Programmering C

Eksamensprojekt

Indholdsfortegnelse

[Abstract 3](#_Toc40023799)

[Problemformulering 3](#_Toc40023800)

[Programmet 4](#_Toc40023801)

[GUI 4](#_Toc40023802)

[Events 5](#_Toc40023803)

[Position og arrayList 6](#_Toc40023804)

[Sorteringsmekanismen 7](#_Toc40023805)

[Knapperne 7](#_Toc40023806)

[Sortering 7](#_Toc40023807)

[Infobokse 8](#_Toc40023808)

[Zoom og panorering 8](#_Toc40023809)

[Test af programmet 10](#_Toc40023810)

[Forbedringer 10](#_Toc40023811)

[Konklusion 11](#_Toc40023812)

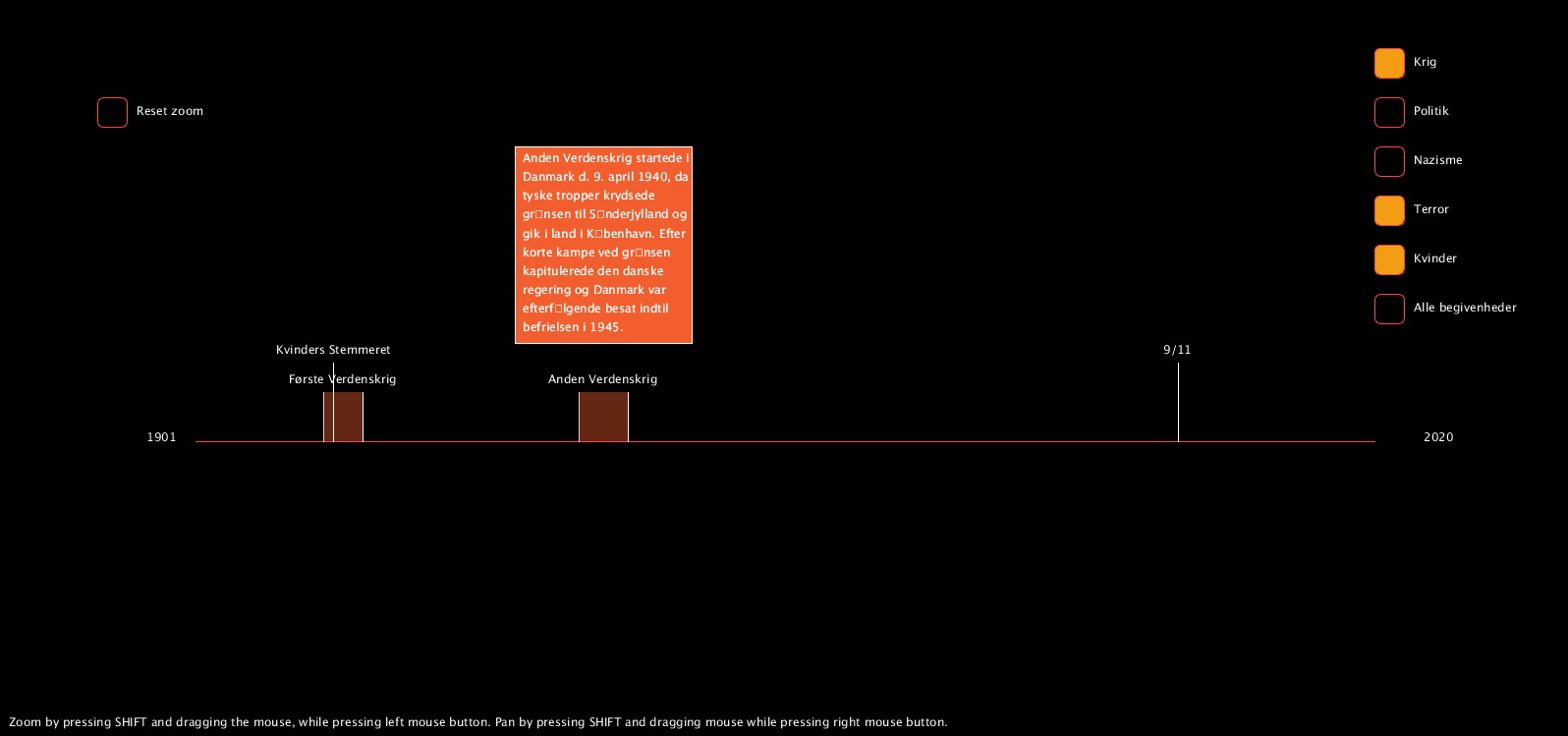
[Kildeliste 12](#_Toc40023813)

# Abstract

Denne opgave er lavet af Karla Jacobsen som eksamensprojekt i programmering C i 2.g på Aarhus Gymnasium, Aarhus C. Programmet er lavet i Processing 3.5.4 (Processing Development Environment (PDE)).

