

Indholdsfortegnelse

1	Indledning	2
1.1	Introduktion	2
1.2	Omkring Bilag	2
2	Introduktion til Problem	3
3	Problemanalyse	4
3.1	Datastruktur	4
4	Implementationsbeskrivelse	5
5	Afprøvning	6
5.1	Afprøvning	6
	Metode	6
	Ækvivalensklasser	6
	Dokumentation	7
6	Evaluerings af programmet	8
6.1	Hastighed	8
6.2	Fejl og mangler	8
6.3	Brugervenlighed	8
7	Evaluerings og refleksion over proces	9
7.1	Evaluerings og refleksion over proces	9
8	Konklusion	11
Bilag		12
8.1	Forventningstabeller	12
	Vector	12
	Box	12

Indledning

1.1 Introduktion

Denne rapport indeholder gruppe J's analyse og beskrivelse af den implementerede løsning af "Project: part 1: Visualisation". implementationen og rapporten blev udfærdiget i løbet af 4 uger.

1.2 Omkring Bilag

I rapporten har vi vedlagt følgende bilag:

- Kravsspecifikation: Den af kursusansvarlige udleverede PDF
- Brugsvejledning: En simpel vejledning til brug af vores implementering
- Implementeringsdokumentation: Vores Javadoc-genereret dokumentation af vores implementation
- Krakdokumentation: Den medfølgende dokumentation til vores datapakke
- Arbejdslog: Vores log over den ugentlige proces
- Arbejdsblade: Vores log over det udførte arbejde
- Samarbejdsaftale: Vores aftale omkring hvordan vi ville håndtere vores arbejdsforløb
- Forventningstabeller: Oversigt over resultater fra programafprøvnin-gen

Introduktion til Problem

Vores opgave går ud på at lave et interaktivt Danmarkskort, med indbyggede zoom og scroll-funktioner. Kortet skal have alle veje i det udleverede datasæt og stier markeret og være i stand til at vise den nærmeste vej, på der hvor ens cursor er. For den præcise kravsspecifikation se "LAV REFERENCE TIL BILAGET MED KRAVSSPECIFIKATIONEN"

Ud over de obligatoriske krav, har vi implementeret følgende features:

- Zoom out: Man kan zoome ud på keyboardet v.h.a. o,+ eller num(+), eller ved at rulle sit mussehjul baglæns
- Zoom in: Man kan zoome ind på keyboardet v.h.a. i,- eller num(-), eller ved at rulle sit mussehjul fremad
- Scrolling: Man kan scrolle kortet på keyboardet via piletasterne eller med musen ved at venstreklikke og hive kortet rundt
- Map reset: Man kan vende tilbage til startposition og zoomgrad på kortet v.h.a. r, backspace eller space, eller ved at trykke på midterste musseknop
- Vi har tegnet Danmarks kystlinje

Problemanalyse

3.1 Datastruktur

I implementeringen af ethvert kort må den bagvedliggende datastruktur nøje overvejes, da denne i høj grad afgør applikationens ressourcekrav. Overvejelserne omkring datastrukturen involverede blandt andet muligheden for kun at indlæse specifikke dele af dataet samt søgeegenskaber. Det givne datasæt består af punkter og tilhørende linjesegmenter, som forbinder to punkter. Punkterne indeholder information om deres individuelle placering, mens linjesegmenterne er forbundet med en række forskellige oplysninger såsom vejnavn og -type. Da kortdataet består af linjesegmenter, er det også af betydning, at disse kan repræsenteres gennemførligt. Værd at overveje var også datastørrelsen - datakilden fra Krak indeholder over en million punkter, og disse skal på ethvert tidspunkt under programkørslen kunne tilgås hurtigt.

To forskellige datastrukturer syntes besidde de nævnte egenskaber: en sorteret tabel og et quadtree. Den sorterede tabel brillere med en ukompliceret implementering. En quadtree-struktur, mens kompliceret at implementere, udmærker sig derimod ved dens simple datasegmentering samt hastige indlæsning af specifikke områder. Derudover er det muligt at justere centrale dele af en quadtree-struktur, så denne indpasses datakilden. Det vurderedes at quadtree-strukturen ville udmunde i en både hurtigere og mere dynamisk datastruktur, og således sås som den mest passende løsning.

