Eksamensprojekt i programmering C

Anton Christensen

2.x

HTX-Aabenraa

Programmering C

Christian Bruhn

30.04.2023

Indholdsfortegnelse

[2 Indledning 3](#_Toc133784156)

[3 Metode 3](#_Toc133784157)

[3.1 UML-diagram 3](#_Toc133784158)

[3.2 Model-View-Controller 4](#_Toc133784159)

[3.2.1 Model 4](#_Toc133784160)

[3.2.2 View 4](#_Toc133784161)

[3.2.3 Controller 4](#_Toc133784162)

[3.3 Rutediagram 5](#_Toc133784163)

[3.4 Processing 5](#_Toc133784164)

[3.5 Programmeringsparadigme 6](#_Toc133784165)

[3.5.1 Sekventiel programmering 6](#_Toc133784166)

[3.5.2 Procedural programmering 6](#_Toc133784167)

[3.5.3 Objektorienteret 6](#_Toc133784168)

[3.6 ControlP5 6](#_Toc133784169)

[4 Analyse 7](#_Toc133784170)

[5 UML class diagram 8](#_Toc133784171)

[5.1 Rutediagram 9](#_Toc133784172)

[5.2 Test af programmet 12](#_Toc133784173)

[5.3 Forbedringer som kan laves 14](#_Toc133784174)

[6 Konklusion 14](#_Toc133784175)

[7 Bilag 15](#_Toc133784176)

[7.1 Referencer 15](#_Toc133784177)

[7.2 Rutediagram 16](#_Toc133784178)

[7.3 UML class diagram 17](#_Toc133784179)

[7.4 Koden 17](#_Toc133784180)

# Indledning

Programmering i tekstbaserede programmeringssprog kan være udfordrende at sætte sig ind i. De grundlæggende færdigheder er de vigtigste at have greb om for at kunne videreudvikle sine programmeringsevner, da det er disse som ligger til grund for den videre læring.

I min HTX-programmeringsklasse observerede jeg, i starten af programmeringsforløbet, at elever uden eksisterende erfaring med programmering havde udfordringer. Problemerne lå ved overblikket over de enkelte kommandoers funktion, samt håndtering af variabler og overblik over programmets struktur.

Dette problem kunne løses ved at fremstille et visuelt programmeringssprog som indeholder basale programmeringsfeatures. Hvis elever kan starte med at lære basis og derigennem få forståelse for hvordan der skal arbejdes med programmering i tekstbaserede sprog, vil det hjælpe dem videre med deres læring.

For at løse problemet er en problemformulering opstillet:

Hvordan kan jeg ved hjælp af programmering fremstille et simpelt blokprogrammeringssprog som kan hjælpe nye elever med at få basal erfaring med programmering?

# Metode

## UML-diagram

Et UML-diagram eller Unified Modeling Language, er en standardiseret måde at visualisere et system. UML-diagrammer er et bredt begreb da det dækker over mange forskellige diagramtyper, i dette projekt anvendes diagramtypen ”class diagram”. Et class diagram anvendes til at få overblik over de klasser som er med i programmet og hvilke relationer klasserne har til hinanden.

Et billede, der indeholder bord

Automatisk genereret beskrivelseDiagrammet er opbygget af former. Klasse-formen (Figur 1) er et rektangel opdelt i tre dele: Klasses navn, attributter og metoder. Oftest undlades getters og setters at noteres. Interaktionerne i programmet visualiseres ved at forbinde klasser med pile. En hul pil betyder nedarvning og en fyldt pil betyder at den første klasse anvender informationer fra den anden.

Figur 1: Klasse-formen

Med et class diagram kan strukturen, klasserne og interaktionerne i et program let overskues, hvilket kan hjælpe i programmeringsfasen.[[1]](#footnote-2)

## Model-View-Controller

Model-View-Controller eller MVC opdeler et program i tre dele for at gøre programmet mere overskueligt og derved også lette fejlfinding.