Programmet er en interaktiv tidslinje over Danmarkshistorien fra 1901 til i dag. Der kan zoomes ind og ud og der kan ved hjælp af knapper sorteres i hvilke begivenheder, man som bruger, gerne vil se ud fra forskellige tags, der hører til hver begivenhed. Det er også muligt for brugeren at trykke på de enkelte begivenheder og på den måde få mere information om den pågældende begivenhed.

Herunder ses et skærmbillede fra programmet, der giver et hurtigt overblik over de forskellige dele og funktioner.



# Problemformulering

Det kan være svært at få overblik over de vigtige historiske begivenheder, der har formet Danmark gennem tiderne. Et værktøj, man kan bruge til at gøre det nemmere, er en tidslinje. Dog er mange af de tidslinjer man kan finde i dag enten nogle man selv skal lave, eller også giver de kun det store eller det lille overblik og ikke begge dele. Derfor vil et program, der viser en tidslinje over historiske begivenheder i Danmark, som er interaktiv og kan zoome både ind og ud, være en god løsning på dette problem. Den vil give elever både det store og det lille overblik over hvilke begivenheder, der formede vores samfund hvornår. Et sådant program ville også kunne sortere i begivenhederne og f.eks. kun vise de begivenheder, der handler om krig, teknologi eller noget tredje. Der er dog nogle problemstillinger, der skal løses, for at dette kan udføres. De vigtigste af dem ses herunder:

* Hvordan gøres programmet overskueligt og brugervenligt?
* Hvordan zoomes der bedst ind og ud, så det ikke ødelægger overblikket?
* Hvordan gøres sorteringsmekanismen så smooth og god som muligt?

