


# JANUARPROJEKT

---

02121 INTRODUKTION TIL SOFTWARETEKNOLOGI

## Afleveringsgruppe 16:

Abinav Aleti s224786  
Yahya Alwan s224739  
Aslan Behbahani s224819  
Rasmus Wiuff s163977  
[github.com/rwiuff/Reversi](https://github.com/rwiuff/Reversi) 


20. januar 2023

# INDHOLD

	Side
<b>1 Introduktion &amp; Problemanalyse</b>	<b>1</b>
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
<b>2 Design</b>	<b>2</b>
2.1 MVC	2
2.2 Main	3
2.3 Controller	3
2.4 Board	3
<b>3 Implementering</b>	<b>3</b>
3.1 Board	3
3.2 Main	5
3.3 Controller	6
<b>4 Test</b>	<b>11</b>
4.1 Controller-klassen	11
4.2 Board-driver klassen	11
4.3 Unit tests	11
<b>5 Project Management</b>	<b>11</b>
5.1 Tidsplan og tidsstyring	11
5.2 Arbejdsfordeling	12
5.3 Git, Github og Gradle	13
<b>6 Konklusion</b>	<b>14</b>
6.1 Vores mål	14
6.2 Vores mangler	14
6.3 Hvad har vi lært	15
<b>Figurer</b>	<b>16</b>
<b>Tabeller</b>	<b>16</b>
<b>Appendicer</b>	<b>17</b>
<b>A Afsnitsforfattere</b>	<b>17</b>

## 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 brikker<sup>1</sup>. 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

<sup>1</sup>[Reversi på Wikipedia](#)

sammenflettret og kompliceret<sup>2</sup>.

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 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*<sup>3</sup> 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.

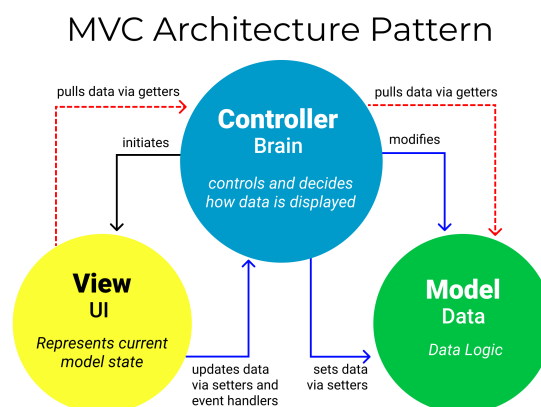
## 2 DESIGN

---

### 2.1. MVC

Vi har som løsning til opgaven valgt at lave 3 klasser. Disse klasser er skabt efter 'Model-View-Controller' - mønstret. 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.

Figur 1: Model Viewer Controller konceptet (af Rafael D. Hernandez) [freecodecamp.org](https://www.freecodecamp.org)



Vores klasser er de tre følgende:

---

<sup>2</sup>Spaghettikode

<sup>3</sup>Scene Builder

## 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).

## 2.4. BOARD

Som model i MVC konceptet laves en klasse der indeholder brættet som en datastruktur, og som indeholder logikken til at placere brikker, hvilke af modstanderens brikker der bliver indeklemmt samt lovligheden af et givent træk. Board klassen skal kommunikere med controller klassen igennem en række heltals koder.

# 3 IMPLEMENTERING

## 3.1. BOARD

Modellen for brættet implementeres i `Board.java` klassen. Tabel 1 indeholder klassens felter.

*Tabel 1: Felter i `Board.java`*

Felt	Formål
<code>int[][] board</code>	Datastruktur der repræsenterer brættet.
<code>int turnCount</code>	Turnummeret i spillet
<code>int boardsize</code>	Størrelsen $n$ for et $n \times n$ bræt.
<code>int forfeitCounter</code>	Antal gange en spillerne har meldt pas
<code>ArrayList&lt;int[]&gt; flipped</code>	Log over de sidste vendte brikker
<code>HashMap&lt;Integer, String&gt; players</code>	Spillernes interne ID og navne
<code>HashMap&lt;String, HashMap&lt;Integer, HashMap&lt;Integer, Integer[]&gt;&gt;&gt; validMoves</code>	Datastruktur for mulige træk for en given farve, en given tur.

Brættet repræsenteres af en heltalsmatrix med værdierne 0 for tomt felt, 1 for hvid placering og 2 for sort placering. Ved at gemme turnumre og antal gange spillere har meldt pas kan board klassen afgøre hvis tur det er og om spillet er slut. Loggen over vendte brikker endte med ikke at blive brugt, men formålet var at controller klassen skulle vise hints om hvilke brikker der ville blive vendt ved et givent træk. Det blev stemt ned da det gør spillet for nemt. Tabel 2 indeholder klassens metoder.