### Model

Model har med data-delen af et program at gøre det vil sige at hente data fra en database, manipulere det og uploade data. I et mindre lokalt program som denne rapport omhandler, kommer dataene ikke fra en database, men model er stadig den data-ansvarlige.[[2]](#footnote-3)

### View

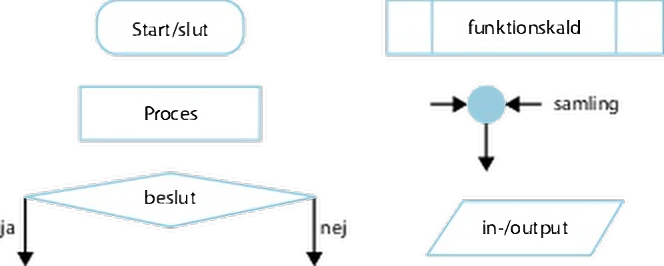
View er ansvarlig for alt det visuelle i et program herunder den grafiske brugerflade som brugeren skal interagere med. [[3]](#footnote-4)

### Controller

Controlleren står for logikken i programmet altså hvornår hvilke funktioner og metoder skal kaldes. Den interagerer med Model for at manipulere data og interagere med brugerfladen ved hjælp af View.[[4]](#footnote-5)

## Rutediagram

Et rutediagram er en visualisering af den rute som en del af et program skal igennem for at komme fra et input til det ønskede output. Et rutediagram fremstilles for at tanker kan gøres om koden før den skrives. Når tankerne er gjort, er det lettere at skrive koden. Et rutediagram opstilles med former som beskriver forskellige funktioner i koden, de enkelte former forbindes så ved hjælp af pile for at vise hvordan programmet skal forløbe, se Figur 2.



Figur 2: Figurer som anvendes i et rutediagram

Det visuelle element gør rutediagrammer til et godt kommunikationsmiddel, da flere personer kan følge med i ruten og den tilhørende tankestrøm.[[5]](#footnote-6)

## Processing

Processing er en gratis IDE som er udviklet til nye programmører for at undervise i programmering, med en visuel sammenhæng.

Processing baserer sig på programmeringssproget Java, med enkelte forsimplinger, hvilket vil sige at det er meget af Java’s syntax som anvendes når der skrives kode. Alt kode i Java skal køres fra en klasse, i Processing er dette ikke nødvendigt, da man skriver i en ”sketch” som repræsenterer den oprindelige Java-class.[[6]](#footnote-7)

I dette projekt er der anvendt Processing.

## Programmeringsparadigme

### Sekventiel programmering

Sekventielle programmeringssprog understøtter kun ét kontrolforløb. Det vil sige at der ikke kan blive manipuleret med rækkefølgen som koden bliver udført i. Sekventiel kode vil starte på første linje, udføre koden og gå til næste linje, dette vil fortsætte indtil alle linjer med kode er udført. Fordelen ved sekventiel programmering er at det er nemt at få overblik over hvad der sker.[[7]](#footnote-8)

### Procedural programmering

Procedural kode er opbygget af et sæt funktionskald og et sæt funktioner som udfører en bestemt handling. Funktionerne kan genbruges forskellige steder i koden. For at styre hvornår hvilke funktioner bliver kaldt anvendes forskellige kontrolstrukturer som if-else statements og løkker. [[8]](#footnote-9)

### Objektorienteret

I Objektorienteret programmering (OOP) er koden opbygget af objekter som hver har et sæt attributter som beskriver objektet og tilhørende metoder som beskriver hvordan objektet opfører sig. OOP er en modulær måde at programmere på da objekter kan laves ud fra en klasse som virker som en skabelon for objektet og derved genbruge koden. Når der oprettes et objekt ud fra en klasse, kaldes objektet for en instans af klassen.[[9]](#footnote-10)

## ControlP5

ControlP5 er et GUI (Graphical user interface) bibliotek skrevet af Andreas Schlegel. Biblioteket tilføjer forskellige slags controllers som Tekstfelter, knapper, sliders osv.[[10]](#footnote-11) Dette bibliotek anvendes til at lave tekstfelter som brugeren kan interagere med og derigennem indtaste værdier som programmet kan anvende.

