Udarbejdet i forbindelse med Førsteårsprojektet på bacheloruddannelsen   
Softwareudvikling på IT-Universitetet i København forår 2014.

Førsteårsprojekt del 1

Danmarkskort

Gruppe A: Peter Clausen, Kristian Nielsen, Adam Engsig og Johan Arleth

Email: [pvcl@itu.dk](mailto:pvcl@itu.dk), [kvin@itu.dk](mailto:kvin@itu.dk), [adae@itu.dk](mailto:adae@itu.dk), [johaa@itu.dk](mailto:johaa@itu.dk)

Indhold

[Forord 1](#_Toc384144619)

[Problemstilling og Baggrund 1](#_Toc384144620)

[Problemanalyse 2](#_Toc384144621)

[KRAV 1 & 2 2](#_Toc384144622)

[KRAV 3 2](#_Toc384144623)

[KRAV 4 3](#_Toc384144624)

[KRAV 5 3](#_Toc384144625)

[VORES KRAV 1&2 4](#_Toc384144626)

[VORES KRAV 3 4](#_Toc384144627)

[Brugervejledning og eksempler 5](#_Toc384144628)

[Teknisk beskrivelse af programmet 6](#_Toc384144629)

[Struktur: 6](#_Toc384144630)

[StartMap: 6](#_Toc384144631)

[Road: 7](#_Toc384144632)

[CurrentData: 7](#_Toc384144633)

[Canvas: 7](#_Toc384144634)

[ML: 7](#_Toc384144635)

[DrawInterface & Graphics2DDraw 7](#_Toc384144636)

[QuadTree 8](#_Toc384144637)

[GUI: 8](#_Toc384144638)

[Afprøvning / tests 9](#_Toc384144639)

[Manuel test: 9](#_Toc384144640)

[Test af zoom: 9](#_Toc384144641)

[Test af reset: 9](#_Toc384144642)

[Test af panning: 9](#_Toc384144643)

[Test af nærmeste vej: 10](#_Toc384144644)

[Test af flere input samtidig. 10](#_Toc384144645)

[Refleksion over gruppearbejdet 10](#_Toc384144646)

[Brugen af GitHub, Google Docs og Facebook. 10](#_Toc384144647)

[Samarbejdsaftalen, Gantt diagram, logbog og worksheets. 11](#_Toc384144648)

[Arbejdsfordeling og teamroller. 11](#_Toc384144649)

[Andre arbejdsværktøjer 12](#_Toc384144650)

[Møder og kommunikation 12](#_Toc384144651)

[Konklusion 12](#_Toc384144652)

[Diskussion og mangler 12](#_Toc384144653)

[Bilag 13](#_Toc384144654)

# Forord

Denne rapport omhandler java programmet TimTim, som vi i en gruppe af fire bachelorstuderende, over et fire uger langt forløb, har udviklet i forbindelse med første del af Førsteårs-projekt, forår 2014 på IT-Universitetet i København. 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.

Vi vil gerne takke Magnus og Sune vores teaching assistents, for deres introduktion til Git, som fik os overtalt til at gøre brug af det, og tak til Sebastian Ehlers som viste os ‘GitHub for Windows’ der simplificerer brugen af Git yderligere.

TimTim er et program der giver en grafisk præsentation af Danmarks vejnet, stier og søruter. Programmet stiller en række intuitive funktionaliteter til rådighed for at hjælpe brugeren rundt i programmet. Det minimalistiske design gør TimTim meget håndgribeligt at bruge, selv for brugere uden meget computer erfaring.

I rapporten vil der gås i dybden med hvilke tanker vi har gjort omkring vores valg af implementation, samt en teknisk beskrivelse af den faktiske implementation, med de vigtigste metoder og klasser nævnt. Til sidst vil der være en  diskussion af vores implementation samt mangler, og om man i bagklogskabens lys, kunne have valgt en bedre løsning på dele af problemet.

# Problemstilling og Baggrund

Problemstillingen er at lave et interaktivt Danmarkskort ud fra et udleveret datasæt lavet af Krak. Kraks datasæt er 2 tekstfiler bestående af al den information der er værd at vide om veje i Danmark. Programmet skal kunne bruges af en almindelig bruger, der ved hvordan man betjener en computermus. Kortet *skal* opfylde de 5 krav angivet herunder, men må gerne indeholde mere funktionalitet:

Til programmet blev vi stillet følgende krav:

1. Danmarkskortet skal tegnes visuelt ud fra datasættet 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 indenfor firkanten skaleres op til det nye billede.
5. Vis navnet på vejen der er tættest på musen.

Vores egne krav til programmet lød som følgende:

1. Programmet skal lave en inddeling af Kraks data, således at programmet ikke skal tegne hele kortet, når man har zoomet helt ud eller helt ind.
2. 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.
3. Mulighed for at bevæge kortet, med musen (Panning).