Tabel 2: Metoder i *Board.java*

Metode	Formål
<code>Board()</code>	Konstruktør for brættet. Kalder <code>setPlayers()</code> .
<code>getBoard()</code>	Getter for board feltet.
<code>setPiece(int row, int column, int colour)</code>	Sætter manuelt brik, uanset lovlighed. Bruges til unit tests.
<code>getTurn()</code>	Getter for <code>turnCount</code> .
<code>turnClock()</code>	Øger <code>turnCount</code> med 1.
<code>getPlayers()</code>	Getter for spiller hashmap.
<code>getValidMoves()</code>	Getter for <code>validMoves</code> hashmap.
<code>setPlayers()</code>	Tildeler tilfældigt spillere en farve.
<code>setPlayers(int id1, String player1, int id2, String player2)</code>	Tildeler spillere specifikke farver.
<code>setPlayerName()</code>	Setter for specifikt spillernavn.
<code>resetBoard()</code>	Nulstiller brættet, turnummer og pastæller.
<code>turnState(int colour)</code>	Kalder <code>moveAnalyser(colour)</code> , og udregner om der er ledige træk.
<code>gameOver()</code>	Afgør om spillet er slut hvis brættet er fyldt, eller der er meldt pas to gange i træk.
<code>initPlace(int row, int column, int colour)</code>	Indeholder logik for at udfylde startfelterne.
<code>place(int row, int column, int colour)</code>	Placere en brik hvis placeringen er tom og indeholdt i <code>validMoves</code> . Kalder <code>flip(move, colour)</code> .
<code>flip(String move, int colour)</code>	Vender brikker ved et lovligt træk.
<code>filled()</code>	Afgør om brættet er fyldt.
<code>checkWinner()</code>	Tæller brikker og afgør vinderen.
<code>moveAnalyser(int colour)</code>	Finder lovlige træk (se Afsnit 3.1.2).
<code>findOwn(int[] checkboard, int i, int j, int direction, int colour)</code>	Findere mulig indkapsling af modstanderen.
<code>saveFlips(int[] ownPiece, int i, int j, int direction)</code>	Gemmer indkapslede brikker.

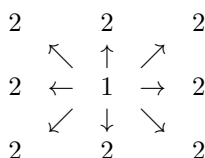
De fleste metoder er simple if/else eller for-løkker, og de er vel dokumenterede i kildekoden.

Den vigtigste funktion i *Board.java* er `moveAnalyser(int colour)` metoden som skal finde og afgør alle træks lovlighed for et givent bræt setup og en given brikfarve. Metoden gennemgås i Afsnit 3.1.2.

**3.1.1. AdvancedReversi board klasse** Der er enkelte ændringer i den avancerede udgave af spillet. `setPlayers()` metoden ændres til ikke at foretage tilfældige valg, da brugeren selv skal skrive navne ind i systemet. Pastælleren, `gameOver()` og `filled()` fjernes helt, for at overlade denne logik til controller klassen da dette bliver en mere integreret del af brugerinteraktionen. Til gengæld introduceres to metoder: `checkBlackScore()` og `checkWhiteScore()` som bruges til at live-opdatere optaget terræn for de to spillere.

**3.1.2. moveAnalyser() metoden og validMoves feltet** Flere gange i løbet af spillets afvikling er det praktisk at vide om hvilke træk der er lovlige for en given farve. Betragt Figur 2. Her ses at en vilkårlig brik har 8 nabofelter. Særligt fælde er i kanten af brættet. For at undgå særligt fælde blev første implementationsvalg her at udfører alle tests i et bræt med *padding*. Derfor starter `validMoves` ud med at lave en kopi af brættet med en kant af nuller.

Figur 2: En briks naboer



Herefter itererer metoden over alle felter i brættet. Mødes et tomt felt testes alle 8 retninger for en modstander. Hver retning har sin egen heltalskode. Hvis der er en modstander itererer metoden i den retning og leder efter en brik af egen farve for at fastslå en indkapslingsmulighed, med metoden `findOwn()`. Hvis metoden støder på kanten af brættet eller en tom placering indstilles søgningen. Dennes algoritme beskrives mere grafisk i Figur 3.

**Figur 3:** `findOwn()` leder efter en indkapsling i nordvestlig retning (heltalskode 1).

(a) Første skridt	(b) Andet skridt
1 0 0 0	1 0 0 0
0 2 0 0	0 2 0 0
0 0 2 0	0 0 2 0
0 0 0 1	0 0 0 1

Den fundne nabobriks koordinater gemmes og returneres til `validMoves`. Hvis `validMoves` modtager sådan et koordinat iværksættes `saveFlips` som gemmer de indkapslede brikker i et hashmap efter samme skridt som i Figur 3. Her vil de røde brikkers koordinater blive gemt.

Når `validMoves` har et hashmap med hashmaps af koordinater fra alle retninger gemmes disse i `validMoves` hashmappet med trækets koordinat som nøgle. Tabel 3 viser strukturen af `validMoves`.

**Tabel 3:** Datastrukturen `validMoves` efter afsøgningen i Figur 3

Træk	Retning	Vendbare brikker
<code>HashMap&lt;String,</code>	<code>HashMap&lt;Integer,</code>	<code>HashMap&lt;Integer, Integer[]&gt;&gt;</code>
"3,3"	1	0,{2,2} 1,{1,1}

Årsagen til at retning indgår som nøgle i `validMoves` er simpel: Alle værdier skal have en unik nøgle. Hvis ikke f.eks. retning blev brugt ville `moveAnalyser` bare overskrive brikker fundet i en retning med brikker fundet i en tidligere retning.

### 3.2. MAIN

**3.2.1. Basic** I Main klassen, `FXMLLoader` klassen indlæser "main.fxml", som indeholder layoutet for, hvordan spillet skal se ud visuelt. Ved at indlæse `FXML`-filen tildeler den det resulterende overordnede element til variabelen "root", og opretter et nyt `Scene`-objekt med "root" som dets rodelement. Dette sceneobjekt er indstillet som den nye scene i den primary stage. Endvidere tilføjes en titel samt ikonbilleder til den primary stage. Den opretter også en reference til controller-objektet, der oprettes, når `FXML`-filen indlæses, ved at kalde `'getController()'`-metoden på `FXMLLoader` objektet. Den kalder derefter `'in()'` fra controller-klassen. Til sidst viser dette `primaryStage` til brugeren.

