Danmarks
Tekniske
Universitet

Projekt i Januar

Introduction to Software Technology

Forfattere

Gruppe 25
Christopher Zwinge - s204459
Jacob Martens - s204443
Simon Peter Sundt Poulsen - s204445
Tobias Collin - s183713

22. januar 2021

Indholdsfortegnelse

Begrebsliste

- Grid : refererer til selve banen og dens grid
- Tile : refererer til et felt i banen
- Board : refererer til banen.
- MVC : Model View Controller
- Edge Case : Noget der kun opstår i ekstremum
- Embedded layout : layout i JavaFX bestående af flere sammensatte layouts
- GUI : Graphical User Interface
- Settings : Indstillinger
- 3BV : En algoritme for at finde mindst mulige klik for at løse et minesweeper spil
- array : Java Array-/ArrayList-objekt
- - navn : Hvem der har lavet hvad. Skifter først når et nyt navn kommer.
- Leaderboard : Leaderboardet er en samlet betegnelse for alle Highscores-klasserne. Leaderboard er listen over highscores, som brugeren kan se og interagere med.

Problem - Christopher

Simpelt Minesweeper

I dette projekt skal vi udvikle vores egen version af Minesweeper ved hjælp af JavaFX. Minesweeper er et spil, hvor der er et antal kvadratiske felter, hvor der under hvert felt enten er et tal, et blankt felt eller en bombe. Tallet beskriver antallet af bomber, der er i felterne omkring det felt, der er blevet vist. Spillet går ud på at vise alle felter, der ikke har en bombe. Hvis der klikkes på et felt, bliver indholdet vist, og hvis det er en bombe, har man tabt. Hvis det ikke er en bombe, kan man blive ved med at klikke indtil alle felter, der ikke har bomber, er blevet klikket på, hvorefter man har vundet.

I dette projekt udvikler gruppen to hovedversioner, en der er simpel og kun indeholder de funktioner, der er beskrevet ovenover, og en der er mere avanceret, som bliver udvidet med flere funktioner samt et funktionelt GUI til spillet.

Til den avancerede del er der taget udgangspunkt i at implementere flag, startfelt, bombetæller, stopur, hints, blankfeltsløser, highscore, reset og spilindstillinger.

Udvidelser

Spillet

Til selve spillet er der fire udvidelser. Flagudvidelsen gør at man kan højreklikke på et felt og placere et flag. Dette gør at man ikke kan trykke på feltet, som er markereret med et flag. Flagene er til at holde bedre styr på hvilke felter brugeren ikke mener, man må trykke på og sørge for at man ikke kommer til at trykke på dem ved et uheld. Derudover bruges flagene til at opdatere en bombetæller der tæller hvor mange bomber der er tilbage hvis man har sat alle flagene korrekt. Der skal dog forstås at bombetælleren går ned selvom et flag placeres forkert.

Der bliver også implementeret en blankfeltsløser. Blankfeltsløserern gør, at når der klikkes på et blankt felt, viser den automatisk alle de blanke felter og talfelterne der er direkte forbundet med de blanke felter. Dette er lavet så man skal klikke væsentligt færre gange for at løse spillet. Dette stopper også brugeren i at løbe ind i et loop, hvori personen klikker på felter uden at tænke sig om og til sidst ved et uheld rammer en bombe eller får information ved et forkert klik, som de ikke skulle kende til.
Til dette bliver der også implementeret et startfelt, som gør, at det første venstreklik i spillet aldrig kan være en bombe, samt at der er garanteret et blanktfelt. Dette første klik giver som minimum et 2x2 felt, med et blankt felt hvor musen klikkede.

GUI

Til GUI'en af spillet er der valgt fem udvidelser. Der bliver lavet et stopur som starter med at tælle op når der første gang venstreklikkes på spillebrættet. Dette bliver ved indtil spillet bliver tabt ved at trykke på en bombe eller hvis spillet bliver vundet. Der bliver også lavet spilindstillinger, hvor man kan vælge sværhedsgraden af spillet eller definere sin egen. Til dette bliver der lavet en Reset-knap som genstarter spillet og genererer et nyt spillebræt med de nuværende indstillinger.

Der er også valgt at lave en Hints-knap, som viser hvor man har nok information til at se at der er bomber. Selve hints-systemet viser ikke bomber, hvor du har sat et flag og den vil kun vise åbenlyse bombeplaceringer.
Den sidste udvidelse der er blevet valgt er et highscore-system, som kan gemme ens highscore når man vinder spillet. Den bliver også lavet til at gemme highscoren mellem åbninger af spillet.

Fravalgte udvidelser

Vi har generelt fravalgt udvidelse ud fra et tidshenssyn, og hvor meget det tilføjede til spillet. Vi har valgt ikke at undersøge forskellig geometri og en 3d udgave af Minesweeper på grund at det virkede meget kompliceret at implementere og det ikke virkede til at tilføje særlig meget til spillet, derudover siden ingen i vores gruppe havde et større kendskab til JavaFX valgte vi at fokusere på den kendte udgave af Minesweeper.
Vi valgte ikke at undersøge multiplayer i Minesweeper, fordi det kom med komplikationen af hvordan et Minesweeper multiplayer spil skulle udformes. Skulle det være et klik på samme bræt efter hinanden, skulle det være to forskellige bræt, skulle man løse sit felt så hurtigt som muligt eller kæmpe om at komme længst. Det virkede uklart hvad der var bedst og om det overhovedet tilføjede noget til spillet. På grund af dette valgte vi ikke at gå videre med den udvidelse.

I forhold til en gemme funktion overvejede vi at implementere det, men vi valgte at den i sidste ende ikke var særlig interessant siden Minesweeper er designet til at være et forholdsvist kort spil og at det derfor ikke ville være særligt ofte at en sådan funktion ville blive brugt.

Mål - Tobias

Hovedmål

Fungerende Minesweeper

Det første hovedmål er at have en fungerende version af Minesweeper uden udvidelser, hvorefter valgte udvidelser kan implementeres. Heri vil den simple version indeholde funktionaliteten til at klikke på bomber, klikke på blanke felter, se at hvert tile giver enten et blankt felt, tal mellem 1 og 8 eller en bombe.

Flag

Håndtering af flag er det første som skal blive implementeret efter Minesweeper virker. Her vil brugeren kunne sætte et flag, for at markere, hvor brugeren tror, der er bomber. Når et sted er markeret med et flag, skal det heller ikke være muligt at venstreklikke på flaget for at aktivere feltet, som ligger nedenunder. Derved sikrer man sig, at brugeren ikke trykker på feltet ved en fejl.

Tællere

For at holde styr på antallet af bomber og tiden der er gået, sættes der to tællere, én i hver side af toppen af spillet. Den i venstre side, vil indenholde antallet af bomber, som tælles ned ved brug af flag. I den højre side bliver der placeret et andet felt, som viser, hvor lang tid der er gået i sekunder, og efter 60 sekunder vil den også vise tallene i minutter.

Hints

Hints tager en version af AI, som giver et forslag til et træk, som vil være fuldstændig sikkert. Heri vil den kunne vise, hvorhenne den mener, der er bomber, uden at der er blevet placeret et flag. Den skal herefter vise, hvor de bomber er på spillefeltet.

3BV

`3BV` er en måde at måle sværhedsgrad i Minesweeper. Dette gøres ved at se på, hvor mange klik det tager optimalt at løse selve boardet. Dette giver således hver bane en unik `3BV` -score, som bruges til at måle highscoren eller sværhedsgraden af spillet.

