time without adding the times or their differences |
| Waiting time for a bus | Buses don’t necessarily leave at the same time and waiting may be required to get a bus. | If Comparison Section looks only at the arrival time at the next bus stop it would not matter what the departure time was from that bus stop; so long as the bus arrived earlier at the next bus stop |
| Bus change over time | This problem cannot be over come without changing the complexity of the original algorithm | The program will have a bus change over time/risk |

##### New features to take note of

* Departure time and arrival time – this is a substitution for length()
* Best Link – substitution for prev[] and dist[]
  + Based on CTimetableRecord class, it includes:
    - arrTime - Arrival time
    - depTime - Departure time
    - course - Course number
    - link - Link between two bus stops (based on CLink class), includes:
      * bus
      * busStopA
      * busStopB
* Bus Change Risk – additional parameter to be included in the calculations
* Time – this is the initially specified time.

The classes and database structure have been designed to operate around these problems and introduce these features into the algorithm.

The pseudo code of ‘Depart After’ algorithm:

**Function DepartAfter(Graph, source, destination, time):**

*//Initialisation*

*// for the rest of the nodes in the Graph* **for each vertex v in Graph:**

*// resets bestLink to default* **bestLink[v].arrTime ← undefined  
 bestLink[v].depTime ← undefined  
 bestLink[v].course ← 0  
 bestLink[v].link ← undefined**

*// add all nodes initially to Unvisited (nodes)* **add v to Unvisited**

**end for**

*// arrival time of best link of [source] is set to the given time* **bestLink[source].arrTime ← time**

*// Calculation* **while (Unvisited is not empty) and (v is not destination):**

**u ← vertex in Unvisited with min bestLink[v].arrTime  
 remove u from Unvisited** *// u is source in the first case*

*// for each link u has* **for each CLink link in u:**

*// for each record in timetable* **for each CTimetableRecord record in link.timetable:**

*//Comparison Section  
 //excludes all records that depart before bestLink[u].arrTime  
 // replaces the bestLink with record if it is ‘better’* **if record.depTime is later than bestLink[u].arrTime: if record.arrTime is less than BestLink[link.busStopB].arrTime:**

**bestLink[link.busStopB] = record**

**endif  
endif**

**end for**

**end while**

**return bestLink[]**

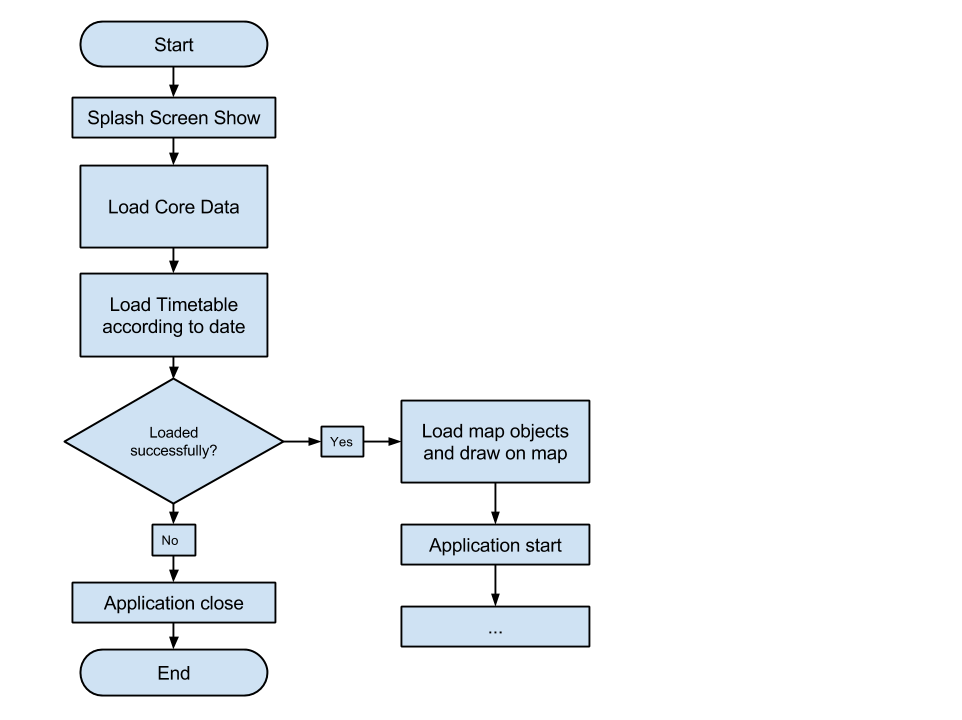
**end function**

### Space and time complexity of the algorithm

Original Dijkstra’s algorithm has a O(N2) time complexity, where N is the number of nodes in the graph. I decided to load the data into memory to increase the speed of the calculation.

## System Flow Diagram

To optimise the time of calculation of the algorithm the data is loading into memory. I predicted that objects do not take up more than 10 Mbytes of random access memory. The data would be loaded at the start up of the program and kept in memory until the main form is closed. Here is a flow diagram representing the flow of processes in the new proposed system.

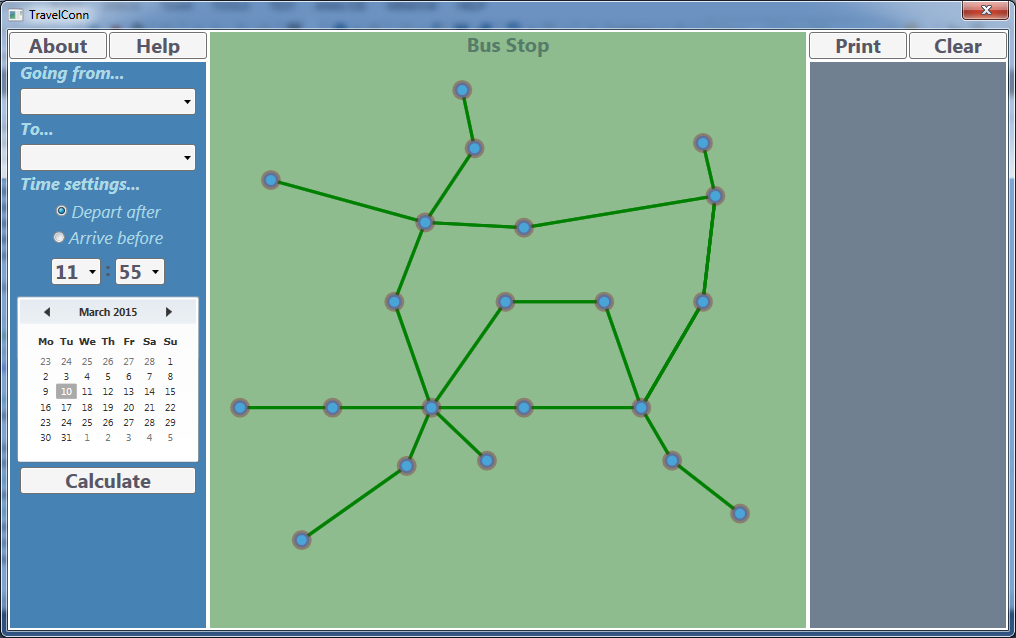


Attempts to create objects from the data that it is loading

## User Interface Design

Here are some proposed user interface layouts.

### Design 1: (Screenshot of a prototype)

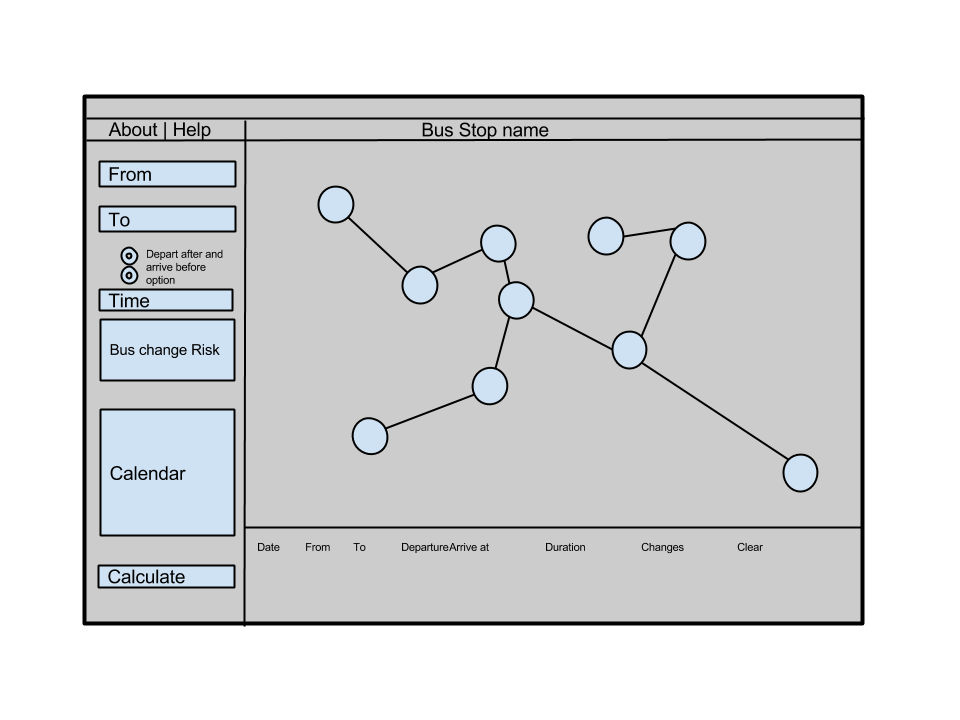


(2)

(3)

(1) 

### Design 2: Sketch of an improved design



(3)

(2)

(1) 

### Interface description and interaction

The design proposals share 3 common interface features:

Search panel – here the user chooses the start of journey bus stop and the destination bus stop, time and date of the journey and bus change risk. The user can select bus stops from a drop down menu, avoids spelling error of the user.

Feedback table – the user will be able to see and compare calculated bus journeys here. Basic information concerning the departure time and the arrival time at the destination bus stop and the number of changes. The user will be able to choose to display more information and print or save it.

Map – an abstract map - the user can also choose the bus stops as well and see the results of the calculated bus journey by highlighted path. The map should be interactive and highlight the bus stops as the cursor is moved over then. The map will also have a zoom in and out feature so that big networks can fit in the space provided.

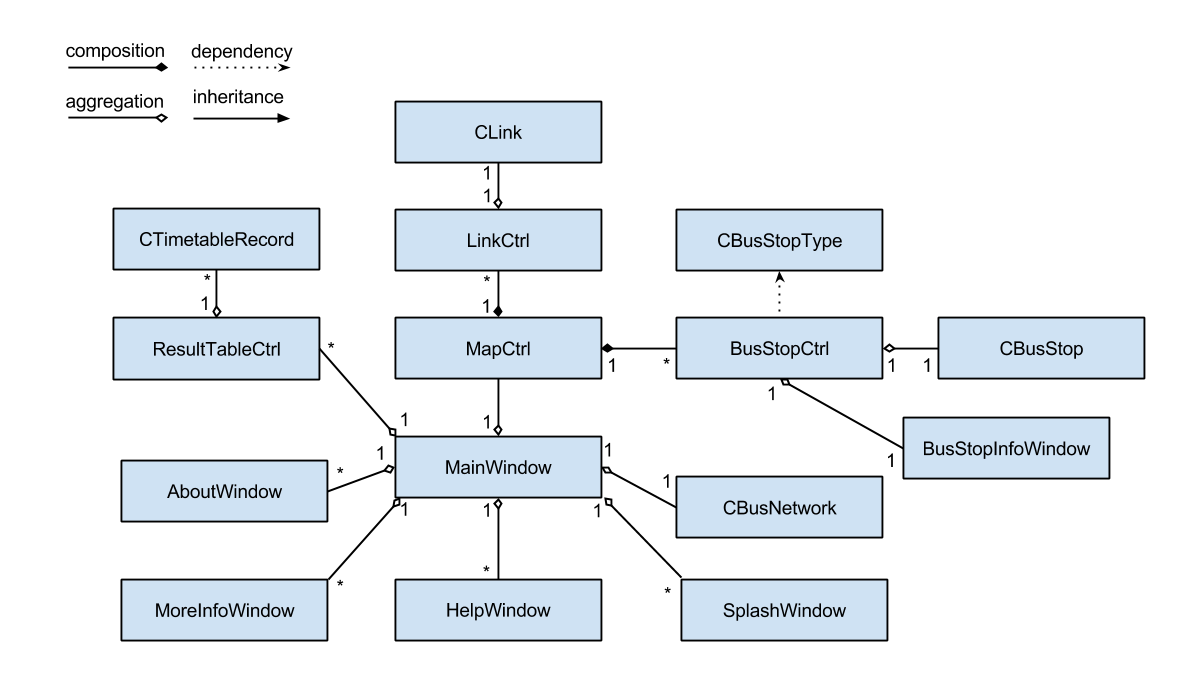
A map will have visual components corresponding to the links and bus stops in the instance of CBusNework class.

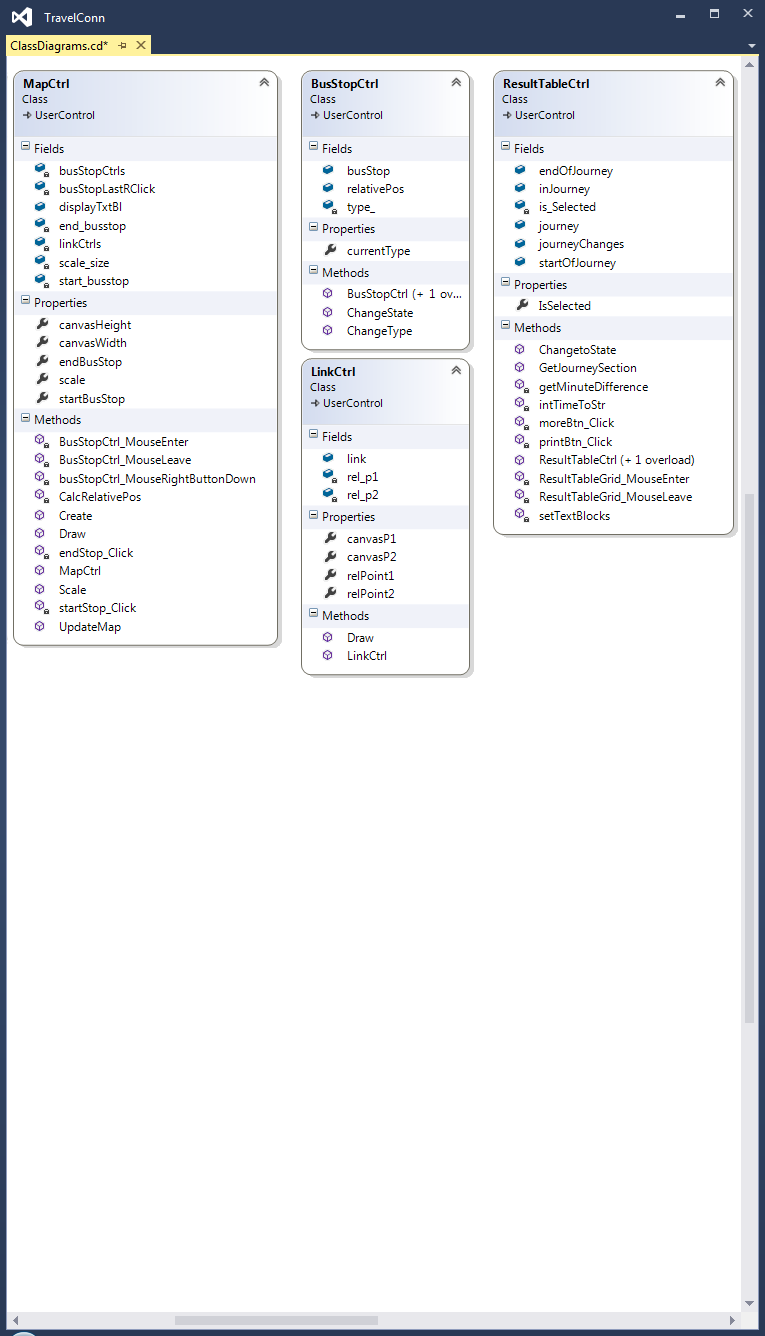
Design 2 is more practical as the user will be about to see more information about the journey at once; making it easier to compare journeys.

## Visual Component Class Diagrams

In this section, other classes and components are described, focusing on the visual components of user interface.

Here is a class diagram with all classes. (N.B. CBusNetwork has more links which are not visible in the diagram below but where described earlier in the Design Section).

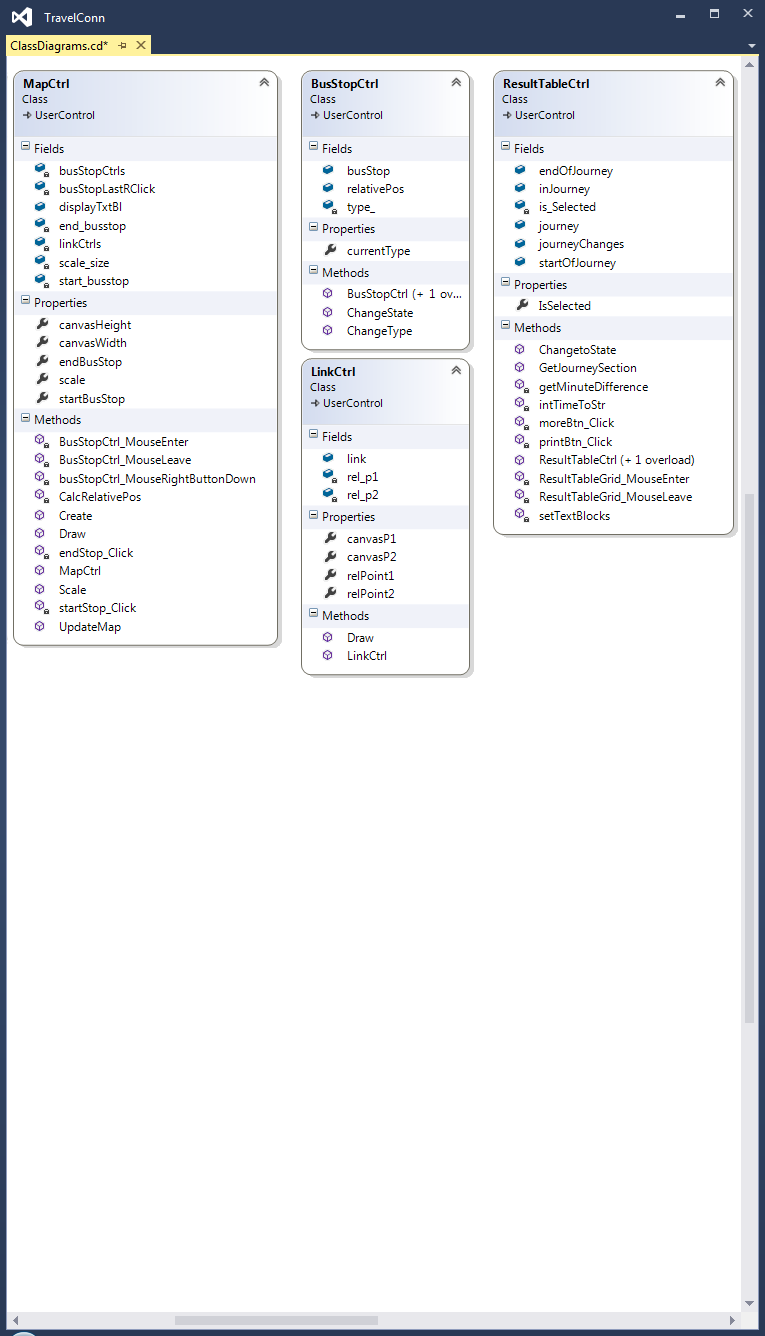


Classes explained

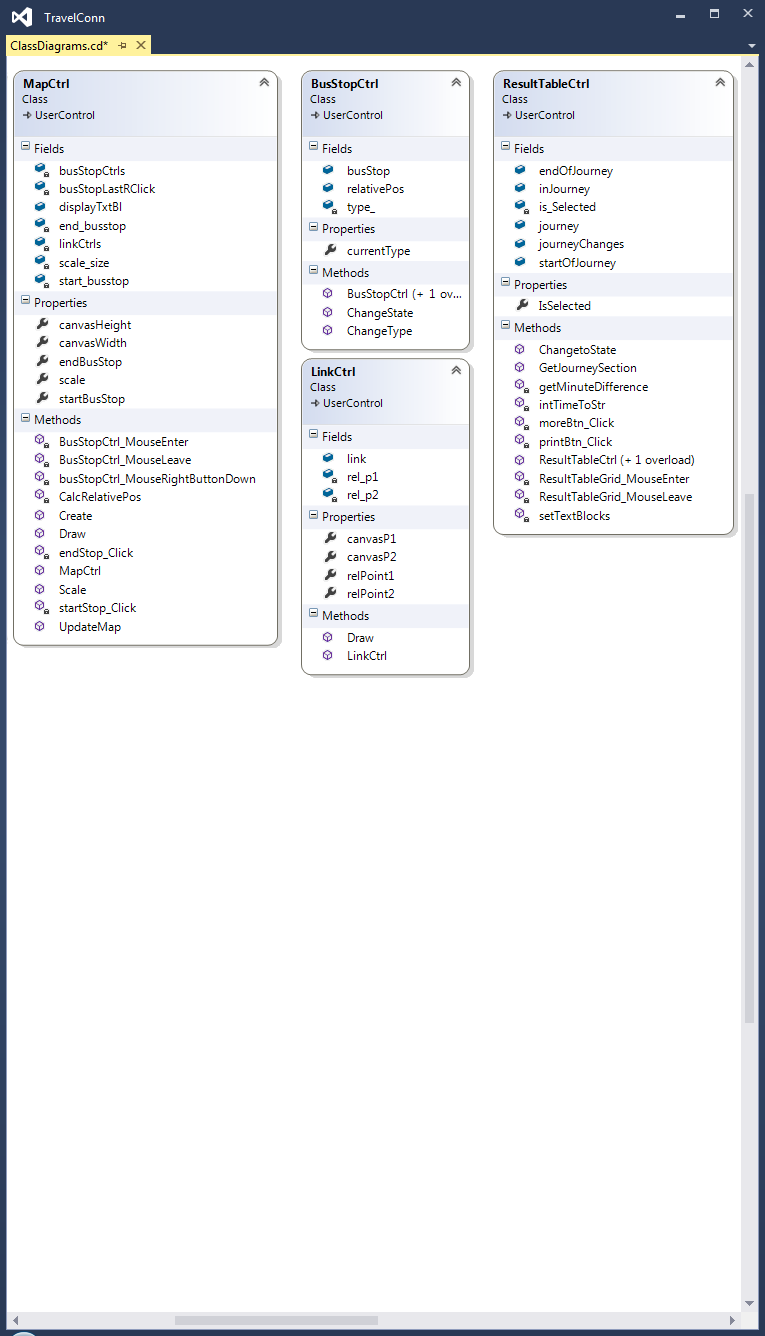
### Class relationship

MapCtrl – visual representation of the bus network

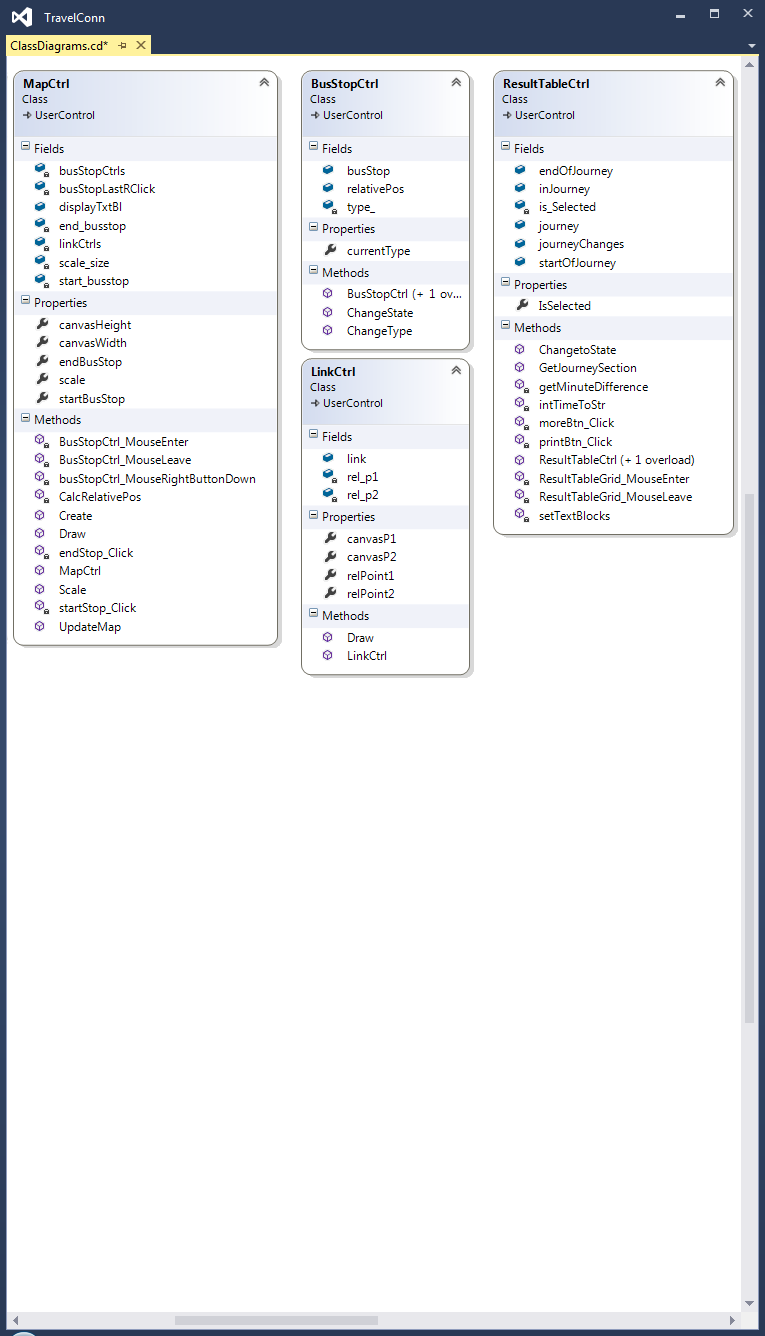
Contains all visual bus stops and links

BusStopCtrl – visual representation of the bus stops

An ellipse that highlights when mouse over

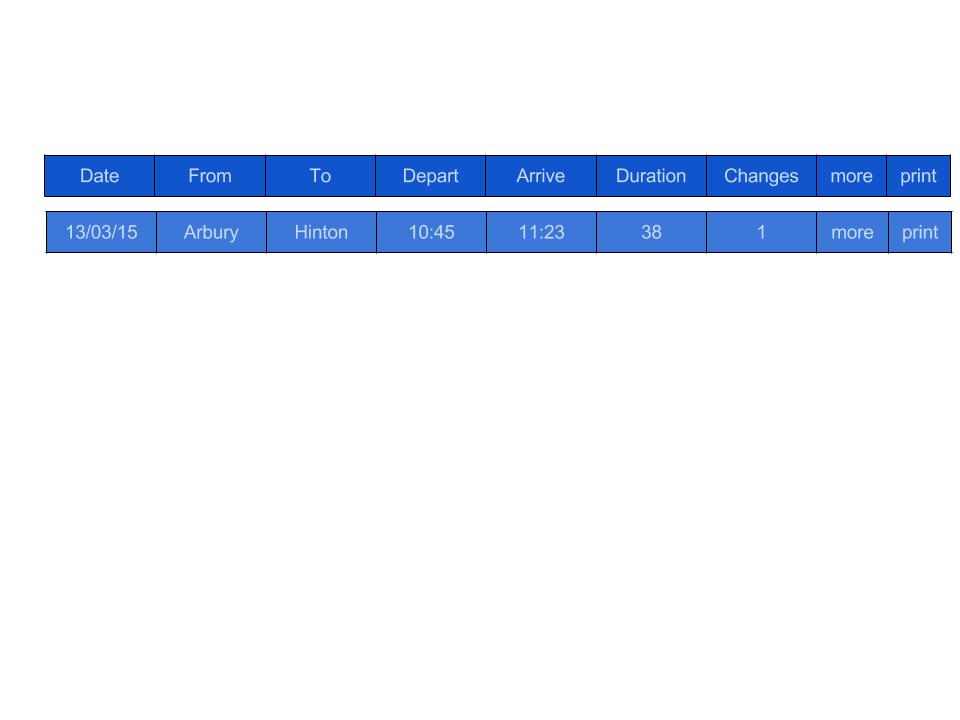
LinkCtrl – visual representation of the link

A straight line or a curve to indicate different links



ResultTableCtrl – visual result, easy to compare with eachother

### ResultTable design



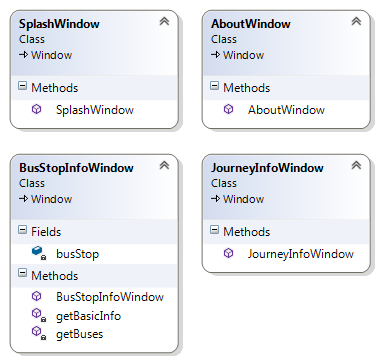
Examples:



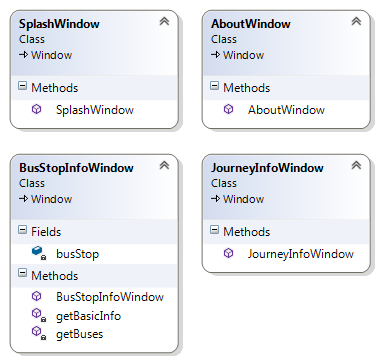
This way it will be easy to compare results

‘more’ – is a button to show more information

‘print’ – is a button to print the information

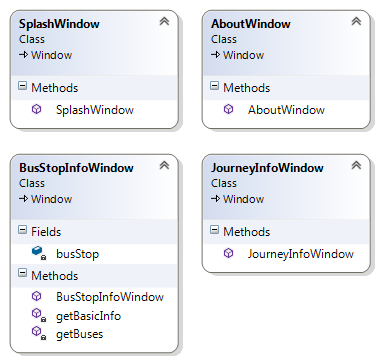


Journey Info Window will be displayed when the ‘more’ button is clicked. The window will contain text mostly.

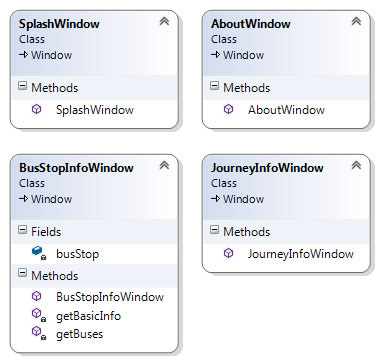
BusStopInfoWindow - The user will be about to get more informaiton about a bus stop. This window will grab the information stored about the bus stop from database or memory if already loaded. The window will contain text mostly.

