Christian Bruhn

IMPRISONED

Escaperoom

Kanusha Anandakumar 2.x

Teknisk Gymnasie Sønderborg

Programmering

Gruppemedlemmere:

Julius J. V. G. Thomas

Rasmus Bakman

Afl. 06.06.2021

Indholdsfortegnelse

1. Opgave formulering

For at kunne udvikle en opgave formulering, blev der afleveret en problemformulering der virkede passende, som skulle godkendes af læreren, hvorefter resten af opgaven kunne udformes. Efter den blev godkendt, var der mulighed for at ændre den hvis der i gruppen blev besluttet om at den skulle ændres. Efter hele processen for at udvikle en opgaveformulering, er der i følgende opgave valgt at fokusere på følgende problemformulering:

Hvordan skal controlleren tage imod brugerens interaktioner, og skifte til den korresponderende klasse med nye funktioner, uden at de forrige klassers-funktioner kører?

- Hvilke metoder skal tages i brug for at deaktivere nogle klasser, og aktivere andre?
- Hvornår skal controlleren skifte klassen?
- Hvordan skal controlleren deaktivere funktioner som hører til en specifik klasse, og aktivere andre?
- Hvordan skal controlleren få informationer fra modellen og sende det videre til viewet?

Disse problemer opstod under forarbejdningen, hvor der var nogle problemer som ikke kunne løses uden yderligere undersøgelse. Disse problemer blev delt op i Model View Controller, som forklares yderligere længere nede, hvor disse spørgsmål høre til controlleren. Disse spørgsmål besvares gennem denne rapport, ved hjælp af analyse.

2. Indledning

I denne opgave startede hele klassen ud med at diskutere mulige problemer der skulle løses ved hjælp af softwareudvikling. Efter dette diskuterede de individuelle grupper om hvilke emner der var inden for deres interesseområder. I denne gruppe var interesseområdet indenfor spil udvikling. Gruppen endte med at have to emner og de var enten *rollespil* eller *escaperoom*. Der blev stemt og flertallet stemte på escaperoom, og blev derfor det der skulle udvikles. Til at starte med diskuterede gruppen lidt om hvordan spillet skulle gribes an, og endte med at lave lidt af et rutediagram, hvorefter pseudokoden blev skrevet. Pseudokoden var med til at give den største ide over hvordan spillet skulle udformes, og hvad der skulle være med for at spillet var komplet. Derefter blev der diskuteret angående hvilket ansvar de forskellige gruppemedlemmer havde, da der blev arbejdet ud fra MVC. Efter en del diskussion blev de forskellige ansvarsområder delt op, så alle i gruppen havde noget at arbejde med. I denne opgave er der fokuseret på controlleren, og hvordan den kan løse de problemer der opstod under udviklingen. Ud fra den ide der var omkring spillet, blev der skabt forskellige problemformuleringer inden for de forskellige ansvarsområder. I dette tilfælde havde controlleren følgende problemformulering:

> Hvordan skal controlleren tage imod brugerens interaktioner, og skifte til den korresponderende klasse med nye funktioner, uden at de forrige klassers-funktioner kører?
>
> - Hvilke metoder skal tages i brug for at deaktivere nogle klasser, og aktivere andre?
> - Hvornår skal controlleren skifte klassen?
> - Hvordan skal controlleren deaktivere funktioner som hører til en specifik klasse, og aktivere andre?
> - Hvordan skal controlleren få informationer fra modellen og sende det videre til viewet?

Ud fra denne problemformulering er der blevet valgt nogle metoder, som senere bruges til at analysere problemerne. Disse metoder skal være med til at besvare problemstillingerne.

3. Metode

For at kunne lave spillet blev der brugt nogle metoder som skulle gøre det nemmere at løse problemerne der blev opstillet.

Den første var sproget der blev brugt, og det var Java. Dette Java var sproget som skulle bruges til opgaven. Dette sprog bruges da det har været et krav for denne eksamensopgave. Denne metode anvendes til at lave kodningen til spillet.

Rutediagram bliver brugt, for at give et overblik over hvordan spillet skal fungere. Det er i denne fase der kan ses, hvad der skal gentages, være i en if statement eller andet. Dette bliver brugt før programmeringen, da det er med til at give et overblik over hvordan spillet skal forgå. Dette kan give et visuelt overblik.

Der er også blevet anvendt Pseudokode, til at skabe et større overblik. Dette blev brugt før programmerings-fasen, da dette er hvad programmet overordnet skal kunne, og hvornår det skal kunne udføre de forskellige instruktioner. Dette er en mere detaljeret version af rutediagrammet, og skrives med ord.

