2014

Førsteårsprojekt

DANMARKSKORT

Peter Clausen, Kristian Nielsen, Johan Arleth og Adam Engsig

Gruppe A

Indholdsfortegnelse

Forord og indledning

Rapporten og java programmet, er udarbejdet i forbindelse med førsteårsprojektet på bachelorlinjen Softwareudvikling på IT-universitet i København, foråret 2014. Rapporten beskriver, det færdige produkt der løser problemet, overvejelser og gruppearbejdet undervejs.

Rapporten henvender sig mod læsere, der har erfaring med det objektorienteret programmeringssprog Java SE, samt kendskab til Swing i denne, da koden og rapporten ligger med fundament på disse.

Til projektet er vi blevet bedt om at udvikle et program, der tillader en bruger at interagere med et kort over Danmark. Udover muligheden for at navigere et kort over Danmark, skal programmet blandt andet tillade brugeren, at indtaste en start og slut destination, hvortil programmet skal tegne og beskrive ruten. Ydermere har det været vigtigt, at programmet har kunnet håndtere den store mængde data hurtigt nok, så programmet er bekvemt at bruge.

For at kunne lave et program, der løser problemstillingen, har vi analyseret de forskellige krav, for at finde de bedste løsningsmuligheder. Kravet om et program der kører flydende har haft meget fokus, da vi har med en relativ stor datamængde at gøre. Nogle løsninger har resulteret i et ulideligt langsomt program, hvorfor vi har måtte komme op med nye og bedre løsninger på de forskellige problemer.

For at opbygge vores tiltro til at programmet virker, har vi udført en række test af programmet. Da omfanget af projektet, samt tiden til det, ikke har lagt op til at gennemteste hele programmet, har vi udvalgt nogle vigtige dele af programmet og testet disse grundigt. Ligeledes er der blevet udført en systemtest for at tjekke helheden af programmet.

Sammen med denne rapport følger selve programmet i form af en .jar fil, samt en mappe med alt kildekode.

Problemstilling

Problemstillingen er at lave et interaktivt Danmarkskort ud fra udleverede datasæt lavet af Krak og OpenStreetMap (OSM). Programmet skal kunne bruges af en almindelig bruger, der kan betjene en computermus og tastatur. Programmet skal opfylde de givne krav i opgaveformuleringen, men må derudover gerne indeholde ekstra funktionalitet.[1]

For at løse problemet er det nødvendigt at finde algoritmer og datastrukturer, der kan håndtere de store datamængder fra Krak og OSM, så programmet virker hurtigt og er bekvemt at bruge.

Udover at opfylde de stillede krav, har vi selv et ønske om, at koden til programmet skal være logisk struktureret og vedligeholdelsesvenligt. Dette vil sige at koden til programmet skal have lav kobling, og et

[1] Se bilag Krav til programmet

højt abstraktions- og kohæsions niveau. På denne måde vil programmet ligeledes være videreudviklings venligt.

Problemanalyse

For at udvikle programmet har vi skulle vælge et sprog at kode programmet i. Vi har valgt at kode programmet i JavaSE, som vores primære programmeringssprog, da det er sproget alle i gruppen er blevet undervist i, og da det er sproget vi alle finder os mest kendt med. Derudover har alle introduktioner i undervisningen til forskellige algoritmer foregået i Java.

Krav nummer 1 - Danmarkskortet skal tegnes visuelt ud fra datasættene Krak og OSM i et programvindue.

For at kunne tegne vejene i vinduet, har der været behov for at tilpasse dem, så koordinaterne passer inden for vinduet. Til dette har vi to muligheder: Den første mulighed var at bruge java klassen AffineTransform der kan ændre et billede. Ved hjælp af dette, er det muligt at skalere billedet. Den anden mulighed var skalere punkterne før kortet blev tegnet, så de passer inden for vinduet. Dette kan gøres ved at skalere området, kortet ligger inden for, op eller ned til at passe med området vinduet dækker. Det vil sige at koordinater skal skaleres med forholdet mellem vinduet og kortet.

Da vi var lavede vores egne tegnemetoder, valgte vi at bruge den sidstnævnte løsning, da den passede med det vi i forvejen havde lavet.

Krav nummer 2 - Forskellige slags vejsegmenter skal tegnes med forskellige farver.

Hvert vejsegment har sin egen vejtype i både Krak og OSM datasættene. For at farve disse har vi lavet en switch-case, der tjekker vejens type, og indstiller tegnefarven inden vejen bliver tegnet.

Dette synes vi var udviklings- og vedligeholdelsesvenligt, da man ved tilføjelse af flere vejtyper eller farveændring, kun skal ændre koden ét sted. En alternative løsning ville være, at lave en Enum for vejtypen, men man ville stadig ikke slippe for switch-casen, og vi synes derfor det var pænest og nemmest, at implementere vejfarven med en enkelt switch-case.

Krav nummer 3 - Danmarkskortet skal skalere efter størrelse, når der trækkes i vinduets ramme.

Når et vindue gøres større, ville det være ideelt, hvis vinduets kanter havde samme størrelsesforhold som det gamle vindue. Her er det forholdet mellem vinduets størrelse, og størrelsen på den del af kortet programmet viser, der tages i betragtning. Dette er dog ikke realiteten, da vi synes at vinduet skal kunne laves højere eller bredere, som ønsket af brugeren eller skærmstørrelsen, og stadig vise samme del af kortet. Programmet skal tage stilling til, om det skal skalere efter det nye forhold på højden eller bredden for at hele kortet bliver vist i skærmvinduet.

Vi har valgt at kigge på både forholdet mellem den gamle- og nye højde samt den gamle- og nye bredde, og tegne kortet efter størrelsen på den mindste kant. På denne måde har vi altid hele kortet vist, dog på bekostning af, at der kan komme tomme pletter, hvis forholdet mellem højde og bredde er forskelligt.

Havde man omvendt altid skaleret efter den største kant, ville kortet altid fylde hele vinduet, men samtidig gå ud over den ene vindueskant, således at denne del af kortet blev udregnet, men ikke vist. Det udregnede forhold mellem højde og bredde på vinduet bruger vi til at skalere med.

Krav nummer 4 - Det skal være muligt at fokusere på et bestemt område af kortet.

For at muliggøre, at brugeren kan fokusere på et bestemt punkt af kortet, kræver det at man kan zoome, enten på et bestemt område, eller ved at scroll zoome ud fra et punkt på kortet, eller andre metoder der zoomer lidt af gangen.

I gruppen blev vi enige om, at det skulle være muligt både at zoome ind på et område brugeren definerer og scroll zoome mod et punkt, så derfor har valgte vi at implementere begge dele.

For at zoome ind på et bestemt område, havde vi brug for at kunne definere et område, som skulle tegnes. Det kan løses ved at skalere det område, som brugeren definerer ved at trække en firkant med musen op, til den størrelse det svarer til i Model, og så finde det relevante data i Model. Derefter kan Model sende dataen til View, der så skalere det ned igen, så det passer med det vinduet brugeren ser.

Til scroll zoom er det nemt at bruge samme opsætning, som det at zoome ind på et bestemt område. Da man ud fra det gamle område kan definere et nyt område, der er ændret med en fast skallering. En ting ved dette er dog, at programmet skal zoome relativt til musen, da det giver en bedre oplevelse, og er den oplevelse, man forventer fra andre programmer i denne tid som fx Google Maps. Dette kan løses ved at beregne hvor på kortet musen befinder sig, skalere området, og derefter rykke området så punktet på området, som musen holdte over, bliver rykket tilbage under musen.

Det sidste med at kunne bevæge kortet, ved at ”panne”, passer ligeledes godt med denne struktur, da man kan bruge det gamle område der blev vist, og så rykke det med den mængde brugeren angiver med mus eller keyboard. Derved skabes et nyt område, med samme størrelse som det gamle, der er ligger et andet sted.

Krav nummer 5 - Vis navnet på vejen der er tættest på musen.

For at vise navnet på den nærmeste vej, overvejede vi først, hvad der ville være den bedste måde at finde de relevante veje at kigge i. Vi blev hurtigt enige om, at kravet kunne løses ved at udnytte musens position til at lave en søgning i quadtræet efter veje der ligger tæt på. For derefter at finde den nærmeste af de veje man får ud af quadtræet, valgte vi først at kigge på afstande til vejenes midtpunkter pga. det var den nemmeste løsning, selvom den var delvist forkert. Vi har senere ændret implementationen til i stedet, at kigge på afstanden til linjestykkerne, for derved altid at få den reelt nærmeste vej.

Krav nummer 6 - Det skal være muligt at finde korteste vej mellem to punkter på kortet.

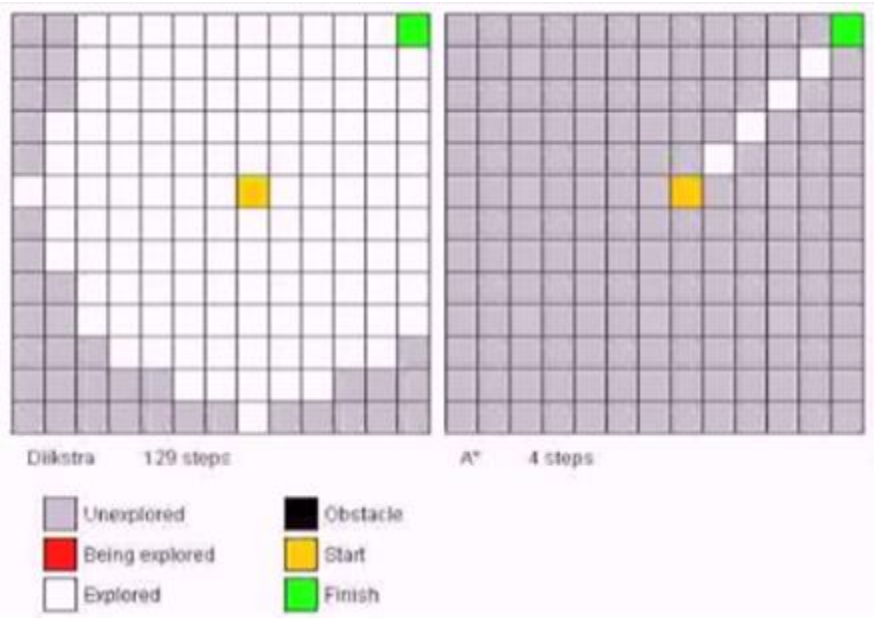
Vores program stiller til rådighed at kunne finde enten den hurtigste- eller korteste rute mellem 2 ønskede veje. Hvor den hurtigste rute er baseret på Dijkstras grafsøgningsalgoritme, er den korteste vej baseret på

A*(læs: A Stjerne), som ligger med grundlag i Dijkstras algoritme. A* er en betydeligt hurtigere algoritme, men kan ikke bruges med den hurtigste rute. Forklaring følger senere.

For at kunne bruge Dijkstras algoritme, må Danmarks vejnet anskues som en directed graph, hvor veje anskues som en vægtede kanter (edges), og vejkryds og rundkørsler som punkter (nodes). I denne sammenhæng er det vigtigt at bemærke, at alle ikke ensrettet veje kan køres i begge retninger, og af samme grund skal de fleste kanter betragtes i begge retninger. Når en ny søgning bliver lavet for punktet P, tager Dijkstras algoritme kun udgangspunkt i P og P's naboer. P's naboer er de punkter som P er forbundet til via en kant. Fra start betragtes afstanden som en uendeligt stor værdi, fra punktet P til alle andre punkter, selv P's naboer. Punkt P selv er hvor vi starter, så afstanden fra P til P må være 0. Vi kigger nu på alle P's naboer, og specielt kigger vi på den kant som forbinder P med naboerne. Den kant, der har den mindste værdi er den vi ser på først, og det punkt, som førhen blev betragtet som uendeligt langt, bliver opdateret til den nye værdi, som er værdien af kanten plus værdien af det tidligere punkt. Ydermere må vi holde styr på den korteste rute til punktet, og derfor lader vi hvert punkt vi har besøgt, vide hvor vi kom fra. I vores tilfælde vil værdien altså bare være værdien af kanten da P er lig med 0, og vi lader punktet vide at vi kom fra P. Punkterne vil altså altid indeholde den akkumuleret værdi af alle kanterne fra P og til det punkt. Hvis man når til et punkt som allerede har en værdi, vil punktets værdi ikke blive opdateret da den nye vej, som er fundet til dette punkt, altid vil have større værdi. Samme proces vil som udgangspunkt bliver gentaget indtil der ikke er flere punkter tilbage, og vi kender alle værdier til alle punkter.

Det vil dog være spild af arbejde og meget tidskrævende for store grafer at finde distancen til alle punkter. Vi lader i stedet algoritmen stoppe i det vi når til det ønskede punkt. Da vi altid har prioriteret at følge den korteste kant, kan vi være sikre på at der ikke er en hurtigere vej til punktet selvom vi ikke har været alle igennem.

A* er ligeledes som beskrevet ovenfor, forskellen ligger dog i valget af hvilket punkt der skal behandles som det næste. Hvor Dijkstra kun kigger på den akkumuleret værdi fra vores udgangspunkt, kigger A* ydermere på en beregnet værdi for hvor langt der er til slut punktet. Den heuristiske værdi for afstanden til slutpunktet, kan udregnes ved Pythagoras, antaget at hvert punkt har et koordinatsæt. Pythagroas læresætning siger: $a^2 + b^2 = c^2$. Med vores koordinatsæt kan a og b udregnes ved: $a = x_1 - x_2$ og $b = y_1 - y_2$. Da kvadratet af a og b bliver taget, ændre det ikke noget at de engang imellem kan være negative. Den udregnet afstand til slutpunktet er altså i fugleflugt. Når 2 punkter bliver sammenlignet i A*, kigger man altså på værdien fra startpunktet + heuristiske værdi. Det punkt med den mindste totale værdi, vil blive valgt.



Figur 1: Illustration af Dijkstra og A i brug*[2]

Antaget at der er lige langt imellem alle punkter, er her et eksempel på hvordan A* opfører sig i forhold til Dijkstra. Som det ses udvider Dijkstra sig i en cirkel rundt om startpunktet, hvor A* går direkte imod slutpunktet netop pga. den heuristiske værdi. Forskellen I dette eksempel er altså 125 skridt. Dijkstra bruger 129 skridt, hvor A* kun bruger 4 skridt.

A* bruger vi kun til at beregne den korteste vej, hvor den hurtigste vej bruger Dijkstra. Den korteste vej sammenligner på afstand, hvor den hurtigste vej kigger på længden af vejen divideret med fartgrænsen på den givne vej, altså tiden det vil tage at kører det vejstykke. Problemet i at finde den hurtigste vej med A*, er at fx en motorvej kan godt kører en stor bue rundt, men stadig være hurtigere pga. fartgrænsen. En simpel løsning på dette problem har vi ikke kunne finde, og derfor lader vi ruteplanlægning via den hurtigste vej bruge Dijkstra, da vi ved at dette vil give den korrekte rute.

Bredde-først-søgning (BFS) og Dybde-først-søgning (DFS) er begge algoritmer der bruges til at søge i en graf efter den korteste rute mellem 2 punkter. De finder dog ikke den korteste rute i forhold længde men i forhold til hvor mange kanter der er til det givne punkt. Dette vil altså ikke være en effektiv måde at få en navigeret rutevejledning i vores tilfælde.

Alternativer til Dijkstra og A* er ikke blevet undersøgt og taget i betragtning, da det er netop er disse 2 algoritmer vi er blevet introduceret til, og undervist i. Ydermere har vi ikke haft problemer med køretiden af hverken Dijkstra eller A*, og da vi har haft en deadline har vores fokus ligget på algoritmer og dele af implementationer der har virket knapt så godt.

[2] Kilde: UNSWMechatronics YouTube kanal: https://www.youtube.com/watch?v=cSxnOm5aceA

Krav nummer 7 - Programmet skal have en sammenhængende brugergrænseflade for kortet og ekstra funktionalitet.

Til udvikling af brugergrænsefladen har vi brugt udelukkende JFC Swing. Vi har selv designet og skrevet vores brugergrænseflade fra bunden i Swing, uden brug af drag-and-drop hjælpemidler som eksempelvis NetBeans IDE udbyder. Vi har valgt at gøre dette for at have fuld kontrol over vores kode og program. Andre alternative GUI-toolkits er ikke blevet taget i betragtning, da alle gruppemedlemmer har brugt Swing til tidligere projekter, og da vi synes Swing kunne alt hvad vi skulle bruge til denne projektstørrelse.

Til design af brugergrænseflade ville alle gruppens medlemmer gerne have et simpelt og stilrent design. Vi vil gerne gøre kortet så stort som muligt, da dette er hovedfokus i vores program. Søgefunktionalitet har vi valgt at placere i venstre side af programvinduet, da vi synes at det var den mest intuitive placering.

Vi har valgt ikke at implementerer knapper med få funktioner som fx én knap der zoomer ind, eller én knap der zoomer ud, da dette ville tage næsten unødvendig plads fra vores kort og søgebar. Vi har valgt ikke at lave en menu til ændring af styringsfunktioner som fx farve på veje, eller sprog, da vi synes dette ville tage for lang tid at implementere i forhold til programmets funktionalitet og krav.

I stedet har vi fokuseret på at gøre programmets design enkelt og med ekstra funktionalitet. Det er muligt at have fokus på de forskellige elementer i vores program som fx kortet, søgeboksene eller rutevejledningen, så styringen af disse sker gennem musen eller tastaturknapper. Brugergrænsefladen virker fuldt ud uanset dimensionerne på programvinduet (dog med et krav om en minimumsstørrelse).

Vi ville gerne have alle programmets funktioner i ét vindue, men stadig have mulighed for at gøre programvinduet tilpasseligt lille eller stort efter brugerens behov, dog med en minimums begrænsning. Gruppens medlemmer var enige i programmets udseende og denne blev udvidet undervejs gennem projektet. Gruppens medlemmer er tilfredse med resultatet af brugergrænseflade design, og vi mener det opfylder projektets og egne krav til programmet.

Krav nummer 8 - Optimering af programmets hastighed efter opstart

Både Krak og OpenStreetMaps data er meget stort, da de indeholder data for hele Danmarks vejnet. Især OpenStreetMaps data er meget detaljeret og fylder derfor væsentlig mere end Kraks data. Så for at programmet skal køre i en hastighed man kan arbejde med, kræver det at man på en effektiv måde kan udvælge et relevant subset af data, så man ikke behøver at bruge alt dataen hele tiden.

For at løse dette problem, har vi kigget på to datastrukturer: quad-træ og KD-træ, som vi gennem forelæsningerne på ITU blev introduceret til. Da begge strukturer synes af have tilnærmelsesvis samme effektivitet, valgte vi at gå med den struktur der intuitivt gav mest mening for os. Begge strukturer er rimelig simple at forholde sig til, men quad-træer er alligevel mest intuitive (for os), da det er ensformige firkanter, i samme forhold som kortet, man arbejder med, og derfor gik vi med denne struktur. KD-træet deler data op i K dimensioner (i dette tilfælde to dimensioner), hvilket giver forskellige størrelses firkanter alt efter hvordan dataen er fordelt.

