Map of Denmark

First-Year Project,
Bachelor in Software Development,
IT Univ. of Copenhagen

Group 12
Emil Juul Jacobsen, ejuu@itu.dk (08/04-1988)
Jens Dahl Møllerhøj, jdmo@itu.dk (24/05-1990)
Jakob Melnyk, jmel@itu.dk (18/02-1991)
Niklas Hansen, nikl@itu.dk (18/12-1991)

Supervisor: Lars Birkedal
Advisors: Jonas B. Jensen & Filip Sieczkowski

May 25th, 2011

Contents

Chapter 1

Preface

This report is the result of a project at the bachelor in Software Development on the IT University of Copenhagen spanning from early March 2011 to late May 2011. The project was given as a first-semester project in groups of 4 to 5 randomly picked students. Two and a half months of the semester was set aside for this project. The time not spent on the report was spent on planning, coding, thorough testing and bugfixing of the actual program.

Chapter 2

Background

2.1 Problem area

Over the last decade people have switched from traditional, printed roadmaps to using maps on the internet. The online services remove all the problems of determining the quickest route between two points and you spend no time flipping the pages of the map trying to find what you need. This is a change without any significant negative side-effects. With the smartphones on the market today, the online maps are even more useful, because you no longer need to prepare your trip before you leave.

The online maps have now been used for many years and have not been slow at adopting new features to improve their usability. Many has implemented satellite maps, that allow us to see the entire planet from above, and lately the feature called Google Street View has upped the stakes by allowing us to look in any direction from a given point on a road. The two maps that we use the most are Google Maps and the Danish map called Krak. These maps both have the mentioned features but slight differences in the way the user navigates and searches for routes.

Because of the widespread knowledge of the online maps, the users have grown accustomed to certain features and ways of using the map. It is very important that we, with a new map program, use this knowledge to our advantage and do not try to reinvent the wheel. By using some of the commonly used controls in our map, a user will be able to quickly adapt to our program and use it efficiently.

2.2 Requirements for the map

There were a few required features in this project. The requirements were presented to us in three steps during development. This made it easier for us to focus on getting the basic features of the program to work, but it made it harder for us to plan ahead as well. Here is a list of the features we had to implement:

- We had to make a visual representation of all the roads from the data set (see section 2.4).
- We had to draw different kinds of roads in different colors.
- We had to adjust our drawing of the roads, to the window size of our user interface.
- We had to make mouse zooming possible. The user should be able to drag and select an area with the mouse, creating a square or rectangle, and zoom in on the selected area.
- We had to implement an algorithm to find a route from one point on the map to another, and we had to allow the user to find that route by clicking at each of the points.

We had to consider these requirements while deciding how the program should be designed. Some of the requirements were so basic that we would have made them regardless of them being required.

2.3 Our requirements

In the process of designing the program, we made some requirements, that we wanted to make sure was met, before making more advanced features. Because it was required to make the user able to zoom-in on the map, we found it logical to allow him or her to:

- Zoom out
- Pan around on the map

With these basic features covered, we decided which advanced featured we wanted to implement. In order to give the user a chance to find more specific places, we decided to show the user the name of the road closest to the cursor. This means that the user can get orientated without clicking or pressing any buttons.

The most interesting feature of our Map of Denmark project is perhaps the option of travelling by car or bicycle. Many Danish people use bicycles to get around and as such this feature is a very relevant to have in the software. It is a feature not usually seen on international maps - this is most likely due to the size of these maps.

We also decided to let the user create routes with an unlimited amount of waypoints. This is supposed to make our map well suited for planning bike trips or longer car rides where you want to reach more than one destination. Because this feature can be quite demanding on performance when a lot of waypoints are selected, we had to make sure that the algorithm for finding routes was fast enough to ensure that the map is pleasant to use.

We wanted to make sure that the shape of Denmark is recognizable when the user is zoomed far enough out to show the entire country. There is a balance between showing a big amount of roads and the delay when navigating. We wanted to make the user able to navigate the map with a reasonably acceptable delay.

The fact that we chose specific requirements for our program gave us two big advantages. It both made the planning process easier, and it made the design phase a lot easier, when we knew what the program should be able to do. In the process of creating the program, we had to change and create new requirements for ourselves, when we felt some feature was necessary for the end-user to have available to him. In the last part of the coding process, we decided on our final requirements and worked towards completing them.

2.4 Data set

We have been provided with a data set of roads and intersections in Denmark as text files for this project. Additionally we were given some code for loading the data from the text files.

2.4.1 UTM-coordinates

When using the UTM standard the earth is divided into 60 rectangles that each have the origo at the south-west corner. Most of Denmark is within the UTM-32 rectangle, so all the coordinate data is given as UTM32-coordinates. These coordinates need some conversion when used in Java since the origo is placed differently in Java. This will be explained further in the section about UTM-conversion on page 16.

2.4.2 Graph

A Graph is a wrapper object for storing nodes and edges. This data can be considered as points (nodes) that can be connected to each other by lines (edges).

When the data has been loaded it is stored as a Graph object containing KrakNodes and KrakEdges. The KrakEdges are the road segments and contains the name of the road, an estimated drive time, a direction of traffic and references to the two KrakNodes that are at either end of the road. The KrakNode itself contains only the coordinates for the point. The Graph contains a number of useful methods for searching the data like getting all edges that are connected to a KrakNode. We will be using these methods in the project to find a route between two points.

2.5 MVC structure

In order to achieve a decent code structure, it is important to split the responsibilities of the program into different parts, which work together to make the program work as intended.

One way to do this is to have the graphical user interface in a class of its own, and the rest of the program in another class. The downside to this approach, is that it can get ambiguously where to put specific pieces of code (button listeners and the like).

We chose to utilize the Model-View-Controller (MVC) architecture, which is another way of structuring a program (in most cases a better one). By using MVC we divide our code in three main parts, in order to achieve a decent separation of data, logic and the graphical interface.

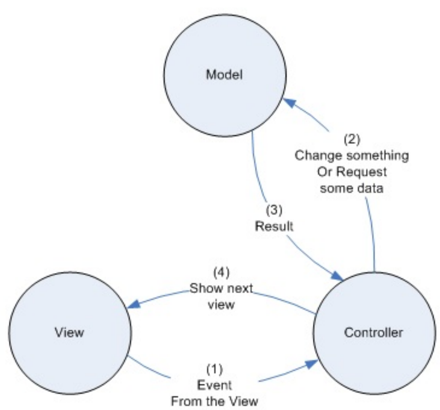

Figure 2.1: Overview of MVC

Every graphical window has its own class. These classes are called “views”. Where we beforehand had one class to handle both communication with the data and the graphical windows, we now split this into two: “models” and “controllers”.

The models handle all communication with the data sources, and every model handle one data source. If we were to use relational databases, we would have a model for each table in the database. In our case we only have one data source (the provided data set), and therefore only have one model, to communicate with this.

Now we only need a way to connect the graphical interface with the data. This is where controllers enter the picture. In MVC you have one controller per view. This controller provides the view with all the necessary data from the models. The controllers also take care of all data processing. It translates data that the model can understand, to something the view can understand, and the other way around. Another thing the controllers are responsible for, is listening to its view. The controller have so called “listeners” that listen to events on the view (like when the user presses a button, clicks his/her mouse, etc.). Then the controller can provide new data to the view or save new data from the view to the data source (through the models).

An example of this could be that the user updates some info in a program and presses the “Save”-button. The controller listens to this event, passes the new data to the models and it gets saved. In our case we have listeners to (among other things) mouseclicks, so we can place markers (part of our pathfinding) when the user clicks the map.

Chapter 3

User Interface analysis

In this chapter we describe our decisions and present our analysis and arguments regarding possible features that we find might have been interesting to have implemented in our Map Of Denmark program.

3.1 User interface as a whole

When we designed the first version of the graphical user interface in the first part of the project, we decided to make a canvas inside of the graphical user interface, where the actual map should be displayed. We chose to have this canvas placed on the right side of our graphical user interface and interaction with the user mainly placed on the left. See figure 3.1 for an example.

We believe that this is a simple way of representing a user interface for a map. A lot of software use a menu bar with dropdown menus for selecting different functions. When we designed our outline for the graphical user interface, we did not design it with a huge amount of functions in mind.

The features that we have implemented in this version can easily fit in our simple user interface, but if features like searching for roads or other features are included, then space and overview may become an issue on the left side.

If new features are included, we feel it would be beneficial to let the main window change when different feature types are selected.

How to use our will be explained in the Manual on page 32.

Figure 3.1: Screenshot of GUI

3.2 Interesting features

This section presents some of the interesting features we have implemented.

3.2.1 Zoom

We have a few options for zooming in and out on the map. As described in section 2.2 Requirements for the map on page 6, it was required that we made it possible to zoom by dragging a box around the part of the map the user wants to look closer at.

In addition to the option of using the mouse to zoom, we have implemented a zoom-in and out function on the GUI and a hotkey for zooming out to the original zoom level. We made the original zoom level function a hotkey only because we did not want to have too many buttons on the left side. We considered making it a menu bar item, but we did not manage to get it into this version.

We felt we really needed a zoom out function, so users do not need to close the program and start it again, when the user wants to view the map further zoomed out. A combination of the zoom in and out functions helps the user a lot when navigating the map.

We have limited how far a user can zoom in and out. If the user tries to zoom out further than the original zoom level, the view will default to the original zoom level. If the user attempts to zoom in too far compared to the models view, the zoom function will do nothing.

3.2.2 Navigation

We have made it possible for the user to navigate the map by using the arrow buttons on the graphical user interface. When one of the buttons are pressed, the "viewport" will move in the direction specified by the button. While it was not specified as a requirement for the project, we felt it was a necessity to implement at least basic navigation functionality.

Like we did with the two zoom functionalities, we have limited how far a user can move around the map. If the user moves too far outside the map, the move function will not do anything.

3.2.3 Hotkeys

We have implemented hotkeys for all the buttons on the graphical user interface plus an additional hotkey for zooming back to the original zoom level. When we discussed the benefits of hotkeys, we felt it was important for experienced users of the software to have a less cumbersome time navigating the map.

At first we just had hotkeys for clearing the markers (mentioned in section 3.2.4) and zooming out to the original zoom level, but we later added hotkeys for the rest of the functionalities. If more features are added in a future version, it would be important for us that a hotkey were provided, if at all possible.

3.2.4 Route planning and markers

Part of the requirements for the project was to provide the user with a way to get the fastest or shortest route from one point to another. We accomplish this by putting a "marker" at the spot where the user clicks with the mouse. The marker shows which number in the sequence of markers it is. This will change if a marker is removed. Originally we had "pins" instead of markers, but we changed it, as we felt the pins we had were a bit large.

We have made it possible to place more than the two markers the project requirements asks for. If the user places more than two markers, the software will find the shortest route between 1->2 and 2->3 and so on. This was cheap for us to implement, and we felt it added a nice touch to our program.

We have implemented two ways of removing markers from the map. We have assigned a hotkey to the graphical user interface button "Clear Markers", which removes all the markers from the map. The other way of removing markers is by clicking on them. This functionality is both intuitive and confusing at the same time. It is intuitive to click the marker you have just placed if you want to remove it, but it is not obvious in our interface. We believe that it is enough to have the "clear all markers" functionality for those who do not find it intuitive to click markers to remove them, and for the users that do find it intuitive, we offer them an easy way to undo a missclick.

3.2.5 Bike/car

Another feature in our Map of Denmark project is the option to switch between bike and car routes. The user interface will start with car selected when the program starts.

Whenever a route is calculated, it will display the length and the estimated travel time on the left part of the user interface. When the bike option is selected, it recalculates the route for a bike, without visiting highways and other roads that bikes are not allowed to drive on. If the user switches back to the car mode, it recalculates the route again, but not visiting small paths and other roads where a car is not allowed to drive. The estimated travel time is also recalculated. The user does not need to have planned a route before he/she changes the type of transportation.

The bike time is calculated using a set speed.

We have implemented this to help our software target a wider group of people. The bike/car option were a bit costly to implement, but we categorized it as a very beneficial feature and we did not feel we could leave it out.

3.3 Features not implemented

This section presents some of the features we chose not to implement. These features are not in the final program, because we did not feel there were compelling arguments for implementing them.

Features that we wanted to implement, but did not make it into the final version, will be discussed in chapter 8 Product conclusion on page 36.

3.3.1 Choice of road types to be displayed

We chose not to implement the option of selecting which road types to be displayed. In a sense our program already does this by showing more detail the further zoomed in the map is. It could become very confusing if the user had the option of selecting roads, because the graphical user interface could become very cluttered, if all the roads were listed.

We felt that our gradually increasing level of detail is enough, and if the user were to enable all types of roads at once, the map could get quite slow and cumbersome to use.

If we were to implement something like this, it would be to give the user the choice of which types of roads should be included in the route planning - ferries, highways, bridges etc. This could be too big a choice for the user to handle though. The user might doubt if he or she has chosen the best options for finding a route.

3.3.2 Smooth scrolling

We made an attempt to let the keyboard arrows scroll smoothly over the map, but we could not get it to work fast enough, and the user would experience lockups and an unresponsive interface. A solution to this could be to save the map as images that you can scroll across - this would be faster, but would require more disk space. Because we store the data the way we do, which forces us to draw every line individually every time the user moves the viewport, we cannot do this fast enough.

In the end, we decided the benefits of the smooth scrolling were not large enough for us to spend a lot of time implementing this feature. The cost of changing the way we draw the map was simply too high compared to the benefits.

3.3.3 Dynamic route finding

In the final project description, a dynamic route finding feature was suggested. If we had implemented the suggested dynamic route finding feature, a user would be able to mark a spot and then whenever he moused over a node on the map, it would show the route instantly.