Highscore

Highscores skal kunne gemme scorer, så de kan hentes og skrives til, når brugeren starter spillet. Når programmet kører, skal scorerne kunne vises på en passende måde.

Settings

Settings er en måde for brugeren at ændre på banens størrelse og antallet af bomber. Heri kan der vælges sværhedsgrader til spillet, som er muligt at blive husket til highscoren.

Brugervenlighed

For at gøre spillet mere brugervenligt skal der under settings være en rød tekst-prompt, således man kan se, hvis ens tal/tekst, man har skrevet, er enten for stort, lille eller forkert, i forhold til hvad programmet forventer af input. Hvis der bliver skrevet noget, der er forkert, skal der refereres til et allerede defineret tal i stedet for det forkerte, og derefter laves banen.

I leaderboards er der et tekstfelt, der viser, hvilket leaderboard man er på. Dette kommer i form af et tekstfelt, der ændrer sig, i forhold til hvilken sværhedsgrad, brugeren har aktiveret.

Når brugeren bevæger musen henover et felt i spillet, highlightes det. Herved kan man se, hvilket felt man er på, så man ikke klikker forkert, i forhold til hvilket `tile` man er på. Dette giver også en ekstra visuel indikator på hvorhenne musen er, idet man nogle gange kan miste overblik over musen.

Når der trykkes på Hints-knappen, skal tiles der med sikkerhed er bomber på highlightes. For at fjerne highlightet skal musemarkøren bevæge sig henover det markerede tile. Hints-funktionen skal ikke markere felter med flag.

En sidste del, for at gøre spillet mere brugervenligt, ligger i, at når der første gang klikkes på boardet, sørges der for, at brugeren ikke klikker på en bombe.

Potentielle udvidelser

AI

En implementation der giver muligheden for en AI der kan løse minesweeper. Der kan ikke forventes en fuldstændig sikker AI, idet der kan være lidt gætteri.

Boardet

Selve programmet skal laves til, ved lidt ændringer, at kunne håndtere flere bomber og/eller flag på det samme tile.

Læringsmål

Gantt skema

Til dette projekt lærte gruppen at lave et Gantt-skema. Gantt-skemaet blev brugt til at holde styr på, hvornår forskellige dele af projektet skulle være færdige.

JavaFX

De fleste af udfordringerne ligger i at skrive koden til spillet, da gruppen ikke har meget erfaring i at bruge JavaFX og samarbejde om kode til et program. Gruppen har ikke kendskab til alle funktioner/objekttyper i JavaFX-pakken, og derfor kan der være brug for at lave en del research inden selve koden skrives. Derudover skal gruppen sætte sig ind i syntaksen til JavaFX.

Succeskriterier og ambitioner

Projektet ses som en succes i det tilfælde, at spillet og alle dets udvidelser virker, samt at rapporten opfylder de mål, der er givet i projektbeskrivelsen.

Idet selve minesweeper skulle laves fra bunden ved at lave vinduet, datastrukturen og tilesystemet indenfor tre uger, er der visse ting, man skal regne med ikke bliver færdigt. Derfor, ud fra alle de ting, man kan lave, regnes der med, at de fleste udvidelser kan nås, men ikke dem alle sammen. Hvis der er tid til flere udvidelser, vil der implementeres flere dele. Gruppen håber på tocifret karakter.

Analyse

Board og Tile - Jacob og Tobias

Tobias -Boardet består af en mængde tiles og har det formål at håndtere tile-objekternes funktioner. Her består problemet i at få alt dataet til at stå rigtigt og kunne blive hentet, således at man kan finde ud af, om der er en bombe, et flag eller et tal på et felt. Derudover skal tile-objekterne håndtere, hvad der sker, når musen interagerer med et bestemt felt.
Jacob - Boardet skal kunne håndtere felterne og få dem til at spille sammen. Der ligger derfor en udfordring i at læse data fra og holde styr på status af felterne omkring et bestemt felt. Boardet skal altså kunne opdatere flere felter, selvom der kun trykkes på et enkelt felt.

Felterne på boardet skal kunne håndteres individuelt, hvilket vil sige, at de enten kan blive oprettet som objekter med de samme egenskaber som felterne i det originale Minesweeper, eller hele boardet kan oprettes som et visuelt grid, hvor hvert felt håndteres funktionelt afhængigt af musens position (i koordinater) på boardet. I dette projekt vil der gøres brug af enkelte objekter, da det gør det nemmere at implementere diverse udvidelser. Det leder dog til problemer i forhold til:

- Hvordan tile-objekterne skal oprettes og gemmes, så det bliver nemt at hente og skrive data til et bestemt felt.
- Hvordan felterne stilles op i et grid, så der holdes orden i rækkefølgen, eftersom felterne også skal kunne interagere med nabofelterne.
- Hvordan musen interagerer med felterne.
- Hvordan hvert felt håndterer flag, bomber og værdier/tal.

Gruppen har tidligere erfaring med at håndtere et grid af objekter, så hvert element nemt kan tilgås. Derfor gemmes objekterne i et `array`,så det individuelle tile-objekt til enhver tid kan hentes og anvendes. I dette projekt bliver objekterne gemt i et todimensionelt tile-array, så hvert felt kan blive lagret afhængigt af den række og kolonne, feltet ligger i på boardet. Dette giver også en god mulighed for at ændre størrelsen på boardet, og det løser problemet med at stille felterne op i et grid, så hvert felt kan interagere med nabofelterne, da nabofelterne også vil være naboobjekterne i array'et.

I forhold til at få musen til at interagere med de enkelte felter, kunne JavaFX's Button-objekter bruges, da de kan registrere, når musen bevæger sig henover dem samt registrere venstreklik. Denne objekt-type er derfor brugbar i den simple version af Minesweeper. Button-objekterne har dog nogle begrænsningerne i forhold til udseende og funktionalitet, som kan skabe problemer i det avancerede spil, hvor der skal være plads til udvidelser. Dette kan løses ved at skrive en klasse, der opretter Tile-objekter med brugerdefinerede egenskaber og udseende. Dertil kan JavaFX's Rectangle-shape bruges, da den giver mulighed for et firkantet layout til felterne, og den kan registrere events fra musen, samtidig med at den nemmere kan tilpasses både visuelt og funktionelt.

Felterne på boardet skal kunne vise et flag, en bombe, et tal eller være blankt, når der klikkes på feltet (med højreklik eller venstreklik). Dette kan løses ved at indlæse et billede eller noget tekst på Tile-objekterne, som giver et visuelt billede af feltets stadie, samt have en eller flere variable til at gemme status på feltet, som kan anvendes til funktionaliteten af feltet.

Settings - Simon

Settings har til formål at ændre sværhedsgraden af spillet og initialisere Boardet til den nye sværhedsgrad. Hovedudfordringen består i, hvordan man skal vælge sværhedsgrad, og hvordan boardet skal initialiseres. Der er tre officielle sværhedsgrader: Beginner, Medium og Expert. Disse skal initialisere boardets højde, bredde og antal bomber afhængigt af sværhedsgraden. Stigende sværhedsgrad betyder, at `3BV`-scoren stiger. Det er klart, at som banen bliver større bliver `3BV`-scoren lavere og når antallet af bomber stiger, bliver `3BV`-scoren ligeledes større og banen sværere.

