

JANUARPROJEKT

02121 INTRODUKTION TIL SOFTWARETEKNOLOGI

Afleveringsgruppe 16:

Abinav Aleti s224786

Yahya Alwan s224739

Aslan Behbahani s224819

Rasmus Wiuff s163977

github.com/rwiuff/Reversi 


20. januar 2023

INDHOLD

	Side
1 Introduktion & Problemanalyse	1
1.1 Tilgængelighed	1
1.2 Forfatterskab	1
1.3 Opgaven	1
1.4 Specifikationer	1
1.5 Målsætninger	1
1.6 Tilgang til opgaveløsningen	2
2 Design	2
2.1 MVC	2
2.2 Main	2
2.3 Controller	2
2.4 Board	3
3 Implementering	3
3.1 Board Design	3
4 Test	3
5 Project Management	3
5.1 Tidsplan og tidsstyring	3
5.2 Arbejdsfordeling	4
5.3 Git, Github og Gradle	5
6 Konklusion	5
Listings	6
Figurer	7
Tabeller	7
Appendicer	8
A Afsnitsforfattere	8

1 INTRODUKTION & PROBLEMANALYSE

1.1. TILGÆNGELIGHED

Hele projektet (inklusive denne rapport) er tilgængeligt online på [Github](#) . I mappestrukturen er *“BasicReversi”* og *“AdvancedReversi”* rodmapper for hver deres projekt som kan importeres i Eclipse og Visual Studio Code. Læs mere i Afsnit 5.3.

1.2. FORFATTERSKAB

En oversigt over hvilke gruppemedlemmer der har skrevet hvad i rapporten kan ses i Appendiks A.

1.3. OPGAVER

I dette projekt er målet at udvikle et program der implementerer spillet “Reversi”; et brætspil der handler om at vinde mest muligt terræn ved at indkapsle og vende modstanderens brikker¹. Vi benytter os af regelsættet i projektoplægget, og jf. samme oplæg skrives den grafiske brugerflade med JavaFX.

Med udgangspunkt i Model-Viewer-Controller konceptet (*MVC*) skal der implementeres en klasse med den underliggende logik for spillet, en klasse der implementerer den grafiske brugerflade, og en klasse der binder disse sammen.

1.4. SPECIFIKATIONER

Reversi er et relativt simpelt spil med få regler. Vi ved, at vi skal lave et bræt med 8x8 felter. I begyndelsen af spillet skal der placeres 4 brikker af to forskellige farver, én farve pr. spiller. Herefter skal de to spillere placere deres brikker én efter én, og vende modstanderens brikker ved at klemme dem inde mellem ensfarvede brikker. Brikkerne kan kun stilles et sted, hvor man med sikkerhed klemmer en brik af modsat farve inde. Vinderen er den spiller, der har flest brikker på brættet, når alle 64 felter er udfyldt. Alternativt kan man vinde spillet, ved at modstanderen siger ”pas” to ture i træk.

- Begyndende spiller: Tildeles tilfældigt mellem de to spillere. Ved efterfølgende ture, skiftes spillerne.
- Startkonfiguration: Første spiller vælger to placeringer inden for de midterste fire felter. Næste spillers brikker optager resterende to felter og første spiller har næste træk.
- Placering af brikker: En brik må kun ligges på et tomt felt således at en eller flere af modstanderens brikker indkapsles mellem den lagte brik og en af spillers andre brikker. Indkapslingen kan ske ad vertikale, horisontale og diagonale retninger.
- Vending af brikker: Ved placering af en brik vil alle modstanderens, i det øjeblik, indkapslede brikker mellem den lagte brik og spillerens egne brikker blive vendt.
- Spillets afslutning: Når brættet er fyldt eller begge spillere har meldt pas efter hinanden.
- Vinder: Den spiller som har flest brikker (og dermed vundet mest terræn) har vundet spillet.
- Spillet skal være indeholdt i et eksekverbart jar-arkiv som skal kunne køre på alle maskiner (selv uden at JavaFX er installeret på end-user maskinen).

Ud fra disse specifikationer bliver programmet udviklet. Der er artistisk frihed ift. brikkers og brættets farve, hvor stort brættet skal være ift. computerskærmen og hvordan layout’et skulle se ud mht. placering af eventuelle knapper og tekstområde ift. selve brættet.