The functions getBasicInfo will load a bus stop’s name, location and address

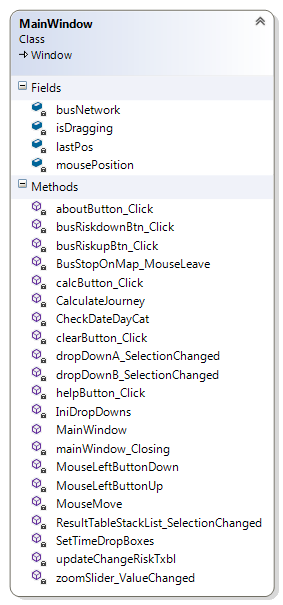
getBuses will get the buses departing to and from the bus stop



About window will provide general informaiton about the program, will contain text mostly.



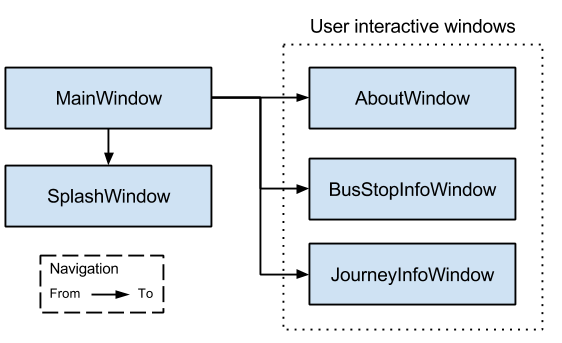
Splash window will be displayed when the data is being loaded. This window will not be interactable in any way – it should just inform the user that something is happening.



Main Window Class

This class contains absolutly all components of the program. It contains the start up function.

# Window Navigation

This section gives an outline of the from navigation interface design. The diagram shows that any window can be accessed through the main window. 

The detailed design of the main window is outlined in earlier in the section. The user interactive windows will be opened by the user when they want. These windows will most probably contain plain text or tables.

JouneyInfoWindow exampleThe printable document will be of similar format. This format clearly shows which bus to take, when to get off or on.

BusStopInfoWindow design (example)

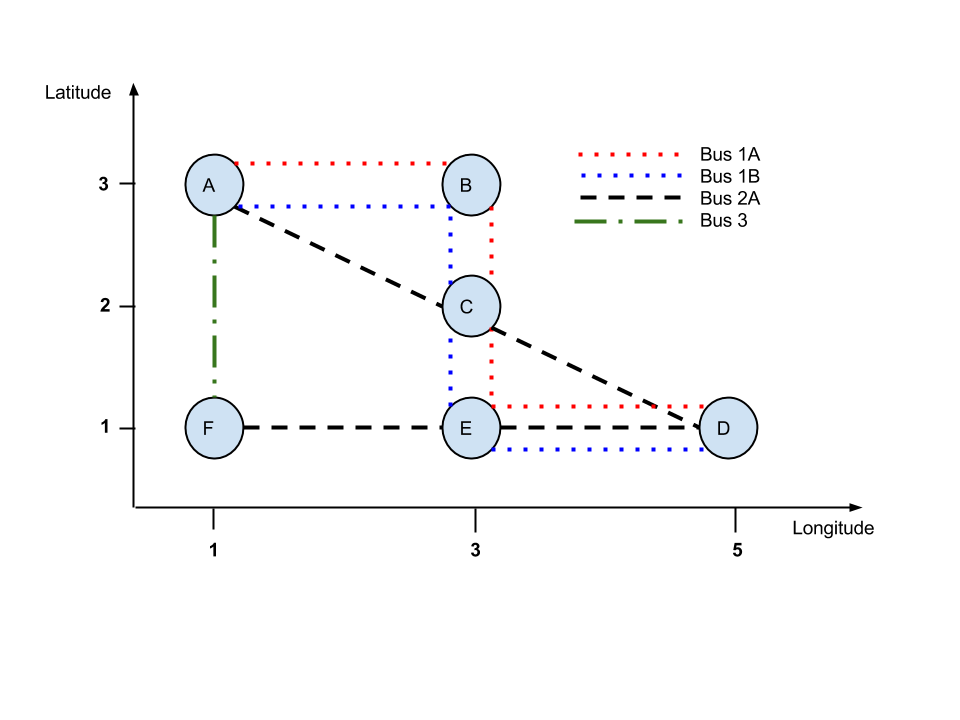
This format of design clearly shows the buses that go to and from the bus stop

## Test Plan - Introduction

Bus Network Designs for testing. These are designed especially to test extreme cases and they will test the basic principle. The test cases will be outlined later in this section.

#### Small Network

Bus stops are represented as letters of simplicity and convenience.



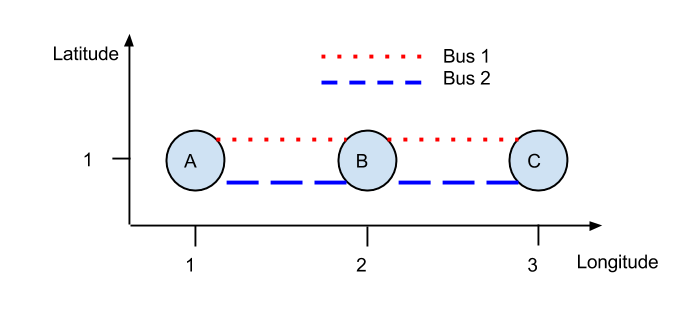
|  |  |  |
| --- | --- | --- |
| Bus | Route | Extra information |
| Bus 1A | A B C E D | - |
| Bus 1B | D E C B A | - |
| Bus 2A | A C D E F | - |
| Bus 3 | A F | Special Bus |

##### Links and timetable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bus | Link | Departure time | Arrival Time | Extra info. |
| Bus 1A | A B  B C  C E  E D | 0600  0605  0608  0611 | 0605  0608  0611  0616 | Course 1  Day Category 1 |
| A B  B C  C E  E D | 1600  1605  1608  1611 | 1605  1608  1611  1616 | Course 2  Day Category 1 |
| A B  B C  C E  E D | 0900  0905  0908  0911 | 0905  0908  0911  0916 | Course 1  Day Category 2 |
| Bus 1B | D E  E C  C B  B A | 0600  0605  0608  0611 | 0605  0608  0611  0616 | Course 1  Day Category 1 |
| D E  E C  C B  B A | 1600  1605  1608  1611 | 1605  1608  1611  1616 | Course 2  Day Category 1 |
| D E  E C  C B  B A | 0900  0905  0908  0911 | 0905  0908  0911  0916 | Course 1  Day Category 2 |
| Bus 2A | A C  C D  D E  E F | 0600  0606  0612  0615 | 0606  0612  0615  0620 | Course 1  Day Category 1 |
| A C  C D  D E  E F | 1600  1606  1612  1615 | 1606  1612  1615  1620 | Course 2  Day Category 1 |
| A C  C D  D E  E F | 0900  0906  0912  0915 | 0906  0912  0915  0920 | Course 1  Day Category 2 |
| Bus 3 | A F | 0619 | 0620 | Course 1  Day Category 1 |

#### Multi Link Network

Designed to see what is going on when there are multiple links between bus stops



|  |  |  |
| --- | --- | --- |
| Bus | Route | Extra information |
| Bus 1 | A B C | Time between bus stops does not change throughout the day |
| Bus 2 | A B C | Time changes throughout the day to test different scenarios |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bus | Link | Departure time | Arrival Time | Extra info. |
| Bus 1 | A B  B C | 0600  0605 | 0605  0610 | Course 1  Day Category 1 |
| A B  B C | 0900  0905 | 0905  0910 | Course 2  Day Category 1 |
| A B  B C | 1200  1205 | 1205  1210 | Course 3  Day Category 1 |
| Bus 2 | A B  B C | 0600  0604 | 0604  0608 | Course 1  Day Category 1 |
| A B  B C | 0900  0908 | 0908  0909 | Course 2  Day Category 1 |
| A B  B C | 1205  1207 | 1207  1210 | Course 3  Day Category 1 |

#### Big Network

This network is a 26 (column) by 100 bus stops. First bus course runs from 6 am and the last bus course starts at 7pm and have a random link interval of between 1 min to 3 min journey time. The buses are made to go every 20 min. This network was automatically generated. There is no diagram since would take up lots of space having so many bus stops. The idea is to test the performance of the program when operating on a large network. The size of the network like this, measuring 2600 bus stops, is similar to that of a bigger town.

Very Large Network

This network has a very similar idea to the big network. It will also be randomly generated and will contain 18,000 bus stops similar in size like London. I suspect this network will cause problems

## Testing Outline

Testing basic user interface functionality:

1. Drop downs
   1. Bus stop drop downs displaying all bus stops
   2. Time selection drop downs
2. Buttons
   1. Calculate button
   2. About button
   3. Help button
   4. Bus change risk buttons
3. Map (Interface)
   1. Bus stops and links all fit in to the map and all are displayed in the correct position
   2. Zoom in and out
   3. Moving across the map
   4. Bus stops highlights when mouse moves over them
   5. Choosing start and end bus stop from the map

Testing retrieving data from a data source

1. Loading data from a text file
2. Loading data from a database

Testing the algorithm functionality

1. Depart after algorithm
   1. A series of typical, erroneous and extreme cases
2. Arrive before algorithm
   1. A series of typical, erroneous and extreme cases

Testing feedback mechanism functionality

1. Feedback mechanism (Interface)
   1. Displays correct information
   2. Bus stops in the journey change colour appropriately
   3. Produces a printable document containing more detail
   4. Produces an electronic document containing more detail

Testing other features

1. Other
   1. Splash screen
   2. About Window
   3. Help Window
   4. Closing Window

## Test Plan

1. Drop downs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 1.1 | Testing if the bus stop drop downs display all bus stops | Bus Network:  Small network | Typical | A list of A,B,C,D, E, F |
| 1.2 | Testing if the drop downs displayed time in every 5 min. | Application Start | Typical | Drop downs display time in 5 min intervals |
| 1.3 | Testing to see if the default time is set to be to the nearest 5min rounded up | Operating System time: 15:31 | Typical | 15:35 |
| 1.4 | Operating System time: 15:59 | Extreme | 16:00 |
| 1.5 | Operating System time: 23:56 (boundary testing) | Very Extreme | 23:55 |

1. Buttons

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 2.1 | Testing the Calculate button | Click  Journey found | Typical | Feedback table produced |
| 2.2 | Click  No journey found | Extreme | Error message displayed |
| 2.3 | Testing the About button | Click | Typical | About window appears |
| 2.4 | Testing the Help button | Click | Typical | User Manual Appears |
| 2.5 | Testing the Bus change risk button | Increase up to 20 | Typical | Increments by 1 until 20 min |
| 2.6 | Decrease down to 0 | Typical | Decrements by 1 until 0 min |

1. Map (Interface)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 3.1 | Map is draw in the middle and all bus stops are in view no matter the location of bus stop | After loading | Typical | Map is drawn in the middle and all bus stops are in view no matter the location of bus stop |
| 3.2 | Zoom in and out slider | Dragging the zoom slider to the right | Typical | Increasing separation between bus stops but not the bus stop size |
| 3.3 | Dragging the zoom slider to the left | Typical | Decreasing separation between bus stops but not the bus stop size |
| 3.4 | Map can be moved dragged across when zoomed. | Mouse down and drag  Zoom slider set to 1 | Typical | The map moves in the same direction and through the same distance as the cursor |
| 3.5 | Mouse down and drag  Zoom slider set to 4 | Typical | The map moves in the same direction and through the same distance as the cursor. Every part of the map can be accessed when zoomed in |
| 3.6 |  | Zoom slider set to 0.75 | Typical | The map moves in the same direction and through the same distance as the cursor. Every part of the map can be accessed when zoomed out. |
| 3.7 | Bus stops highlighting responsively. | Mouse over  and mouse leaves  Case: Bus stop in ‘resting’ state – yellow colour | Typical | Bus stop changes to orange colour and changes back to resting state colour when mouse leaves |
| 3.8 | Context menu option selection | Right click, ‘from’ clicked,  Mouse over,  Mouse leave  Case: Bus stops in ‘resting’ state – yellow colour | Typical | Bus stop changes to green colour permanently until a different ‘from’ bus stop a selected |
| 3.9 | Right click, ‘to’ clicked,  Mouse over,  Mouse leave  Case: Bus stops in ‘resting’ state – yellow colour | Typical | Bus stop changes to red colour permanently until a different ‘to’ bus stop a selected, highlight when mouse moves over them |

Testing retrieving data from a data source

1. Loading data from a text file

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 4.1 | Loading a valid set of bus network data from a text file | A standard, complete and correct data. | Typical | Data loaded correctly, i.e. all information is consistent with that in the file |
| 4.2 | Loading an almost valid set of data that has some invalid ID records. | A standard, (nearly) complete data with some invalid ID’s. | Erroneous | Error message displayed and program closing |
| 4.3 | Trying to load a text file that does not exist | Invalid directory path | Erroneous | Error message display and program closing |
| 4.4 | Loading an almost valid set of data that has the timetable times and courses mixed. | A standard, (nearly) complete data with some bus courses and times faulty | Erroneous | No error detection |

1. Loading data from a database

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 5.1 | Loading a valid set of bus network data from a database | A standard, complete and correct data. | Typical | Data loaded correctly, i.e. all information is consistent with that in the file |
| 5.2 | Loading an almost valid set of data that has the timetable times and courses mixed. | A standard, (nearly) complete data with some invalid ID’s. | Erroneous | Error message displayed and program closing |
| 5.3 | Trying to connect to a database which does not exist | Invalid directory path | Erroneous | Error message display and program closing |
| 5.4 | Loading an almost valid set of data that has the timetable times and courses mixed. | A standard, (nearly) complete data with some bus courses and times faulty | Erroneous | No error detection |

Testing the algorithm functionality

Due to the nature of a network that is being tested, for some tests there might be more than one right answer, since there could be two solutions taking the same amount of time. Therefore the program will not be wrong quoting either of the results. To allow for this fact I added a preferred and alternative result to the ‘Expected Result’ column.

1. Depart after algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 6.1 | Testing if the depart after algorithm calculates starting with the link that is straight after the given time  Constant:  Bus Network:  Small network  From: A  To: B  Change Risk: 0  Day Category: 1 | Depart after: 05:45 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 |
| 6.2 | Depart after:  06:00 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 |
| 6.3 | Depart after:  06:01 | Typical | Get Bus1A at 10:00 from A and arrive at B at 10:05 |
| 6.4 | Testing if the algorithm calculates the correct journey  Constant:  Bus Network:  Small network  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: F  Depart after:  06:00 | Typical | Get Bus3 at 06:19 and arrive at 06:20 |
| 6.5 | From: B  To: F  Depart after:  06:00 | Typical | Preferred:  Get Bus1B from B at 06:11 to A, then take Bus3 from A to F at 06:19 and arrive at 06:20 |
| 6.6 | From: C  To: F  Depart after:  06:00 | Typical | Get Bus1A from C at 06:08 to E, then take Bus2A from E to F at 06:15 and6arrive at 06:20 |
| 6.7 | From: C  To: F  Depart after:  13:05 | Typical |  |
| 6.8 | From: C  To: F  Depart after: 23:00 | Typical |  |
| 6.9 | Testing if the algorithm manages with bus changes  Constant:  Bus Network:  MultiLinkNetwork  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: C  Depart after:  06:00 | Typical | Depart at 06:00 and arrive at 06:08 with Bus 2 |
| 6.10 | From: A  To: C  Depart after:  09:00 | Typical | Take Bus 2  Depart at:  09:00  Arrive at:  09:09 |
| 6.11 | From: A  To: C  Depart after:  12:00 | Typical | Take Bus 2  At 12:05  Arrive at 12:09 |

1. Arrive before algorithm

Same sort of tests as with the Depart After Algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 7.1 | Testing if the depart after algorithm calculates starting with the link that is straight before the given time  Constant:  Bus Network:  Small network  From: A  To: B  Change Risk: 0  Day Category: 1 | Arrive  Before: 06:10 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 |
| 7.2 | Arrive  Before: 06:05 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 |
| 7.3 | Arrive  Before: 06:01 | Typical | Error – no route found |
| 7.4 | Testing if the algorithm calculates the correct journey  Constant:  Bus Network:  Small network  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: F  Arrive before:  06:20 | Typical | Get Bus3 at 06:19 and arrive at 06:20 |
| 7.5 | From: B  To: F  Depart after:  06:20 | Typical | Preferred:  Get Bus1B from B at 06:11 to A, then take Bus3 from A to F at 06:19 and arrive at 06:20  An alternative result:  Take Bus1A from B to E at 06:05, then at 06:15 get Bus2A to F and arrive at 6:20 |
| 7.6 | From: C  To: F  Depart after:  06:00 | Typical | Preferred:  Get Bus1A from C at 06:08 to E, then take Bus2A from E to F at 06:15 and arrive at 06:20  An alternative result:  Take Bus2B from C to A at 06:08, then at 06:19 get Bus3 to F and arrive at 6:20 |
| 7.7 | From: C  To: F  Arrive before:  13:05 | Typical |  |
| 7.8 | From: C  To: F  Arrive before: 23:00 | Typical |  |
| 7.9 | Testing if the algorithm manages with bus changes  Constant:  Bus Network:  MultiLinkNetwork  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: C  Arrive before:  06:08 | Typical | Depart at 06:00 and arrive at 06:08 with Bus 2 |
| 7.10 | From: A  To: C  Arrive before:  06:10 | Typical | Depart at 06:00 and arrive at 06:08 with Bus 2 |
| 7.11 | From: A  To: C  Arrive before  12:10 | Typical | Take Bus 2  At 12:05  Arrive at 12:09 |

Testing feedback mechanism functionality

1. Feedback mechanism (Interface)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Description | Variable input data | Data type | Expected result |
| 8.1 | The Information displayed after calculation complete is valid and correct. | After a successful calculation |  | The result showing should appear correctly |
| 8.2 | When a new feedback table is created the route is highlighted on the map. | After a 3 successful calculations. |  | The last result should be highlighted |
| 8.3 | Changing journey feedback table selection. | When mouse clicks on a different feedback table |  | The selected feedback table is highlighted, journey appears on map. |
| 8.4 | The feedback table highlights when mouse moves over |  |  | Feedback table changes colour to lighter blue. |
| 8.5 | More information displayed | When ‘more’ button clicked |  | Window appears with more information |
| 8.6 | Printing the timetable | When ‘print’ button clicked |  | Print dialogue appears that allows selecting a printer and printing the file |

Testing other features

1. Other
   1. Splash screen
   2. About Window
   3. Bus Stop Info Window
   4. Closing Window

Screenshots which document the testing are at the back.

# Testing

Testing basic user interface functionality:

1. Drop downs
   1. Bus stop drop downs displaying all bus stop names
   2. Time selection drop downs
      1. Time options every 5 min
      2. Default time algorithm

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
| 1.1 | Testing if the bus stop drop downs display all bus stops | Bus Network:  Small network | Typical | A list of A,B,C,D, E, F | As expected - passed |  |
| 1.2 | Testing if the drop downs displayed time in every 5 min. | Application Start | Typical | Drop downs display time in 5 min intervals | As expected - passed |  |
| 1.3 | Testing to see if the default time is set to be to the nearest 5min rounded up | Operating System time: 15:31 | Typical | 15:35 | As expected - passed |  |
| 1.4 | Operating System time: 15:59 | Extreme | 16:00 | As expected - passed |  |
| 1.5 | Operating System time: 23:56 | Very Extreme | 23:55 | As expected - passed |  |

1. Buttons
   1. Calculate button
   2. About button
   3. Help button
   4. Bus change risk buttons

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
| 2.1 | Testing the Calculate button | Click  Journey found | Typical | Feedback table produced | As expected - passed |  |
| 2.2 | Click  No journey found | Extreme | Error message displayed | As expected - passed |  |
| 2.3 | Testing the About button | Click | Typical | About window appears | As expected - passed |  |
| 2.4 | Testing the Help button | Click | Typical | User Manual Appears | No user manual - Fail | This section was not finished. There was no neat way of transferring the word document to be opened at run time. |
| 2.5 | Testing the Bus change risk button | Increase up to 20 | Typical | Increments by 1 until 20 min | As expected - passed |  |
| 2.6 | Decrease down to 0 | Typical | Decrements by 1 until 0 min | As expected - passed |  |

1. Map (Interface)
   1. Bus stops and links all fit in to the map and all are displayed in the correct position
   2. Zoom in and out
   3. Moving across the map
   4. Bus stops highlights when mouse moves over them
   5. Choosing start and end bus stop from the map

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
| 3.1 | Map is draw in the middle and all bus stops are in view no matter the location of bus stop | After loading | Typical | Map is drawn in the middle and all bus stops are in view no matter the location of bus stop | Almost as expected -partially passed  See test cases with different bus networks | The network is drawn upside down because the y coordinate is increasing from top to bottom |
| 3.2 | Zoom in and out slider | Dragging the zoom slider to the right | Typical | Increasing separation between bus stops but not the bus stop size | Passed |  |
| 3.3 | Dragging the zoom slider to the left | Typical | Decreasing separation between bus stops but not the bus stop size | Passed |  |
| 3.4 | Map can be moved dragged across when zoomed. | Mouse down and drag  Zoom slider set to 1 | Typical | The map moves in the same direction and through the same distance as the cursor | Passed |  |
| 3.5 | Mouse down and drag  Zoom slider set to 4 | Typical | The map moves in the same direction and through the same distance as the cursor. Every part of the map can be accessed when zoomed in | Passed |  |
| 3.6 |  | Zoom slider set to 0.75 | Typical | The map moves in the same direction and through the same distance as the cursor. Every part of the map can be accessed when zoomed out. | Passed |  |
| 3.7 | Bus stops highlighting responsively. | Mouse over  and mouse leaves  Case: Bus stop in ‘resting’ state – yellow colour | Typical | Bus stop changes to orange colour and changes back to resting state colour when mouse leaves | Passed |  |
| 3.8 | Context menu option selection | Right click, ‘from’ clicked,  Mouse over,  Mouse leave  Case: Bus stops in ‘resting’ state – yellow colour | Typical | Bus stop changes to green colour permanently until a different ‘from’ bus stop a selected | Passed |  |
| 3.9 | Right click, ‘to’ clicked,  Mouse over,  Mouse leave  Case: Bus stops in ‘resting’ state – yellow colour | Typical | Bus stop changes to red colour permanently until a different ‘to’ bus stop a selected, highlight when mouse moves over them | Passed |  |

Testing retrieving data from a data source

1. Loading data from a text file

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
| 4.1 | Loading a valid set of bus network data from a text file | A standard, complete and correct data. | Typical | Data loaded correctly, i.e. all information is consistent with that in the file | Passed |  |
| 4.2 | Loading an almost valid set of data that has some invalid ID records. | A standard, (nearly) complete data with some invalid ID’s. | Erroneous | Error message displayed and program closing | Passed | There is an error but it does not give you whether there was something wrong will getting to a file or data is corrupt |
| 4.3 | Trying to load a text file that does not exist | Invalid directory path | Erroneous | Error message display and program closing | Passed |
| 4.4 | Loading an almost valid set of data that has the timetable times and courses mixed. | A standard, (nearly) complete data with some bus courses and times faulty | Erroneous | No error detection | Error not detected | It is expected that the data is not corrupt. |

1. Loading data from a database

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
| 5.1 | Loading a valid set of bus network data from a database | A standard, complete and correct data. | Typical | Data loaded correctly, i.e. all information is consistent with that in the file | Passed |  |
| 5.2 | Loading an almost valid set of data that has the timetable times and courses mixed. | A standard, (nearly) complete data with some invalid ID’s. | Erroneous | Error message displayed and program closing | Passed | There is an error but it does not give you whether there was something wrong will getting to a file or data is corrupt |
| 5.3 | Trying to connect to a database which does not exist | Invalid directory path | Erroneous | Error message display and program closing | Passed |
| 5.4 | Loading an almost valid set of data that has the timetable times and courses mixed. | A standard, (nearly) complete data with some bus courses and times faulty | Erroneous | No error detection | As expected | It is expected that the data is not corrupt. |

Testing the algorithm functionality

Due to the nature of a network that is being tested, for some tests there might be more than one right answer, since there could be two solutions taking the same amount of time. Therefore the program will not be wrong quoting either of the results. To allow for this fact I added a preferred and alternative result to the ‘Expected Result’ column.

1. Depart after algorithm
   1. A series of typical, erroneous and extreme case

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
| 6.1 | Testing if the depart after algorithm calculates starting with the link that is straight after the given time  Constant:  Bus Network:  Small network  From: A  To: B  Change Risk: 0  Day Category: 1 | Depart after: 05:45 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 | As expected |  |
| 6.2 | Depart after:  06:00 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 | As expected |  |
| 6.3 | Depart after:  06:01 | Typical | Get Bus1A at 10:00 from A and arrive at B at 10:05 | As expected |  |
| 6.4 | Testing if the algorithm calculates the correct journey  Constant:  Bus Network:  Small network  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: F  Depart after:  06:00 | Typical | Get Bus3 at 06:19 and arrive at 06:20 | As expected |  |
| 6.5 | From: B  To: F  Depart after:  06:00 | Typical | Preferred:  Get Bus1B from B at 06:11 to A, then take Bus3 from A to F at 06:19 and arrive at 06:20  An alternative result:  Take Bus1A from B to E at 06:05, then at 06:15 get Bus2A to F and arrive at 6:20 | As expected |  |
| 6.6 | From: C  To: F  Depart after:  06:00 | Typical | Preferred:  Get Bus1A from C at 06:08 to E, then take Bus2A from E to F at 06:15 and6arrive at 06:20  An alternative result:  Take Bus2B from C to A at 06:08, then at 06:19 get Bus3 to F and arrive at 6:20 | As expected |  |
| 6.7 | From: C  To: F  Depart after:  13:05 | Typical | Take Bus2B from C to A at 16:08, then at 16:19 get Bus3 to F and arrive at 16:20 | As expected |  |
| 6.8 | From: C  To: F  Depart after: 23:00 | Typical | No route found! error | As expected |  |
| 6.9 | Testing if the algorithm manages with bus changes  Constant:  Bus Network:  MultiLinkNetwork  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: C  Depart after:  06:00 | Typical | Depart at 06:00 and arrive at 06:08 with Bus 2 | As expected |  |
| 6.10 | From: A  To: C  Depart after:  09:00 | Typical | Take Bus 2  Depart at:  09:00  Arrive at:  09:09 | Take Bus 1 and change at B to Bus 2  Depart at:  09:00  Arrive at:  09:09  Failed | Depart at:  09:00  Arrive at:  09:09  Which is correct but takes a journey that involves an unnecessary change  Not the best path |
| 6.11 | From: A  To: C  Depart after:  12:00 | Typical | Take Bus 2  At 12:05  Arrive at 12:09 | Take Bus 1 and change at B to Bus 2  Depart at:  12:00  Arrive at:  12:09  Failed | Takes a journey that involves an unnecessary change and departs too early  Not the best or shortest path |

Tests 6.10 and 6.11 show how the algorithm fails in calculating the quickest path but fails as calculating the shortest path. The difference is, as outlined in the Design Section, that the algorithm will find the quickest time someone can be at the destination bus stop but will fail sometimes fail to find the shortest path. The algorithm fills the criteria for finding the quickest time but fails when minimising the number of bus changes.

1. Arrive before algorithm
   1. A series of typical, erroneous and extreme cases

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
| 7.1 | Testing if the depart after algorithm calculates starting with the link that is straight before the given time  Constant:  Bus Network:  Small network  From: A  To: B  Change Risk: 0  Day Category: 1 | Arrive  Before: 06:10 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 | As expected |  |
| 7.2 | Arrive  Before: 06:05 | Typical | Get Bus1A at 06:00 from A and arrive at B at 06:05 | As expected |  |
| 7.3 | Arrive  Before: 06:01 | Typical | Error – no route found | As expected |  |
| 7.4 | Testing if the algorithm calculates the correct journey  Constant:  Bus Network:  Small network  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: F  Arrive before:  06:20 | Typical | Get Bus3 at 06:19 and arrive at 06:20 | As expected |  |
| 7.5 | From: B  To: F  Arrive before:  06:20 | Typical | Preferred:  Get Bus1B from B at 06:11 to A, then take Bus3 from A to F at 06:19 and arrive at 06:20  An alternative result:  Take Bus1A from B to E at 06:05, then at 06:15 get Bus2A to F and arrive at 6:20 | As expected |  |
| 7.6 | From: C  To: F  Arrive before:  09:00 | Typical | Preferred:  Get Bus1A from C at 06:08 to E, then take Bus2A from E to F at 06:15 and arrive at 06:20  An alternative result:  Take Bus2B from C to A at 06:08, then at 06:19 get Bus3 to F and arrive at 6:20 | As expected |  |
| 7.7 | From: C  To: F  Arrive before:  06:00 | Typical | Error no route found | As expected |  |
| 7.8 | From: C  To: F  Arrive before: 23:00 | Typical | Take Bus2B from C to A at 16:08, then at 16:19 get Bus3 to F and arrive at 16:20 | As expected |  |
| 7.9 | Testing if the algorithm manages with bus changes  Constant:  Bus Network:  MultiLinkNetwork  Bus Change Risk: 0 min  Day Category: 1 | From: A  To: C  Arrive before:  06:08 | Typical | Depart at 06:00 and arrive at 06:08 with Bus 2 | As expected |  |
| 7.10 | From: A  To: C  Arrive before:  06:10 | Typical | Depart at 06:00 and arrive at 06:08 with Bus 2 | Take Bus 2 and change at B to Bus 1  Depart at:  06:00  Arrive at:  06:10  Failed | takes a journey that involves an unnecessary change. And takes longer to travel.  Not the best path |
| 7.11 | From: A  To: C  Arrive before  12:10 | Typical | Take Bus 2  At 12:05  Arrive at 12:09 | Depart at:  12:00  Arrive at:  12:10  Failed | Not the best or shortest path |

The tests failed here, however that is only when looking under the perspective of the best of the best route. In fact, this special case will only be valid in some situations. Although the algorithm is not the best for finding the shortest path it works for finding the quickest path.