Der skal også implementeres en brugerdefineret-sværhedsgrad, hvor brugeren selv kan få lov til at vælge board-højde, -bredde og antal bomber dog med begrænsninger: F.eks. skal man ikke have lov til at lave et board, der har højde, bredde eller antal bomber mindre end nul (man bør også sætte en øvre grænse, så spillet ikke fylder mere end skærmstørrelsen).

For at sikre mod input-fejl, kan man lave passende if-else-statements, der tester for tilladte inputs, og for at sikre at man kun kan vælge én sværhedsgrad, anvendes der radio buttons, hvor det kun er tilladt, at ændre én ad gangen. Man kunne også have brugt normale buttons, men det smarte ved radio buttons er, at man kan se hvilken knap, der er valgt, i modsætning til buttons, der ikke har noget visuel fremvisning.

Highscores - Simon

Hovedudfordringen med at implementere highscores er, hvordan dataene skal gemmes. En måde at gemme data på er ved at skrive til en data-fil. Det kunne f.eks. være en tekstfil med en bestemt struktur, der egner sig til at gemme de ønskede data.
Der er to ting, som kan være relevant at gemme: den opnåede highscore og nogle karakterer, der viser hvilken spiller, der opnåede scoren; f.eks. et navn eller initialer. Det giver derfor meget god mening at bruge en tekstfil, da man kan nøjes med bare at gemme dataene i et `String array` ved at indlæse fra filen vha. et Scanner-objekt, og derefter skrive til filen vha. et PrintStream objekt.

Først overvejede gruppen, at holde highscorerne i et heltals-array. Men eftersom navn, initialer og highscores er forbundne, giver det bedre mening at bruge et string-array, der kan holde dataene samlet.

Leaderboards - Simon

Et leaderboard skal kunne vise en liste med highscores i rækkefølge fra bedste score til laveste score. Vi har valgt at implementere scorer, så lave værdier er bedre end høje værdier. Dette blev valgt, fordi vi ønsker, at scoreren kan variere meget, og de parametre, der varierer mest fra spil til spil, er tidsparameteren og antal klik. Der bliver gået mere i detaljer med implementationen i afsnittet "Implementation", men pointen er, at scoren stiger med tiden og antal klik udover 3BV-scoren.

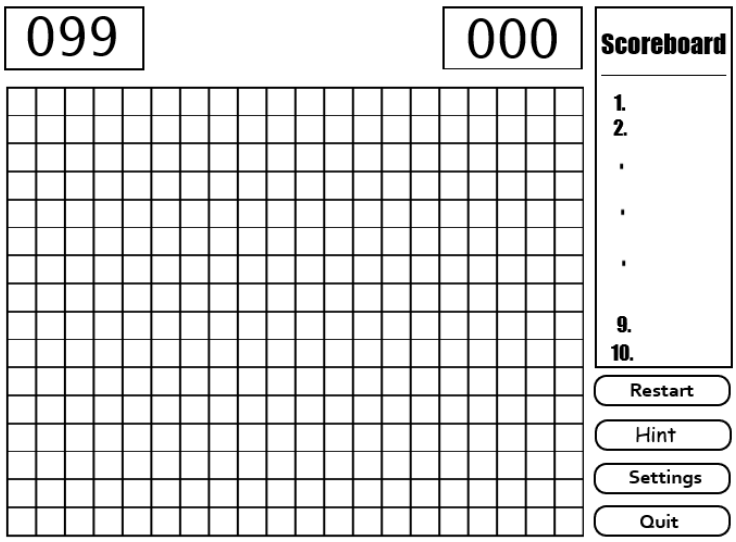
Der skal altså som minimum laves to metoder: En, der sorterer arrayet, så den ønskede rækkefølge opnås, og en der loader informationen ind i en liste i JavaFX.

Det leder os videre til det sidste problem med leaderboards; at man skal kunne vælge, hvilken sværhedsgrads tilhørende highscores, man vil se. Det løses med buttons og label tekst, der viser, hvilken sværhedsgrad, der er valgt.

GUI - Christopher

Til selve spillet skal spilfeltet vises derudover skal der vises en knap til hints, en knap til reset, en måde at åbne settings og leaderboards på. Derudover en tællere til antallet af bomber og den tid der er gået siden spillet startede.

Til at lave et pænt, men forholdsvist nemt layout i JavaFX kan der med fordel bruges `BorderPane`som opdeler skærmen i en top, en bund en midte og de to sider. På denne måde kan man hurtigt opdele elementerne i spillet og gøre at de holder en relativ afstand til hinanden uanset om vinduet ændrer størrelse. Der findes også andre layouts som man kan bruge som hovedlayout, såsom `VBox`eller `HBox`, men `BorderPane` virker som den nemmeste til at lave et design der kan bruges til mange forskellige elementer. Derfor er der blevet lave en model af hvordan et GUI kan se ud for spillet som vi til at starte med vi gå efter at lave.



Figur 1: Første model af hvordan spillet kan se ud

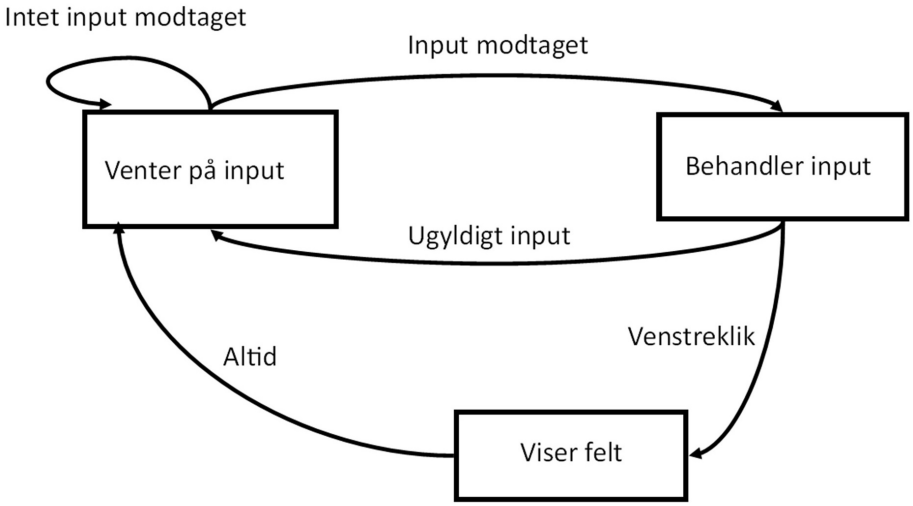
Til at lave knapperne er det oplagt at bruge `Button`objekter siden det ikke skal have mere funktionalitet end at køre noget kode når man trykker på dem. Til settings og leaderboard kan man enten som i modellen integrere det i hovedvinduet eller man kan åbne det i et nyt vindue ved at trykke på en knap eller man kan lave en `Menu`og sætte knapperne ind i for at mindske antallet af knapper på hovedvinduet. Til at starte med vil vi fokusere på at lave settings som en eksternt vindue siden der ikke er nogen grund til at se mulighederne hele tiden. Leaderboardet kan både blive en del af vinduet eller blive eksternt afhængig af hvad der virker bedst når det skal implementeres.

Design

Selve designet består af tre hoveddele, Model, View og Controller. Der laves tre pakker; én til hver del. Derudover kommer der til en pakke til main filen der starter programmet.