# Analyse

For at finde udfordringerne ved at starte med at programmere blev enkelte personer i klassen interviewet om deres udfordringer ved at starte med at programmere. De udfordringer som var tydeligst var:

* Viden om de enkelte instruktioner
* Håndtering af variabler
* Overblik over programmets struktur
* Overblik over programmets forløb

For at gøre det let at lære om disse aspekter i tekstbaseret programmering kan en forsimplet visualisering med blokke af kode fremstilles. Denne forsimplede visualisering kan med givne instruktioner give en grundlæggende forståelse af hvordan et program forløber, samt et overblik over programmet. For at et program kan løfte denne opgave skal det have følgende funktionalitet:

* Oprette figurer
* Ændre farven på figurer
* Oprette og ændre variabler
* Visualisere programmets forløb
* Være sekventiel

For at være helt afstemt med hvad programmet skal kunne laves en beskrivelse af funktionaliteten af blokprogrammerings-programmet:

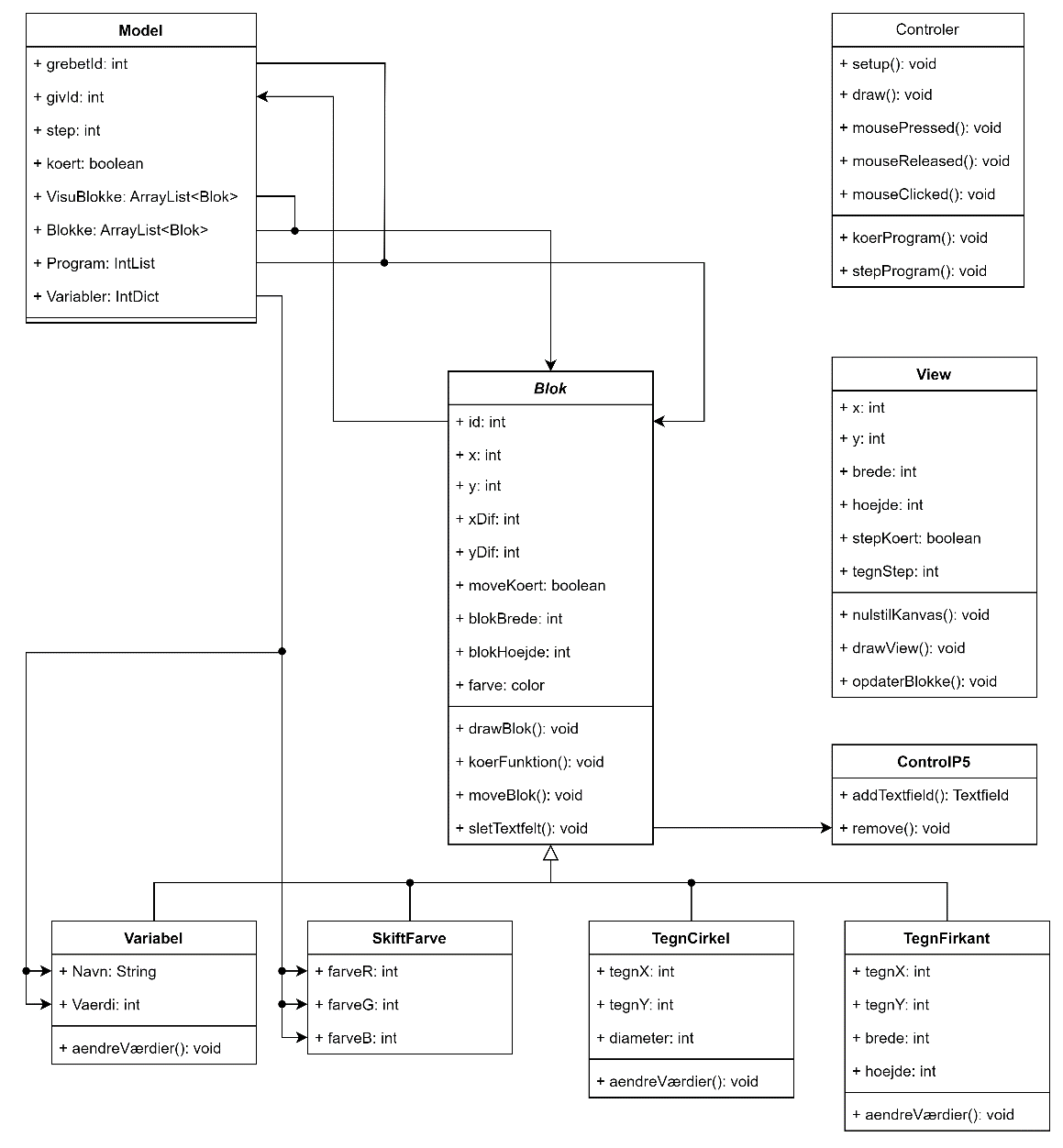
Programmet skal have en Programmeringsliste, denne liste er tom når programmet starter. Brugeren skal kunne trække instruktionsblokke fra en menu i venstre side af skærmen og sætte dem på Programmeringslisten i en ønsket rækkefølge. Der trykkes på en play-knap og det af brugeren lavede program kører instruktionsblokkene sekventielt. Programmet visualiseres på et kanvas i højre side af skærmen. En step-knap skal kunne trykkes som kører det lavede program en instruktion ad gangen. For overblik skal der være mulighed for at nulstille hele programmet eller bare kanvas.