Implementationsbeskrivelse

Afprøvning

5.1 Afprøvning

I bestræbelse efter sikring af applikationens kvalitet og pålidelighed er en række afprøvninger af dennes bestanddele udført løbende med udviklingen. Afprøvningen er ikke foretaget med en målsætning om at udtømme samtlige muligheder for systemfejl, men derimod blot for fyldestgørende at forhindre invaliderende defekter samt bidrage til udviklingsprocessen.

Metode

Da afprøvningsmetoden, som anvendes under udarbejdelsen af programprøver, er afgørende for afprøvningens dækningsgrad, fastsattes den indledningsvist som *black-box testing*. Det er således påkrævet at samtlige kode-segmenter i en afprøvet kodeblok køres én gang som minimum.

Eftersom udviklingen er foretaget i Java, er afprøvning udført i samspil med prøvemiljøet JUnit - et populært Java værktøj til netop afprøvning. JUnit er således anvendt under kørsel af prøverne, hvorfor disse også er skrevet op imod JUnits API.

Som nævnt er formålet med afprøvningen ikke at gennemteste samtlige kodefragmenter i applikationen, hvorfor kun en række konkrete klasser og deres metoder er afprøvet. Især *Vector* og *Box* klasserne ses brugt på tværs af applikationen, og derfor er netop disse udvalgt.

Ækvivalensklasser

Vector og *Box* klasserne er dataklasser, som indeholder værktøjsmetoder til hyppige operationer. Derfor håndterer samtlige metoder i bund og grund kommatal. Ækvivalensklasserne for afprøvning af klassernes metoder er således negative input, positive input samt nul-input.

Dokumentation

Dokumentation for de udførte prøver findes i bilag 8.1. Her ses de forventede og faktiske resultater af afprøvningsmetoderne opstillet i forventningstabeller.

Evaluering af programmet

Programmet understøtter alle de påkrævede features, såvel som flere af de valgfrie. Vi er derfor grundlæggende tilfredse med programmets funktionalitet. Vi kunne godt have tænkt os at have været mere kreative, og fundet på nogle features selv, men vi havde ikke det fornødne overskud.

6.1 Hastighed

Programmet er 5 eller færre sekunder om at indlæse data ved opstart, hvilket vi synes er acceptabelt taget i betragtning af at data er lagret i rå tekstfiler.

Vi har arbejdet en hel del med at få applikationen til at have en hurtig opdateringsfrekvens uanset udsnit og type af brugerinteraktion, og det er vores indtryk at det langt hen ad vejen er lykkedes. Dog kan der opstå relativt lave opdateringsfrekvenser når der navigeres i særligt krævende udsnit af kortet (f.eks. helt zoomet ud, eller overblik over København).

6.2 Fejl og mangler

Vi har ikke umiddelbart noget kendskab til fejl og mangler.

6.3 Brugervenlighed

Vores program giver ikke brugeren nogen informationer om hvad mulighederne for interaktion er, og det er derfor op til brugeren selv at udforske programmets features. Særligt galt kunne det gå hvis brugeren ikke er bekendt med digitale kort generelt, og derfor ikke har en forhåndsviden om typiske muligheder for interaktion i denne type program.

Evaluering og refleksion over proces

7.1 Evaluering og refleksion over proces

I dette afsnit reflekteres og evalueres der over hele processen fra problemidentifikation til det midlertidige produkt.

I den indledende fase udarbejdede gruppen i fællesskab en samarbejdsaftale, hvor der blev sat prioriteringer, et ambitionsniveau, præferencer m.m. Efter hvert møde skulle der føres log og udfylde arbejdsblade, samt planlægge arbejdsopgaverne for næste gang. Samarbejdsaftalen var med til at afklare hvilke forventninger man har til samarbejdet, og var med til at sætte et fælles mål for projektet.