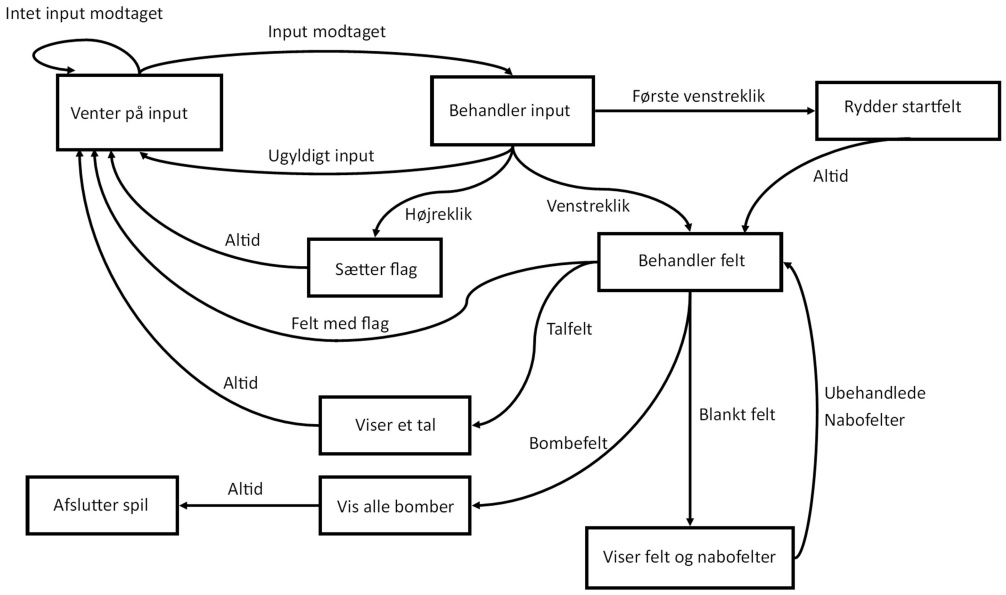
Generel programstruktur - Jacob og Tobias

Jacob billede, Tobias text



Figur 2: Tilstandsmaskine for simpelt Minesweeper

For at strukturere funktionaliteten af spillet er der modelleret en tilstandsmaskine (FSM), som viser de stadier spillet går igennem afhængigt af brugerens input. Gruppen har lavet en FSM for både den simple og avancerede version af selve spillet/boardet (main game loop). I den simple version læser programmet kun, når brugeren venstreklikker på et felt, og det er kun det felt, der bliver vist.



Figur 3: Tilstandsmaskine for advanceret Minesweeper - Jaco

Til forskel har den avancerede version nogle nye funktionaliteter. Spillet skal håndtere både højre- og venstreklik samt lave en anden handling for det første venstreklik, hvorfor der er større behov for at behandle brugerinputs. Derudover skal der vises mere end et felt, når der klikkes på et blankt felt. Spillet skal stoppes, hvis der klikkes på en bombe, højreklik skal sætte flag, og venstreklik på et flag skal ikke gøre noget. Heraf kommer behovet for at behandle felterne.

MVC - Tobias

Model

Model håndterer det der sker bagved; heri vil man se datakontrol. Model er altså uafhængig af GUI og står kun for at behandle programmets logik og data.

View

View-pakken håndterer alt det GUI-relaterede; heri håndteres visning af vinduet, alle knapperne der er tilgængelige og opdatering af banen. For eksempel visning af boardet. View-delen skal **ikke** håndtere, hvordan boardet skal genereres, men kun vise det board der genereres ved interaktion mellem Controller og Model.

Controller

Controller pakken tager sig af alt, som brugeren kan interagere med. F.eks. knapper og deres funktioner samt højre- og venstreklikfuntionalitet til boardet med tiles. Hvori hvis knapperne kalder på bestemte metoder til at ændre data eller vise noget, vil det håndteres respektivt

af View-delen eller Model-delen.

Grunden til at MVC (Model View Controller) bruges, skyldes at den giver en form for overblik over ens opgave. Idet at man kan regne med at det visuelle står for sig selv, og det interne databehandling ligger for sig selv. Derudover giver det en form for mutabilitet til koden, idet at selve de forskellige dele er et kald af en funktion og ikke en fuldkommen integration i hinanden. Hvis noget skal ændres, kan det altså gøres hurtigt og let i forhold til at holde det altsammen sammen i en stort fil.

Tech/platforms - Tobias

Discord

Discord er et gratis voice over IP-applikation system som tilader en gruppe af folk at have en samtale sammen, hvori det er muligt at kunne vise en skærm, med lyd eller uden. Det er også muligt at opsætte en Discord-server, som tillader at have adskillige tekstkanaler, hvori billeder, filer og videoer også kan blive lagt op, voice kanaler, hvori man kan dele sin skærm, videoopkald osv.
Alle disse muligheder sammen med en god Voice-Codec til lyden gør at gruppen har valgt Discord frem for dens konkurrenter til kommunikation.

GitHub Desktop

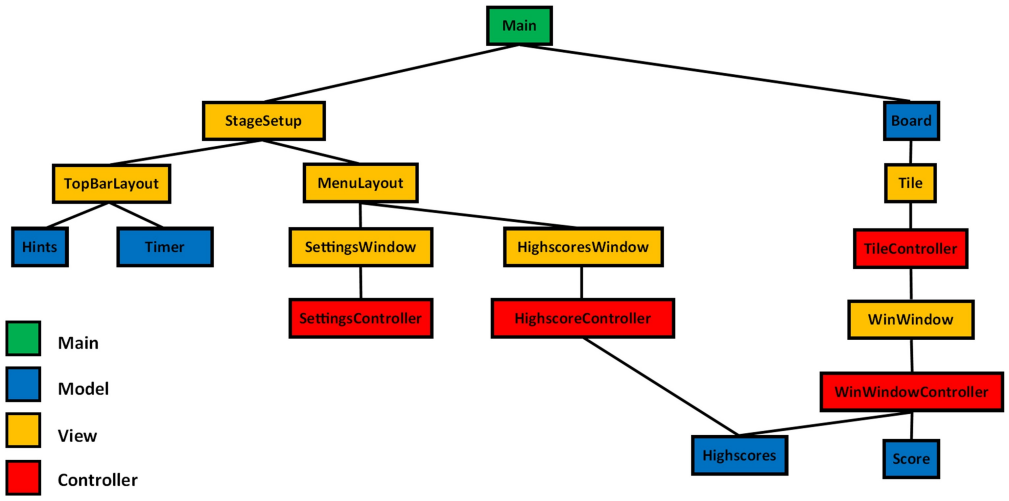
For at samle og holde styr på koden bruges GitHub Desktop, som er en Windows-app, der håndtere det samme som en normal git-account giver. Heri tilader det at pushe, til et drev i skyen som man så kan pull ned til hinanden, dette gør at flere kan arbejde med koden og når noget nyt er klart kan alle få den nyeste kode let. Det giver ikke automatisk kode streaming som gør at det man pusher er valgt med vilje. Hvis noget går galt kan man altid gå tilbage til en tidligere version. Selve programmet virker rigtig godt med Ecllipse, idet at når noget nyt pulls ned opdateres det automatisk i Eclipse.

Overleaf

Til at skrive rapporten i har gruppen valgt at beskrive programmet i Overleaf. Idet Overleaf virker ved at alle for gruppen kan skrive i det samme dokument med latex fordelene. Dette tillader gruppen at aktivt se dokumentet og den tilstand der er nået til. Dette tillader folk at hjælpe hinanden lettere end hvis man skrev i hvert sit latex dokument.