Testing feedback mechanism functionality

1. Feedback mechanism (Interface)
   1. Displays correct information
   2. Bus stops in the journey change colour appropriately
   3. Produces a printable document containing more detail
   4. Produces an electronic document containing more detail

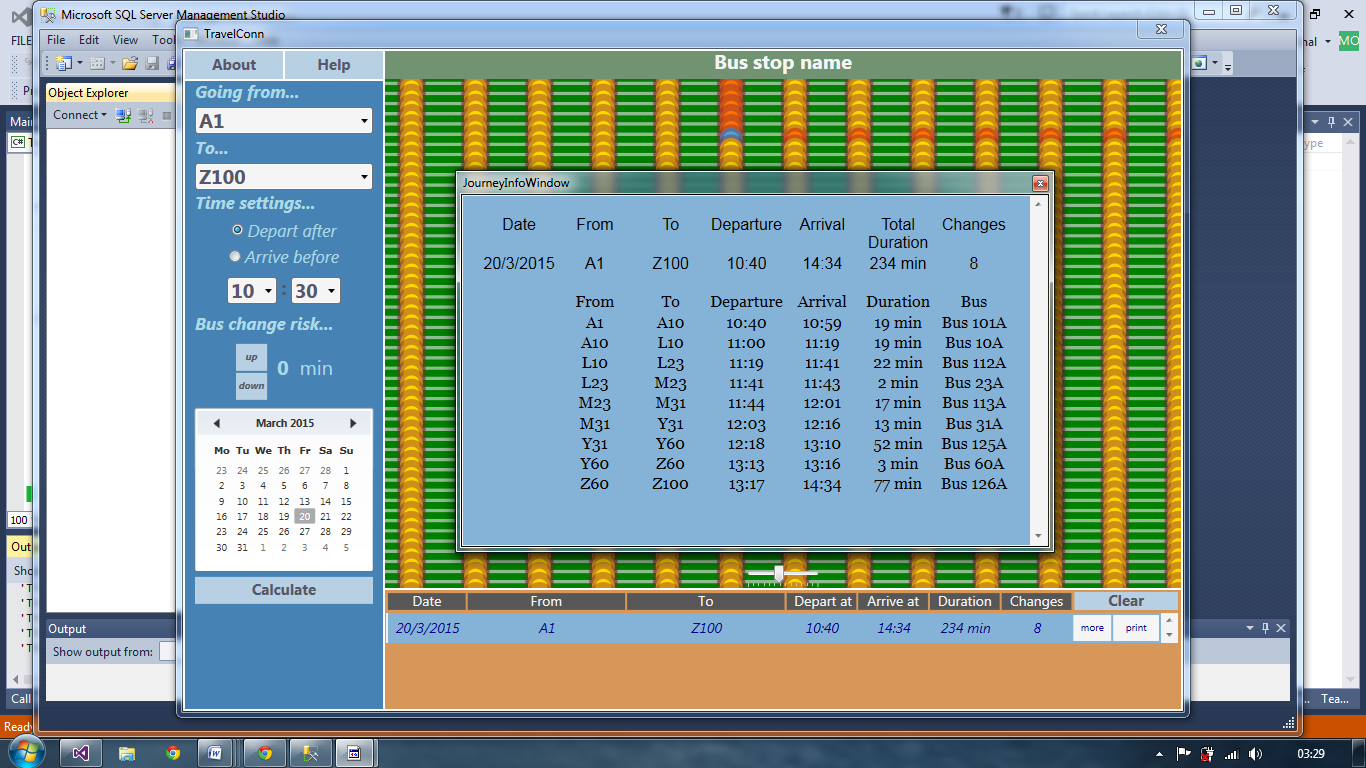
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
|  | The Information displayed after calculation complete is valid and correct. | - | - | - | Passed | Lots of data tested in the tests above. If the result produced was not the same as the actual worked out result then the tests above would be invalid |
|  | When a new feedback table is created the route is highlighted on the map. | Mouse over | - | journey on map highlights | As expected  passed |  |
|  | Changing journey feedback table selection. | When mouse clicks on a different feedback table | - | The selected feedback table is highlighted, journey appears on map. | As expected  passed |  |
|  | The feedback table highlights when mouse moves over | Mouse over | - | Feedback table changes colour to lighter blue. | As expected  passed |  |
|  | More information displayed | When ‘more’ button clicked | - | Window appears with more information | As expected  passed |  |
|  | Printing the timetable | When ‘print’ button clicked | - | Print dialogue appears that allows selecting a printer and printing the file | As expected  passed |  |

Testing other features

1. Other
   1. Splash screen
   2. About Window
   3. Bus Stop Info Window
   4. Closing Window

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Description** | **Variable input data** | **Data type** | **Expected result** | **Actual Outcome** | **Comments** |
|  | Splash screen at start up | Program Run | Typical | Splash Screen appears before the Main Window to load the data | As expected  passed |  |
|  | Splash screen when the calculate button clicked and loading new timetable into memory. | Calculate button click  Date changed corresponding to a different day category | Typical | Splash Screen appears before the journey is calculated | As expected  passed |  |
|  | About Window appears | About button clicked | Typical | About window appears | As expected  passed |  |
|  | Bus Stop info window | Right click on a BusStopCtrl and ‘Show more information’ selected | Typical | A window appears showing more information about the bus stop, including location, address | As expected  passed |  |
|  | When closing the program an message appears to confirms the window closure | Exit button clicked | Typical | Error message appears | As expected  passed |  |

1. Testing performance under a big network of bus stops and buses



This screen shows a bus network of about 26000 bus stops automatically genereated. It takes about one second to calculate the data.

# System Maintenance

This section will cover the necessary information concerning the new system and code. This section is for the person who might want maintains the system.

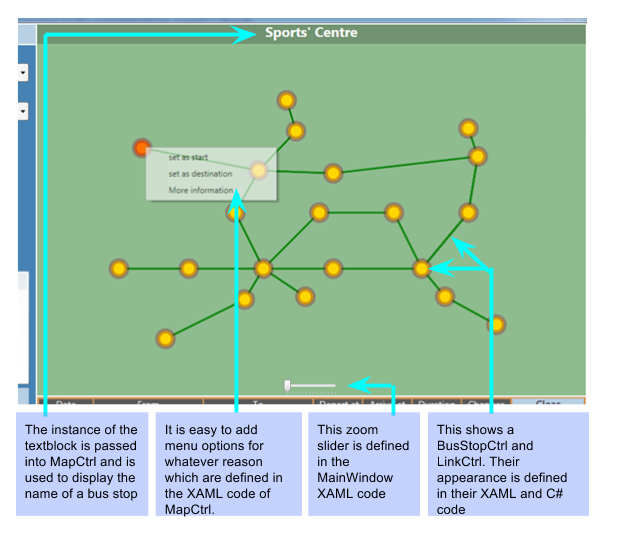
## Overview of the new system

The program provides a solution the quickest path between two bus stops problem. The system uses several classes which either have been outlined in the Design Section or will be outlined here.

## Interface

Most of the interface is described in the Design section under User Interface heading

### MapCtrl - in depth description



The program can be easily modified to show links of different buses between bus stops in different colour – by only changing the design of the LinkCtrl. The code has been designed so that features can be added / modified easily to a suit the needs.

# Accessing bus network data

### Automatically generated bus networks

For development purposes some of the bus network data was made up - some of the data was created manually and some generate automatically, like Small Network and other networks referred to in the design section. The auto generated networks, i.e. any types of grid networks, were generated by an outside program which is independent of the whole system, refer to Python Code section. The code was written by the developer (me) and used to generate random and valid buses, bus stops, links and timetable. The generated timetable was made to be consistent with the database design and to work with TravelConn. This was used for development and testing purposes only; it does not make up any part of the final system.

### Important information

The program only validates the bus network data to an extent and will not detect if the arrival and departure times have been corrupted. It is important to check if the course number and bus times are in the right order.

Example of a valid timetable and invalid timetable:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DayCat  ID | LinkID | Course | TimeA | TimeB |
| 1 | 1 | 1 | 0600 | 0605 |
| 1 | 2 | 1 | 0605 | 0608 |
| 1 | 3 | 1 | 0608 | 0611 |
| 1 | 4 | 1 | 0611 | 0616 |

The table on the left shows a valid timetable of one bus and course with LinkID corresponding to consecutive links.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DayCat  ID | LinkID | Course | TimeA | TimeB |
| 1 | 1 | 1 | 0600 | 0605 |
| 1 | 2 | 1 | 0605 | **0608** |
| 1 | 3 | 1 | **0600** | 0606 |
| 1 | 4 | 1 | 0611 | 0616 |

The table on the right shows an invalid timetable. The table is still of one bus and course with LinkID corresponding to consecutive links in the route of the bus. It claims that a bus arriving at bus stop at 0608 leaves the bus stop at 0600 which physically is impossible (travelling back in time).

The program will not detect this fault since it assumes that receives valid data which is checked by the external company like Stagecoach.

### Connecting to a database

TravelConn database was created using Microsoft SQL Server 2012. SQL was used to create tables and insert the data into the database, refer to SQL Code section to find code that was used.

If connecting to a different database is required, then the only means of changing a database is to hard-code it into the program. N.B. When changing a database source, make sure table names match.

The place in the code where the data source and path are set is located in the MainWindow() method (constructor) of the MainWindow class (C#). Change the line which creates an instance of the CBusNetwork class:

E.g. loading from a file:

busNetwork = new CBusNetwork("file", @"TestNetwork\");

E.g. loading from a database:

busNetwork = new CBusNetwork("database", "Server=.\\SQLEXPRESS;Database=TravelConnDB;Integrated Security=true");

To connect to a database a valid connection string needs to be set **before** the program is compiled; otherwise TravelConn will fail to connect to the database.

## Code overview

#### Variable and procedure naming

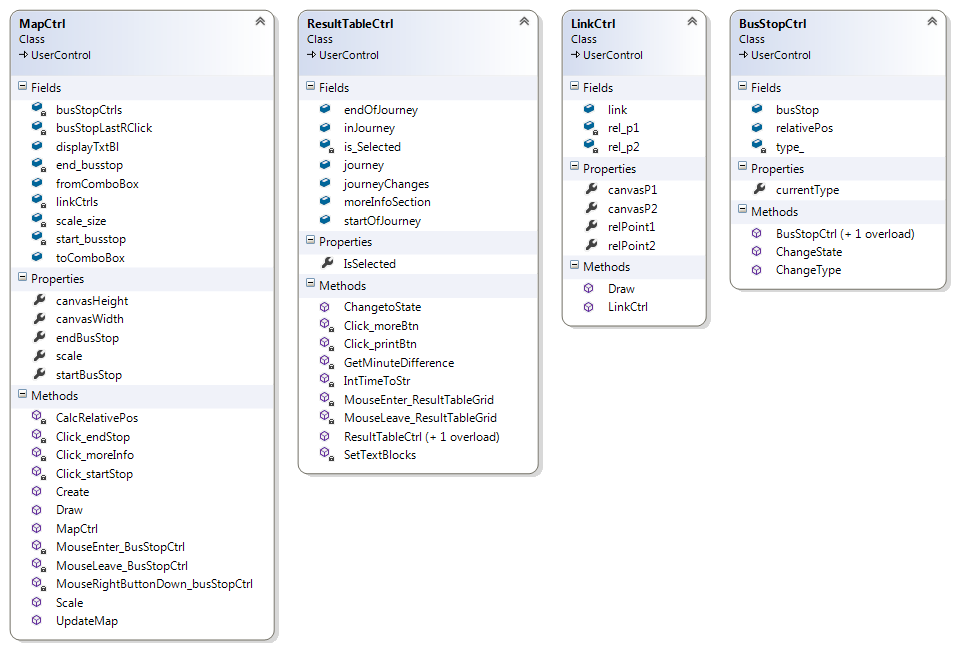
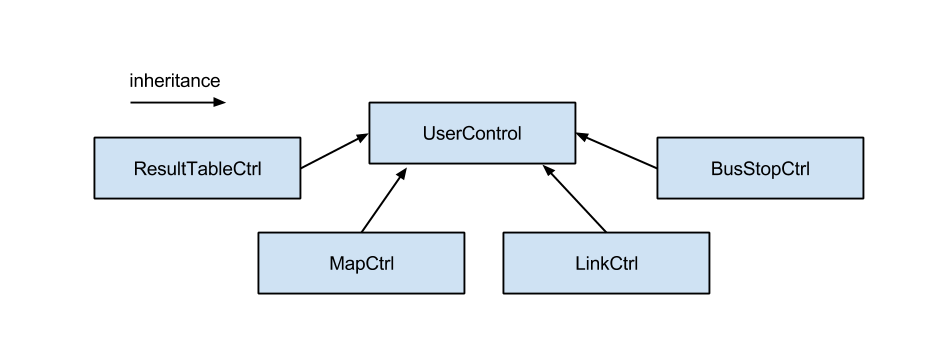
The code was intended to have consistent variable and procedure naming, making it easier to follow. The code will normally follow those rules:

* busNetwork – lowercase first word followed up by new word starting with an uppercase letter – this is a public property and/or field
* is\_Dragging – lowercase first word followed up by an underscore followed by a word starting with an uppercase letter – this is a private property and/or field
* CalculateJourney – first letter of the first word is uppercase, followed up by new word starting with an uppercase letter – this is a public procedure/ function
* Click\_clearButton – first letter of the first word is uppercase, followed up by an underscore, followed up by new word – are functions/ procedures which are triggered by some user interaction

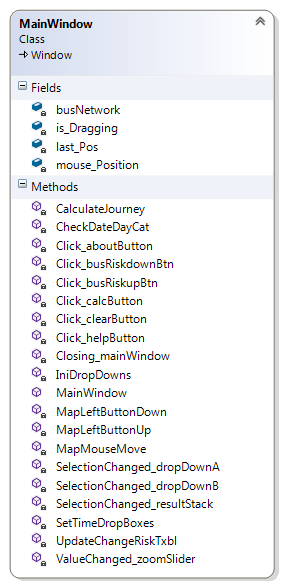
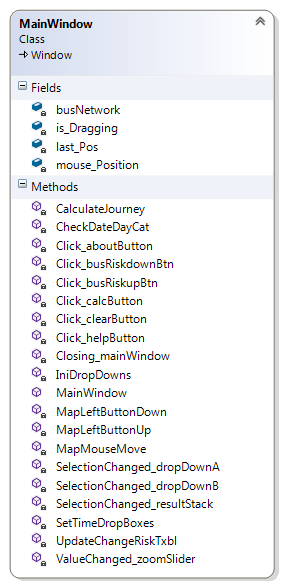
## Additional Class Description

A general overview of the classes is described in the Design Section. However not all aspects were foreseen and were not described in the Design sections.

### User Controls Classes

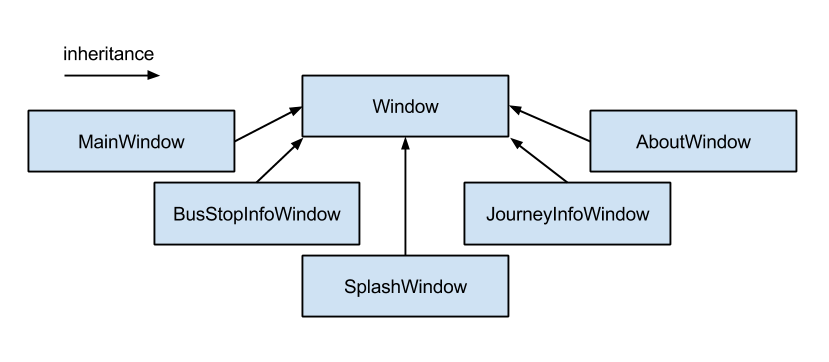


Key:

 PrivatePublic

These classes inherit from UserControl Class which taken from the System Library.

### Windows

 All windows inherit from System.Windows

## MainWindow Class

Here is a more in depth description of the Main Window class. MainWindow inherits from Windows

### Functions and procedures

|  |  |
| --- | --- |
| Procedures/ Functions | |
| Access | Name | Description | Parameters | Return values |
| Public | MainWindow() | Handles loading of data and the initialisation of the program map | None | None |
| Private | Closing\_mainWindow() | Handles the closure of the program | 1.object sender  2.System.ComponentModel.CancelEventArgs e | None |
| Private | IniDropDowns() | Sets the values in the drop down menu | None | None |
| Private | SetTimeDropBoxes() | Based on the Windows current clock time, sets a default time in the drop downs | None | None |
| Private | Click\_calcButton() | When the calculate button is clicked | 1.object sender 2.RoutedEventArgs e | None |
| Private | CalculateJourney() | Interface method to calculate the quickest journey given that details provided passes all validation | 1.DateTime date 2.CBusStop startbusStop 3.CBusStop endbusStop 4.string hours 5.string mins | None |
| Private | CheckDateDayCat() | Checks what date was selected and looks if the day categories match; reloads timetable if the day category loaded is different  Returns the date that is selected | 1. ref bool loaded | DateTime  ref bool loaded |
| Private | Click\_clearButton() | Clears all the result Stack forms from the stack list | 1.object sender, 2.RoutedEventArgs e | None |
| Private | SelectionChanged\_dropDownA() | Is triggered when the selection of dropDownA changes | 1.object sender 2.SelectionChangedEventArgs e | None |
| Private | SelectionChanged\_dropDownB() | Is triggered when the selection of the dropDownB changes | 1.object sender, 2.SelectionChangedEventArgs e | None |
| Private | Click\_busRiskupBtn() | bus change risk is increased, updates the text block | 1.object sender, 2.RoutedEventArgs e | None |
| Private | Click\_busRiskdownBtn() | bus change risk is decreased, updates the text block | 1.object sender 2.RoutedEventArgs e) | None |
| Private | ValueChanged\_zoomSlider () | When the slider is moved rescale the map | 1.object sender 2.RoutedPropertyChangedEventArgs<double> e | None |
| Private | MapLeftButtonDown() | Map clicked | 1.object sender 2.MouseEventArgs e | None |
| Private | MapLeftButtonUp() | Left mouse button up; drop the map so that it no longer moves with the mouse | 1.object sender, 2.MouseEventArgs e | None |
| Private | MapMouseMove() | Dragging the map across | 1.object sender, 2.MouseEventArgs e | None |
| Private | Click\_aboutButton() | when about button clicked | 1.object sender, 2.RoutedEventArgs e) | None |
| Private | Click\_helpButton() | when help button clicked | 1.object sender, 2.RoutedEventArgs e) | None |

### Variables

|  |  |  |  |
| --- | --- | --- | --- |
| Access | Name | Type | Description |
| Private | busNetwork | CBusNetwork | The global variable that contains all of the bus network data |
| Private | mouse\_Position | System.Windows.Point | Stores the point of the mouse relative to the map where the left mouse button was pressed down |
| Private | Last\_Pos | System.Window.Point | Stores the previous displacement position of the map |
| Private | Is\_Dragging | bool | True when left mouse button is down and inside the map, false otherwise. |

### Class Diagram

The class diagram is located in the Design Section.

## Bus Network Classes

Described in the Design Section

For the description of what each variable and procedure do also refer to the comments in code.

# Trace Tables

## Depart After Algorithm (CBusNetwork)

The algorithm’s pseudo code is described in the Design Section. Here is the copy of the algorithm’s implementation in C#. The following trace tables will operate with Small Network Data (See Design for more details about this bus network).

First let’s consider the first part of the algorithm: set up and reset of values.

Initial Input:

* CBusStop startBusStop <= bus stop with name A
* CBusStop endBusStop <= bus stop with name E
* int time <= 0600

Expected output:

* best links of...

Trace table

|  |
| --- |
| **Testing the first loop in the DepartAfter Algorthm** |
| **Description** |
| This loop operates when  currentBusStop is startBusStop  This is a test to see if best links are chosen for neighbours of currentBusStop. |
| **Code (C#)** |
| foreach (CLink link in currentBusStop.departureLinks)  {  foreach (CTimetableRecord record in link.linkTimetable)  {  if ((record.departTime >= time) && (record.arrivalTime < link.busStopB.bestLinkRecord.arrivalTime))  {  link.busStopB.bestLinkRecord = record;  }  }  } |
| **Expected result** |
| |  |  |  | | --- | --- | --- | | link.busStopB: B | link.busStopB: C | link.busStopB: F | | departTime: 0600  arriveTime: 0605 | departTime: 0600  arriveTime: 0606 | departTime: 0619  arriveTime: 0620 |   . |

To see what is going on letter/names are going to be used instead of IDs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLink link | CTimetableRecord record | link.busStopB.bestLinkRecord | | |
| link.busStopB: B | link.busStopB: C | link.busStopB: F |
| - | - | departTime: 9999  arriveTime: 9999 | departTime: 9999  arriveTime: 9999 | departTime: 9999  arriveTime: 9999 |
| A B | departTime: 0600  arriveTime: 0605 | departTime: 0600  arriveTime: 0605 | -“- | -“- |
| A B | departTime: 1600  arriveTime: 1605 | -“- | -“- | -“- |
| A B | departTime: 0900  arriveTime: 0905 | -“- | -“- | -“- |
| A C | departTime: 0600  arriveTime: 0606 | -“- | departTime: 0600  arriveTime: 0606 | -“- |
| A C | departTime: 1600  arriveTime: 1605 | -“- | -“- | -“- |
| A C | departTime: 0900  arriveTime: 0905 | -“- | -“- | -“- |
| A F | departTime: 0619  arriveTime: 0620 | -“- | -“- | departTime: 0619  arriveTime: 0620 |
| - | - | departTime: 0600  arriveTime: 0605 | departTime: 0600  arriveTime: 0606 | departTime: 0619  arriveTime: 0620 |

End state (as expected):

|  |  |  |
| --- | --- | --- |
| link.busStopB: B | link.busStopB: C | link.busStopB: F |
| departTime: 0600  arriveTime: 0605 | departTime: 0600  arriveTime: 0606 | departTime: 0619  arriveTime: 0620 |

|  |
| --- |
| **Testing the main loop in the DepartAfter Algorthm** |
| **Description** |
| Testing the main loop. It is assumed that is code follows the first loop and therefore has the results from the previous loop. The first 2 line of the following code will select bus stop ‘B’ and remove it, since it’s still in the unvisitedBusStops list and has the smallest bestLink.arrivalTime (as the result of the first loop). Therefore currentBusStop **← (bus stop B)** |
| **Code (C#)** |
| currentBusStop = FindQuickestArrival(unvisitedBusStops);  unvisitedBusStops.Remove(currentBusStop);  while ((currentBusStop != endBusStop) && (unvisitedBusStops.Count > 0) && (currentBusStop.bestLinkRecord.arrivalTime != 9999)) {  foreach (CLink link in currentBusStop.departureLinks){  if (unvisitedBusStops.Contains(link.busStopB)) {  foreach (CTimetableRecord record in link.linkTimetable) {  if (record.departTime >= currentBusStop.bestLinkRecord.arrivalTime){  if (record.arrivalTime <= record.link.busStopB.bestLinkRecord.arrivalTime) {  if (record.departTime >= AddMinToTime(currentBusStop.bestLinkRecord.arrivalTime,change\_risk)) {  record.link.busStopB.bestLinkRecord = record;  }  else if ((record.course == currentBusStop.bestLinkRecord.course)  && (record.link.bus == currentBusStop.bestLinkRecord.link.bus))  {  record.link.busStopB.bestLinkRecord = record;  }  }  }  }  }  }  currentBusStop = FindQuickestArrival(unvisitedBusStops);  unvisitedBusStops.Remove(currentBusStop);  } |
| **Expected result** |
| These are the best links for each bus stop. From this it can determined that taking Bus 2A will take you to ‘D’ the quickest, arrival time is 0612   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | A | B | C | D | E | F | | arrTime: 0600  depTime:9999  course: 0  bus: - | arrTime: 0605  depTime:0600  course: 1  bus: Bus 1A | arrTime: 0606  depTime:0600  course: 1  bus: Bus 2A | arrTime: 0612  depTime:0606 course: 1  bus: Bus 2A | arrTime: 0611  depTime:0608 course: 1  bus: Bus 1A | arrTime: 0620  depTime:0615 course: 1  bus: Bus 2A | |

These values for the bestLink of each bus stop are assumed at the start

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F |
| arrTime: 0600  depTime:9999  course: 0  bus: - | arrTime: 0605  depTime:0600  course: 1  bus: Bus 1A | arrTime: 0606  depTime:0600  course: 1  bus: Bus 2A | arrTime: 9999  depTime:9999  course: 0  bus: - | arrTime: 9999  depTime:9999  course: 0  bus: - | arrTime: 0620  depTime:0619  course: 1  bus: Bus 3 |

Start of the main loop

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Current  BusStop | Best link of current bus stop | link | record | UnvisitedBus stops.Count | bestLink of link.busStopB |
| B | arrTime: 0605  depTime: 0600 course: 1  bus: Bus 1A | B A | - | 4 | -“- |
| -“- | -“- | B C | arrTime: 0608  depTime: 0605 course: 1  bus: Bus 1A | -“- | -“- |
| -“- | -“- | -“- | arrTime: 1605  depTime: 1608 course: 2  bus: Bus 1A | -“- | -“- |
| C | arrTime: 0606  depTime:0600  course: 1  bus: Bus 2A | C A | - | 3 | -“- |
| -“- | -“- | C B | - | -“- | -“- |
| -“- | -“- | C E | arrTime: 0611  depTime: 0608 course: 1  bus: Bus 1A | -“- | (bus stop: ‘E’)  arrTime: 0611  depTime: 0608 course: 1  bus: Bus 1A |
| -“- | -“- | -“- | arrTime: 1611  depTime: 1608 course: 2  bus: Bus 1A | -“- | -“- |
| -“- | -“- | C D | arrTime: 0612  depTime: 0606 course: 1  bus: Bus 2A | -“- | (bus stop: ‘D’)  arrTime: 0612  depTime: 0606 course: 1  bus: Bus 2A |
| -“- | -“- | -“- | arrTime: 1612  depTime: 1606 course: 2  bus: Bus 2A | -“- | -“- |
| E | arrTime: 06011  depTime: 0608 course: 1  bus: Bus 1A | E D | arrTime: 0616  depTime: 0611 course: 1  bus: Bus 1A | 2 | -“- |
| -“- | -“- | -“- | arrTime: 1616  depTime: 1611 course: 2  bus: Bus 1A | -“- | -“- |
| -“- | -“- | E C | - | -“- | -“- |
| -“- | -“- | E F | arrTime: 0620  depTime: 0615 course: 1  bus: Bus 2A | -“- | (bus stop: ‘F’)  arrTime: 0620  depTime: 0615 course: 1  bus: Bus 2A |
| -“- | -“- | -“- | arrTime: 1620  depTime: 1615 course: 1  bus: Bus 2A | -“- | -“- |
| D | arrTime: 0612  depTime: 0606 course: 1  bus: Bus 2A | - | - | 1 | - |

Current bus stop is destination bus stop, therefore stop.

End stage of best links (as expected):

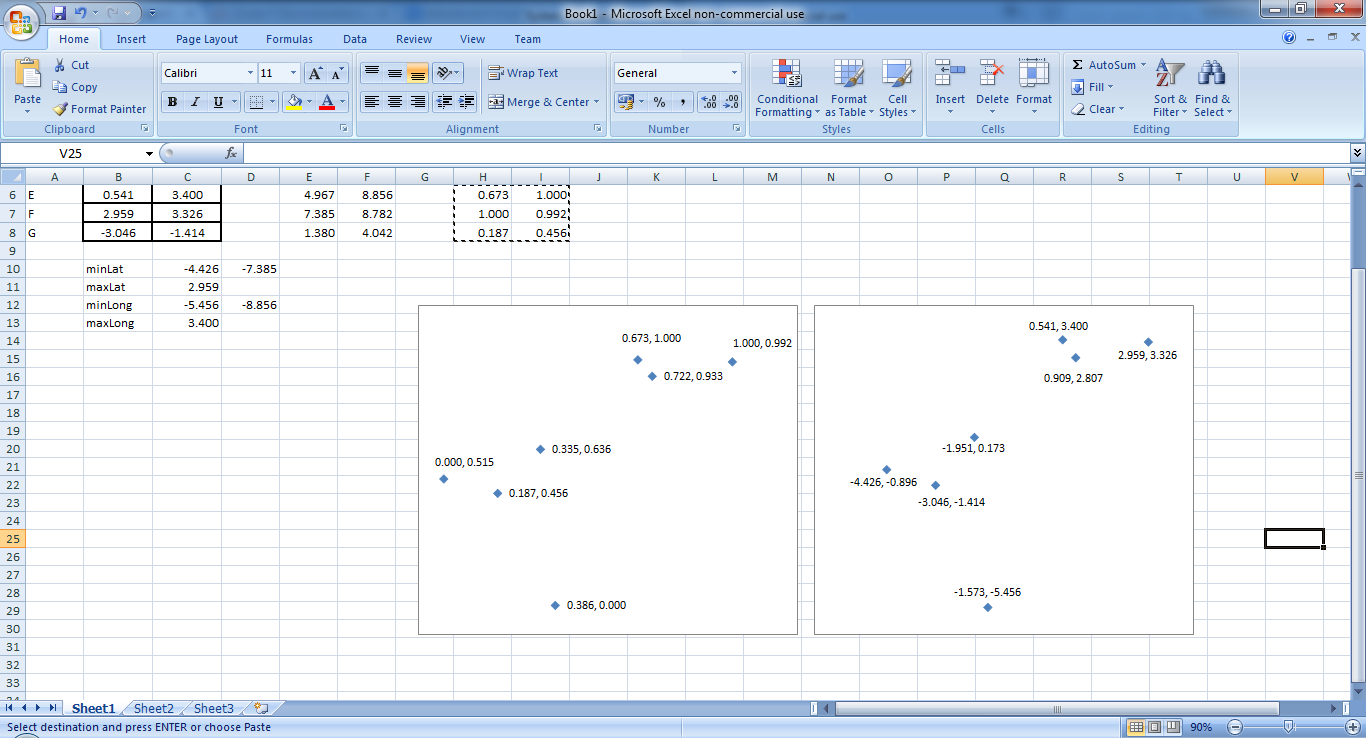
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F |
| arrTime: 0600  depTime:9999  course: 0  bus: - | arrTime: 0605  depTime:0600  course: 1  bus: Bus 1A | arrTime: 0606  depTime:0600  course: 1  bus: Bus 2A | arrTime: 0612  depTime:0606 course: 1  bus: Bus 2A | arrTime: 0611  depTime:0608 course: 1  bus: Bus 1A | arrTime: 0620  depTime:0615 course: 1  bus: Bus 2A |

CalcRelativePos() (MapCtrl)

|  |
| --- |
| **Testing the first loop** |
| **Description** |
| This algorithm makes sure that all bus stops are in view and inside the MapCtrl instance. This method is called before bus stops are drawn to calculate their relative position on the map.  Using data for longitude and latitude   |  |  |  | | --- | --- | --- | | Bus stop | latitude | longitude | | A | -4.426 | -0.896 | | B | -1.573 | -5.456 | | C | -1.951 | 0.173 | | D | 0.909 | 2.807 | | E | 0.541 | 3.400 | | F | 2.959 | 3.326 | | G | -3.046 | -1.414 | |
| **Pseudo Code** |
| **Function CalcRelativePos():**  **minLat ← 99.0** *// the smallest value of latitude, set to be high initially*  **maxLat ← -99.0** *// the largest value of latitude, set to be low initially*  **minLong ← 199.0** *// the smallest value of longitude, set to be high initially*  **maxLong ← -199.0** *// the largest value of longitude, set to be low initially*  *// The values are set so that they are the most extreme values of latitude and*  *// longitude possible*  *// Loop 1*  **for each busStop in busNetwork.busStops:**  **if busStop.latitude > maxLat:**  **maxLat ← busStop.latitude**  **end if**  **if busStop.latitude < minLat:**  **minLat ← busStop.latitude**  **end if  if busStop.longitude > maxLong:**  **maxLong ← busStop.longitude**  **end if  if busStop.longitude < minLong:**  **minLong ← busStop.longitude**  **end if**  **end for**  **relativePos ← undefined**  **// Loop 2**  **for each busStop in busNetwork.busStops:**  **relativePos.X ← (busStop.latitude - minLat) / (maxLat - minLat)**  **relativePos.Y ← (busStop.longitude - minLong) / (maxLong - minLong)**  **busStop.relativePos ← relativePos**  **end for**  **end function** |
| **Expected result** |
| All relative point should be between 0 and 1  and have the same distance ratio as in real world.   |  |  | | --- | --- | | minLat | -4.426 | | maxLat | 2.959 | | minLong | -5.456 | | maxLong | 3.400 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bus stop | minLat | maxLat | minLong | maxLong |
| Beginning of loop 1 | 99.0 | -99.0 | 199.0 | -199.0 |
| A | -4.426 | -4.426 | -0.896 | -0.896 |
| B | -“- | -1.573 | -5.456 | -“- |
| C | -“- | -“- | -“- | 0.173 |
| D | -“- | 0.909 | -“- | -“- |
| E | -“- | -“- | -“- | 3.400 |
| F | -“- | 2.959 | -“- | -“- |
| G | -“- | -“- | -“- | -“- |
| Output at the end of loop 1 (as expected) | -4.426 | 2.959 | -5.456 | 3.400 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bus stop | **busStop.latitude - minLat** | **maxLat - minLat** | **busStop.longitude - minLong** | **maxLong - minLong** | Relativepos  X Y | |
| A | 0.000 | 7.385 | 4.560 | 8.856 | 0.000 | 0.515 |
| B | 2.853 | -“- | 0.000 | -“- | 0.386 | 0.000 |
| C | 2.475 | -“- | 5.629 | -“- | 0.335 | 0.636 |
| D | 5.335 | -“- | 8.263 | -“- | 0.722 | 0.933 |
| E | 4.967 | -“- | 8.856 | -“- | 0.673 | 1.000 |
| F | 7.385 | -“- | 8.782 | -“- | 1.000 | 0.992 |
| G | 1.380 | -“- | 4.042 | -“- | 0.187 | 0.456 |



This shows that the points have been scaled, so the result is correct.