Der kunne nu tages hul på den næste fase, hvor problemet identificeres og defineres. Her foretog gruppen en grundig gennemgang af de obligatoriske krav, samt hvilke valgfrie krav der kunne tilføjes. Dernæst afgrænsede gruppen sig i første omgang til at tilføje en zoom-ud funktion udover de obligatoriske krav. Det var på daværende tidspunkt svært at tage stilling til, hvilke valgfrie funktioner vi ville tilføje. Det vigtigste for os var at kunne få lavet et første udkast af visualisering af kortet, der får fremvist vejene. På baggrund af dette var gruppen afklaret med at benytte KrakData som datakilde, da den var nemmere at benytte og håndtere.

Det næste trin i forløbet var at få opstillet en løsningsmodel, hvor gruppen tog udgangspunkt i at strukturere koden i et Model View Controller framework. Vi foretog en tavlediskussionen der omhandlede hvilke klasser der skulle være i hver task. Dette var med til at give gruppen et godt overblik over hvad programmet skulle indeholde, hvilke arbejdsopgaver der skulle udføres og hvilke centrale klasser der skulle laves.

Efter at have opstillet en løsningsmodel, kunne vi arbejde os hen imod at få lavet et første udkast af programmet. Det var ikke alle der fik skrevet kode i første omgang, det var primært dem, som havde styr på at kode, der tog styringen. Dette kunne ikke undgås, da der i starten var få og store

arbejdsopgaver, og det var vigtigt for gruppen at opnå resultater på kortest tid. Grundet de få arbejdsopgaver opstod der tilfælde, hvor kun én person fik skrevet kode, mens de andre ville følge med og komme med input. Dette medførte at produktiviteten var lav, og når der opstod fejl i koden gik hele udviklingsprocessen i stå. Det var også denne del der var tungest og tog længst tid. Mere strategi, mere overblik. Nedbrudt ting til flere klasser. Ingen erfaring med at lave sådan nogle programmer.

Produktiviteten begyndte først at blive relativt høj efter at vi fik tegnet kortet for første gang. Der blev arbejdet på forskellige moduler på en gang og Model View Controller frameworket gav stor glæde og viste sig her at være særlig nyttigt. Der blev i gruppen arbejdet mere effektivt, og der blev tilføjet flere valgfrie funktioner. Vi fik i gruppen mere overskud og større overblik og kunne lave væsentlige ændringer med henblik på at nå det midlertidige produkt.

For at undgå den lave produktivitet, kunne vi have lagt en bedre strategi. Vi kunne have fået dannet os et større overblik og overvejet om vi kunne nedbryde nogle klasser til flere klasser og evt. lave et interface. Vi har i gruppen ikke haft den store erfaring med at lave større programmer, og det bl.a. en af grundene til vi støtte ind på denne problematik. Gruppearbejdet har ellers fungeret godt og samarbejdsaftalen er tildels blevet overholdt. Der har været god kommunikation i gruppen og arbejdsmiljøet har været fornuftigt. Vi har en idé om at brugere der er bekendte med Google Maps umiddelbart kan finde ud af at bruge hovedparten af programmets features da mulighederne for interaktion ligner dem i Google Maps. Vi har dog ikke fået efterprøvet denne idé, eftersom vi ikke lavet bruger-tests i denne omgang, men overvejer kraftigt at gøre det i forbindelse med næste fase af projektet.

Konklusion

Bilag

8.1 Forventningstabeller

Vector

Tabel 8.1 – Vector:Constructor

Input data	Forventet output	Egentligt output
vec(1,2)	vec(1,2)	vec(1,2)

Tabel 8.2 – Vector:Set

Input data	Forventet output	Egentligt output
vec(2,3)	vec(2,3)	vec(2,3)

Tabel 8.3 – Vector:Copy

Input data	Forventet output	Egentligt output
vec(1,2)	vec(1,2)	vec(1,2)

Tabel 8.4 – Vector:Add

Input data	Forventet output	Egentligt output
vec(1,2)+vec(1,1)	vec(2,3)	vec(2,3)
vec(1,2)+vec(-1,-1)	vec(0,1)	vec(0,1)
vec(1,2)+vec(0,0)	vec(1,2)	vec(1,2)