Et quad-træ deler data op i flere små dele. Man tager sit data og deler det op i fire firkanter alt efter koordinater. Derefter kigger man i hver firkant om der er flere elementer end man ønsker, og deler derefter firkanten op i fire mindre firkanter, hvor elementerne igen fordeles efter koordinater. Dette fortsætter indtil man når det antal elementer i hver firkant som man ønsker (i vores program er antallet af elementer sat til 1000). Når man er færdig med at lave firkanter, har man en datastruktur, der i områder med mange elementer, er delt op i meget små firkanter, mens områder med få elementer består, af færre store firkanter. Dette giver god mulighed for at søge effektivt efter elementer på et bestemt område af et kort, uden af skulle igennem resten af data. Dette er en vital funktionalitet for mange features der skal bruge dataen, da programmet ellers ville køre meget langsomt, hvis programmet skulle søge i alle veje, inklusiv veje der ikke skulle tegnes.

Selvom man nu hurtigt kan finde relevant data, er det relevante subset af data stadig for stort til at behandle hver gang brugeren interagerer med programmet eller der sker en ændring på en anden måde, især dataen fra OpenStreetMap er for stor til at behandle hele tiden, sågar når det er en mindre del af dataen man arbejder med.

Derfor skal der ligeledes bruges en struktur til at reduceret antallet af gange dataen skal behandles. Af de muligheder vi har undersøgt udgør java's BufferedImage der kan gemme det man tegner som et billede, en vital rolle, da man kan gemme den behandlede data på billedet.

Den Første og mest simple løsning vi kiggede på, var at tegne et dobbelt så stort område, som det der vises på skærmen til et BufferedImage. Det BufferedImage kan man så bruge indtil behovet for et nyt opstår, f.eks. hvis man bevæger sig ud for det tegnede område eller zoomer. Dette er nemt at implementere og undgår nogle vanskeligheder med lange veje som beskrevet under det næste løsningsforslag med tiles. Ulempen er dog at man skal behandle en væsentlig større mængde data end det brugeren egentlig ser. Denne mængde data skal endda behandles igen, selvom man kun bevæger sig lidt udover det, i forvejen, tegnede område.

Den anden løsning vi kiggede på, var at opdele kortet i tiles, hvilket vil sig man opdeler kortet i et gitter. Dette medfører man kan tegne de kvadrater af gitteret brugerne kan se noget af. Dette fjerner for det første behovet for at behandle alt dataen igen, og for det andet kan man gemme en i forvejen tegnet tile, da den har en fast position i forhold til resten af tiles. Dermed skal data kun behandles en gang indtil, man zoomer ind eller ud, da det skaber et nyt billede der skal deles op.

Selvom dette giver bedre performance, har det nogle tekniske vanskeligheder, som f.eks. det der blev nævnt tidligere, da et tile ikke nødvendigvis er bredt nok til at tegne en hel vej, men vejen bliver heller ikke tegner i tiles ved siden af.

Den tredje løsning minder meget om det første, men i stedet for at behandle dataen når behovet opstår, kører en anden tråd i baggrunden der behandler dataen, mens man rykker kortet eller zoomer, dette gør at

man undgår ventetid mens dataen skal behandles igen. Denne løsning er dog meget teknisk svær, da det kræver at man skal have to tråde kørende.

Vi valgte at bruge den anden, da den første ikke gav den oplevelse vi ønskede, da brugeren ville opleve ventetider, hver gang man rykker kortet ud over det behandlede område, selv hvis det kun var et lille område det ikke var behandlet i forvejen.

Den tredje løsning, selvom det er en rigtig god løsning på problemet, blev for avanceret og med for få forbedringer i forhold til at arbejde med tiles, at vi ikke følte for at give os i kast med den.

Måden hvorpå tiles kan laves er at lave et gitter for hele billedet hvor tiles er nummeret fra venstre mod højre, fra top til bund. Måden hvorpå dette gitter kan udregnes er ved at dele det billede man får når man skalere original billedet ned til at passe med zoom, op bredden af billedet delt med tile bredden rundet op, for at få det hele med, og det samme gøres for højden[3]. Dette gøres når programmet startes eller når tiles skal tegnes om, f.eks. når man zoomer.

Herefter skal man finde de første tiles der skal tegnes. Dette regnes for x-aksen ved at dele den mindste x værdi for området det vises delt med tile bredden rundet ned. Ud fra det fås start nummeret på den første tile. Den sidste tile i bredden regnes ved at dividere maks x værdien, for området der vises, med tile bredden rundet op. Bagefter gøres det for y-aksen, med samme fremgangsmåde for y-værdierne.

Derefter har man start og stop værdier for en for-løkke med en indlejret for-løkke, til at løbe alle tiles igennem der skal tegnes.

Det sidste punkt i forhold til programmet skal være hurtigt nok er load tid for programmet. Kraks data kan, med den medfølgende parser, hentes forholdsvis hurtigt ind i programmet. Datafilen for Danmark, fra OpenStreetMap, er derimod som standard 4.1 GB, mod de 146 MB for Kraks data. For at loade dataen fra OpenStreetMap i en acceptabel tid, skal den behandles, så størrelsen på dataen reduceres eller gøres nemmere at læse.

Der er en del muligheder for at indlæse dataen fra OpenStreetMap i en ordentlig tid bl.a. at dele dataen op i flere filer, så man kan hente det vigtigste først og så videre. En anden mulighed var at lave sit eget filformat med bedre muligheder end XML, som det som standard er i. Først og fremmest skal man dog reducere dataen ved at sortere irrelevant data fra.

Selvom der muligvis er fordele at hente ved at ændre til flere filer eller et andet format, valgte vi at blive ved det standard XML format og nøjes med at reducere dataen samt ændre lidt ved værdierne, da vi hellere ville bruge tid på andre ting. Vi konverterede koordinaterne fra wts84 til UTM 32 og satte dem ind, samtidig med at vi gav knuderne id fra 0 og opefter. Dette gør der ikke skal bruges tid på at projektere alle koordinator til UTM 32 i programmet. Ændring af id gør at knuderne kan projekteres til

[3] Tiles er i dette tilfælde kvadrater så højde = bredde

Krav nummer 9 - Brugeren skal kunne vælge mellem KRAK eller OpenStreetMaps.

For at kunne vælge mellem OSM og Krak skal man kunne hente dataen ind i samme format så programmet ikke skal bruge forskellige metode og klasser til hver. For at gøre det, har vi overvejet om vi skulle skifte til et mere generisk model for dataen eller sætte dataen fra OpenStreetMap op, i samme model som dataen fra Krak.

Vi har valgt at bruge den model vi har fra Kraks data og ændre OpenStreetMaps data til sammen model. Det gjorde vi fordi det for os gav mest mening, i forhold til at forstå og løse opgaver.

Til Krak medfulgte en parser som vi valgte at bruge, da vi så undgik at skulle ændre ved Kraks data, som bruger et ret specielt format. OpenStreetMap dataen er gemt som ".osm" filer der bruger XML's format. Vi har undersøgt tre mulige måder at parse XML formatet på: DOM, SAX og StAX.

Da DOM loader hele dokumentet ind i hukommelsen, var det ikke en mulighed da dataen var alt for stor til at ligge i hukommelsen. SAX læser XML-dokumentet fra lageret og skubber begivenheder, f.eks. når man møder et nyt element, til programmet. StAX gør næsten det samme som SAX, men trækker trækker i stedet begivenheder til programmet. Umiddelbart skulle StAX være at foretrække over SAX, fordi StAX skulle være nemmere at arbejde med.

Valget stod derfor mellem SAX og StAX, men fordi vi fik en introduktion til SAX, valgte vi at bruge den type parser til programmet, da der ikke skulle være noget nævneværdig forskel i performance.

Krav nummer 10 - Programmet skal indeholde mindst én udvidelse fra listen 'Extensions'

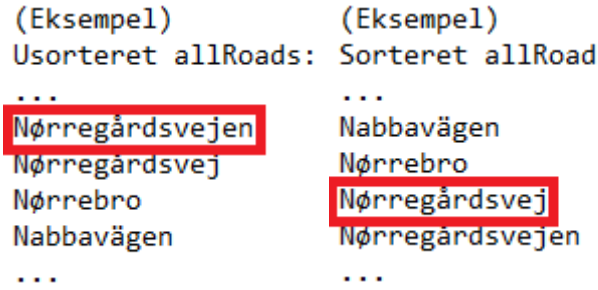
Som udvidelse af programmet har vi lavet en Search-as-you-type feature. Vi valgte at lave denne feature som "type-ahead-funktion", der færdigskriver vejnavnet brugeren er i gang med at indtaste med markeret tekst, som ikke forstyrrer brugeren mens han eller hun skriver, men kan forslå endelsen på adressen man gerne vil søge på. Dette kan hjælpe brugeren med at skrive vejnavnet hurtigere, eller foreslå vejnavne med lignende start, som findes i datasættet.

En anden variant af search-as-you-type, som vi også var i gang med at implementere, men ikke nåede at blive færdige med, var et drop-down vindue til hvert SearchField, der kom med forslag til hvilken vej brugeren vil vælge. Brugeren ville så kunne vælge en adresse fra listen. Java har dog ikke en nem klasse man bruge til dette, andet end en JPopupMenu, som vi ikke havde tid til at gøre færdig.

Vi har valgt at bruge en KeyListener til styrring af search-as-you-type, der tjekker længden på den brugerindtastede adresse i søgefeltet og starter search-as-you-type funktionen, hvis adressen er blevet længere siden sidste tryk på en tast. Egentlig ville vi have implementeret en DocumentListener på SearchField, som skulle lave samme tjek for adressens længde. Vi kunne dog ikke få DocumentListener til at virke på vores SearchField, og valgte derfor at lave samme funktionalitet med en KeyListener (klassen KL) i stedet. En ulempe ved dette er dog, at SearchField bliver tjekket hver gang vi trykker på en tast på keyboardet fx shift, i stedet for at udføre samme funktion kun når tekstfeltet ændres. Fra et

vedligeholdelses- og videreudviklings synspunkt, er brugen af KeyListener til dette formål en dårlig taktik, da denne løsning giver lidt ekstra kode, og lavere kohæsions niveau for klassen KL. Dette får også programmet til at skulle lave flere redundante udregninger. Dette har vi dog ikke på nuværende tidspunkt tid til at ændre, og derfor har vi valgt løsningen med KeyListener frem for DocumentListener.

Til søgning af vejennavn laver vores program lige nu en lineær gennemgang af alle vejene og sammenligner brugerens input med starten af vejnavnet. Dette går hurtigt nok til search-as-you-type funktionaliteten med både Krak og OSM datasættene. Når vejlisten ”allRoads” bliver lavet i StartMap, bliver den herefter sorteret. Dette er med henblik på lineær gennemgang i vejnavnssøgningen. Indtaster man fx ”Nørregårdsvej”, vil man muligvis i en usorteret liste få vejen ”Nørregårdsvejen”, mens at man i en sorteret liste vil få vejen ”Nørregårdsvej”, som faktisk er hele navnet på en vej. Det ville derfor ikke være muligt at finde vejen ”Nørregårdsvej” i en usorteret liste ved linæer gennemgang. Dette gør også, at vores search-as-you-type altid vil foreslå det korteste vejnavn med samme start.



Figur 2: Eksempel på vejnavnssøgning af ”Nørregårdsvej” i henholdsvis usorteret og sorteret liste

Vi har overloadede metoder i klassen RoadNameSearcher, der kan søge på veje med både vejnavn, eller vejnavn og postnummer. I Kraks datasæt har vi den fordel, at hver edge også indeholder postnummer. Vi kan derfor udnytte denne fordel til, at lokaliserer den eksakte vej i en bestemt by. OSM datasættet indeholder ikke postnumre, og vi kan derfor ikke være sikre på, om brugeren mener Nørregårdsvej, 2610 Rødovre, eller Nørregårdsvej, 5771 Stenstrup, selvom brugeren også har indtastet postnummer når der sammenlignes med OSM dataen. Dette har vi ikke mulighed for at ændre på med OSM datasættet, med mindre vi skriver alle postnumrene ind manuelt i datasættet, men vi har mulighed for at udnytte datafeltet med postnumre fra Kraks datasæt.

Man ville også kunne lave metoder der foreslog bygningsnummer, bygningsbogstaver, etagenummer osv. Vi synes dog ikke dette gav mening i vores type-ahead funktion af search-as-you-type, og vi har derfor ikke valgt at bruge tid på at implementerer disse. Det ville dog give mening, hvis man havde lavet drop-down løsingen af search-as-you-type, som vi desværre ikke fik tid til.

Et andet alternativ til vejnavnssøgning vi desværre ikke blev færdige med, var at implementere datastrukturen Trie. En Trie ville gøre søgningen hurtigere, da søgningen kun skulle foregå i veje med samme start, og ikke i hele listen. Vores lineære søgning søger i værste fald hele listen igennem, hvis vejnavnet er navnet på sidste vej på listen vi leder efter. Triens hastighed ville i værste fald være bestemt af

alfabetets størrelse, og længden på det længste ord. Efter nogle udregninger fandt vi ud af at denne søgning ville optimere hastigheden på vejnavnssøgningen, vi blev dog ikke færdige med implantationen af Trien, og måtte derfor holde os til den lineære søgning af vejnavne. En ikke færdig version af en Trie blev dog implementeret, men ikke brugt. Læs mere om denne i diskussionen.

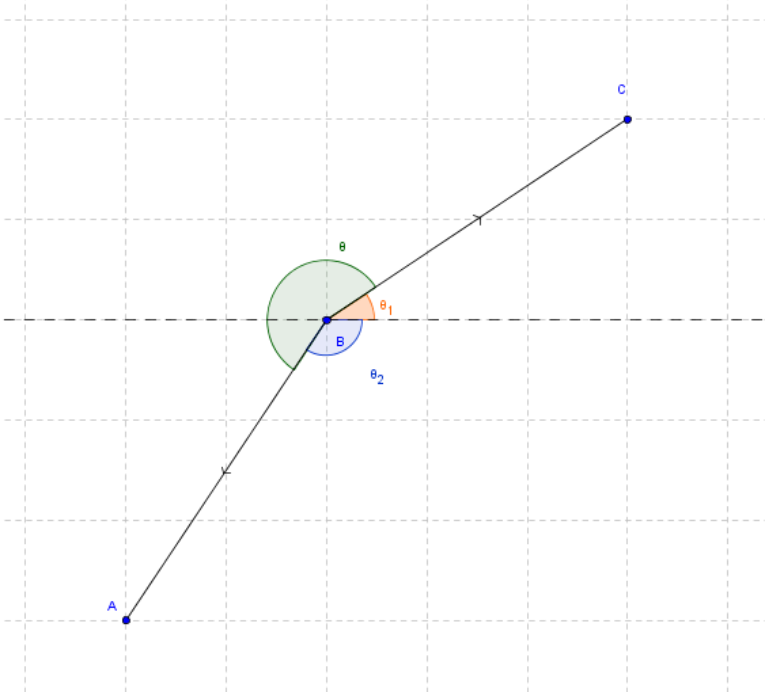
Krav nummer 11 - Rutevejledning i tekstform.

Se krav 6 som grundlag.

For at kunne lave en god og detaljeret rute vejledningsbeskrivelse, kræves det at vide hvilken vej man skal dreje. Dette er ikke noget vi har fået opgivet men kan udregnes. For at udregne svinget har vi brug for at kende 2 kanter, og deres punkter. Den kant vi er på, og den kant vi skal dreje af til. Hvis vi antager at vi har punkterne A, B og C og kanterne fra A til B og fra B til C, kan vinklen mellem AB og BC udregnes ved hjælp af vektorregning. Vi har først brug for at vores 2 vektorer har udgangspunkt i samme sted, netop hvor de er forbundet altså punkt B. Hvorfor vi må udregne $\overrightarrow{BA}$ og $\overrightarrow{BC}$.

$$\overrightarrow{BA} = \begin{pmatrix} a_1 - b_1 \\ a_2 - b_2 \end{pmatrix} \text{ og } \overrightarrow{BC} = \begin{pmatrix} c_1 - b_1 \\ c_2 - b_2 \end{pmatrix}.$$

Vi kan nu med tan^{-1} hver især finde vinklen til x-aksen for de 2 vektorer, da tangens siger at vinkel $\theta = \frac{mod.katete}{hos.katete}$. Som en del af java bliver der stillet til rådighed at når man tager tan^{-1} kan man altid få de i forhold til den positive x-akse, altså inden for π og $-\pi$. Metoden hedder atan2[4].



Figur 3: Illustration af vinkel udregning mellem 2 vektorer. Det er antaget at vi kommer fra A mod B, mod C

[4] http://docs.oracle.com/javase/6/docs/api/java/lang/Math.html#atan2%28double,%20double%29

Det vil sige at vi kan finde vinklen mellem de 2 vektorer ved: $\theta_2 = atan2(\overrightarrow{BA})$ og $\theta_1 = atan2(\overrightarrow{BC})$ hvoraf vi kan finde $\theta = \theta_2 - \theta_1$. I nogle tilfælde, som fx på figur 3, vil θ være negativ, hvilket ikke er ønsket. I de tilfælde hvor $\theta < 0$ vil vi lægge $2 * \pi$ til , altså en hel cirkel, og den ønskede vinkel fås.

Vi kan da nu konkludere at hvis vinkel θ er mindre end π må det være et venstre sving, og hvis θ er større end π på det være et højre sving.

Disse sving er dog kun antagelser, idet det ikke er sikkert at vejstykket slutter i samme retning som svinget man tager. Forestil en motorvejsafkørsel hvor man tit tager en afkørsel til højre, men kører over en bro med det samme, så man rent faktisk ender på venstre side af motorvejen. Svinget vil blive udregnet til venstre i forhold til ovenstående antagelse. Det er dog den tætteste og mest præcise beregning vi kan lave, vedrørende hvilken vej man skal dreje, og i langt de fleste tilfælde vil det være det rigtige sving der bliver foreslået.

Udover svingets retning indeholder hvert punkt værdien der er fra start til sig selv, hvor den kommer fra, og hvilken kant den kom med. Med disse oplysninger kan man lave en perfekt detaljeret rutevejledning på tekstform. Rute vejledningen er i pseudo kode som følger:

"Total længde: 'n_længde' m, Total tid: 'n_tid' min"
"'1_vejnavn': tag '1_sving' om '1_afstand' mod:"
" '2:_vejnavn' tag '2_sving' om '2_ afstand' mod:"
" 'n-1_vejnavn' tag 'n-1_sving' om 'n-1_afstand' mod:"
"'n_vejnavn' og destinationen er nået."

Hvor tallene 1 og 2 repræsenterer det første og andet punkt, og n repræsentere det sidste punkt, som er destinationen. Den total længde og tid må altså være det samme som værdien af det sidste punkt, punkt n da værdierne er akkumulerede.

Eget krav nummer 1 - Logisk struktureret og vedligeholdelsesvenlig kode med lav kobling, og et højt abstraktions- og kohæsions niveau.

Vores kode har haft brug for en struktur der tillader brugeren at interagerer med programmet, håndtere dataen og vise dataen til brugeren. Vi overvejede at tage udgangspunkt i et af de to design patterns: Model-View-Controller (MVC) og Model-View-Presenter (MVP), da de passer utrolig godt med det programmet skal kunne.