Da produktet skulle laves i en gruppe af 3, var den optimale måde at dele koden op på med MVC. Model View Controller er en måde at dele programmet op mellem folk på, for at gøre det nemmere at fordele arbejdet imellem hinanden. Denne metode bruges under hele programmeringen. Modellen står for alt dataen, viewet står for alt det visuelle på skærmen, og controlleren står for alt logikken.

Hele denne proces er en iterativ proces, da der altid er plads til forbedringer. Hvis noget i programmet ikke fungere, skal pseudokoden ændres, som kan betyde at rutediagrammet også skal ændres. Denne proces bruges helt fra udviklingen af ideen til det færdige produkt.

Gamestate er en String som ikke er en indbygget komando i processing. Det er en variable der er skabt af brugeres, og kunne hedde hvad som helst. Den kan bruges til at skifte mellem de forskellige funktioner eller klasser som skal vises på skærmen. Der kan gives et bestemt ord som skal kaldes, når en bestemt funktion eller klasse skal aktiveres. Dette er meget ensartet til en case, som kunne være brugt i stedet for dette.

En globale variabel er når en variabel kan kaldes i hele programmet. Alle klasser og funktioner har mulighed for at kunne vide hvilken værdi den er tildelt, og kan også ændre den hvis nødvendigt. Dette bruges når billederne skal loades i model klassen, men skal kaldes ind i view klassen. Da klasser kun kan lave variabler, som kan kaldes af controlleren eller klassen selv er det ikke muligt at vise billederne i viewet. Dette er grunden til at der bruges globale variabler. For at gøre en variabel global deklereres variablen i controlleren, som giver alt adgang til den.

4. Hoveddel

4.1 Forarbejdning

Før problemstillingerne kan besvares ved analysering, skal der redegøres for processen først. Til at starte med blev der lavet et rutediagrammet som gav et overblik der kunne hjælpe under programmeringen af escape-rummet. Rutediagrammet gav et overblik over hvornår hvad skulle aktiveres, og hvilken interaktion der skulle få en ændring til at ske på skærmen. Funktionen *mouseClicked()* blev kaldt hver eneste gang en ændring skulle ske på skærmen, da alle interaktioner der skulle skabe en ændring altid arr et klik. Rutediagrammet kan ses i bilag 1.

Efter der var skabt et overblik over hvornår controlleren skulle gøre noget ved hjælp af rutediagrammet, blev der lavet pseudokode. Pseudokoden havde samme princip som rutediagrammet, men denne kode blev skrives med ord over hvad spillet skull indeholde. Dette betød at i pseudokoden kunne der skrives som eksempel ”skal udregne highscore vha. mouseclicks og tid, hvis spillet er gennemført”. Dette gav et større overblik over hvordan spillet skulle kodes, da denne sætning fortalte hvad der skulle udregnes, hvornår det skulle udregnes og hvad det skulle udregnes med. Pseudokoden kan findes på bilag 2. Dette betød at kodningen ikke ville være så svær, da alt varr skrevet klar. Når disse to faser varr klaret, skulle disse to metoder kombineres og omskrives til Java sprog, for at spillet kunne spilles. Hvis der sammenlignes med rutediagram, så er forskellen at rutediagrammet kun fortæller at highscoren skal udregnes. Den fortæller ikke noget om hvordan det skal udregnes, eller hvad der skal bruges til at udregne den. Den fortæller kun hvornår den skal udregnes. Dette er grunden til at begge blev lavet, de gav begge to et overblik over forskellige dele af programmet.

Når pseudokoden skrives kan der opstå nogle problemer som gør at rutediagrammet skal omskrives. Den iterative proces blev brugt hele tiden under udviklingen af spillet. Under programmeringen opstod der nogle problemer som betød at pseudokoden skulle ændres, som måske betød at rutediagrammet skulle ændres. Et af problemerne der skulle ændres, var at alle de forskellige interaktive dele i den visuelle del af programmet var delt op i forskellige klasser, som udmundede i problemformuleringen der blev opstillet.

Når alt denne forarbejdning var lavet, kunne kodningen begynde. Nu kan problemerne løses ved hjælp af analysen og diskussionen.

4.2 Analyse

I dette afsnit analyseres de forskellige problemer ved hjælp af de definerede metoder fra metode afsnittet. Ved at analysere de forskellige problemer kan opgaveformuleringen besvares, som er formålet med denne opgave. Opgaveformuleringen er formuleret til klasser, men programmet er optimeret så der fokuseres på funktioner. Derfor er disse metoder fokuseret på funktioner, men kan også bruges til klasser.

4.2.1 Aktivere og deaktivere funktioner

For at aktivere og deaktivere de forskellige funktioner, kunne der laves *gameStates* som gjord det nemmere at skifte mellem de forskellige funktioner. Når *gameStaten* så er ændret vil der være en masse *else if statements* som *draw* går igennem for at finde ud af hvilken funktion der skal aktiveres og hvilke der skal deaktiveres. GameState er ikke en funktion der kommer fra API’en, men er en variable som selv er lavet. Efter gameState blev lavet, var det næste problem at skifte mellem if-statements i funktioner.