Ud fra denne beskrivelse og den ønskede funktionalitet konkluderes det at der er brug for to klasser: En klasse til at styre det visuelle i programmet og en til at lave blokkene. Blokkene kan laves som en superklasse med nedarvning til de enkelte specifikke instruktionsblokke.

Koden kommer til at være en blanding mellem procedural og objektorienteret kode, da Controlleren skrives procedural.

# UML class diagram

For at finde strukturen er et UML class diagram fremstillet, se Figur 3 en større version kan ses i bilag 7.3.



Figur 3: UML-class-diagram

Dette UML class diagram er den sidste iteration, undervejs i fremstillingen af programmet er der tilføjet og ændret på relationer mellem objekter, attributter og metoder. Ændringerne er sket eftersom jeg har arbejdet iterativt i udviklingen.

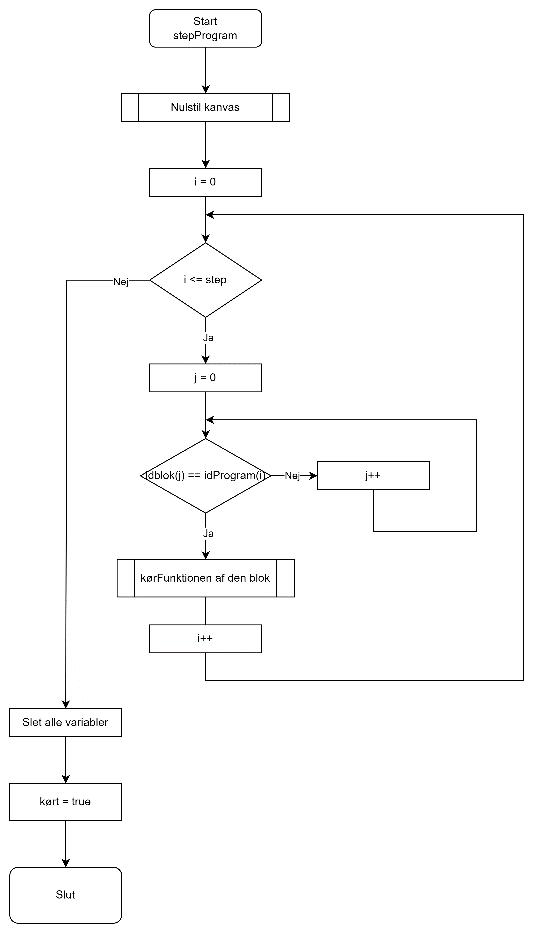
ControlP5 er en klasse for sig selv men metoderne bruges af Blok for at lave tekstfelter og få værdier fra brugeren. Eftersom ControlP5 er en GUI burde den være del af eller virke igennem View da det er View som står for alt visuelt, det gør den dog ikke i mit program da jeg havde problemer med implementeringen af ControlP5.

I programmet er Model og Controller skrevet sammen, da model kun står for at hente og manipulere data.

## Rutediagram

Med programmets struktur nedskreven kan programmets funktioner kodes.