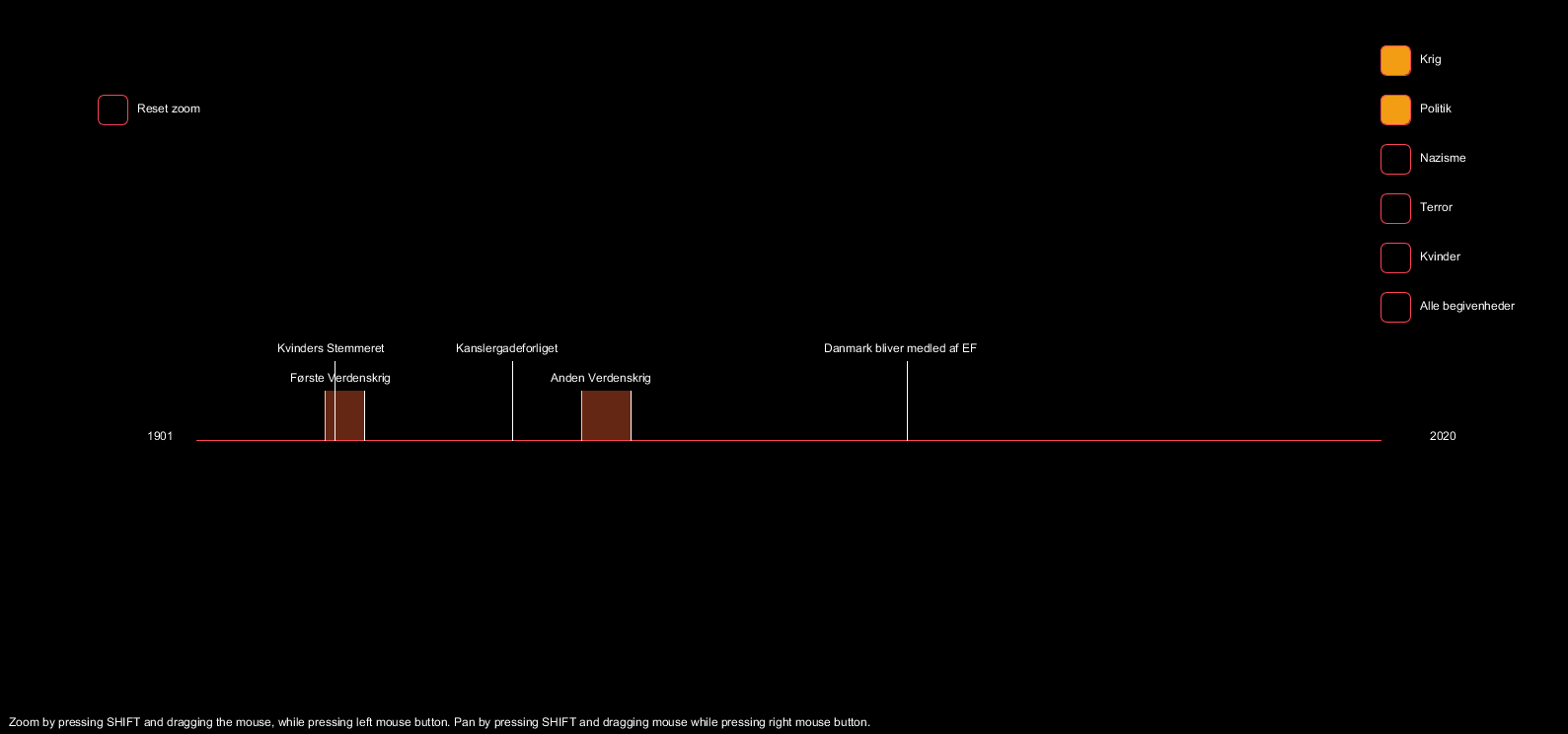
# Programmet

I dette afsnit vil selve programmet beskrives. Der vil bl.a. kommes ind på hvad brugeren ser når han/hun bruger programmet, programmets objekter og funktioner og programmet syntaks både på det overordnede og på underordnede niveauer.

## GUI

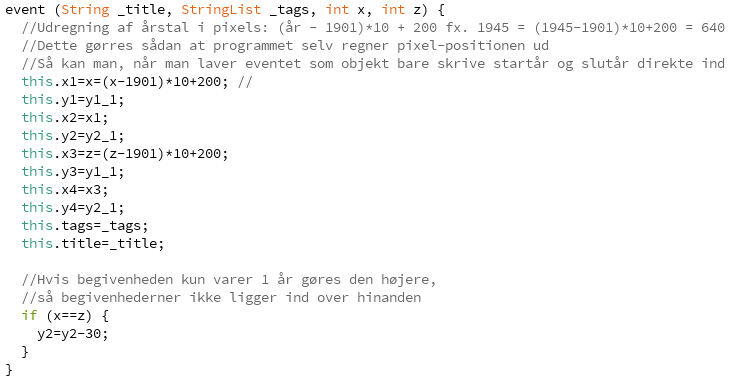
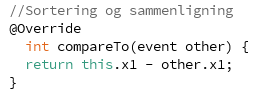
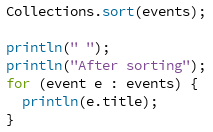
I dette afsnit beskrives den del af programmet som brugeren ser og interagerer med. Dette kaldes Graphical User Interface, GUI. I dette program kan GUI’en deles ind i fire dele: Selve tidslinjen, der kan zoomes ind og ud omkring, knapperne i højre side, hvor brugeren kan styre, hvilke begivenheder de gerne vil se, begivenhederne, der kommer frem når brugeren trykker på knapperne og informationsboksene der kommer frem når brugeren trykker på de forskellige begivenheder. Når brugeren åbner programmet, vil det se ud som på billedet herunder: 

Brugeren vil altså se en tom tidslinje og knapper i øverste venstre hjørne, som brugeren kan klikke på for at sortere i begivenhederne. Klikker brugeren på f.eks. knapperne krig og politik, vil der på tidslinjen fremkomme de begivenheder, der indeholder dette tag. Se billede herunder:



Har brugeren lyst til at zoome ind og ud omkring tidslinjen gøres dette ved tryk på SHIFT og trækning af musen op for at zoome ind og ned for at zoome ud. Brugeren kan også panorere. Dette gøres ved tryk på SHIFT og trækning af musen med tryk på højre museknap. I bunden af skærmen ses en tekst, der forklarer brugeren, hvordan zoom og panorering virker. Ved tryk på knappen ”reset zoom” i øverste højre hjørne, kan brugeren komme tilbage til udgangspunktet efter at have zoomet, hvis brugeren har lyst til det.