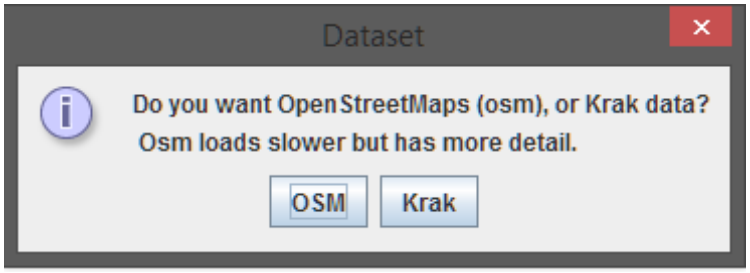
MVC og MVP, som navnene antyder, minder meget om hinanden, forskellen ligger i Controller mod Presenter. I MVP spiller Presenter en rolle som mellemmand, View kender ikke til Model og omvendt, så når der sker noget i View, bliver Presenter prompted, som ændrer i Model og returnere det nye data til View. I MVC ændrer Controller, Model som så opdatere View, som igen prompter Controller, når der sker noget.

Da der var et bedre kendskab til MVC i gruppen og fordi vi ville arbejde ud fra et design pattern og ikke følge det slavisk, faldt valget på MVC. Dette medfører en logisk struktur i programmet, som vi gerne ville opnå.

For at gøre det nemt at arbejde uden kendskab til quadtræet, har vi valgt at give det et interface. Derved kan man arbejde med interfacet, uden at have kendskab til de indre funktioner i quadtæet.

Brugervejledning og eksempler

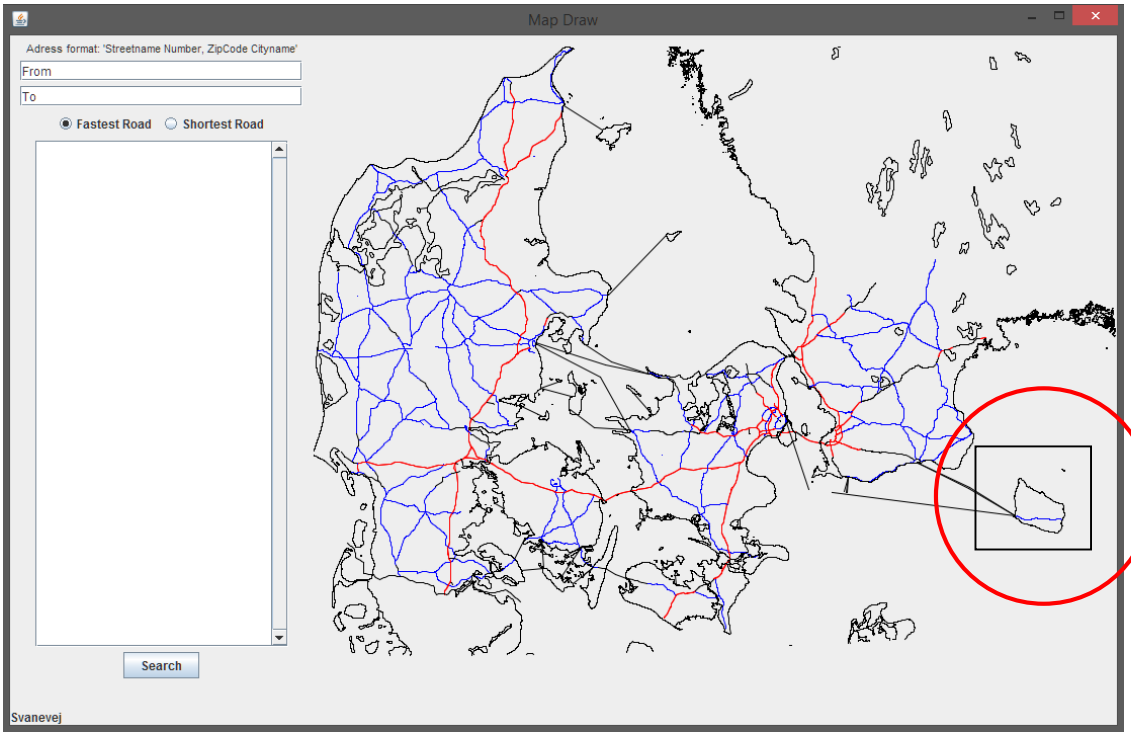
Når programmet køres vil det første man ser, være en prompt der spørger om man vil bruge krak data eller open street maps data. Der er samme funktionalitet med begge set, bortset fra at skåne kun er med på krak, men open street maps kan være langsommere på mindre hurtige computere, men er mere detaljeret end krak. Tryk "OSM" for open street maps eller "Krak" for krak. Hvis open street maps er valgt, vær da klar på at der kan være op til flere minutters opstartstid.



Figur 4: Billede af valg af data

Når en korttype er valgt kan der gå et stykke mens programmet loader kortet op først gang. Når kortet er færdigt med at loade, kommer kortet til syne. Til at starte med vil man se hele Danmark uden mange detaljer. Nu åbner mulighederne sig for hvad man vil. Funktionerne bruges ved hjælp af musen, men noget funktionalitet kan ligeledes opnås ved hjælp af tastaturet. Venstre musetast holdes nede for at panne kortet, piletasterne på tastaturet kan bruges til samme, dog vil piletasterne ikke fungere hvis der er blevet trykket på andet end kortet, indtil kortet trykkes på igen.

Zoom kan foregå på to måder. Der er scroll zoom der foregår ved at rulle på musens rullehjul, og zoomer i retning af musen. En anden måde er at holde højre musetast nede og trække et område man vil zoome ind på, det trækkede område markeres med en sort firkant mens dette udføres.

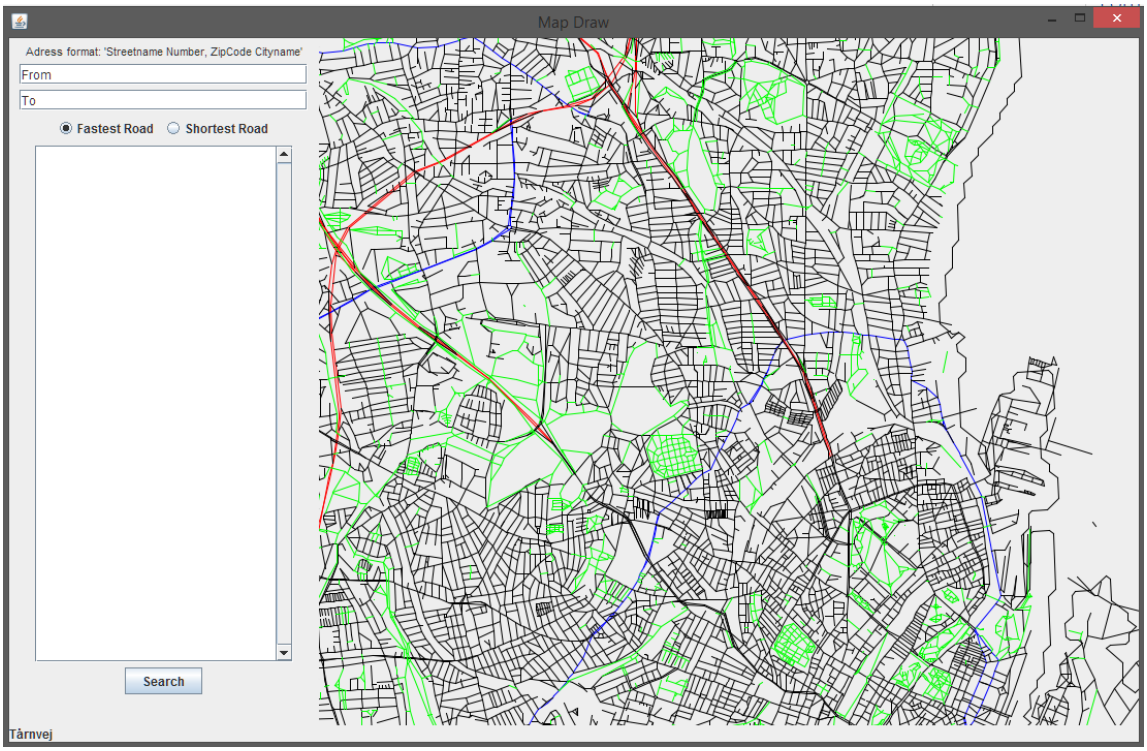


Figur 5: Billede af program vindue med firkantszoom

En anden funktion af højre musetast er at resette kortet til det originale zoom niveau hvor hele Danmark kan ses. Dette udføres ved at klikke på højre musetast uden at flytte musen.

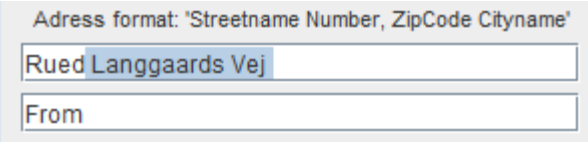
Når der zoomes ind på områder af kortet vil der komme flere og flere detaljer til syne.

Når musen bevæges over kortet vil man nederst til venstre se navnet på den nærmeste vej fra musens position.



Figur 6: Billede af zoomet by

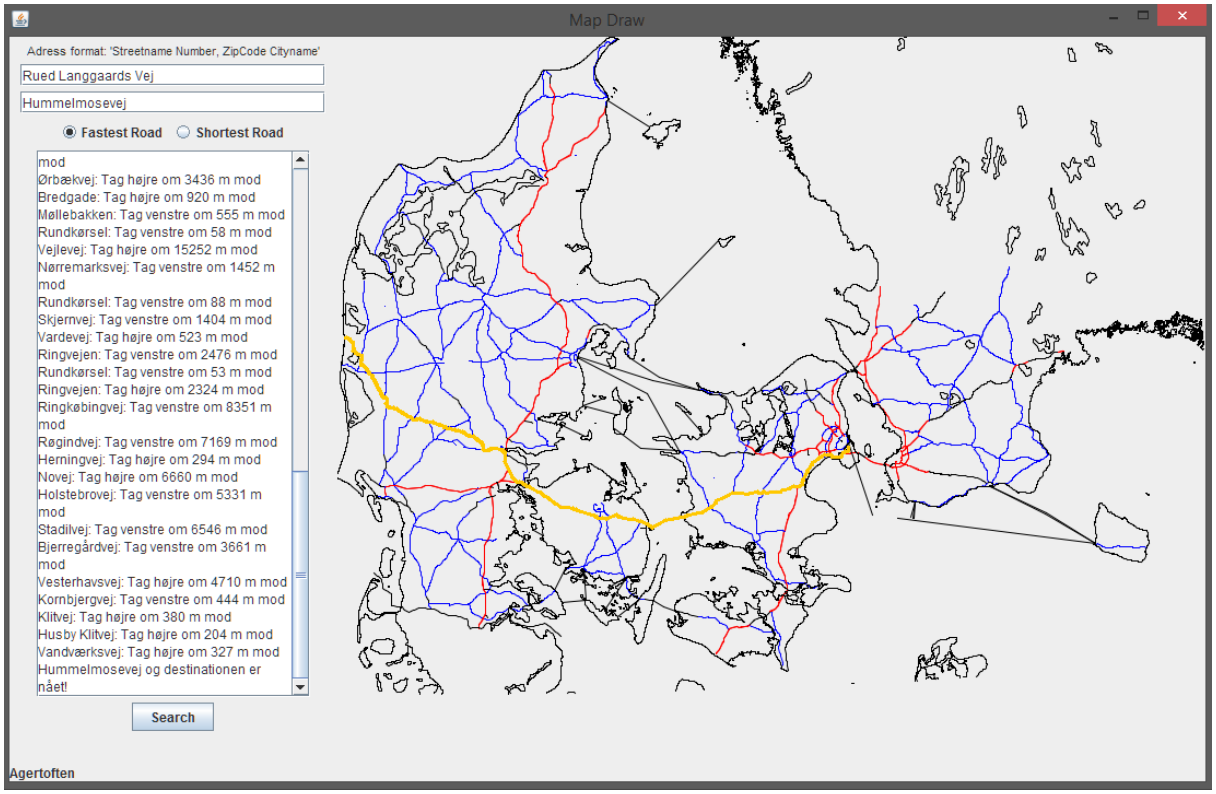
Ud over at kunne kigge på Danmark og skåne, kan man ydermere bruge programmet til ruteplanlægning. I venstre side af det opstartede program er der to tekstfelter, kaldet "to" og "from", her skriver man sin startadresse i "from" og hvor man vil hen i "to". Mens adresserne udfyldes, vil der komme autoudførelse forslag med adresser der findes på kortet. I toppen står det format adresserne skal skrives, dog kan simplere versioner, som f.eks. kun at skrive et vejnavn, ligeledes virke.



Figur 7: Billede af search-as-you-type færdiggørelse

Når man har valgt hvor man vil fra og til, trykkes der på søg, og man vil derefter se en visuel repræsentation af ruten på kortet med gult. Derudover vil der i tekstboksen under adresse felterne komme en tekstbaseret rutevejledning.

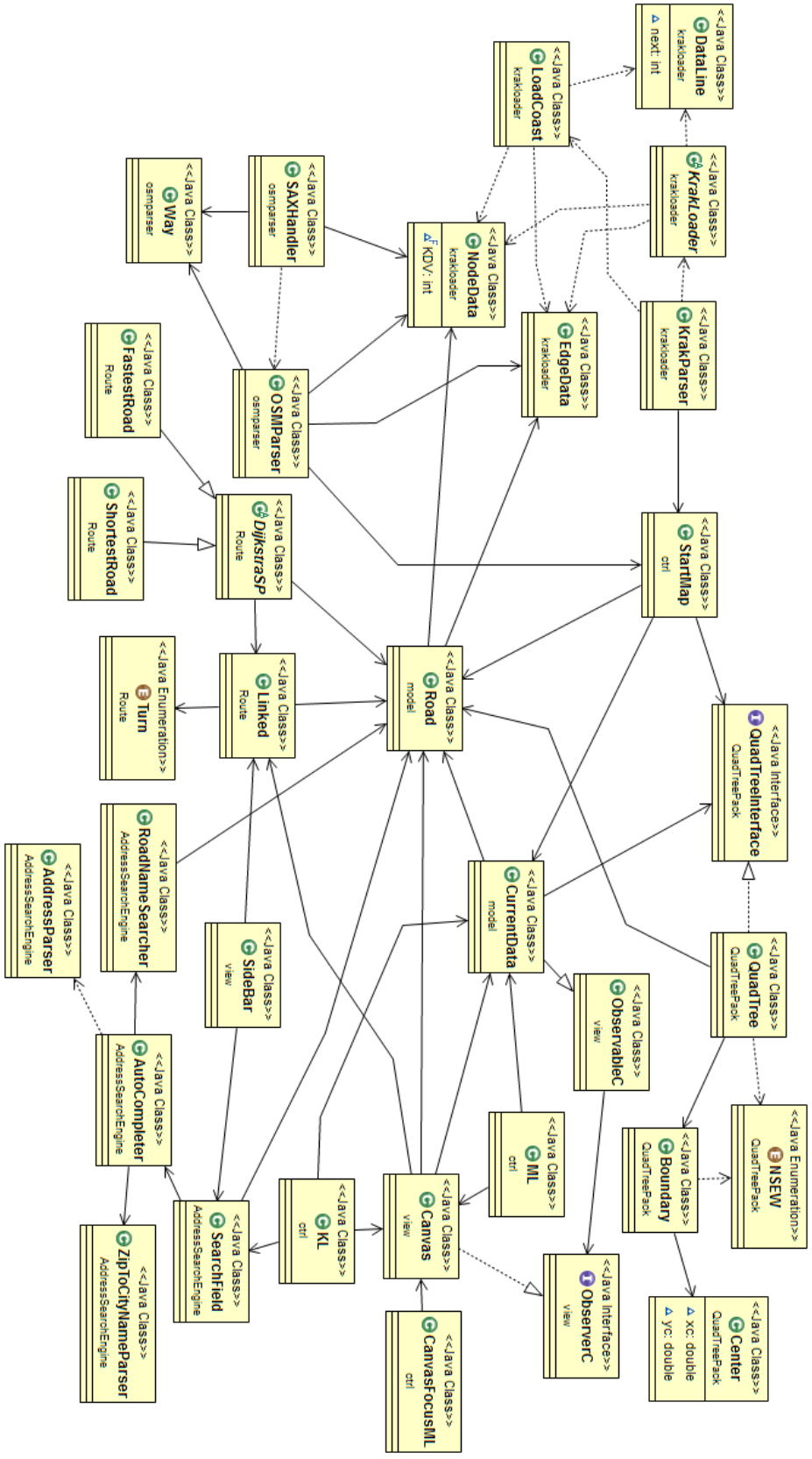
Man har mulighed for at vælge imellem "Fastest Road" og "Shortest Road". Fastest vælger veje ud laveste køretid, men shortest vælger ud fra korteste distance.



Figur 8: Billede af tekstbaseret rute

Tillykke! Du burde nu være i stand til at bruge programmet.

Teknisk beskrivelse af programmet



Figur 9 klassediagram og relationer for hele programmet.

Som beskrevet har vi taget udgangspunkt i MVC som struktur for vores program. På klasse diagrammet kan man se ML og KL, som her svarer til Controller, og har forbindelse til både CurrentData og Canvas, der svarer til henholdsvis Model og VIew. CurrentData har ligeledes, gennem QuadTreeInterface, forbindelse til QuadTree klassen, der er hoveddelen af Model, da den indeholder alt dataen fra Krak/OpenStreetMap. CurrentData er derved den klasse resten af programmet har forbindelse til i Model.

SideBar der er den anden del af View, laver lidt samme opsætning, da den via SearchField er forbundet til KL, som forbinder til CurrentData. Derudover er SearchField ligeledes forbundet til CurrentData.

Som vist i klassediagrammet har StartMap en forbindelse til både KrakParser og OSMParser, så det er muligt at hente enten Krak eller OpenStreetMap dataen, alt efter hvad brugeren vil benytte. Derudover har StartMap forbindelse til QuadTree via QuadTreeInterface, samt CurrentData, så quad-træet og CurrentData kan blive bygget op.

Som der ligeledes kan ses, har rigtig mange klasser en forbindelse til Road, da det er det objekt vi bruger til at definere et vejstykke og derfor det objekt der hovedsageligt bliver sendt rundt i programmet.

QuadTreePack

Pakken QuadTree i programmet indeholder klasser der tilsammen danner et quadtree som kan sortere objekter af typen road der er lavet i model pakken.

Pakken indeholder klasserne: QuadTree, Boundary, Center, NSEW og QuadTreeInterface. De er nævnt i rækkefølge efter hvor meget der foregår i hver enkelt klasse. QuadTree indeholder to public metoder, “public void insert(Road rd)” som tager et “Road” object og indsætter i quad-træet, her bliver holdt styr på mængden af indsatte objekter i hver quad og der bliver delt op i mindre quads når grænsen bliver overskredet.

Den anden er “public ArrayList<Road> search(double x1, double x2, double y1, double y2)”. Metoden tager to punkter der repræsenterer det område på kortet du ønsker at få veje fra, og de fundne veje bliver returneret i en arrayliste. Selve search metoden er et mellemled der kalder videre til den private metode “getRoads” som laver rekursive kald ned igennem quad-træet for at finde de relevante veje. Search laver en arraylist og giver den som pointer til getRoads, der så bruger den til at gemme vejene i efterhånden som de bliver fundet.