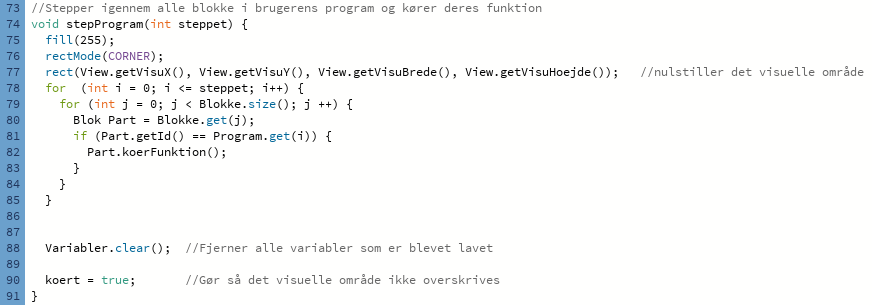
Rutediagrammet for at steppe igennem prokrammet kan ses herunder på Figur 4, en større version kan ses i bilag 7.2.



Figur 4: Rutediagram for funktion stepProgram

For at steppe igennem programmet nulstilles kanvas først og så køres programmet fra start og indtil det step som programmet er kommet til. Som kan ses på rutediagrammet bliver der anvendt to for-løkker, en inden i den anden. Den yderste for-løkke holder styr på hvilket index programmet er nået til og den inderste for-løkke looper alle eksisterende blokke igennem og sammenligner deres Id med det Id som programmet er nået til, hvis de er ens køres den bloks funktion. Efter blokkens funktion er kørt tælles indexet i programmet op og det hele gentager sig igen. Funktionen stopper når der tælles en over step for så er det ønskede antal funktioner kørt. Til sidst nulstilles Variablerne, dette gøres for at forhindre at en initieret variabel eller farve påvirker den næste kørsel af programmet og eftersom programmet køres fra starten hver gang bliver variablerne initieret igen. Til sidst bliver kørt sat til true så kanvas ikke bliver overskreven.

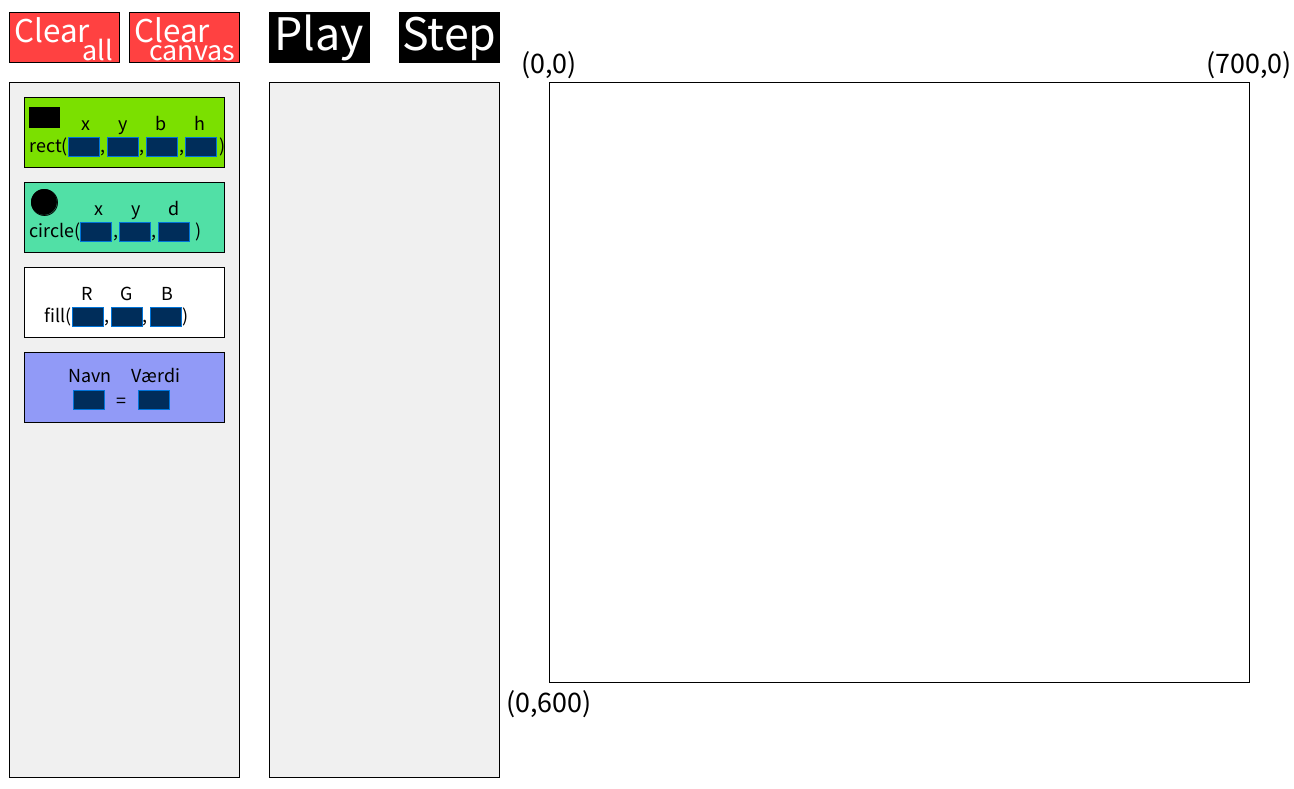
Koden skrevet i Processing kan ses herunder på Figur 5.