1.5. MÅLSÆTNINGER

Projektets primære mål er at udvikle et program der overholder specifikationerne ovenfor. Spillet har skulle kunne køre, og været testet adskillige gange for robusthed og eventuelle små fejl.

Krav til brugerfladen er at letlæselighed, overskuelighed, og simpel struktur, uden unødvendig redundans. Programmets struktur skal ligeledes være simpelt og uden redundans, således at det er nemt at redigere i koden, for at implementere ændringer undervejs, da dette kan blive en stor udfordring, skulle man have skrevet noget

¹[Reversi på Wikipedia](#)

sammenflettret og kompliceret².

Sekundære mål er at implementere så mange valgfri features som muligt, som vi dømte inde for vores faglige kapacitet. Igen hjælper det her med simpel og overskuelig kode, der gør, at man let kan genbruge noget, man tidligere har skrevet, til at udvide og forbedre programmet.

Et andet vigtigt mål er at arbejde som en gruppe. Med det menes der at lære, hvorledes man arbejder sammen, dele arbejdet op når det kommer til et projekt af denne størrelse, og tage stilling til hvordan ens kode kan sammenflettes med de andre gruppemedlemmers kode.

1.6. TILGANG TIL OPGADELØSNINGEN

Projektet har budt på adskillige overvejelser ift. løsning af problemer. Som det første havde vi det visuelle, og dernæst havde vi det logiske. Vi blev enige om at dele arbejdet op i grupper af to og to. Så kunne vi spørge os selv: Hvordan vil vi gribe vores opgave an? Hvad angår den grafiske del, besluttede vi os for at bruge *Scene Builder*³ til at lave vores *Scene* og nogle af vores *Nodes*. Fordelen her er at have al grafisk info stående på en let overskuelig måde og kun definere interaktionen i vores klasser. Ulempen vil dog være, at det vil tage længere tid at skrive hver enkelt ting op (specificere en knaps layout, størrelse mm). En anden fordel ved Scene Builder er, at det er hurtigt, overskueligt (inde i selve programmet), og det er let at forbinde evt. kode fra andre klasser med det, man har designet i programmet.

Dog er det ikke alt, der ikke let kan løses i Scene Builder. Detaljer som actions og id på nodes er nemmere at redigere direkte i *FXML* filen efter layoutet er defineret i Scene Builder.

2 DESIGN

2.1. MVC

Vi har som løsning til opgaven valgt at lave 3 klasser. Disse klasser er skabt efter 'Model-View-Controller' - mønsteret. Derudover har vi også en *.fxml* fil, der indeholder al information om vores stage og scene. Denne fil redigeres igennem Scene Builder. Dette vælger vi dog ikke at gøre, da det, i vores optik, vil være mere praktisk at designe det grafiske i scene-builder, da man kan placere diverse objekter/nodes som 'Buttons', 'Panels' osv. på 'scene' og redigere dem, uden at skulle have en eksakt viden om de specifikke koordinater, hvorpå vi stiller dem, og uden at skulle skrive deres længder, bredder osv. ned. Vores klasser er de tre følgende:

2.2. MAIN

En main-klasse, der fungerer som view; altså alt det, som spilleren kan se. Den sætter altså programmet igang, viser stage, scene, nodes osv. En board-klasse, der fungerer som model; den står for alt al logik, der bruges til at implementerer de forskellige regler og funktioner ved spillet, som f.eks. at vende brikker, der er indeklemte, eller at afgøre, om hvorvidt man kan stille en brik på et bestemt felt.

2.3. CONTROLLER

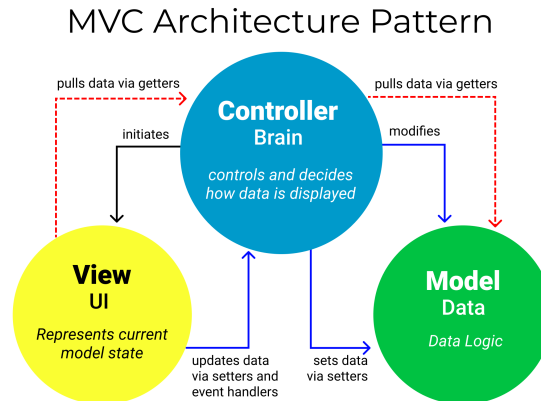
En controller-klasse, der fungerer controller; denne klasse binder board og *.fxml*-filen sammen. Dette gør den ved at lave en masse metoder, der responderer på et klik med musen eller et klik på en "Button" på "scene", således at der sker nogle ændringer visuelt, og disse ændringer har forklaring med rod i logikken (board-klassen).