**3.2.2. Avanceret** I lighed med `BasicReversi` Main klassen udfører `AdvancedReversi` Main klassen de samme funktioner. `Basic Reversi` indlæser "main.fxml"-filen, som kun indeholder boardet, genstartsknappen og en label for at vise meddelelserne til brugeren, hvilket er en af forskellene mellem de to Main klasser. I modsætning hertil indlæser `AdvancedReversi` Main klassen spillets hovedmenu, "menu.fxml." Tre separate knapper, inklusive `BeginGame`, `HighScore` og `Exit`, er til stede i denne hovedmenu. "Main.fxml"-filen, som indeholder boardet og andre sofistikerede funktioner, indlæses af `FXMLLoader`, når der trykkes på `BeginGame`-knappen. `Highscore`-knappen kan bruges til at se spillets højeste score, mens `Exit`-knappen kan bruges til at lukke applikationen. En anden forskel er, at hver gang "menu.fxml" udføres, oprettes en fil med navnet "HighScore.txt" til at gemme `highscore` information, hvis en sådan file ikke allerede eksisterer. Derudover giver `Avanceret Reversi` mulighed for at spille spillet i fuldskærmstilstand og beder brugeren om at bekræfte sit valg, når man trykker på `exit`.

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

### 3.3. CONTROLLER

#### 3.3.1. Basic

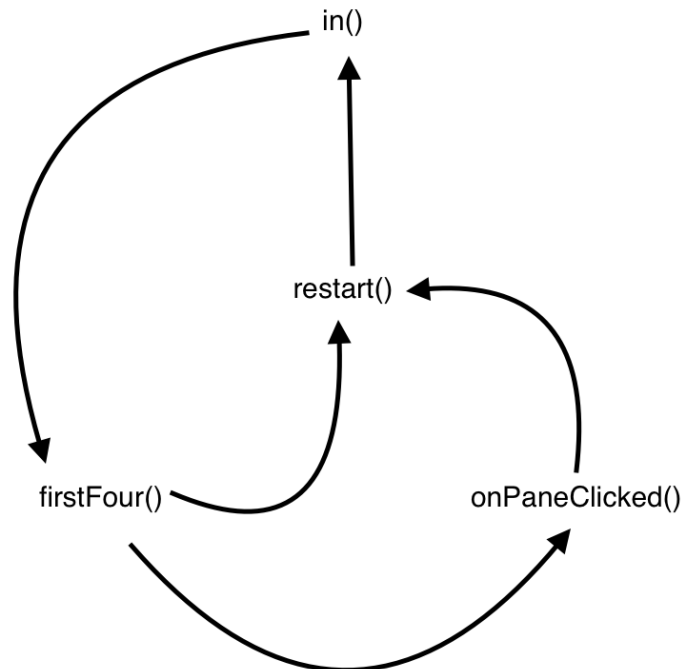
*Tabel 4: Metoder brugt i BasicReversi Controller.java*

Metode	Kalder metoderne:	Beskrivelse
in()		tildeler spillerne deres farver
firstFour(int row, int column, Pane pane)		Spilleren kan sætte de første fire brikker
onPaneClicked(MouseEvent event)	getRowIndex() getColumnIndex() firstFour() update()	Dette fet der bliver klikket på et felt Giver spillerne besked, om hvorvidt deres træk er lovlige Melder pas, når der ikke er noget lovligt træk Annoncerer spillets slutning, når der ikke er Erklærer vinderen
Restart()	in()	Genstarter spillet. Spiller der lavet første træk i sidste runde, begynder anden.
DrawCircle(int id, Pane pane)		Tegner hvid og sort brikker på brættet, baseret på spillerens tur
reset()		Itererer gennem brættet og rydder det
update()	drawCircle()	Opdatere brættet og tegner cirkler(brikker) på brættet ifølge Board objektet
getRowIndex(MouseEvent event)		Returnerer rækkeindekset for pane, der er klikket på
getColumnIndex(MouseEvent event)		Returnerer søjleindekset for pane, der er klikket på

**3.3.2.** En alternativ model til at kortlægge de overordnede metoder i basis-versionen af Reversi kan se således ud:



Figur 4: Basic Reversi: overordnede metoder



Den røde tråd er meget enkel: spillet igangsættes med metoden 'in()', hvor spillerne bliver tildelt deres farver. Derefter skal den første spiller sætte de første fire brikker ved 'firstFour()'. Nu kan spillerne vælge at genstarte spillet ved 'restart()', i hvilket tilfælde de føres tilbage til 'in()', eller de kan spille videre, hvilket vil føre dem videre til 'onPaneClicked'. Spillet fortsætter, indtil en af afslutningsbetingelserne er opfyldt, hvorefter spillet afsluttes, pointene optælles og vinderen kåres. Nu kan spillerne vælge at genstarte spillet, hvilket tager dem tilbage til 'in()'.