# Code

Code

# SQL

Code used to create tables in the database and insert data

## Creating tables

use TravelConnDB;

Create TABLE Buses (

BusID int Primary Key Identity NOT NULL,

Name varchar(20) NOT NULL,

Descript varchar(50),

);

CREATE TABLE BusStops (

BusStopID int Primary Key Identity NOT NULL,

Name varchar(30) NOT NULL,

Longitude decimal(8,6) NOT NULL,

Latitude decimal(8,6) NOT NULL,

Addres varchar(50),

);

CREATE TABLE DayCategories(

DayCatID int Primary Key Identity NOT NULL,

DayNam varchar(30) NOT NULL,

);

CREATE TABLE DaysOfWeek(

DayID int Primary Key Identity NOT NULL,

Name varchar(15) NOT NULL,

DayCatID int FOREIGN KEY REFERENCES DayCategories(DayCatID) NOT NULL,

)

CREATE TABLE SpecialDays(

DateID int Primary Key Identity NOT NULL,

CalendarDate date NOT NULL,

DayCatID int FOREIGN KEY REFERENCES DayCategories(DayCatID) NOT NULL,

);

CREATE TABLE BusLinks(

LinkID int Primary Key Identity NOT NULL,

BusID int FOREIGN KEY REFERENCES Buses(BusID) NOT NULL,

BusStopAID int FOREIGN KEY REFERENCES BusStops(BusStopID) NOT NULL,

BusStopBID int FOREIGN KEY REFERENCES BusStops(BusStopID) NOT NULL,

);

CREATE TABLE Timetable(

DayID int FOREIGN KEY REFERENCES DayCategories(DayCatID) NOT NULL,

LinkID int FOREIGN KEY REFERENCES BusLinks(LinkID) NOT NULL,

CourseNumber int NOT NULL,

DepTime int NOT NULL,

ArrTime int NOT NULL,

Constraint ID PRIMARY KEY (DayID,LinkID,CourseNumber),

);

## Populating tables

SET IDENTITY\_INSERT dbo.Buses ON;

Insert into dbo.Buses (BusID,Name,Descript)

Values(1,'Bus 1A','some bus'),

(2,'Bus 1B','some other bus'),

SET IDENTITY\_INSERT dbo.Buses OFF;

SET IDENTITY\_INSERT dbo.BusStops ON;

Insert into dbo.BusStops(BusStopID,Name,Longitude,Latitude,Addres)

Values (65,'A',1,7,'address A'),

(66,'B',2.5,7,'address B'),

(67,'C',4.1,7,'address C'),

SET IDENTITY\_INSERT dbo.BusStops OFF;

SET IDENTITY\_INSERT dbo.BusLinks ON;

Insert into dbo.BusLinks(LinkId,BusID,BusStopAID,BusStopBID)

Values (11,1,65,66),

(12,1,66,67),

(13,1,67,68),

SET IDENTITY\_INSERT dbo.BusLinks OFF;

SET IDENTITY\_INSERT dbo.DayCategories ON;

Insert into dbo.DayCategories(DayCatID,Descript)

Values (1,'Normal'),

(2,'Weekends'),

(3,'Bank Holidays');

SET IDENTITY\_INSERT dbo.DayCategories OFF;

Insert into dbo.DaysOfWeek(Name,DayCatID)

Values ('Monday',1),

('Tuesday',1),

('Wednesday',1),

('Thursday',1),

('Friday',1),

('Saturday',2),

('Sunday',2);

Insert into dbo.SpecialDays(CalendarDate,DayCatID)

Values ('2015-03-16',3),

('2015-03-23',3);

Insert into dbo.Timetable (DayID,LinkID,CourseNumber,DepTime,ArrTime)

Values (3,11,100,0600,0605);

# XAML

Xaml code to create the visual components of the program

## MainWindow

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

x:Name="mainWindow" mc:Ignorable="d" x:Class="TravelConn.MainWindow"

xmlns:Controls="clr-namespace:TravelConn.Controls"

Title="TravelConn" Height="700" Width="1016" ResizeMode="NoResize" WindowStartupLocation="CenterScreen"

Closing="Closing\_mainWindow"

>

<Window.Resources>

<Style TargetType="TextBlock">

<Setter Property="FontStyle" Value="Normal" />

<Setter Property="FontWeight" Value="Normal" />

<Setter Property="FontFamily" Value="Comic Sans" />

<Setter Property="FontSize" Value="20"/>

<Setter Property="Foreground" Value="#FF575666"/>

<Setter Property="Margin" Value="10,2,10,2"/>

</Style>

<Style x:Key="ResultTableTextBlock" TargetType="TextBlock">

<Setter Property="FontStyle" Value="Normal" />

<Setter Property="FontWeight" Value="Normal" />

<Setter Property="FontFamily" Value="Comic Sans" />

<Setter Property="FontSize" Value="14"/>

<Setter Property="Foreground" Value="WhiteSmoke"/>

<Setter Property="Background" Value="#FF575757"/>

<Setter Property="Margin" Value="1,3,1,2"/>

<Setter Property="TextAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="TextWrapping" Value="Wrap"/>

</Style>

<Style TargetType="Button">

<Setter Property="FontStyle" Value="Normal" />

<Setter Property="FontFamily" Value="Comic Sans" />

<Setter Property="FontSize" Value="15"/>

<Setter Property="FontWeight" Value="Bold"/>

<Setter Property="Height" Value="27"/>

<Setter Property="Foreground" Value="#FF575666"/>

<Setter Property="Margin" Value="2,2,0,0"/>

<Setter Property="Cursor" Value="Hand"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border x:Name="Border" Background="#FFB8D0E4">

<ContentPresenter VerticalAlignment="Center" HorizontalAlignment="Center" />

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style TargetType="ComboBox">

<Setter Property="FontStyle" Value="Normal" />

<Setter Property="FontFamily" Value="Comic Sans" />

<Setter Property="FontSize" Value="20"/>

<Setter Property="FontWeight" Value="Bold"/>

<Setter Property="Height" Value="27"/>

<Setter Property="Foreground" Value="#FF575666"/>

<Setter Property="Background" Value="WhiteSmoke"/>

<Setter Property="Margin" Value="10,2,10,2"/>

</Style>

</Window.Resources>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="30" />

<RowDefinition/>

<RowDefinition Height="123"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*" />

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="8\*" />

</Grid.ColumnDefinitions>

<Button x:Name="aboutButton" Content="About" Grid.Column="0" Height="Auto" Click="Click\_aboutButton" />

<Button x:Name="helpButton" Content="Help" Grid.Column="1" Height="Auto" Click="Click\_helpButton"/>

<TextBlock x:Name="busStopNameTextBlock" Margin="2,2,2,0" Grid.Column="2" Grid.Row="0" TextAlignment="Center" Background="#FF719371" Foreground="WhiteSmoke" FontWeight="Bold" Text="Bus stop name"/>

<StackPanel x:Name="mapStack" Grid.Column="2" Grid.Row="1" Grid.RowSpan="1" Background="DarkSeaGreen" Grid.ColumnSpan="1" Margin="2,0,2,0">

<Canvas x:Name="backCanvas" Height="466" Margin="10" MouseMove="MapMouseMove" MouseLeftButtonDown="MapLeftButtonDown" MouseLeftButtonUp="MapLeftButtonUp">

<Controls:MapCtrl x:Name="mapCtrl" Margin="0"/>

</Canvas>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" >

<Slider x:Name="zoomSlider" Canvas.Top="450" Canvas.Left="700" TickFrequency="0.25" Height="25" Width="80" Minimum="0.75" Maximum="4" Value="0.75" TickPlacement="BottomRight" ValueChanged="ValueChanged\_zoomSlider"/>

</StackPanel>

</StackPanel>

<StackPanel x:Name="searchPanel" Background="SteelBlue" Margin="2,2,0,2" Grid.Row="1" Grid.ColumnSpan="2" Grid.RowSpan="2" >

<TextBlock Text="Going from..." FontStyle="Italic" Foreground="LightBlue" FontSize="17" FontWeight="Bold"/>

<ComboBox x:Name="dropDownA" Text="From" SelectionChanged="SelectionChanged\_dropDownA"/>

<TextBlock Text="To..." FontStyle="Italic" Foreground="LightBlue" FontSize="17" FontWeight="Bold"/>

<ComboBox x:Name="dropDownB" Text="To" SelectionChanged="SelectionChanged\_dropDownB" />

<TextBlock Text="Time settings..." FontStyle="Italic" Foreground="LightBlue" FontWeight="Bold" FontSize="17"/>

<RadioButton Name="departafterRadBut" Content="Depart after" Margin="2" FontStyle="Italic" Foreground="LightBlue" FontSize="17" HorizontalAlignment="Center" IsChecked="True"/>

<RadioButton Name="arrivebeforeRadBut" Content="Arrive before" Margin="2" FontStyle="Italic" Foreground="LightBlue" FontSize="17" HorizontalAlignment="Center"/>

<StackPanel x:Name="timerPanel" Orientation="Horizontal" HorizontalAlignment="Center" Margin="5">

<ComboBox Name="hoursComboBox" Width="50" Margin ="3" HorizontalAlignment="Right" TextBlock.TextAlignment="Right"/>

<TextBlock Text=":" Height="27" FontWeight="Bold" Margin="1"/>

<ComboBox Name="minCombobox" Width="50" Margin ="3" />

</StackPanel>

<TextBlock Text="Bus change risk..." FontStyle="Italic" Foreground="LightBlue" FontWeight="Bold" FontSize="17"/>

<StackPanel x:Name="busChangeRiskPanel" Orientation="Horizontal" HorizontalAlignment="Center" Margin="5">

<StackPanel Orientation="Vertical">

<Button x:Name="busRiskupBtn" Content=" up " FontStyle="Italic" Foreground="#FF575666" FontWeight="Bold" FontSize="10" Click="Click\_busRiskupBtn"/>

<Button x:Name="busRiskdownBtn" Content=" down " FontStyle="Italic" Foreground="#FF575666" FontWeight="Bold" FontSize="10" Click="Click\_busRiskdownBtn"/>

</StackPanel>

<TextBlock x:Name="busChangeRiskTxbl" Text="0" FontWeight="Bold" Height="27" Foreground="LightBlue"/>

<TextBlock Text="min" Height="27" Margin="1" Foreground="LightBlue"/>

</StackPanel>

<Calendar x:Name="calendarControl" HorizontalAlignment="Center" VerticalAlignment="Top" />

<Button x:Name="calcButton" Content="Calculate" Margin="10,2,10,2" Click="Click\_calcButton"/>

</StackPanel>

<StackPanel Background="#FFD69758" Grid.Column="2" Grid.Row="2" Margin="2,2,2,2">

<Grid Margin="2,0,1,0" x:Name="header">

<Grid.RowDefinitions>

<RowDefinition Height="1\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0"/>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="1.35\*"/>

</Grid.ColumnDefinitions>

<TextBlock x:Name="dateTxbl" Text="Date" Grid.Column="1" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="fromTxbl" Text="From" Grid.Column="2" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="toTxbl" Text="To" Grid.Column="3" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="departTimeTxbl" Text=" Depart at " Grid.Column="4" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="arriveTimeTxbl" Grid.Column="5" Text=" Arrive at " Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="totalTimeTxbl" Grid.Column="6" Text=" Duration " Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="changesTxbl" Grid.Column="7" Text=" Changes " Style="{StaticResource ResultTableTextBlock}"/>

<Button x:Name="clearButton" Background="#FFE48E12" BorderBrush="#FFE48E12" Content="Clear" Margin="2" Click="Click\_clearButton" Height="Auto" Grid.Column="8" Grid.ColumnSpan="1"/>

</Grid>

<ListBox Name="resultStack" HorizontalContentAlignment="Stretch" Grid.Column="2" Grid.Row="2" Margin="-1,-1,1,-1" Background="Transparent" Grid.ColumnSpan="1" Grid.RowSpan="1" BorderBrush="Transparent" MaxHeight="97" ScrollViewer.VerticalScrollBarVisibility="Visible" SelectionChanged="SelectionChanged\_resultStack"/>

</StackPanel>

</Grid>

</Window>

## SplashWindow

<Window x:Class="TravelConn.SplashWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="SplashWindow" Height="140" Width="300" WindowStyle="None" ResizeMode="NoResize" WindowStartupLocation="CenterScreen">

<Grid Margin="5">

<Grid.Background>

<LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">

<GradientStop Color="#FF5595CB" Offset="0"/>

<GradientStop Color="#FF125185" Offset="1"/>

</LinearGradientBrush>

</Grid.Background>

<Grid.RowDefinitions>

<RowDefinition Height="1\*"/>

<RowDefinition Height="1\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

</Grid.ColumnDefinitions>

<TextBlock Text="TravelConn - Loading..." Foreground="LightBlue" FontStyle="Italic" FontSize="20" TextAlignment="Center" HorizontalAlignment="Center" VerticalAlignment="Center" Margin="0,30,0,0" />

<TextBlock Text="Please Wait..." Foreground="LightBlue" FontStyle="Italic" FontSize="14" TextAlignment="Center" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Row="1" Margin="0,0,0,30"/>

<TextBlock Text="source" Name="sourceTxtBlk" Foreground="LightBlue" FontStyle="Italic" FontSize="10" TextAlignment="Center" HorizontalAlignment="Center" VerticalAlignment="Center" Grid.Row="1" Margin="0,0,0,-20" TextWrapping="Wrap"/>

</Grid>

</Window>

## BusStopInfoWindow

<Window x:Class="TravelConn.Windows.BusStopInfoWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="BusStopInfoWindow" MinHeight="300" MinWidth="600" Height="300" Width="600" WindowStyle="ToolWindow" WindowStartupLocation="CenterScreen">

<Grid>

<FlowDocumentScrollViewer x:Name="pageViewer" Cursor="Arrow" MaxZoom="100" MinZoom="100" ZoomIncrement="100">

<FlowDocument Background="WhiteSmoke">

<Section x:Name="textField"/>

</FlowDocument>

</FlowDocumentScrollViewer>

</Grid>

</Window>

## JourneyInfoWindow

<Window x:Class="TravelConn.JourneyInfoWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="JourneyInfoWindow" MinHeight="300" MinWidth="600" Height="300" Width="600" WindowStyle="ToolWindow" WindowStartupLocation="CenterScreen">

<Grid>

<FlowDocumentScrollViewer x:Name="pageViewer" Cursor="Arrow" MaxZoom="100" MinZoom="100" ZoomIncrement="100">

<FlowDocument Background="#FF85B3D8">

<Section x:Name="textField"/>

</FlowDocument>

</FlowDocumentScrollViewer>

</Grid>

</Window>

<Window x:Class="TravelConn.Windows.AboutWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="AboutWindow" Height="300" Width="300" WindowStartupLocation="CenterScreen" WindowStyle="ToolWindow">

<Grid>

<TextBlock Text="TravalConn is a bus journey finder. For any issues please contact the developer Mateusz Ochal" TextWrapping="Wrap" Margin="30" TextAlignment="Center"/>

</Grid>

</Window>

## BusStopCtrl

<UserControl x:Class="TravelConn.Controls.BusStopCtrl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

mc:Ignorable="d"

d:DesignHeight="30" d:DesignWidth="30">

<UserControl.Resources>

<Style x:Key="baseStyle" TargetType="Ellipse">

<Setter Property="Stroke">

<Setter.Value>

<SolidColorBrush Color="#FF822337" Opacity="0.4"/>

</Setter.Value>

</Setter>

<Setter Property="StrokeThickness" Value="7"/>

</Style>

<Style x:Key="buschange" BasedOn="{StaticResource baseStyle}" TargetType="Ellipse">

<Setter Property="Fill" Value="#FF49A4D8" />

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Fill" Value="#FF93C2DC"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="start" BasedOn="{StaticResource baseStyle}" TargetType="Ellipse">

<Setter Property="Fill" Value="#FF2FD136"/>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Fill" Value="#FF2F6D36"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="end" BasedOn="{StaticResource baseStyle}" TargetType="Ellipse">

<Setter Property="Fill" Value="#FFFF4C4C"/>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Fill" Value="#FF9B4C4C"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="resting" BasedOn="{StaticResource baseStyle}" TargetType="Ellipse">

<Setter Property="Fill" Value="Gold"/>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Fill" Value="#FFFF7300"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="injourney" BasedOn="{StaticResource baseStyle}" TargetType="Ellipse">

<Setter Property="Fill" Value="#FFFF7300"/>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Fill" Value="#FFFF3A1B"/>

</Trigger>

</Style.Triggers>

</Style>

</UserControl.Resources>

<Grid>

<Ellipse Name="ellipseOnMap" Style="{StaticResource resting}" Height="30" Width="30" />

</Grid>

</UserControl>

## LinkCtrl

<UserControl x:Class="TravelConn.Controls.LinkCtrl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

mc:Ignorable="d" >

<Grid>

<Line Name="Line" x:FieldModifier="private" StrokeThickness="3" Stroke="Green" Fill="#FF6EB86E" />

</Grid>

</UserControl>

## MapCtrl

<UserControl x:Class="TravelConn.Controls.MapCtrl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

mc:Ignorable="d"

d:DesignHeight="495" d:DesignWidth="780">

<UserControl.Resources>

<ContextMenu x:Key="ContextMenu">

<MenuItem Header="set as start" Name="startStop" Click="startStop\_Click"/>

<MenuItem Header="set as destination" Name ="endStop" Click="endStop\_Click"/>

<MenuItem Header="More information" Name="moreInfo" Click="moreInfo\_Click"/>

</ContextMenu>

</UserControl.Resources>

<Grid>

<Canvas x:Name="canvas" Height="495" Width="780" Background="Transparent" />

</Grid>

</UserControl>

## ResultTableCtrl

<UserControl x:Class="TravelConn.Controls.ResultTableCtrl"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

mc:Ignorable="d" Height="30">

<UserControl.Resources>

<Style TargetType="Grid">

<Setter Property="Background" Value="#FF2B6BA0"/>

<Setter Property="Margin" Value="0"/>

</Style>

<Style x:Key="ResultTableTextBlock" TargetType="TextBlock">

<Setter Property="FontStyle" Value="Italic" />

<Setter Property="FontWeight" Value="Normal" />

<Setter Property="FontFamily" Value="Comic Sans" />

<Setter Property="FontSize" Value="14"/>

<Setter Property="Foreground" Value="DarkBlue"/>

<Setter Property="Margin" Value="1,3,1,2"/>

<Setter Property="TextAlignment" Value="Center"/>

<Setter Property="VerticalAlignment" Value="Center"/>

<Setter Property="TextWrapping" Value="Wrap"/>

</Style>

<Style x:Key="ResultTableButton" TargetType="Button">

<Setter Property="FontStyle" Value="Normal" />

<Setter Property="FontFamily" Value="Comic Sans" />

<Setter Property="FontSize" Value="10"/>

<Setter Property="FontWeight" Value="Normal"/>

<Setter Property="Foreground" Value="DarkBlue"/>

<Setter Property="BorderThickness" Value="3"/>

<Setter Property="VerticalContentAlignment" Value="Center"/>

<Setter Property="Margin" Value="0,2,2,2"/>

<Setter Property="Cursor" Value="Hand"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="Button">

<Border x:Name="Border" Background="WhiteSmoke">

<ContentPresenter VerticalAlignment="Center" HorizontalAlignment="Center" />

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<Style x:Key="resting" TargetType="Grid">

<Setter Property="Background" Value="SteelBlue"/>

</Style>

<Style x:Key="highlighted" TargetType="Grid">

<Setter Property="Background" Value="#FF85B3D8"/>

</Style>

</UserControl.Resources>

<Grid Margin="0" x:Name="ResultTableGrid" Style="{StaticResource resting}" MouseEnter="ResultTableGrid\_MouseEnter" MouseLeave="ResultTableGrid\_MouseLeave">

<Grid.RowDefinitions>

<RowDefinition Height="1\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="1\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="2\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="0.9\*"/>

<ColumnDefinition Width="0.5\*"/>

<ColumnDefinition Width="0.6\*"/>

</Grid.ColumnDefinitions>

<TextBlock x:Name="dateTxbl" Text="Date" Grid.Column="0" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="fromTxbl" Text="From" Grid.Column="1" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="toTxbl" Text="To" Grid.Column="2" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="departTimeTxbl" Text="Time departure" Grid.Column="3" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="arriveTimeTxbl" Grid.Column="4" Text="Time arrival" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="totalTimeTxbl" Grid.Column="5" Text="Total duration" Style="{StaticResource ResultTableTextBlock}"/>

<TextBlock x:Name="changesTxbl" Grid.Column="6" Text="Changes" Style="{StaticResource ResultTableTextBlock}"/>

<Button x:Name="expandBtn" Grid.Column="7" Content="more" Style="{StaticResource ResultTableButton}" Click="moreBtn\_Click"/>

<Button x:Name="printBtn" Grid.Column="8" Content="print" Style="{StaticResource ResultTableButton}" Click="printBtn\_Click"/>

</Grid>

</UserControl>

# **C#**

## **BusNetwork Classes**

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Shapes;

namespace TravelConn

{

/// CBus, CBusStop, CLink, CTimetableRecord, CDayCategory, CSpecialDay and CDayOfWeek

/// are classes that match the tables in the database.

public class CBus

{

//Core properties

public int busID { get; set; }

public string name { get; set; }

public string description { get; set; }

}

public class CBusStop

{

//Core properties

public int busStopID { get; set; }

public string name { get; set; }

public double longitude { get; set; }

public double latitude { get; set; }

public string address { get; set; }

//Algorithm use

public List<CLink> departureLinks;

public List<CLink> arrivalLinks;

public CTimetableRecord bestLinkRecord;

//type - identifies what type of bus stop it is

//i.e. start or end of journey bus stop, bus change bus stop, in journey bus stop,

//This field will be read off and will determine the colour of the visual component

public string currentType = new CBusStopType().resting();

public CBusStop() { } // empty constructor

public CBusStop(CBusStop model)

{

busStopID = model.busStopID;

name = model.name;

longitude = model.longitude;

latitude = model.latitude;

address = model.address;

departureLinks = model.departureLinks;

arrivalLinks = model.arrivalLinks;

bestLinkRecord = model.bestLinkRecord;

}

}

public class CLink

{

///Core properties

public int LinkID { get; set; }

public CBus bus { get; set; }

public CBusStop busStopA { get; set; }

public CBusStop busStopB { get; set; }

//Algorithm use

public List<CTimetableRecord> linkTimetable;

}

public class CTimetableRecord

{

//Core properties

public CDayCategory dayCategory { get; set; }

public CLink link { get; set; }

public int course { get; set; }

public int departTime { get; set; }

public int arrivalTime { get; set; }

}

public class CDayCategory

{

//Core properties

public int dayCatID { get; set; }

public string description { get; set; }

}

public class CSpecialDay

{

//Core properties

public DateTime dayDate { get; set; }

public CDayCategory dayCat { get; set; }

}

public class CDayOfWeek{

public string dayOfWeek { get; set; }

public CDayCategory dayCat { get; set; }

}

/// <summary>

/// CBusNetwork is the core class that includes all the procedure needed by the algorithm independent of the UI

/// </summary>

public class CBusNetwork

{

/// Lists bellow act as a database in object form

public List<CBus> buses;

public List<CBusStop> busStops;

public List<CLink> links;

public List<CDayCategory> dayCategories;

public List<CSpecialDay> specialDays;

public List<CDayOfWeek> daysOfWeek;

public CDayCategory currentDayCat;

private int change\_risk = 0;

public int changeRisk

{

get { return change\_risk; }

set {

if ((value <= 20)&&(value >= 0))

change\_risk = value;

}

}

/// Read only properties

public string dataSource { get { return data\_source; } }

public string path { get { return path\_; } }

private string data\_source; // could be "file" or "database"

private string path\_; // could be the path to folder with the text files or the database connection string

/// Constructor

public CBusNetwork(string source,string path) {

buses = new List<CBus>();

busStops = new List<CBusStop>();

links = new List<CLink>();

dayCategories = new List<CDayCategory>();

specialDays = new List<CSpecialDay>();

daysOfWeek = new List<CDayOfWeek>();

this.data\_source = source;

this.path\_ = path;

}

/// Method publically visible to load core data.

/// It decides whether to load from a database or a file,

/// The decision is based on the datasource set during instantiatiation of the CBusNetwork class

public bool LoadCoreData()

{

try

{

if (data\_source == "file")

GetCoreFileData();

else if (data\_source == "database")

GetCoreDBsData();

this.currentDayCat = GetDayCatByDate(DateTime.Today);

return true;

}

catch

{

return false;

}

}

//Loads the timetable according to the data source

public bool LoadTimetable()

{

try

{

if (data\_source == "file")

GetFileTimetable(currentDayCat.dayCatID);

else if (data\_source == "database")

GetDBsTimetable(currentDayCat.dayCatID);

return true;

}

catch

{

return false;

}

}

/// Loading data from a database

/// Data will be converted into string and then into appropiate types

private void GetCoreDBsData()

{

SqlConnection DBConnection = new SqlConnection(this.path\_);

DBConnection.Open();

string[] elements = new string[6];

SqlCommand DBCommand = new SqlCommand("Select \* From BusStops;", DBConnection);

SqlDataReader DBReader = DBCommand.ExecuteReader();

while (DBReader.Read())

{

for (int i = 0; i < 5; i++)

{ // will change according to the number of columns

elements[i] = DBReader.GetSqlValue(i).ToString();

}

CBusStop newBusStop = new CBusStop();

newBusStop.busStopID = Convert.ToInt32(elements[0]);

newBusStop.name = elements[1];

newBusStop.latitude = Convert.ToDouble(elements[2]);

newBusStop.longitude = Convert.ToDouble(elements[3]);

newBusStop.address = elements[4];

newBusStop.departureLinks = new List<CLink>();

newBusStop.arrivalLinks = new List<CLink>();

busStops.Add(newBusStop);

}

DBReader.Close();

DBCommand = new SqlCommand("Select \* From DayCategories;", DBConnection);

DBReader = DBCommand.ExecuteReader();

while (DBReader.Read())

{

for (int i = 0; i < 2; i++)

{

elements[i] = DBReader.GetSqlValue(i).ToString();

}

CDayCategory newDayCat = new CDayCategory();

newDayCat.dayCatID = Convert.ToInt32(elements[0]);

newDayCat.description = elements[1];

dayCategories.Add(newDayCat);

}

DBReader.Close();

DBCommand = new SqlCommand("Select \* From SpecialDays;", DBConnection);

DBReader = DBCommand.ExecuteReader();

while (DBReader.Read())

{

// Ignoring the ID fields

for (int i = 1; i < 3; i++)

{

elements[i] = DBReader.GetSqlValue(i).ToString();

}

CSpecialDay newSpecDay = new CSpecialDay();

newSpecDay.dayDate = DateTime.Parse(elements[1]);

newSpecDay.dayCat = GetDayCatbyID(Convert.ToInt32(elements[2]));

specialDays.Add(newSpecDay);

}

DBReader.Close();

DBCommand = new SqlCommand("Select \* From DaysOfWeek;", DBConnection);

DBReader = DBCommand.ExecuteReader();

while (DBReader.Read())

{

// Ignores the ID field which is first

for (int i = 1; i < 3; i++)

{

elements[i] = DBReader.GetSqlValue(i).ToString();

}

CDayOfWeek newDayOfWeek = new CDayOfWeek();

newDayOfWeek.dayOfWeek = elements[1];

newDayOfWeek.dayCat = GetDayCatbyID(Convert.ToInt32(elements[2]));

daysOfWeek.Add(newDayOfWeek);

}

DBReader.Close();

DBCommand = new SqlCommand("Select \* From Buses;", DBConnection);

DBReader = DBCommand.ExecuteReader();

while (DBReader.Read())

{

for (int i = 0; i < 3; i++)

{ // will change according to the number of columns

elements[i] = DBReader.GetSqlValue(i).ToString();

}

CBus newBus = new CBus();

newBus.busID = Convert.ToInt32(elements[0]);

newBus.name = elements[1];

newBus.description = elements[2];

buses.Add(newBus);

}

DBReader.Close();

DBCommand = new SqlCommand("Select \* From BusLinks;", DBConnection);

DBReader = DBCommand.ExecuteReader();

while (DBReader.Read())

{

for (int i = 0; i < 4; i++)

{ // will change according to the number of columns

elements[i] = DBReader.GetSqlValue(i).ToString();

}

CLink newLink = new CLink();

newLink.LinkID = Convert.ToInt32(elements[0]);

newLink.bus = GetBusByID(Convert.ToInt32(elements[1]));

newLink.busStopA = GetBusStopByID(Convert.ToInt32(elements[2]));

newLink.busStopB = GetBusStopByID(Convert.ToInt32(elements[3]));

newLink.linkTimetable = new List<CTimetableRecord>();

//add to list of Links to bus stop the bus is departing from

newLink.busStopA.departureLinks.Add(newLink);

newLink.busStopB.arrivalLinks.Add(newLink);

links.Add(newLink);

}

DBReader.Close();

}

public void GetDBsTimetable(int dayCatID)

{

SqlConnection DBConnection = new SqlConnection(this.path\_);

DBConnection.Open();

SqlCommand DBCommand = new SqlCommand("Select \* From Timetable;", DBConnection);

SqlDataReader DBReader = DBCommand.ExecuteReader();

string[] elements = new string[6];

while (DBReader.Read())

{

for (int i = 0; i < 5; i++)

{

elements[i] = DBReader.GetSqlValue(i).ToString();