# Problemanalyse

Dette afsnit rummer en argumentation for måden vi har valgt at opbygge systemet og løse de krav vi har fået tildelt. Vi vil diskutere nogle af de centrale overvejelser, vi har gjort os før det endelige design blev fastlagt.

## KRAV 1 & 2

Det udleveret datasæt kommer med en parser som læser de 2 filer og opdeler dem i edges og notes. En vej består af en eller flere edges, og hver edge består af 2 nodes. Kraks datasæt er således lavet at det kun består at rette linjer (læs edges), så når en vej svinger har de opdelt vejen i mange små brudstykker som hver især er rette. Hver edge indeholder information omkring hvilken node den starter og slutter i. Disse nodes indeholder de præcise koordinatsæt til hvor vejen faktisk starter og slutter i Danmark. Al information vedrørende vejtype, vejlængde, postnummer m.m. findes i den pågældende edge. For at tegne Danmarks veje må altså for hver edge finde vej typen samt de 2 tilhørende nodes og tegne et linjestykke mellem disse i den korresponderende farve. Vi oplevede i vores forsøg på at tegne Danmarks veje at dataen havde værdier svarende til Danmarks rigtige størrelse, hvilket skulle skaleres ned. Til at starte med skalerede vi alt ned med en fastslået konstant. Dette viste sig dog at være upraktisk i forhold til nogle af de kommende krav, specielt at skulle have muligheden for at resize vinduet. Vi valgte at skifte til at skalere i forhold til størrelsen på vores JComponent.

De nedskaleret tal blev til nogle lange decimaltal. Dette blev til et overraskende problem da Graphics som var vores umiddelbare bud på et tegne library kun tager heltal. Det ville være spildt arbejde for mindre præcision at runde alle tal af til heltal. Heldigvis viste det sig at Graphics2D, en klasse som arver fra Graphics, kan tegne former med decimaltal som værdier. Dette er dog gemt væk bag et interface, så vi ikke er afhængige af at bruge Graphics2D, hvilket vil blive forklaret senere i denne rapport.

Det andet krav kunne ordnes med en switch-case med TYP feltet fra EdgeData som case, dette har vi holdt fast i og ikke overvejet nogle andre løsningsmodeller for, da denne løsning er kodemæssig overskuelig, kort og videre udviklingsvenlig.

## KRAV 3

Vores første løsning omkring skaleringen af vinduet var at bruge mouseListeners metoder, mouseExited og mouseEntered således at man tog størrelsen på vinduet ved mouseExited og igen ved mouseEntered og ud fra det, beregne hvor meget større eller mindre størrelsen på vinduet havde ændret sig. Denne løsning fungerede ikke videre godt, da metoderne betragtede musen stadig var indenfor Canvas når man ændrede vinduesstørrelsen, man var derfor nødt til at fører musen ind og ud af vinduesrammen for at få den til at opdatere efter ændring af vinduesstørrelse, hvilket vi så som en fejl i implementationen.

Vi overvejede så at bruge en componentListener. ComponentListeneren har metoden componentResized, som kaldes når componentet (her vinduesstørrelsen) ændres. Man kunne derfor kalde repaint fra denne metode så kortet blev tegnet på nyt, med den nye størrelse.

Til sidst fandt vi ud af, at metoden repaint, herunder metoden paintComponent, bliver kaldt hver gang størrelsen af vinduet ændres. Vi endte derfor med at beregne skaleringen ud fra components størrelse i metoden drawMap, som også tegner Danmarkskortet på ny, og som kaldes hver gang der kaldes paintComponent.

## KRAV 4

Vores første forsøg på at implementerer en zoom funktion, var at bruge AffineTansform biblioteket. Dette havde dog det problem, at alt blev forstørret, svarende til at holde en lup over det valgte område, som man ville gøre hvis man ville forstørre et billede. Det vil sige at vores vejes tykkelse også blev skaleret op.

I stedet brugte vi det rektangel, der bliver dannet når man trækker musen, til at hente veje i vores QuadTree. Bagefter kunne rektanglet bruges til at skalere længden på vejene ned, så de passer med den nye vinduesstørrelse.

Vi valgte også at tegne det rektangel man har markeret, mens man trækker musen for at vise området man ønsker at zoome på. Vi overvejede at bruge et glass-pane, som fungere som en glasplade over kortet, som man tegner på, i stedet for at tegne alle veje hver gang størrelsen på firkanten ændres. Pga. tidspress nøjes vi med at tegne rektanglet og kalde repaint hver gang størrelsen på rektanglet ændres, dvs. hele alle veje også tegnes på ny i billedet. Vi ved dette ikke er en optimal løsning, men den var nem og hurtig at implementere. Dette ændre vi nok i anden del af projektet.