### 3.3.3. Avancerede overordnede metoder

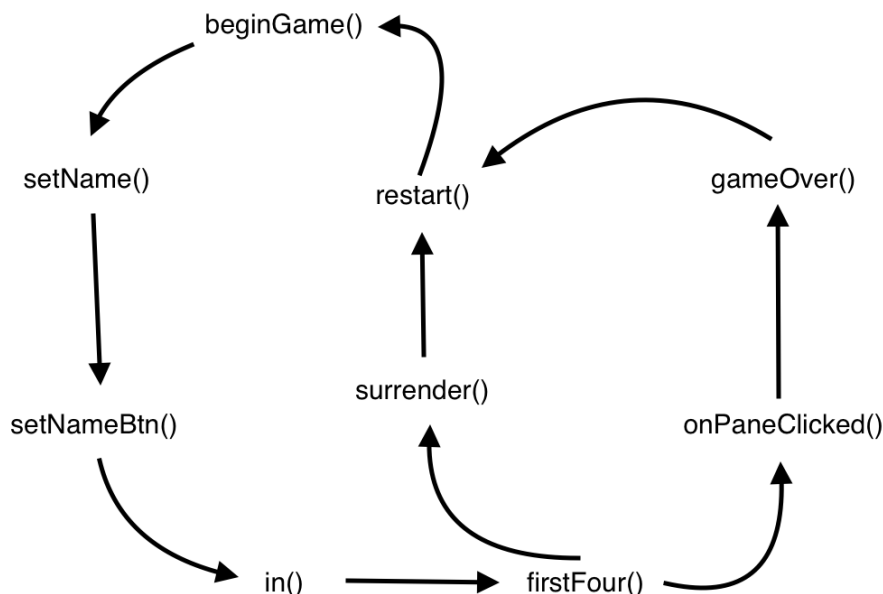
I den avancerede controller klasse er der rykket om på rækkefølgen, hvori metoderne kaldes. Nedenfor ses en tabel, der viser kronologien af de "overordnede metoder", samt de metoder, de kalder.

*Tabel 5: Overordnede metoder i AdvancedReversi*

Metode	Kalder metoderne:	Beskrivelse
beginGame()	setName()	Starter spillet og loader Reversi brættet.
setName()	setNameBtn() checkScore()	Anmoder spillerne om at indtaste deres navne.
setNameBtn()	in()	Fastlåser navnene til spillerne.
in()	drawCircle()	Tildeler spillerne deres farver og viser de fire første mulige træk.
firstFour()	update() checkscore()	Får spiller nr. 1 til at sætte de fire første brikker.
onPaneClicked()	getRowIndex() getColumnIndex() firstFour() hideLegalMoves() update() checkScore() showLegalMoves()	Styrer hvad der sker, når der bliver klikket på et felt. Annoncerer spillets slutning, når der ikke er flere lovlige træk.
gameOver()	loadHighScore() checkHighScore()	Erklærer vinderen og opdaterer highscoren (hvis den er blevet slået)

En alternativ repræsentation af metodernes rækkefølge kan vises ved figuren nedenfor:

Figur 5: AdvancedReversi: overordnede metoder



Her man se, at det hele begynder ved `'beginGame()'`. Skriver spillerne deres navne og fortsætter til `'firstFour'`, hvorefter de nu er i stand til- (men ikke tvunget) at overgive sig, ved at trykke på `'surrender'`. Dette vil føre dem til at trykke på `'restart'`, hvorefter kredsløbet gentager sig. Hvis de ikke giver op, føres de videre til `'onPaneClicked()'`, som fungerer som spillets "hovedfase", hvor spiller sætter brikker og fylder brættet ud. I sidste ende, når der ikke er flere lovlige træk, sendes spillerne videre til `'gameOver()'`, hvor pointene tælles op og vinderen kåres.

En metode der er ganske bemærkelsesværdig under controller-klassen er `'beginGame()'`. Denne metode, som kaldes ved et tryk på knappen "Begin Game" fra hovedmenuen, igangsætter hele spillet, viser brættet og kalder `'setName()'`-metoden. Hvad der er usædvanligt ved denne metode er, at den fungerer ikke just som en controller-metode, der henter data fra `'Board'` og bearbejder det. Den fungerer snarere som en metode inde fra `'Main'`-klassen, altså klassen der fungerer som "view", hvis vi tager udgangspunkt i MVC konceptet. Den har til job at skulle "vise", men den befinder sig inde i `'controller'`-klassen. Grunden til at den befinder sig her, er fordi den kaldes via. et "ActionEvent event", altså kaldes den, når spilleren interagerer manuelt med knappen i hovedmenuen.

### 3.3.4. Avancerede Underordnede Tabel 6

#### Metoder

*Tabel 6: Metoder brugt til avancerede funktioner og i overordnede metoder `AdvancedReversi Controller.java`*