Klassen Boundary bliver lavet i hver Instans af klassen QuadTree, og indeholder informationer om den pågældende quads grænser. dvs. hvilket koordinatsæt den dækker, samt metoder til at checke om et givent punkt eller en given firkant er inden for/overlapper grænsen. Boundary opretter ligeledes en instans af Center klassen til at holde styr på midtpunktet i den pågældende quad.

NSEW klassen er en enum der bruges til at styre hvad der skal ske når en ny quad laves i henholdsvis nordøst, nordvest, sydøst og sydvest retningerne. Klassen boundary bruger denne information til at udregne de korrekte grænser og center punktet når nye quads laves.

QuadTreeInterface er lavet for at gøre det nemt at se hvad man skal give og hvad man får tilbage når man kalder quad-træet, samt for at gøre det muligt at kalde i gennem interfacet. Interfacet indeholder abstrakte versioner af insert og search som implementeres i QuadTree.

Route package:

Pakken Route indeholder de klasser der bruges til at finde enten den hurtigste, eller korteste vej, samt at få denne rute gemt. Den kantvægtedegraf algoritmen bruger (feltet adj i StartMap) er uforanderlig igennem hele programmet levetid, hvorfor det er final. Samtidigt tager det tid at bygge denne graf, og af netop disse to grunde bliver grafen lavet mens programmet starter op, så den er tilgængelig allerede ved første rute søgning[5]. I det en ny edge bliver tilføjet til grafen vil dens retning blive taget i betragtning, og alt efter om den er ensrettet eller ej, vil den enten blive tilføjet til adj i begge retninger eller kun den ene[6].

DijkstraSP - ShortestRoad, FastestRoad

Klasserne FastestRoad og ShortestRoad, er stort set identiske. De afviger kun i den måde de sammenligner to Road objekter på, som sker i henholdsvis deres comparator og i metoden relax. For at undgå kodeduplikation er klassen DijkstraSP lavet til en abstrakt superklasse som lader sig extende af FastestRoad og ShortestRoad. DijkstraSP har derfor 2 protected abstrakte metoder, netop de to metoder som Shortest- og FastestRoad adskiller sig fra hinanden. DijkstraSP kan ikke i sig selv finde en rute mellem 2 punkter og det vil derfor ikke give mening at kunne lave en instans af denne.

Alle Road objekter består af 2 nodes samt en edge imellem disse. Det er ikke til at vide om vejen går fra node1 til node2 eller omvendt, derfor vil der blive lavet en masse checks på dette. Det kan blandt andet ses i linje 63 i FastestRoad og linje 151 i Linked. Hvis disse checks ikke blev lavet kunne man komme ud for at både start og slut node var den samme, og dette vil naturligvis skabe problemer.

I klassen ShortestRoad's comparator bliver der gjort brug af A*, og ved hjælp af pythagoras udregnes den heuristiske længde fra punktet til det ønskede slut punkt.

Metoden mapRoute i DijkstraSP udregner ruten mellem 2 givne punkter. Den tager 2 Road objekter som parametre, hvoraf den finde de involveret nodes unikke ID, som derfra bliver brugt til resten af processen.

Metoden returnerer en liste af Linked. Denne liste indeholder alt hvad der er brug for, for at lave en tekstbaseret rutevejledning. Listen er i omvendt rækkefølge, så det første element i listen, er vejen hvorpå slutpunktet er.

Som udgangspunkt ville det virke logisk at lade listerne adj og distTo i DijkstraSP være et HashMap med en Point som key, da hver node netop kan blive identificeret på deres koordinatsæt. Vi har dog erfaret af for de data mængder som vi har i brug, tager et HashMap alt for meget ram. Vi har løst problemet ved at lade

[5] Linje 73 i OSMparser og linje 75 i KrakParser
[6] Linje 141 i DijkstraSP

hver node få et unikt ID fra 0 og op, og bruge en ArrayListe med samme længde som antallet af nodes, hvor hver node har tilsvarende plads i listen som dets ID.

DistTo indeholder den akkumuleret værdi af punktet gemt i instanser af Linked. Fra start bliver alle pladser i listen sat til null, fordi, som det kan ses i analysen er det slet ikke alle punkter der bliver berørt, hvorfor det vil være meget ram spildt på ingen ting. Derfor bliver hver plads i ArrayListen først initialiseret til en Linked når der er brug for det, som det ses i både i Fastest- og ShortestRoad[7].

ArrayListen adj, havde oprindeligt en bag på hver plads. Vores Bag implantation, var opbygget som en linked list (ikke med klassen Linked), dette tog et hav af ram, hvorfor vi måtte lave det til en ArrayList. Problemet bestod i at hver del af den linked liste var et objekt for sig selv, og der gik en masse overhead ram spildt for hver instans.

Linked

Klassen Linked fungere som en linked list til at finde ruten til det givne punkt. Linked indeholder et felt from, som fortæller hvilken node denne kom fra, og ydermere den akkumuleret værdi. En ny instans af Linked bliver initieret med from = -1, samt drivetime og length til uendeligt[8]. Drivetime, length og from vil alle blive opdateret til de 'rigtige' værdier når den tilsvarende node pårørers. Dog vil det første item af den linked list have værdiere from=-1, length=0, drivetime=0. Det vil sige at det tidspunkt from er -1 vil man være nået det punkt man har startet søgningen i, hvis man "backtracker" listen for at få udskrevet ruten. Når en instans af Linked for sat sin edge, bliver svinget mellem den forrige edge og denne udregnet.

Turn

Enum klassen Turn indeholder de vejsvings muligheder der er. Det er hhv. højre, venstre eller ligeud. Turn bliver brugt i klassen Linked.

View package

View package står, meget indlysende, for View delen af MVC. Pakkens to vigtigste klasser er Canvas og SideBar. Canvas står for at tegne

Canvas

Canvas klassen er den største del af View, som står for at projektere kortdata til et billede på skærmen. Canvas tegner den data den får fra CurrentData. For at Canvas opdatere når Model ændres, bruges observer pattern, hvor Canvas er observer og CurrentData observable. Til det er der blevet lavet to interfaces: ObserverC og ObservableC.

[7] Hhv. linje 71 og 87
[8] Se analyse

Canvas udvider JComponent, så man kan tegne med et Graphics objekt og vise det på en JFrame. Udover det implementere Canvas, FocusListener, så controls kan skifte alt efter om det er SideBar (Beskrevet nedenfor) eller Canvas der har fokus.

Canvas står ligeledes for at dele kortet op i tiles, der gøres ved at have et integer 2D-array, der svarer til alle tiles. Hver plads i arrayet indeholder en nøgle til et HashMap med BufferedImage, som kan bruges til at hente i forvejen tegnede billeder. Hvis pladsen i arrayet derimod er 0, laves et nyt BufferedImage som lægges i HashMap'et. Når der zoomes ændres det samlede billede og et nyt 2D-array laves.

SideBar

SideBar klassen står for venstre side af GUI'en hvor det er muligt at søge efter en rute, og få printet denne rute i tekstform. Klassen indeholder to SearchAsYouType felter som er hhv. from og to. Disse to felter bruges til indtastning af vejnavne til ruteplanlægning. Der bliver tjekket for illegale karakterer i de 2 tekstfelter.

Ved søgning af rute gives en valgmulighed, om man vil finde den korteste rute eller den hurtigste. Som udgangspunkt er den hurtigste vej valgt.

Det der fylder mest i sidebaren er et stort textarea. Dette bruges til at udskrive den fundne rute i tekst. I koden[9] følger man den Linked liste hvoraf vejnavne og værdierne fås.

ObservableC

ObservableC en simpel implementation af observer designet. Klassen bruges til at gøre klasser observerbare sådan at observatørerne kan blive notificeret om ændringer.

ObserverC:

ObserverC er et interface som fungerer sammen med ObservableC. Klasser der implementerer dette interface kan observere andre klasser der arver fra ObservableC.

Model package

CurrentData:

CurrentData er den klasse i Model som resten af programmet interagerer med, den indeholder information om hvilken del af kortet der pt. skal tegnes og kan hente kortdata fra quad-træerne. CurrentData er som nævnt observable ved at implementere ObservableC. Dette gør at View delen bliver opdateret, når informationen i CurrentData ændres.

[9] linje 132 og 183

CurrentData indeholder fire quad-træer som indeholder veje der skal tegnes i hvert zoom niveau. Så når man er zoomet helt ud bliver der kun søgt i et quad-træ og når man er zoomet helt ind bliver der søgt i alle fire quad-træer.

Road:

Road klassen bruges til at indeholde information om et stykke vej, bestående af to knuder og en kant, samt vejstykkets midtpunkt. Det betyder den indeholder referencer til to NodeData og en EdgeData fra Kraks parser package, samt to doubles svarende til x og y koordinaterne for midtpunktet. Road objekter er den dataform vi har valgt at bruge til at opbevare vores vejdata, og det er derfor road objekter der indsættes i quadtræet. I quadtræet bruges midtpunktet til at bestemme hvilken quad en road tilhører.

Krakloader package

KrakParser

Klassen der står for at parse og opsætte dataen fra Krak. Til dette bruger den KrakLoader klassen og dens hjælpe klasser. Dataen bliver lagt i quad-træer alt efter deres type og og i en ArrayList med alle Road objekter.

Klassen LoadCoast er til for at hente kystlinjer ind, når kraks dataset bruges, da kystlinjer ikke er en del af kraks standard data.

Osmparser package

OSMParser

Klassen der står for at parse og omdanne dataen fra OpenStreetMaps dataen til det format der bruges i programmet. Der laves en SaxParser hvorfra der kan dannes en XMLReader. Til XMLReaderen bruges SaxHandler til at fortæller hvordan den skal håndtere input, i dette tilfælde OpenStreetMap dataen.

Fra XMLReaderen fås en ArrayList med NodeData objekter, en List med Way objekter og en Rectangle2D tilsvarende området hvori kortet ligger.

Way objekterne bliver så omdannet til Road objekter og lagt i deres tilsvarende quad-træ, alt efter typen af vejen, samt en ArrayList med alle Road objekter.

SaxHandler

SaxHandler udvider DefaultHandler og er dermed en klasse der styrer hvad der skal ske ved forskellige XML begivenheder, så som starten på et nyt element.

SaxHandler er lavet til at indlæse OpenStreetMap data i XML format og gemme det som Way og NodeData objekter, i hver sin ArrayList, der sendes til OSMParser instansen.

Way

Et Way objekt er lavet til at indeholde dataen fra et way element i OpenStreetMap dataen. Den indeholder navn og type for vejen, samt reference til alle knuder på vejen. Grunden til at der er flere knuder er at OpenStreetMap bruger et format hvor en vej er opbygget som en liste af referencer til knuder i den rækkefølge de forbindes.

Ctrl package

StartMap

Denne klasse indeholder main metoden og står for at bygge programmet op. Den bruger en JOptionPane til at få brugerens valg mellem OpenStreetMap og Krak. Derefter henter den dataen ind gennem den relevante parser og sætter model op, for bagefter at sætte GUI, View og Control op. Ydermere indeholder den forskellige lister i dens felter som er afgørende for mange andre klasser.

ML

Denne klasse implementerer: MouseListener, MouseMotionListener, MouseWheelListener. ML spiller den største rolle i Ctrl pakken, da meget af interaktionen mellem bruger og programmet foregår mellem med. Hver gang en af de tre event typer opstår, opdateres CurrentData.

Vi har oplevet en race condition med MouseWheelListener, som fik scroll zoom til at bugge, når man scrollede for hurtigt Vi har ikke kunne finde ud af hvilken variabel og hvor der blev ændret, så i stedet har vi fundet en løsning med en timer. Ved at sætte et delay på 10 ms før ML må beregne et nyt zoom niveau, undgår vi den race condition der opstår ved hurtig scrolling. Samtidig er 10 ms kort nok til ikke at blive bemærket.

KL

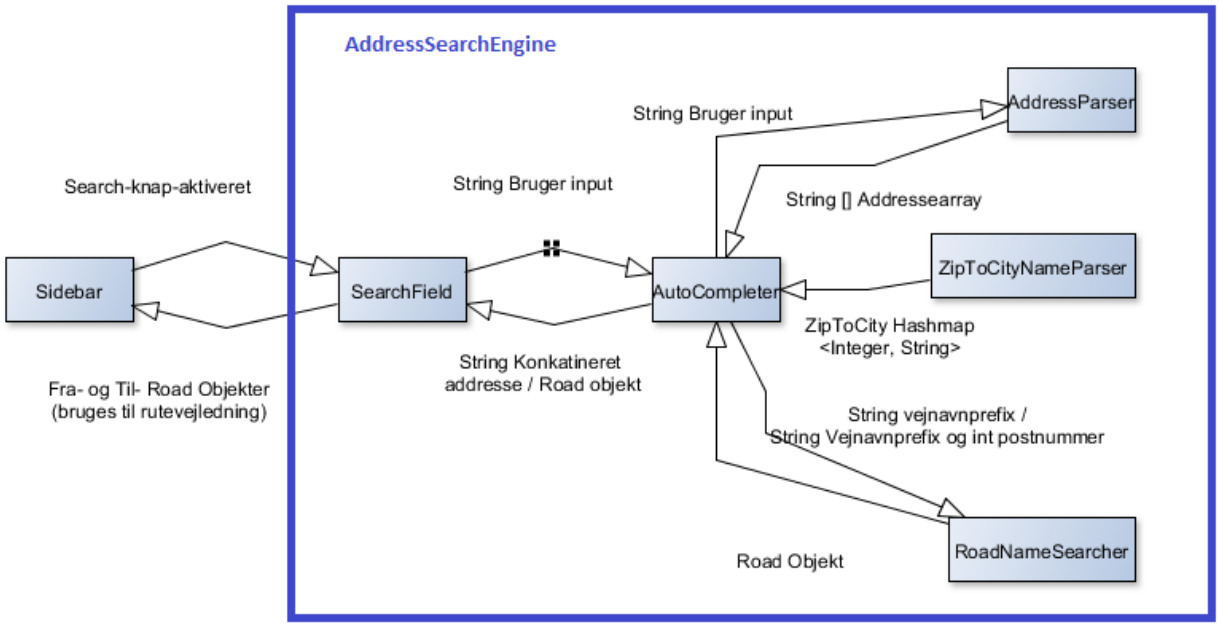
Denne klasse implementerer: KeyListener. KL bruges både til Canvas og SearchField og styres af hvilken del af programmet der sidst er trykket på (har fokus). KL har metoder der kan få Canvas til at panne 2% op, ned, højre eller venstre ved hjælp af piletasterne, hvis Canvas har fokus. KL er også med til search-as-you-type funktionaliteten da den tjekker hvad der står i søgefeltet, og kalder autoComplete hvis der tilføjes tekst i søgefeltet.

CanvasML

Denne klasse implementerer: MouseListener. CanvasML's rolle er at give canvas en måde at bede om fokus i programmet, sådan at tastatur panning kun virker når canvas har fokus. Interaktive jcomponents har normalt en indbygget måde at bede om fokus på passende tidspunkter, canvas arver imidlertid ikke interaktivt funktionalitet, og har derfor brug for en måde at bede om fokus, når der klikkes på canvas. Fokus er vigtigt for canvas på grund af at tastatur panning ikke skal foregå når canvas ikke har fokus.

AddressSearchEngine package

AddressSearchEngine package indeholder klasser til vejsøgning fra bruger inputtet:



Figur 9: Illustration af Search-as-you-type klassestruktur og dataudveksling

SearchField:

Searchfield extender JTextField og implementerer FocusListener. Klassen står for at modtage brugerinput og har front-end rollen for vejnavnssøgning samt search-as-you-type funktionen. To instanser af klassen bliver lavet i Sidebar; én fra-destination og én til-destination.

AutoCompleter:

Klassen AutoCompleter bruges til seach-as-you-type funktionaliteten. Klassen har tre formål: At fuldende bruger adresseinputtet og returnerer denne til SearchField, at være en opdeler i søgemetodekald, hvis inputtet indeholder vejnavn eller vejnavn og postnummer, og at være bindeled mellem SearchField, AddressParser, ZipToCityNameParser og RoadNameSeacher.

AddressParser:

Klassen AddressParser står for parsing af brugerens adresseinput, dvs. den modtager et tekstinput af hele adressen brugeren har indtastet, og returnerer et adressearray med forudbestemte pladser for vejnavn, husnummer, husbogstav, etage, postnummer og bynavn.

RoadNameSearcher:

Klassen RoadNameSearcher står for søgningen af et vejnavn eller vejnavn og postnummer i listen af allRoads, som er en ArrayList af alle roads parset fra datasættet af OSM eller Krak.

ZipToCityNameParser:

ZipToCityNameParser er en singelton og står for parsingen af dokumentet bynavne.txt til et HashMap, der mapper 4-cifrede postnumre til bynavne. Denne klasse eksisterer fordi der ikke findes bynavne i hverken OSM eller Krak datafilerne. Klassen kan returnerer dette HashMap, som kan bruges til opslag af postnumre til bynavne i AutoCompleter.

AddressSearchEngine.UnusedCode package

Denne pakke indeholder en ikke færdig implementation af en Trie (SearchTrie) til vejnavnssøgning samt klassen AlphabetParser til parsing af tekstfilen alphabet.txt, som skulle bruges til Trien.

Afprøvning

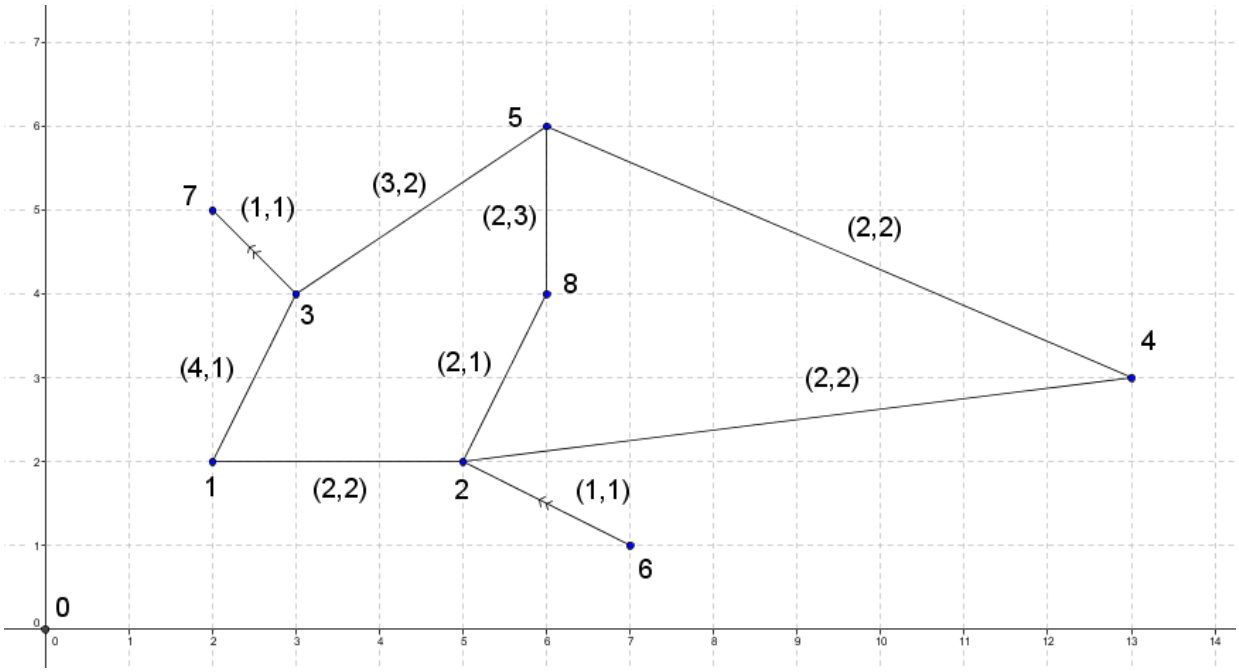
Ruteplanlægning test:

ShortestRoad og FastestRoad er opbygget på præcist samme måde, og deler endda nogle metoder. Som beskrevet i analysen adskiller de sig kun i de værdier de sammenligner på. Hvor ShortestRoad sammenligner vejstykker på deres længde, sammenligner FastestRoad på deres køretid. Det vil sige at de særtilfælde der er vedrørende blandt andet ensretning må gælde ens for begge klasser, og vil ikke blive testet i begge tilfælde.