We considered implementing this as we thought it was a nice feature to have, but it conflicted with the algorithm we use for calculating the route. More about the algorithms can be read in section 4.3 Dijkstra vs A* on page 18.

Chapter 4

Implementation

Our implementation uses the MVC structure to create a practical class design. We have prioritized making our design as simple and intuitive as possible. We used object oriented principles to assure that our design has both low connection between classes (loose coupling) and a high degree of cohesion.

By following these principles, we have achieved implementation with little code duplication and with good possibility of extension.

Some of the basic thoughts on our design is explained in figure 4.1 and figure 4.2.

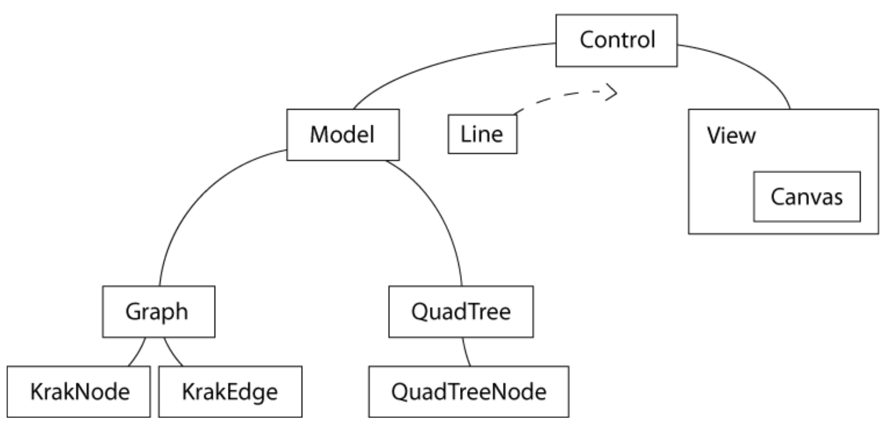

Figure 4.1: The basic design of our implementation. The controller gets a collection of lines and the view draws them.

This rest of this chapter describes how we have implemented some of the more interesting features of the software. We aim to describe it to enough detail that

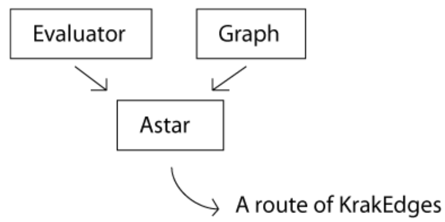

Figure 4.2: The pathfinding part of our implementation. The static A function takes an Evaluator, a Graph, a start* note *and an end* note *and returns a path.*

this chapter can serve as a guideline for implementing the functionalities we describe.

4.1 UTM-conversion

When the graphical user interface part of the map tries to communicate with the model through control, some conversions of the different kind of values are necessary. Both when going from coordinates in the Java-coordinate system to UTM32-coordinates and back.

We need to convert the values when we want to use the mousezoom and when we want to place the markers for pathfinding. We get an input on the graphical user interface when we use mouse zoom and this needs to be converted to UTM-coordinates so we can create the new boundaries of the zoomed rectangle.

When we place markers for pathfinding, we do the same as when we mouse zoom, but instead we store the point as UTM32-coordinates and whenever we move the map, we convert it back to pixel-coordinates so that we know where to draw the markers.

The Java-coordinates have origo placed in the top left corner with the y-coordinate increasing the further down the y-axis you go. UTM32-coordinates are a bit different. UTM32 has origo in the bottom left corner and the y-coordinate increasing the further up you go on the y-axis.

We have a utility class with methods for converting the points back and forth. Class PointMethods uses the below calculations to convert the points.

Figure 4.3 is an illustration of the conversion from pixel to UTM.

We click on a point, 'a', on the canvas. To calculate the UTM coordinate corresponding to that point, we use the formulas:

$$UTM_x = bounds_x + \frac{a_x}{canvas_{width}} \times bounds_{width}$$

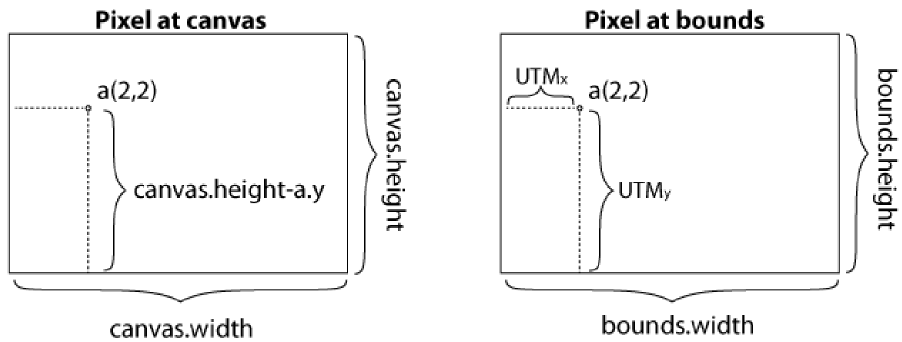

Figure 4.3: Illustration of UTM conversion

$$UTM_y = bounds_y + \frac{canvas_{height} - a_y}{canvas_{height}} \times bounds_{height}$$

To convert from UTM to pixels, we use the same formulas but reversed:

$$Pixel_x = \frac{(a_x - bounds_x)}{bounds_{width}} \times canvas_{width}$$

$$Pixel_y = canvas_{height} - \frac{a_y - bounds_y}{bounds_{height}} \times canvas_{height}$$

4.2 Mousezoom

We have implemented mouse zoom by using mouseEvents on our canvas. When the user presses the left mouse button, it generates a mousePressed event. We record the position of the mouse at the time of the mousePressed event and wait for the mouseReleased event. When the left mouse button is released, it generates a mouseReleased event. We use the positions of the mouse at the mousePressed event and the mouseReleased event to calculate new bounds for the model.

Often the user will not drag a square that is in perfect ratio with the canvas. If it does not have the same ratio as the canvas, we change the ratio of the dragged square behind the scenes. We do this by either adding length or width to the dragged square. We always make sure to at least show what was inside the box the user dragged.

If the ratio of the dragged square is smaller than the canvas ratio, we make the dragged box wider. If the ratio of the dragged square is larger, we make the dragged box taller.

4.3 Dijkstra vs A*

When planning the pathfinding feature, we had to decide between two algorithms: Dijkstras algorithm[1] and A*[2]. The latter is based on Dijkstras algorithm, but achieves better performance in some situations by using heuristics.

Dijkstras algorithm uses a minimum priority queue[3] to find the shortest path from a given node to every other node by looking at the edges connecting the nodes. The algorithm will take a node from the priority queue and add all the other nodes that are connected to the current node to the priority queue. The priority queue stores a value with the node, which is the distance to the current node plus the length of the edge between the two. Since the priority is made to return the node with the smallest value associated with it, the next node in line will always be the one which is closest to the start node. This procedure continues until all nodes have been visited and by logging what edge led to all the nodes, it is possible to trace back the route to the start node.

This algorithm is great if we need to find the distance from one point to many other points, but can be quite slow when using it for finding a path between two nodes, since it just searches in all directions without considering which direction the target node is.

This is where A* is different. The A* algorithm is a modification of Dijkstras algorithm that also looks at the estimated distance from the given node when determining the value for the priority queue. When using the geographical distance as a measure of best route, the value would be the current distance from the start node plus the direct distance to the target (as if there were a road directly to the target). See figure 4.4 for an example.

With this subtle change, the algorithm will prioritize nodes that are relatively closer to the target rather than those that are in the other direction. This makes the algorithm much faster since it will not pay much attention to the roads that are not in the direction of the target. We have decided to use the A* algorithm since we only calculate routes between two distinct nodes and therefore don‘t need the route from the start node to all others. The time reduction that A* gives is also a definite plus since no user wants to sit and wait too long for the program to find a route.

The smaller the difference from the direct distance is to the actual fastest possible route, the more we will benefit from the A* algorithm. This difference will often be bigger for the routes calculated for the car. This is because the fastest route would be a straight highway, which often is far from possible. We can

[1]We had a lecture about this in the Algorithms and Data Structures course and read about it in [1, p. 556].

[2]The advisors explained how this was faster and how we should change Dijkstra to turn it into A*.

[3]The minimum priority queue has been borrowed from [2]

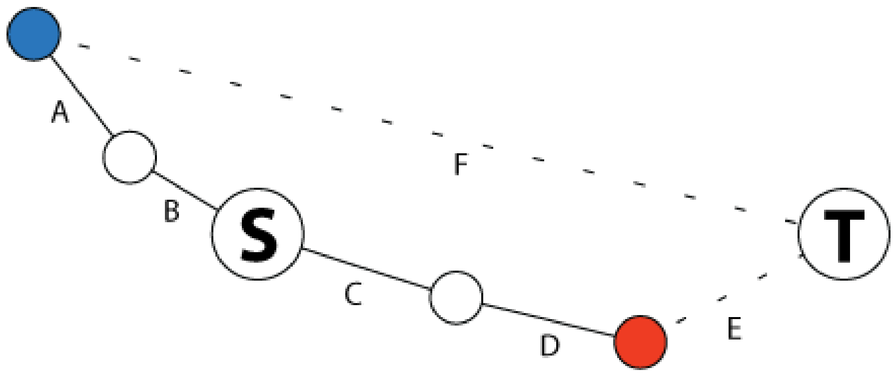

Figure 4.4: A vs. Dijkstra example. The dashed lines are the direct distances, used by A*. The Dijkstra algoritm will check the blue node before the red node, because B+A <C+D. The A* algorithm will check the red node before the blue node, because C+D+E <B+A+F.*

therefore conclude that the A* algorithm typically is more of a benefit when calculating routes for bicycling because they rely on the distance.

4.4 Evaluator

In order to make our path finding algorithm flexible enough for different interpretations of the "best route", we have added an class called Evaluator. This is an object that has the responsibility of evaluating a node relative to the target node.

The Evaluator also has the responsibility of calculating the heuristics that the A* algorithm relies on. By using the Evaluator we are able to use the same path finding algorithm for two very different tasks; the biking route and the car route. The major difference between these is that the bike's heuristics is based on the distance, whereas the car's heuristics is based on the total drive time.

$$dist(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$time_{drive} = \frac{dist(p_1,p_2)}{1000 \cdot \frac{speed_{max}}{60}}$$

We use these formulas to calculate the heurestics for bike routes and car routes, respectively. There are currently three different evaluators in the program, saved as static variables. These are called BIKE, CAR and HAS_NAME. The

HAS_NAME evaluator is used when getting the name of the road closest to the mouse. There is also a fourth variable called DEFAULT that is a help to any future developers if they are unsure of what to use. This variable is currently just referring to CAR.

The Evaluator implementation is a good example of making our code ready for future features, since if we needed to add other means of transportation or simply variations of the ones we have, we would only need to create new Evaluators and not change a single line of code in the A* algorithm.

4.5 Quadtree

In order to improve the performance drawing of our map, we have implemented a data structure called "quadtree". The quadtree divides our map into four rectangles. In these four rectangles it divides our map again into smaller rectangles, and we keep doing this, until we reach an amount of roads that is manageable. When we want to retrieve data from the map, we can give the quadtree a rectangle, and it will return all the roads within our smaller rectangles that is contained in or intersects the given rectangle. This technique optimizes the drawing of the roads, because we greatly limit the amount of roads we draw.

However, when viewing the entire map, this implementation does not help us. Therefore, it is necessary to only to draw the bigger roads when zoomed out. We have discussed two different ways of doing this.

The simplest solution would be to iterate over the roads returned and then remove the roads we do not want to draw. This is illustrated by figure 4.5. However, it would be more efficient to sort the roads before we put them in the quadtree, and then have them sorted in the quadtree. By doing this, we will not have to iterate over all the roads and remove them all the time.

We have discussed two different ways of sorting the roads in the quadtree.

The first way relied on dividing the types of roads into different quadtrees. With this implementation, we have one quadtree for each level of detail we want to have and only search in the quadtrees that are needed for a given rectangle. This is multiple quadtrees, see figure **??**.

The second way relied on putting the bigger roads at the top of the quadtree when building it. Then we could specify at which depth we wanted to search the quadtree. We called this 'the depth-controlled quadtree'. Illustrated by figure **??**.

The two methods both have advantages and disadvantages.

The depth-controlled quadtree would return more roads than the multiquadtree, because the large roads (highways, etc.) would be in the outer rectangles, and

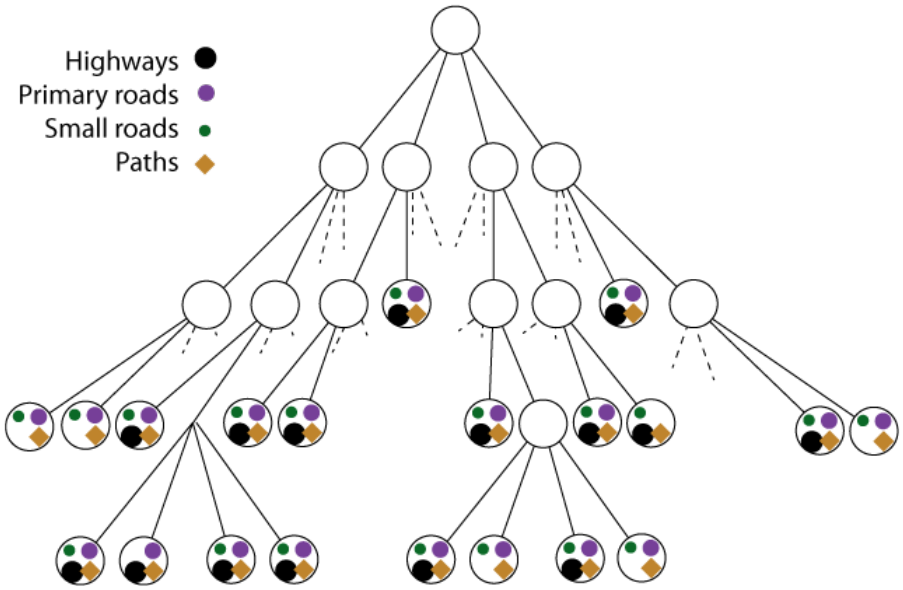

Figure 4.5: Unsorted Quadtree. A quadtree where the roads are not sorted. The dashed lines are parts of the quadtree that we do not show for practical reasons.

therefore be drawn more often.