Box

Tabel 8.5 – Vector:Subtract

Input data	Forventet output	Egentligt output
$\text{vec}(1,2) - \text{vec}(1,1)$	$\text{vec}(0,1)$	$\text{vec}(0,1)$
$\text{vec}(1,2) - \text{vec}(-1,-1)$	$\text{vec}(-1,-1)$	$\text{vec}(-1,-1)$
$\text{vec}(1,2) - \text{vec}(0,0)$	$\text{vec}(1,2)$	$\text{vec}(1,2)$

Tabel 8.6 – Vector:Multiplication

Input data	Forventet output	Egentligt output
$\text{vec}(1,2) * 2$	$\text{vec}(2,4)$	$\text{vec}(2,4)$
$\text{vec}(1,2) * 0.5$	$\text{vec}(0.5,1)$	$\text{vec}(0.5,1)$
$\text{vec}(1,2) * 1$	$\text{vec}(1,2)$	$\text{vec}(1,2)$

Tabel 8.7 – Vector:Division

Input data	Forventet output	Egentligt output
$\text{vec}(1,2) / 0.5$	$\text{vec}(2,4)$	$\text{vec}(2,4)$
$\text{vec}(1,2) / 2$	$\text{vec}(0.5,1)$	$\text{vec}(0.5,1)$
$\text{vec}(1,2) / 1$	$\text{vec}(1,2)$	$\text{vec}(1,2)$

Tabel 8.8 – Vector:Distance

Input data	Forventet output	Egentligt output
$\text{vec}(1,2)$ to $\text{vec}(2,3)$	$\sqrt{2}$	$\sqrt{2}$
$\text{vec}(1,2)$ to $\text{vec}(0,1)$	$\sqrt{2}$	$\sqrt{2}$
$\text{vec}(1,2)$ to $\text{vec}(1,2)$	0	0

Tabel 8.9 – Vector:Translate

Input data	Forventet output	Egentligt output
$\text{vec}(1,2)$ from $\text{box}(0,0;2,4)$ to $\text{box}(0,0;4,8)$	$\text{vec}(2,4)$	$\text{vec}(2,4)$

Tabel 8.10 – Vector:MirrorY

Input data	Forventet output	Egentligt output
$\text{vec}(1,2)$ in $\text{box}(0,0;1,3)$	$\text{vec}(1,1)$	$\text{vec}(1,1)$

Tabel 8.11 – Vector:Equals

Input data	Forventet output	Egentligt output
vec(7,21) og vec(7,21)	true	true
vec(7,21) og vec(21,7)	false	false
vec(7,21) og vec(0,-90)	false	false

Tabel 8.12 – Box:Constructor

Input data	Forventet output	Egentligt output
vec(1,2) og vec(3,4)	box(1,2;3,4)	box(1,2;3,4)

Tabel 8.13 – Box:Dimensions

Input data	Forventet output	Egentligt output
box(1,2;3,4)	vec(2,2)	vec(2,2)

Tabel 8.14 – Box:RelativeToAbsolute

Input data	Forventet output	Egentligt output
vec(0.5,0.5) i box(1,2;3,4)	vec(2,3)	vec(2,3)

Tabel 8.15 – Box:Ratio

Input data	Forventet output	Egentligt output
box(1,2;3,4)	vec(1,1)	vec(1,1)
box(0,0;2,1)	vec(1,0.5)	vec(1,0.5)
box(0,0;1,2)	vec(0.5,1)	vec(0.5,1)

Tabel 8.16 – Box:Scale

Input data	Forventet output	Egentligt output
box(1,2;3,4)*2	box(0,1;4,5)	box(0,1;4,5)

Tabel 8.17 – Box:FlipX

Input data	Forventet output	Egentligt output
box(11,0;-3,0)	box(-3,0;11,0)	box(-3,0;11,0)

Tabel 8.18 – Box:FlipY

Input data	Forventet output	Egentligt output
box(0,11;0,-3)	box(0,-3;0,11)	box(0,-3;0,11)

Tabel 8.19 – BoxProperCorners

Input data	Forventet output	Egentligt output
box(51,-10;0,80)	box(0,-10;51,80)	box(0,-10;51,80)