## Events

Den vigtigste del af de fire dele af programmet er de historiske begivenheder, da det er dem hele programmet drejer sig om, og det er det, som brugeren gerne vil have ud af programmet. For at lave begivenhederne er der blev brugt Objekt Orienteret Programmering også bare kaldet OOP. OOP er en rigtig god metode at bruge, hvis man skal lave mange af de samme objekter, der dog alligevel skal være forskellige. Dette gør man ved at oprette en Class og under denne class oprette forskellige objekter. I dette program hedder classen for historiske begivenheder *event* og under den laves forskellige historiske begivenheder ud fra forskellige variabler. I *constructoren* opstilles de variabler, som er klassens input parametre, der skal gives når de enkelte objekter oprettes i *void draw().* For de historiske begivenheder er disse variabler navnet på begivenheden, start og slutåret for begivenheden (varer begivenheden kun et år, sættes startåret og slutåret til det samme) og de tags der hører til begivenheden. For at undgå at navnet på begivenheder, der kun varer et år ikke kommer til at stå oveni begivenheder, der varer flere år, ændres y-positionen for toppen af den linje der danner begivenheden, hvis startåret og slutåret er det samme. På den måde flyttes navnet for disse begivenheder op over navnet for de andre begivenheder. Se figur 1.

### Position og arrayList

For at gøre det så nemt som muligt for programmøren at tilføje og overskue nye og eksisterende events, laves nye events ved at tilføje dem til arraylisten events. På den måde er alle events det samme sted og kan nemt hives ud af arraylisten ved at bruge eventets position i listen. For at gøre det nemt at finde ud af hvilke begivenheder, der har hvilke positioner sorteres eventsne efter startår (x1-position). Se figur 1 og 2. For at gøre det endnu nemmere for programmøren at tilføje events, regner programmet selv pixel-værdien ud for et givent årstal mellem 1901 og 2020. Dette gøres vha. udregningsmodellen:

Eksempel med årstallet 1945:

Det skal dog siges at denne udregningsmodel kun gælder når tidslinjen har længden 1200 pixels.

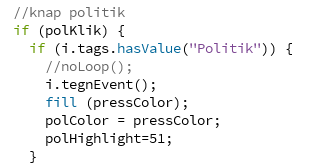
## Sorteringsmekanismen

I dette afsnit bliver sorteringsmekanismen og de knapper, brugeren skal trykke på for at sortere gennemgået.

### Knapperne

Knapperne ses som sagt i øverste højre hjørne og i øverste venstre hjørne. Knappen i øverste venstre hjørne er ”reset zoom” knappen og den vil blive beskrevet i afsnittet Zoom. I dette afsnit er der fokus på de resterende knapper og deres funktion. Hver knap, der ses i højre side, repræsenterer et specifikt tag. Når brugeren trykker på knappen, vises der på tidslinjen de events, der indeholder dette tag. Knapperne består af en firkant, med runde hjørner og et navn. Når musen befinder sig over firkanten, farves denne en Highlight farve, sådan at brugeren kan se at hans/hendes mus er over knappen.

### Sortering

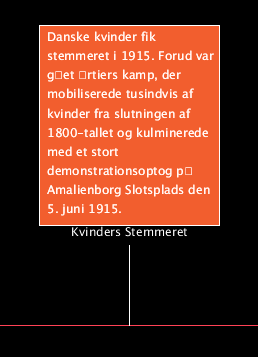
Når brugeren så trykker på knappen, går sorteringen i gang. Processen for dette kan ses på figur 4. Ved hjælp af booleans tjekkes det først om musen er over knappen. Vha. if-sætninger gøres det sådan, at hvis dette er tilfældet, farves knappen highlightColor. Derefter tjekkes det om knappen er blevet trykket på og hvis den er, itereres der over alle events i arraylisten med et enhanced loop, og hvis disse indeholder det korresponderende tag til knappen, tegnes de. Det er den indbyggede boolean *hasValue(value)*, der bruges til dette. Den kan sige om en stringList indeholder en bestem String. For et eksempel på hvordan dette bruges i programmet se figur 5. Når brugeren derefter gerne vil fjerne de tegnede events igen, gøres dette ved endnu et tryk på knappen. Efter knappen er blevet trykket på første gang ændres highlight farven. Dette gør, at man ved at sige at ”Knappen er trykket på” skal være true og ”highligt farve = den nye farve” skal være true, kan gøre den boolean, der tegner selve eventet i void Draw(), false igen og dermed fjernes eventet.

Figur 5: Eksempel på hvordan hasValue bruges til at sortere i eventsne.