}

if (Convert.ToInt32(elements[0])==dayCatID)

{

CTimetableRecord linkTimetable = new CTimetableRecord();

linkTimetable.dayCategory = GetDayCatbyID(Convert.ToInt32(elements[0]));

linkTimetable.link = GetLinkByID(Convert.ToInt32(elements[1]));

linkTimetable.course = Convert.ToInt32(elements[2]);

linkTimetable.departTime = Convert.ToInt32(elements[3]);

linkTimetable.arrivalTime = Convert.ToInt32(elements[4]);

//add the timetable record to each bus stop timetable

linkTimetable.link.linkTimetable.Add(linkTimetable);

}

}

DBReader.Close();

}

private void GetCoreFileData()

{

string[] lines = File.ReadAllLines(this.path\_ + "DayCategories.txt");

foreach (String line in lines)

{

if (line != lines[0])

{

string[] elements = line.Split('\t');

CDayCategory newDayCat = new CDayCategory();

newDayCat.description = elements[1];

newDayCat.dayCatID = Convert.ToInt32(elements[0]);

dayCategories.Add(newDayCat);

}

}

lines = File.ReadAllLines(this.path\_ + "BusStops.txt");

foreach (String line in lines)

{

if (line != lines[0])

{

string[] elements = line.Split('\t');

CBusStop newBusStop = new CBusStop();

newBusStop.busStopID = Convert.ToInt32(elements[0]);

newBusStop.name = elements[1];

newBusStop.latitude = Convert.ToDouble(elements[2]);

newBusStop.longitude = Convert.ToDouble(elements[3]);

newBusStop.address = elements[4];

newBusStop.departureLinks = new List<CLink>();

newBusStop.arrivalLinks = new List<CLink>();

busStops.Add(newBusStop);

}

}

lines = File.ReadAllLines(this.path\_ + "Buses.txt");

foreach (String line in lines)

{

if (line != lines[0])

{

string[] elements = line.Split('\t');

CBus newBus = new CBus();

newBus.busID = Convert.ToInt32(elements[0]);

newBus.name = elements[1];

newBus.description = elements[2];

buses.Add(newBus);

}

}

lines = File.ReadAllLines(this.path\_ + "Links.txt");

foreach (String line in lines)

{

if (line != lines[0])

{

string[] elements = line.Split('\t');

CLink newLink = new CLink();

newLink.LinkID = Convert.ToInt32(elements[0]);

newLink.bus = GetBusByID(Convert.ToInt32(elements[1]));

newLink.busStopA = GetBusStopByID(Convert.ToInt32(elements[2]));

newLink.busStopB = GetBusStopByID(Convert.ToInt32(elements[3]));

newLink.linkTimetable = new List<CTimetableRecord>();

//add to list of Links to bus stop the bus is departing from

newLink.busStopA.departureLinks.Add(newLink);

newLink.busStopB.arrivalLinks.Add(newLink);

links.Add(newLink);

}

}

lines = File.ReadAllLines(this.path\_ + "SpecialDays.txt");

foreach (String line in lines)

{

if (line != lines[0])

{

string[] elements = line.Split('\t');

CSpecialDay newSpecDay = new CSpecialDay();

newSpecDay.dayDate = new DateTime(Convert.ToInt32(elements[0].Split('-')[0]), Convert.ToInt32(elements[0].Split('-')[1]), Convert.ToInt32(elements[0].Split('-')[2]));

newSpecDay.dayCat = GetDayCatbyID(Convert.ToInt32(elements[1]));

specialDays.Add(newSpecDay);

}

}

lines = File.ReadAllLines(this.path\_ + "DaysOfWeek.txt");

foreach (String line in lines)

{

if (line != lines[0])

{

string[] elements = line.Split('\t');

CDayOfWeek newDayOfWeek = new CDayOfWeek();

newDayOfWeek.dayOfWeek = elements[0];

newDayOfWeek.dayCat = GetDayCatbyID(Convert.ToInt32(elements[1]));

daysOfWeek.Add(newDayOfWeek);

}

}

}

//get a timetable from a file

public void GetFileTimetable(int dayCatID)

{

string[] lines = File.ReadAllLines(this.path\_ + "Timetable.txt");

foreach (String line in lines)

{

if (line != lines[0])

{

string[] elements = line.Split('\t');

if (Convert.ToInt32(elements[0]) == dayCatID)

{

CTimetableRecord linkTimetable = new CTimetableRecord();

linkTimetable.dayCategory = GetDayCatbyID(Convert.ToInt32(elements[0]));

linkTimetable.link = GetLinkByID(Convert.ToInt32(elements[1]));

linkTimetable.course = Convert.ToInt32(elements[2]);

linkTimetable.departTime = Convert.ToInt32(elements[3]);

linkTimetable.arrivalTime = Convert.ToInt32(elements[4]);

//add the timetable record to each bus stop timetable

linkTimetable.link.linkTimetable.Add(linkTimetable);

}

}

}

}

// Get a day category according to the date

// Looks up the special days and the day of week to determine which day category belongs to the date

public CDayCategory GetDayCatByDate(DateTime date)

{

int i = 0;

while ((specialDays[i].dayDate != date.Date) && (i < specialDays.Count() - 1))

{

i++;

}

if (specialDays[i].dayDate == date.Date)

{

return specialDays[i].dayCat;

}

else

{

i = 0;

while ((daysOfWeek[i].dayOfWeek != date.DayOfWeek.ToString()))

{

i++;

}

return (daysOfWeek[i].dayCat);

}

}

// Get bus stop by id

public CBusStop GetBusStopByID(int id)

{

int i = 0;

while (busStops[i].busStopID != id)

{

i++;

}

return busStops[i];

}

//Gets a bus stop by name, this means that the name also has to be unique

// This method is used in the Main Window file

public CBusStop GetBusStopByName(string name)

{

int i = 0;

while (busStops[i].name != name)

{

i++;

}

return busStops[i];

}

// Get bus by ID

public CBus GetBusByID(int id)

{

int i = 0;

while (buses[i].busID != id)

{

i++;

}

return buses[i];

}

//Get link by ID

public CLink GetLinkByID(int id)

{

int i = 0;

while (links[i].LinkID != id)

{

i++;

}

return links[i];

}

// Get Day Category by ID

public CDayCategory GetDayCatbyID(int id)

{

int i = 0;

while (dayCategories[i].dayCatID != id)

{

i++;

}

return dayCategories[i];

}

/// This is the first of the two main algorithms for calculating the shortest path

/// Accepting 2 CBusStop objects and time as integer will output a CTimetableRecord for each link in the calculated journey

public List<CTimetableRecord> CalcDepartAfterPath(CBusStop startBusStop, CBusStop endBusStop, int time)

{

/// Recreating all the best links so that no data is left behind when running the algorithm for the next time

RecreatebestLinkRecords(true);

/// A new list of all the bus stops which have not been visited by the algorithm yet

List<CBusStop> unvisitedBusStops = new List<CBusStop>(busStops);

/// currentBusStop will keep track of the bus stop the algorithm uses at an instance.

/// Starts at the startBusStop

CBusStop currentBusStop = startBusStop;

/// This loop finds bestLinkRecord for each of the first bus stop neighbours

/// Since it doesn't take the busChangeRisk into consideration it's a special case and is outside the main loop;

/// it doesn't take the busChangeRisk into consideration because the journey can begin by taking any bus departing from the bus stop;

foreach (CLink link in currentBusStop.departureLinks)

{

foreach (CTimetableRecord record in link.linkTimetable)

{

/// If the "departure time" from the "current bus stop" is after or equal to the set "time"

/// AND

/// the "arrival time" of that timetable record, beats the "arrival time" of the "current best link"

/// THEN overwrite the "best link" with the timetable record that was just found

if ((record.departTime >= time) && (record.arrivalTime < link.busStopB.bestLinkRecord.arrivalTime))

{

link.busStopB.bestLinkRecord = record;

}

}

}

/// After working out the best links to the neighboring bus stops of the start bus stop,

/// remove the current bus stop (start bus stop) from the unvisited bus stops list

unvisitedBusStops.Remove(currentBusStop);

///FindQuickestArrival given a list of bus stops will chooses the one with the lowest/best "arrival time" store in the "best link record" of each bus stop

currentBusStop = FindQuickestArrival(unvisitedBusStops);

unvisitedBusStops.Remove(currentBusStop);

/// This is the main loop. It works by finding the "best link record" like in the loop above.

/// Going through each bus stop in turn,

/// until the "current bus stop" is the "end bus stop"

/// OR

/// When ALL bus stops have been visited

/// OR

/// When the best link of current bus stop is 9999 (this implys that no valid records have been found)

while ((currentBusStop != endBusStop) && (unvisitedBusStops.Count > 0) && (currentBusStop.bestLinkRecord.arrivalTime != 9999))

{

foreach (CLink link in currentBusStop.departureLinks){

/// Checks if the next bus stop has been visited already

if (unvisitedBusStops.Contains(link.busStopB))

{

foreach (CTimetableRecord record in link.linkTimetable)

{

/// If the "departure time" of the bus is after the arrival time of the bus to the current bus stop

/// (without taking bus change risk into account)

if (record.departTime >= currentBusStop.bestLinkRecord.arrivalTime)

{

/// If the "arrival time" of that timetable record, beats(is less than) the current best link "arrival time" at the next bus stop

if (record.arrivalTime <= record.link.busStopB.bestLinkRecord.arrivalTime)

{

/// If the "departure time" from the "current bus stop" is more or equal to the arrival time at currentBusStop

/// taking bus change risk into consideration

if (record.departTime >= AddMinToTime(currentBusStop.bestLinkRecord.arrivalTime,change\_risk))

{

/// THEN overwrite the "best link" with the timetable record that was just found

record.link.busStopB.bestLinkRecord = record;

}

/// Or if the bus and course of the next record is the same as in the bestlinkrecord

else if ((record.course == currentBusStop.bestLinkRecord.course) && (record.link.bus == currentBusStop.bestLinkRecord.link.bus))

{

/// THEN overwrite the "best link" with the timetable record that was just found

record.link.busStopB.bestLinkRecord = record;

}

}

}

}

}

}

//Choose the busstop with the quickest arrival time out of the unvisited bus stops

currentBusStop = FindQuickestArrival(unvisitedBusStops);

unvisitedBusStops.Remove(currentBusStop);

}

//Get the route by looping back from the end bus stop to the start bus stop

List<CTimetableRecord> quickestJourney = new List<CTimetableRecord>();

//if all bus stops have been visited but the arrival time at the end bus stop is the default time it means that there was no route found

if (endBusStop.bestLinkRecord.arrivalTime != 9999)

{

currentBusStop = endBusStop;

quickestJourney.Add(currentBusStop.bestLinkRecord);

while ((currentBusStop.bestLinkRecord.link.busStopA != startBusStop))

{

currentBusStop = currentBusStop.bestLinkRecord.link.busStopA;

quickestJourney.Add(currentBusStop.bestLinkRecord);

}

quickestJourney.Reverse();

}

else quickestJourney = null;

return quickestJourney;

}

/// In principle this method works in the same way as the CalcDepartAfterPath() method.

/// It differs by accessing bus stops' arrival links in the main loop rather than the departure links.

public List<CTimetableRecord> CalcArriveBeforePath(CBusStop startBusStop, CBusStop endBusStop, int time)

{

/// Recreating all the best links so that no data is left behind when running the algorithm for the next time

/// The passing parameter false indicates that it is not a depart after algorithm,

/// and to fill the bus stop's best links with negative values

RecreatebestLinkRecords(false);

/// A new list of all the bus stops which have not been visited by the algorithm yet

List<CBusStop> unvisitedBusStops = new List<CBusStop>(busStops);

/// currentBusStop will keep track of the bus stop the algorithm uses at an instance.

/// Starts at the endBusStop

CBusStop currentBusStop = endBusStop;

currentBusStop.bestLinkRecord.arrivalTime = time;

/// Visiting the first bus stop (which is current bus stop)

/// This loop finds bestLinkRecord for each of the first bus stop neighbours

/// Since it doesn't take the busChangeRisk into consideration it's a special case and is outside the main loop;

/// it doesn't take the busChangeRisk into consideration because the journey can begin by taking any bus departing from the bus stop;

foreach (CLink link in currentBusStop.arrivalLinks)

{

foreach (CTimetableRecord record in link.linkTimetable)

{

/// If the "arrival time" to the "current bus stop" is more or equal to the "time"

/// AND

/// the "departure time" of that timetable record, beats the "departure time" of the "current best link"

/// THEN overwrite the "best link" with the timetable record that was just found

if ((record.arrivalTime <= time) && (record.departTime > link.busStopA.bestLinkRecord.departTime))

{

link.busStopA.bestLinkRecord = record;

}

}

}

/// Removes the endbusStop from the unvisited

unvisitedBusStops.Remove(currentBusStop);

/// Main loop

while ((currentBusStop != startBusStop) && (unvisitedBusStops.Count > 0) && (currentBusStop.bestLinkRecord.arrivalTime != -9999))

{

foreach (CLink link in currentBusStop.arrivalLinks)

{

/// Checks if the next bus stop has been visited already

if (unvisitedBusStops.Contains(link.busStopA))

{

foreach (CTimetableRecord record in link.linkTimetable)

{

/// If the "arrival time" of the bus is before the departure time of the bus to the current bus stop

/// (without taking bus change risk into account)

if (record.arrivalTime <= currentBusStop.bestLinkRecord.departTime)

{

/// If the "departure time" of that timetable record, beats(is less than) the current best link "depart time" at the next bus stop

if (record.departTime >= record.link.busStopA.bestLinkRecord.departTime)

{

/// If the "arrival time" from the "current bus stop" is more or equal to the departure time at currentBusStop

/// taking bus change risk into consideration

if (record.arrivalTime <= AddMinToTime(currentBusStop.bestLinkRecord.departTime, change\_risk))

{

/// THEN overwrite the "best link" with the timetable record that was just found

record.link.busStopA.bestLinkRecord = record;

}

/// Or if the bus and course of the next record is the same as in the bestlinkrecord

else if ((record.course == currentBusStop.bestLinkRecord.course) && (record.link.bus == currentBusStop.bestLinkRecord.link.bus))

{

/// THEN overwrite the "best link" with the timetable record that was just found

record.link.busStopA.bestLinkRecord = record;

}

}

}

}

}

}

//Choose the busstop with the quickest arrival time out of the unvisited bus stops

currentBusStop = FindLatestArrival(unvisitedBusStops);

unvisitedBusStops.Remove(currentBusStop);

}

//Get the route by looping back from the end bus stop to the start bus stop

List<CTimetableRecord> quickestJourney = new List<CTimetableRecord>();

if (currentBusStop.bestLinkRecord.arrivalTime != -9999)

{

currentBusStop = startBusStop;

quickestJourney.Add(currentBusStop.bestLinkRecord);

while ((currentBusStop.bestLinkRecord.link.busStopB != endBusStop))

{

currentBusStop = currentBusStop.bestLinkRecord.link.busStopB;

quickestJourney.Add(currentBusStop.bestLinkRecord);

}

}

else quickestJourney = null;

return quickestJourney;

}

// Resets the best links to

public void RecreatebestLinkRecords(bool departAfter)

{

foreach (CBusStop busStop in busStops)

{

busStop.bestLinkRecord = new CTimetableRecord();

if (departAfter)

{

busStop.bestLinkRecord.arrivalTime = 9999;

busStop.bestLinkRecord.departTime = 9999;

}

else

{

busStop.bestLinkRecord.arrivalTime = -9999;

busStop.bestLinkRecord.departTime = -9999;

}

busStop.bestLinkRecord.link = new CLink();

busStop.bestLinkRecord.course = 0;

busStop.bestLinkRecord.dayCategory = new CDayCategory();

}

}

// returns a bus stop with the quickest/smallest arrival time

public CBusStop FindQuickestArrival(List<CBusStop> list)

{

CBusStop tempBusStop = new CBusStop();

tempBusStop.bestLinkRecord = new CTimetableRecord();

tempBusStop.bestLinkRecord.arrivalTime = 9999;

foreach (CBusStop busStop in list)

{

if (busStop.bestLinkRecord.arrivalTime < tempBusStop.bestLinkRecord.arrivalTime)

tempBusStop = busStop;

}

return tempBusStop;

}

// returns a bus stop with the latest/largests arrival time of it's best link

public CBusStop FindLatestArrival(List<CBusStop> list)

{

CBusStop tempBusStop = new CBusStop();

tempBusStop.bestLinkRecord = new CTimetableRecord();

tempBusStop.bestLinkRecord.arrivalTime = -9999;

foreach (CBusStop busStop in list)

{

if (busStop.bestLinkRecord.arrivalTime >= tempBusStop.bestLinkRecord.arrivalTime)

tempBusStop = busStop;

}

return tempBusStop;

}

//This function adds mins to time

public int AddMinToTime(int time, int min)

{

int tempTime = time;

int count = 0;

while (tempTime >= 100)

{

tempTime -= 100;

count++;

}

tempTime += min;

min = tempTime % 60;

return ((count + tempTime / 60) \* 100 + min);

}

}

//The methods in this class return the bus stop type as a string.

//This class was made to keep the Types the same to avoid human spelling error.

public class CBusStopType{

private string current\_;

public string currenttype {get {return current\_;}}

public string start() { current\_ = "start"; return current\_; }

public string end() { current\_ = "end"; return current\_; }

public string injourney() { current\_ = "injourney"; return current\_; }

public string resting() { current\_ = "resting"; return current\_; }

public string buschange() { current\_ = "buschange"; return current\_; }

}

}

## MainWindow

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.Drawing;

using TravelConn.Controls;

namespace TravelConn

{

public partial class MainWindow : Window

{

/// global private variable

/// busNetwork - this object includes all the core objects used for the algorithm

private CBusNetwork busNetwork;

/// These variables are used for dragging the map

/// Dragging the map

private Point mouse\_Position;

private Point last\_Pos;

private bool is\_Dragging = false;

/// Initialising the Main Window

public MainWindow()

{

/// splashWindow - used to inform the user that the program is doing something

/// when there is a lot of data loading and creating objects can use up a lot of time

SplashWindow splashWindow = new SplashWindow("");

splashWindow.Show();

/// Initialises all the main window XAML components

InitializeComponent();

/// Creates the bus network object

busNetwork = new CBusNetwork("file", @"TestNetwork\");

/// An example of a string that has to be passed when connecting to a database

/// "database", "Server=.\\SQLEXPRESS;Database=TravelConnDB;Integrated Security=true"

/// "file", "C:\TestTimetable\"

bool coreLoaded = busNetwork.LoadCoreData();

bool timetableLoaded = busNetwork.LoadTimetable();

// Checks whether to proceed or not

if (coreLoaded && timetableLoaded)

{

/// Initialises all the UI components that are dependent on the data just loaded

mapCtrl.Create(busNetwork.busStops, busNetwork.links);

mapCtrl.Draw(730, 460);

mapCtrl.displayTxtBl = busStopNameTextBlock;

mapCtrl.fromComboBox = dropDownA;

mapCtrl.toComboBox = dropDownB;

/// Add appropiate options to the drop downs

IniDropDowns();

/// Hides the splash scren

splashWindow.Close();

}

else

{

MessageBox.Show("Coundn't load the data! Make sure all directories are correct, and that the data is valid!", "TravalConn - Loading", MessageBoxButton.OK, MessageBoxImage.Exclamation);

this.Close();

/// Hides the splash scren

splashWindow.Close();

}

}

//Main Window Closing

private void Closing\_mainWindow(object sender, System.ComponentModel.CancelEventArgs e)

{

//Asks before closing

var result = MessageBox.Show("Would you like to close TravalConn?", "TravalConn - Closing", MessageBoxButton.YesNo, MessageBoxImage.Question);

if (result == MessageBoxResult.No)

{

// Cancel the Closing event from closing the form.

e.Cancel = true;

}

}

private void IniDropDowns()

{

/// Go through each busStop in the busNetwork and add their names to the drop down menus

foreach (CBusStop busStop in busNetwork.busStops)

{

dropDownA.Items.Add(busStop.name);

dropDownB.Items.Add(busStop.name);

}

/// Add integers 0 to 23, inclusive, to the hour drop down menu

for (int i = 0; i < 24; i++)

{

if (i < 10)

hoursComboBox.Items.Add("0" + i.ToString());

else

hoursComboBox.Items.Add(i.ToString());

}

///Add minutes to the minute drop down menu, every 5 min

for (int i = 0; i < 60; i = i + 5)

{

if (i < 10)

minCombobox.Items.Add("0" + i.ToString());

else

minCombobox.Items.Add(i.ToString());

}

//These where added for testing purposes

//minCombobox.Items.Add("01");

//minCombobox.Items.Add("08");

//minCombobox.Items.Add("12");

//minCombobox.Items.Add("16");

//minCombobox.Items.Add("06");

//minCombobox.Items.Add("04");

///Set the time in the drop downs to be the current time

SetTimeDropBoxes();

}

///Based on the Windows current clock time, sets a default time in the drop downs

private void SetTimeDropBoxes()

{

///Gets the current time

DateTime now = DateTime.Now;

int nowHour = now.Hour;

int nowMin = now.Minute;

///The time is set so that it's always rounding up to the nearst 5 min

///There is a special case when the time is 23:55 because rounding up would run it into the next day

if (nowMin >= 55){

if (nowHour == 23)

nowMin = 55;

else{

nowHour += 1;

nowMin = 0;

}

}

else

nowMin = ((nowMin / 5) + 1) \* 5;

if (nowHour < 10)

hoursComboBox.SelectedItem = "0" + nowHour.ToString();

else

hoursComboBox.SelectedItem = nowHour.ToString();

if (nowMin < 10)

minCombobox.SelectedItem = "0" + nowMin.ToString();

else

minCombobox.SelectedItem = nowMin.ToString();

}

//Interface method to calculate the quickest journey given that details provided passes all validation

private void CalculateJourney(DateTime date, CBusStop startbusStop, CBusStop endbusStop, string hours, string mins)

{

///Convert time to a string form

int time = Convert.ToInt32(hours + mins);

///Instantiate the journey object

List<CTimetableRecord> journey = new List<CTimetableRecord>();

///Chooses appropriate method to calculate the quickest time

if (departafterRadBut.IsChecked == true)

journey = busNetwork.CalcDepartAfterPath(startbusStop, endbusStop, time);

else if (arrivebeforeRadBut.IsChecked == true)

journey = busNetwork.CalcArriveBeforePath(startbusStop, endbusStop, time);

///If returns null display error else add a ResultTable table control to the ResultTable stack list

if (journey == null){

const string message = "No journey found!";

const string caption = "TravalConn - Calculating";

MessageBox.Show(message, caption, MessageBoxButton.OK, MessageBoxImage.Warning);

}

else

{

Controls.ResultTableCtrl resultForm = new Controls.ResultTableCtrl(date, journey);

resultStack.Items.Add(resultForm);

resultStack.SelectedIndex = resultStack.Items.Count -1;

}

}

///When the calculate button is clicked

private void Click\_calcButton(object sender, RoutedEventArgs e)

{

///Display the splashWindow while calculating

SplashWindow splashWin = new SplashWindow("");

splashWin.Show();

//Validates whether the bus stops have been selected from the drop down menus

if ((dropDownA.SelectedItem != null) && (dropDownB.SelectedItem != null))

{

///Gets selected bus stops and time from the drop down menus

string hours = hoursComboBox.SelectedItem.ToString();

string mins = minCombobox.SelectedItem.ToString();

bool loaded = true;

///Checks the date and whether the day category matches the seleced date from the calendar

DateTime date = CheckDateDayCat(ref loaded);

if (loaded)

{

CalculateJourney(date, mapCtrl.startBusStop, mapCtrl.endBusStop, hours, mins);

}

else

{

///Displays an error if could not load the new timetable

const string message = "Timetable count not be loaded!";

const string caption = "TravalConn - Loading";

MessageBox.Show(message, caption, MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

else {

///Displays an error if destination or departure bus stops have not been selected from the bus stop

const string message = "Select start and destination bus stop!";

const string caption = "TravalConn - Calculating";

MessageBox.Show(message, caption, MessageBoxButton.OK, MessageBoxImage.Warning);

}

//Once calculated hide the splash screen

splashWin.Close();

}

// Checks what date was selected and looks if the day categories match;

// reloads timetable if the day category loaded is different

// Returns the date used

private DateTime CheckDateDayCat(ref bool loaded)

{

///Gets the selected date from the calendar

DateTime? selectedDate = calendarControl.SelectedDate;

//the question mark means it is a nullable variable type

//null is returned if a new date was not selected

// default is set to true

// because only changes value when failed to load the new timetable

loaded = true;

if (selectedDate != null){

///Gets the day category of the selected date

CDayCategory selectedDateDayCat = busNetwork.GetDayCatByDate((DateTime)selectedDate);

/// if the day cat. is not the same as the current day cat. then load the correct timetable

if (busNetwork.currentDayCat.dayCatID != selectedDateDayCat.dayCatID){

foreach (CLink link in busNetwork.links){

link.linkTimetable = new List<CTimetableRecord>();

}

busNetwork.currentDayCat = selectedDateDayCat;

loaded = busNetwork.LoadTimetable();

}

return (DateTime)selectedDate;

}

else return DateTime.Today;

}

///Clears all the result Stack forms from the stack list

private void Click\_clearButton(object sender, RoutedEventArgs e)

{

resultStack.Items.Clear();

}

private void SelectionChanged\_dropDownA(object sender, SelectionChangedEventArgs e)

{

mapCtrl.startBusStop = busNetwork.GetBusStopByName(dropDownA.SelectedItem.ToString());

mapCtrl.UpdateMap();

}

private void SelectionChanged\_dropDownB(object sender, SelectionChangedEventArgs e)

{

mapCtrl.endBusStop = busNetwork.GetBusStopByName(dropDownB.SelectedItem.ToString());

mapCtrl.UpdateMap();

}

//bus change risk values and text blocks

private void Click\_busRiskupBtn(object sender, RoutedEventArgs e)

{

busNetwork.changeRisk += 1;

UpdateChangeRiskTxbl();

}

private void Click\_busRiskdownBtn(object sender, RoutedEventArgs e)

{

busNetwork.changeRisk -= 1;

UpdateChangeRiskTxbl();

}

private void UpdateChangeRiskTxbl() {

busChangeRiskTxbl.Text = busNetwork.changeRisk.ToString();

}

// Selection changed of the Result table stack

private void SelectionChanged\_resultStack(object sender, SelectionChangedEventArgs e)

{

foreach (CBusStop busStop in busNetwork.busStops)

{

busStop.currentType = new CBusStopType().resting();

}

foreach (ResultTableCtrl table in resultStack.Items)

{

table.IsSelected = false;

}

ResultTableCtrl selected = (ResultTableCtrl)resultStack.SelectedItem;

if (selected != null)

selected.IsSelected = true;

mapCtrl.UpdateMap();

}

/// When the slider is moved rescale the map

private void ValueChanged\_zoomSlider(object sender, RoutedPropertyChangedEventArgs<double> e)

{

ScaleTransform transformation = new ScaleTransform(zoomSlider.Value,zoomSlider.Value, 390, 242);

mapCtrl.RenderTransform = transformation;

mapCtrl.scale = 1/zoomSlider.Value;

mapCtrl.Scale();

}

// Map clicked

private void MapLeftButtonDown(object sender, MouseEventArgs e)

{

is\_Dragging = true;

var draggable = sender as Canvas;

if (draggable != null)

{

mouse\_Position = e.GetPosition(mapStack);

draggable.CaptureMouse();

Mouse.SetCursor(Cursors.Hand);

}

}

//Left mouse button up; drop the map so that it no longer moves with the mouse

private void MapLeftButtonUp(object sender, MouseEventArgs e)

{

is\_Dragging = false;

var draggable = sender as Canvas;

if (draggable != null)

{

var transform = draggable.RenderTransform as TranslateTransform;

if (transform == null)

{

transform = new TranslateTransform();

draggable.RenderTransform = transform;

}

last\_Pos.X = transform.X;

last\_Pos.Y = transform.Y;

draggable.ReleaseMouseCapture();

Mouse.SetCursor(Cursors.Arrow);

}

}

//Dragging the map across

private void MapMouseMove(object sender, MouseEventArgs e)

{

var draggable = sender as Canvas;

if (is\_Dragging && draggable != null)

{

var transform = draggable.RenderTransform as TranslateTransform;

// First case

if (transform == null)

{

transform = new TranslateTransform();

draggable.RenderTransform = transform;

}

Mouse.SetCursor(Cursors.Hand);

Point currentPosition = e.GetPosition(mapStack);

transform.X = currentPosition.X - mouse\_Position.X + last\_Pos.X;

transform.Y = currentPosition.Y - mouse\_Position.Y + last\_Pos.Y;

Console.Write("X: " + transform.X.ToString());

Console.Write(" Y: " + transform.Y.ToString());

Console.WriteLine();

}

}

// when about button clicked

private void Click\_aboutButton(object sender, RoutedEventArgs e)

{

Windows.AboutWindow aboutWindow = new Windows.AboutWindow();

aboutWindow.Show();

}

// when help button clicked

private void Click\_helpButton(object sender, RoutedEventArgs e)

{

// This has not been implemented yet...