Figur 5: Funktion for at steppe igennem programmet

I udviklingen af de andre funktioner og metoder i Programmet er der også anvendt rutediagrammer, de er for det meste skrevet i hånden og har hjulpet med hurtigt at danne overblik over hvordan funktion/metoden kunne kodes.

Det færdige program ser ud som på Figur 6 når det åbnes.



Figur 6: Programmet som det åbnes

## Test af programmet

For at teste om mit program lever op til de krav som der blev sat testes det på en person uden programmeringserfaringer. Under testen observerede jeg kun og gav ikke tips.

Personen fandt hurtigt ud af hvordan instruktionerne rect() og circle() virkede og hvordan parametre blev indtastet. Det var intuitivt at blokke blev slettet når de blev trukket tilbage på menuen i venstre side og at nye blev skabt når man trak dem ud fra listen.

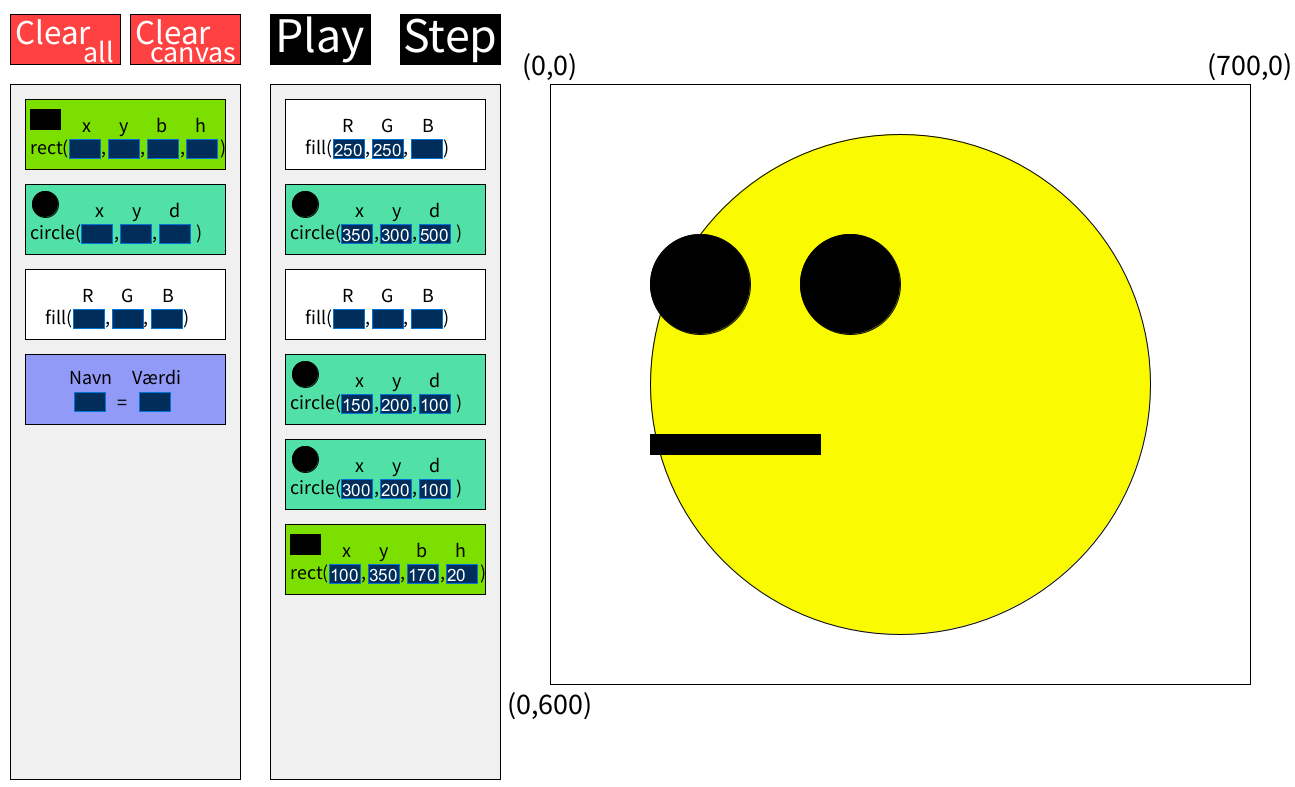
At skifte farve virkede også intuitivt bortset fra at testpersonen havde problemer med at ramme den ønskede farve.