The depth-controlled quadtree requires less RAM than the multiquadtree, because it requires less instances of object QuadTreeNode.

The multiple quadtrees are slighty less efficient to search through than the depth-controlled quadtree, because we must run through each quadtree individually.

Looking at the advantages and disadvantages of these quadtrees, we have chosen to implement the multiple quadtrees. We have done this because it makes it a lot easier for us to layer the roads by their roadtype. Also it is the drawing of the roads that slows down our implementation. The time needed for searching through the quadtree is decided by the amount of roads in it, which affects the height of the tree. When using more quadtrees, the amount of roads are smaller, and because of that the height is smaller. Even when using multiple quadtrees, the efficiency of the map is primarily affected by the number of roads drawn, not the amount of quadtrees.

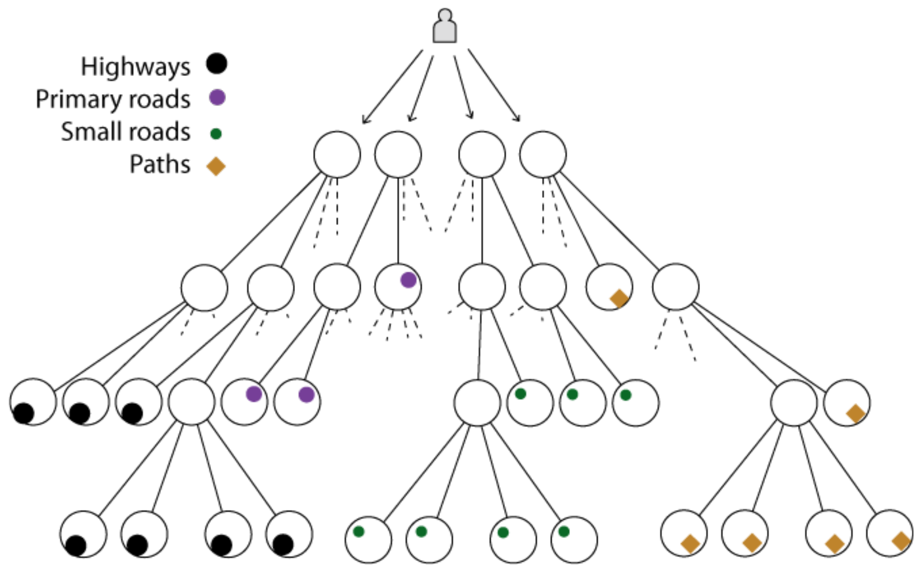

Figure 4.6: Multiple quadtrees. The user can choose which quadtrees to run through.

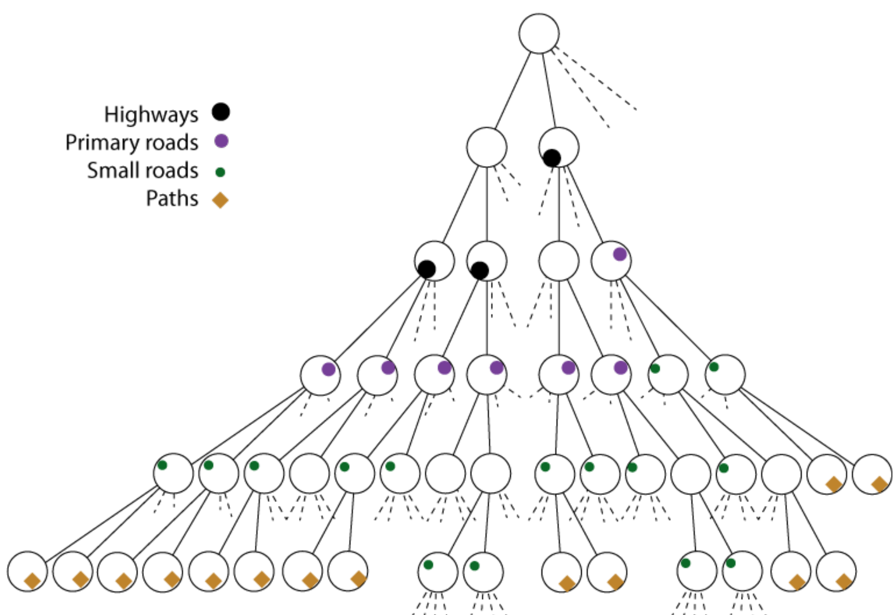

Figure 4.7: The depth-controlled quadtree.

4.6 Serialization

We observed that the user had to wait quite a long time for the program to start. This was because every time we start the program, we loop through the entire provided data set. This data set is huge, and because of this it takes quite a lot of time to start the program. Because we need to load all this data, the user is presented with a blank screen for a long time, before all this data are loaded and the program starts.

We started looking for a way to speed up the loading process, so the user has a map in front of him or her quickly, when the user starts the program. What takes the most time is looping through the data and creating the needed datastructures (quadtrees, the graph and so on), so if we could skip these steps or speed them up, we could save a lot of time.

This is where serialization comes into play. By serializing an object, you transform your object into something that can be passed around through streams. So by serializing objects, you can save them to files. If the object to be serialized contains references to other objects, these will also be serialized (if they are Serializable / implements `java.io.Serializable`).

By doing this, we only need to build our datastructures the first time you start the program. After the objects have been created, they are been serialized and saved to files. The next time the user starts the program, we check whether the data has been changed. We check this by checking the MD5 checksum of the file, with `MD5Checksum`[4]. When we serialize the objects, we also save the checksum to a config file. If the data has not been changed (i.e. the checksum is the same), we load the objects that we saved, instead of making them all over.

We serialize all the quadtrees, the graph and the maxbounds-object (specifying the bounds of Denmark), as these are the objects we need for the program to run. When saved to the harddrive, it is around 65MiB, which should not be an issue, given the size of harddrives today.

We save all these objects to one single file in order to keep the references. If we did not do this, the references will be ruined, which can break the program. We experienced problems with this, as `nodes` and `edges` are both stored in the quadtrees and the graph. If we serialized and saved the quadtrees in one file, and the graph in other file, a given `node` would be saved twice, and when we load the data from the serialized files, the `node` will exist twice, and it won't be the same object. But if we save the objects to a file through the same stream, we only get one of each, which leads to less RAM usage, less harddrive usage, faster loadtimes and hopefully fewer bugs.

[4]We borrowed this code from [3]

4.6.1 Threading

By serializing our main objects, we cut several seconds of our load times. But we can do it even faster. We are serializing several objects, but only few of them are needed right from the start of the program. So what we can do to speed it up even more, is loading the few necessary objects, and then load the rest in the background. We do this with threads. We load the few objects we need from the start, then create a new thread to load the rest of the objects, and in the mean time, we create the interface and draw the map.

The same goes for the first run. The user does not need to wait for the program to finish serializing and saving to files. By using threads, we can create the data structures, and then immediately show the window to the user, while saving the objects to files in the background.

During load time (when loading from the serialized files, after the window is shown to the user) not all quadtrees are loaded. So we did encounter a problem when querying the quadtrees, when not all of them were loaded, as our code would try to use quadtrees not yet loaded. We solved this by putting the querying in a try/catch, and when a problem occurred (index out of bounds, when trying to access a quadtree not yet loaded), we simply stopped looking through more quadtrees and just return what we found so far. Then later on, when all quadtrees were loaded, we could return all edges. The user would not notice the lack of roads, as the user only sees a limited amount of roads anyway.

We did encounter another problem, when trying to use the graph (for pathfinding) when the graph was not done loading. We solved this by using a simple loop, that check if the graph was set. As long as the graph is not available, the main thread would simply hold and wait for the loading to finish. This is probably the only time where the user would notice that everything is not quite loaded, but the loading happens so fast, that it probably will not be a problem anyway.

With the nature of threads in mind, we cannot guarantee this behaviour, but no problems should arise if the loading occurs in the wrong order.

Chapter 5

UML-diagrams

Figure 5.1 on page 26 shows our implementation of the MVC architecture. Because we have exactly one window, we chose to name our view and controller "View" and "Control", respectively. The same goes for the models. We only have one data source (the provided data set), and because of that our model is called "Model". We could have named these three classes something different that may have been more meaningful in terms of the Map of Denmark context, but we chose these name in order to make our architecture clear.

We wanted to keep the classes as "trimmed" as possible, although our Model is quite long. But the amount of (public) methods is small, so looking at it from outside, it is a skinny architecture. The reason we wanted to keep the classes skinny, is because it is not the models responsibility to deliver the same data in different ways, do a lot of calculations or provide similar functionality. It only acts like a "middle-man", delivering data to other classes. If some class wants the data in another way, they will need to convert it themselves.

Control makes sure the models and the view can understand each other. The view only knows about pixels, but it has no idea about UTM coordinates. The model only knows about UTM coordinates, but does not know anything about pixels. So for getting these two to communicate, we need to convert pixels to UTM and vice versa.

We created some helper-classes (PointMethods and RectangleMethods), which are located in the utils package. These take care of checking whether a point is within the maximal bounds of the map, converting a pixel coordinate to a UTM coordinate and vice versa. We did this for being able to do this in several files, without the need to have duplicate code. For our program right now, this is not really a problem, as we only have one model, one controller and a view. But if we were to have more, we would either need to copy these helper-methods into the other classes (BAD), or put them in a separate class. But even though we only have one of each, the helper-classes are still an advantage, as they make the code cleaner and easier to maintain and test, because we can test these helper-classes seperately.

So in essence, we have two parts (the model and the view) that need to communicate somehow, in order to display the data on the screen. But they speak different languages, so we put in a middle-man (the controller), responsible for the communication between the two.

5.1 Control flow

The easiest way to understand our flow through the program and how the individual parts talk together, is by using an example. Let us say the user has already clicked the map and placed a marker. Now the user clicks on the map again to place another marker.

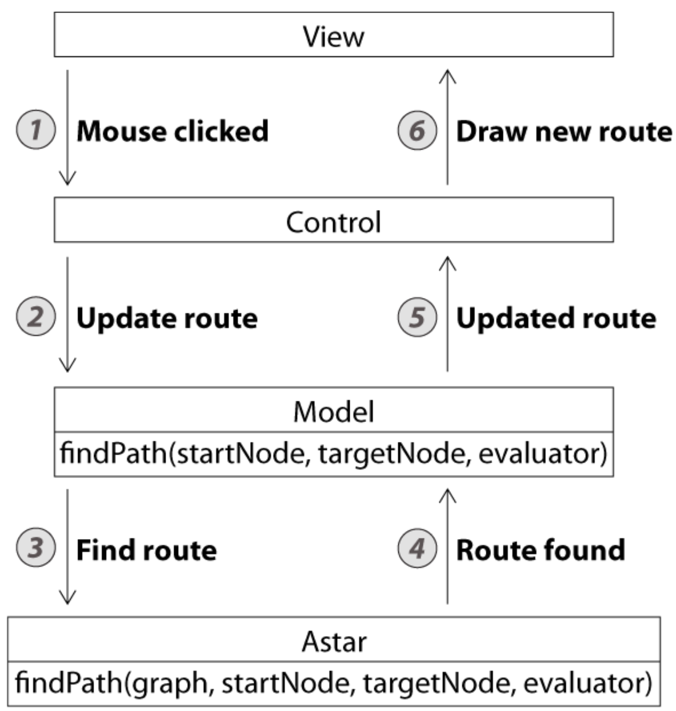

Figure 5.1: The basic control flow of our Map program

The controller has placed listeners in the view, so when a MouseClicked event happens, the controller is called. First it checks if there is another marker at the spot of the click. If there is, this will be removed, and the model is told to clear the route. If there is still over two markers placed, the model is asked to calculate a new route.

If there is not a marker where the user clicks, we place a new marker. If the user has placed two or more pins, the controller calls its own findPath-method from point 1 to point 2, point 2 to point 3 and so on. The findPath()-method tells the model to find a path between the two points given.

The model then asks a helper-class to find a path, using the A* algorithm, and provides it with the graph and the two points. When a path is found, it is saved in the model, ready for use in the controller.

The final step is getting the view to draw the route. The controller gets ready for a repaint by fetching the route from the model. Then it passes this route to the view's repaint-method, and the view paints the road.

Chapter 6

Tests

6.1 White box: closestEdge

A white box test is a test that is made with knowledge of the internal structure of the code, and uses this knowledge to make sure all flows through the code is covered. We are using the "getClosestNode" method for this test since it contains both loops and conditional statements. The method is supposed to find the closest Node to a given point. It accomplishes this by first searching another method for the closest Edge (which is trivial). This procedure is performed with greater and greater search radius until it finds something. When it has found an Edge, it will determine which end (Node) is closest to the point. In order to secure full coverage of the tests we have created the following scheme of tests and expected results (using our test graph seen on page 30).