}

}

}

## SplashWindow

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace TravelConn

{

/// <summary>

/// Interaction logic for SplashWindow.xaml

/// </summary>

public partial class SplashWindow : Window

{

public SplashWindow(string source)

{

InitializeComponent();

sourceTxtBlk.Text = source;

}

}

}

## **BusStopInfoWindow**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace TravelConn.Windows

{

/// <summary>

/// Interaction logic for BusStopInfoWindow.xaml

/// </summary>

public partial class BusStopInfoWindow : Window

{

private CBusStop busStop;

public BusStopInfoWindow(CBusStop busStop)

{

InitializeComponent();

this.busStop = busStop;

textField.Blocks.Add(getBasicInfo());

textField.Blocks.Add(getBuses());

}

private Section getBasicInfo()

{

Table table = new Table();

table.Columns.Add(new TableColumn());

table.Columns.Add(new TableColumn());

table.RowGroups.Add(new TableRowGroup());

table.RowGroups[0].Rows.Add(new TableRow());

TableRow currentRow = table.RowGroups[0].Rows[0];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Bus Stop Name:"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(busStop.name))));

table.RowGroups[0].Rows.Add(new TableRow());

currentRow = table.RowGroups[0].Rows[1];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Latitude: "))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(busStop.latitude.ToString()))));

table.RowGroups[0].Rows.Add(new TableRow());

currentRow = table.RowGroups[0].Rows[2];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Longitude: "))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(busStop.longitude.ToString()))));

table.RowGroups[0].Rows.Add(new TableRow());

currentRow = table.RowGroups[0].Rows[3];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Address: "))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(busStop.address.ToString()))));

Section section = new Section();

section.Blocks.Add(table);

return section;

}

private Section getBuses()

{

Section section = new Section();

Table header;

TableRow currentRow = new TableRow();

if (busStop.departureLinks != null)

{

header = new Table();

header.Columns.Add(new TableColumn());

header.RowGroups.Add(new TableRowGroup());

header.RowGroups[0].Rows.Add(new TableRow());

currentRow = header.RowGroups[0].Rows[0];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Departures"))));

header.RowGroups[0].Rows.Add(new TableRow());

currentRow = header.RowGroups[0].Rows[1];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Bus"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Description"))));

int count = 2;

foreach (CLink link in busStop.departureLinks)

{

header.RowGroups[0].Rows.Add(new TableRow());

currentRow = header.RowGroups[0].Rows[count];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(link.bus.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(link.bus.description))));

count++;

}

section.Blocks.Add(header);

}

if (busStop.arrivalLinks != null)

{

header = new Table();

header.Columns.Add(new TableColumn());

header.RowGroups.Add(new TableRowGroup());

header.RowGroups[0].Rows.Add(new TableRow());

currentRow = header.RowGroups[0].Rows[0];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Arrivals"))));

header.RowGroups[0].Rows.Add(new TableRow());

currentRow = header.RowGroups[0].Rows[1];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Bus"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Description"))));

int count = 2;

foreach (CLink link in busStop.arrivalLinks)

{

header.RowGroups[0].Rows.Add(new TableRow());

currentRow = header.RowGroups[0].Rows[count];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(link.bus.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(link.bus.description))));

count++;

}

section.Blocks.Add(header);

}

return section;

}

}

}

## **JourneyInfoWindow**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace TravelConn

{

/// <summary>

/// Interaction logic for JourneyInfoWindow.xaml

/// </summary>

public partial class JourneyInfoWindow : Window

{

public JourneyInfoWindow()

{

InitializeComponent();

}

}

}

**JourneyInfoWindow C# code was generated by Visual Studio automatically and nothing was added to it.**

## **AboutWindow**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace TravelConn.Windows

{

/// <summary>

/// Interaction logic for AboutWindow.xaml

/// </summary>

public partial class AboutWindow : Window

{

public AboutWindow()

{

InitializeComponent();

}

}

}

**AboutWindow C# code was generated by Visual Studio automatically and nothing was added to it.**

## **BusStopCtrl**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace TravelConn.Controls

{

/// <summary>

/// Interaction logic for CBusStopCtrl.xaml

/// </summary>

public partial class BusStopCtrl : UserControl

{

public CBusStop busStop = new CBusStop();

public Point relativePos; // always between 0 and 1;

private string type\_;

public string currentType {

get { return type\_; }

}

public BusStopCtrl()

{

InitializeComponent();

}

public BusStopCtrl(CBusStop busStop, Point relativePos, int d)

{

InitializeComponent();

this.relativePos = relativePos;

this.busStop = busStop;

}

/// bus stop will change colour according to type

public void ChangeType(string type)

{

switch (type)

{

case "resting":

case "start":

case "end":

case "injourney":

case "buschange":

ellipseOnMap.Style = (Style)FindResource(type);

this.type\_ = type;

break;

}

}

/// bus stop will highlight according to state

public void ChangeState(string state)

{

switch (state)

{

case "selected":

ellipseOnMap.Style = (Style)FindResource(type\_);

break;

case "resting":

ellipseOnMap.Style = (Style)FindResource("default");

break;

}

}

}

}

## **LinkCtrl**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace TravelConn.Controls

{

/// <summary>

/// Interaction logic for CLinkCtrl.xaml

/// </summary>

public partial class LinkCtrl : UserControl

{

public LinkCtrl()

{

InitializeComponent();

}

public CLink link = new CLink();

//value between 1 and 0

private Point rel\_p1;

private Point rel\_p2;

//Relative point 2

public Point relPoint1{

set{

if ((value.X >= 0) && (value.X <= 1) && (value.Y >= 0) && (value.Y <= 1))

{

rel\_p1 = value;

}

}

}

//Relative point 2

public Point relPoint2{

set{

if ((value.X >= 0) && (value.X <= 1) && (value.Y >= 0) && (value.Y <= 1))

{

rel\_p2 = value;

}

}

}

//actual points relative to the canvas

public Point canvasP1

{

get

{

Point point = new Point();

point.X = this.Line.X1;

point.Y = this.Line.Y1;

return point;

}

}

public Point canvasP2

{

get

{

Point point = new Point();

point.X = this.Line.X2;

point.Y = this.Line.Y2;

return point;

}

}

public void Draw(double canvasWidth, double canvasHeight)

{

// The offset (+15) is to account for the size of the bus stop ctrl

// It will be constant unless the size of bus stop is changed

this.Line.X1 = canvasWidth \* rel\_p1.X +15;

this.Line.Y1 = canvasHeight \* rel\_p1.Y + 15;

this.Line.X2 = canvasWidth \* rel\_p2.X + 15;

this.Line.Y2 = canvasHeight \* rel\_p2.Y + 15;

}

}

}

## **MapCtrl**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace TravelConn.Controls

{

/// <summary>

/// Interaction logic for Map.xaml

/// </summary>

public partial class MapCtrl : UserControl

{

public MapCtrl()

{

InitializeComponent();

}

//Start and end bus stop selections

private CBusStop start\_busstop = new CBusStop();

private CBusStop end\_busstop = new CBusStop();

// Keeps track of the last busStopCtrl that was clicked a mouse

// This is used when the context menu is brought up to identify the bus stop it belongs to

private BusStopCtrl busStopLastRClick = new BusStopCtrl();

/// List of the bus stops and links on canvas

private List<BusStopCtrl> busStopCtrls = new List<BusStopCtrl>();

private List<LinkCtrl> linkCtrls = new List<LinkCtrl>();

// specifies the scale size of the individual bus stop links

private double scale\_size = 1;

public double scale

{

get { return scale\_size; }

set

{

scale\_size = value;

}

}

public void Scale()

{

foreach (BusStopCtrl control in busStopCtrls)

{

ScaleTransform scaleTransform1 = new ScaleTransform(scale\_size, scale\_size, 15, 15);

control.RenderTransform = scaleTransform1;

}

}

//Canvas height and width

public double canvasHeight

{

get

{

return canvas.Height;

}

set

{

canvas.Height = value;

}

}

public double canvasWidth

{

get

{

return canvas.Width;

}

set

{

canvas.Width = value;

}

}

/// Textblock passed into the control so that whenevee a bus stop is highlighted the name can be passed and displayed

public TextBlock displayTxtBl;

/// Two drop downs that are referenced by the map

public ComboBox fromComboBox;

public ComboBox toComboBox;

//start of journey bus stop

public CBusStop startBusStop

{

get

{

return start\_busstop;

}

set

{

if (start\_busstop != null)

{

start\_busstop.currentType = new CBusStopType().resting();

}

start\_busstop = value;

start\_busstop.currentType = new CBusStopType().start();

}

}

// end of journey bus stop

public CBusStop endBusStop

{

get

{

return end\_busstop;

}

set

{

if (end\_busstop != null)

{

end\_busstop.currentType = new CBusStopType().resting();

}

end\_busstop = value;

end\_busstop.currentType = new CBusStopType().end();

}

}

/// adds bus stop and links to the map control but does not draw them yet.

/// it calculates the relative position of each one

public void Create(List<CBusStop> busStops, List<CLink> links)

{

BusStopCtrl busStopCtrl;

foreach (CBusStop busStop in busStops)

{

/// Create and add busStopCtrl objects to the list

busStopCtrl = new BusStopCtrl();

busStopCtrl.busStop = busStop;

busStopCtrls.Add(busStopCtrl);

//adds the event when a mouse is moved over a bus stop

busStopCtrl.MouseEnter += BusStopCtrl\_MouseEnter;

busStopCtrl.MouseLeave += BusStopCtrl\_MouseLeave;

busStopCtrl.MouseRightButtonDown += busStopCtrl\_MouseRightButtonDown;

busStopCtrl.ContextMenu = Resources["ContextMenu"] as ContextMenu;

}

LinkCtrl linkCtrl;

foreach (CLink link in links)

{

linkCtrl = new LinkCtrl();

linkCtrl.link = link;

linkCtrls.Add(linkCtrl);

}

CalcRelativePos();

}

//Calculates the relative position of the components

private void CalcRelativePos()

{

///All bus stops must fit into the map and by taking the to the most extreme

/// values of latitude and longitude the distances are scaled appropiatly

///Extreme values for lat and long

///Longitude is always from -90 to +90 degrees

double minLat = 99;

double maxLat = -99;

///Latitude is from -180 to +180 degrees

double minLong = 199;

double maxLong = -199;

///Gets the most extreme values for lat and long

foreach (BusStopCtrl busStopCtrl in busStopCtrls)

{

if (busStopCtrl.busStop.latitude > maxLat) maxLat = busStopCtrl.busStop.latitude;

if (busStopCtrl.busStop.latitude < minLat) minLat = busStopCtrl.busStop.latitude;

if (busStopCtrl.busStop.longitude > maxLong) maxLong = busStopCtrl.busStop.longitude;

if (busStopCtrl.busStop.longitude < minLong) minLong = busStopCtrl.busStop.longitude;

}

///relativepos - a point with coordinates between 0 and 1 in relation to other bus stops

Point relativePos = new Point();

///Calculating the relative point

foreach (BusStopCtrl busStopCtrl in busStopCtrls)

{

relativePos.X = (busStopCtrl.busStop.latitude - minLat) / (maxLat - minLat);

relativePos.Y = (busStopCtrl.busStop.longitude - minLong) / (maxLong - minLong);

busStopCtrl.relativePos = relativePos;

}

foreach (LinkCtrl linkCtrl in linkCtrls){

relativePos.X = (linkCtrl.link.busStopA.latitude - minLat) / (maxLat - minLat);

relativePos.Y = (linkCtrl.link.busStopA.longitude - minLong) / (maxLong - minLong);

linkCtrl.relPoint1 = relativePos;

relativePos.X = (linkCtrl.link.busStopB.latitude - minLat) / (maxLat - minLat);

relativePos.Y = (linkCtrl.link.busStopB.longitude - minLong) / (maxLong - minLong);

linkCtrl.relPoint2 = relativePos;

}

}

public void Draw(double width, double height)

{

///Height and width of the map control

canvasHeight = height;

canvasWidth = width;

///The extra amount taken from the size is to allow for the correction of the size of the bus stop control

foreach (LinkCtrl linkCtrl in linkCtrls)

{

linkCtrl.Draw(canvasWidth, canvasHeight);

canvas.Children.Add(linkCtrl);

}

foreach (BusStopCtrl busStopCtrl in busStopCtrls)

{

Canvas.SetLeft(busStopCtrl, busStopCtrl.relativePos.X \* canvasWidth);

Canvas.SetTop(busStopCtrl, busStopCtrl.relativePos.Y \* canvasHeight);

canvas.Children.Add(busStopCtrl);

}

}

///When mouse enters the bus stop control

private void BusStopCtrl\_MouseEnter(object sender, MouseEventArgs e)

{

var busStopCtrl = sender as BusStopCtrl;

displayTxtBl.Text = busStopCtrl.busStop.name;

}

///When mouse enters the bus stop control

private void BusStopCtrl\_MouseLeave(object sender, MouseEventArgs e)

{

displayTxtBl.Text = "Bus stop name";

}

// Selection from the context menu

private void startStop\_Click(object sender, EventArgs e)

{

fromComboBox.SelectedItem = busStopLastRClick.busStop.name;

UpdateMap();

}

// Selection from the context menu

private void endStop\_Click(object sender, EventArgs e)

{

toComboBox.SelectedItem = busStopLastRClick.busStop.name;

UpdateMap();

}

private void moreInfo\_Click(object sender, EventArgs e)

{

Windows.BusStopInfoWindow busStopInfoWin= new Windows.BusStopInfoWindow(busStopLastRClick.busStop);

busStopInfoWin.Show();

}

private void busStopCtrl\_MouseRightButtonDown(object sender, MouseButtonEventArgs e)

{

busStopLastRClick = sender as BusStopCtrl;

}

public void UpdateMap()

{

/// Updates the map according to the state of the bus stops

/// by looping through all of the bus stops

foreach (BusStopCtrl busStopCtrl in busStopCtrls)

{

busStopCtrl.ChangeType(busStopCtrl.busStop.currentType);

}

}

}

}

## **ResultTableCtrl**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace TravelConn.Controls

{

/// <summary>

/// Interaction logic for CResultTableCtrl.xaml

/// </summary>

public partial class ResultTableCtrl : UserControl

{

//storing if the journey should be selected on map or not

private bool is\_Selected = false;

// Section uses the System.Windows.Documents class

// The Section will be added to a printable document and/or to more information window

public Section moreInfoSection = new Section();

public bool IsSelected{

get{ return is\_Selected; }

set

{

is\_Selected = value;

if (value == true)

{

ChangetoState("highlighted");

startOfJourney.currentType = new CBusStopType().start();

endOfJourney.currentType = new CBusStopType().end();

foreach (CBusStop busStop in inJourney)

{

busStop.currentType = new CBusStopType().injourney();

}

foreach (CBusStop busStop in journeyChanges)

{

busStop.currentType = new CBusStopType().buschange();

}

}

else

{

ChangetoState("resting");

}

}

}

// journey is the result of the calculation for the quickest journey

public List<CTimetableRecord> journey;

/// These keep track of the bus stops which are in the journey.

/// There will always be a start bus stop and destination bus stop

public CBusStop startOfJourney = new CBusStop();

public CBusStop endOfJourney = new CBusStop();

/// These lists could be empty

public List<CBusStop> journeyChanges = new List<CBusStop>();

public List<CBusStop> inJourney = new List<CBusStop>();

// Default constructor

public ResultTableCtrl()

{

InitializeComponent();

}

// constructor takes in the date and the journey, result of calculation

public ResultTableCtrl(DateTime date, List<CTimetableRecord> journey)

{

InitializeComponent();

this.journey = journey;

setTextBlocks(date, journey);

}

// Sets the textblock text to appropiate values and creates a section which contains more detail.

private void setTextBlocks(DateTime date,List<CTimetableRecord> journey)

{

// Date text block to the date of the journey

string dateString = date.Date.Day + "/" + date.Date.Month + "/" + date.Date.Year;

startOfJourney = journey[0].link.busStopA;

endOfJourney = journey[journey.Count() - 1].link.busStopB;

// Arrival time and departure time

string departTime = intTimeToStr(journey[0].departTime);

string arriveTime = intTimeToStr(journey[journey.Count() - 1].arrivalTime);

// Total difference in minutes

string durationString = getMinuteDifference(departTime, arriveTime) + " min";

// Date text block to the date of the journey

dateTxbl.Text = dateString;

// The fisrt element of journey will always be the start of journey bus stop

fromTxbl.Text = startOfJourney.name;

// Destination of the journey

toTxbl.Text = endOfJourney.name;

// Arrival time and departure time

departTimeTxbl.Text = departTime;

arriveTimeTxbl.Text = arriveTime;

// Total difference in minutes

totalTimeTxbl.Text = getMinuteDifference(departTime,arriveTime) + " min";

///Setting up the middle table responsible for displaying more detail

Table detailTable = new Table();

detailTable.Columns.Add(new TableColumn());

detailTable.Columns.Add(new TableColumn());

detailTable.Columns.Add(new TableColumn());

detailTable.Columns.Add(new TableColumn());

detailTable.Columns.Add(new TableColumn());

detailTable.Columns.Add(new TableColumn());

detailTable.Columns.Add(new TableColumn());

detailTable.RowGroups.Add(new TableRowGroup());

detailTable.RowGroups[0].Rows.Add(new TableRow());

TableRow currentRow = detailTable.RowGroups[0].Rows[0];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(" "))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("From"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("To"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Departure"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Arrival"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Duration"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Bus"))));

/// Setting values to count the number of changes (see iteration bellow)

/// the first bus in the journey

CBus tempBus = journey[0].link.bus;

int tempCourse = journey[0].course;

int busChanges = 0;

CBusStop tempFrom = journey[0].link.busStopA;

CBusStop tempTo = journey[0].link.busStopB;

int tempDeparture = journey[0].departTime;

int tempArrival = journey[0].arrivalTime;

/// This iteration goes through each record in the timetable and counts number of changes.

/// It also changes the type of each bus stop as it does so and adds/sets appropiate variables;

foreach (CTimetableRecord record in journey)

{

if ((tempCourse != record.course)||(tempBus != record.link.bus)){

busChanges++;

journeyChanges.Add(record.link.busStopA);

//Add a row with more detailed information

detailTable.RowGroups[0].Rows.Add(new TableRow());

currentRow = detailTable.RowGroups[0].Rows[busChanges];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(" "))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(tempFrom.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(tempTo.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(intTimeToStr(tempDeparture)))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(intTimeToStr(tempArrival)))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(getMinuteDifference(intTimeToStr(tempDeparture), intTimeToStr(tempArrival)) + " min"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(tempBus.name))));

//Keeps track of the last course and bus

tempCourse = record.course;

tempBus = record.link.bus;

tempFrom = record.link.busStopA;

tempDeparture = record.departTime;

}

else

{

if (record.link.busStopA != startOfJourney)

inJourney.Add(record.link.busStopA);

}

tempArrival = record.arrivalTime;

tempTo = record.link.busStopB;

}

detailTable.RowGroups[0].Rows.Add(new TableRow());

currentRow = detailTable.RowGroups[0].Rows[busChanges+1];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(" "))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(tempFrom.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(tempTo.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(intTimeToStr(tempDeparture)))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(intTimeToStr(tempArrival)))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(getMinuteDifference(intTimeToStr(tempDeparture), intTimeToStr(tempArrival)) + " min"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(tempBus.name))));

// sets textblock text to the number of changes

changesTxbl.Text = busChanges.ToString();

// Header of the table with more detail

Table header = new Table();

//Add 7 columns

header.Columns.Add(new TableColumn());

header.Columns.Add(new TableColumn());

header.Columns.Add(new TableColumn());

header.Columns.Add(new TableColumn());

header.Columns.Add(new TableColumn());

header.Columns.Add(new TableColumn());

header.Columns.Add(new TableColumn());

header.RowGroups.Add(new TableRowGroup());

header.RowGroups[0].Rows.Add(new TableRow());

//Row to add text to

currentRow = header.RowGroups[0].Rows[0];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Date"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("From"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("To"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Departure"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Arrival"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Total Duration"))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run("Changes"))));

header.RowGroups[0].Rows.Add(new TableRow());

currentRow = header.RowGroups[0].Rows[1];

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(dateString))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(startOfJourney.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(endOfJourney.name))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(departTime))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(arriveTime))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(durationString))));

currentRow.Cells.Add(new TableCell(new Paragraph(new Run(busChanges.ToString()))));

header.TextAlignment = TextAlignment.Center;

header.FontFamily = new FontFamily("Arial");

moreInfoSection.Blocks.Add(header);

if (busChanges != 0)

{

detailTable.TextAlignment = TextAlignment.Center;

moreInfoSection.Blocks.Add(detailTable);

}

}

// given two times, as strings, in form of 'hhmm', it will get the minute difference

private string getMinuteDifference(string time1, string time2)

{

int hours1 = Convert.ToInt16(time1.Split(':')[0]);

int hours2 = Convert.ToInt16(time2.Split(':')[0]);

int mins1 = Convert.ToInt16(time1.Split(':')[1]);

int mins2 = Convert.ToInt16(time2.Split(':')[1]);

int difmins = (mins2 - mins1) + (hours2 - hours1) \* 60;

return difmins.ToString();

}

/// converts int form of time, hhmm, to a string in form of 'hh:mm'.

private string intTimeToStr(int intTime)

{

string stringTime = "";

string temp = intTime.ToString();

if (temp.Count() == 3)

stringTime = "0" + temp[0] + ":" + temp[1] + temp[2];

else

stringTime = temp[0]+ "" + temp[1] + ":" + temp[2] + temp[3];

return stringTime;

}

public void ChangetoState(string state)

{

switch (state)

{

case "resting":

ResultTableGrid.Style = (Style)FindResource("resting");

break;

case "highlighted":

ResultTableGrid.Style = (Style)FindResource("highlighted");

break;

}

}

//Mouse hover

private void ResultTableGrid\_MouseEnter(object sender, MouseEventArgs e)

{

//change state to highlighted only if not selected

if (!is\_Selected)

{

ChangetoState("highlighted");

}

}

private void ResultTableGrid\_MouseLeave(object sender, MouseEventArgs e)

{

if (!is\_Selected)

{

ChangetoState("resting");

}

}

private void moreBtn\_Click(object sender, RoutedEventArgs e)

{

//ChangetoState("selected");

JourneyInfoWindow infoWindow = new JourneyInfoWindow();

infoWindow.textField.Blocks.Add(moreInfoSection);

infoWindow.Show();

}

/// When the print button is clicked it will generate a section that could be displayed

private void printBtn\_Click(object sender, RoutedEventArgs e)

{

PrintDialog printDlg = new PrintDialog();

FlowDocument printDocument = new FlowDocument();

printDocument.Name = "JourneyPrintForm";

var separator = new Rectangle();

separator.Stroke = new SolidColorBrush(Colors.Blue);

separator.StrokeThickness = 3;

separator.Height = 3;

separator.Width = double.NaN;

printDocument.Blocks.Add(moreInfoSection);

IDocumentPaginatorSource idpSource = printDocument;

printDlg.PageRangeSelection = PageRangeSelection.AllPages;

printDlg.UserPageRangeEnabled = true;

// Display the Print dialog. This returns true if the user presses the Print button on the dialog.

Nullable<Boolean> print = printDlg.ShowDialog();

if (print == true)

{

printDlg.PrintDocument(idpSource.DocumentPaginator, "Bus Journey Printing");

}

}

}

}

for the tourists and visitors.

|  |  |
| --- | --- |
| TravelConn - bus journey finder | User  Manual |
| TravelConn | A solution to the quickest path problem |

# 

# TravelConn - User Manual

This is a user manual for set up and usage of the application.

Welcome to TravelConn. This program was designed to calculate the quickest bus journey between two bus stops after a given time.

## Instilalisation from a CD disk

To install TravelConn on a machine from a CD-ROM disk follow the instructions below:

1. Insert the TravelConn CD disk into the disk drive.
2. Copy and paste the files into a convient directory.
3. Run the setup.exe file and the program will install and run automatically.

Data sources  
Text file

To load bus network data stored in a text file, copy the folder with your data into your C: disk partition directory and open TravelConn. Depending on the size of the bus network and the speed of your computer, the data should be loaded between a fraction of a second up to 20 seconds.

### Database

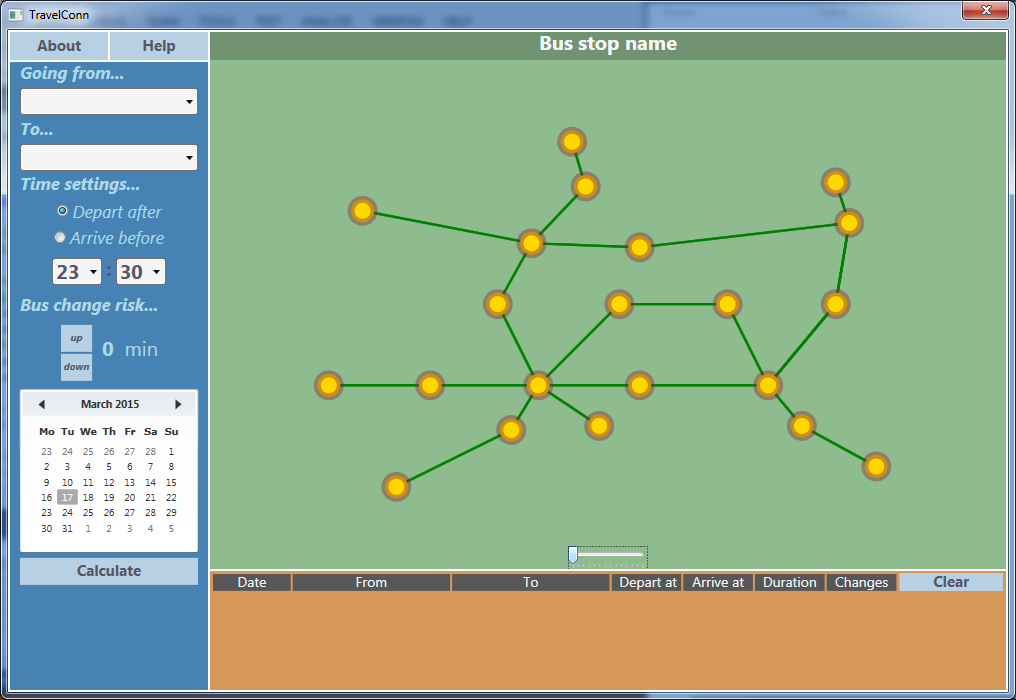
To load the data from a database contact the developer and supply the database’s name and description.

## Hardware and software configuration

Here is a list of required hardware and software configuration:

* Windows XP or above
* 1 GB of RAM
* 15 MB of free hard disk space
* Monitor capable of at least 600x800 resolution
* .NetFramework 4.5.1

## Main Window



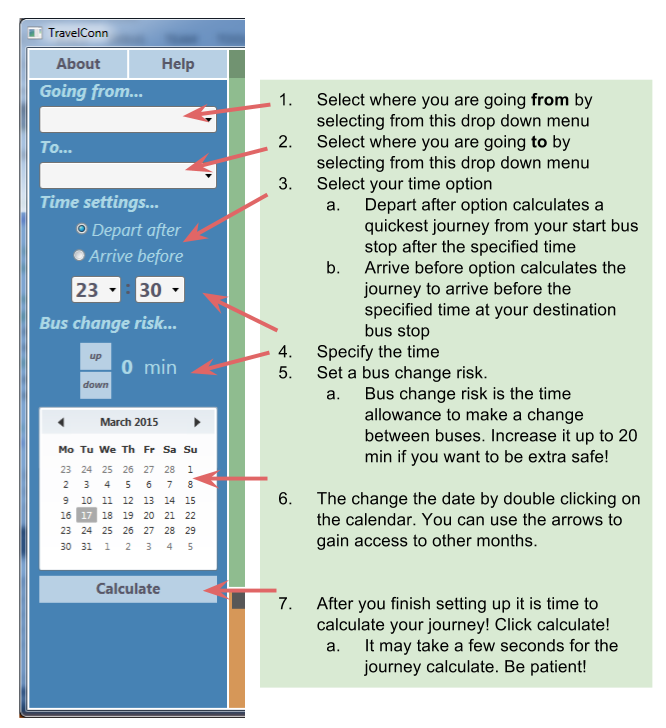
This is the Main Window from which you can navigate through the application.

## Basic Lay Out

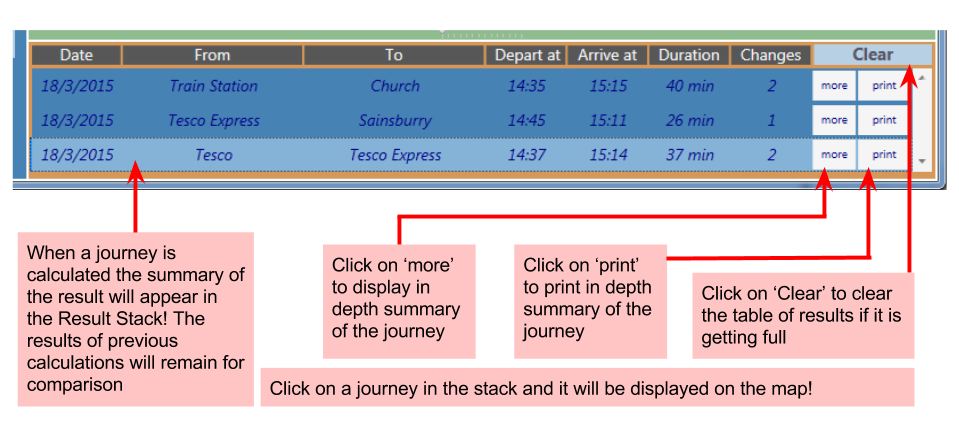
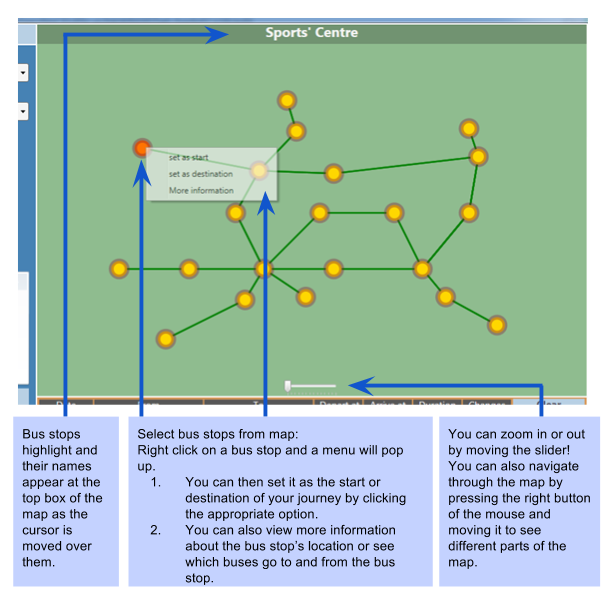
The Main Window is made up of three main parts:

### Search Panel – Selecting your journey

– You specify all of your journey details here.

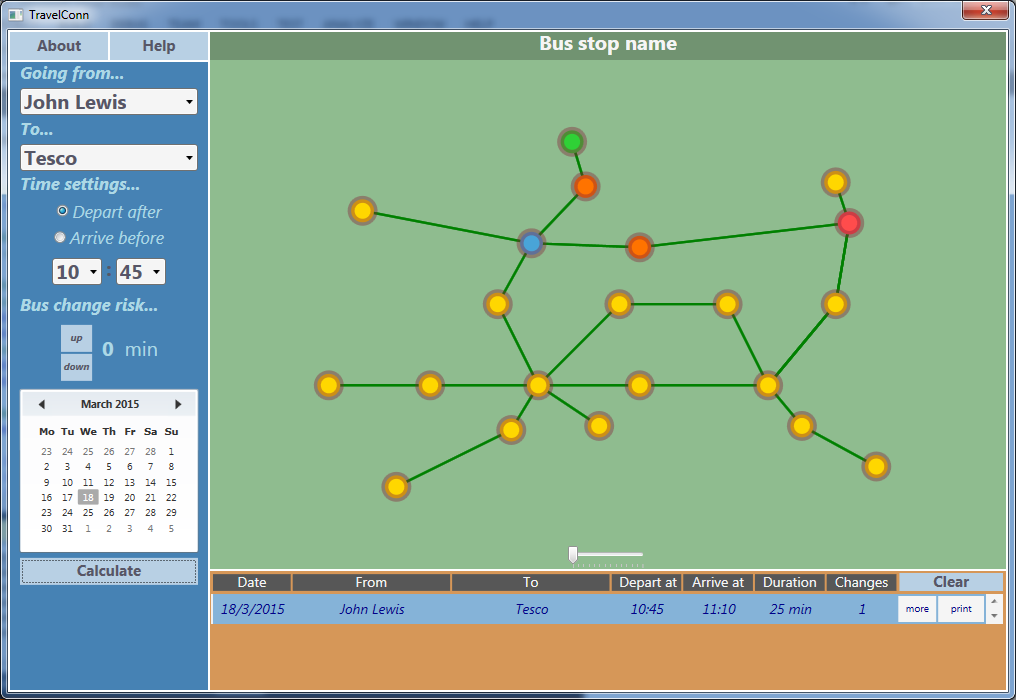


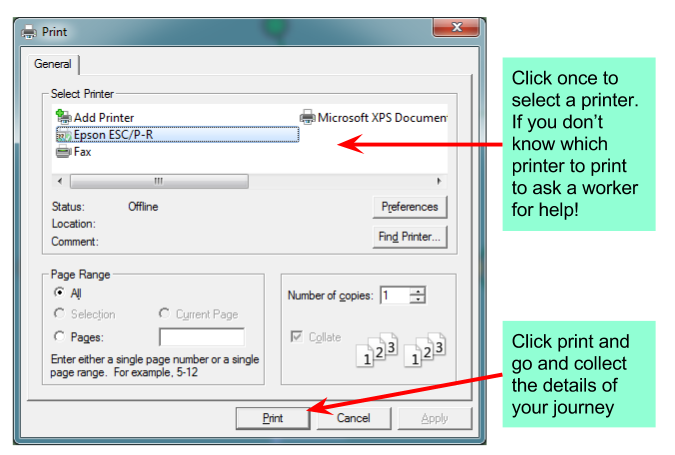
### Map

Abstract graphical representation of the bus network

Results List – Viewing the results

## Printing

You can print the result of a calculation by clicking the print button next to your bus journey result.