Variabel-blokken virkede ikke intuitivt men da funktionaliteten var forstået, var denne også intuitiv.

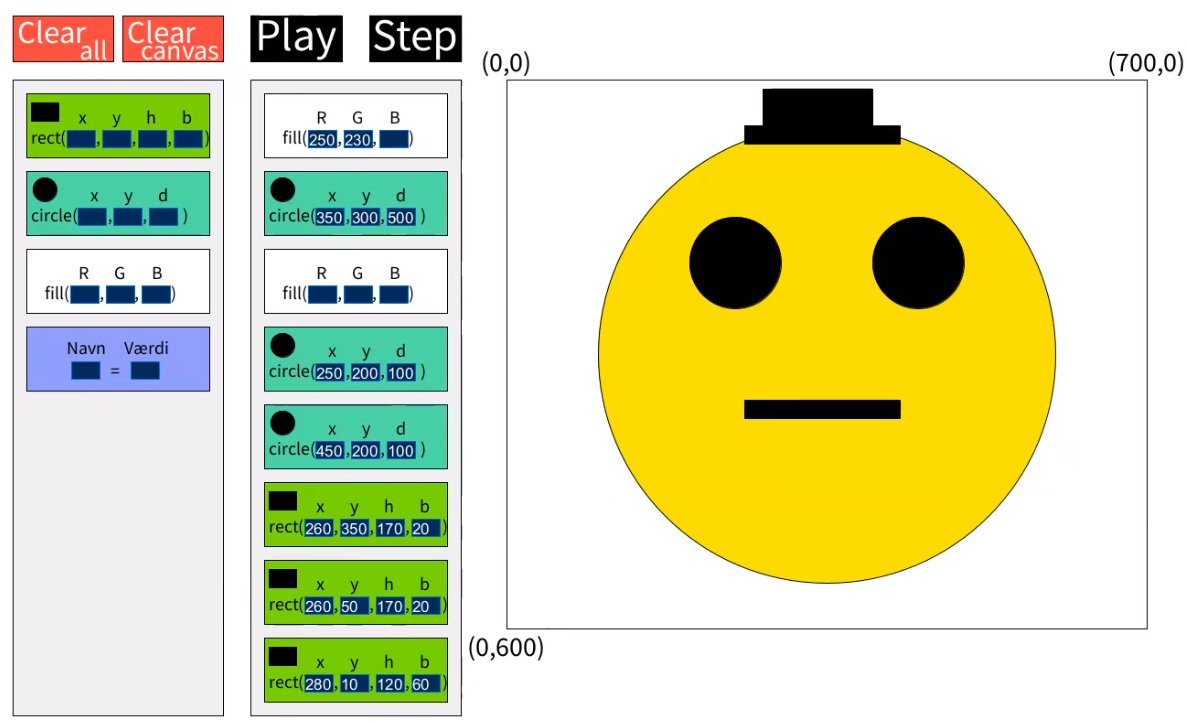
Det var et stort ønske fra testpersonen at der kunne ændres på rækkefølgen af blokkene da der på nuværende tidspunkt ikke er muligt, hvilket hindrede den gode oplevelse men ikke var katastrofal.

