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

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 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 the 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 needs 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 are 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 and 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 in 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 `double`) 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 to 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.

Out 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

The `FormatConverter` has static methods only, and only one public method. This methods takes an `ArrayList<Edge>` and converts it to the type `int[][][]` (`int[type][number of edges][edge coordinates]`).

The FormatConverter uses an instance of the Coordinates class, which converts the given UTM32 coordinates to pixels as shown in the GUI.

3.3.5 KrakToXMLConverter class

The KrakToXMLConverter is not directly a part of the application (it is not used runtime). It is a util class, reading in the data supplied by Krak, writing it to an XML file of the KRAX format (once again using the external `xom` library).

3.4 View package

The `View` package contains all the classes managing the graphical user interface.

3.4.1 View class

The `View` class is the front end class of the `View` package in the MVC design pattern. It contains an instance of the `MainFrame` class, which is the basic `java.swing` GUI (the window to be displayed), which then again contains an instance of our custom panel, `MapPanel`.

The `View` itself implements the interface `ViewListener`, of which an instance is stored in both the `MainFrame` and the `MapPanel` classes. This allows for these class to invoke a method in the `View`, telling it that changes has been made, which then invokes a similar method in the `Controller`, which then updates the GUI according the the changes.

3.4.2 MapPanel class

The `MapPanel` class extends the `java.swing.JPanel` class and cuntions as a panel with added functionality and with an overridden `paint` method.

The `MapPanel` stored an `int[][][]` (as generated by the `FormatConverter`), from which it it draws lines of the canvas, each corresponding to an edge stored in the `Model`.

The `MapPanel` also has to listeners from the `java.awt` library; a MouseWheelListener, which invokes a static method of the `ZoomHandler` class when the user scrolls on the panel, sending info about the mouse coordinates and the amount scrolled, and a MouseMotionListener, which invokes a static method of the `DragHandler` class when the user drags the mouse on the panel, sending info about how far the mouse has been dragged (and along which axes).

3.4.3 ZoomHandler class

The `ZoomHandler` handles all the zooming. When receiving a method call (from the `MapPanel`) saying that the user tries to zoom out, there are to possible cases; if the current viewbox is not close to the max of the width and height values, it zooms out, keeping the current center of the viewbox the center. Else, zooming is done so the viewbox follows the borders created by the max values.

Zooming in is followed by a method call on the `DragHandler` class, moving the viewbox towards the current location of the cursor.

Zooming is simply changing the global values of what is shown (in the `MinAndMaxValues` class) / changing the size of the viewbox and then invoking a repaint of the `MapPanel`.

3.4.4 DragHandler class

The `DragHandler` class, like the `ZoomHandler` class, changes the viewbox according to input data (drag amount and direction) and according the the max values / borders of how far left / right / up / down the viewbox can go.

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 out project so far.

5.1 Test 2

This is a subsection