Metode	Kaldte metoder	Beskrivelse
<code>restart(ActionEvent event)</code>	<code>reset()</code> <code>setName()</code>	Genstarter spillet og beder spillerene om at indtaste deres navne i tekstfelterne igen. Spiller der lavet første træk i sidste runde, begynder anden
<code>reset()</code>		Itererer gennem brættet og rydder det
<code>update()</code>	<code>drawCircle()</code>	Opdatere brættet og tegner cirkler (brikker) på brættet ifølge Board objektet
<code>exitGame(ActionEvent event)</code>		lukker spille
<code>checkScore()</code>		Opdaterer scoren for begge spillere
<code>saveHighScore(int score,String name)</code>		Gemmer vinderens score og navnet adskilt af komma inde i "Highscore.txt" file, hvis vinderens score er højre end nuværende highscore
<code>loadHighScore()</code>		Læser highscore og spillersnavne fra "Highscore.txt" filen og returnere de som arrayet.
<code>showHighScore(ActionEvent event)</code>	<code>loadHighScore()</code>	Når man trykker på highscore-knappen, vises en advarselsboks/-prompt, der viser highscore og spillerens navn, hvis highscore er indstillet.
<code>surrender(ActionEvent event)</code>		Annoncere overgivelsen af den spiller der trykkede på knappen, og den anden spillers sejr
<code>getRowIndex(MouseEvent event)</code>		Returnerer rækkeindekset for pane, der er klikket på
<code>getColumnIndex(MouseEvent event)</code>		Returnerer søjleindekset for pane, der er klikket på
<code>showLegalMoves(int color)</code>	<code>drawCircle()</code>	Tegner en cirkel hvor spiller kan mulig placer en brik
<code>hideLegalMoves()</code>		Rydder brættet for cirkler, hvilket indikerer mulige træk
<code>mainMenu(ActionEvent event)</code>		Returner brugeren tilbage til main-menu af spillet
<code>DrawCircle(int id, Pane pane)</code>		Tegner hvid og sort brikker og brikker det kan mulig placer på brættet, baseret på spillerens tur

**3.3.5. Genstarter Spillet** Et af de grundlæggende kriterier for det basis Reversi-spil er evnen til at genstarte spillet uden at genstarte applikationen. Det er gjort muligt at genstarte spillet uden at genstarte programmet ved at oprette en knap. Genstart-knappen havde oprindelig funktionalitet til at genstarte spillet ved at skifte scener, det vil sige at skifte til scenen, hvor ingen brikker er placeret på brættet. Den tom brættet er den samme scene, der vises ved kørsel af FXML-filen i starten af spillet/programmet. Genstart af spillet med at skifte scener var ikke et optimalt og vellykket middel til at genstarte spillet, da det ikke tillod brugeren at spille spillet igen uden at lukke programmet, på grund af den uretmæssige initialisering af spillet samt kun indlæse scenen (tom brættet) med ingen funktionalitet. For at undgå denne ukorrekte genstart blev hver pane indeholdt i GridPane tildelt et id (fx: 05 repræsenterer Pane

i 0. række og 5. kolonne i GridPane). Hver gang man trykker på genstart, vil to for-loops iterere gennem hver række og kolonne i GridPane, få adgang til den unikke identifikator for hver Pane ved hjælp af række- og kolonneindekset og derefter rydder brikkerne på den. At tildele et id til ruderne og bruge for-loops til at iterere gennem Pane og slet brikkerne placeret på dem, gav den mest effektive og effektive metode til at genstarte spillet igen. Derudover, når spillet genstartes, vil den spiller, der foretager det første træk, før spillet genstartes, nu starte som nummer to.

## 4 TEST

### 4.1. CONTROLLER-KLASSEN

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 kikkede på dem. Vi skulle ikke arbejde med nogen kompliceret logik. Vi testede konstant programmet, da detaljerne var vigtige, som f.eks. at cirklerne skulle placeres i midten af et felt, at der skulle skiftes farve mellem hvert klik osv.

### 4.2. BOARD-DRIVER KLASSEN

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.

### 4.3. UNIT TESTS

For at tilgodese at Board klassens logik virker efter ændringer blev en testklasse, `BoardTest.java` skrevet som kører hver gang et jar-arkiv kompileres eller klassen køres. Testen tester setter og getter metoder for player hashmappet, moveAnalyser og turnState metoderne.