Implementation

Vores projekt er bygget op af nogle hovedkomponenter, som udgør størstedelen af programmet, og som er lavet for at skabe struktur og give overblik i koden. Billedet nedenfor viser, hvordan programmets klasser er hierarkisk bygget op, så hovedkomponenterne udgør de øverste felter, som håndterer resten af komponenterne.



Figur 4: Klassehierarki og MVC-fordeling

Board - Jacob

Board-klassen er implementeret sent i projektet og er lavet for at give mere struktur i Tile- og Main-klassen. De fleste af Board-klassens felter og metoder stod før i Tile- og Main-klassen, hvilket gav en ulogisk struktur i programmet. Board-klassen er derfor oprettet til de felter og metoder, som har en specifik funktionel betydning for selve Minesweeper-boardet. Dette indebærer følgende metoder:
`initTiles og initBombs` initialiserer boardet med et bestemt antal Tile-objekter fra Tile-klassen og sætter X bomber på tilfældigt valgte felter. Metoden `initBombs` er defineret to gange; den ene metode tager ingen argumenter og sætter et antal bomber bestemt ved feltet noOfBombs, mens den anden metode tager et heltal (int) og et HashSet som argumenter, fordi denne metode er lavet til at sætte et givent antal bomber på nye Tiles, som ikke står i det givne HashSet af Tile-objekter (bruges ved første klik). `initBombs` placerer bomberne tilfældigt, men for at forhindre programmet i at forsøge at sætte en bombe på et felt, der allerede indeholder en bombe, bruger metoden et array af ledige Tile-objekter til at sætte bomberne hurtigt.

`getNeighbors og getGroup` returnerer en liste (HashSet) med hhv. nabofelterne til et givent felt eller gruppen af tomme felter + omkransende tal-felter, som et givent felt hører til.

`disableAll` sørger for at alle Tile-objekterne ikke længere reagerer på inputs fra musen. Metoden bruges når spillet tabes eller vindes.

`firstClick` sørger for at første klik i spillet viser det felt der klikkes på + det felts nabofelter. Det vil sige, at der altid bliver vist mindst fire ledige felter på boardet.

`reset` genstarter spillet.

`HashSets`bliver brugt flere steder i klassen, idet HashSets er hurtigere end f.eks. ArrayLists, når der skal hentes, tjekkes for eller tilføjes et objekt. HashSets bliver også brugt, fordi der ikke var brug for at gemme det samme objekt flere gange eller holde orden i objekterne (indeksere objekterne).

Tile - Jacob

Tile-klassen blev implementeret for at give et alternativ til JavaFX Button-objekterne, der blev brugt som Tiles i det simple Minesweeper program. Tile-klassen bruges til at oprette Tile-objekter, som lettere kunne blive modificeret til et passende format, der fungerer som quasi-Button-objekter. Derfor nedarver klassen fra JavaFX Rectangle-klassen, da Tile-objekterne bruger en kvadratisk form og op til flere metoder/funktioner fra den klasse.

Tile-klassen er gradvist blevet simplificeret siden den først blev oprettet, eftersom mange af metoderne og nogle globale felter kunne flyttes til Board-klassen. Ændringerne i Tile-klassen er lavet for at holde klassen mere overskuelig og give bedre struktur i forhold til MVC-princippet. Tile-klassen er gået fra at håndtere det meste af programmets Model-del, samt stå for Controller- og View-delen af Tile-objekterne, til kun at stå for en lille del af Model- og hele View-delen. Controller-delen af Tile-objekterne er deraf blevet flyttet til en separat klasse, TileController, som håndterer brugerinputs i form af museklik og -bevægelser.

Tile-klassen er lavet med en konstruktør, som initialiserer Tile-objektet med den rigtige størrelse, tekst og effekter, samt sørger for at objektet bliver placeret korrekt på boardet. Tile-klassen indeholder desuden nogle metoder, som har en specifik indflydelse på Tile-objekterne, herunder:

`revealTile`, som sørger for at vise værdien på et felt, når der klikkes på det (tal, bombe eller blankt). Metoden kalder sig selv rekursivt, hvis der klikkes på et blankt felt, da alle nabofelterne også skal vises.

`getValue`, som returnerer talværdien af et felt. Altså antallet af bomber på de otte nabofelter.

`setHighlight`, som sætter en highlight-effekt på Tile-objektet med en given farve.

`handle`, som er implementeret fra JavaFX's EventHandler interface. Metoden kalder `onClick`-

metoden fra TileController-klassen, som håndterer events fra musen, herunder højre- og venstreklik samt bevægelse ind på og væk fra et Tile.

Topbar - Jacob og Christopher

Christopher -Topbaren består af flere klasser, som tilsammen står for layout og funktionalitet af selve topbaren og de fire elementer, som topbaren består af: Bombetælleren, Hints-knappen, Reset-knappen og Timeren.

TopBarLayout-klassen håndterer udseendet af topbaren med metoden getTopBar, som sætter de fire topbar-elementer op med korrekt indrykning og mellemrum mellem hinanden, samt initialiserer elementerne med hver deres startværdi. Reset-knappen refererer direkte til reset-metoden i Board-klassen.

Jacob - Hints-klassen er implementeret som en ekstra udvidelse til Minesweeper-spillet, hvor brugeren kan få hints til, hvor der ligger bomber. Hints-klassen består af disse metoder:

`getHint` highlighter felter, hvor der kan sættes flag ved brug af en simplificering af Straightforward-algoritmen fra en WordPress-side omhandlende Mineswepper AI.[1] getHint-metoden bliver kaldt fra knappen i topbaren.
`getUnclickedNeighbors` finder nabofelterne til et givent felt, som ikke er blevet klikket på og gemmer dem i et HashSet.

Jacob - Timer-klassen indeholder funktioner til at modificere timeren. Timer-klassen består af en konstruktør, som opretter en JavaFX Timeline, der tæller forløben tid i reeltid. Timer-klassen er desuden implementeret med metoder til at pause/starte timeren, genstarte timeren og omdanne tiden fra sekunder til minutter hvert 60. sekund. Når timeren når til et defineret maksimum værdi pauser den for at undgå at den fylder for meget i GUI'et og for at der ikke kan kommer en overflow fejl af at den bliver ved med at køre.

Score - Simon og Tobias

Tobias - `get3BV`- metoden returnerer det minimum, som der skal til for at løse banen. Her bruges et Hashset, som tæller antallet af unikke tile-grupper. For hver gruppe sættes der et tjek. Derfra tæller den grupper fra øverste venstre hjørne til nederste højre hjørne. Når alle tiles er talt retunere den en integer med antallet af grupper den har fundet.

Simon - `getScore`-metoden beregner scoren vha. tre parametre: 3BV, timesClicked og tiden. Gruppen har valgt at beregne scoren i to trin: Først beregnes multiplier $= \frac{\text{timesClicked}}{\text{3BV}} \cdot 10$. Dvs. at jo mere man klikker ud over 3BV'en, jo større bliver brøken. I andet trin bestemmes scoren som score $=$ time $\cdot$ multiplier. Dette betyder, at hvis man har klaret spillet optimalt,

[1] https://luckytoilet.wordpress.com/2012/12/23/2125/

vil tiden blive ganget med 10. Derfor vil det, med denne formel, være bedre at få en lav score end en høj score.