Vi vil lave en unittest af ruteplanlægningen, hvor vi vil sammenholde den endelige rute med den forventet. Ydermere vil vi skridt for skridt med NetBeans debugger følge om algoritmen opfører sig som ønsket. Det kan sagtens være at den finder den korteste vej, men det er ikke sikkert at den gør det mest optimalt.

Der vil ikke blive lavet en komplet whitebox test med forventnings tabeller, da vi har valgt at lægge fokus på at algoritmen opfører sig optimalt og hensigtsmæssig, fremfor at teste om den virker.

For bedst at kunne følge processen og sammenholde med forventninger har vi lavet et testmiljø i form af en bruger defineret graf:



Figur 10: Illustration af test graf. Hver edge har 2 værdier (p,q) hvor p er længde og q er køre tid. Vi antager vi skal fra punk 1 til 5

Bare ved at kigge på illustrationen, og koden kan vi hurtigt sætte nogle forventninger op:

a) Den hurtigste vej tager 3 tids enheder og er ruten: 1, 3, 5.
b) Den korteste vej tager 6 længde enheder og er ruten: 1, 2, 8, 5.
c) Punkt 6 burde aldrig blive berørt, da 6 er ensrettet fra 6 til 2.
d) I den korteste vej vil punkt 4 vil blive prioriteret under punkt 8 da kanten fra 2-4 plus den heuristike værdi er $2 + \sqrt{7^2 + 3^2} = 9{,}6$, hvor punkt 2-8 er $2 + \sqrt{0^2 + 2^2} = 6$.
e) Punkt 7's arrayliste i adj er tom, da vejen er ensrettet og slutter i ingen ting
f) Den korteste vej burde bestå af svingene: Ligeud, venstre, venstre
g) Den hurtigste vej burde bestå af svingene: Højre, højre

Vi har ved hjælp af JUnit sat en test klasse op med denne grafen fra illustrationen. Udover hvad der ses på illustrationen indeholder JUnit koden yderligere et punkt og en kant, punkt 9 som ligger i koordinatsystemet (6,7), og kanten som går fra 5 til 9. Grunden til denne, er at koden er sat til at stoppe så snart at enten start- eller slut punktet er nået. Så for at sikre vi først stopper i præcist punkt 5, har vi brug for denne "usynlige" kant og punkt.

Vi har testet disse 7 punkter, og alle stemte overens med forventningerne. Ydermere opførte algoritmen sig hensigtsmæssigt, og arbejdede som den skulle. Grafen til testen er kort og præcis, og burde alle de tilfælde man kan komme ud for.

Den opstillede graf er dog lige på grænsen til ikke at virke med A*. Det skyldes at for A* virker skal der være et fast forhold mellem den faktisk afstand af 2 punkter, og den givne afstand. Hvis vi kigger på illustrationen ser vi at afstanden mellem punkt 2 og 8 er 2. Samtidigt er afstanden mellem 2 og 4 ligeledes 2, selvom man med det blotte øje kan se at der burde være længere. Dette bryder algoritmen, for havde afstanden mellem 2 og 4 nu være 1, ville den vej rundt faktisk være hurtigere, men den ville ikke blive set på grund af den heuristiske værdi. Altså skal der være en fast overensstemmelse mellem den givne længde og den længde man selv kan udregne med Pythagoras.

Da der nu er opsat en automatisk test i form af JUnit test, vil man i fremtiden stadig kunne følge om algoritmen virker efter hensigt hvis ændringer bliver lavet.

Whitebox test af Quadtræet

Quadtræet er den underliggende datastruktur for mange af funktioner i programmet, og som sådan er det en vital del, som det er vigtigt fungerer korrekt. Derfor har vi valgt at teste quadtræets funktionalitet. Dette gør vi ved at lave test over metoderne "insert" og "getRoads" fra klassen Quadtree. Igennem disse metoder bliver alt vigtig underliggende kode kaldt, og det er derfor ikke nødvendigt at lave mange underliggende tests, hvis vi kan påvise at disse to metoder fungerer som forventet.

Quadtræets implementation indeholder ikke beskyttelse mod indsættelse af mange veje i samme punkt, hvilket betyder at quadtræet vil crashe hvis flere end tusind veje der ligger i præcis samme punkt bliver indsat. Det er en kendt begrænsning, og vi har derfor ikke lavet test omkring dette.

Første del er whitebox test af insert metoden i klassen QuadTree. Gennem testene bliver kode omkring koordinater, fra klassen Boundary, ligeledes testet. Test foregår ved hjælp af JUnit.

Expectancy table[10]:

Input data set	Contents	Expected output	Actual output	Outcome
A	1 road inside bounds	1 road in undivided quadtree	1 road in undivided quadtree	Passed
B	1 road outside bounds	empty quadtree	empty quadtree	Passed
C	2000 roads inside bounds divided into the first 4 divideable sectors	divided quadtree with the correct 500 roads in each quad	divided quadtree with the correct 500 roads in each quad	Passed

Testene blev bestået, og man kan gennem testene se at veje indsættes korrekt i quadtræet. Testene kører med små dataset og det er derfor svært at sige noget konkret omkring hvad der sker ved virkelig store dataset, ud over at det visuelt ser ud til at virke i system testene fra selve programmet. Det burde dog ikke være nødvendigt at teste med større dataset end der er, for at se om koden virker.

Whitebox test af metoden getRoads i klassen QuadTree. Nogle test her er uden JUnit og har i stedet foregået ved hjælp af debugger i netbeans. Dataset der hedder "a.X" har implicit hvad der står som "contents" for A, dvs. 2000 veje i quad-træet.

[10] Se bilag Quadtreetest 1 for coverage table og testet kode

Expectancy table[11]:

Input data set	Contents	Expected output	Actual output	Outcome
A	2000 roads in quadtree	Sub-quads resulting in boolean true.	Sub-quads resulting in boolean true.	Passed
B	999 roads in quadtree	1 - false. 7 - more than once. 8 - true. With correct behavior	1 - false. 7 - more than once. 8 - true. With correct behavior	Passed
a.C	Search area touching both northeast and southeast	Roads from northeast and southeast totalling 1000	Roads from northeast and southeast totalling 1000	Passed
a.D	Search area touching only northwest	Roads from northwest quad totalling 500	Roads from northwest quad 500	Passed
a.E	Search area outside quadtrees coverage	No roads	No roads	Passed
a.F	Search area touching only southwest	Roads from southwest quad totalling 500	Roads from southwest quad totalling 500	Passed
H	Search in quadtree with no roads	No roads result in check after roads and nothing found	No roads result in check after roads and nothing found	Passed
I	Search in quadtree containing 1 road	Path triggered and correct behavior	Path triggered and correct behavior	Passed

Det er svært at være sikker på om alle mulige søge tilfælde fungerer korrekt, men de generelle søge situationer som alle andre tilfælde vil være versioner af, er testet, og det mener vi giver en god chance for at de andre tilfælde ligeledes virker. Her er der tale om at det ikke er testet hvis et søgeområde f.eks. rører ved 571 forskellige quads, i stedet er der testet for søgeområder der rører henholdsvis 1, eller 2 quads.

[11] Se bilag Quadtreetest 2 for coverage table og testet kode

Whitebox test af AddressParser

Vi har valgt at teste address parseren, fordi det er en vital del af programmet, at brugeren skal kunne indtaste en adresse, med vejnavn, husnummer osv., som bliver ekstraktet ordentligt, så den rigtige vej findes ved søgning.

I white box testen har vi valgt at teste alle adress parserens metoder, da hver sin metode parser sin del af adresse input String. Vi gør brug af brach coverage white box testing, hvor vi for hvert "if" og "else if" statement har angivet et nummer, som vi tester for true og false værdier[12]. Efter coverage table blev lavet, var der stadig åbenlyse input Strings vi synes, der skulle testes for. Disse er angivet med nøgleordet: "Special". Testene blev skrevet i en JUnit test klasse, som med fordel kan køres igen ved ændringer i koden.

Herunder ses resultaterne af testene:

Expectancy table[13]**:**

Input data set	*Input:*	*Expected output*	*Actual output*	*Outcome*
A	”Rued Langgaards Vej 7 2300”	{“Rued Langgaards Vej”, “7”, “”, null, “2300”, null}	{"Rued Langgaards Vej", "7", "", null, "2300", null}	(Passed) **
B	“Rued Langgaards Vej”	{“Rued Langgaards Vej”, null, null, null, null, null}	{"Rued Langgaards Vej", null, null, null, null, null}	Passed
C	“Rued Langgaards Vej7 2300”	{“Rued Langgaards Vej”, null, “”, null, “2300”, null} *	{"Rued Langgaards Vej", null, "", null, "2300", null} *	(Passed) **
D	“Rued Langgaards Vej ”	{“Rued Langgaards Vej”, null, null, null, null, null}	{“Rued Langgaards Vej”, null, null, null, null, null}	Passed
E	“Rued Langgaards Vej 7”	{“Rued Langgaards Vej”, “7”, null, null, null, null}	{"Rued Langgaards Vej", "7", null, null, null, null}	Passed
F	“Rued Langgaards Vej 7A”	{“Rued Langgaards Vej”, “7”, “A”, null, null, null}	{"Rued Langgaards Vej", "7", "A", null, null, null}	Passed
G	“Rued Langgaards Vej 7 5. Sal”	{“Rued Langgaards Vej”, “7”, null, “5”, null, null}	{"Rued Langgaards Vej", "7", null, "5", null, null}	Passed

[12] Se bilag AddressParserTest 1
[13] Se bilag AddressParserTest 1 for Coverage tables

*H	“Rued Langgaards Vej 7 København S”	{”Rued Langgaards Vej”, ”7”, null, null, null, ”København S”}	{”Rued Langgaards Vej”, ”7”, null, null, null, null}	Failed
I	“Rued Langgaards Vej 7 2300 København S”	{“Rued Langgaards Vej”, “7”, “”, null, “2300”, København S}	{“Rued Langgaards Vej”, “7”, “”, null, “2300”, København S}	(Passed) **
J	“Rued Langgaards Vej 7, København S”	{”Rued Langgaards Vej”, ”7”, null, null, null, ”København S”}	{”Rued Langgaards Vej”, ”7”, null, null, null, ”København S”}	Passed
K	“Rued Langgaards Vej 7, 2300 København S”	{”Rued Langgaards Vej”, ”7”, ””, null, ”2300”, ”København S”}	{”Rued Langgaards Vej”, ”7”, ””, null, ”2300”, ”København S”}	(Passed) **
L	”Rued Langgaards Vej, 2300 København S”	{"Rued Langgaards Vej", null, "", null, "2300", "København S"}	{"Rued Langgaards Vej", null, "", null, "2300", "København S"}	(Passed) **
M	“Rued Langgaards Vej, 2300 København S”	{”Rued Langgaards Vej”, null, ””, null, ”2300”, ”København S”}	{”Rued Langgaards Vej”, null, ””, null, ”2300”, ”København S”}	(Passed) **
N	“Rued Langgaards Vej,”	{”Rued Langgaards Vej”, null, null, null, null, null}	{”Rued Langgaards Vej”, null, null, null, null, null}	Passed
Special1	“Rued Langgaards Vej 7 5.”	{“Rued Langgaards Vej”, “7”, null, “5”, null, null}	{“Rued Langgaards Vej”, “7”, null, “5”, null, null}	Passed
Special2	“Rued Langgaards Vej 7 5”	{“Rued Langgaards Vej”, “7”, null, 5, null, null}	{“Rued Langgaards Vej”, “7”, null, null, null, null}	Failed
Special3	“Rued Langgaards Vej, 2300”	{“Rued Langgaards Vej”, null, “”, null, “2300”, null}	{"Rued Langgaards Vej", null, "", null, "2300", null}	(Passed) **
Special4	“Rued Langgaards Vej, København S”	{”Rued Langgaards Vej”, null, null, null, null, ”København S”}	{"Rued Langgaards Vej", null, null, null, null, "København S"}	Passed

Special5	"Rued Laanggards Vej 7A, 5. sal 2300 København S"	{"Rued Langgaards Vej", "7", "A", "5", "2300", "København S"}	{"Rued Langgaards Vej", "7", "A", "5", "2300", "København S"}	Passed
Special6	"Rued Laanggards Vej 7A 5. sal 2300 København S"	{"Rued Langgaards Vej", "7", "A", "5", "2300", "København S"}	{"Rued Langgaards Vej", "7", "A", "5", "2300", "København S"}	Passed

*NB: "7" is part of the street name and thereby considered as a part of street name, not as a building number.

**NB: Unexpected event discovered -If an address contains a zip-address E.G: "Rued Langgaards Vej 2300" – Our parser returns: {"Rued Langgaards Vej", null, "", null, "2300", null}, instead of: {"Rued Langgaards Vej", null, null, null, "2300", null}. Instead of making all tests with a zip address fail, we expect an empty String "" from building number when we parse an address containing a zipcode, and null when the address does not contain a zipcode. This is strictly not "passed", but for most of our tests, this is not what the test is made for, but we are aware of this bug in our code.

Vi tester ikke for en tom streng input: "", fordi vi i en anden klasse (SidePanel) tjekker dette før parsing af input String begynder.

Under white box testen fandt vi små fejl i koden som efterfølgende blev rettet, og JUnit testene blev kørt igen.

Ved disse tests kan vi argumentere for, at vores address parser virker for størstedelen af legitime adresse brugerinput. Dog har vi visse adresse input der stadig giver fejl[14].

System test

Her gennemgås alle funktionaliteter i programmet og hvor godt de virker samt eventuelle mangler i dem.

Funktion	**Test**
Vælg mellem krak eller osm dataset	Fungerer uden problemer
Scroll Zoom	Zoomer korrekt imod musen, men hvis der scroll zoomes ved maksimum zoom, bliver der lavet en form for pan operation i stedet.
Firkant Zoom	Der zoomes korrekt på de markerede områder.

[14] Se H og Special2

Zoom reset (højre museklik)	Fungerer uden problemer
Panning	Fungerer uden problemer
Nærmeste vej	Virker umiddelbart korrekt, men da især kraks dataset indeholder mange veje uden navn, kan der opleves forvirrende opførsel.
Piletast panning	Virker korrekt. Der kan kun pannes når kortet har fokus, men dette er intentionelt.
Autocomplete på adresser	Virker, dog ikke så intuitiv. Autocomplete funktionen læser efter store og små bogstaver, og det er derfor nødvendigt at skrive stort R i f.eks "Rued" for at få Rued Langgaards Vej. Der kan opleves problemer som vi ikke ved hvad skyldes, som f.eks. at skriver man "Ru" sker der ikke autrocomplete, mens f.eks. "ru" virker.
Vejfinding	Vores OSM fil indeholder ikke information om ensretninger, hvilket kan skabe ruter hvor man må køre imod ensretningen. Ydermere har rundkørsler heller ikke en retning, hverken i krak eller osm. Det betyder at den fundne rute kører imod rundkørslen hvis det er den hurtigste/korteste vej. Nogle få veje har ikke et navn, så den tekstbaseret rute vil vise blankt navn.

Ud over at teste specifikke funktioner i programmet, har vi ydermere lavet en overordnet gennemgang af programmet. Gennem denne gennemgang fandt vi en bug og muligvis dårlig funktionalitet. Lange veje som f.eks. færgeruter kan nemt komme ud for at blive klippet over på kortets visuelle repræsentation. Dette skyldes at vi har gemt veje ud fra deres midtpunkt i quadtræet, og der kan derfor opstå situationer hvor veje ligger i en anden quad, men stikker langt ind i de omkringliggende quads, hvilket leder til at vejen ikke bliver tegnet korrekt hvis alle de pågældende quads ikke bliver loadet. Dette kan løses ved at lægge veje i alle de quads de berører så de altid bliver tegner i alle tiles de berører. Dette ville kræve en lidt større gennemgang af programmet, da der ydermere skulle tjekkes for at en vej ikke blev returneret/brugt flere gange til samme, hvilket ville gøre programmet langsommere.

Ud over denne bug, fungerer programmet som forventet, dog kan det måske betragtes som en bug/dårlig funktionalitet at man kan zoome uendelig meget ud, og panne uden for kortet.

Samarbejdsværktøjer

Kommunikation: Facebook, Skype

Som primær kommunikation har vi brugt Facebook. Her har vi haft en gruppe, hvor vi kunne dele idéer, planlægge møder, eller diskuterer implementationsidéer når vi var fraværende. Da ikke alle gruppemedlemmer tjekker deres Facebook dagligt, viste denne kommunikationsform svagheder i forhold til hurtig kommunikation.

Skype blev brugt til digitale møder, hvis vi skulle diskuteres eller planlægge ting, som tog for lang tid at skrive over Facebook. Skypemøderne blev ligeledes flittigt brugt til digitale aftenmøder mellem gruppens medlemmer.

Web baserede samarbejdsværktøjer: Github, Google Docs

I starten af projekt blev koden delt over Facebook og Dropbox. Da vi blev introduceret for Github, synes vi, at denne løsning lød interessant, og vi var straks enige om at udnytte muligheden. Vi fik mange konflikter, da vi i starten arbejdede meget i samme klasser, hvilket tog lang tid at løse.

Vi har brugt meget tid på at slås med Github, men meget af tiden har vi fået igen. I det store hele er vi glade for at have brugt Github til versionsstyring af koden. For fremtidige projekter vil vores viden om Github klart komme os til gode.

Google Docs har vi alle brugt til tidligere projekter, og kom os klart til fordel lige fra starten af projektet. Med Google Docs har vi kunnet skrive afsnit af samme rapport, samtidig over internettet, og give kommentarer til hvad andre har skrevet. Vi fandt desuden på, at skrive med hver vores skriftfarve, således at vi kunne se hvem, der har skrevet hvad i rapporten. Vi har brugt dette fantastiske værktøj til rapportskrivning, logbog og worksheets.

Med Google Docs har vi sparet meget tid og besvær. Alle gruppens medlemmer var godt kendt med Google Docs og vi vil ligeledes bruge samme værktøj til fremtidige projekter.