²Spaghettikode

³Scene Builder

2.4. BOARD

Figur 1: Model Viewer Controller konceptet (af Rafael D. Hernandez) freecodecamp.org)



Som en tilføjelse, der ikke vil indgå i den endelige aflevering, og som ikke indgår i eksekveringen af de tre klasser for oven, laver vi også en board-driver klasse, der vil være i stand til at kunne teste "board" klassens funktionsdygtighed. Dette gør vi, eftersom der ikke vil være mulighed for at teste logikken rent visuelt, før controller/scene-builder delen er færdig. Dette skyldes, at controller-klassen modtager og arbejder med data fra board, og hvis controllerklassen så ikke er færdig, er den ikke i stand til at gøre dette på en ordentlig måde.

3 IMPLEMENTERING

3.1. BOARD DESIGN

Brættet er lavet af en 8x8 gridpane med en pane i hver celle. Hver pane indeholdt i gridpanes cellen er tildelt et unikt id. For eksempel er 05 et unikt id tildelt til en pane, hvor 0 repræsenterer rækkeindekset, og 5 repræsenterer kolonneindekset for panes placering i gridpane. Hver pane er tildelt et id for at gøre det nemmere at placere/tegne eller slette en brik fra brættet.

4 TEST

Når vi skulle teste programmet, havde vi forskellige tilgange til det, alt efter hvad vi arbejdede med. Os, der arbejdede med det visuelle, skulle bare åbne programmet og se, om vi var tilfredse med overfladen, og om de diverse knapper gjorde som de skulle, når vi klikkede på dem. Vi skulle ikke arbejde med nogen kompliceret logik.

5 PROJECT MANAGEMENT

I følgende afsnit vil der blive diskuteret de valgte løsninger til at organisere projektet og løse opgaven.