4.2.2 Skifte mellem if-statements

Det der menes med at skifte mellem if-statements er at der under funktionen for skabet som eksempel, skal vise et lukket skab til at starte med, hvorefter nøglen skal åbne skabet, og kurven skal kunne rykket ned. Alle disse under funktionaliteter i funktionen skab, bliver delt op med if-statements. For at den så kan skifte mellem de forskellige er der lavet en anden String som hedder skabb. Den kan have forskellige ord alt efter hvilken if-statement der er tale om. Den fungerer helt på samme måde som gameState, det eneste der er anderledes er, at gamestaten skal have et specifikt ord i Stingen. Koden kan ses nedenfor:

```
1.  //Når nøglen puttes i skabet
2.      else if(mouseX > 1200 && mouseX < 1200+80 && mouseY > 980 && mouseY < 980+80 &&
    gameState == "skab"){
3.          skabb = "hylde";
4.          gameState = "p4";
5.      }
6.      // Flyt kurv
7.      else if (mouseX <= 650+350 && mouseX >= 650 && mouseY <= 580+220 && mouseY >= 580 &&
    gameState == "skab" && skabb == "hylde") { // åbent skab med kurv på jorden
8.        skabb = "papir";
9.        if (skabb == "papir") { //Når kurven rykkes ned
10.         background(223, 45, 128);
11.         image(m8, 400, 200, 900, 900); //åbnet
12.         image(m9, 950, 740, 350, 350); //tøj
13.       }
14.     }
```

Figur 1: Denne figur er noget kode fra controlleren, og viser hvordan Stringen skabb er blevet brugt.

Som der kan ses på billedet ændre gameStaten sig alt efter hvilket skabb String der er tale om. Og ved flyt kurv kommentaren, lige under, kan det ses at to states skal være sande før den kan aktiveres. Både gameStaten skal være specifik men også skabb Stringen.

Dette kan gøre det nemmere at skifte mellem de forskellige if-statements, da viewet nu kun behøver at skrive skabb Stringens ord i if-statementen, som vist nedenfor:

```
1.  void skab(){
2.    if (skabb == "hylde") {
3.      background(223, 45, 128);
4.      image(m8, 400, 200, 900, 900); //åbnet
5.      image(m9, 650, 440, 350, 350); //tøj
6.    }
7.  }
```

Figur 2: Denne figur er noget kode fra view, og viser hvordan Stringen skabb er blevet brugt.

Det der gøres er at skabb, har et bestemt ord i sin String, som er det den starter med, hvorefter ordet i Stringen skiftes, som kan ses på figur 1. Når den skifter vil den i viewet skifte hvad den viser, da det der blev vist før ikke længere opfylder de nye parametre, som får den til at skifte til det næste billede. Da dette er meget éns til gameState vil der være større overblik, da det meste at kodningen følger samme princip.

Et tredje problem der opstod under programmering, var at gøre interaktionen nemmere mellem model og view. Da det er modellen der står for data, var det den der stod for at loade billederne ind, og viewet skulle stå for GUI'en som skulle bruge billederne. Problemet var dog at billederne skulle loade i en klasse og vises i en anden. Måden der blev skabt interaktion mellem model og view var ved at lave alle billederne til globale variabler. Dette var med til at viewet kunne kalde dem, da de blev deklareret i controlleren. Alle klasser kan kalde globale variabler fra controlleren, og gør det derfor nemt at løse problemet.

Efter alle disse problemer var løst, kunne spillet spiles og det fungerede som det skulle, men der var nogle diskussioner angående nogle af de valgte metoder. Grunden til dette var at, hvis programmet skulle optimeres ville andre metoder have været bedre. Der diskuteres derfor i det kommende afsnit angående valgte metoder, og om andre metoder ville være bedre.

4.3 Diskussion

For at kunne skifte mellem de forskellige interaktioner blev der lavet gameStates. Men der er en komando indbygget i Processing, som gør det samme som gameState, en *case*[1]. Hvis case blev brugt, ville det meste af det der står i draw ikke være nødvendigt, da de bliver tildelt funktionerne i viewet. Grunden til at det ikke er blevet brugt i dette program er dog, at det vil skabe mindre overblik. en case kan kun navngives med en integer, float eller char, som ikke gør det så nemt at se hvad der er at gøre med ud fra navnet. Hvorimod hvis stringen gameState blev brugt, ville navnet kunne gøres så detaljeret som der behøves. Et eksempel kunne være menu, hvor en gameState hedder menu, og den viser menuen. Dette er meget overskueligt, og nemt for en anden at kunne finde rundt i. Selv under programmeringsprocessen ville det være nemmere at huske hvad en gameState hedder, da den relatere til hvad der skal kaldes, hvorimod en case kun er ud fra én karakter eller et nummer. Men hvis der vægtes efter hvor meget gavn det giver i forhold til yderligere problemer, så giver den størst gavn. Dette vil sige at hvis programmet skulle optimeres ville det uden tvivl være bedst at bruge case i stedet for gameState.