## KRAV 5

For at finde den tætteste vej på musen, må vi først lave en antagelse omkring hvor på vejen vi laver sammenligning med musen. Er det vejens startpunkt, midtpunkt eller måske et tredje sted?. Vi synes den letteste måde at implementere dette, er at sammenligne med vejens midtpunkt, men dette kan stadig godt skabe upræcision på lange lige veje, da disse kun har ét midtpunkt, mens at buede vejsegmenter har flere. Vi har dog valgt at abstrahere en smule, og sammenligne musen med vejenes midtpunkt, selvom vi godt ved det ikke er helt præcist.

For at finde den tættest liggende vej, ville den umiddelbare løsning ville være at kigge alle veje igennem og sammenligne med musens koordinater for at finde den tætteste. Denne løsning vil dog tage for lang tid. Vi har derfor valgt at opdele alle veje i et QuadTree, som vi søger i. Implementationen af QuadTree vil vi uddybe længere nede i rapporten. Simplificeret har vi delt alle veje op i en masse små firkanter, hvilket gør det muligt for os kun at søge efter den tætteste vej i netop den firkant musen befinder sig i på et givent tidspunkt, hvilket gør søgningen af den tætteste vej på musen væsentligt hurtigere.

For at vise vejen har vi et JLabel, som vi har placeret i bunden af skærmen under kortet. Vi opdagede at kortet godt kunne “hoppe” på bestemte tidspunkter, når navnet på den tætteste vej ikke fandtes, hvilket resulterede i at JLabel’et forsvandt, og ændrede skaleringen på kortet, hvilket fik kortet til at “hoppe” på skærmen. For at løse dette problem ændrede vi søgningen, så vi finder den tætteste vej med et navn, i stedet for bare tætteste vej. Også indsatte vi teksten: “Vejen kunne ikke findes!” når der ikke var noget vejnavn at vise. Sidstnævnte sker dog kun når firkanten i QuadTræet ikke indeholder nogen veje, hvilket sjældent forekommer.

## VORES KRAV 1&2

Programmet er designet med udgangspunkt i MVC (Model - View - Controller) design pattern. Da man har Model til at opbevare dataen fra Krak, View til at vise dataen i form af et kort og Controller til at håndtere input fra brugeren. Derfor er vores program lavet med udgangspunkt i MVC, da det giver en god struktur, der er nem at arbejde med.

En af de store udfordringer ved projektet var at lave Model ordentligt, selvom vores kort ‘kun’ skal tegne Danmark, er der stadig nok linjer og punkter der skal bearbejdes til at programmet vil køre meget langsomt, hvis ikke det var optimeret. For at finde vejnavnet der er tættest på musen ville det være spild af tid at gå alle veje igennem og sammenligne med musen position, hver gang musen bare bevæges 1 pixel. Ligeledes vil det være spild af computerkraft, at tegne hele Danmark, hvis man er zoomet ind på amager.

Til at starte med havde vi to ArrayLists, med henholdsvis edges og nodes. Det var nemt at tegne ud fra, men det var svært at filtrerer unødvendig data fra, hvilket medførte programmet kørte meget langsomt.

De to andre muligheder for at sortere i dataen vi overvejede at benytte os af, var: KD-tree og Quadtree, der inddeler dataen i mindre rektangler, men på forskellige måder.

Vi valgte at Quadtree, da det opdeler dataen i fire lige store rektangler, hvilket virker mest relevant, i forhold til at det er et kort, i modsætning til KD-tree, der opdeler de i en masse forskellige rektangler alt efter punkternes fordeling. Dette giver mulighed for at sortere i dataen alt efter det rektangel, det vises.

For at gemme dataen i vore quadtree, brugte vi en klasse, Road, til at indeholde både dataen fra edge og nodes.For at gøre det nemt at lægge Road objekter i vores quadtree, valgte vi at lave et midtpunkt til vær Road (dvs. vær vejstykke), som man kunne opdele efter. Dette medførte dog det problem, at nogle steder ville man kunne se at der manglede veje rundt i kanterne af kortet, hvis man er zoomet ind. Dette kunne løses ved, meget simpelt, at forstørre det område, man faktisk tegnede, til 125% af det man ser.

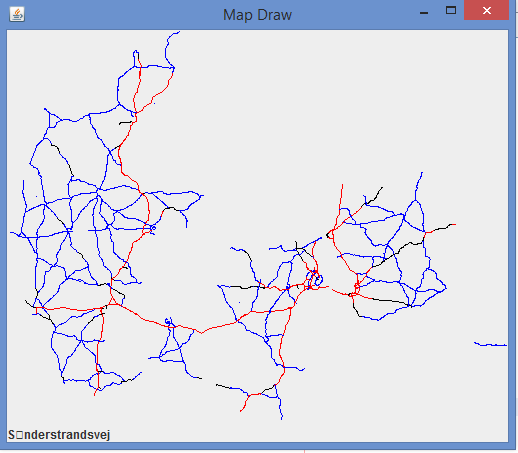
I forhold til de krav vi stillede os selv, løste vi det første via et quadtree. Det andet har vi, for at opnå et højt abraktionsnivuae og løs kobling, lavet to interfaces til de to klasser vi ville være afhængige af. Netop vores quadtree og dets data, samt vores tegne klasse. Det gør at vi til hver en tid meget nemt kan skifte til fx. KDTree og Graphics eller andet.

