
Krak-vejkort

Visualiseringen

af

Jacob Stenum Czepluch (jstc@itu.dk),
Niels Liljedahl Christensen (nlch@itu.dk),
Mikkel Larsen (milar@itu.dk),
Sigurt Bladt Dinesen (sidi@itu.dk)

IT-University
Copenhagen
First year project
Rasmus Pagh
April 2, 2012

Contents

1	Introduction	2
2	Design choices	3
2.1	Outline	3
3	Implementation	4
3.1	Controller package	4
3.2	Global package	4
3.3	Model package	4
3.3.1	Model class	4
3.3.2	XMLReader class	4
3.3.3	QuadTreeDS class	5
3.3.4	FormatConverter class	5
3.3.5	KraxToXMLConverter class	5
3.4	View package	5
3.4.1	View class	6
3.4.2	MapPanel class	6
3.4.3	ZoomHandler class	6
3.4.4	DragHandler class	6
3.5	Outline	6
4	Discussion	7
4.1	Outline	7
5	Conclusion	8
5.1	Test 2	8

1 Introduction

This will be our introduction to this small report...

This is a section. Use it. Love it. This is an example of a long text This is an example of a long text This is an example of a long text This is an example of a long text This is an example of a long text This is an example of a long text

Then we have a new line with indent This is an example of a long text This is an example of a long text This is an example of a long text This is an example of a long text

2 Design choices

Here we should write something about our design choices.

2.1 Outline

MVC

Great!

Data structure

QuadTree

Visualisation

Platform

How everything is drawn

How the user interacts

Types of the Krax-data

3 Implementation

The implementation of the application consists of four different packages; the Model, View and Controller packages used as in the MVC design pattern, and a Global package storing global fields to be accessed and modified from all other packages.

3.1 Controller package

The Controller package consists solely of the Controller class, which is both the main class (it has the main method run when the application starts), and it is the link between the Model and View packages handling the flow of data between the two. When a change is made by the user, the View calls a method in the Controller once again updating the graphical user interface according to both the input from the user and the data stored in the Model.

3.2 Global package

This package contains only the MinAndMaxValues class which has fields that need to be accessed from the entire application. These fields include initial values such as the current "viewbox", min and max values for x- and y-coordinates, limits for when the different types of road segments are drawn etc. It also contains methods for checking whether or not the current viewbox results in a need for re-filtering the data to be drawn. The class is statically imported by all classes needing to access this information.

3.3 Model package

The Model package consists of all the classes managing data storage, filtering, and conversion.

3.3.1 Model class

The Model class is the front-end class of the Model package (the only class which is directly connected to the Controller class). This is where the data structure is stored in a field and where the methods for filtering and converting data are called. The data structure is stored with the type `DataStructure`, which is an interface allowing us to easily switch between data structures, as long as they implement this interface.

3.3.2 XMLReader class

This class reads in data from an XML file of the KRAX format and converts it to instances of the Edge class (a simple class representing an edge on the roadmap), which then are added to a given data structure.

The XMLReader makes use of an external library, xom (www.xom.nu), for reading the XML data.

3.3.3 QuadTreeDS class

The QuadTreeDS is the basis of the entire application. It is an instance of this all data is stored after being read by the XMLReader class. In order for it to be used as our data structure, it implements the DataStructure interface.

The class consists of four instances of the QuadTree class (one for each type of road segment). A QuadTree consists of nodes, which has an x- and a y-coordinate (stored as doubles) and a reference to an Edge object. Each QuadTree contains all edges of a given type. Each Edge object is stored twice; both referenced to by the start- and end-coordinates of the edge.

Inserting a node into a QuadTree is done recursively; the given node is compared to root node, deciding to which of the four children of the root the given node is to be compared to next. This continues until a null-reference / a leaf is found.

Retrieving information is done using an instance of the Interval2D class (representing a rectangle), which again consists of two instances of the Interval class (representing a line). This too is done recursively; it is checked whether the coordinates of the root node is within the given rectangle. If it is, it is added to a given collection of edges. It is then checked which of the subtrees might contain nodes within the rectangle, and for each that match, the same method is invoked, now with each of the matching children as the root. The call returns at null references.

Our implementation of the Quadtree (including the Interval and Interval2D classes) are heavily based upon implementations from `algs4.cs.princeton.edu`.

3.3.4 FormatConverter class

Converts from the data type pulled out of the data structure to the data type needed by the View package.

3.3.5 KraxToXMLConverter class

Not directly part of the implementation, but the class used for converting the data from Krak to the krax XML format.

3.4 View package

Consists of all the classes handling the graphical user interface.

3.4.1 View class

Front end class in the MVC pattern. Contains the rest of the (non-static) GUI classes. Implements the MapListener interface. The overall class structure of the View package.

3.4.2 MapPanel class

Draws lines according to input data.

3.4.3 ZoomHandler class

Handles all the zooming.

3.4.4 DragHandler class

Handles all the dragging.

3.5 Outline

Model

- The Edge data type

- XMLReader

- QuadTree

- FormatConverter

- /KraxToXMLConverter

View

- The overall class structure

- The use of the Observer design pattern

- Mouse and window listeners

Controller

- The method calls from and to the map / data structure

4 Discussion

In this sections we will discuss what could have been done better, and/or what we think we have done right.

4.1 Outline

Limitations

Possible improvements

Data structure discussion

- QuadTree vs KD-tree

- Balanced vs unbalanced QuadTree

5 Conclusion

This section will make a quick summary and conclusion on our project so far.

5.1 Test 2

This is a subsection