Step-funktionaliteten mentes at give et godt overblik og god forståelse af hvordan programmet forløb. Hvilket ifølge brugeren ”Virker ret godt”.

Efter 10 min have testpersonen fremstillet en smiley som kan ses på Figur 7 og efter 20 min fremstillet en smiley med hat som kan ses på Figur 8.



Figur 7: Fremstillet af testpersonen efter 10min



Figur 8: Smiley med hat, fremstillet efter 20min

Den endelige kommentar var: ”Det er ret sjovt!”. Det kan derfor siges at programmet er lykkedes med at opnå de opstillede mål.

## Forbedringer som kan laves

Hvis programmet skulle gøres bedre, kan følgende ting, som blev fundet til at være forhindringer under test, tilføjes:

* Bedre forklaring af variabel-blok
* Gøre det muligt at ændre rækkefølgen af blokke i programmet
* Gøre det muligt at regne med variabler

# Konklusion

Målet med at programmere et blokprogrammeringsprogram kan siges at være nået. Målene for programmet var:

* Oprette figurer
* Ændre farven på figurer
* Oprette og ændre variabler
* Visualisere programmets forløb
* Være sekventiel

Den ønskede funktionalitet blev opnået og igennem en test med en ikke-programmeringserfaren person kunne det ses at de udfordringer som er til stede når man starter med at programmere, ikke var til stede, i lige så stort omfang. De enkelte instruktioner kunne forstås, variabler kunne oprettes og anvendes, der kom overblik over strukturen og step-funktionen hjalp med overblikket over programmets forløb.

# Bilag

## Referencer

Buch, J. (2023). *5.3 Rutediagrammer*. Hentet 30. april 2023 fra https://programmering.systime.dk/: https://programmering.systime.dk/?id=148

Computer Hope. (10. november 2021). *Processing*. Hentet 30. april 2023 fra computerhope.com: https://www.computerhope.com/jargon/p/processi.htm

Lucidchart. (u.d.). *UML Class Diagram Tutorial*. Hentet 30. april 2023 fra lucidchart.com: https://www.lucidchart.com/pages/uml-class-diagram

Nørmark, K. (1994). *Aarhus universitet.* Hentet 30. april 2023 fra Homes: https://homes.cs.aau.dk/~normark/ps1-94-notes/pdf/grund.pdf

Spot the difference. (u.d.). *Forskel mellem Procedural, Structural og Object Oriented Programmeringssprog*. Hentet 30. april 2023 fra da.spot-the-difference.info: https://da.spot-the-difference.info/difference-between-procedural

## Rutediagram

Et billede, der indeholder diagram

Automatisk genereret beskrivelse

## UML class diagram

Et billede, der indeholder diagram

Automatisk genereret beskrivelse

## Koden

Hele koden er vedlagt i afleveringsmappen, filerne har navnene ”Blokke.pde”, ”BlokProgrammering.pde” og ”View.pde” de skal i en mappe kaldet ”BlokProgrammering” og så kan de åbnes.

1. (Lucidchart, u.d.) [↑](#footnote-ref-2)
2. (Tutorialspoint, u.d.) [↑](#footnote-ref-3)
3. (Tutorialspoint, u.d.) [↑](#footnote-ref-4)
4. (Tutorialspoint, u.d.) [↑](#footnote-ref-5)
5. (Buch, 2023) [↑](#footnote-ref-6)
6. (Computer Hope, 2021) [↑](#footnote-ref-7)
7. (Nørmark, 1994) [↑](#footnote-ref-8)
8. (Spot the difference, u.d.) [↑](#footnote-ref-9)
9. (Spot the difference, u.d.) [↑](#footnote-ref-10)
10. (Schlegel, 2016) [↑](#footnote-ref-11)