## VORES KRAV 3

I forhold til det tredje krav, med panning havde vi næsten en løsning, fra zoom, da programmet arbejder med rektangler, var det bare at flytte det rektangel, vi skalerede med, ved at lægge den afstand musen havde flyttet sig, til rektanglets øverste venstre hjørne.

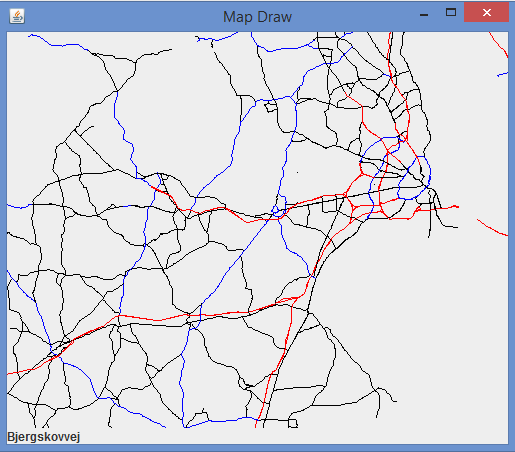
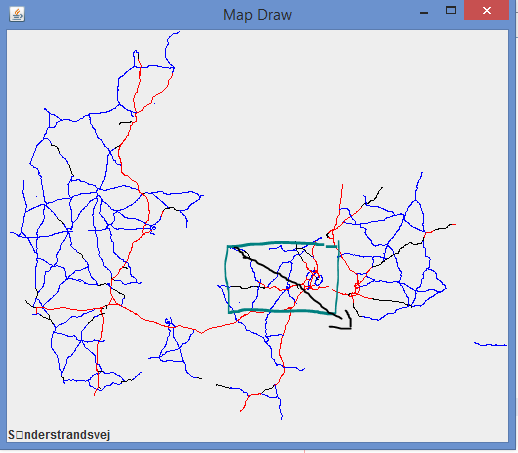
# Brugervejledning og eksempler

Når programmet startes vil man se kortet, zoomet helt ud (se figur 1), når der er zoomet helt ud, vil der kun blive vist motor- og hovedveje, efterhånden som man zoomer ind, vil flere typer veje blive vist.. I bunden af kortet vil navnet på vejen nærmest musen, blive vist.

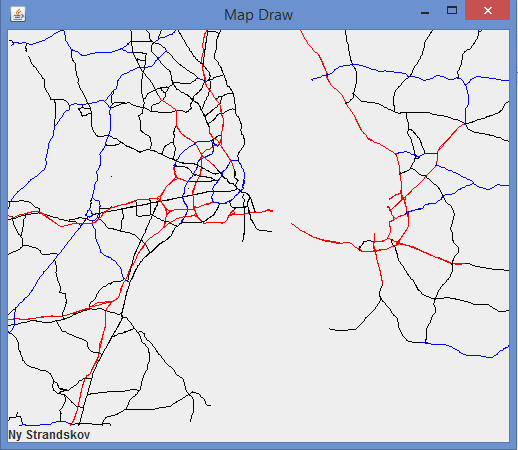
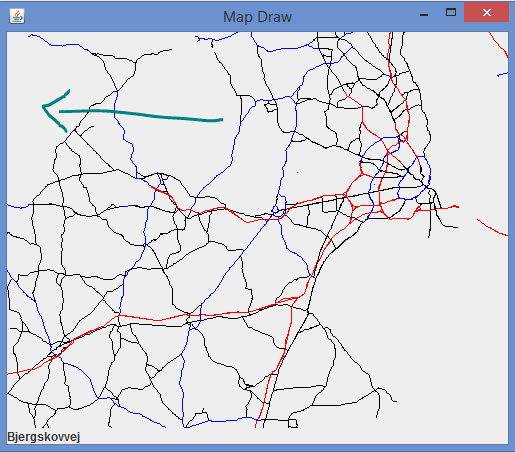


Figur

Ved at trække en firkant, med venstreklik, kan man zoome ind på det område rektanglet indkredser (se figur 2 og 3).



Figur Figur

Man kan også bevæge kortet med musen ved at trække med højre klik. 

Figur Figur

Ved et enkelt venstre klik på kortet, kan man genskabe det originale zoom og centrering. Når man ændre vinduets størrelse vil kortet skalere til at passe, i forhold til den mindste side.

# Teknisk beskrivelse af programmet

## Struktur:

Programmets struktur tager udgangspunkt i MVC design pattern. Helt grundlæggende svarer ML klasen til Control, Canvas klassen til View og CurrentData klassen til Model.

I vores ctrl-pakke har vi StartMap-klassen, der opsætter View, og loader-dataen ind i Model, samt ML der er vores mouseListener/mouseMotionListener, der ændrer model når brugeren gør noget.