Part	**Case**	**Input**	**Expected Output**
while(once)	Some point that finds an Edge within the initial search radius	6000, 2000, Evaluator.CAR	Node with index 2
while(more)	Some point that has to widen the search radius in order to find anything	-200, -200, Evaluator.CAR	Node with index 4
if-block	The first Node is closest	10000, 8500, Evaluator.CAR	Node with index 8
else-block	The second Node is closest	8500, 8500, Evaluator.CAR	Node with index 7

This procedure is great for testing every corner of the code because we are making a test for reaching all sections within if-statements and make sure we run while-loops zero, once and multiple times[1]. It should be noted that the white box tests should be accompanied by black box tests, because these tests uses the input needed for reaching a specific block. This means they do not check if invalid input breaks the program. Since some of the input overlaps with what is needed for the black box, tests they have been located in the same method in our Model Tests found in appendix A.1.3 on page 47[2].

6.2 JUnit

This section describes the JUnit tests we have made for our software. We have made tests for the public methods in Model, PointMethods and RectangleMethods.

6.2.1 ModelTest

The model class has some of the most interesting functions and algorithms of our program. The most advanced functions in the model are those regarding path finding. The task of finding a path from one node of the graph to another, takes many different kinds of input. The coverage table for the JUnit tests of the Model class can be found in appendix A.1.3 on page 47.

The tests are chosen so that they test for different input. But they are also chosen so that they test the logical problems our path finding algorithm might run into. In order to test these problems properly, we constructed some simple test data that we knew we could rely on:

Figure 6.1 on page 30 is a visualization of the graph that we use in our tests. The graph is constructed so that we can test different problems that might occur in the graph our map uses.

6.2.2 PointMethodsTest

We have tested the public methods in PointMethods using JUnit testing. The complete coverage table can be found in appendix A.1.1 on page 45.

[1] We cannot run this while loop zero times since the loop runs when edge == null and the edge variable is defined as null just before the loop.

[2] In the testGetClosestNode() method, the first, third, fourth and fifth asserts.

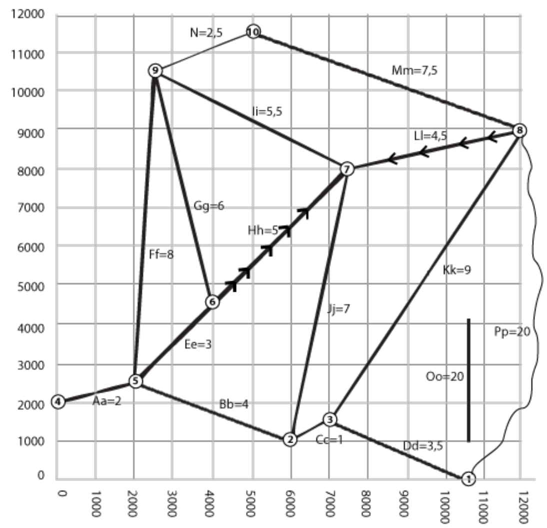

Figure 6.1: Test graph. The name and length of each edge is shown. The edge named Nn is a path and is therefore undriveable for cars. The lines with arrowheads are directed.

6.2.3 RectangleMethodsTest

As with the PointMethods and Model classes, we have created JUnit tests for the public methods in RectangleMethods. We considered doing more comprehensive testing of the "newBounds", "fixByInnerBounds" and "fixByOuterBounds", but since there are no checks for null or other types of bad input for these methods, we felt it was unnecessary. The coverage table can be found in appendix A.1.1 on page 45.

6.3 System test

We have written a number of system tests that need to be performed by a user interacting with the program. These test the basic functionality of the program as well as some advanced combinations of features. The system tests are a necessary addition to the unit tests since many bugs are located in the interactions between algorithms and the user interface, and thus can often not be found through unit tests.

The tests have been divided into six groups that increase in complexity and the later tests may rely on success in previous tests. There are a total of 34 tests and the complete list and coverage table with instructions and expected results can be found in appendix A.2 on page 50. All the tests in the coverage table have been done on a computer with the Windows 7 operating system.

Chapter 7

Manual

This manual will explain the basic use of the Map of Denmark, as well as its advances features. The images are from Mac OS X and the map might look slightly different on other operating systems.

7.1 Navigation

At startup, the entire country of Denmark is shown. We can move around the map with both the direction buttons in the top left corner, and the arrow keys. To move west, click the button  or press the left arrow key on the keyboard. This goes for all four directions. Figure 7.1 illustrates a moved map.

7.2 Zoom

To zoom-in on the map, click the 

button in the navigation panel or the i key on the keyboard.

To zoom-in on a specific area, click the mouse and drag a rectangle around that area on the map. A blue transparent rectangle will show you what you have selected. To zoom-in, release the mouse-button.

For example, if you want to zoom-in on Copenhagen, click the upper left corner of the city, and drag the cursor to the lower right corner. Figure 7.2 illustrates this.

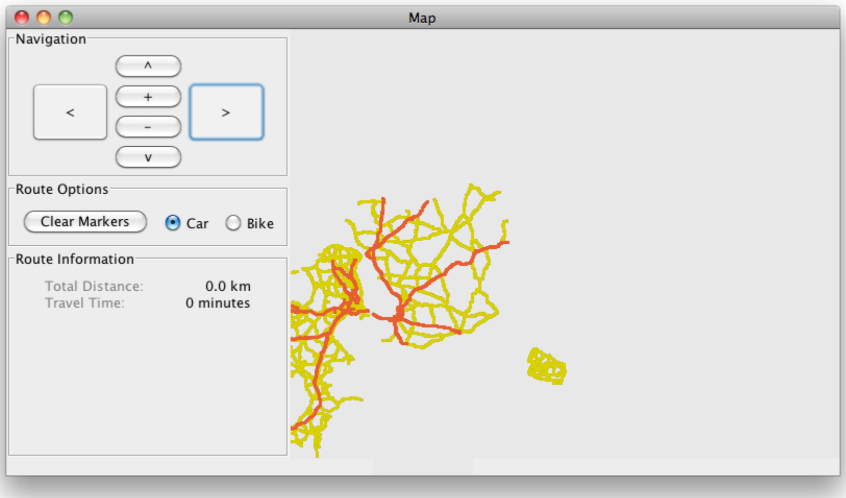

Figure 7.1: Moving the map

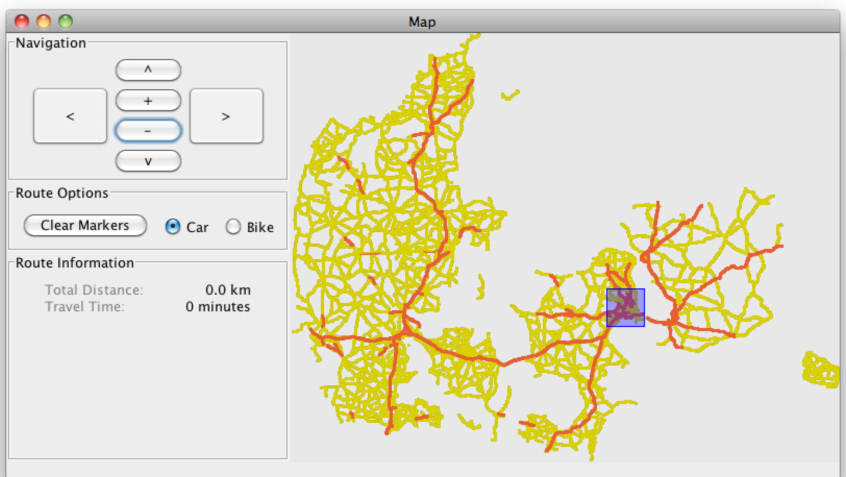

Figure 7.2: Mouse zoom box dragged around Copenhagen

To zoom-out on the map, click the  button or the `o` key on the keyboard. To return to the startup view (showing the entire map of Denmark) press the `ESC` key on the keyboard.

7.3 Route find

To find a route from one point to another, you must specify a start and an end location. Click anywhere on the map to choose your start location. A light blue marker will appear, containing the number “1”. To choose your end location, click at another location. A marker containing the number “2” will appear. The best route from 1 to 2 will be calculated, and shown on the map as a blue path. This is illustrated by figure 7.3.

To extend your route with more markers, you can click at a new location on the map. You can repeat this as many times as you like. You can delete one of your markers by clicking at its root. To delete all markers, click the button 'clear markers' or press `c` on the keyboard.

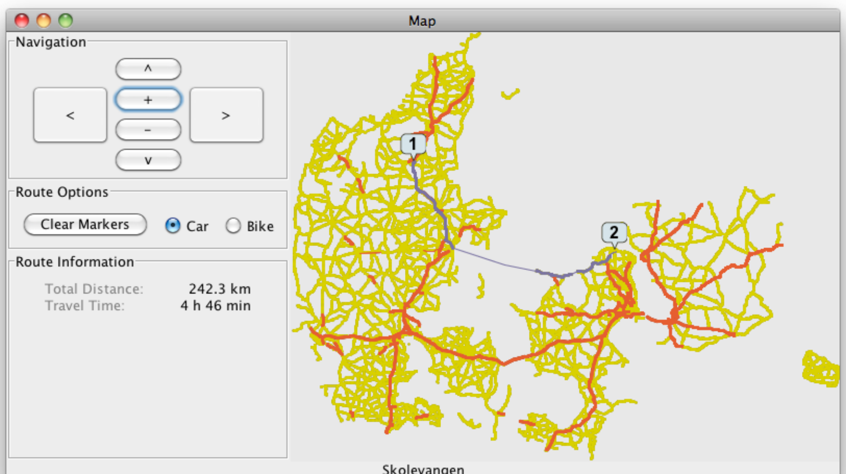

Figure 7.3: A route from 1 to 2 has been calculated

7.4 Bike/car

When calculating routes, it is important to specify which form of transportation you wish to use. Choose your preferred style of transportation by selecting the corresponding radio button.

The form of transportation you use will have big influence on what route is calculated. If the car option is chosen, the route will be optimized for car transportation. If the bicycle option is chosen, the route will be optimized for bicycling.

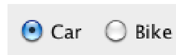

Figure 7.4: The radiobuttons

7.5 Resize

To resize the map, drag the window as you would with any other application. The map will automatic adjust to the new size of the window. See figure 7.5.

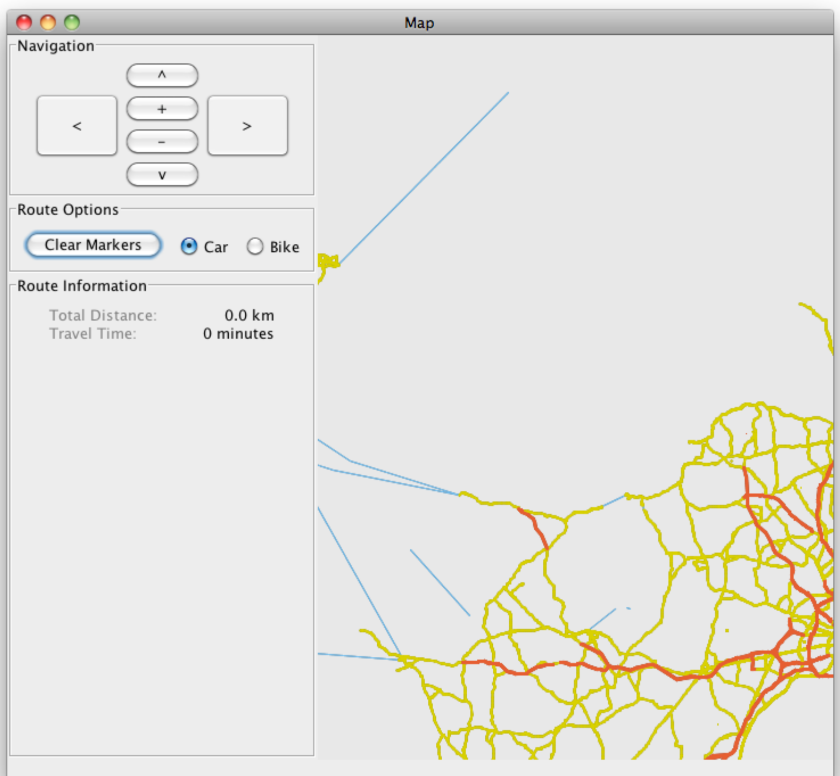

Figure 7.5: The map has been resized

Chapter 8

Product conclusion

We believe our product live up to the project requirements described in chapter 2 Background. Our program also runs fast enough when using the interface, in our opinion. We tested this on both a fast computer and an older, slower computer and we did not feel it was cumbersome to use the program, even on the slower computer. We did not thoroughly test this however, and we cannot guarantee that there will not be circumstances where our program will run slowly.

The system tests were all done on a Windows 7 computer and because of this, we cannot be certain that all features work on other operating systems. However, two of our group members use Mac OS X and they have experienced no problems with our program besides the ones mentioned in section 8.1.

We did all of our testing internally. For a even more thorough testing of our software, we could have asked other groups to test the software for us, so that we could get a different look on the tests. We did not do any usability testing throughout the project, and while our user interface is simple, we should have tested to make sure we designed our interface in the best way possible.

8.1 Issues in the software

We have discovered a few bugs in our software that did not show themselves through our testing, but instead manifested with regular use of the software.

We have a bug in our zoom-function, which enables the user to zoom very far in by zooming in, then resizing the window. This caused the program to be locked at a very far zoomed in view, but we chose to let the user zoom out when they are zoomed far in. The problem with zooming in too far can still happen, but the view is not locked when it happens.

8.2 Future features

If we had more time, we would have liked to have included more features in our program. We feel our program could still do with some fine tuning for it to be of real use besides living up to the requirements. During development, we have focused on getting the basic requirements to work properly, before working on more advanced features. The additional features that we would have like to implement is described here:

We would have liked to have implemented a "road search" function, enabling the user to search for the name of a road and place a marker there. In addition, we would have like to implement the option for the user to view a route description with turns. We were actually working on making a route description, but we could not get it to look decent in time, so we felt we would be better off without it.