Planlægning og arbejdslog: Logbog, Worksheets, Trello

Logbogen har, af størstedelen af gruppens medlemmer, været set som dokumentation af gruppens arbejde, og ikke som et godt planlægningsværktøj. Har en person været syg eller på ferie, har der været mulighed for at læse hvad de andre har lavet i logbogen. Vi har ikke udnyttet det potentiale gruppe logbogen kan have, fordi vi synes værktøjet var for tidskrævende og besværligt i forhold til udbyttet. I stedet fandt vi planlægningsværktøjet Trello, som fungerede hurtigere, mere overskueligt, og som var mere bekvemt at bruge.

Worksheets er blevet skrevet ugentligt og bagudrettet, som dokumentation for hvad gruppens medlemmer har lavet, eller er i gang med. Dette har skabt et overblik over gruppemedlemmernes arbejdsindsats, og deres nuværende projekt.

Den praktiske funktionalitet ved dette værktøj blev ligesom logbogen erstattet af Trello, fordi Trello gav mulighed for samme funktionalitet, hurtigere, mere overskueligt, og var mere bekvemt at bruge. Vi har dog fortsat med at skrive worksheets fordi de var et krav til projektet.

For at få større overblik over vores arbejde valgte vi at tage et planlægningværktøj i brug kaldet Trello[15]. I Trello har man en tavle for hvert projekt man er med i, og på tavlen kan medlemmer lægge opslag op, om diverse statusopdateringer, checklister eller mødetider. Værktøjet er meget brugervenligt og fyldt med smarte funktioner der gør det hurtigt og nemt at informere hele eller dele af gruppen, når der sker noget nyt, samt holde styr på deadlines, opgaver folk arbejder på, og er blevet færdig med.

Da samarbejdsværktøjet blev introduceret til gruppen, blev det hurtigt accepteret som primært planlægningsredskab. Trello har sparet os tid og besvær, og har styrket planlægningsevnen i gruppen. Gruppens medlemmer er enige om, det er værd at udnytte Trello som planlægningsværktøj til fremtidige projekter.

Gruppeanalyse

Gruppen er ikke selvvalgt, hvorfor stridigheder og uenigheder kan forestilles at opstå på grund af forskellige opfattelser af arbejdsformer og tider.

Vi har forebygget uenigheder ved at lave en samarbejdsaftale, ved stiftelsen af vores gruppe. Denne samarbejdsaftale indeholder netop 'reglerne' og ambitionerne for gruppearbejdet som vi alle var enige om. Dette har uden tvivl forebygget frustrationer, der kunne opstå i uaftalte områder.

Vores gruppe har bestået af personer med et forholdsvis ens fagligt niveau inden for programmering. Dette har betydet at alle har kunnet være med til at diskutere og finde på løsninger til de forskellige problemer, og ydermere at man har kunnet hente hjælp i gruppen hvis man sad fast i noget kode. Alle har haft mulighed for-, og har lavet en god del kode hver især, pga. af det lige faglige niveau, hvilket uden tvivl har sparet os for frustrationer. Hvis en person var langt over de andre, kunne man forestille sig at han ville blive tvunget til et lavere niveau end han har lyst til.

Overordnet set har der ikke været nogle konflikter i gruppen, og generelt har vi været ret enige om hvilken retning projektet har skulle tage. Så gruppen kan konkluderes at have fungeret ganske udmærket.

Der har dog været nogle små irritationsmomenter, og problemer begge i form af manglende kommunikation.

Den manglende kommunikation har mest vedrørt hvordan de klasser man har skrevet virker og hvad de gør. Dette har medført at dele af programmet, man ikke selv har været inde over, har været meget ukendte for en. Så oftest når man har skulle starte på at implementere noget nyt, har der været en del man har skulle forstå hvordan hænger sammen for at kunne komme i gang, hvilket nok har kostet en del

[15] https://trello.com/

tidsmæssigt. Hertil ville interfaces f.eks. have været en god løsning, da gruppen ville kunne lave et interface sammen, hvorefter et gruppemedlem kunne implementere det. Dette kunne have hjulpet med at gøre projektet mere flydende og generelt have givet et bedre produkt i sidste ende, da alt ville være implementeret ud fra en fælles vision.

Den anden side af manglende kommunikation er i form af vores møder og mødetidspunkter. Vi havde et problem med at folk kom for sent til møder uden at give lyd fra sig eller at det tog lang tid at komme i kontakt med hele gruppen, gennem den aftalte kommunikationsform, Facebook. Dette har der været arbejdet meget med og generelt har folk været væsentlig bedre til at oplyse resten af gruppen, hvis de var forsinkede eller til at besvare spørgsmål fra andre gruppemedlemmer. Vi har ydermere suppleret Facebook kommunikationen med sms, som gav bedre resultater for kommunikationen.

Så gruppearbejdet har haft nogle problemer, der hovedsageligt kommer af sammensætningen af gruppen. Det har ikke været et samarbejde fyldt med konflikter, men oftest er der nok blevet spildt tid pga. manglen af organisation og kommunikation. Til fremtidige projekter, kan man tage med at det er vigtigt at være opmærksom på hvilke typer personer der er blevet sat sammen, så man kan undgå forskellige problemer.

Dertil er det vigtigt at kunne udnytte de evner der er til rådighed, hvilket er blevet gjort godt ved at fordele opgaverne og ellers lade folk arbejde, der har bare manglet nogen til at binde det sammen.

Det andet problem ligger i typen af personer og er muligvis en stor medspiller i problemet med kommunikationen. Gruppen har bestået af personer der er bedst til at få en opgave og så arbejde med den. Dette har betydet at der har manglet en eller flere personer til at tage styringen og sørge for at der blev organiseret og diskuteret.

Der har været forsøg på at tage styringen og skabe noget kommunikation, men da det ikke er noget der falder nogen i gruppen naturligt, er det hurtigt gået i sig selv igen, indtil næste forsøg. Det betyder at ofte har folk valgt noget at arbejde med og så siddet med det. Det medført små ting, der skabte problemer, som så skulle tages hånd om, hvilket godt kunne gå meget langsomt, da der sjældent har været oversigt over projektet. Dette medførte, at medlemmerne i gruppen har "arbejdet på tværs af hinanden", således at to i gruppen har kommet til at arbejde på samme opgave samtidig, hvilket ligeledes har skabt problemer og tidsspilde.

Det er dog ikke kun negativt med fire personer der arbejder godt når de har fået en opgave. Generelt blev størstedelen af forskellige implementationer lavet ret hurtigt, hvilket, trods manglen på organisation og kommunikation, har gjort at vi har bevæget os fremad i et godt tempo.

Konklusion

Vi har kreeret et program som giver en grafisk præsentation af Danmark. Vi blev stillet en række krav til hvad vores program skulle kunne, hvilket vi har opfyldt efter bedste evne. Vores program fungerer efter hensigten, og kører forholdsvist hurtigt og flydende. Vi har så vidt muligt bestræbt os på at skrive en kode der har en logisk struktur og er vedligeholdelsesvenligt, omend det ikke er blevet helt som vi ville have ønsket det.

Vi har lavet en række test af programmet for at kunne blive overbevist om at programmet virker tilnærmelsesvist fejlfrit. Vores test har kørt med tilfredsstillende resultater, og vi har opbygget en tiltro til at vores kode fungerer efter hensigten.

Vi blev sat i ikke selvvalgte grupper, og vi har gjort vores bedste for at arbejde som enhed med de værktøjer vi har lært om gruppearbejde. Vores gruppe arbejde har fungeret uden store konflikter.

Vores program har få mangler og kendte problemer:

Ydermere har vi haft nogle ambitioner om forbedringer der ikke er blevet realiseret på grund af tidsmangel. De er som følger:

En mere vedligeholdelsesvenlig og modulær kode, som vi fx. er begyndt på med DrawInterface, hvor tanken var at være uafhængig af en type Graphics.

Der er mange forbedringer der kunne have været implementeret i koden, der ville have gjort programmet hurtigere og mere stabilt. Det drejer sig blandt andet om:

Bedre søge former i quadtræet, som f.eks. kun at søge i bunden af træet, samt en grænse på dybden i træet der ville have sikret imod stackoverflow exceptions ved indsættelse af flere end tusind veje på samme punkt.

Ved indlæsning af veje fra datasættene, kunne der ligeledes have været lavet forbedringer. Lange veje kunne være blevet delt op så de passer ind i deres quads, eller quads kunne indeholde alle veje der rører dem. Ved indlæsning af osm data, har vi en anden ikke optimal løsning, i det at vi deler osm vej objekter op i mindre dele for at få det til at passe med kraks format. Det leder til enormt mange vej objekter med osm's data, der allerede er et meget stort datasæt. Samtidig skal alle dataen fra OSM løbes igennem igen, for at dele dem op i krak format, hvilket resultere i en ekstra lang loading tid.

Diskussion

Projektet er gået udemærket, vi har nået hvad vi skulle i forhold til kravene til projektet, men planlægning har været et svagt punkt i vores gruppearbejde. Hvis planlægningen havde fungeret på et bedre niveau, kunne gruppen sandsynligvis have nået, at lave et mere færdigt program med flere features. Det har også hængt sammen med, at vores initielle ambitionsniveau ikke var sat særligt højt, at vi ikke ville have at projektet skulle gå ud over andre fag, og der derfor blev planlagt herefter. Der blev ikke sat noget mål for

projektet, hvorfor ingen delmål blev sat. Dette har haft konsekvensen at ingen deadlines er blevet sat, gruppens medlemmer måtte selv vælge hvad de ville arbejde på, og i hvilken hastighed de ville arbejde i, og projektet er derfor gået langsommere frem end det kunne.

For at imødegå dette problem kan man i fremtidige projekter sætte et lidt for ambitiøst mål til at starte med, der kan justeres senere, og derefter planlægges og have deadlines ud fra det, så arbejdet får nogle faste rammer og et klart mål, for det gruppen vil opnå.

Kodning i gruppen kunne have fungeret bedre, f.eks. ville det have været godt at komme til enighed om, hvordan koden skulle udformes, blandt andet ved brug af interfaces, som et planlægningsværktøj. Derved kunne koden have været skrevet så modulært, at der ikke behøves nogen viden omkring den indre virken af andre dele af koden, for at arbejde med dem.

Selve koden er ikke blevet så struktureret og vedligeholdelsesvenligt som vi kunne have tænkt os. Selvom vi i begyndelsen snakkede om at det var noget vi ønskede, faldt det hurtigt lidt fra hinanden pga. den manglende kommunikation. Dog har vi alligevel med diverse design patterns haft ambitioner og ønske om en logisk struktur. Der kan argumenteres for at netop fordi vi hver især har haft fokus, og kun fokus på vores egen del af koden, har vi opnået et relativt godt abstraktionsniveau i koden.

Klassen SearchTrie ville vi gerne videreudvikle og implementerer i programmet, men blev desværre i tidsnød i sidste del af projektet. Vi har dog valgt at inkludere ikke færdiglavet kode i klassen SearchTrie i pakken AddressSearchEngine.UnusedCode, da vi gerne vil snakke om teorien bag disse klasser til eksamen også, selvom der ikke laves en instans af klassen ved programkørsel, og denne klasse derfor ikke har nogen effekt på programkørsel.

Vi har testet forskellige vigtige dele af programmet, samt en systemtest af hele programmet. Umiddelbart har vi ikke fundet nogle større fejl i programmet. De fejl der er blevet fundet har ikke nogen stor indflydelse på hvor godt programmet virker, men er mere kosmetiske.

Ud fra vores test af programmet er vi nået dertil hvor vi stoler på at programmet virker. Fejl som vejfinding der kører venstre om en rundkørsel, skyldes mere mangel på detaljer i datasættene, end de skyldes koden i programmet. Vi kan heller ikke udelukke at der er legitime adresse input formater som vi ikke har testet for. Vi kan dog argumenterer for, at vores program virker for størstedelen af input adresse formater.

Vi har generelt ordnet bugs efterhånden som de er blevet fundet, men først til sidst har vi JUnit test. Indtil da har vi lavet test ved at observere kodens opførsel, samt brug af debugger i netbeans. Vi føler ikke at det ville have været en stor fordel at skrive tests tidligere, da bugs der har haft betydning for udviklingen af koden, er blevet fundet gennem almindelig programkørsel og ved debugging. Der blev fundet store fejl gennem JUnit test, men det var ikke fejl der som sådan ville have ændret noget for udviklingen af programmet. F.eks. mistede quadtræet hver 1001'ne vej, men fungerede ellers efter hensigten.

Ressourcer

- OpenStreetMap data for Danmark er hentet fra www.geofabrik.de (sidst besøgt d. 20/05/2014)
- Til at sortere i dataen fra OpenStreetMap er java command-line applikationen Osmosis, samt OpenStreetMap wiki'en, hvorfra Osmosis kan hentes (Sidst besøgt d. 20/05/2014).
- Til at projektere koordinaterne i OpenStreetMap fra wts84 til UTM32 har vi brugt java klassen GeoConvert.java fra https://github.com/Jotschi/geoconvert/ (sidst besøgt d. 20/05/2014).
- Til kystlinjen i Krak har vi brugt dataen fra gruppe K.
- Javas API: http://docs.oracle.com/javase/7/docs/api/ (sidst besøgt d. 20/05/2014).
- Algorithms 4th Edition af Robert Sedgewick og Kevin Wayne (Afsnit 4.3 og 4.4)

Bilag

Arbejdsfordeling

	Kode	Rapport
Adam	**Route Package**, Hele pakke **Klasserne**: Sidebar, DrawInterface, Graphics2DDraw Road Tidlig udgave af ML. SearchLabel (Focus listener med tekst der forsvinder når man tryk **Testklassen** RouteJUnit	**Teknisk beskrivelse af** Route Package SideBar. **Analyse af** Route Package Krav nummer 3. **Test af** Route Package. **Introduktion**, **Gruppeanalyse**, **Konklusion**, **System test,** **Diskussion**
Kristian	**ctrl:** ML (inde over meget af den), StartMap. **krakloader:** Alt andet end LoadCoast. **Model:** CurrentData, Road. **OSMParser:** Hele pakken. **View:** Canvas **Route:** Hjulpet med ændring til at bruge lists	**Forord og indledning** **Problemanalyse** Krav: 1 (struktur delen), 4, 8 (alt andet end om quad/kd-træ), 9. **Teknisk afsnit:** Klassediagram, struktur, Canvas, ML, osmparser pakken, KrakParser, Model pakken. **Gruppeanalyse.** **Diskussion.**
Johan	QuadTreePack Package (alt),	**Teknisk beskrivelse:**

	krakloader Package (alt der ikke er kraks egen kode). **Klasserne:** Road, CurrentData, ObserverC, ObervableC, Canvas(Hjulpet med dele, men kun skrevet en smule i paintComponent), SideBar (ændret i initializationen af keylisteners, men ellers ikke arbejdet med), CanvasML, KL, ML, StartMap.	QuadTreePack, ObserverC, ObservableC, CanvasML. **Problamanalyse:** krav 5, krav 8, Eget krav nummer 1. **Afprøvning:** Whitebox test af Quadtree. System test. **Samarbejdsværktøjer** : Trello. **Brugervejledning og eksempler:** Brugervejledning og eksempler.
Peter	**AddressSearch package:** AddressParser, AutoCompleter, SearchField, ZipToCityNameParser, RoadNameSearcher. **ctrl package:** KL. ML(scrollZoom). StartMap. **View:** Canvas(tidligere versioner) Switchcase for vejtypefarvning. **QuadTreePack package** (hjulpet med design, debugging og en smule implementation under parkodning med Johan). **Test package:** AddressParserTest. **Andet:** AlphabetParser, SearchTrie	**Baggrund og problemstilling** **Problemanalyse:** Første del af krav nummer 1. Krav nummer 2. Krav nummer 7. Krav nummer 10. **Teknisk beskrivelse:** KL, AddressParser, AutoCompleter, RoadNameSearcher, SearchField, ZipToCityNameParser. AddressSearch.UnusedCode package **Test:** Whiteboxtest af AddressParser. **Samarbejdsværktøjer**: Kommunikation: Facebook, Skype, telefon.

		Web baserede samarbejdsværktøjer: Github, Google Docs. Planlægning og arbejdslog: Logbog, Worksheets og Trello. **Gruppeanalyse.** **Store dele af Logbogen.**

Bilag – krav til programmet

Til programmet blev vi stillet følgende krav:

1. Danmarkskortet skal tegnes visuelt ud fra datasættene Krak og OSM i et programvindue.
2. Forskellige slags vejsegmenter skal tegnes med forskellige farver, fx motorveje røde, hovedveje blå, stier grønne og resten sorte.
3. Danmarkskortet skal skalerer efter størrelse når der trækkes i vinduets ramme, således at hvis vinduet bliver større eller mindre, skal vinduet stadig vise samme vejsegmenter, men skaleret op i en henholdsvis større eller mindre størrelse.
4. Det skal være muligt at zoome ved at trække en firkant med musen, således at firkanten bliver det nye billede, og alle vejsegmenter inden for firkanten skaleres op til det nye billede.
5. Vis navnet på vejen der er tættest på musen.
6. Det skal være muligt at finde korteste vej mellem to punkter på kortet (enten ved at indtaste adressen eller ved at klikke to steder på kortet).
7. Programmet skal have en sammenhængende brugergrænseflade for kortet og ekstra funktionalitet. Brugere skal kunne interagerer brugervenligt med programmet ved hjælp af mus og keyboard.
8. Programmet skal køre hurtigt nok til at være bekvemt at bruge, også for datasæt af hele Danmark. Start-up tiden kan være svær at optimerer da datafiler er gemt i tekstfiler, men efter startup skal brugergrænsefladen reagere i bekvem hastighed.
9. Brugeren skal kunne vælge mellem KRAK eller OpenStreetMaps.
10. Programmet skal indeholde mindst én udvidelse fra listen 'Extensions' angivet i projektbeskrivelsen.
11. For en gruppe af 4 personer, skal programmet kunne udskrive rutevejledning i tekstform.

Vores egne krav til programmet lød som følgende:

1. Koden til programmet skal være logisk struktureret og vedligeholdelsesvenligt. Det vil sige have lav kobling, og et højt abstraktions- og kohæsions niveau. På denne måde vil programmet også være videre udviklingsvenligt.