I model pakken har vi CurrentData der indeholder den data, view skal vise. Road klassen svarer til et stykke vej bestående af en edge og to nodes, samt et midtpunkt.Udover disse klasser bruger model Quadtree pakken til at gemme data og hente det der skal bruges.

I view pakken har vi Canvas klassen, der tegner det data CurrentData indeholder. Canvas og CurrentData bruges i et observer pattern, hvor Canvas er observer og CurrentData er observable. View indeholder også et interface med metoder til at tegne streger og ændrer farve, samt interfacets implementation.

Tilsidst har vi krakloader pakken, der indeholder det udleverede kode fra krak, med få modificeringer, der kan hente data fra filerne.

Klassediagrammet kan ses i figur 6 i bilaget.

## StartMap:

Ligger i ctrl-pakken og indeholder main-metoden. StartMap opsætter vinduet, samt læser og indsætter dataen, via den KrakLoader, der fulgte med dataen.

## Road:

Et Road-objekt indeholder en EdgeData og de to, tilhørende, NodeData, fra tildelte KrakLoader.Ved instantieringen af Road-objektet, udregnes også et midtpunkt, for den graf EdgeData og de to NodeData udgør, som bruges til at indsætte i quadtree og søge efter nærmeste vej.

## CurrentData:

CurrentData bruger singleton design pattern da der kun skal bruges et. Den indeholder den relevante information som Canvas skal tegne.

CurrentData extender også Oberservable, som er Javas implementation af den ene type i observer design pattern. Som Observer har den Canvas, hvilket betyder at Canvas vil blive gjort opmærksom på ændringer i CurrentData.

CurrentData indeholder: en ArrayList med alle de relevante Roads at tegne, et Rectangle2D der svarer til den områder af kortet man ser, hvilke koordinater det øverste venstre hjørne af kortet svarer til og størrelsen på området der tegnes, når man zoomer helt ud.

## Canvas:

Klassen Canvas, der udgør hoveddelen af View, kan skalere og tegne et ArrayList af Road objekter. Canvas tegner ved hjælp af DrawInterface, der har metoder til at tegner en linje og et rektangel ud fra decimaltal, samt metoder til at skifte farve på det der tegnes. DrawInterface implementeres af Java2DDraw, der bruger Javas Graphics2D, til at tegne og skifte farve.

Canvas tegner den data der ligger i CurrentData og implementere derfor interfacet observer, der bruges sammen med CurrentData, der er observable.

Canvas extender også JComponent, hvilket gør det muligt at tildele det, til en frame. Siden det er JComponent har det også en højde og bredde, hvilket bruges til at beregne skaleringen. Det område man se og som skal skaleres ned gemmes som et Rectangle2D, i skaleringen 1, så den svarer til de værdier punkterne normalt har og de værdier quadtree’et bruger.

## ML:

ML implementere mouseListener og mouseMotionListener, og er den klasse der håndterer input fra brugeren. Den implementere begge, da de forskellige funktioner skal bruge måde klik, træk og bevægelse med musen.

Mange af de beregninger der laves når man zoomer og trækker kortet, ligger i ML, da værdierne skal skaleres op og rykkes, for at passe med punkternes koordinater. ML bruger information fra både CurrentData og Canvas til at beregne det nye område der skal tegnes og/eller beregnes hvor musen er i forhold til punkterne.

## DrawInterface & Graphics2DDraw

DrawInterface er et af vores mange skridt i at opnå løs kobling, samt et højere abstraktionsniveau. Interfacet indeholder kun lige de få metoder vi har brug for, for at kunne tegne og farve vores grafiske præsentation af Danmarks veje. Det er dog yderst nemt at tilføje nye metoder. Da vi i vores program har brug for at tegne linjer og firkanter af double værdier, har vi valgt at implementere vores DrawInterface på en klasse vi har kaldt Graphics2DDraw. Denne klasse gør brug af Graphics2D til at tegne streger og firkanter.

## QuadTree

Pakken QuadTree i programmet indeholder klasser der tilsammen danner et quadtree som kan sortere objekter af typen road der er lavet i en af de andre pakker.

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 quadtræet, her bliver holdt styr på mængden af indsatte objekter i hver quad og der bliver delt op i mindre quads når grænsen bliver nået.

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 quadtræet 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 også 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 laver i henholdsvis nordøst, nordvest, sydøst og sydvest retningerne. Klassen boundary bruger denne information til at udregne de korrekte grænser og center punktet.

QuadTreeInterface er lavet for at gøre det nemt at se hvad man skal give og hvad man får tilbage når man kalder quadtræet, klassen indeholder abstrakte versioner af insert og search som implementers i QuadTree.

## GUI:

Vores GUI er opbygget af en frame med layouttypen: BorderLayout, som opdeler frame i fem dele: center, nord, syd, øst og vest. I center har vi placeret instansen af Canvas og i syd har vi placeret det label hvor navnet på den nærmeste by står.