Figur 4: Flowdiagram over sorteringsmekanismen

Det er altså bl.a. knapperne, der gør programmet interaktivt, hvilket var et af målene i problemformuleringen. Brugeren kan altså selv bestemme hvilke begivenheder han/hun vil se udfra hvilke tags, der falder i hans/hendes interesse. Der findes også en ”Alle begivenheder” knap, der, hvis man trykker på den, viser alle begivenheder på tidslinjen.

## Infobokse

Når brugeren så har fundet et event der er interessant, kan han/hun trykke på eventet og der vil derefter poppe en infoboks op, hvori der står information om det pågældende event. Disse infobokse er lavet som en klasse, der hedder info. At tjekke om eventet er blevet trykket på fungerer præcis ligesom ved knapperne. De parametre der bestemmer, hvor der kan trykkes for at få infoboksen frem bestemmes bare af eventets variabler i stedet for variabler, der er lavet specielt til knappen, ligesom det er tilfældet med de andre knapper. Disse parametre hentes ud af arraylisten med events vha. den indbyggede funktion *get().* Der vil selvfølgelig være et problem for de events, der kun varer 1 år og dermed kun er en streg, da det af den grund kan være svært for brugeren at ramme præcist på linjen med musen. Derfor blev der lagt 10 pixels til på hver side af eventet, sådan at området man skal ramme bliver større.

Billede 1: Eksempel på infoboks. Som det ses, er Æ, Ø og Å byttet ud med små firkanter.

Når man så har trykket på eventet, popper infoboksen altså frem. Den tekst, der står i boksen hentes af programmet fra eksterne tekstfiler (.txt). Dette gør det nemt at tilføje, fjerne og ændre i teksten uden at man skal ind og rode i koden. Når Processing loader tekstfiler på den her måde, loades de som et array af Strings. Dette duer ikke når man gerne vil skrive dem som tekst i GUI’en, da den funktion, der gør dette ikke tager imod et array af Strings som input parameter. Derfor benytter man den indbyggede funktion *join(String[], seperator)* til at samle arrayet til én String, der kan bruges som input parameter i funktionen *text()*. Desværre kan Processing ikke finde ud af at læse Æ, Ø og Å, og disse bliver derfor vist som små firkanter i stedet for. Dette kan virke lidt forstyrrende, men det er heldigvis forholdsvis nemt at læse teksten alligevel.

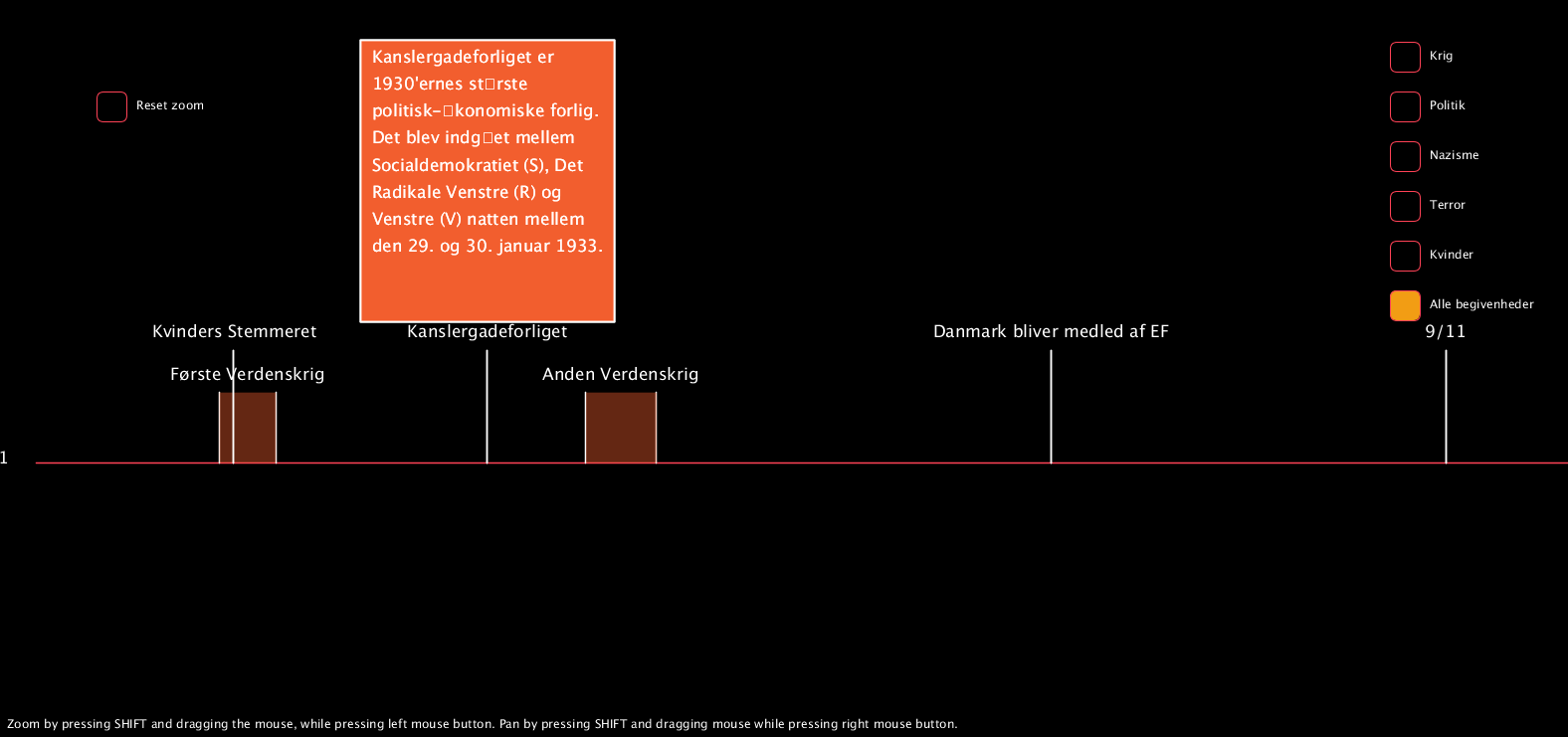
Nogle af teksterne er skrevet af mig (Karla), mens andre er hentet fra Danmarkshistorien.dk og modificeret af mig. Kilder til disse kan ses under kildelisten.