Bilag Quadtreetest 1

```
private void getRoads(double x1, double x2, double y1, double y2, ArrayList<Road> rl) {

    if (northeast != null) {                                    //1
        if (northeast.boundary.containsBox(x1, y1, x2, y2)) {   //2
            northeast.getRoads(x1, x2, y1, y2, rl);
        }
        if (northwest.boundary.containsBox(x1, y1, x2, y2)) {   //3
            northwest.getRoads(x1, x2, y1, y2, rl);
        }
        if (southeast.boundary.containsBox(x1, y1, x2, y2)) {   //4
            southeast.getRoads(x1, x2, y1, y2, rl);
        }
        if (southwest.boundary.containsBox(x1, y1, x2, y2)) {   //5
            southwest.getRoads(x1, x2, y1, y2, rl);
        }
    } else {
        if (roadList[0] != null) {                              //6
            for (Road road : roadList) {                        //7
                if (road == null) {                             //8
                    break;
                }
                rl.add(road);
            }
        }
    }
}
```

Coverage table:

Choice	Input property	Input data set
1 true	Road inside bounds	A
1 false	Road outside bounds	B
2 true	More than 999 roads inside bounds	C
2 false	Less than 1000 roads inside bounds	A

3 true	Road inside northeast	C
3 false	Road outside northeast	C
4 true	Road inside northwest	C
4 false	Road outside northwest	C
5 true	Road inside southeast	C
5 false	Road inside southwest	C
6 true	1000 roads inside bounds	C
6 false	Less than 1000 inside bounds	A

Bilag Quadtreetest 2

```
private void getRoads(double x1, double x2, double y1, double y2, ArrayList<Road> rl) {

    if (northeast != null) {                                    //1
        if (northeast.boundary.containsBox(x1, y1, x2, y2)) {   //2
            northeast.getRoads(x1, x2, y1, y2, rl);
        }
        if (northwest.boundary.containsBox(x1, y1, x2, y2)) {   //3
            northwest.getRoads(x1, x2, y1, y2, rl);
        }
        if (southeast.boundary.containsBox(x1, y1, x2, y2)) {   //4
            southeast.getRoads(x1, x2, y1, y2, rl);
        }
        if (southwest.boundary.containsBox(x1, y1, x2, y2)) {   //5
            southwest.getRoads(x1, x2, y1, y2, rl);
        }
    } else {
        if (roadList[0] != null) {                              //6
            for (Road road : roadList) {                        //7
                if (road == null) {                             //8
                    break;
                }
                rl.add(road);
            }
        }
    }
}
```

Coverage table:

Choice	Input property	Input data set
1 true	More than 1000 roads in quadtree	A
1 false	1000 or less roads in quadtree	B
2 true	Search area inside or overlaps bounds	a.C
2 false	Searcharea does not touch bounds	a.E
3 true	Search area inside or overlaps bounds	a.D
3 false	Searcharea does not touch bounds	a.E

4 true	Search area inside or overlaps bounds	a.C
4 false	Searcharea does not touch bounds	a.E
5 true	Search area inside or overlaps bounds	a.F
5 false	Search area does not touch bounds	a.E
6 true	1 - 1000 roads in quadtree	B
6 false	No roads in quadtree	H
7 zero	No roads in quadtree	H
7 once	1 road in quadtree	I
7 more than once	1000 or less roads in quadtree	B
8 true	Less than 1000 roads in quadtree	B
8 false	1 or more roads in quadtree	I

Bilag AddressParserTest 1

Code references:

```
private static String parseStreetName(String adr)
{
    String streetName = null;
    p = Pattern.compile(".\\d+.");
    m = p.matcher(adr);

    if(m.find())                                                    //1
    {
        String[] temp = adr.split("\\d+");
        String address = temp[0];

        if(address.charAt(address.length()-1) == ' ')               //2
            address = address.substring(0, address.length()-1);
        if(address.charAt(address.length()-1) == ',')
            address = address.substring(0, address.length()-1);     //12

        streetName = address;
    }
    else
    {
        String[] temp = adr.split("\\d+");
        String address = temp[0];

        if(address.charAt(address.length()-1) == ' ')               //3
        {
            address = address.substring(0, address.length()-1);
        }
        if(address.matches(".*[,].*"))                              //13
        {
            String[] temp2 = adr.split(",");
            address = temp2[0];
        }
        streetName = address;
    }
    return streetName;
}
```

```
private static String parseBuildingNumber(String adr)
{
    String number = null;
    p = Pattern.compile("\\b\\d{1,3}[,]\\b|\\b\\d{1,3}"
            + "[a-zA-Z]\\b|\\b\\d{1,3}\\b");
    m = p.matcher(adr);
    if(m.find())                                                    //4
    {
        number = m.group().replaceAll("[a-zA-ZæøåÆØÅéÉäÄöÖ]", "");
    }
    return number;
}
```

```
private static String parseBuildingLetter(String adr)
{
    String buildingLetter = null;
    p = Pattern.compile("\\d+\\w");
    m = p.matcher(adr);
    if(m.find())                                                    //5
    {
        buildingLetter = m.group().replaceAll("\\d+", "");
    }
    return buildingLetter;
}
```

```
public static String parseFloor(String adr)
{
    String floor = null;
    p = Pattern.compile("\\d+[.]");
    m = p.matcher(adr);
    if(m.find())                                                    //6
    {
        floor = m.group().replaceAll("\\.", "");
    }
    return floor;
}
```

```
public static String parseZip(String adr)
{
    String zip = null;
    p = Pattern.compile("\\d{4}");
    m = p.matcher(adr);
    if(m.find())                                                    //7
    {
        zip = m.group();
    }
    return zip;
}
```

```
public static String parseCity(String adr)
{
    String city = null;
    p = Pattern.compile("\\s[A-Za-zæøåÆØÅéÉäÄöÖ]+");
    m = p.matcher(adr);

    if(m.find())                                                    //8
    {
        p = Pattern.compile("\\d{4}\\s[A-Za-zæøåÆØÅéÉäÄöÖ]+");
        m = p.matcher(adr);
        if (m.find())                                               //9
        {
            String [] temp = adr.split("\\d{4}\\s");
            city = temp[temp.length-1];
        }
        else if(adr.contains(", "))                                 //10
        {
            String[] temp = adr.split("[,]\\s");
            if(!temp[temp.length-1].matches(".*\\d.*"))             //11
            {
                city = temp[temp.length-1];
            }
        }
    }
    return city;
}
```

Coverage table:

Choice	*Input property:*	*Input example*	*Input data set*
1 true	Streetname followed by streetnumber and zip	”Rued Langgaards Vej 7 2300”	A
1 false	Address with no streetnumber or zip	“Rued Langgaards Vej”	B
2 true	Same as A, but anddress ends with whitespace (after 1st if-statement)	”Rued Langgaards Vej 7 2300”	A
2 false	Same as A, but string does not end with whitespace (after 1st if-statement)	“Rued Langgaards Vej7 2300”	C
3 true	Streetname ends with whitespace	“Rued Langgaards Vej ”	D

3 false	Streetname does not end with whitespace	“Rued Langgaards Vej”	B
12 true	Address contains “,” with no house number	“Rued Langgaards Vej, 2300 København S”	L
12 false	Address does not contain “,” with no house number	“Rued Langgaards Vej 2300 København S”	M
13 true	If address contains comma “,”	“Rued Langgaards Vej,”	N
13 false	If address does not contain “,”	“Rued Langgaards Vej”	B
4 true	With building number	“Rued Langgaards Vej 7”	E
4 false	Without building number	“Rued Langgaards Vej”	B
5 true	With building letter	“Rued Langgaards Vej 7A”	F
5 false	Without building letter	“Rued Langgaards Vej 7”	E
6 true	With floor number	“Rued Langgaards Vej 7 5. Sal”	G
6 false	Without floor number	“Rued Langgaards Vej 7”	E
7 true	With 4-digit zip number	“Rued Langgaards Vej 7 2300”	A
7 false	Without 4-digit zip number	“Rued Langgaards Vej 7”	E
8 true	With city name	“Rued Langgaards Vej 7 København S”	H
8 false	Without city name	“Rued Langgaards Vej 7”	D

9 true	With city name and 4-digit zip	"Rued Langgaards Vej 7 2300 København S"	I
9 false	City without 4-digit zip	"Rued Langgaards Vej 7 København S"	H
10 true	Address contains ", "	"Rued Langgaards Vej 7, København S"	J
10 false	Address does not contain ", "	"Rued Langgaards Vej 7 København S"	H
11 true	If contains ", " is followed by zip	"Rued Langgaards Vej 7, 2300 København S"	K
11 false	If contains ", " is not followed by zip	"Rued Langgaards Vej 7, København S"	J

Special case coverage table:

Explanation	*Input property*	*Expected results*	*Input data set*
Without "Sal"	"Rued Laanggaards Vej 7 5."	Expected to pass	Special1
Without "."	"Rued Laanggaards Vej 7 5"	Expected to fail**	Special2
With "," zip	"Rued Laanggaards Vej, 2300"	Expected to pass	Special3
With "," city	"Rued Laanggaards Vej, København S"	Expected to pass	Special4 (13 sequel)

With "," Everything	"Rued Laanggards Vej 7A, 5. sal 2300 København S"	Expected to pass	Special5
Without "," Everything	"Rued Laanggards Vej 7A 5. sal 2300 København S"	Expected to pass	Special6

**NB: Our AddressParser cannot find floor without ".".

Worksheets

Worksheet 1 Group A

Work subset	Drawing the map - Requirements 1-3
Step	Basic features
Date	03/03/2014
Assigned to	Whole group
Description of work	• Drawing the map • Scaling map to frame • Road types assigned to a color
Implementation	Canvas

Next step	• Zoom
	  ○ Quad tree
	• Class structure
	• Detect nearest road
	  ○ Priority queue

Issues	Current implementation of zooming does not work properly. Consider hardcoded aspects of current implementation.

Work subset	Implement QuadTree - Expansion
Step	Create QuadTree
Date	17/03/2014

Assigned to	Johan, Peter helped by Kristian
Description of work	• Implemented a quadtree
Implementation	QuadTree Package
Next step	• Integrate quadtree into program
Issues	Can't query for nodes colliding with selected area..

Work subset	Implement zoom - Requirement 4
Step	Detect area of zoom - Draw and zoom
Date	17/03/2014
Assigned to	Adam & Peter
Description of work	• Draw zoom area • Zoom, so that box = new window • Zoom accurate • Zoom using QuadTree
Implementation	Canvas, QuadTree Package
Next step	• Make zoom more accurate • Zoom using Quad tree
Issues	Current implementation of zooming does not work properly. Zoom area and zoom is not accurate.

	We need the implementation of QuadTree to be done before starting implementing QuadTree to zoom method.

Worksheet 2 Group A

Work subset	Class structure, zoom
Step	Rework class structure after MVC, implement better zoom
Date	26/03/2014
Assigned to	Kristian
Description of work	• Restructure classes in package after the Model View Controller pattern. • Improve the zoom function to zoom correctly
Implementation	All packages/classes, CurrentData
Next step	• Implement optional features
Issues	None known

Work subset	Zoom, draw zoom area, graphics interface, report writing
Step	Finishing some stuff regarding the graphics
Date	26/03/2014
Assigned to	Adam

Description of work	• Drawing the map • Scaling map to frame • Road types assigned to a color
Implementation	Canvas
Next step	• Implement optional features • Fixing small bugs
Issues	The drawed square disapears when the mouse is not moving

Work subset	Integrate QuadTree into program, Implement closest road feature
Step	Finish QuadTree and make it work with our map functions. Make a feature that shows the road closest to the cursor when browsing the map.
Date	26/03/2014
Assigned to	Johan and Peter
Description of work	• Finish implementation of quadtree • Fixed final bugs in quadtree and implemented it into the project • Implemented feature to show name of road closest to the cursor, using the quadtree.
Implementation	QuadTree Package, ctrl Package, model Package.
Next step	• Implement optional features • Write report
Issues	None known

Worksheet 3 Group A

Work subset	Write first part of report and make adjustments to code
Step	Write, edit, format and send group project
Date	01/04/2014
Assigned to	Everyone
Description of work	• Write the first part of the report • Edit and correct until satisfied • Format it so it looks nice • Add it to the project and send group project • Make small adjustments to code if necessary
Implementation	Førsteårsprojekt del 1.docx
Next step	• Implement optional features • Plan next part of project
Issues	None known

Worksheet 4 Group A

Work subset	AddressSearchEngine
Step	Make first implementation of road Search

Date	09/04/2014
Assigned to	Peter
Description of work	• Make JTextField and keyListener • Search RoadList and report if road is there • Make Road object comparable by VEJNAVN for later sorting and searching
Implementation	SearchEngine package Classes: KL, SearchLabel, Road - CompareTo method
Next step	• Consider if sort and binary search pays off for just few searches on roads? • Sort RoadList • Search for a Road object using a string given in JTextField • Implement feature to project
Issues	Road lists is in collection List, which makes it hard to sort and search.

Worksheet 5 Group A

Work subset	Road route
Step	
Date	20/04/2014
Assigned to	Adam
Description of work	• Find the fastest route from point a to point b
Implementation	Fastest Route package

Next step	• Make it faster.. Not fast right now •
Issues	Nullpoiner exception og langsom

Work subset	Implement Coastline on krakdataset. Make strange characters show correctly.
Step	Find coastline data and integrate it with krak data Make strange characters show correctly.
Date	20/04/2014
Assigned to	Johan
Description of work	• Find and coastline data and make it compatible with krakdata • Add it into quadtree loading and drawing in program. • Set the correct encoding to show characters.
Implementation	Krakloader Package, ctrl Package
Next step	• Implement optional features • Write report
Issues	None known

Work subset	Key Panning and Google maps style Scroll Zoom
Step	Implement Key Panning in KeyListener, make first implementation of Scroll Zoom

Date	20/04/2014
Assigned to	Peter
Description of work	• Implement Key panning in KL • Make methods for scrollZoom in ML.
Implementation	ctrl package Classes: KL, ML.
Next step	• Make ScrollZoom work proberly
Issues	ScrollZoom wants to zoom to top left corner, but I can see a “Google maps” effect - Zooming more when i lower right corner than upper left.

Worksheet 6 Group A

Work subset	fastest road
Step	Draw fastest road Make shortest road to A*
Date	28/04/2014
Assigned to	Adam
Description of work	• same as step
Implementation	Canvas, shortestRoad
Next step	

Issues	

Work subset	Implement zoom animation and correct zoom rectangle
Step	Make the zoom rectangle display correctly. Implement an animation that is showed when a zoom action is performed.
Date	28/04/2014
Assigned to	Johan
Description of work	• same as step
Implementation	Canvas, ML.
Next step	• finish zoom animation • Set up quadtree's for different zoom levels with different road types.
Issues	The observer pattern in java has an eventque which interferes with repaint calls, which make painting the animation an issue.

Work subset	Make adjustments to AddressParser
Step	Make it functional to this project
Date	21/04/2014
Assigned to	Peter

Description of work	• Split addressparsing into methods • Remake methods for parsing
Implementation	AddressSearchEngine: AddressParser, SeachField,
Next step	• Make autocomplete functionality for SearchField, new class might be created for this purpose
Issues	Minor adjustments to Regex in AddressParser to make it work just right

Work subset	Ændring af OSM data
Step	ændre dataen med osmosis
Date	21/04/2014
Assigned to	Kristian
Description of work	• Lær at bruge osmosis • find brugbar information
Implementation	osmparser
Next step	• find evt. overset information
Issues	None known

Work subset	Parser til OSM data
Step	lav en parser der kan hente OSM dataen ind.

Date	21/04/2014
Assigned to	Kristian
Description of work	• Lav en SAX parser • Find relevante tags etc.
Implementation	osmparser
Next step	• Parse i et brugbart format
Issues	Ikke nogen som ikke forventes.

Work subset	Make scroll zoom work proberly
Step	Use pairprogramming technique with Peter and Johan to make it work properly
Date	28/04/2014
Assigned to	Peter
Description of work	• Implement Key panning in KL • Make methods for scrollZoom in ML.
Implementation	ctrl package ML.
Next step	• Make ScrollZoom work proberly
Issues	None

Worksheet 7 Group A

Work subset	Road route
Step	
Date	5/05/2014
Assigned to	Adam
Description of work	• Finishing Route package • Making SideBar • Text based route
Implementation	Fastest Route package SideBar class
Next step	
Issues	

Work subset	Set up quadtrees for different zoom levels
Step	create multiple quadtrees and sort roads between them
Date	5/05/2014
Assigned to	Johan
Description of work	• same as step
Implementation	Canvas, ML, CurrentData
Next step	• Document project

Issues	

Work subset	Ændring af OSM data
Step	Finpudse det med java applikation
Date	12/05/2014
Assigned to	Kristian
Description of work	• Lav java klasse til at gennemlæse osm fil og oprette ny osm fil med rigitg information og ekstra information fjernet.
Implementation	osmchanger
Next step	• find manglende information
Issues	Har ikke fundet og inkluderet noget information, så som post nr

Work subset	Parser til OSM data
Step	Ret til at få alt information med og skip en gennemgang af data.
Date	12/04/2014
Assigned to	Kristian
Description of work	• Rette switch i SaxHandler • slå to for-løkker sammen i OSMParser
Implementation	SaxHandler, OSMParser

Next step	• Ændre parser til ikke at lave ekstra gennemløb af data.
Issues	Ikke nogen kendte

Work subset	Skifte Canvas til at bruge tiles
Step	Impementer tiles.
Date	12/04/2014
Assigned to	Kristian
Description of work	• Finde måde at lave et gitter på • Lav metode til at tegne tiles • Lav metode til at udregne og gennemløbe tiles
Implementation	Canvas
Next step	• Ændre generelt i systemet så lange edges bliver tegnet.
Issues	Meget lange edges bliver ikke tegnet ordenligt.

Work subset	Make adjustments to search-as-you-type-functionality
Step	Integrate it into project, and make it work right
Date	30/04/2014
Assigned to	Peter

Description of work	• Make small adjustments to regex to make it parse just right. • Make adjustments to AutoCompleter • Consider ways to Autocomplete city name
Implementation	AddressSearchEngine: AddressParser, SearchField, Autocompleter, RoadNameSearcher
Next step	• Integrate cityautocompletion
Issues	

Work subset	Zip to city translation
Step	Make ZipToCityParser and input data
Date	02/05/2014
Assigned to	Peter
Description of work	• Find input dataset online for zip to city (Denmark only) • Make class that can parse • Implement into search-as-you-type-functionality
Implementation	AddressSearchEngine: AddressParser. bynavne.txt
Next step	• Make autocomplete functionality for SearchField, new class might be created for this purpose
Issues	Sweden wants us to pay 3600 swedish kroner for a textfile with zip and citynames… http://www.postnummerservice.se/filer/postnummerfiler/postnummerfilen rediculus….

Work subset	Start report writing
Step	Plan reportwriting, set up paragraphs with headlines on Google Docs so that groupmembers can start writing
Date	07/05/2014
Assigned to	Peter
Description of work	• Find input dataset online for zip to city (Denmark only) • Make class that can parse • Implement into search-as-you-type-functionality
Implementation	AddressSearchEngine: AddressParser. bynavne.txt
Next step	• Make autocomplete functionality for SearchField, new class might be created for this purpose
Issues	Sweden wants us to pay 3600 swedish kroner for a textfile with zip and citynames… http://www.postnummerservice.se/filer/postnummerfiler/postnummerfilen rediculus….