# Afprøvning / tests

## Manuel test:

Da programmet afhænger meget af bruger input via musen har vi valgt at hold os til nogle manuelle test.

## Test af zoom:

|  |  |
| --- | --- |
| Zoom med rektangel, der passer med vinduet | Fungerer helt perfekt |
| Zoom med rektangel, det er smalt og langt | Tegner kun kort, på et smalt område af vinduet. |
| Test for grænsen af zoom | Hvis man zoomer langt nok ind vil den stopper med at tegne nogle streger (sandsynligvis pga. anti-aliasing), men det er ikke noget der skaber problemer ved normalt brug. |
| Test af zoom, hvor musen slutter uden for vindue | Den zoomer, men det områder der tegnes, bliver ikke skåret af ved kanten af vinduet, hvilket resulterer i at den egentlig zoomer ud. Vi ser det som et problem, da den helst ikke bør zoome ud, på den måde |

\*Rektanglet der tegnes når man trækker musen, forsvinder mens musen står stille.

## Test af reset:

Dette fungerer fint, da den ikke har nogle variabler og reagerer på et klik.

## Test af panning:

|  |  |
| --- | --- |
| Normal panning, inden for vinduet | Fungerer som det skal. |
| Panning, med mus uden for vinduet | Panningen fortsætter når musen kører ud over vinduet, hvilket vi ikke ser som et problem, men egentlig noget der giver god mening. |
| Test af grænsen for panning | Umiddelbart, kan man trække så langt til siden man vil, men der tegnes så heller ikke noget. Muligvis vil den loopetil sidst pga. doubles max størrelse. |

## Test af nærmeste vej:

|  |  |
| --- | --- |
| Nærmeste vej, ved normal bevægelse af musen | Fungerer som det skal |
| Nærmeste vej, mens man zoomer/panner | stopper med at opdatere indtil man stopper med at trække. |
| Nærmeste vej, mus ude for vinduet | Opdatere ikke. |
| Nærmeste vej, mus ude for det område der tegnes, men inden for vinduet. | Opdatere ikke. |

## Test af flere input samtidig.

|  |  |
| --- | --- |
| Zoom og panning samtidig, zoom først | Den panner normalt og afbryder zoom. |
| Zoom og panning samtidig, zoom sidst | Den zoomer normalt, når man slipper første museknap, men zoomer igen når man slipper den anden. |

Programmet understøtter de funktioner det skal, men der er nogle ekstreme situationer, samt nogle situationer, der ikke normalt sker, der ikke fungerer helt efter hensigten.

# Refleksion over gruppearbejdet

## Brugen af GitHub, Google Docs og Facebook.

Til versionsstyring af koden, har vi gjort brug af Git. Vi havde alle hørt om Git og GitHub, men brugte før en gruppe på Facebook til at dele den nyeste implementation. Brugen af Git har gjort det nemmere for os at arbejde flere personer samtidigt og mere effektivt på samme kode, uden manuelt at skulle lave rettelser. Dog har konverteringen til brugen af Git, herunder slåskampe med git-kommandoer og løsning af konflikter, også taget en del af vores tid i starten af læringsforløbet. Vi er dog alle blevet gode til brugen af Git nu, og regner derfor med at få tiden tilbagebetalt gennem øget effektivitet. Summa summarum er vi glade for at have blevet introduceret til Git.

Ved brug af Google Docs har vi kunnet dele dokumenter med hinanden meget nemt over internettet. På vores Google Docs ligger både vores samarbejdsaftale, logbog og rapport samt andre dokumenter vi ønsker at dele og redigere i. På den måde har vi alle haft adgang til de nyeste delte dokumenter, aftaler og logs, som vi også kunne skrive på i fællesskab samtidigt.

Som primær kommunikationsplatform har vi gjort brug af Facebook, hvor vi har oprettet en gruppe hvor alle beskeder angående projektet bliver delt. Alle har mulighed for at tjekke op på de nyeste beskeder, og da vi alle er brugere af Facebook i dagligdagen, burde disse også blive læst hver dag, men som der vil blive forklaret under “Møder og kommunikation”, kunne gruppen godt have været bedre til at kommunikere. I starten brugte vi også Facebook-gruppen til deling af nyeste version af koden, men denne er som sagt blevet erstattet med versionsstyring programmet GitHub, da vi med sidstnævnte har opnået større effektivitet.

## Samarbejdsaftalen, Gantt diagram, logbog og worksheets.

Samarbejdsaftalen blev udviklet i starten af projektet som en slags “kontrakt” vi havde imellem os. Vi var alle rimelig enige om hvordan arbejdsforholdene skulle være i gruppen. Samarbejdskontrakten er blevet overholdt som den skulle angående mødetider, kommunikationsform og arbejdsmorale, dog er der ikke rigtigt nogen i gruppen der har kigget på den, siden den blev skrevet i starten af projektet. Til gengæld fik den os til at diskutere hvilke normer og arbejdsforhold vi ønskede under projektet, så der var enighed fra starten af projektet.