En anden måde at optimere koden kunne være ved at skifte alle *else if-statements*[2] med en *switch*[3]. Denne switch søger efter en case der passer til de parametre der er opstillet på et givent tidspunkt. Et argument for hvorfor det ikke bruges er at selv hvis der tilføjes *break*[4], så ville spillet ikke nødvendigvis gå hurtigere, da den går i kronologisk rækkefølge. Den bliver ved med at læse den næste case ind til at den har fundet den case der passer til parametrene. Dette gør programmet også lige nu med alle else if-statements, da den går kronologisk indtil den finder det korrekte statement. Dette betyder at ved brug af en switch vil der ikke være særlig meget optimering og derfor er det bedst bare at beholde de mange else if-statements.[5]

5. Konklusion

For at gøre det nemmere at deaktivere klasser eller funktioner, er gameState en meget effektiv metode at bruge, da det er en String hvor ordet kan ændres alt efter hvilken funktion eller klasse der skal aktiveres. Dog ville en case være mere optimerende og er den bedste metode at anvende i denne opgave. For at kunne skifte mellem de forskellige under funktioner, bruges der også en String, som følger samme princip som en gameState. Der bruges else if-statements som skal være med til at afgøre

[1] Processing 2021, case
[2] Processing 2021, else if
[3] Processing 2021, switch
[4] Processing 2021, break
[5] W3school 2021, switch statement

hvornår en funktion skal aktiveres eller deaktiveres. For at kunne skifte mellem funktionerne skal modellen og viewet, kunne interagere hvilket gøres med globale variabler.

6. Kildeliste

Processing. (1. januar 2021). Hentet fra https://processing.org/reference/

W3school. (2021). Hentet fra https://www.w3schools.com/java/java_switch.asp

7. Bilag

Bilag 1: Rutediagram

Rutediagrammet ligger som en separat fil med navnet bilag 1. Det er et billede, som har filtypen .jpg.

Bilag 2: Pseudokode

Herunder kan hele pseudokoden ses:

Load bibliotek

Deklarere alle globalevariabler

Deklarere klasserne

Setup()

- Load model og view
- Initiere alle startværdierne
- Loadpixels

Draw()

- Hvis gameState er menu
    - Vis menu
- Hvis gameState er start
    - Vis tekst
- Hvis teksten lukkes ned
    - Vis mørkt rum
- Hvis gameState er score
    - Vis score
- Hvis gameState er exit
    - Lik spillet ned
- Hvis gameState er potteplante
    - Vis potteplante
- Hvis gameState er kontakt
    - Vis kontakt
- Hvis gameState er skab
    - Vis skab
- Hvis gameState er lås
    - Vis lås
- Hvis nøgle
    - Lig nøgle i toolbar
- Timer()

MouseClicked()

- Museklik++
- Hvis tilbageknap
    - Gå tilbage til rum hvis den er i enten skab, potteplante eller lås
    - Hvis den er i rum gå til menu
- Hvis dør
    - GameState lig med lås
- Hvis kontakt tændt
    - GameState lig med rum
- Hvis skab
    - GameState lig med skab
- Hvis nøgle
    - Skabb lig med clue teksten
- Hvis udenfor clue teksten
    - Skabb lig med åbent skab
- Hvis kurv
    - Skabb lig med skab uden kurv
- Hvis potteplante
    - GameState lig med potteplante
- Hvis koden er 312
    - Spillet er løst
- Hvis spillet er løst
    - Udregn tiden brugt
    - Udregn point ud fra museklik og tiden brugt
    - Vis slut knap
- Hvis slutknap
    - GameState lig med menu
- Hvis restart
    - GameState lig med mørkt rum

Timer()

- Hvis start
    - Vis timer
    - Tiden er 10 minutter
    - Tæl ned
    - Hvis under 2 minutter
        - Skift farve på timer
- Hvis spil færdig
    - Stop timer
- Hvis tiden er løbet ud
    - Skriv mission failed
    - Restart knap

Bilag 3: Processen

Den iterative proces er dokumenteres på Github i organisationen ved navn EscapeRoomEksamensProjekt-EscapeRoom-Kansha branch. Det kan hele udviklingen af spillet ses. Alle faserne af spiludviklingen bortset fra forarbejde kan ses der.