Der var et problem med at nogle af infoboksene overlappede og at man på den måde ikke kunne læse nogen af dem. Dette problem blev løst ved vha. booleans at sige, at den ene infoboks ikke kunne vises, hvis den anden blev vist. Det blev gjort på en sådan måde, at når man som bruger trykker på det ene event kommer infoboksen frem for dette event, mens den forsvinder for det andet event, der lå oveni.

## Zoom og panorering

Den sidste del af programmet er zoom og panorering. Denne funktion er tilføjet fordi en af hoveidéerne med dette program er, at man som bruger får en fornemmelse af hvilke historiske begivenheder, der skete hvornår, og hvordan de indbyrdes ligger i forhold til hinanden i det rigtige målestoksforhold. Derfor er det også vigtigt at zoom ikke ændrer på begivenhedernes position, men blot ændrer på hvor (0,0) er, og på den måde viser brugeren tidslinjen endnu tættere på, sådan at han/hun både kan få det store og det lille overblik.

Zoom er i dette program blevet lavet vha. et library, der hedder giCentre. Skaberne bag dette har lavet et library man kan hente og på den måde kan man bruge den zoom-funktion de allerede har lavet. Til denne zoom-funktion findes også en del metoder, der gør at man kan modificere zoom-funktionen sådan at den passer til ens program. Der findes bl.a. *setMouseMask()*, der gør at man kan bestemme at zoom kun sker, hvis der trykkes på musen, men samtidig trykkes på en bestemt tast. Denne tast kan være SHIFT, ALT eller CONTROL. Dette gøres sådan, at der ikke zoomes og panoreres når musen bruges til andre ting så som knapperne i dette program.



Herover ses et skærmbillede, hvor der er blevet zoomet ind. Som det ses, er det kun selve tidslinjen, eventsne og infoboksene, der zoomes på og ikke knapperne og den forklarende tekst i bunden. Dette gøres vha. af *pushMatrix()* og *popMatrix()*. *PushMatrix()* gemmer den aktuelle transformationsmatrix og *popMatrix()* finder denne frem igen. På den måde kan man styrer hvilke scopes man gerne vil have bliver transformeret, altså zoomet på, og hvilke der ikke gør.

For at gøre det nemt for brugeren at få programmet tilbage til udgangspunktet og altså få (0,0) tilbage til øverste venstre hjørne bruges metoden *reset()*, der er en indbygget metode i zoom fra giCentre. I øverste venstre hjørne ses knappen ”reset zoom”. Denne knap virker præcis ligesom alle de andre knapper og når man trykker på den, resettes zoom vha. en if-sætning.

# Test af programmet

I dette afsnit testes programmet og der er også et underafsnit hvori, der kommer forslag til hvordan programmet kan forbedres, hvis det skulle videreudvikles.

## Forbedringer

# Konklusion

# Kildeliste