To make the map look nicer, we attempted to implement a function for the map to create an outline of Denmark, separating the land from the sea, so that it was more obvious what is land and what is water. It would first of all make our program prettier, but with it we would be able to draw fewer roads when zoomed far out, resulting in better performance. We would also have liked to make a logo for our program, but we did not get around to implementing it.

Chapter 9

Group norms

We wrote a constitution before the project was handed out to us. In this constitution we describe what we require of each other and ourselves in terms of working on the project. We felt we made some rather strict requirements so that we were sure to get some work done. This backfired a little bit, because we were not able to keep all of the agreements we made, but it worked for the most part. We later changed the agreements a bit once our schedule cleared up from lectures early May.

This is the requirement part of our constitution:

- Check mail at least once a day
- Tell the rest of the group in time if you have trouble getting things done on time.
- Admit when you are not done on time.
- Do not waste time when we have meetings.
- Respect that different people work in different paces and different ways.
- We need to evaluate often.

We also tried to get a average of our level of ambition. Our goal was always to do what we could manage to do in the time frame that we had and without wasting time. In other words, we strived for being done early by planning ahead and getting the basic features implemented first.

9.1 Meetings

We structured our work together in “meetings”. A meeting was whenever we were together working on the project - these were to be done at the ITU. The structure of a meeting was simple:

- Leader of the meeting presents his plans, if any, for today.
- Leader selects someone to write down what happened at the meeting.
- What have we done since last time?
- Who does what today?
- Work today.
- Fifteen minutes before work ends: decide on homework for next time and select leader of the next meeting.

Before our lectures ended in early May, we had meets Tuesdays and Fridays from 10 AM to 4 PM. After lectures ended, we felt only meeting Tuesdays and Fridays would be too little time spent in meetings. So we decided to make our meetings one and a half hours shorter, but instead meet on Mondays, Wednesdays and Fridays from 10 AM to 2:30 PM.

Chapter 10

Diary

In this project we have kept two types of diaries. One type in our meeting documents and one in a spreadsheet. The meeting document diary was done from meeting to meeting, so that we kept track of what everyone had done before each meeting.

The spreadsheet diary was kept as a seperate document where we wrote down whenever we had done something outside of meetings.

We did not focus very much on keeping the spreadsheet up to date, because we had the meetings (mentioned in chapter 9), where we kept track of our progress in the group. This made the spreadsheet sort of obsolete. We have included it anyway though, as we feel it has been a part of the process.

Our worksheets can be found in appendix B on page 55. They are written in Danish, as we had all of our meetings in Danish. The spreadsheet diary is in appendix B.2 Spreadsheet diary on page 80. The spreadsheet diary is also written in Danish.

Chapter 11

Worksheets

For this project we made a meeting document for all our meetings. We used these to make sure we kept to agreements, homework and had structured meetings. It was important for us to have these, because we wanted to have something to fall back on if we had disagreements and for us to have an easy way of being on top with everyone's homework. They are all written in Danish, becuase our meetings happened in Danish.

The documents are all structured in the following way:

- Leader of meeting
- Keeper of minutes (in charge of the summary for each meeting)
- Homework done since last time
- Work to be done during the meeting
- Summary of the meeting
- Homework

All of our meeting documents can be found in appendix B.1 on page 55.

Chapter 12

Process description and reflection

As a group we have had some ups and downs during the last two and a half months we have been on this project. Filip from our group dropped out early May, so when we were about to start working on this report, we were down to four group members.

We kept a pretty tight structure as far as having meetings pretty often, but sometimes we got a bit unfocused during the meetings. We changed from meeting two days of six hours to three days of four hours and a half once lectures stopped in early May, and this helped a lot. While our spreadsheet diary was not kept up-to-date properply, our meeting documents helped us have a good idea of what was going with everyone in terms of what they were doing on the project. The meeting documents can be found in appendix B.1 on page 55.

Not only did the meeting documents work as our diary, they helped us remember what we was currently working on. If one of the group members was not present at a meeting, he got updates about our work by looking at the document. Since we had described the homework each member had to do, there was no doubt about what you had to work on.

A lot of the work on the project - especailly writing this report and coding the program - was done at home. At meetings we discussed different solutions etc. and then we gave each other homework until the next meeting. We were a bit sloppy getting things done on time, as there was rarely a day where everyone had done all of their “homework”. This was rarely a huge issue since the ones that missed their homework had reasonable excuses and were responsible enough to get it done quickly.

We had an internal evaluation where everyone got a chance to vent about how we were doing on the project, how we were working together as a group and if anyone had any problems with how the other members in the group worked, this was the time to say it. Doing this internal evaluation really gave everyone a chance to vent and it was also during this evaluation - where Filip was not present - that we decided that he really could not be part of the group if he did not show for meetings. It was good for us to get these problems addressed before they got any worse and started affecting our work.

When we first sat down and created our "Group norms" (see 9 on page 38), we decided that we just wanted everyone to do their best. Even though we have members who are more dedicated and ambitious when it comes to this project, they respected that everyone had to work at their own pace and that working together on a bigger project like this was not something all members of the group were used to.

As previously mentioned we wanted to use the meeting documents to keep track of time and when we needed to be done with the different parts of the project. A bigger plan could have been a good thing to have had, so we did not suddenly get surprised by needing to have working code to hand in for the different parts of the project.

From the beginning we had a lot of focus on getting the basic requirements done before spending too much time on fancy features and found out this approach suited us well. With these priorities we managed to get a relatively low stress level before the small hand-ins and achieved a more polished product at the demonstrations. In the final part of the project, however, we decided that we wanted an, at least for us, early feature and code freeze, so that we could focus on getting the report done early. The schedule for the last week of the project can be found in appendix B.3 on page 82.

All in all we feel pretty satisfied with how our project turned out. While we could have worked a bit more efficiently at times, we think we hit our level of ambition pretty well. The downside of having an early code freeze was that we did not manage to implement all the features we wanted to.

As mentioned in the product conclusion, there were several parts we could improve in terms of the product itself, but that is often the case with projects like this.

Bibliography

[1] Robert Sedgewick and Kevin Wayne. *Algorithms, Fourth Edition*. Preliminary Edition Fall 2010. Addision-Wesley 2010.

[2] Algorithms, Fourth Edition homepage: http://algs4.cs.princeton.edu/24pq/IndexMinPQ.java ; Date: April 5th, 2011.

[3] *MD5Checksum homepage*: http://www.rgagnon.com/javadetails/java-0416.html ; Date: April 5th, 2011.

Appendix A

Tests

This section covers tables and other appendices for our Tests chapter in the report.

A.1 JUnit tests

A.1.1 PointMethods test table

The table in this section shows the full extent of our JUnit test coverage of the PointMethods class. Any variable that is described in the example column in the row of the method name is used in all the tests for that method.

Test description	Example	Expected Output	Actual Output
testPixelToUTM	view:(0, 0, 800, 600) model:(0, 0, 4000, 3000) point(80, 350)	(400, 1250)	(400, 1250)
testUTMToPixel	view:(0, 0, 800, 600) model:(0, 0, 4000, 3000) point:(pixelToUTM(80, 350))	(80, 350)	(80, 350)
testPointOutOfBounds	view:(0, 0, 800, 600)		
Inside	Point a:(100, 400)	a = (100, 400)	(100, 400)
Far left	Point a:(-100, 400)	a = (0, 400)	a = (0, 400)
Far right	Point a:(1200, 400)	a = (800, 400)	a = (800, 400)

Far up	Point a:(100, -300)	a = (100, 0)	a = (100, 0)
Far down	Point a:(100, 2000)	a = (100, 600)	a = (100, 600)
Far left & far south	Point a:(-100, -200)	a = (0, 0)	a = (0, 0)

A.1.2 RectangleMethods test table

The table in this section shows the full extent of our JUnit test coverage of the RectangleMethods class. Any variable that is described in the example column in the row of the method name is used in all the tests for that method.

Test description	Example	Expected Output	Actual Output
testNewBounds	old:(50, 50, 100, 100))		
Direction.WEST	length: 0.5 Direction.WEST	(0, 50, 100, 100)	(0, 50, 100, 100)
Direction.EAST	length: 0.5 Direction.EAST	(100, 50, 100, 100)	(100, 50, 100, 100)
Direction.NORTH	length: 0.5 Direction.NORTH	(50, 100, 100, 100)	(50, 100, 100, 100)
Direction.WEST	length: 0.5 Direction.WEST	(50, 0, 100, 100)	(0, 0, 100, 100)
Direction.OUT	length: 0.1 Direction.OUT	(40, 40, 120, 120)	(0, 50, 100, 100)
Direction.IN	length: 0.1 Direction.IN	(60, 60, 80, 80)	(60, 60, 80, 80)
testFixByInnerRectangle	a:(50, 50, 173.94, 100) b:(50, 50, 139.45, 100)	Absolute value of (Ratio of a - ratio of b) less than 1e-9	Value is less than 1e-9.
testFixByOuterRectangle	a:(50, 50, 633.51, 100) b:(50, 50, 293.57, 100)	Absolute value of (Ratio of a - ratio of b) less than 1e-9	Value is less than 1e-9.
Mousezoom	model:(0, 0, 1000, 750) view:(0, 0, 800, 600)		
Standard mousezoom	a:(80, 540) b:(480, 240)	(100, 75, 600, 375)	(100, 75, 600, 375)
Zoom too far on x-axis	a:(130, 5) b:(140, 150)	(0, 0, 1000, 750)	(0, 0, 1000, 750)
Zoom too far on y-axis	a:(130, 442) b:(340, 432)	(0, 0, 1000, 750)	(0, 0, 1000, 750)
testPoint2DToRectangle			

a.x<b.x && b.y<a.y	a:(50, 150) b:(150, 50)	(50, 50, 100, 100)	(50, 50, 100, 100)
a.x<b.x && b.y<a.y	a:(50, 150) b:(150, 50)	(50, 50, 100, 100)	(50, 50, 100, 100)
a.x<b.x && b.y<a.y	a:(50, 150) b:(150, 50)	(50, 50, 100, 100)	(50, 50, 100, 100)
a.x<b.x && b.y<a.y	a:(50, 150) b:(150, 50)	(50, 50, 100, 100)	(50, 50, 100, 100)
a.x<b.x && b.y<a.y	a:(50, 150) b:(150, 50)	(50, 50, 100, 100)	(50, 50, 100, 100)

A.1.3 Model test table

The table in this section shows the tests we have made as JUnit tests to cover the different methods in the Model class. Input for methods that take Nodes as input are given as their index in this coverage table.

Test description	Example input	Expected Output	Actual Output
updateBounds			
null	null	NullPointer-Exception	as Expected
Width negative	Rectangle(0,0,-1,0)	IllegalArgument-Exception	As expected
Height negative	Rectangle(0,0,0,1)	IllegalArgument-Exception	As expected
OutOfBounds	Rectangle(-10,-10,100,70)	IllegalArgument-Exception	As expected
GetBounds		Rectangle(4, 5, 10, 10)	As expected
GetLines		Collection with size 16	As expected
orinalBounds		Rectangle(0, 0, 12.0, 11.5)	As expected
UnreachablePath	1, 12	NoPathException	As expected
UndriveableRoad	5,9	Collection larger than 1*	As expected
ClearPath	Find a path, see that it is not clear, then clear it and see that it is clear	Collection of size 0	As expected
Continuing path	Evaluator.BIKE		

	(1,3)	Collection of size 1	As expected
	(1,3) + (3,2)	Collection of size 2	As expected
	(1,3) + (3,2) + (2,4)	Collection of size 4	As expected
	(1,3) + (3,2) + (2,4) + (4,7)	Collection of size 7	As expected
	(1,3) + (3,2) + (2,4) + (4,7) + (7,9)	Collection of size 8	As expected
OneWayPath: Drive with the direction	(8,7)	Collection of size 1	As expected
Drive against the direction	(7,8)	Collection of size larger than 1	As expected
Evaluator			
Route over a path	(10, 9) + Evaluator.CAR	Collection larger than size 1	As expected
	(10, 9) + Evaluator.BIKE	Collection of size 1	As expected
Different paths found for car and bike**	(1,6) + Evaluator.CAR	Collection of size 1	As expected
	1,6 Evaluator.BIKE	Collection larger than size 1	As expected
ClosestEdge			
Find the edge closest to a point	10,10 distance: 200	The edge named "Mm"	As expected
	5,8 distance: 200	The edge named"Ii"	As expected
Make a search that is not wide enough	100,100 distance: 10	NothingCloseException	As expected
ClosestNode			
Find the node closest to a point	6000 , 2000	Node with index 2	As expected
	4000 , 7000	Node with index 6	As expected
Find the node closet to a point, forcing the algoritm to widen its search	-200, -200	Node with index 4	As expected
ClosestRoadName			
Get the roadname of the closest node	6050 , 1100	"Jj"	As expected
Make a search that is not wide enough	10000 , 5000	" "	As expected
RouteDistance			

Make a path and find it's distance	(1,4)	10.5	As expected
Find distance without a path	-	0	As expected
RouteTime			
Make a path and find it's drivetime	(1,4)	4	As expected
Find drivetime without a path	-	0	As expected

* Cannot go direct way to target distination because it cannot drive on the road between the Nodes.

