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

Abinav Aleti **s224786**
Yahya Alwan **s224739**
Aslan Behbahani **s224819**
Rasmus Wiuff **s163977**

1 Introduktion & Problemanalyse

1.1. Arbejdsfordeling

Arbejdsfordelingen under projektet kan ses i Appendiks A.

1.2. Opgaven

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 indkapse og vende modstanderens brikker[1] 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.

1.3. Specifikationer

Reversi er et relativt simpelt spil med få regler. Vi ved, at vi skal lavet 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 spilleres 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.

1.4. 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 implementerer ændringer undervejs, da dette kan blive en stor udfordring, skulle man have skrevet noget sammenfiltret og kompliceret[2].
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] Reversi på Wikipedia
[2] Spaghettikode

Abinav Aleti **s224786**
Yahya Alwan **s224739**
Aslan Behbahani **s224819**
Rasmus Wiuff **s163977**

1.5. Tilgang til opgaveløsningen

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 nemere at redigere direkte i *FXML* filen efter layoutet er defineret i Scene Builder.

2 Design

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 er blevet redigeret igennem scene-builder. Alle vores klasser skrev vi i JDE'en Eclipse,

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.

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 knap på "scene", således at der sker nogle ændringer visuelt, og disse ændringer har forklaring med rod i logikken (board-klassen).

3 Implementation

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

6 Konklusion

[3] Scene Builder

Abinav Aleti **s224786**
Yahya Alwan **s224739**
Aslan Behbahani **s224819**
Rasmus Wiuff **s163977**

Listings

Abinav Aleti **s224786**
Yahya Alwan **s224739**
Aslan Behbahani **s224819**
Rasmus Wiuff **s163977**

Figurer

Tabeller

Abinav Aleti **s224786**
Yahya Alwan **s224739**
Aslan Behbahani **s224819**
Rasmus Wiuff **s163977**

Appendices

A Arbejdsfordeling

Tabel 1 viser fordelingen af opgaver og metoder/klasser, og hvor i rapporten man kan læse om samme.

Tabel 1: *Arbejdsfordeling i projektet*

Navn	Opgaver / klasser / metoder	Referencer i rapporten
Abinav Aleti		
Yahya Alwan		
Aslan Behbahani		

Rasmus Wiuff	Gradle (build-fil, mappestruktur, o.lign.)
	Versionsstyring (Sammenfletnnig af features, o.lign.)
	`Board.moveAnalyser(int colour)`
	`Board.getValidMoves()`	