## 5 PROJECT MANAGEMENT

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

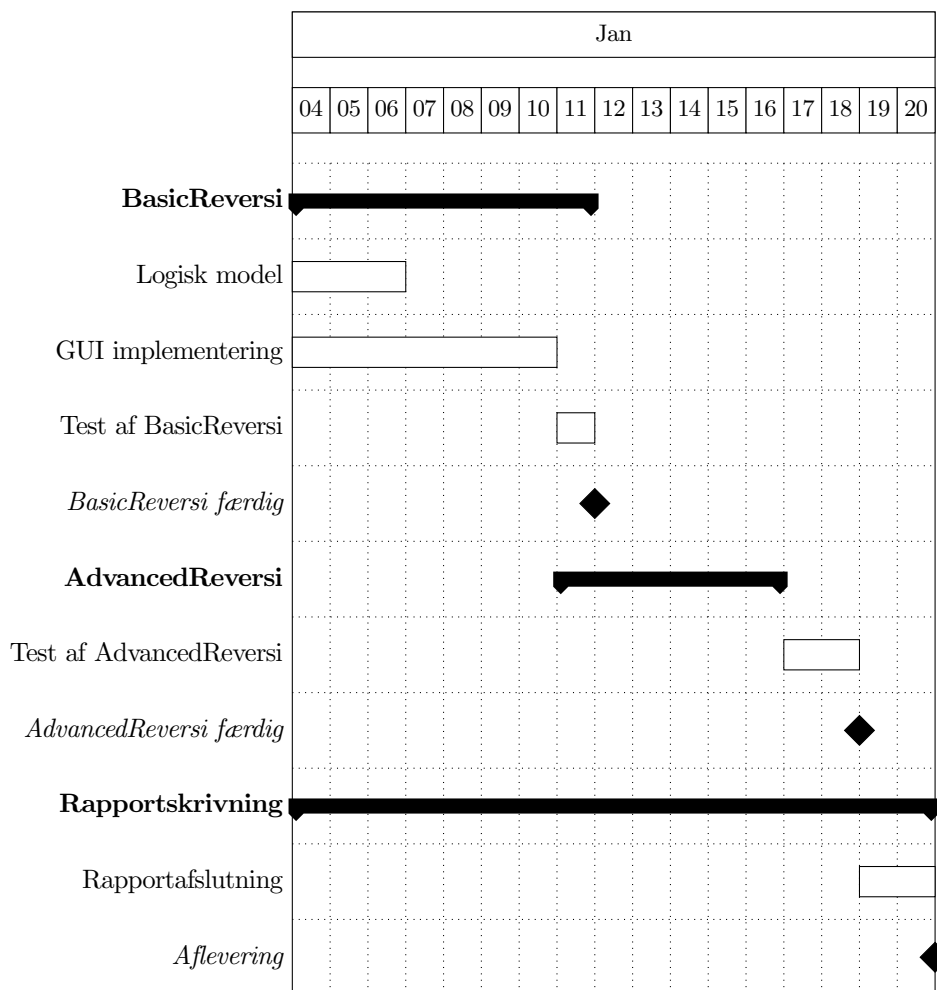
### 5.1. TIDSPLAN OG TIDSSYRING

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

1. Nær-daglig status på underopgaver.
2. Alle planer er tentative.
3. Fælles udvikling af logik og algoritmer i vigtige metoder.

Den oprindelige tidsplan kan ses i Figur 6.

Figur 6: 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.

**5.2.1. Basic** Måden vi valgte at arbejde på i den Basic-Reversi havde både fordele og ulemper.  
Fordele:

- At fokuserer på de tildelte metoder så man kan gå helt i dybden med disse fejl man møder under processen, samtidig med at forberede sig til en forklaring på det til de andre gruppemedlemmer, så alle er med på opdateringer i forbindelse med metoden og de løsninger man bringer med hvis man oplever nogle fejl, hvilket også føret til at styrke samarbejde og kommunikation i gruppen.

Ulemper:

- Hvert medlem har sat mere fokus på de tildelt/valgte metoder som man have ansvar for, hvilket gør at man ikke sætter ligevægt på alle processorerne i projektet. Vi talte bestemt med hinanden om alle faser og gav tilstrækkelige forklaringer på de ideer og koder, der blev brugt, samt løsninger, hvis der var fejl, men det opnåede læring vil ikke være på samme niveau, som hvis vi havde arbejdet sammen på alle faser med samme effektivitet.

**5.2.2. Advanced** Avanceret del bajet på at udvikle noget vi allerede har, eller tilføj nye funktioner hvor man synes at det kunne være interessant for spiller, og som gør spillet mere effektiv. Måden vi har arbejdet på i den avanceret del, var lidt mere individuelt end Basis. Vi begyndte at tale om de forskellige idéer vi havde. Derefter gik vi i gang med de udvalgte idéer, da alle har en idé eller to at arbejde videre med. Der har været lidt mere plads til at arbejde med de faser som man ikke havde i Basis del, fx Yahya og Rasmus havde hoved ansvar for logikken (Board class) i den Basis version, hvor imod den avanceret, valgte Yahya at arbejde mere med Design " main.FXML", hvor han tog en del af Design Udviklingen, hermed (farver, knapper, størrelser, tekster), for at komme ind på detaljer og lærer SceneBuilder bedre at kende.

### 5.3. GIT, GITHUB OG GRADLE

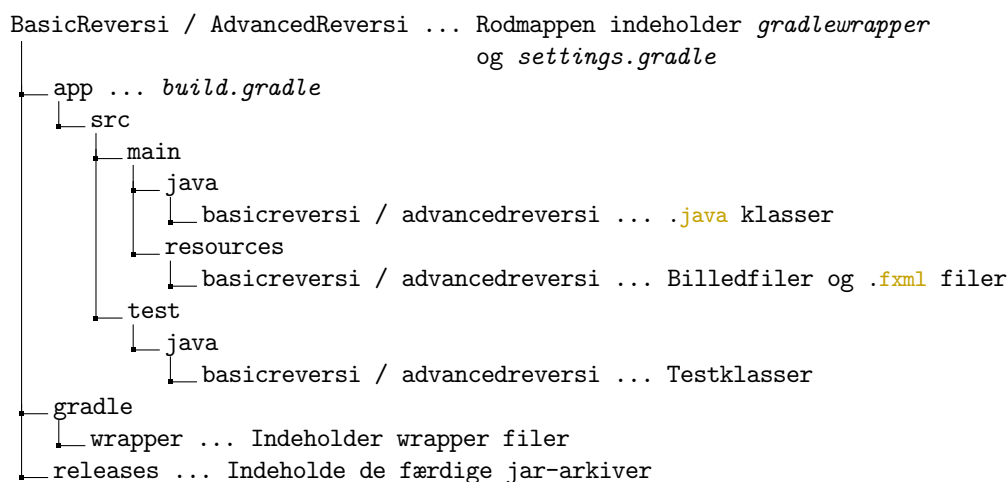
Som projektstyringsværktøjer er der valgt følgende løsninger:

- **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.
- **Gradle**: Projektstyringsværktøj til at automatisere *dependencies* (JavaFX), mappestrukturer og pakning af jar-filer.

**5.3.1. Git og Github** Som nævnt i Afsnit 1.1 er projektet på Github <https://github.com/rwiuff/Reversi>. Dette *repository* (repo) er udgangspunkt for versionskontrol i projektet. Desuden bruges Overleafs Git integration til at synkronisere projektet med lokale maskiner. Git som versionstyringsprogram har været essentielt under projektet. Der har været meget læring i at bruge *branch* og *merge* funktionerne til at kunne arbejde på forskellige aspekter af kode i de samme klasser på samme tid. Det har mestendels været uden de største problemer, men der er til stadighed meget mere at lære om samarbejde med VCS.