Highscores - Simon

Highscores-klassen håndterer alt, hvad der har med highscores datafiler at gøre. Klassen har tre `public` felter af typen File: datafilerne for hver af de i settings definerede sværhedsgrader. De skal være public, så andre dele af programmet kan passe dem til metoderne i **Highscores**-klassen.
Klassen har en metode `createFile`, der sørger for at filerne bliver oprettet, hvis de ikke eksisterer i forvejen. Dette er **nødvendigt** for at man kan læse fra og skrive til filerne, efter programmet er konverteret til en jar-fil.

Hoveddelen af **Highscores**-klassen består af to metoder: `read` og `write`.

- `read` håndterer indlæsning af data fra en datafil til et `String array`. Dette gøres vha. en Scanner og et for-loop, der initialiserer hvert element i arrayet.
- `write` håndterer det at skrive til datafilerne efter nye highscores er opnået. Metoden starter med at læse data ind i String arrayet. Dernæst går den igennem alle elementerne i arrayet indtil den finder en default highscore: -1. Hvis den finder sådan en bliver den første af dem erstattet. Ellers gennemgås arrayet igen, og der tjekkes for om brugeren har opnået en ny highscore. Hvis brugeren har opnået en ny highscore erstattes den dårligste score med den nye. Dernæst sorteres arrayet med `sort` metoden, og der skrives til filen.

`sort` metoden håndterer sortering af alle elementer i String arrayet. Metoden har en boolean, der holder styr på, om alle elementerne i arrayet er sorteret. Metoden kalder sig selv rekursivt indtil en integer i = 0. i sættes til at starte med til det højeste index af arrayet, og så arbejdes der ned derfra. Det sikrer, at metoden vil blive kaldt indtil alle elementer er ombyttet. Metoden ombytter så -1 kommer til at ligge sidst i arrayet og scorer kommer til at ligge så lave værdier har lave indices og høje værdier høje indices. Dette ændre selve fremvisningen ved at det array med den bedste score kommer til at stå øverest.

Highscores-klassen har yderligere tre metoder: `isNewHighscore`, `reset` og `getHighscores`.

- `isNewHighscore` tjekker for en datafil, om brugeren har opnået en ny highscore enten ved at der i datafilen er en default highscore -1, eller ved at brugeren har fået en bedre score end én, der er der i forvejen. Bruges i samspil med **winWindow**.
- `reset` sætter alle elementerne i en datafil til en default state (-1:***). Bruges når filerne oprettes.
- `getHighscores` returnerer en klon af highscores arrayet. Bruges i samspil med **HighscoresWindow** til at initialisere listView.

Eksterne vinduer - Christopher og Simon

Christopher - SettingsWindow-klassen håndterer Settings-menuen. Klassen indeholder en enkelt metode, `display`, som står for opsætning af Settings-vinduet samt funktionaliteten af knapperne. Settings-vinduet indeholder fem forskellige knapper hvor de tre af dem bliver brugt til at vælge sværhedsgrad og den femte bliver brugt til sende ændringen videre til klassen og laver et nyt spillefelt. Derudover er der text bokse som viser hvad højden, bredden og antallet af bomber er samt tekstfelter til custom sværhedsgradhed. Settings-vinduet tjekker hvilken sværhedsgrad spillet var sat til før det åbner og sætter den tilsvarende knap til at være valgt. Derudover hvis custom sværhedsgraden var valgt før indstillingsvinduet blev åbnet så henter den den højde, bredde og antal bomber der var valgt sidst.Når det nye spillefelt bliver lavet sørger denne klasse også for at størrelsen af vinduet bliver ændret til at passe til det nye spillefelt og at spillet bliver centreret på skærmen.

Christopher - winWindow-klassen håndterer win-vinduet som vises når et spil bliver gennemført. Den indholder metoden `display`, som laver opsætningen af vinduet. win-vinduet indhenter den score der er blevet opnået i spillet, tiden og den sværhedsgrad spillet er sat til. Derudover tjekker den ved hjælp af **Highscores**-klassen om det er en highscore. Hvis det er en ny highscore så viser den to tekstbokse et tekstfelt og en knap. Den første tekstboks informerer spilleren om at det er en ny highscore og den anden står lige over tekstfeltet og beder om spillerens initialer. Til tekstfeltet er der brugt en textFormatter til at sørge for at der kun kan stå 3 bogstaver og at den sletter det første bogstav hver gang der bliver skrevet mere. Koden til tekstFormatteren er hovedsageligt kopieret fra en bruger som blev kaldt kleopatra's kommentar på et Stackoverflow spørgsmål[2] . Den er taget derfra fordi vi først implementerede winWindow sent og fordi textFormatter havde en noget anderledes syntaks end vi havde brugt før, derfor kunne vi ikke væsentligt ændre den. Knappen bliver brugt til at skrive highscoren samt initialerne videre til **Highscores**-klassen

Simon - HighscoresWindow-klassen håndterer view-delen af highscores. Den indeholder et embedded layout bestående af tre underlayouts:

- En VBox headerLayout, der håndterer top layoutet. Den indeholder et Label, der viser, hvilken sværhedsgrad, der er valgt og et Label, der angiver placeringen af Player og Highscore.
- Et ListView listView, der håndterer midterlayoutet. Et ListView er ikke i sig selv et layout, men kan tilføjes til et primaryLayout. Det skal kunne fyldes med highscores.
- En HBox buttonLayout, der håndterer bund layoutet. Den indeholder tre buttons, der, når man klikker på dem, kalder `listViewFill`metoden.

Klassens primære layout (primaryLayout) er en VBox, bestående af headerLayout øverst, listView i midten og buttonLayout nederst.
Klassen har én metode, `listViewFill`, der læser fra en af datafilerne og tilføjer hvert element

[2]https://stackoverflow.com/questions/15159988/javafx-2-2-textfield-maxlength

(hver String bestående af highscore og initialer). Hvis en String er default (-1:***) tilføjes en bindestreg i stedet for highscore og initialer til listView. Hvis datafilen indeholder reelle highscores tilføjes disse, så de bedste scorer tilføjes først og ligger øverst og de dårligste nederst (som beskrevet i afsnittet om **Highscores**-klassen). **HighscoresWindow**-klassen har ikke selv data, men henter dataene fra **Highscores**-klassen, og dermed brydes MVC ikke.
Der blev lavet en **highscoresController**-klasse, der indeholder en metode `controller`, der håndterer button kliks og opdatering af listView.

Test - Tobias

Types of test

Fundne fejl - Tobias

Hvis en fejl blev fundet i programmet og det ikke var noget ligefrem, skulle det prøves at fikses med det samme. Dette skyldes at få fjernet fejl så hurtigt som muligt vil stoppe os i at bruge den del med fejl i. Idet at dette ville skabe "technical debt"altså en form for lån i form af en fejl, som der således bliver bygget oven på indtil den endelig bliver fundet og derefter skal fjernes ved at alt skal omskrives.

Fremprovokering af fejl - Tobias

Til at fremprovokere fejl benyttes edge cases, altså tilfælde som ligger på grænsen af hvad programmet skal kunne. En edge case kan være at se om størrelsen af grid'et der laves svarer til mængden. Derefter se om programmet kan håndtere det, ved at finde disse restriktioner og derefter fikse dem vil man kunne opnå et mere robust spil.