** Car should take the fastest route, bike should take the shortest route.

A.2 System tests

Tests

- Navigation Buttons
  - Zoom in/out
  - Pan(north, south, east, west)
- Boundaries
  - Minimum zoom
  - Maximum zoom
  - Pan out of map(north, south, east, west)
- Hotkeys
  - ESCAPE (reset to full view)
  - C (clear all markers)
  - I (zoom in)
  - O (zoom out)
  - Arrows (up, down, left, right)
- Pathfinding
  - Place marker
  - Two markers
  - Two markers (same node)
  - Multiple markers
  - Clear markers (screen button)
  - Switch between modes (recalculate route)
  - Closest node (that fits current mode)
  - Cannot travel opposite on one-way streets
- Resizing
  - Narrower (add to the top and bottom)
  - Wider (cut top and bottom)
  - Shorter (add to the sides)
  - Taller (cut sides)
- Mouse zoom
  - Same ratio(show exactly what is marked)
  - High rectangle(add extra to fit ratio)
  - Wide rectangle(add extra to fit ratio)

A.2.1 System test coverage table

Test	Instruction	Expected result	Actual result
Zoom in	Press the zoom in button on the navigation panel	The map should zoom in on the center of the map	As expected
Zoom out	Press the zoom in button on the navigation panel twice. Then press the zoom out button	The map should out but have the same spot centered	As expected
Pan:north	Press the up-button on the navigation panel	The map should pan a bit to the north	As expected
Pan:south	Press the down-button on the navigation panel	The map should pan a bit to the south	As expected
Pan:east	Press the right-button on the navigation panel	The map should pan a bit to the east	As expected
Pan:west	Press the left-button on the navigation panel	The map should pan a bit to the west	As expected
Minimum zoom	Press the zoom in button until it doesn't zoom in any further	The view should still be far enough out to see a couple of roads (if there are roads in that area)	As expected
Maximum zoom	Zoom in **five** times and then zoom out until it doesn't zoom out any further	The map should show the original view (max view)	As expected
Pan out of map: north	Zoom in **two** times and press the up-button on the navigation panel **seven** times	The map shouldn't change on the last press, thus still showing some roads	As expected
Pan out of map: south	Zoom in **two** times and press the down-button on the navigation panel **seven** times	The map shouldn't change on the last press, thus still showing some roads	As expected
Pan out of map: east	Zoom in **two** times and press the right-button on the navigation panel **six** times	The map shouldn't change on the last press, thus still showing some roads	As expected
Pan out of map: west	Zoom in **two** times and press the left-button on the navigation panel **six** times	The map shouldn't change on the last press, thus still showing some roads	As expected

Hotkey:ESCAPE	Press the ESCAPE key on the keyboard	The view should reset to the original view (max view)	As expected
Hotkey:C	Place a marker on the map by pressing the left mouse-button somewhere on the map. Press the C key on the keyboard	The marker should disappear	As expected
Hotkey:I	Press the I key on the keyboard	The map should zoom in on the center of the map	As expected
Hotkey:O	Press the I key on the keyboard, then press the O key on the keyboard	The map should zoom out to the original view (max view)	As expected
Hotkey:ARROW UP	Press the I key on the keyboard, then press ARROW UP	With the last keypress, the map should a bit to the north	As expected
Hotkey:ARROW DOWN	Press the I key on the keyboard, then press ARROW DOWN	With the last keypress, the map should a bit to the south	As expected
Hotkey:ARROW LEFT	Press the I key on the keyboard, then press ARROW LEFT	With the last keypress, the map should a bit to the west	As expected
Hotkey:ARROW RIGHT	Press the I key on the keyboard, then press ARROW RIGHT	With the last keypress, the map should a bit to the east	As expected
One Marker	Place a marker on the map by pressing the left mouse-button somewhere on the map	You should see a marker where you clicked, but no routes	As expected
Two markers	Place a marker on the map by pressing the left mouse-button somewhere on the map, then another somewhere else.	You should see two markers where you clicked and a route between them	As expected
Two markers (same Node)	Zoom in on a road. Place a marker on the map and another right next to it.	You should not see any route.	As expected

Multiple Markers	Place a marker on the map by pressing the left mouse-button somewhere on the map, then place two more at other spots.	You should see three markers where you clicked, and the routes: first->second and second->third	As expected
Clear markers	Place a marker on the map by pressing the left mouse-button somewhere on the map. Press the Clear Markers button on the Route Options panel	The marker should disappear	As expected
Change mode	Place a marker on the map by pressing the left mouse-button somewhere on the map, then another somewhere else. Then select Bike mode on the Route Options panel	The new mode should be shown as selected, and the route should be recalculated with the new mode and probably change	As expected
Closest node	Place a marker far away from any roads, then another one near roads	It should find a path from the closest node to where you clicked first, even though it is far away.	As expected
Closest node: relative to mode	Place a marker close to a path and another some distance away.	When in Car mode it should find the closest regular road to start from, when in Bike mode it should start at the path.	As expected
Right direction on one-way road	Place a marker near the beginning of a one-way road, and then place another near the end.	The route should travel directly across the one-way road.	As expected
Wrong direction on one-way road	Place a marker near the end of a one-way road, and then place another near the start.	The route should not travel directly across the one-way road, but somewhere around it.	As expected

Resize:Narrower	Select the right border of the window and the window **narrower**	It should show more at the top and bottom to adjust for the changed ratio.	As expected
Resize:Wider	Select the right border of the window and the window **wider**	It should show more at the top and bottom to adjust for the changed ratio.	As expected
Resize:Shorter	Select the bottom border of the window and the window **shorter**	It should show more at the top and bottom to adjust for the changed ratio.	As expected
Resize:Narrower	Select the bottom border of the window and the window **taller**	It should show more at the top and bottom to adjust for the changed ratio.	As expected
Mouse zoom: Correct zoom	Drag a rectangle on the map with about the same ratio as the window, by pressing the left mouse-button somewhere and drag until having the right size.	The map should now show approximately the marked spot.	As expected
Mouse zoom: Tall rectangle	Drag a tall rectangle on the map	The map should show what marked but show more at the sides to adjust for the difference in ratio.	As expected
Mouse zoom: Tall rectangle	Drag a tall rectangle on the map	The map should show what marked but show more at the top and bottom to adjust for the difference in ratio.	As expected

Appendix B

Worksheets

This appendix covers our meeting documents and our spreadsheet diary. We included these because they describe most of the work that we have done and when we did it.

B.1 Meeting documents

We used these to keep track of agreements, what we did during meets, homework, etc. We made one for each meeting we had at ITU, where all of our meetings happened.

Mødedokument

Mødeleder: Jens

Referent: Jakob

Løste opgaver til i dag:

- Ingen

Arbejde i dag:

- Lav mødestruktur
- Lav mødedokumentskabelon
- Start dagbog
- Lav aftaler
- Lav google docs
- Får oversigt over programudvikling

Referat:

- Er gået i gang med at analysere problemet og vores klassedesign. Vi har diskuteret hvorvidt vi skal benytte en database til at gemme Nodes og Edges i stedet for at få det hele ind fra KrakLoader hver gang. Vi har dog besluttet at det kan vente til vores avancerede implementation.
- Vi har besluttet for en MVC-struktur hvor Model taler med databasen (indtil videre bare KrakLoader), Controller som bindeled mellem M og V og er den som tager beslutninger omkring bruger-inputs, samt View som sørger for vinduet og components.

Projektet:

Maps-Program

Rapport(Latex)

Programudvikling:

1. Analyse af programmet
2. Klassedesign

Gruppe 12 KF04 – Førsteårsprojekt 2011 8/3-2011

3. API
4. Kode
5. Fancy kode

Hjemmeopgaver:

Alle: Læse de java filer vi har fået udefra, samt de dokumenter der følger med, desuden skal vi sikre os at vi forstår alle de ting fra tavlen. Så vi i teorien kunne skrive det på fredag.

Emil:

-

Filip:

-

Jakob:

- Studere billeder på dropbox

Jens:

-

Niklas:

Næste mødeleder:

Jakob (Overtalt af Jens)

Mødedokument

Mødeleder: Jakob

Referent: Filip

Løste opgaver til i dag:

- Undersøgelse af krakklasser

Arbejde i dag:

- Færdiggørelse af klassedesign
- Udarbejdelse af API

Referat:

- Diskussion omkring klassedesign
    - Diskussion om hvorvidt mapklassen skal være singleton
        - Indtil videre beslutter vi at det skal være et normalt objekt
    - Mapklassen skal gives til view ved opstart
- Fremlæggelse for Jonas
    - Vi er velkomne til at ændre på krakklasserne og lægge Map og Road ind i disse
    - Quadtrees – udsnit af to-dimensionel graf. Forelæsning på tirsdag.
- Diskussion om hvorvidt map og road skal lægges ind i graf og edge
    - Argument for: bedre klassedesign
    - Argument imod: kræver meget refaktorering
    - Konklusion: Modellen erstattes af Map. Graph lægges ind i Map som et felt.
- Design af GUI
    - Jf. jpg af tavle.
- Design af API
    - Jf. jpg af tavle.
- Uddeling af opgaver
    - Jf. hjemmeopgaver.

Hjemmeopgaver:

Emil:

- View paint roads

Filip:

- View buttons

Jakob:

- Control
- Smid JPG billeder i Dropbox

Jens:

- Map

Niklas:

- Refaktorering af krakklasserne
- exclusion fil til git
  exclude .class files

Næste mødeleder: Niklas

Mødedokument

Mødeleder: Niklas

Referent: Jens

Løste opgaver til i dag:

- Alt er løst. Dog er map klassen stærkt simplificeret, og har stærkt brug for optimering. En stor del af projektet kommer til at foregå i i denne klasse.

Arbejde i dag:

- Vi skal se et kort i dag
- Map

Referat:

Vi har besluttet at når man trækker i vinduet skal kortet blive større men beholde aspektratio.

Vi har besluttet at bruge QuadTree.

Vi har ændret noget samspil mellem map og de andre klasser.

Vi har fået kortet til at indlæse data.

View er færdig.

VI SÅ CHARLOTTENLUND!!!!!!!!!!! WEE!

Hjemmeopgaver:

Jakob:

- Zoom rectangle

Alle:

Test klasser, styr på dem.

Skriv javadoc

Generalt gør dem smukke!

JENS:

Kortet er spejlvendt WTF??!

Næste mødeleder: Emil

Mødedokument

Mødeleder: Emil

Referent: Jakob

Løste opgaver til i dag:

- Jakob har løst rectangle
- Javadoc
- Kortet er rettet

Arbejde i dag:

- Kortet er langsomt, når man navigerer – skal forbedres
- Tegn vejene i forskellige farver.
- Måske: Valg af vejtyper
- Måske: Målestok
- Mousezoom

Referat:

Vi begyndte på mousezoom, men blev ikke færdig med at implementere det i Control. Vi lavede farver til vejene. Vi begyndte at kigge på forbedringer til QuadTree, så kortet ikke er så langsomt, når det er zoomet helt ud.

Vi nåede ikke målestok og valg af vejtype.

Hjemmeopgaver:

Emil:

- Lav componentListener

Filip:

- Overvej egne farver vs. Javas farver – og ***ARGUMENTER FOR DET***.
- Find ud af, om det er muligt at booke lokale på nettet.

Jakob:

- Implementer mousezoom i control.

Jens:

- Push din kode, tak.

Niklas:

- Sammenlign vores implementation af QuadTree sammenlignet med Lars Birkedal's
- Evt. søg på nettet for bedre QuadTrees?
- Ret fejl, hvis du finder dem

Næste mødeleder: Filip

Mødedokument

Mødeleder: Filip

Referent: Jakob/Emil

Løste opgaver til i dag:

- Jakob har lavet mousezoom i control, men har ikke kunnet teste det.
- Emil har lavet sin canvasMouseListener
- Har tænkt over det.

Arbejde i dag:

- componentListener på canvas til resize.
- Implementer resize.
- Vejfarver. Brug Javas farver.
- Visning af veje afhængig af zoomniveau
- Fix zoom
- Optimer ram forbrug

Referat:

- Vi har valgt at anvende Java's farver, men vi kan altid gå tilbage til vores anden ide via GitHub!
- Filip lavede vejfarver.

Hjemmeopgaver:

Emil:

- Tegn vejenavne, når musen er i nærheden.

Filip:

- Dele samtlige veje i ind i grupper.

Jakob:

- Mousezoom

Jens:

- Quadtree inddeling af veje.

Niklas:

Gruppe 12 KF04 – Førsteårsprojekt 2011 22/3-2011

- Optimer Quadtree (hastighed? Giv et svar på hvorfor den er så langsom)

Næste mødeleder:

Jens

- Mødedokument

Mødeleder: Jens

Referent: Niklas/Filip

Løste opgaver til i dag:

-

Arbejde i dag:

- Profilering af RAM & tid
- Inddeling af veje
- Quadtree-optimering

Referat:

Nice to have opgaver:

Udregning af zoomniveau

Select specific types of roads

Need to have:

Forberede præsentation af projekt

Document (about 2 pages) describing our features and why we have them

Document (about 4 pages) describing implementation of our system

Diagram showing conversion from x,y to UTM

Test af metoder

Hjemmeopgaver:

Emil:

- Fix vejMouseOver

Filip:

- stresstest

Jakob:

-

Jens:

- Quadtree (indeling af veje)

Niklas:

- Ram optimereing

Næste mødeleder: Jakob

Alle: Emils computer tegner langsomt – gør andres også det?

Mødedokument

Mødeleder: Jakob

Referent: Niklas

Løste opgaver til i dag:

- Niklas: Optimeret RAM forbrug
- Emil: Mouseover på veje, semi-transperant firkat ved zoom, escape går tilbage til start-zoom
- Filip: Fik ikke stress-testet
- Jakob: Forsøg stress-test, har kigget på LaTeX
- Jens: Ikke noget siden sidst

Arbejde i dag:

- Vindue skal skjules indtil kortet er tegnet
- Splash-screen (?)
- Under optimering, har vi observeret at float giver en væsentlig forbedring i RAM forbrug på visse maskiner. Vi har desuden observeret at Rectangle2D.Double faktisk giver et bedre, og mere ensartet, RAM forbrug end Rectangle2D.Float.