**5.3.2. Gradle** Via Gradle kan mappestrukturen i Figur 7 genereres. Mappestrukturen indeholder desuden også bin og build mapper som bruges under kompilering og pakning af jar-arkiver, men disse er kun midlertidige instanser af det program man kører.

Figur 7: Mappestrukturen i JavaFX projekterne BasicReversi og AdvancedReversi



Gradle tilgodeser automatisk udførsel af skrevne tests, håndtering af kompilerede .class filer og pakning af jar arkiver. Desuden tilgodeser Gradle at dependencies er tilgængelige for miljøet der åbner projektet, uden at man skal importere JavaFX moduler manuelt. Er der en fejl i den version af JavaFX der bruges, kan man blot opdatere versionsnummeret i *build.gradle* og herefter opdateres JavaFX.

## 6 KONKLUSION

### 6.1. VORES MÅL

Vi satte os til mål at skulle skrive programmet Reversi i sin mest basale form. Dette opnåede vi, og dermed opfyldte vi vores minimums-krav. Dernæst satte vi for os selv, at vi ville lave så mange avancerede tilføjelser til programmet som muligt. Vi nåede at tilføje flere udvidelser, som forbedrede spillerens brugeroplevelse, hvilket var en sejr, men der var også forsøg på udvidelser, hvor vi måtte se os slået. Se næste vores mangler for yderligere info. Vores kode endte med at være let overskueligt og struktureret. Vi var i stand til at implementere flere udvidelser, enten via. genbrug af kode fra den basale kildekode, eller ved indsættelse af ny kode enkelte steder, uden at forpurre systemet. Ofte kunne vi ganske enkelt oprette en metode, og så kalde den de steder, hvor det var relevant. Vi lærte også at arbejde som en gruppe inden for software/programmerings-feltet. De fleste af os var nye til brugen af "Git-Hub", men takket være Rasmus' erfaring og kyndighed med programmet, lærte vi hurtigt hvordan vi kunne arbejde sammen, dele kode, push'e, pull'e, og skabe nye grene, hvorpå vi kunne arbejde på vores udvidelser i fred, uden nogen uventede ændringer i koden.

### 6.2. VORES MANGLER

**6.2.1. SpeedReversi** En pointe fremhævet i den avanceret del af Yahya, er, at give spillere mulighed for at spille med en timer.

- Timeren er en vigtig del af de fleste strategispil (spil, der kræver en vis forhånds planlægning til næste træk), fordi timeren øvre spillernes tænkehastighed og tvinger dem til at tænke hurtigere og træffe beslutninger for næste træk under pres, hvilket kan føre til, at man træffer en forkert beslutning eller man løber tør for tid. denne tilføjelse til denne type spil gør spillet sjovere for nogle.

I gruppen aftalte vi om at Yahya starter med at implementerer timer metoden ind i vores standard logik og få den til at køre direkte når spillet starter, derefter vil vi så tilføje en Bottom til " menu.fxml " kaldes " Speed-Reversi ", der befinder plads under "BeginGame" (normal game). På den måde adskiller vi "SpeedMode", så den bliver en ekstra valg hvis man ønsker det. Timer-Metoden tog ca. 2 dag at implementerer, for der har været nogle udfordringer undervejs så som:

- Der blev brugt "Thread" til manipulere timeren med spillet, men fejlen var at ved bruge af "Thread.sleep" vil det tage programmet et sekund at kører metoden igennem, det har vi oplevet da vi har prøvet at spille hurtigt, så vil timeren ikke begynd at tælle ned, for den har brug for et sekund at tænke.
- som et forsøge på at løse fejlen, har vi brugt at forkorte rundturstiden for "Thread.sleep" ved at brug "System.nanoTime" hvor man kan regne tiden som metoden bruger i nanoTime og træk det fra. Det hjalp ikke, fejlen var det samme.
- Et andet forsøge, var med at prøve "Animationtimer" men vi kunne ikke få metoden til at ændre timeren fra spiller 1 til spiller 2.
- Løsning: Der blev brugt "TimeLine" hvor den virkede fint med både turnsskiftehastighed mellem spillerne og regering med "getturn()".

**6.2.2. Flette Reversi Timer** Som nogle af de største udfordringer vi har haft med SpeedMode er, at kombinær metoden ind og få den til at køre adskillet, Dvs. kun når man vælge at spille med SpeedMode, som vi har lavet en knappe til i "Menu.FXML" og en "BorderPane" i "Main.FXML".

#### 6.2.3. Metode implementering

- Efter oprettelsen af "ReversiTimer" klassen som indholder Timer metoden, har Yahya tænkte sig at oprette en anden Controller klasse for ReversiTimeren således at, man kan kald den ny "controller1" når man trykker på knappen "SpeedReversi" ellers kaldes den nuværende "Controller". På den måde vil vi kunne nemt flette Timer logikken ind med Standard logik uden væsentlig ændringer i koden, men der har været uenighed om det, fordi det kunne være et problem hvis vi sanerer opdager nogle fejl eller hvis vi skal ændre på noget i hoved Controller klassen, for så bliver vi nødt til at ændre det også i Controller2 klassen.