Gantt diagram var noget vi alle gerne ville have med i planlægning af projektet, dette følte vi dog ikke vi havde tid og overskud til i første del af projektet. Vi regner dog med at gøre brug af dette værktøj til anden del af projektet, således at der ikke sker noget tidsspilde, tidspres og så alle ved hvad de skal lave hvornår. Efter erfaring med aftalen skal den nok revideres i forhold til møder, hvilket vi vil blive beskrevet senere, under “Møder og kommunikation”.

Vi har været gode til at skrive logbog til hver arbejdsgang, således at alle gruppens medlemmer kunne følge med i projektets kronologiske udvikling. Ligeledes kan vi nu også læse projektudviklingen bagudrettet. Indtil videre har vi ikke haft særlig brug for logbogen rent produktivt, men dette ændres nok når vi skal skrive anden del af projektet.

Worksheets valgte vi at skrive bagudrettet, så vi kunne se hvem der havde lavet hvad og hvornår. Vi har forsøgt både at skrive worksheets gruppevis og individuelt. Worksheets er desværre ikke noget vi har brugt så meget endnu, nok fordi det er et nyt arbejdsredskab, som vi endnu ikke helt har lært at arbejde med endnu. Vi har dog tænkt os at undersøge mulighederne med dette værktøj nærmere i anden del af projektet.

## Arbejdsfordeling og teamroller.

Arbejdsfordelingen har været ligeligt fordelt. Vi har hver vores nøglekompetencer, som vi har forsøgt at udnytte. Alle har dog både været med til at kode, skrive rapport og deltage i arbejdsplanlægningen. Personerne i gruppen der har været bedre til at kode fra start, har stået for en større del af koden, men undervejs er der blevet udvekslet idéer, valg og forklaringer af implementationer gennem hele forløbet, således at alle i gruppen har forstået den, lært fra den, og har haft mulighed for at arbejde videre på den. Ligeledes har andre været mere med til planlægning af gruppearbejdet således at gruppen var på rette vej under hele forløbet, at deadlines blev overholdt og arbejdet diskuteret og dokumenteret. Alle i gruppen har været med til hele forløbet, men samtidig har vi også forsøgt at udnytte vores nøglekompetencer så godt vi kunne, hvilket har resulteret i et stærkt team, med tilfredse gruppemedlemmer. Vi har ikke diskuteret teamroller gennem projektet, da vi ikke har lyst til “at sætte folk i bås”, i stedet har vi ladet folk selv vælge hvilken rolle de vil spille i gruppearbejdet, hvilket også har fungeret meget godt.

## Andre arbejdsværktøjer

Gruppen har også gjort brug af “Parprogrammering” til mere indviklede dele af programmet. Dette har været en effektiv arbejdsmetode for nogle af os, da koden tænkes dybere igennem og fremadrettet mens den skrives. Vi har opnået færre fejl, og har brugt kortere tid på at udvikle særligt komplicerede implementationer til vores program.

## Møder og kommunikation

Vi aftalte et fast møde om ugen, hver mandag. Her mødtes vi samlede op på, hvad folk havde lavet og aftalte hvad der videre skulle ske. Dette fungerede godt, mens der ikke var nogle problemer, da folk fik frihed til arbejde som de ville.

Med facebook og Skype havde vi gode muligheder for at kommunikere, men vi har ikke været så gode til at udnytte det og kombineret med de få møder gjorde det, det svært at løse problemer hurtigt.

Det problem løb vi ind, da nogle problemer med strukturen, gjorde det svært at arbejde videre med koden. Dette gjorde at vi i to uger næsten ikke gjorde nogen fremskridt, da vi måtte droppe et møde og først næste møde begyndte at kigge på problemet, samtidig kæmpede vi også med Git, som også havde delt skyld i tidsspildet.

Til næste del af projektet, ville et ekstra, fast møde om ugen, nok være en god idé, samt at folk prøver at blive bedre til at kommunikere.

# Konklusion

Vi har lavet et program der grafisk præsentere Danmarks veje med forskellige farver efter vejtypen. Vi har løst de opstillede krav, om reskalering af vinduet, zooming og tætteste vej på musen samt tilføjet ekstra funktionalitet. Vi har så vidt muligt bestræbt os på at skrive en kode der har en logisk struktur og er vedligeholdelsesvenlig. Vores program har ingen store mangler, og de mindre mangler vil blive diskuteret i diskussionsafsnittet.

I forhold til gruppearbejdet, har vi fundet ud af hvad der fungerer og ikke fungerer, så vi kan gøre det mere effektivt, i næste halvdel af forløbet.

# Diskussion og mangler

Angående at vise den tætteste vej på musen, så er der en nuværende fejl ved visning af bogstaverne “Ææ”, “Øø” og “Åå”, som i stedet vises som “□”. Dette skyldes noktekst formatett på det JLabel der viser vejens navn. Formatet skal nok ændres, således at Æ, Ø og Å også vises (UTF-8), dette har der dog ikke været helt tid til endnu.