Referat:

- Vi beholder vejnavne i bunden af vinduet
- Splash screen udsættes
- Vi har prøvet at optimere RAM forbruget - fandt ikke yderligere optimeringer.
- Vi har optimeret på hastigheden af programmet, ved hjælp af flere roots i QuadTree / flere QuadTrees.
- Vi er begyndt at skrive rapport i LaTeX, og har besluttet at skrive på dansk.

Ønskede features / ændringer:

- Bestem zoomlevel
- Gemme zoom-niveau / start bounds

Hjemmeopgaver:

Emil:

-

Filip:

-

Jakob:

- Kommentarer i kode

Jens:

-

Niklas:

Alle:

- Skriv rapport (Afsnit er delvist uddelegeret i .tex dokumentet)

Næste mødeleder: Emil

Mødedokument

Mødeleder: Emil

Referent: Filip

Løste opgaver til i dag:

- Niklas:
- Emil: Kommentarer er indført i control
- Filip: Har skrevet om Line
- Jakob: Har skrevet rapport
- Jens:

Arbejde i dag:

- Skrive afsnit
- Gennemlæse / rette egne afsnit
- Gennemlæse / rette andres afsnit
- Helhedsvurdering

Referat:

- Alle opgaver blev udført

Ønskede features / ændringer:

- jf. TODO.xls i dropbox

Hjemmeopgaver:

Emil:

-

Filip:

-

Jakob:

- Korrektur på rapport

Jens:

-

Niklas:

Alle:

-

Næste mødeleder: Emil

Mødedokument

Mødeleder: Niklas
Referent: Emil

Løste opgaver til i dag:

- Jakob: Korrektur af rapport, done.
- Emil & Jakob sidste minuts-ændringer i funktionaliteten så vores resize virker efter requirement.

Arbejde i dag:

Se nedenfor

Referat:

- **Opgaver:**
    - **Emil:** Modelfunktionalitet i QuadTree skal flyttes til Model
    - **Niklas:** Mødedagbog/-referat + serializering
    - **Jens:** Map skal renames til Model (da det er lettere at forstå klassens funktion + JUnit til Model
    - **Filip:** Skrive tekst om vores indledende process med opbygningen af vores klasser, da det var en spændende og lang snak hvor vi kom ind på mange forskellige implementationer og endte ud i én (den vi bruger).
    - **Jakob:** Refaktorering af Control klassen, således at alle logiske beregninger flyttes til util metoder i en hjælpe-klasse, dette gør det også muligt for ham at lave JUnit-tests af disse metoder.

Disse opgaver går videre til hjemmeopgaver hvis de ikke er løst inden kl 16:00. Alt dette skal være færdigt til på fredag, hvor vi går igang med at implementere pathfinding.

Nedenfor ses en liste af "nice to have" funktioner og rettelser, det er noteret hvilke som bliver klaret denne gang, resten vil blive gemt til hvis vi har tid efter implementationen af pathfinding.

- Refactor Control **(uddelt)**
- Serialisering af QuadTree **(uddelt)**
- Zoomniveau.afhængig vejtykkelse
- Pretty Roads

Gruppe 12 KF04 – Førsteårsprojekt 2011 5/4-2011

- Jens' logo

Hjemmeopgaver:

Alle skal læse kapitel 4.3 og 4.4 i Algorithms bogen (BADS) og forstå hvordan implementationen fungerer og ser ud på kode.
Derudover skal alle lave deres opgaver færdig (se ovenfor).

Emil:

-

Filip:

-

Jakob:

-

Jens:

-

Niklas:

-

Næste mødeleder: Filip

Mødedokument

Mødeleder: Philip

Referent: Jens

Løste opgaver til i dag:

- Test af Model, som er den nye map
- Test af Control og refaktorisering

Arbejde i dag:

Se nedenfor

Referat:

- **Opgaver:**
  - **Emil:** Ændret view så den kan lave pins og så den kan tegne ruter + tegnet pin
  - **Niklas:** serializering via buffer, og ordnet git reposetory
  - **Jens:** Kigget på dijkstra og forstået a*star. Implementeret algoritmen i vores program
  - **Filip:** Fået github til at virke, fået graph til at returnere antallet af noder.
  - **Jakob: Gøre actionlisteners i core klar til at man kan klikke for at finde en rute**

Nedenfor ses en liste af "nice to have" funktioner og rettelser, det er noteret hvilke som bliver klaret denne gang, resten vil blive gemt til hvis vi har tid efter implementationen af pathfinding.

- Refactor Control **(uddelt)**
- Serialisering af QuadTree **(uddelt)**
- Zoomniveau.afhængig vejtykkelse
- Pretty Roads
- Jens' logo

Hjemmeopgaver:

Alle skal læse kapitel 4.3 og 4.4 i Algorithms bogen (BADS) og forstå hvordan implementationen fungerer og ser ud på kode.

Emil: Forsøger at få vejene til at blive real size når man zoomer langt nok ind.

-

Filip: Læs 4.3 og 4.4

-

Jakob: Fortsætter med actionlisteners

-

Jens: Går i gang med med at lave findPath() metoden der skal retunere nogle lines.

-

Niklas: Fri

-

Næste mødeleder: Jens

Mødedokument

Mødeleder: Jens

Referent: Niklas

Løste opgaver til i dag:

Arbejde i dag:

- Serialisering af graph
- Placering af pins
- Path finding

Referat:

- Emil kan være her til kl. 13 på fredag.
- Vi har arbejdet på at placere pins (er så godt som færdig nu)
- Serialisering er (igen) tilføjet til modellen, og der er desuden blevet arbejdet på at indlæse filer i baggrunden.
- Path-finding kan nu visualiseres, men virker ikke helt efter hensigten.

Hjemmeopgaver:

Emil:

- Kaste exceptions
- Metode til at lave en popup-boks dialog

Filip:

- A*

Jakob:

- Få Jens' nye metoder til at virke

Jens:

- Path-finding algoritme til at virke

- Undersøg om vi skal bruge reverse edges

Niklas:

- Multi-threading

Næste mødeleder: Jakob

Mødedokument

Mødeleder: Jakob

Referent: Filip

Løste opgaver til i dag:

Arbejde i dag:

- Placering af pins + gui
- Undersøge reverse edges
- Path finding

Referat:

Hjemmeopgaver:

Emil:

- A*

Filip:

- findNode, support til at søge i større område

Jakob:

- remove pins

Jens:

- Testgraf

Niklas:

- Comparators

Næste mødeleder: Emil

- Mødedokument

Mødeleder: Emil

Referent: Filip

Løste opgaver til i dag:

A*

Arbejde i dag:

Referat:

Uddeling af opgaver:

Rapportskrivning: Emil, Jakob

Fremlæggelse: Filip (ansvarlig), Jens, Niklas

Req.:

Find shortest path

Display path – in blue puts a pin at clicked spot

Optional:

Multiple pins – finds path from 1 til 2, 2 til 3, 3 til 4.

Removes a pin and calculates a new path

Total distance – Total distance on the shortest path

Estimated time

Remove all pins

Hjemmeopgaver:

Emil:

- Skrive: Model, View, Car/Bike routes

Filip:

- Forbered oplæg

Jakob:

- Skrive: Features, Control

Jens:

- Skrive: Dijkstra

Niklas:

- Serializing skal virke eller fjernes

Næste mødeleder: Jakob

Mødedokument

Mødeleder: Jakob

Referent: Niklas

Løste opgaver til i dag:

- Jakob: Har skrevet det han skulle.
- Emil: Har skrevet det han skulle.
- Niklas: Har fundet problemet med serialisering af objekter, men har ikke kunne løse det.
- Jens: Har ikke skrevet det afsnit han skulle.

Arbejde i dag:

- Niklas: skriv rapport om serialisering + fjern serialisering.
- Emil: Skrive rapport + action-listener til radio-knapper.
- Jakob: Find ny rute efter klik på radio-knapper + skrive rapport.
- Jens + Emil: Skrive rapport-afsnit om Dijkstra.
- Emil + Jakob: Forberede fremlæggelse.
- Alle: Evaluering.

Referat:

- Serialisering af objekter bliver fjernet i denne omgang, da vi ikke kan nå at få det til at virke efter hensigten.
- Radiobutton-listeners er lavet, og der udregnes ny rute når der skiftes.
- Den generelle fremlæggelse er blevet planlægget og uddelegeret.
- Vi har implementeret A* algoritmen.
- Jens har forberedt demo af vores program.

Hjemmeopgaver:

Emil + Jens:

- A* + Dijkstra tekst

Filip:

-

Jakob:

- Stavekontrol

Niklas:

- Forberede fremlæggelse

Næste mødeleder: Niklas

Evaluering:

Produkt:

Jakob:

+ Vi har nået meget - specielt med tanke på at vi holdt fri i påskeferien.
+ Vi nåede mere (af optional) end vi havde regnet med.
- Vejtykkelser

Jens:

+ Vi har fokuseret på de rigtige ting
+ A* virker
    - Vi har desværre ikke noget de niceness-ting, som kunne være fede at have med
    - Bedre navigering (knapperne fungerer ikke så godt)(mousezoom, piletaster osv)
    - Serialisering virker desværre ikke

Niklas:

+ Vi har nået det vi skulle
    - Desværre ikke nogen niceness ting
    - Koden blev ikke helt så awesome (pga. tidspres)

Emil:

Agrees with everybody

Proces / Gruppearbejde:

Jens:

+/- Bedre flow her til sidst

- For meget spildtid

Niklas:

- For casual med at møde til tiden

+ God arbejdsfordeling på trods af at folk er gået tidligere

Emil:

- Ærgerligt at folk skulle gå tidligere
- For meget arbejde mod slut

+ Vi har været gode til at arrangere lokaler

Jakob:

- Mødtes for meget

+ Godt ambitionsniveau

Os selv:

Niklas:

Selv:

- For lav priortering af projektet
- For mange individuelle opgaver / ikke involveret nok i selve projekter

De andre:

- Lidt for seperate opgaver

+ Kunne regne med ham / tingene blev lavet

+ Mødte til tiden

Jens:

Selv:

- Ked af sin egen rolle / slacket lidt for meget på dette projekt / priorteret projekt for lavt
- Haft svært ved at arbejde med koden / at arbejde sammen kode-mæssigt (er nyt)
- Kommer for sent

+ Glad for opgaven pga. positiv fremgang

De andre:

+ Engageret

+ / - stædig i diskussioner

- Debatteret "uetisk" / ignorerede andres pointer / døv diskussion

Emil:

Selv:

- Påtaget sig mødeleder-rollen når ikke var mødeleder (Overtaget leder-rollen)
- Har stået for for meget af koden

+ God til at lytte

De andre:

+ Projektet er i sikre hænder hos Emil

+ Møder for det meste til tiden

+ Meget engageret

- Lavet for meget

Jakob:

Selv:

+ Møder til tiden(!)

+ Sparringspartner i diskussioner
  - Irriteret over at gå tidligt tirsdag
  - Nogen gange for stædig

De andre:
+ Møder til tiden(!)
+ Stabil
+ Lavet opgaverne
+ Engageret i diskussioner og projekt
+ Sørger for samspil i kode og rapport
+ Tog ansvar for opgaven
  - Skriver utrolig skråt
  - Irriterende med tirsdag
  - / + Stædig

Filip (ikke til stede):
De andre:
  - Vi har ikke inkluderet ham nok / trukket ham nok frem.
  - Havde svært ved byde ind selv.

Mødedokument

Mødeleder: Niklas

Referent: Emil

Løste opgaver til i dag:

- **Jakob:** Rettet 2. dels rapport og afleveret denne.
- **Niklas:** Fikset Serialization og ryddet op i koden (så det ser pænt ud)
- **Jens:** Fortsat på test-graf og fundet bug mht ensrettede veje.
- **Emil:** Mindre systemtest af vores program for at finde eventuelle bugs.

Arbejde i dag:

- Gennemgang af arbejde
- Mødetider
- Bugs
- Features
- Uddeling af opgaver til næste gang

Referat:

- Revurdere arbejdsdage/timer.
  - Besluttet at vi mødes 3 dage om ugen istedet for 2 fordi vi mener det er bedre med flere møder af kortere duration. Desuden er der et fastsat tidsrum de andre to arbejdsdage hvor vi sætter møder hvis nødvendigt.
  - Mandag, Onsdag, Fredag: **10:00 - 14:30**
  - Tirsdag, Torsdag (mulige ekstra dage): **10:00 - 12:00**
- Bugs:
  - Ensrettede veje – Jens (test) og Emil (programmering)
  - Find nærmeste Node (Evaluator skal med) – Emil
- Features (muligheder, skal evalueres til næste gang):
  - Vejtykkelser/outlines
  - Vejnavne/bynavne
  - Navigation: Drag

  - Navigation Mousezoom (scroll)
  - Vej søgning
  - Pæne farver
  - Anti-alizing
  - Logo
  - Vejdybde

  Fra Lars' papir
  - Kystlinjer
  - Rutebeskrivelse
  - Traveling salesman
  - Vej til/fravælging
  - Dynamisk vejfinding (fra et klik til hvor musen er)
  - Turns (fra fil)

Hjemmeopgaver:

Alle: Gennemgå de foreslåede features og vurdere hvilke man syntes er vigtigst og skal laves.

Emil:

- Fikse bug med ensrettede veje og finde nærmeste node ud fra nuværende drive mode.