A print dialog will appear:

### Printing - Step by step

To print a calculated journey result:

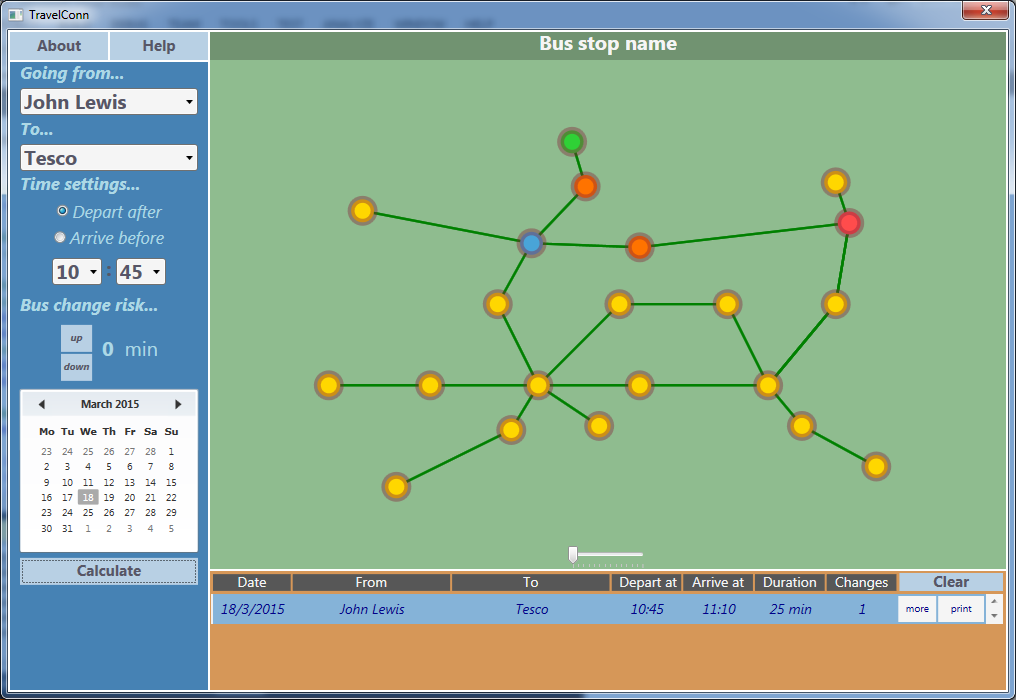
1. Right Mouse Click on the ‘print’ button located next to your journey details
2. Select a printer
3. Click ‘Print’ button
4. Collect your printed file

### Saving an electronic copy of the document

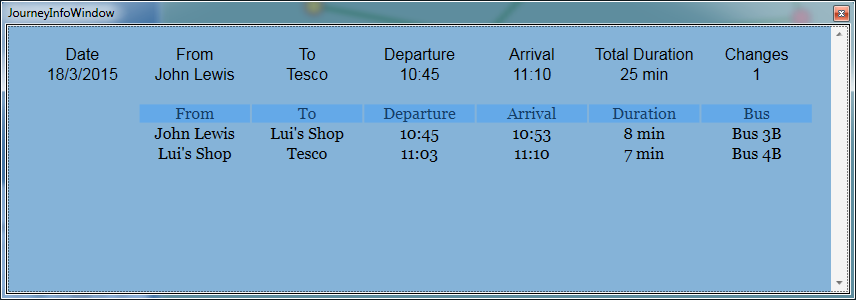
To save an electronic copy of the calculated journey:

1. Right Mouse Click on the ‘print’ button located next to your journey detail
2. Select ‘Microsoft XPS Document’
3. Click ‘Print’ button
4. Rename and save the document

## Viewing more detail

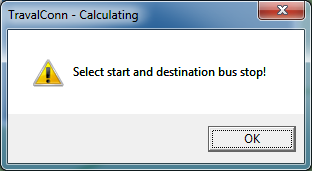
To view more detail about the journey click the ‘more’ button located next to your journey

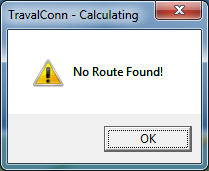
A window wil pop up showing the journey in more detail.



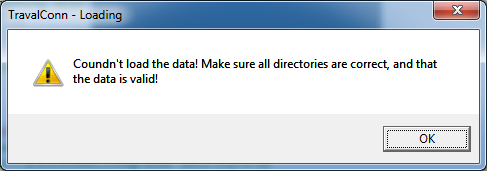
**Error Messeges**

There are several error messages that you can encounter.

You will encounter this error when if you don’t set your start and destination bus stops. When this error comes up, click OK and select your bus stops either from the drop down menus or the map.



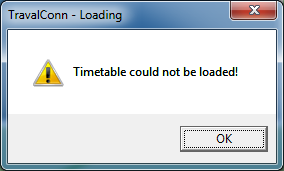
You will enconter this error when the journey between the selected stops for the specified time does not exist.



This error message might come up when TravelConn is opening and fails to load the bus network data. The two common reasons for this error are:

1. Directory or the database connection string are incorrect
2. The bus network data is has been corrupted.

To solve the problem, check if all directories are correct and try and deploy the appropriate bus network files again. If you are having trouble loading any data, contact the developer. Contact detail below.

You will encounter this error when the timetable file is broken and cannot be loaded. To solve the problem deploy the appropriate timetable file again.

# Contact Details

Developer: Mateusz Ochal, Hillsroad Sixth Form College Student

Email: [mateusz.ochal8@gmail.com](mailto:mateusz.ochal8@gmail.com)

Phone: ++447445484212

# Appraisal

## Meeting the objectives

#### Specific Objectives:

|  |  |  |  |
| --- | --- | --- | --- |
| ***No.*** | ***Objective*** | ***Met?*** | ***Comment*** |
| 1 | The program should accept a start and destination bus stop and calculate the shortest journey between them. | Yes | Selecting from dropdowns and the map |
| 2 | The calculations should take into account different bus timetables for different days. | Yes | The program loads appropriate timetables when needed. |
| 3 | There should be two algorithm options the user can select from: ‘arrive before’ and ‘depart after’. | Yes | There are two algorithm options and both of them work. |
| 4 | The user should be able to choose a date from a calendar. | Yes | There is a calendar present |
| 5 | The program should display the shortest bus journey. | Almost | In special case when the bus change risk is 0 it will find the quickest bus journey. |
| 6 | The program should keep track of the previous result of calculated bus journeys showing the time of start of the journey and the time of arrival at the destination, and number of bus changes, total time of journey. | Yes | There is a results stack which keeps track of the previous results. The details are laid out in a grid |
| 7 | Upon selecting a route more details should be shown including: the time and place of the bus changes, the bus name of the buses, time of the journey on each bus and waiting time between bus changes and the total waiting time. | Yes | A window pops up when right mouse clicking the ‘more’ button. |
| 8 | The program should produce a document that could be printed or stored if desired. The information should include: the time and place of the bus changes, the bus name of the buses, time of the journey on each bus and waiting time. | Yes | The solution can be printed – although something goes wrong with formatting and the printed result is not as clear as the electronic version. |
| 9 | When selected, the route will be displayed on a map, showing the bus journey including all bus stops and bus changes involved. | Yes | The map is very responsive a draws the route on map within a fraction of a second even with 2600 bus stops on map |
| 10 | The map should have basic zoom in and out feature and span. | Yes | Objective met – however as the number of bus stops increases the map becomes less responsive and jumps while trying to keep up with the cursor. This would cause a bigger problem with larger networks. |
| 11 | The user should interact with the map and be able to choose start and destination bus stop. | Yes | The bus stops can be selected from the map |
| 12 | The system should be capable of handling about 8Mbytes of data at a time and the bus journeys should be displayed within 2 seconds. | Yes | The calculations are almost instantaneous with small networks counting 40-100 bus stops. It takes about one second to calculate the most extreme route for a grid of 2600 bus stops |

### General Objectives:

|  |  |  |  |
| --- | --- | --- | --- |
| ***No.*** | ***Objective*** | ***Met?*** | ***Comment*** |
| 1 | The program should be intuitive and easy to use for non-experienced tourists to learn how to use it quickly. | Yes | As mentioned before – the program is responsive but lags with greater numbers of bus stops. The program was called ‘intuitive and easy to use’ by my client and people the application was tested with. (see below) |

### Extension Objectives:

|  |  |  |  |
| --- | --- | --- | --- |
| ***No.*** | ***Objective*** | ***Met?*** | ***Comment*** |
| 1 | When clicked on a bus stop on the map the following information could be displayed: the buses going through the bus stop, geographical location. | Yes | It is possible to access additional data concerning the bus stop, by right clicking the menu. |

## Feedback from the client, Joe Johnson

I showed my client Joe Johnson how to set up and use the application on 15th March. I received his feedback by email on 17th March. Here is a copy of the content:

* General feedback about how easy the system is to use:
  + User (tourist): The system is intuitive and easy to use without learning. User interface was properly designed. Most important options can be accessed immediately from the main window. All other options are easy to find (e.g. on right click submenu). All information is presented clearly and unambiguously. UI components are logically grouped together and therefore they are easy to understand. Font styles and application colours make a nice aesthetic impression for the end user.
* How does the system meet the objectives?
  + The system met all required objectives (see table below)
* How easy was the system to set up?
  + The simple deployment scenario is easy as it requires only copying of executable file and bus schedule data file to the target machine. To make it even simpler the developer provided a msi installer that allows to install the application using wizard or to install it in an automatic way without user interaction. In more complex deployment scenarios (with central database) the system requires an upfront database deployment on a server (according to the database instruction). However the deployment of the application itself on the target machines is as easy as it was in the simple scenario.
* Criticisms of what was not so good about the system
  + The results of beta tests with real users showed that they had a very positive experience with the application. However some of them make comments regarding the following issues:
    - The application did not display a total price for the ticket(s)
    - The application calculated the shortest route, however some would prefer to travel longer but with smaller number of bus changes (if possible)
    - The application allowed printing individual journeys, but it did not have an option to print all chosen bus journeys on single sheet of paper.
* Suggestions about how the system could to be improved or extended
  + Create other types of the application clients such as web or mobile version of the application.
  + Extend the list of available algorithms to calculate optimal journey based on different criteria (e.g. as minimal number bus changes)
  + Provided additional information such as ticket prices.
  + Language translation of the program

General comment:

Overall, I think the system meets the expected requirements, builds positive user experience and meets the quality acceptance criteria. Therefore I would recommend starting a deployment process in the company departments.

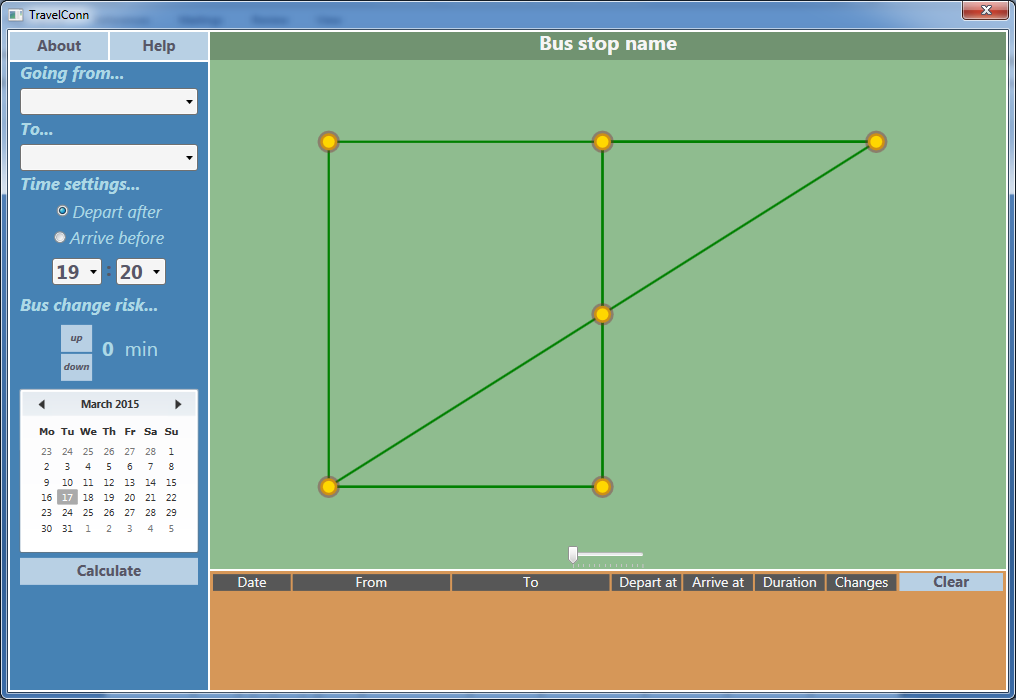
Joe Johnson 2015-03-17

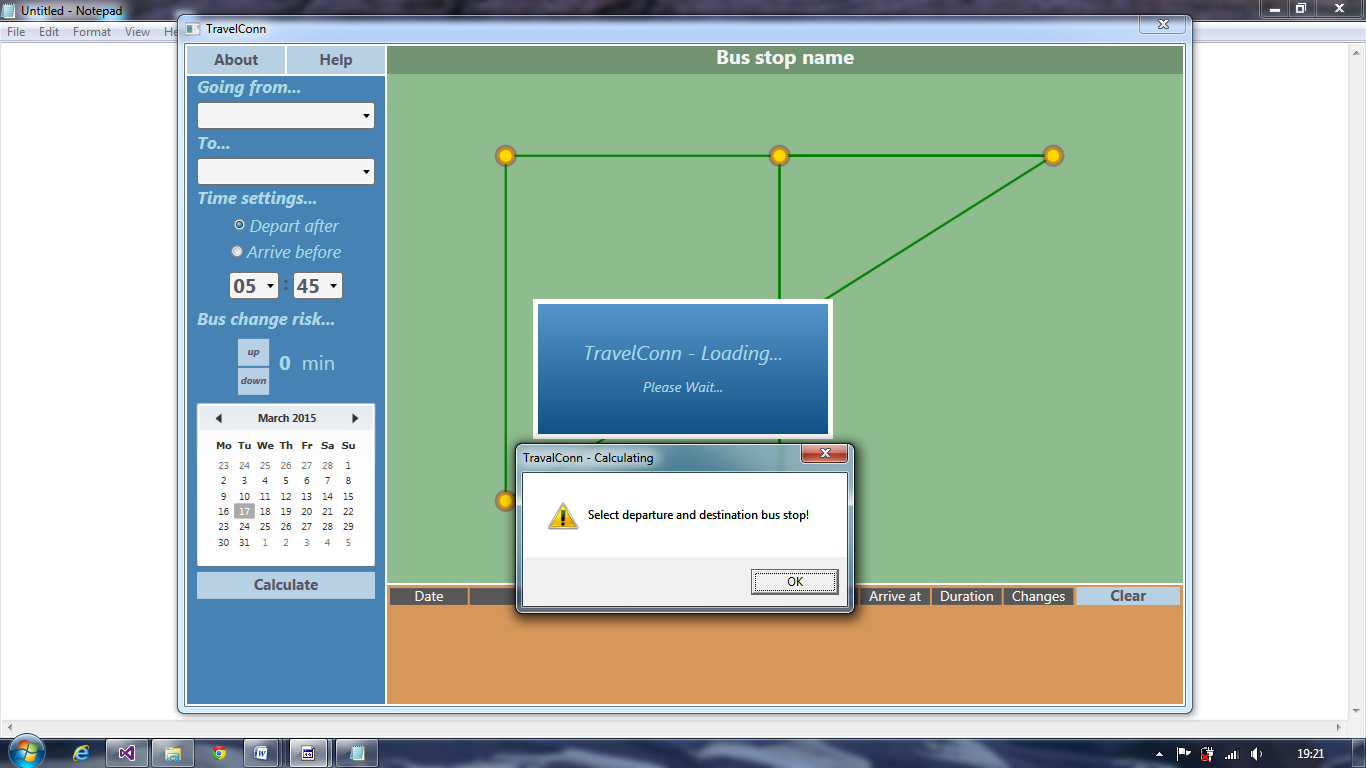
## Response to feedback

The feedback was really positive and seemed to meet Mr Johnson’s expectations. There was some criticism from the client and a clear indication for room for improvement: many some suggestions included:

* *“Create other types of the application clients such as web or mobile version of the application”*
  + A web or smart phone solutions would have perhaps been more helpful as it could be accessed from anywhere
  + To develop a web solution I would have to learn PHP or XTML5 and learn lots of other things along the way. A great idea to extend my program.
* *“Extend the list of available algorithms to calculate optimal journey based on different criteria (e.g. as minimal number bus changes)”*
  + This is a harder problem to address. After doing some research, I have found that a complete solution to this program has complexity of O(Xn) – the number of operations grows exponentially. It is a very hard problem to solve.
  + There could be a potential change I could make in my algorithm that could minimise the number of changes slightly.
    - (Depart After Algorithm – Comparison Section) When the arrival times are the same, pick the one with the smallest number of bus changes.
    - This solution would require the implementation of another variable in addition to the bestLink. The number of bus changes would count how many bus changes have been made up to a particular bus stop
* *“Provided additional information such as ticket prices. “*
  + This suggestion improvement would require getting inside information about ticket prices which might be private. Another column to the ResultTableCtrl would be added called price. The price would be calculated after the journey is calculated.
* *“Language translation of the program”*
  + This improvement would be really useful for foreign tourists/visitors. I would need to hire some translators to translate the text.
    - To implement this idea I would create a text file for each language translation of the program. The program would then look up the language translation and change the texts of text blocks.

# Screenshots

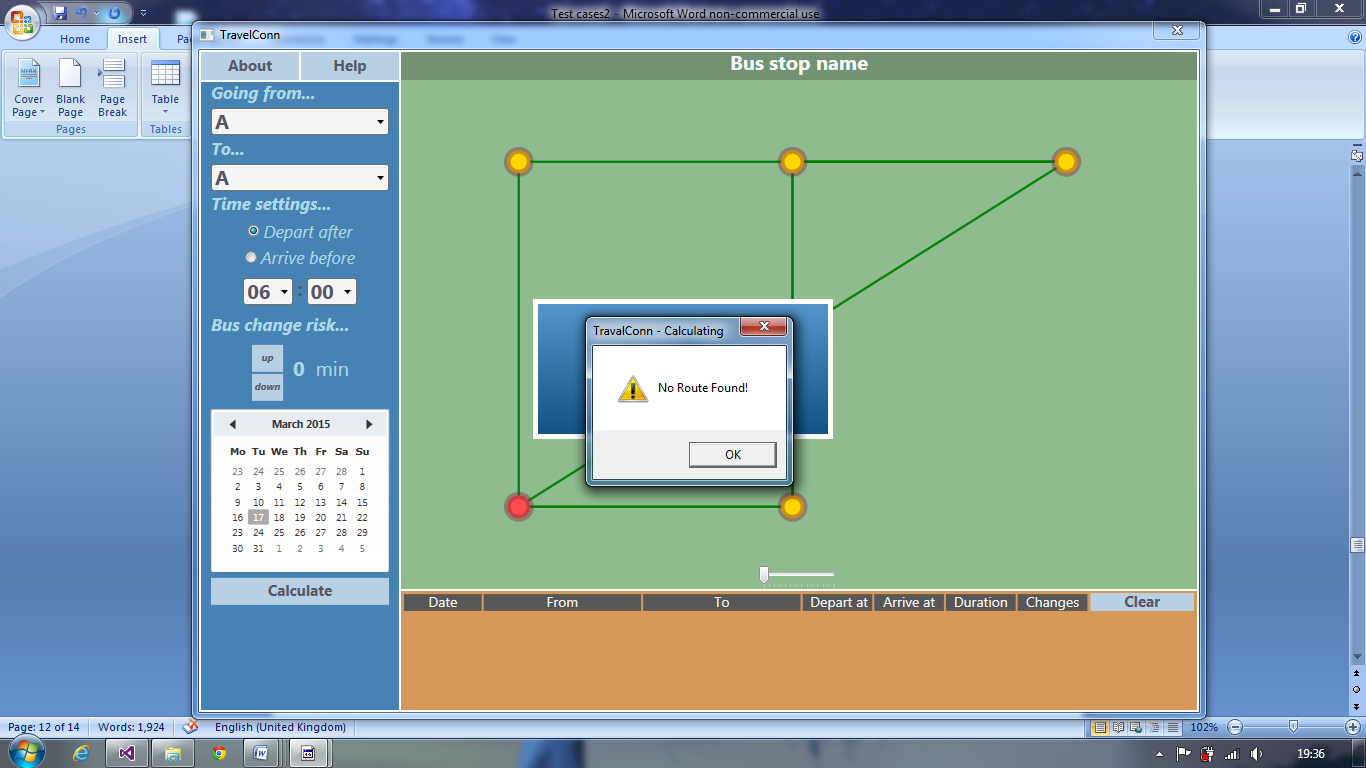


Just loaded

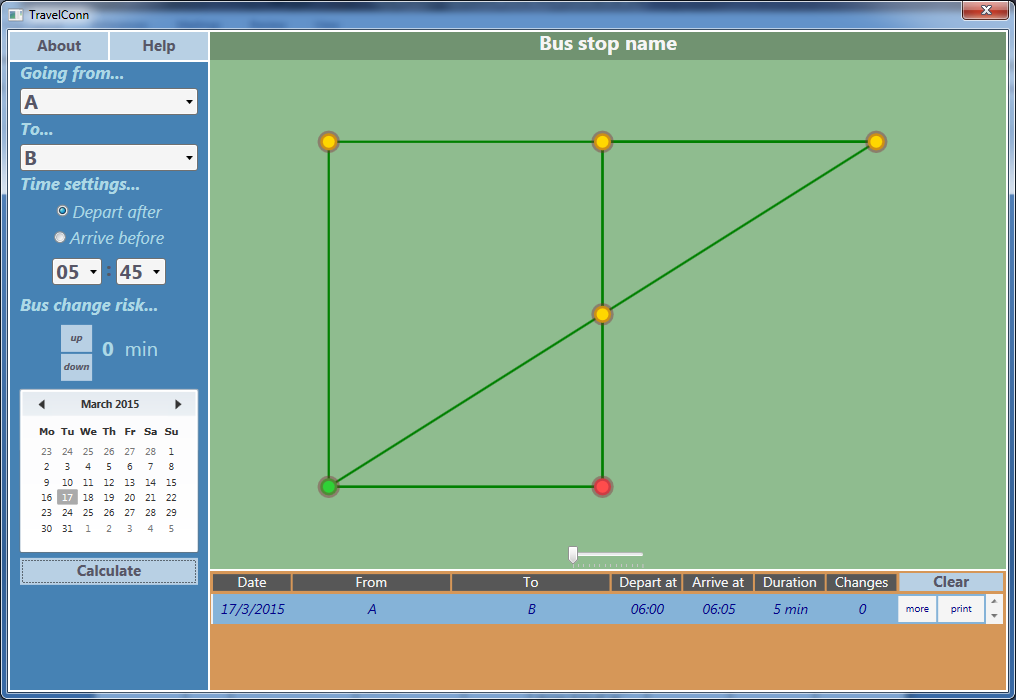
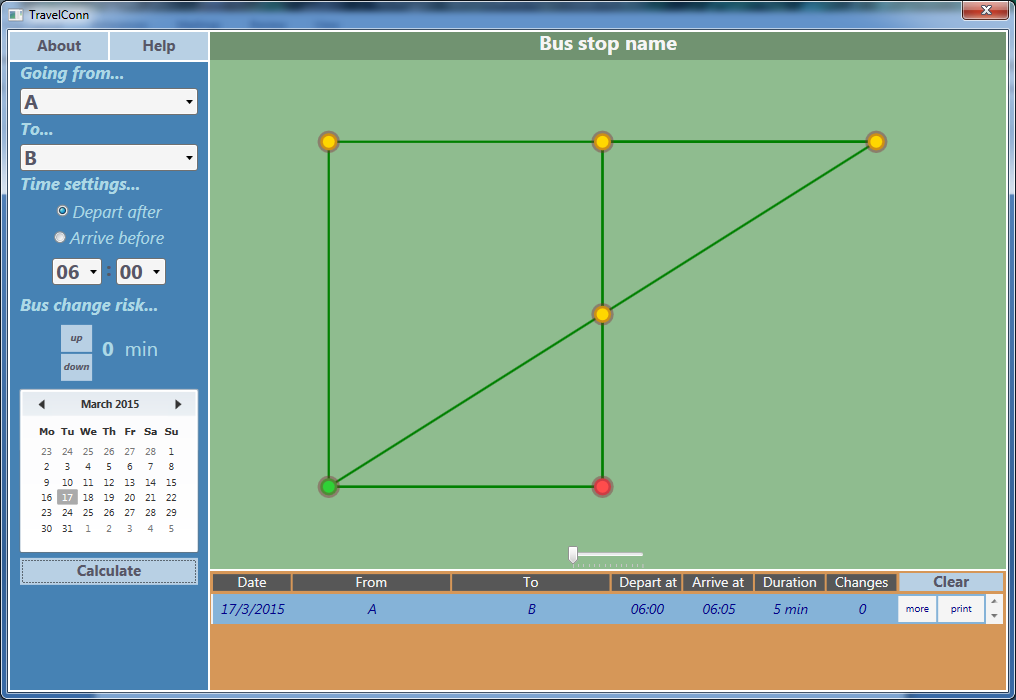
Bus stops not selected

Dropdowns

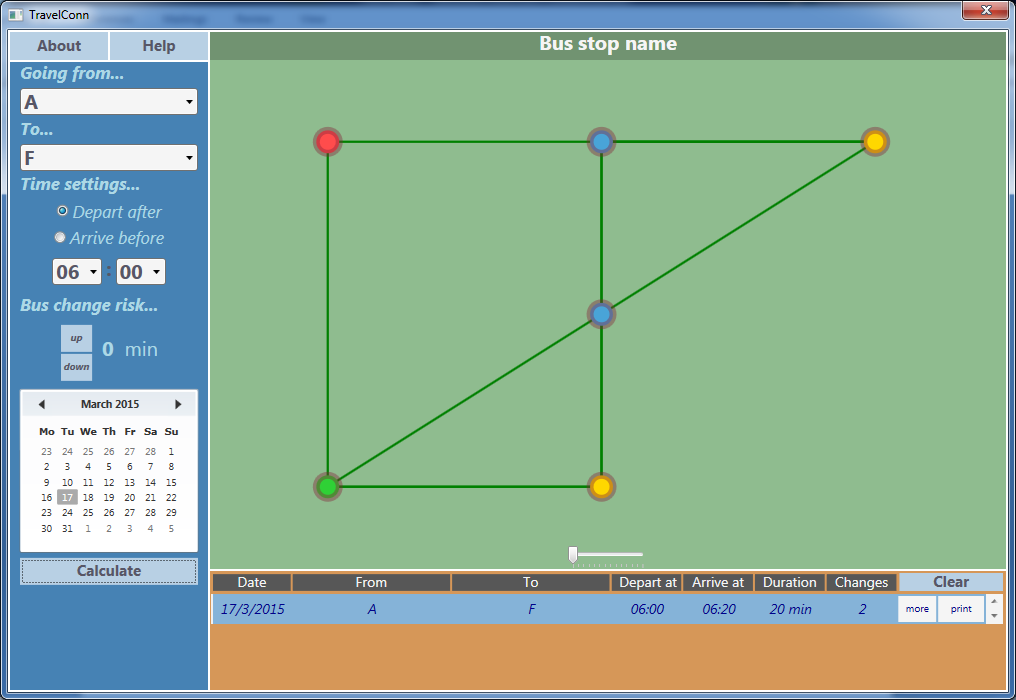
Latitude is flat! So no connections draw because relative points are 0.



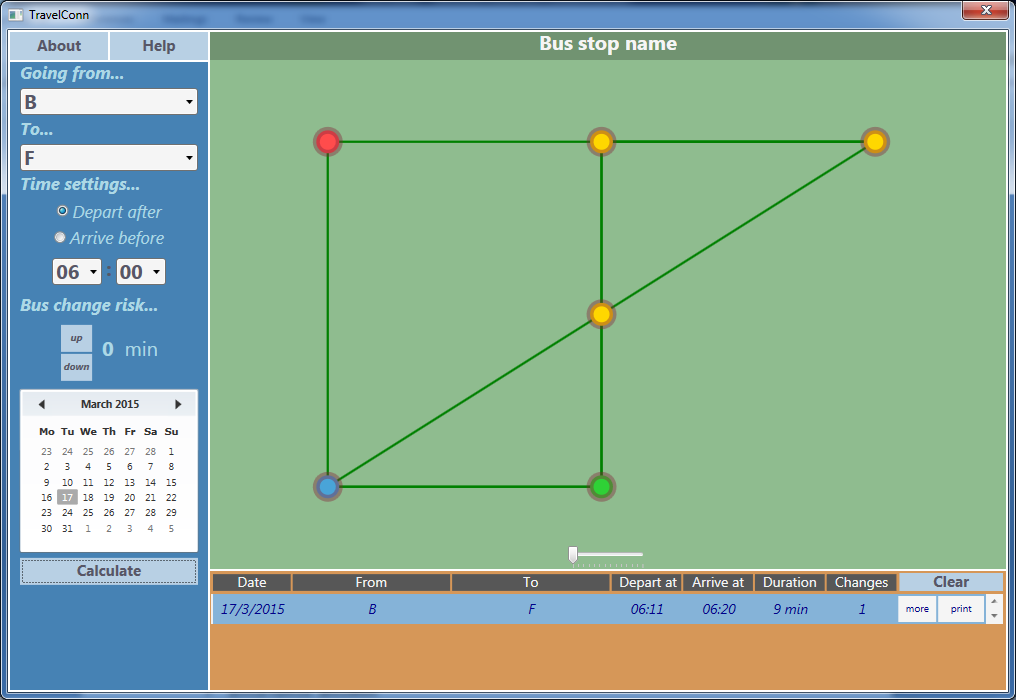
Destination bus stop and start bus stop are the same.