Dokumentation af fejl - Tobias

Tiles - Tobias

Den første store fejl der blev fundet var når reset blev brugt ville den stadig kunne vise nogle af tallene fra sidste spil, heri viste det sig at reset funktionaliteten ikke genlavede `tiles`og information fra sidste spil stadig lå gemt. Dette krævede nogle ændringer i både `tiles`klassen og i metoden til at reset spillet

Board - Tobias

En anden fejl som kom i reset var at når man skiftede bane størrelse, vil den fremvise den rigtige bane størrelse, men stadig huske en større bane som derefter blev populeret med bomber, som gjorde at der var bomber man ikke kunne se på banen som taltet med. Dette blev fikset i `board`klassen.
Når bomberne var sat nær det potentielle maksimum som banen kunne håndtere var der et

problem med at placere bomberne som fremkom. For at fikse dette problem blev der valgt at blive brugt et hashset som gemmer de unikke tal som er gået igennem random funktionen. Hertil vil det stoppe den i at vælge det samme tal igen og derved stopper dette problem i at fremkomme igen og hastigheden på at placer bomber i ekstreme tilfælde tager ikke potentielt uendeligt tid.

Når spillet først genereret tal til bomberne blev de respektive tal naboer ikke placeret ordenligt. Dette fremkom af at metoden for at kigge på boarded ikke var den samme metode som tallene blev placeret hvilket gjorde at det blev byttet om på x aksen og yaksen. Når dette blev indset var det omskrevet.

Timer - Tobias

I timer klassen var der et problem at der blev lavet en ny timeline hver gang man reset banen som ikke var meningen. Derfor i opsætningen skulle der laves en ny timeline og når reset blev kaldt reset man tiden i ens timeline, men ikke lavede en helt ny.

Jar - Tobias

En anden fejl lå i når der laves jar filerne. Det store problem var her at jar filerne ikke gemmer op den tekst fil som blev brugt. Dette fik den betydning er der således skulle designes en funktion som tjekker om filerne eksistere og hvis de ikke gør, lave dem i det samme folder som spillet ligger i og derefter bruge dem.

Highscore - Christopher

Til sortering af highscores kom der problemerne i form af at vi prøvede at bruge kommandoen `Array.sort()`.Vi troede at den ville sortere efter tallene i highscore tekstdokumenterne, men fandt senere ud af at når den sortere `String`objekter sorterer den efter deres unix kode. Derfor blev vi nødt at lave vores egen sorterings metode som også endte med at løse nogle andre problemer som `Array.sort()`lavede.

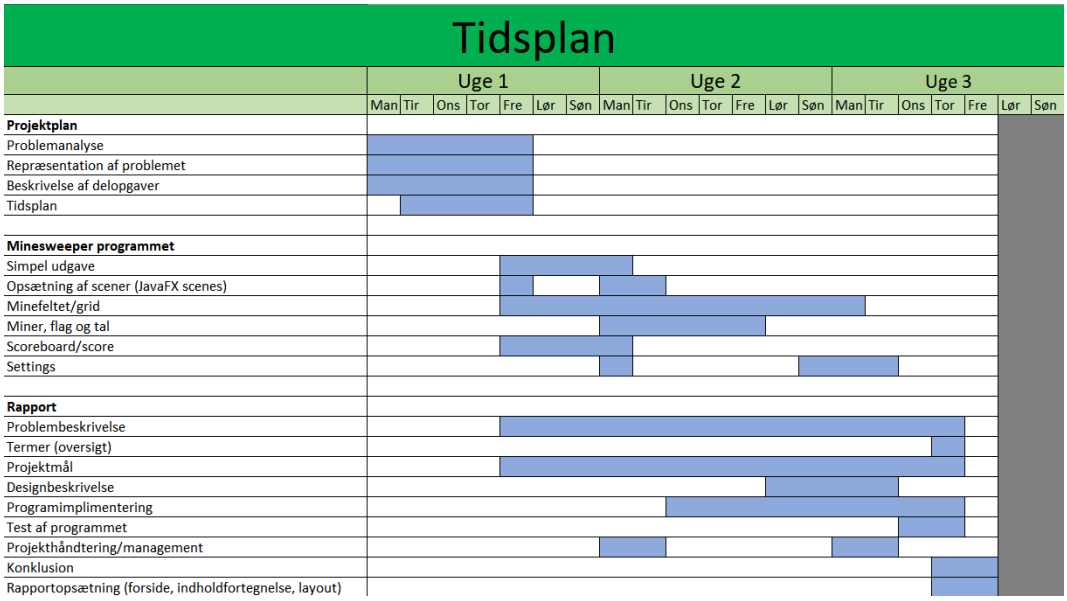
HighscoresWindow - Simon

Undervejs i implementationen af **HighscoresWindow** blev det besluttet, at brugeren ikke må indtaste mere end 3 karakterer som sine initialer af layoutmæssige hensyn, så header-Layout og listView ville line op.

Projekt håndtering - Alle

Gantt skema

Til at håndtere tid og sørge for at vi ikke kommer bagud bruges et Gantt-skema, der indeholder en oversigt over, hvad der forventes at være klar på hvilket tidspunkt. På denne måde kan man se hvor langt man er nået i forhold til hvad der mangler. Hvis man er lidt forud for skemaet kan det tages med lidt mere ro end hvis man var bagud.



Figur 5: Gantt skema

Som beskrevet har der været et Gantt-diagram, som alle blev enige om før gruppen startede på projektet. Ved at vide hvad der var forventet, besluttede vi os dag for dag om morgen, hvad der skulle gøres, og hvad gruppen forventede.

Opgavefordeling

Ved begyndelsen af projektet blev arbejdsopgaverne fordelt, så hvert gruppemedlem havde en del, de havde ansvaret for. Efterhånden som alle var ved at være færdige med deres dele, blev resten fordelt efter behov. Til sidst i processen var der en del mere samarbejde om alle løsningerne og samspil mellem programdele, for at få det hele til at virke sammen. Rapporten blev også gennemgået af alle, for at sikre at alle var enige om, det der var skrevet.

Konklusion - Alle

Hvad var opnået

Hovedspillet blev lavet rimelig hurtigt og uden problemer, som nok kom fra en god planlægning af, hvordan de forskellige dele skulle implementeres.

Hertil fik vi designet flag til selve boardet og til `tiles`. Ved at opsætte banen ved brug af tiles, havde vi mere kontrol over hvad der kunne være i dem, end hvis vi havde brugt knapper til at udgøre funktionaliteten. Implementering af Tile-objekter i stedet for JavaFX Button-objekter krævede en stor udvidelse af Tile-klassen.

Mht. implementeringen af timer viste det sig at der hovedsagligt var to måder at håndtere det på. Heri blev der valgt JavaFX Timeline, som således kunne blive sammensat med et Integer til at fremvise tiden i sekunder og minuter.

Leaderboards blev implementeret, så der kunne læses fra og skrives til filer. Der blev oprettet tre filer: én for hver sværhedsgrad. Der opstod problemer med at læse og skrive til filer, da programmet blev konverteret til jar-fil, men det blev løst ved at generere filerne ved start-up, hvis de ikke var der i forvejen.

Settings viste sig at være forholdsvist ligetil. Designet tog dog en del tid at implementere. Det hjalp dog på at lære hvordan man brugte JavaFX's layout objekter og EventHandler.

Der blev implementeret både startfelt og blankfeltsløser, som krævede en lille omskrivning af håndtering af board og tiles.

Bombetæller var relativt simpel, idet at der blev brugt systemer der allerede var placeret med lidt ekstra kode oveni til fremvisning og opsætning af tælleren.

Reset-knappen krævede en hurtig implementering af en knap i Topbaren, som kun skulle kalde en enkelt metode, der kunne nulstille spillet.