Worksheet 8 Group A

Work subset	Write report
Step	Write whitebox test for quadtree
Date	12/05/2014
Assigned to	Johan

Description of work	Wrote whitebox test of quadtree and fixed found issues
Implementation	
Next step	Write report.
Issues	

Work subset	Write report and test
Step	Test for Route package
Date	12/05/2014
Assigned to	Adam
Description of work	Wrote Junit test and analisys of Route package Fixed turn calculation
Implementation	Linked
Next step	Write report.
Issues	

Work subset	Implement Trie for Road name searching
Step	Start implementing Trie, make AlphabetParser
Date	14/05/2014

Assigned to	Peter
Description of work	Implement Trie and AlphabetParser and make dokument with alphabet
Implementation	AddressSearchEngine.UnusedCode: AlphabetParser, SearchTrie
Next step	See if I can get done… Whiteboxtest
Issues	Dont know if I can get it finished this implementation. Debugging show errors. Exceptions are thrown with dataset

Work subset	Whiteboxtest AddressParser and keep on reportwriting
Step	Test for Route package
Date	16/05/2014
Assigned to	Peter
Description of work	Write whiteboxtest with Junit for AddressParser
Implementation	TestPackage - AddressParser
Next step	Write report.
Issues	Takes a whole day to whiteboxtest AddressParser, does not make sense only to test smaller parts. It all has to be testet.

Work subset	Write report
Step	Write report about areas in which i have had high participation
Date	19/05/2014
Assigned to	Johan
Description of work	Wrote many different parts of report, but especially those concerning quadtree and user manual.
Implementation	
Next step	Finish report together with group
Issues	

Work subset	Write report, and make minor adjustments to code and javadoc
Step	Write and edit report
Date	19/05/2014
Assigned to	Peter
Description of work	Make report and make finishing touches to code and report
Implementation	AddressSearchEngine
Next step	Finish report together with group
Issues	

Logbog

Logbog d. 24/02-2014 - Forfatter: Peter.

24. februar 2014
15:04

Mangler:
Samarbejdsaftale.
Ét worksheet pr. person.
Fordeling af opgaver.
Overblik over arbejdsopgaver.
Overblik over tidsfrist og tidsplan

Diskussion:
Krak vs. Open
Kristian og Adam: Helst krak
Peter: Vi kigger på de forskellige datatyper, det er for tidligt at beslutte noget endnu. Open giver måske nye muligheder for funktionalitet end krak, men krak er nok nemmere at behandle.

Plan:
Kig på data og vælg datagruppe. - **(FDS: Dette diskuteres og undersøges næste gang)**
Vi aftaler en dag i denne uge på facebook (nok i aften) - Tirsdag (i morgen) er nu forslået.
Undersøge: Hvordan man tegner kortet i det hele taget. - **(FDS: Dette diskuteres og undersøges næste gang)**
Interesseret i github, men dropbox kunne også bruges. - Oplæg om Github af TA's TBA
KD-tree eller Quad-tree. **(FDS: Dette diskuteres og undersøges næste gang) - dog skal fokus være på at tegne kortet.**

Information:
d. 7-11 Adam Engsig er på skiferie.
Johan er syg i dag, også vil vi gerne læse lidt op på opgaven, før vi laver samarbejdsaftale og worksheets.

Mangler til næste gang:
Vi aftaler i aften hvornår vi mødes næste gang.

Næste gang:
Analyser opgaven. Lav evt. interfaces. Samarbejdsaftale. Diskussion og undersøgelse af data, tegn af kort, KD-tree og Quad-tree.
Se links til eksempler på Worksheet og Group constitution learnit d. 16-22 februar på learnit.

Ekstra noter fra Isabella:
Gruppelog punkter:

Møder og gruppesamarbejder,
Gruppelogbog,
Mødetider,
Kontakt,
Konfliktløsning,
Ambitionsniveau.

Logbog d.25-02-2014 - Forfatter: Adam

Samarbejdsaftale er blevet lavet.
Vi kigger på kraks data, dokumenter givet hertil og deres loader til på torsdag hvor vi holder skype møde kl 17.

- Evt krak vs openstreetmaps. Hvad skal vi bruge?
- Basic graph library? Edge og nodes.
- Læs dokument og kode.

Vi vil bestræbe os på at få tegnet kortet til på mandag, men den endelig dato for at tegne kortet bliver bestemt under skype mødet

Logbog d.27-02-2014 - Forfatter: Peter

45 minutters møde på Skype.
Data er bestemt: Krak.
Der blev besluttet at se på det over weekenden, vi mødes mandag d. 3/3 kl. 11 og koder kortet færdig.

Logbog d.03-03-2014 - Forfatter: Kristian og Peter

Mødtes kl 11:00 og arbejdede til 15:40 hvor der var møde med Søren. Herefter lavede vi logbog/worksheet og aftalte næste møde / hjemmearbejde.
Da vi mødtes begyndte vi med at se på hvad folk havde fundet frem til og valgte at arbejde videre med Johans kode, da han havde fået den til at tegne Danmark. Derefter blev der arbejdet med at få kortet til at passe i frame og skalere i forhold til det, når det ændres. Til sidst begyndte vi at kigge lidt på at zoome.
Til næste gang vil Johan og Adam kigge på zoom/quadtree, Peter vil kigge på nærmeste veje og Kristian vil kigge på klasse struktur, da koden pt. er ret rodet.
Vi har aftalt at mødes næste mandag kl. 11.
Note: Adam er på skiferie.

Indtressante begreber - af Peter:
Open street maps - data. - Større datasæt, mere detaljeret, hver hustand har en knude.
Quadtræ.
OpenGL - hold os til swing siger Søren, indtil det virker begrænsende.
AffineTransform: http://docs.oracle.com/javase/7/docs/api/java/awt/geom/AffineTransform.html
Model view controller pattern.

Fra mødet med Søren - af Peter:
Dårlige valg skrives i rapporten, der kan spørges om ekstra planer.
Ingen rapport til første del, men flere requirements og rapportskrivning venter.

Vi kan gå amok med ekstra implementationer - udnyt dataen vi har fået tildelt, overvej navigation og multithreading.
Eksamen: Andel del af projektet som fokus. Giv præsentation af hvad vi har lavet, salgsagtigt/forsvar, hvor vi står, gode valg/dårlige valg. Forstå og refflektér.

Logbog d.10-03-2014 - Forfatter: Kristian/Peter

Dagen startede med undervisning af TA's hvor vi lærte om Git og GitHub. Vi tænkte vi af nysgerrighed ville prøve dette.
Pga. det kun var Peter og Kristian der kunne møde, valgte vi at udskyde videre arbejde med vores projekt. I stedet har vi sat os ind i Git og GitHub.
Vi fik opsat et repository som vi alle har tænkt os at arbejde fra. Vi prøvede push og pull funktioner.
Vi mødes i morgen tirsdag efter undervisning i Diskret Matematik, da mandagsmødet ikke blev til noget. Vi sætter alle op på GitHub når vi mødes tirsdag.

Logbog d.11-03-2014 - Forfatter: Peter

Johan, Kristian og Peter mødtes efter Diskret matematik. Vi snakkede om Github og fik Johan connected til vores repository også. Også snakkede vi om alternative implementationer til tegnemetoden for ændring af vinduesstørrelse.
Den nye implementation er pænere kode, og muligvis også mere effektiv, men det kunne godt se ud som om den kommer i komplikationer med vores Zoom metode. Dette har vi dog ikke fået bekræftet endnu da koden opførte sig meget underligt på den computer vi brugte. Til næste gang tjekker vi op på om vores zoommetoden er forandret efter den nye implementation. Også ville Kristian lægge projektet op på Github således at vi kunne "pushe" og "pulle" fra denne i fremtiden.
Næste gang vi mødes bliver formentlig på mandag d. 17 kl. 11 som sædvanligt. Vi mangler at få sat alle op på Github, også mangler vi at løse det sidste krav: Requirement 5 - Display closest toad name. Der er frit slag på at arbejde med hvad man vil.

Logbog d.17-03-2014 - Forfatter: Kristian / Peter

Vi startede med at forbinde alle, samt lægge projektet på github.itu.dk. Efter lidt tid flyttede vi det dog over på github.com, da nogle havde oplevet problemer med at forbinde til det hjemmefra. Bagefter begyndte vi at arbejde på projektet: Adam og Peter arbejdede videre med zoom og ryddede lidt op i koden, Johan og Kristian arbejdede på at integrere Johans implementation af et quadtree i programmet. Peter hjalp Johan til sidst, men quadtree er stadig ikke færdig. Den bliver loaded i starten af programmet, men ikke brugt endnu. Vi skal nok lave vores implentation om så vores requirements understøtter quadtree.

Logbog d.24-03-2014 - Forfatter: Peter / Johan / Kristian / Adam

Vi mødtes kl. 11, planlagde hvad vi ville lave for dagen. Kristian havde lavet strukturen i projektet om, således at den understøttede Model View Controller pattern. Dette synes gruppen var godt, og vi skiftede så til denne version på GitHub.

Johan og Peter brugte dagen på at debugge quadtree for at få det til at virke, efter mange timers arbejde lykkedes det, dog blev kortet stadig ikke tegnet, sandsynligvis på grund af paint metoden ikke tegner koordinaterne korrekt. Denne skal ændres, så den ikke bare tegner kortet, men gør det "smart", således at vi kan få musekoordinater præcist oversat til kort koordinater. Også skal andre requirements metoder ændres, således at de også udnytter QuadTree.
Kristian har arbejdet på at scale det område man zoomer på rigtigt: Dette fungerer pt. i et test projekt, men er endnu ikke blevet sat ind i det rigtige projekt, da quadtree ikke virkede mens Kristian arbejdede med det.
Adam har påbegyndt rapporten, samt arbejdet på den grafiske del. Der er blandt andet blevet lavet et graphicsInterface for mere løs kobling
På onsdag snakker vi om rapport, dertil skal dokumentet Peter Sestoft - "Udformning af rapporter" læses.
Næste mødedag: Onsdag d. 26/3-2014: Peter møder ca. kl 12:30. Adam ville møde lidt før, Kristian skulle på arbejde men ville møde senere, og Johan møder også omkring 12-12:30.

Logbog d.26-03-2014 - Forfatter: Peter / Johan

Vi mødtes fra kl. 12, og hvornår folk kunne mødes efter. Peter og Adam startede med at skrive videre på rapporten. Da Kristian og Johan ankom, og gruppen samledes, snakkede vi om hvordan vi kunne samle projektet. Vi støtte så på fejl med blandt andet QuadTree'et som vi hurtigt fik rettet. Efter at QuadTree'et kom op at køre, arbejdede Johan og Peter videre med at implementere nærmeste vej funktionen. Adam og Kristian fortsatte med at få zooming til at virke korrekt, samt at implementere panning og filtrering af kortets vejsegmenter.
Vi har nu fået programmet til at virke efter omstruktureringen af klasserne samt QuadTree implementationen.
Til sidst aftalte vi, at alle skriver rapporten på Google Docs med deres farver som vi så fletter sammen på mandag.
Peter vil også gerne kigge på zoom ud, og i det hele taget zoom ind og ud via. scroll.
Næste mødedag: Mandag d. 31.03.2014 kl. 11.

Logbog d.31-03-2014 - Forfatter: Adam

Vi mødtes Adam, Peter og Kristian. Johan var syg. Vi gik i fuld gang med rapporten, samt bug fixes. Et par bugs dukkede op efter vi fik ord på vores kode. De blev udrettet løbende. Alle arbejde på rapporten. Der er stadig nogle få bugs, men overordnet går det godt frem. Panning er blevet implementeret og fungerer. Vi færdiggøre rapporten i morgen.

Logbog d.01-04-2014 - Forfatter: Peter

Peter, Kristian og Adam mødtes efter Diskret Matematik kl. 14 og skrev første del af Rapporten færdig. Johan var syg, men havde skrevet lidt til rapporten samt lidt javadoc. Små justeringer på koden blev lavet før aflevering. Rapporten blev rettet, formateret og færdiggjort. Planen er nu, at Adam afleverer i morgen tidligt.

Logbog d.09-04-2014 - Forfatter: Peter

Dagen startede med introduktion til XML filer og SAX parsere. Aftale om at mødes mandag d. 14.04.2014 blev aftalt på Facebook. Opslag af opgaver og opgavefordeling skete efter undervisningen på Facebook. Gruppemedlemmer kan nu arbejde så meget de vil på deres opgave indtil vi mødes mandag.

Logbog d.22-04-2014 - Forfatter: Fælles

I starten påsken havde vi et møde hvor vi aftalte hvad der skulle foregå i påsken, opgaver blev uddelegeret og trello blev introduceret til gruppen. Mandag havde vi et kort Skypemøde og snakkede om hvad der skulle gennemgås idag. I dag har vi haft en opsummering på status af uddelegerede opgaver samt sammensat de forskellige dele af koden. Rettelse af små fejl, diskussion af problemer, plan er lagt for hvornår kodeskriving skal være slut og rapportskrivning skal startes for alvor (5 maj).
Vi mødes i morgen d. 22/4-2014

Logbog d.28-04-2014 - Forfatter: Peter

I dag mødtes vi kl 11, og snakkede om de implementationsidéer vi var igang med. Kl. 12 fik vi feedback af Magnus, som Peter tog noter til, som deles her:
“

Til eksamen bliver vi mest bedømt på Rapporten, derefter programmet, og sidst koden.
Magnus synes vores rapport overordnet virkede grundig og gennemført.
Dog bar der også steder hvor vi var for upræcis.

Problemanalysen skal være mere præcis, mere teknisk, men ikke ‘programmeringsteknisk’.
Her skal der tages alternative løsninger op, disse skal diskuteres, én løsning skal vælges og valget skal begrundes. (Sammenlign specifikationsmæssigt). Valget kan eksempelvis være: Fordi det var let, fordi det kørte bedre, fordi vi kender den.
Afsnittet skal skrives så læseren skal kunne se hvad man har gjort, og skal kunne genskabe det.
I det hele taget argumenter: Hvorfor har vi netop valgt denne løsning?
Krav -> problem -> implementation.
Problemanalysen er det vigtigste og længste del i en akademisk rapport siger Magnus.
Vigtigst for problemanalysen: Hvilken beslutning har vi taget og hvorfor?

Test - Demonstrer at vi kan, men test ikke hele programmet.

Reflektion: Mangler en liste over ting der kan forbedres i gruppearbejdet, også fejl og bugs i programmet.

Problemstilling: Krav skal kunne måles (eksempelvis ‘lav kobling’ - Der skal udskiftes så og så mange linjer kode i programmet for at det er muligt...’)
Undgå ‘flyvske’ specifikationer.
Skriv mere akademisk, mere præcist.

Teknisk beskrivelse - før afsnit om Quadtree mindre, da det er forklaret tidligere i problem analysen.
”

Efter holdte vi frokost og derefter endnu et møde, hvor der blev aftalt at mødes på Skype Onsdag eller Torsdag aften. Alle i gruppen synes aftaleplatformen ‘Trello’ fungerer godt til planlægning af møder, samt overblik over hvad gruppemedlemmer arbejder på.

Mandag d. 5. maj er slår vi de sidste ting sammen, og starter på rapportskrivning, således at vi har god tid til at finjusterer programmet samt skrive rapport før aflevering.

Logbog d.01-05-2014 - Forfatter: Adam

Møde på skype. Snak om implementation.

Logbog d.05-05-2014 - Forfatter: Adam

Vi mødtes i dag med tanken om at samle den kode vi hver især havde lavet, og færdiggøre vores kode. Altså var i dag planlagt som den sidste dag med kode. Der var nogle små ting som ikke fungerede, så helt færdige med kode er vi ikke, men der vil fra i dag af komme mere fokus på opstart af rapporten. Vi har snakket om hvor vidt vi skal bruge mere tid på at implementerer ekstra funktioner, eller om vi bare skal gå direkte til rapporten når de sidste små ting er fixet.
Vi har aftalt at mødes igen på Onsdag, hvor vi vil snakke mere om vores tilgang til rapporten.

Logbog d.12-05-2014 - Forfatter: Peter

Mødtes og snakkede om implementationer.
Skift til at bruge tiles af buffered image.
Hjælp fra Søren til vejfinding.
Aftale onsdag test.
Start på rapporten.

Logbog d.13-05-2014 - Forfatter: Peter

Folk har kodet hjemmefra.

Johan og Peter mødtes og rettede bugs i programmet:
Canvas focus listener - Så Canvas kan tage fokus fra searchLabels (til panning).
Rettelse af controls:
Mousepan højreklik -> venstreklik.
Dragzoom venstreklik -> højreklik.
Reset venstre klik -> højreklik.
(Dette var nødvendig for at undgå reset ved fokus på canvas)
Ret dragzoom firkant så den ikke bliver grøn når stier vises.

Vi har fundet andre bugs som skal rettes:

Gør searchRoad (orange linje) mere tydelig.
Vis orange linje når der søges, uden at skulle påvirke canvas først (kald repaint når der søges).
Ret buffered image/zoom så den zomer ind på buffered tiles i stedet for at loade nye tiles pr. zoomniveau.
Ret tiles så veje ikke forsvinder fra én tile til en anden.
Smalt zoom vertikalt eller horizontal resulterer i et forkert smalt billede (vertikale tiles loades ikke ved smalt horizontalt og omvendt).

Trello er brugt til styring af opgaver.

Onsdag mødes vi for test og rapportskrivning.

Logbog d. 14-05-2014

Programdele til test blev valgt og whitebox tests påbegyndt.
Systemtest blev udsat til et senere tidspunkt til efter whitebox test er færdige og de sidste rettelse i programmet var lavet.

Logbog d. 19-05-2014

Der blev arbejdet med rapport samt system test. Der manglede at blive skrevet om en del ting i rapporten og der skulle også rettes. Langt det meste blev lavet færdigt dog var der nogle få ting der stadig manglede.
Til d. 20 aftalte vi at folk så vidt muligt skulle blive færdige så vi kunne få skrevet rapporten færdigt og sat op.

Logbog d. 20-05-2014

Sidste dele af rapporten blev skrevet færdig, og sat sammen, samt det sidste javadoc blev skrevet. Peter læser rapporten igennem til i morgen før aflevering, så projektet er færdig i morgen inden aflevering. Til i morgen får folk også tid til at skrive evt. manglende javadoc så alt er klar til at blive afleveret.

Samarbejdsaftale:

Gruppe møder:

-Hvornår

Permanent dag: Mandag fra klokken 10:00

Mulig dag: Onsdag (Eftermiddag efter 12)

-Hvor

ITU

-Hvordan

Opsamling fra sidste møde - Worksheet færdiggøres fra sidste uge

Nuværende status

Planer for dagen

Planer til næste gang - Worksheet skrives

Gantt diagram vurderes i forhold til status - OBS: Glemte vi til første del, men har tænkt os at gå i krig med til anden del. (Erstattet af Trello)

Logbogsskrivning

-Andet

Kan man ikke komme informeres det via en eller flere kommunikationsmetoder så hurtigt så muligt

Hvis 1 ikke kan komme mødes vi alligevel

Hvis 2 ikke kan vurderes situationen, evt skype møde.

Logbogsskrivning går på tur

Pauser - 10 min per time.

Frokost pause

Lang dag - en til længere pause udover frokost pause

Ambitioner:

God karakter, men ikke på bekostning af andet og andre fag.

Tidsplanlægning:

Gantt diagram (Senere ændret til Trello)

Gruppe logbog:

Agenda

Færdiggjort arbejde

Store beslutninger

De uddeligeret opgaver

Andet

Kommunikations metoder:

Facebook

Skype

Google drive

Telefon

Flere bliver added.

Arbejdsformer

Der er mulighed også for at arbejde på egen hånd, hvorefter der deles med gruppen.

Vi skal fortælle hinanden hvis vi har fundet ud af noget nyt, som vi bruger til implementationen.