6.16.2

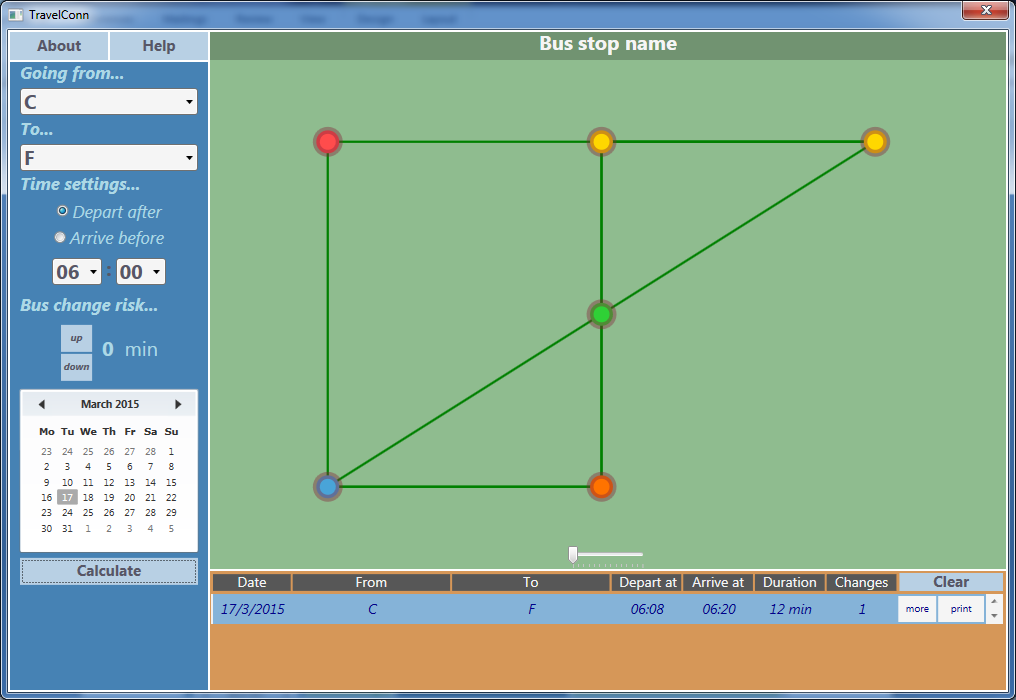
6.3



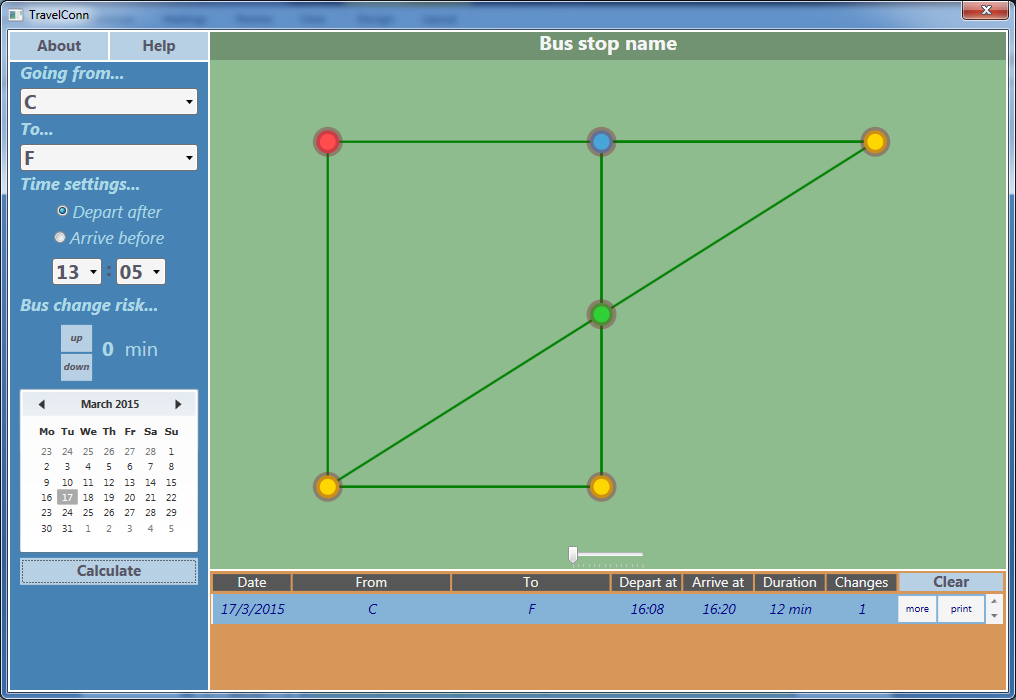
6.4



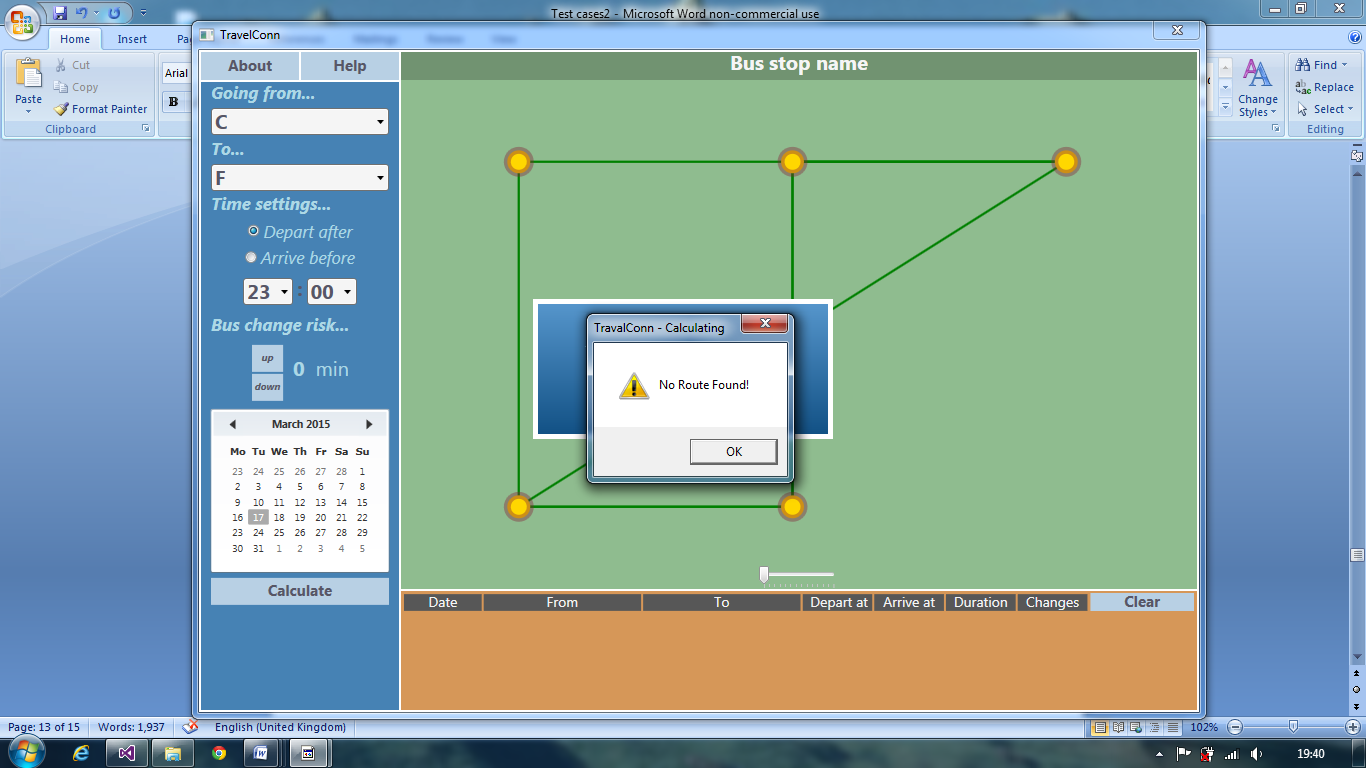
6.5

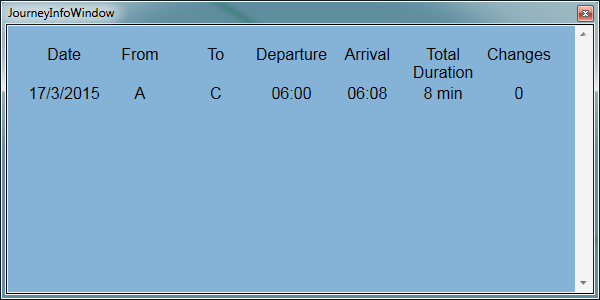


6.6

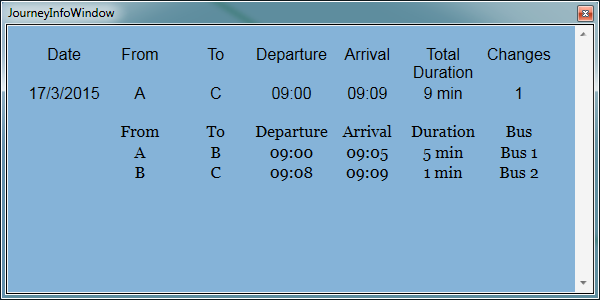
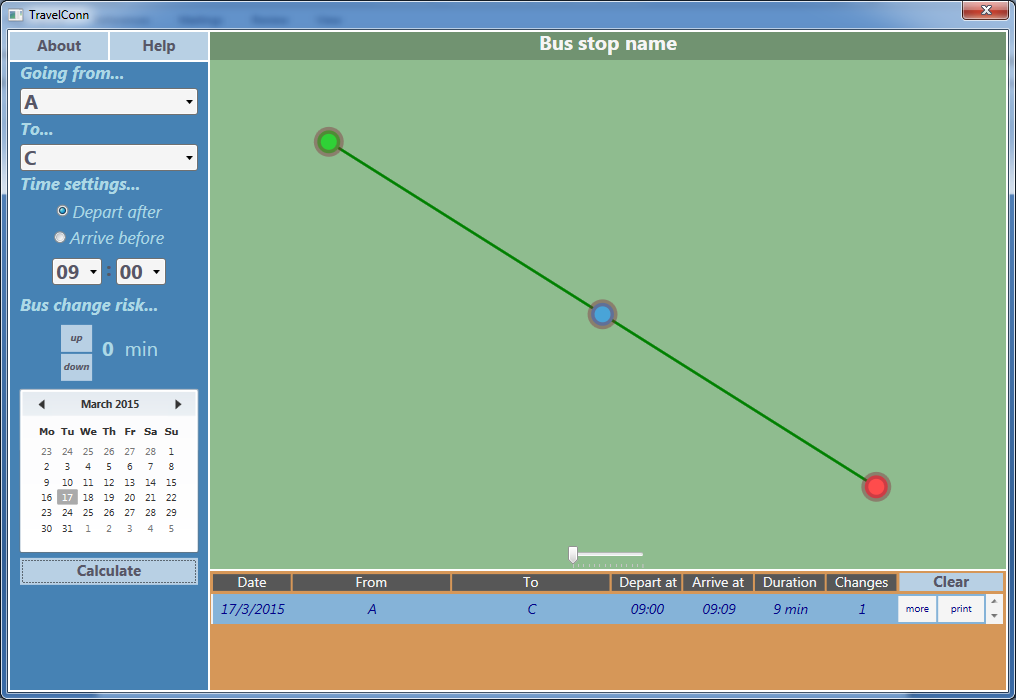


6.7

6.8



6.9

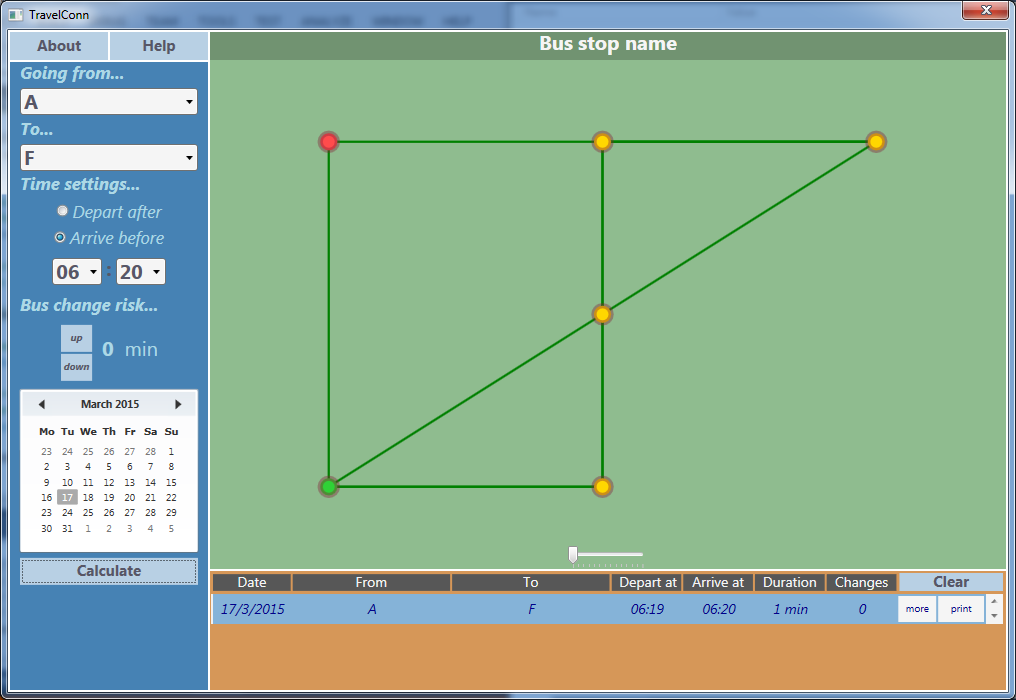


6.10

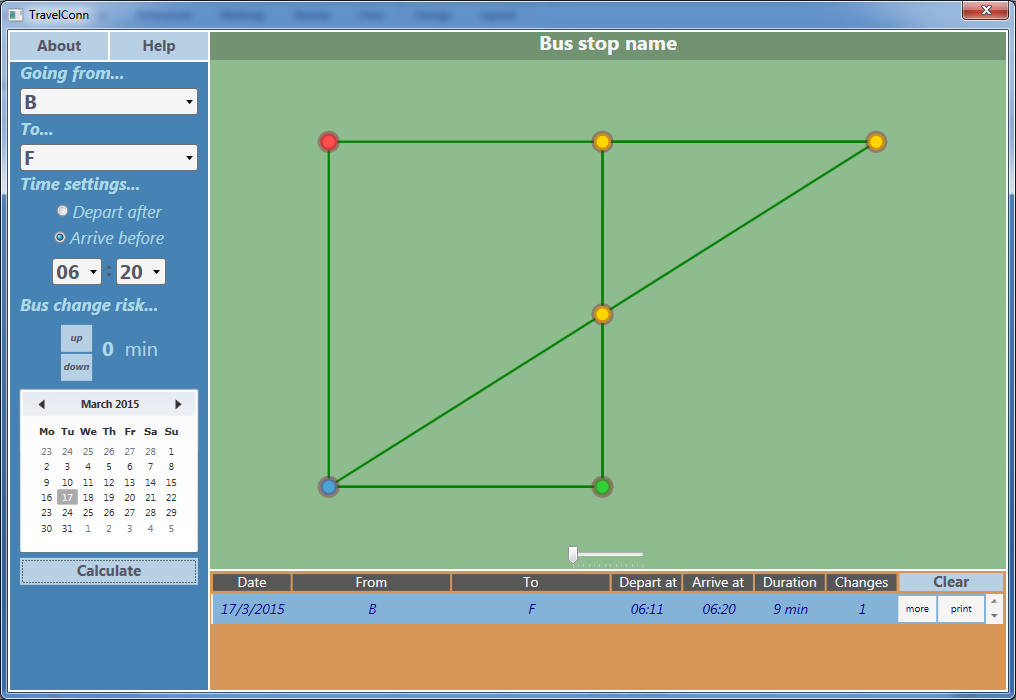




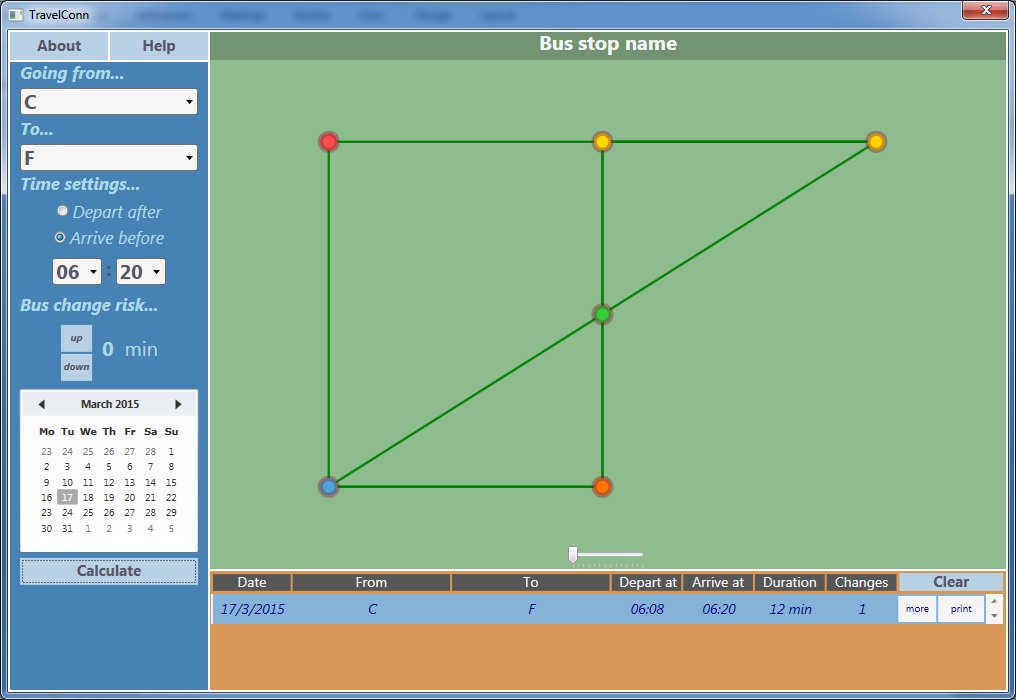
6.11



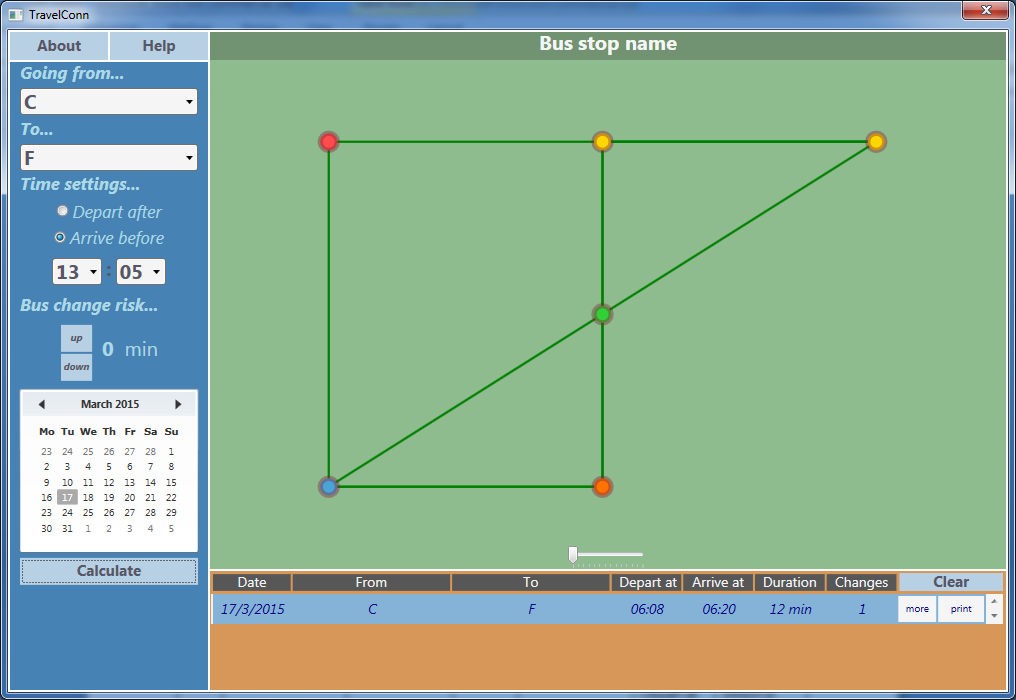
7.4



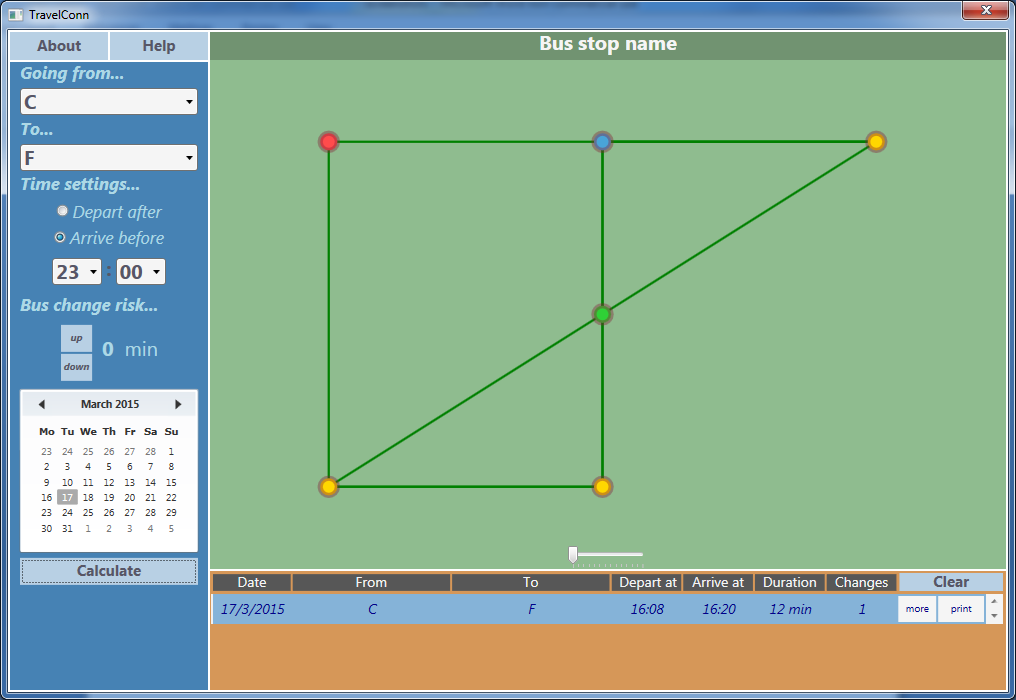
7.5



7.6



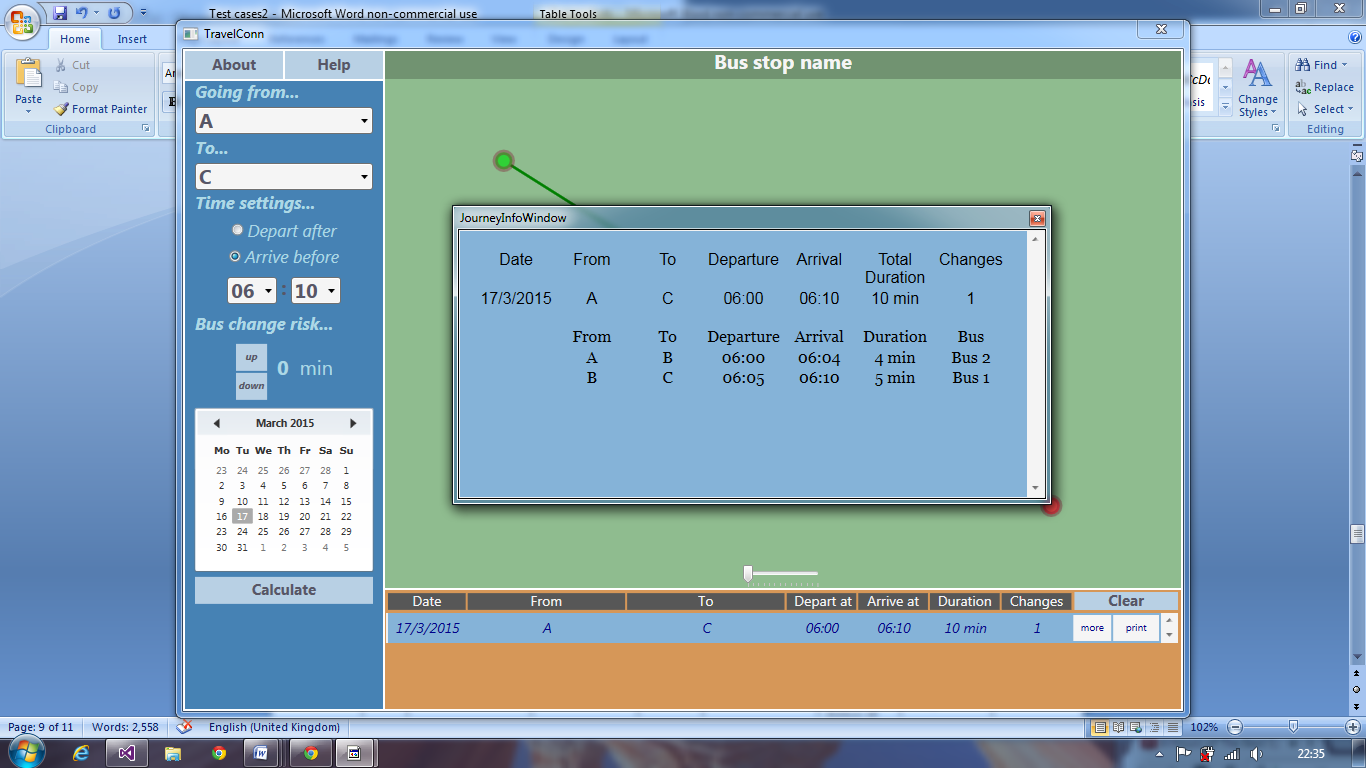
7.7



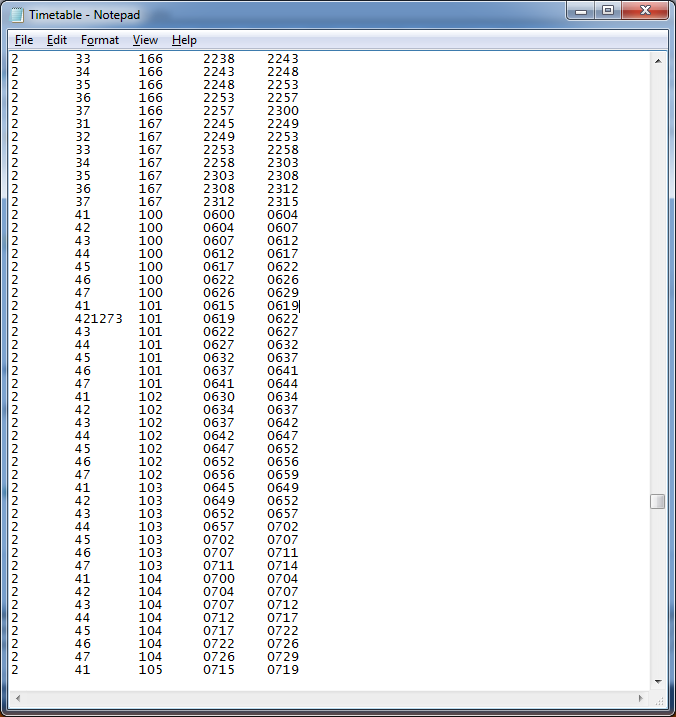
7.8



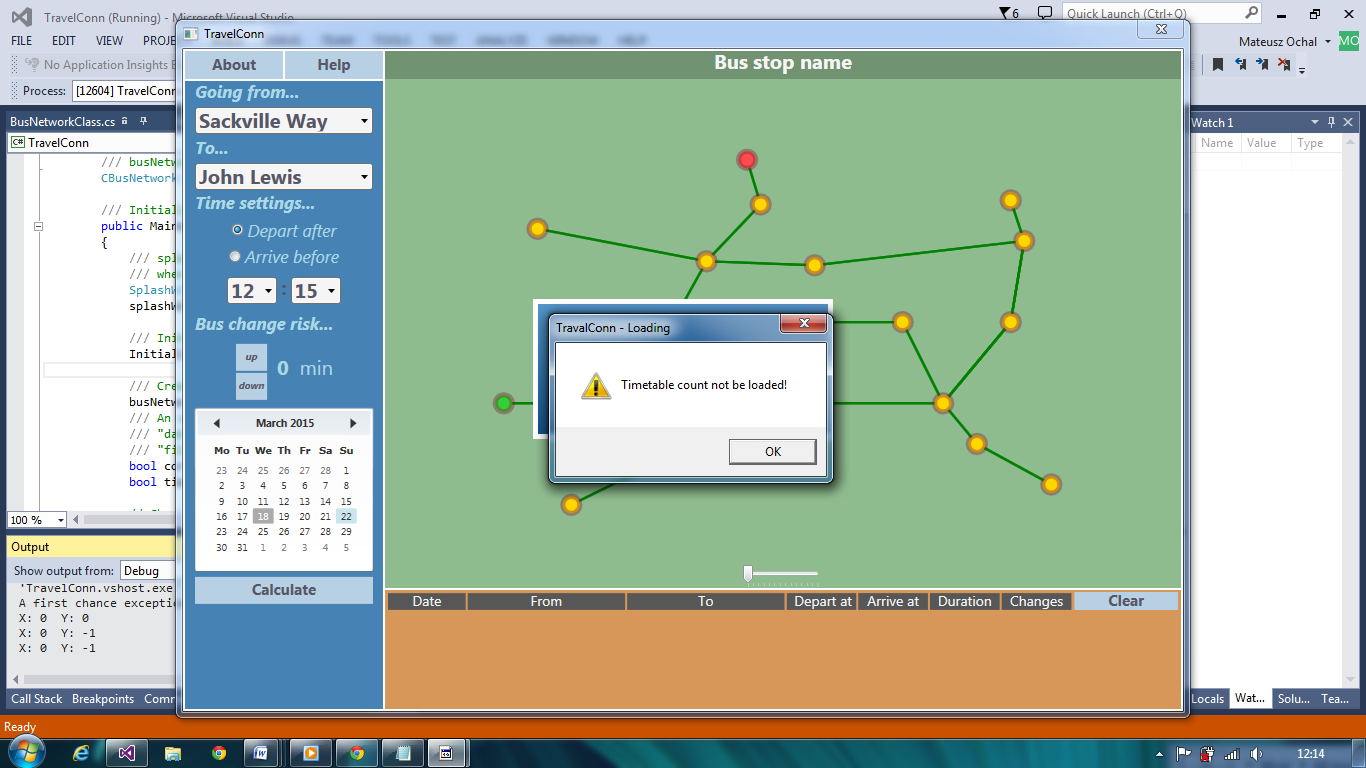
7.9

7.10

7.11



Timetable in a file screen shot



Different