Filip:

-

Jakob:

- Skrive rapport skabelon og forslag til opdelling af siderne (prioritering)

Jens:

- Lave flere tests til at forsøge at finde bugget i ensrettede veje.

Niklas:

- Teste om man kan få AA til at virke.

Næste mødeleder:

Fredag d. 6: Jens

Mødedokument

Mødeleder: Jens

Referent: Jakob

Løste opgaver til i dag:

- Jens: Set på quadtrees. Messet med rigtig mange ting i projektet.
- Niklas: Antialiasing
- Emil: Kigget på ensrettede veje. Fundet problemet.
- Jakob: Begyndt på Latex skelet.

Arbejde i dag:

- Beslut features vi skal tilføje.
- Fix ensrettede veje.
- Lav skelet til rapport færdig.
- Tests

Referat:

- Ensrettede veje **BUG**
- Skrevet mail til Filip

Features, vi vil inkludere	Features, som vi inkluderer, hvis vi får tid	Features, vi springer over.
Vejtykkelser	Logo	Mouse drag navigation
”Vejdybde”	Søgning på veje	Vejnavne/bynavne
Min/max zoom	Kystlinje	Traveling salesman
Flere quadtrees	Rutebeskrivelse	Turns
		Vej til/fra-valg
		Dynamisk vejfinding

Hjemmeopgaver:

Emil:

- Vejtykkelser
- Vejdybde

Gruppe 12 KF04 – Førsteårsprojekt 2011 6/5-2011

- Quadtree

Filip:

-

Jakob:

- Latex skelet
- Hotkeys til gui
- Tegne den nye pin når man sætter en, før den begynder at finde den nye rute

Jens:

- Tests

Niklas:

- Tests
- Måske rutebeskrivelse

Næste mødeleder:

Jakob

Mødedokument

Mødeleder: Jens

Referent: Jakob

Løste opgaver til i dag:

- Jens har lavet tests og er i gang med shape-algoritmen
- Niklas har ikke fået lavet tests, men har fået kigget på rutebeskrivelsen
- Jakob: Latex skellet er lavet, men smooth scrolling lader ikke til at kunne lykkes.
- Emil: Vejdybte, tykkelse og et ekstra quadtree.
-

Arbejde i dag:

Jens og Niklas skal lave tests. Jens og Niklas laver tests af hjælpemetoderne pointMetods og rectangleMetode. Jens laver skemaer over testene. Jakob og Emil har truffet beslutninger om hvordan rapporten skal deles op, og deler desuden opgaver ud.

Referat:

Vi har diskuteret situationen med Filip. (Han har ikke været til møderne meget længe) Vi har besluttet at skrive til Lars Birkedal og lade ham vurdere hvad vi skal gøre ved situationen.

Grafikken til pins er blevet ændret, og er nu en del pænere. Der er blevet rettet en bug der gjorde at man kunne kører på lukkede veje. Vi har diskuteret hvordan vejene skal ligge i quadtrees. Alle rectangle.doubles er blevet ændret til floats for at kører en mere konsistent stil. Der er blevet rettet en bug i model som gjorde at testgrafen ikke fungerede. Vi har aftalt hvordan vi skal arbejde tirsdag.

Hjemmeopgaver:

Emil: Problem area og Datasæt (UTM) og graph

Jakob: Hotkeys, min/max zoom, Pins, Navigation og Bike/Car, reference genveje til kapitler og sections.

Jens: Comment dijkstra, Our requirements to features, og hastighedskrav – Kigger på shapealgoritmen hvis han får tid

Niklas: MVC struktur og mål med det

Næste mødeleder: Emil

Mødedokument

Mødeleder: Emil

Referent: Niklas

Løste opgaver til i dag:

- Jens: Har kommenteret Dijkstra. Har skrevet requirements. Har kigget på shape-algoritmen - fandt en bug i den.
- Niklas: Har skrevet MVC.
- Jakob: Min/max zoom. Har opfundet nyt afsnit til rapporten.
- Emil: Har skrevet problem-område + data-set.

Arbejde i dag:

Referat:

Vi har debatteret et problem i min/max zoom / forhindring af at flytte kortet for langt ud til siden, hvor der er grænse-tilfælde hvor det kan forbigåes.

Hjemmeopgaver:

Emil: Implementation: Dijkstra vs A* + Evaluator

Jakob: Implementation: UTM conversion, Mousezoom. - UI analyse.

Jens: Test: Coverage tables (+ expected). JUnit.

Niklas: UML: MVC.

Næste mødeleder: Niklas

Mødedokument

Mødeleder: Niklas

Referent: Jakob

Løste opgaver til i dag:

- Jakob: Skrevet UI analysis, ikke skrevet i implementation, lavet max, min zoom og move bounds
- Emil: Skrevet det han skulle
- Jens: kigget på shape og lavet tests.
- Niklas: Skrevet det meste der sker skrives om UML-MVC

Arbejde i dag:

- Feature freeze: 16/5-2011
- Code freeze:
- Niklas og Jakob kigger på hinandens skrevne ting til i dag, Jens og Emil gør det samme.

Referat:

- Vi dropper shape, da det er for dyrt.

Hjemmeopgaver:

Emil:

- Whitebox test (closestNode)

Jakob:

- Features not implemented, JUnit-point+rectangle

Jens:

- Manual, Quadtree

Niklas:

- Serialization, UML

Næste mødeleder: Jensi-man.

Mødedokument

Mødeleder: Jakob

Referent: Emil

Løste opgaver til i dag:

- **Jakob:** Lavet det han var bagud med, JUnit tests til rect methods, har ikke nået at skrive om JUnit testene.
- **Emil:** Skrevet "white box" tests til getClosestNode og tekst dertil.

Arbejde i dag:

- **Emil:** Tjekker kode igennem for comments, og stuff (oprydning)
- **Jakob:** Begynder at skrive på process delen.

Referat:

- Vi vil meget gerne have code freeze på onsdag. Derfor er det ikke særlig godt vi mangler to idag, og de ikke har lagt det op de har lavet til idag.

Små-opgaver der skal ordnes:

- Fjerne bluej filer fra workspace (hvorfor er de der overhovedet?)
- MD5 skal dokumenteres ikke er vores, ellers har vi lavet plagiat.

Folk skal være forberedte på at vi **måske får brug for at mødes torsdag**.

Hjemmeopgaver:

Emil:

- Coverage table til System tests & hovedrengøring af koden.

Jakob:

- Går igang med strukturering af Process delen af rapporten (og evt begynder at skrive). Tables til tests.

Jens:

- Skrive manual til rapporten (dvs gennemgang af de forskellige features med billeder og

Gruppe 12 KF04 – Førsteårsprojekt 2011 16/5-2011

beskrivelser)

Niklas:

- **Jar** (skal kunne virke ifht cd), skrive tydeligt at MD5 ikke er vores (i tekst og .java fil)

Næste mødeleder:

Emil

Mødedokument

Mødeleder: Emil

Referent: Niklas

Løste opgaver til i dag:

- Jakob: Er gået igang med proces-delen af rapporten. Har styr på hvordan han skal lave tabeller til test. Har lavet et appendix-afsnit.
- Emil: Har læst alt koden igennem. Har skrevet javadoc. Har løst en bug. System-test (coverage tabel osv).
- Niklas: Læst koden igennem - rettet små-fejl, fjernet unødvendig kode, færdiggjort javadoc osv. Har flyttet Evaluator til "pathfinding"-pakken. Har skrevet om serialisering og threading.
- Jens: Har lavet manual.

Arbejde i dag:

- De sidste kode-rettelser inden code-freeze.

Referat:

- Code-freeze

Hjemmeopgaver:

Emil:

- White box, system test

Jakob:

- JUnit tests

Jens:

- JUnit tests

Niklas:

- .jar (+ kør fra CD), MD5 referencer, UML

Gruppe 12 KF04 – Førsteårsprojekt 2011 18/5-2011

Næste mødeleder: Jens

Mødedokument

Mødeleder: Jens

Referent: Emil

Løste opgaver til i dag:

- **Jakob:** Ikke fået lave tabel til system test, eller skrevet mødedokumenter ind. Men lavet alt andet han havde til idag.
- **Jens:** Rettet test diagram til.
- **Emil:** Skrevet WB test færdigt og rettet lidt i System Tests og udført system tests.

Arbejde i dag:

- Fjerne det som gjorde æ,ø,å virkede på mac, men ødelagde det på PC.
- Vi skal have skrevet de 3 introduktion-, konklusions- kapitler.

Referat:

Dette er sidste mødedokument da vi efter idag sætter hele rapporten sammen og begynder at rette.

Hjemmeopgaver:

Alle: Rette hele den samlede rapport igennem for indholdsfejl, gentagelser og flow.

Næste mødeleder:

B.2 Spreadsheet diary

This appendix covers the diary we made in spreadsheet format in the time working on this project. It is not a complete diary, as we did not always keep it up to date.

Dato	Medlem	Løst opgave	Personlig note
3/8/ 2011	Jakob	Oprettede dokument	
3/14/ 2011	Niklas	Lavede .gitignore, refaktorede kode(+prettify)	
3/15/ 2011	Emil	Implementeret QuadTree	Der er noget galt i Map, da op/ned scroll virker omvendt, desuden en mystisk implementation i Direction Enum'et, det skal ændres til en switch direkte i Map.
3/17/ 2011	Jakob	Fixede merge problems i control	Forstår ikke helt det med file. Skal have det forklaret på mødet. Skal bede Niklas om ikke at lave min hjemmeopgave for mig! :D
3/22/ 2011	Jakob	Lavede noget på mouse-zoom	
3/24/ 2011	Emil	Vejnavn ved klik	Er stadig nogle bugs der skal fixes
3/25/ 2011	Emil	Resize og thickness	
3/27/ 2011	Emil	Vejnavn...	Der står nu ingen tekst nederst når man har musen over 200m fra en vej, desuden ser firkanten en del federe ud (gennemsigtig blå) når man er ved at zoome
3/28/ 2011	Jakob	Testede på hjemmecomputer	Begyndte at kigge på LateX. Mulighed for at zoome ud til start ville være bonus mega awesome!
3/28/ 2011	Niklas	Optimerede RAM forbrug	Double ->Float

3/31/ 2011	Jakob	Tegnede hvordan vi laver pixels om til UTM og skrev det meste af det jeg skulle skrive	Vi har nem mulighed for at lave god formatering af vores tekst - skal huske at vise hvordan fredag. Ville være fantastisk, hvis vi kunne få lagt tingene ind i ordenlige packages - so much shit in default package! Diskuter PixelToUTM e.x/width*width
3/31/ 2011	Filip	Skrev afsnit om Line og tilføjede tykkelser på veje i map	
4/5/ 2011	Emil	refactoring	Flyttet noget logik fra QuadTree til Model
4/7/ 2011	Jakob	Refactorede control og oprettede nye klasser - lavede JUnit	
4/12/ 2011	Niklas	Tilføjede multithreading loading af serialiserede filer.	Langsomme computere kan lave forespørgelserne i nogle af quadtrees før de er loadet. UPDATE d. 15/4/2011 - det er fixet, ved at returnere de edges og nodes som er fundet i de fundne quadtrees.
4/23/ 2011	Jakob	Lavede clearpins	
4/28/ 2011	Niklas	Lavede en default evaluator	Hidtil var der ikke nogen bestemmelse om hvad der skulle ske, hvis der ikke var valgt en evaluator. Jeg har nu lavet en Evaluator.DEFAULT, der bare refererer til Evaluator.CAR. Hvad skal der ske hvis brugeren skifter evaluator, efter ruten er lagt?

4/30/ 2011	Niklas	Fixede serialisering	Det hjalp at serialisere til samme fil, med samme stream. På den måde opstår der ikke dublikater. Desuden er fylder dataen mindre, det går hurtigere, og bruger væsentligt mindre RAM (550MiB =>260MiB)
5/1/ 2011	Niklas	Forårs-rengøring af kode	Har flyttet klasser rundt til de rigtige pakker. Har desuden gennemgået koden sammen med FindBugs, og ryddet op i koden.
5/5/ 2011	Niklas	Tilføjede anti-aliasing	Problemet var at vi tilføjede anti-aliasing på baggrunden også, og derfor kørte det langsomt. Prøvede kun at bruge det på stregerne, og det fungerer ganske udemærket, og er stadig hurtigt.
5/6/ 2011	Emil	Skønhed	Implementerede "tykke" veje med outline og et mere differencieret quadtree PS: jeg har glemt at skrive herinde en masse gange
5/8/ 2011	Jakob	LaTeX	Arbejdede med LaTeX og med smooth scrolling
5/14/ 2011	Emil	White Box Test	Har skrevet white box tests til getClosestNode metoden, mangler dog at lave det til JUnit da jeg skal vide hvordan jeg loader testgrafen, de andre tests der bruger den virker ikke ved mig

B.3 Plan for last week of project

The plan for the last week of the project. We wanted to finish early, so that we did not need to stress before hand-in.

To be done before meet:

Thursday	Friday	Satur/Sunday	Mandag	Tirsdag	Onsdag
Whitebox(Emil)	Manual(Jens)	Read through report		Spelling and grammer check	
Write about JUnit(Jakob + Jens)	UML (Niklas)	(Only content, not grammar/spelling)			
Write last part about system tests (Emil)	Worksheets(Jakob)				
Tests from WB to JUnit (Emil)	Convert tables to LaTeX tables (Jakob)				
Jar (Niklas)					
CD (Niklas)					

Things to be done at the meeting:

Thursday	Friday	Satur/Sunday	Mandag	Tirsdag	Onsdag
	Preface		Discuss corrections	Check quality	Hand-in
	Product conclusion		Write corrections into report	Print	
	Process reflection			Burn CD's	