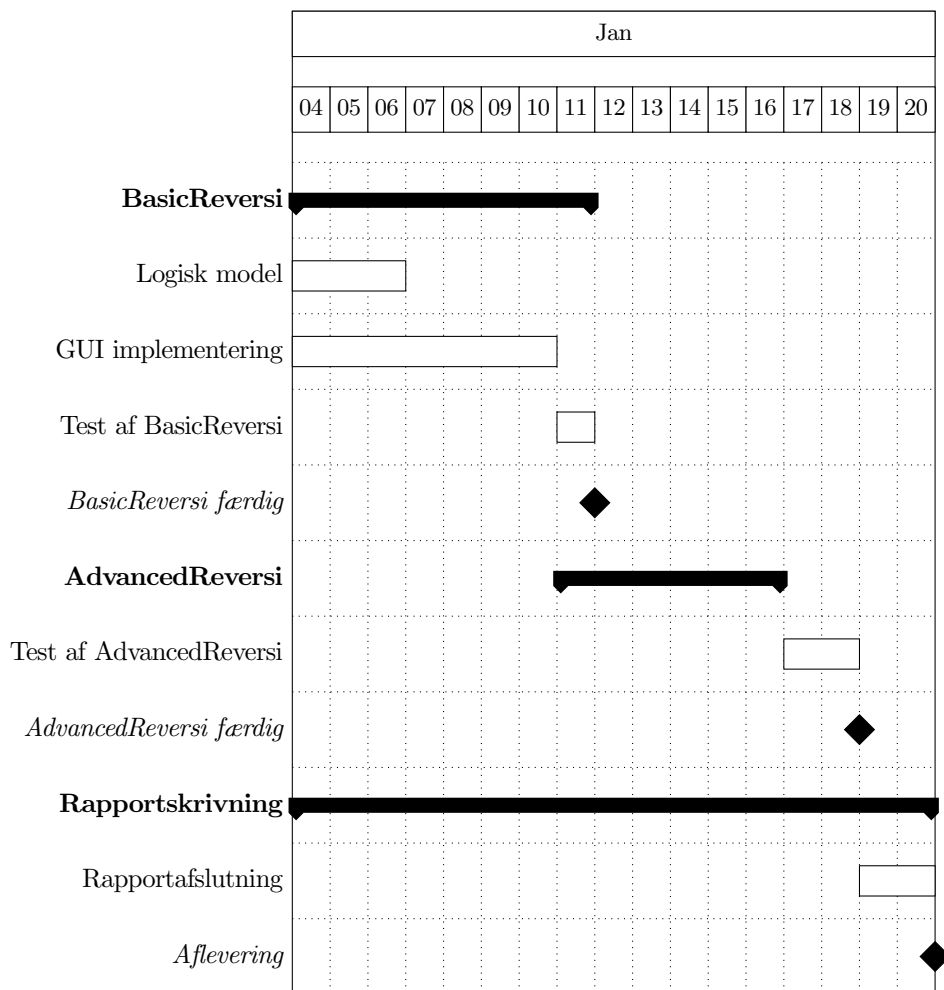
5.1. TIDSPLAN OG TIDSTYRING

Indledningsvist blev gruppen enige om en række principper og en tidsplan for at optimere udviklingsplanen. Principperne lyder:

1. Nær-daglig status på underopgaver.
2. Overlap mellem test og implementation af funktioner.
3. Alle planer er tentative.
4. Fælles udvikling af logik og algoritmer i vigtige metoder.

Den oprindelige tidsplan kan ses i Figur 2.

Figur 2: Første tidsplan (fra Projektplanen)



Under projektet overholdte vi ovennævnte principper, men tidsplanen ændredes markant. BasicReversi var færdig to dage over tid og rapportskrivningen blev lempeligt udskudt til den sidste uge.

5.2. ARBEJDSFORDELING

I forhold til arbejdsfordelingen er projektet opdelt i tre faser:

1. Logik & Brugerflade
2. Controller klassen
3. AdvancedReversi funktionaliteter

I første fase arbejder gruppen i makkerpar. To personer nærstudere JavaFX og prøver at implementere den ønskede brugerflade og giver deres *concerns* til det andet makkerpar. Det andet makkerpar arbejder på at implementere logikken bag brættet og brikkerne i en model klasse.

I anden fase arbejder gruppen sammen på at løse forskellige problemstillinger. En kunne f.eks. arbejde på den overordnede struktur af metodekald i et spil Reversi, en anden arbejder på at oversætte museklik til koordinater, en tredje arbejder på reaktioner fra knapper, og en sidste på opdatering af det grafiske bræt ud fra den interne tilstand af bræt objektet. Til sidst en større sammenfletning som leder til en controller klasse, og efter test af spillet, et færdigt produkt. I tredje fase bliver det oprindelige spil forgrenet via Git og hvert gruppemedlem implementerer et af gruppens tilføjelser af gangen. Når en implementation er færdig flettes denne med hovedgrenen. Her er der valgt en person som mestendels arbejder med at flette de forskellige grene sammen.

5.3. GIT, GITHUB OG GRADLE

Som projektstyringsværktøjer er der valgt følgende løsninger:

- [Gradle](#): Projektstyringsværktøj til at automatisere *dependencies* (JavaFX), mappestrukturer og pakning af jar-filer.
- [git](#) [Git](#): Til at kommunikere mellem lokale maskiner og [Overleaf](#), hvor rapporten kan skrives i fællesskab.
- [Github](#): Til at håndtere versionskontrol mellem medlemmer i gruppen.

6 KONKLUSION

LISTINGS

FIGURER

1	Model Viewer Controller konceptet (af Rafael D. Hernandez) freecodecamp.org	3
2	Første tidsplan (fra Projektplanen)	4

TABELLER

1	Forfatterskab i rapporten	8
---	---------------------------	---

Appendices

A AFSNITSFORFATTERE

Tabel 1 viser en oversigt over hvilke gruppemedlemmer der har skrevet hvilke afsnit i rapporten.

Tabel 1: Forfatterskab i rapporten

Navn	Afsnit
Abinav Aleti	
Yahya Alwan	
Aslan Behbahani	Afsnit 1.3, 1.5, 1.6 og 2.1
Rasmus Wiuff	Afsnit 1.1, 1.4 og 5