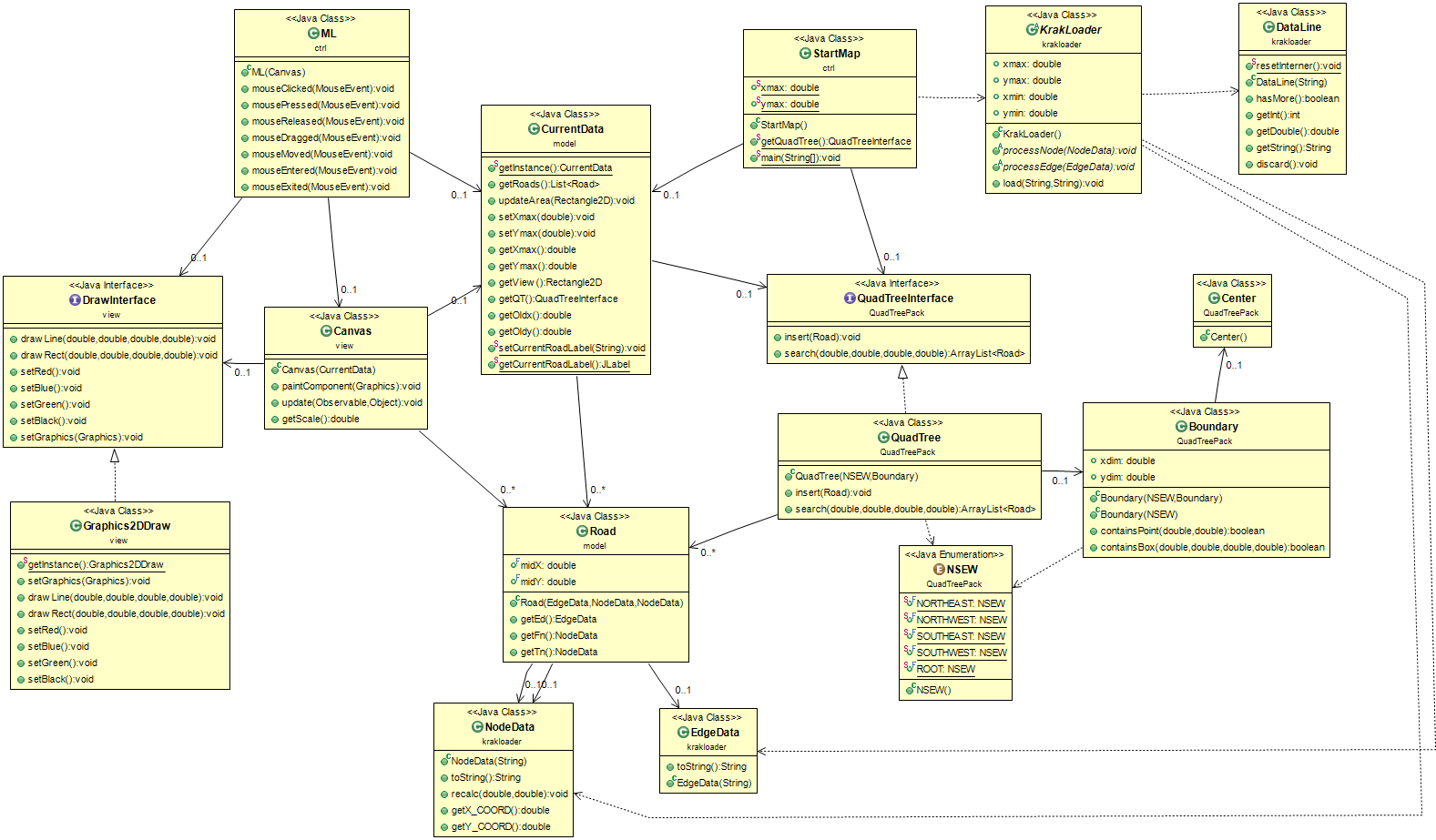
Vi har optimeret vejvisningen idet at vi kun vise de største veje når man er zoomet helt ud. Det har dog den konsekvens at mindre og nogle større øer forsvinder helt fra kortet, indtil man zoomer ind. Dette kan afhjælpes ved at tegne kystlinjen for alle øer i Danmark.

Den firkant man hiver som markerer hvor man zoomer ind, forsvinder når man holder musen stille. Det sker da metoden til at tegne firkanten kun bliver kaldt når musen er i bevægelse, hvilket er et mindre problem, som forhåbentlig kan løses ved nærmere undersøgelse af events, herunder consume() metoden, og repaint.

Derudover opfører zoom og pan sig lidt forkert, ved specielle input, der ikke sker så ofte, hvilket burde kunne løses, ved nogle flere check af inputtet.

# Bilag

FDS: Indsæt samarbejdsaftale, worksheets og logbog

****

Figur Klasse diagram over vores program.