Hint funktionen tog brug af hvad der blev designet til AI som således blev omskrevet til fremvise hvori der var bomber uden at vide mere end spilleren.

Hvad var ikke opnået

Planen var at indføre en form for AI til at hjælpe en og hvis dette ikke var muligt et smarter hint system, men det virker meget uoverskueligt, så det blev hurtigt droppet for at få andre dele implementeret og integrere i systemet.

Deroudover nåede vi alle mål vi endte også med at lave winWindow som bliver vist når man vinder spillet for at tilføje mere funktionalitet.

Hvad kunne være blevet gjort bedre

Selve implementationen af hints kunne være gået bedre idet at der blev lavet en god del metoder som derefter er blevet slettet idet de ting som den skulle hænge sammen med krævede for lang tid og derfor skulle blive sat til side.

Planlægningen kunne måske godt havde haft lidt bedre orden. Det gjorde at der nogle gange var lidt problemer med hvem skulle lave hvad, og på hvilke tidspunkter folk skulle lave hvad. Ved lidt eftertanke kunne nogle planlægningsværktøjer være brugbare.

Et andet problem lå i at, når der skulle kommenteres med kode så andre i gruppen eller til slut når den skulle afleveres manglede, det gav nogle problemer når andre skal forstå hvad der sker og derfor blev der kaldt på deres hjælp når koden skal bruges et andet sted. Dette gjorde at folk som var igang med nogle ting konstant skulle hjælpe folk i at forstås deres kode. Til sidst blev koden mere forståelig.

Mulige udvidelser

Selve systemet er lavet til at kunne blive udvidet med både flere flag og bomber i det samme felt i boardet, heri vil der dog skulle laves et nyt Color system og metode for at tjekke for antallet af bomber og antallet af flag der er placeret. Derudover en måde at vise det på som ser respektabelt ud. Det kunne betyde et større billede som bliver kaldt på.

Dette skyldes at det var indtænkt, at man skulle kunne indføre den ændring, inden programmet blev skrevet. Disse små ændring ville ligge i at omsætte flag og bomber fra boolean til int og nogle checks som så bruger dette. Derudover har hints nogle ubrugte funktioner som skulle bruges til mere avanceret AI til minesweeper, som der ikke blev lavet videre på.

Hvad har vi lært

At lave sådan et projekt tager mere tid, end hvad man ellers vil tro. Det som man tror tager et par dage, kan tage mere end man havde troet. Heri hvor lang tid det tager at implementere diverse udvidelser og få de til at spille med resten af koden. Dette medførte også, at AI blev nedprioriteret til et simple hints system.
Vi har lært at bruge JavaFX og mange af objekterne derunder til at lave et sammenhængende program og i dette tilfælde et sammenhængende spil.

Vi lærte også at bruge Github som tillod os at arbejde samtidigt på den samme program. Herved ved dokumentation af push kunne man let finde ud af hvem har lavet hvad og på hvilken måde.

Personlige oplevelser

s204443 Jacob Martens

Mit hovedansvar i projektet lå i at skrive koden til Board- og Tile-klasserne samt mindre dele af de andre systemkomponenter, hvilket jeg synes har været en tilfredsstillende opdeling af programmet. Modelleringen af tilstandsmaskinerne og klassehierarkiet har jeg gjort i samarbejde med de andre gruppemedlemmer over Discord, som gav et godt indtryk af, at vi kunne arbejde sammen om delopgaverne i projektet. I løbet af dette projekt er jeg blevet bedre til at koordinere min tid og arbejde sammen med min gruppe. Jeg har udfordret mig selv i at kommunikere med de andre gruppemedlemmer om opgaven og om, hvordan vi skulle bruge vores tid.

s204445 Simon Peter Sundt Poulsen

Det meste af min tid blev brugt på at implementere highscores og leaderboards (HighscoresWindow / HighscoresController). Jeg brugte størstedelen af de første dage af projektet på at sætte mig nogelunde ind i JavaFX, da jeg ikke havde meget erfaring med at bruge det. Jeg synes det gik nogenlunde med at få sat leaderboards op og synes generelt vi i gruppen var gode til at samarbejde om at integrere vores kodedele og hjælpe hinanden med tests og kode segementer. Især de sidste par dage, hvor der opstod problemer med at få leaderboards til at fungere med jar-filer, var der hjælp at hente i gruppen. Mht. rapporten fokuserede jeg på at skrive om de klasser og tests, jeg selv havde lavet. Da vi alle var færdige med at skrive vores dele af rapporten gennemgik vi rapporten sammen og rettede og kommenterede undervejs, så alle var enige om den endelige rapport.

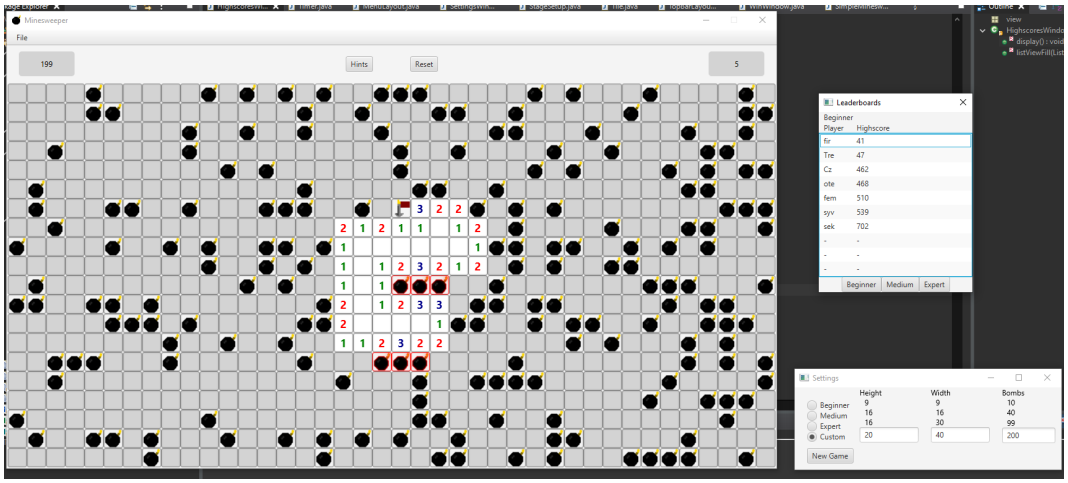
s204459 Christopher Zwinge

Hoveddelen af min tid blev brugt på at skrive kode og at være problemløser når der kom problemer i forskellige dele vores kode. Jeg skrev hovedsageligt rapport omkring de elementer jeg arbejdede mest med i koden og derudover rettede jeg på den tekst der allerede var skrevet. Jeg synes vores gruppearbejde har været meget godt og vores fordeling af arbejde har passet fint med mig.

s183713 Tobias Collin

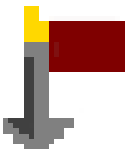
En god del af min tid blev brugt på rapport skrivning, notering og dokumentation af forskellige problemer. Derudover brugte jeg også tid på at hjælpe andre. Ingen rigtige problemer med gruppen og det gik overraskende godt. Over tid blev jeg bedre til at fordele min tid og arbejde med gruppen til at lave et produkt. Deri sidde og diskutere den bedste implementation og ikke den første af forskellige kode dele.

Appendix



Figur 6: Færdigt spil der er åbent

Figur 7: Bombe pixelart - Christopher



Figur 8: Flag pixelart - Christopher