- Derfor har vi tænkt os at få Timeren ind med at tilføje en "boolean" som sætter "SpeedMode=false;" og så begyndt vi at tilføje "if (speedMode == true){ else" til metoderne ( in(),restart(),onPaneClicked(),beginGame) og så har vi sat "speedReversi(ActionEvent event){ speedMode = true; "
- Vi var meget tæt på at få det hele til at køre, men udfordringer var stadig til stede. Vi mødte en fejl der gør at enten timeren kan ses på begge Mode ellers forsvinder den i begge Mode også, dvs. at der har været en sammenhæng mellem den normale logik og den nye med Timeren. Efter en lang analyse af "if" og "else" sætninger, fandt

**6.2.4. Antal Mulige vendinger** Aslan forsøgte sig med en udvidelse til udvidelsen, der gjorde spillerne i stand til at se deres mulige træk. Idéen med udvidelsen var den, at man kunne ved at trykke på en knap, kunne se på de mulige træk, hvor mange brikker der vil blive vendt. Altså hvis man ved at sætte en brik på en mulig placering, vil vende 3 brikker, vil der stå tallet 3 i midten af den mulige placering. Problemet ved implementeringen opstod, da programmet skulle se fremad, fremfor bagud. Strategien var, at skabe en integer i board-klassen, kaldet "tilesFlipped". Hver gang metoden 'flip()' blev kaldt, skulle denne integer sættes lig 0, og jeg satte den til at inkrementere hver gang en brik blev vendt, således at man altid vil få det eksakte tal vendte brikker. Dernæst modificerede jeg 'showLegalMoves' til at vise 'tilesFlipped' i midten af cirklerne. Problemet var følgende: For det første, viste alle cirkler det samme tal. Derudover viste cirklerne altid hvor mange brikker der blev vendt SIDSTE runde. Målet var, at de skulle vise tallene en runde FREMAD. Forsøgene nåede deres ende, ikke på grund af fejl og exceptions, men ganske enkelt grundet en blokade. Jeg havde ingen anelse om, hvor jeg skulle implementere hvad. Nøglen lå i board-klassens 'validMoves', men grundet min manglende erfaring med hashmaps, som board-klassen i høj grad består af, så jeg mig nødsaget til at give op.

**6.2.5. Lyd** Lyd er en af de andre avancerede funktioner, som Abinav forsøgte at implementere. Lyd ville forbedre brugeroplevelsen og hjælpe spillere med at fordybe sig i spillet, hvilket gør det til en fremragende funktion at inkludere. Efter hvert træk ville funktionen have ladet deltagerne vide, hvordan spillet forløb. Derudover ville det have givet spillerne feedback om, hvorvidt de havde placeret brikken korrekt på brættet. At inkludere lyd i spillet ville have mindsket behovet for, at spillere skulle læse teksten efter hvert træk, eller mens de placerede brik på brættet. Musikfilerne blev importeret ved hjælp af Java FX-værktøjer som mediaPlayer og media. Fejl som fil kan ikke findes, og fil kan ikke indlæses, blev stødt på under adgang til mediefiler. Try-catch-sætninger blev brugt til at rette op på disse problemer. Hver lyd- eller mediefil havde en separat metode, der indeholdt funktionerne mediaPlayer.play(), mediaPlayer.stop() og mediaPlayer.stop(). De nævnte teknikker gjorde det muligt at afspille, pause og stoppe musikken. Det vigtigste problem, der opstod under integration af lydkomponenten, var, at musikfilerne af og til, når programmet kørte, ville blive korrupte. Musikfilerne fungerede korrekt, når de blev afspillet ved hjælp af VCL eller groove-musik på computeren. Det var umuligt at afgøre, hvad der fik musikfilerne til at blive korrupte, da programmet kørte. Et andet problem, der udviklede sig, var lydets manglende evne til altid at spille som reaktion på brugerens aktivitet. Der var adskillige forsøg på at rette denne fejl, herunder at kontrollere, om handlingshændelsens kode kaldte den relevante lydfil, men ingen af dem lykkedes. Til sidst blev det besluttet ikke at tilføje funktionen lyd til Reversi, da de fejl, der blev opstået ved at tilføje funktionen, ikke var i stand til at rette og krævede et højt kendskab til java fx mediaPlayer og medieklasser.

---

## 6.3. HVAD HAR VI LÆRT

**6.3.1. Aslan** Dette projekt har bidraget meget til mit billede af, hvordan det er at arbejde inden for software/programmeringsfeltet. Jeg var i begyndelsen meget usikker på, hvordan man skulle arbejde som gruppe på ét projekt, da vi tidligere afleveringsopgaver ganske enkelt har delt de fire forskellige opgaver op imellem os. Jeg fandt dog hurtigt ud af, at systemet er det samme, og dog ikke. Man uddelegerer opgaver, men sparer også meget med hinanden, og kigger på tingene sammen. Man arbejder på at forbedre hinandens arbejde, så man i fællesskab kan komme frem til det bedst-mulige produkt. Derudover har dette forløb også bare været en god mulighed for at udvikle mine programmerings- og problemløsningskompetencer. Det har været fedt at kunne stå over for et problem, og have muligheden for at teste forskellige løsninger af døgnnet rundt, frem for at skulle bekymre sig om andre fag. Jeg håber på flere forløb som dette i fremtiden, hvor vi alle er mere erfarne og kompetente programmører.

**6.3.2. Yahya**

**6.3.3. Rasmus**

**6.3.4. Abinav**

## FIGURER

1	Model Viewer Controller konceptet (af Rafael D. Hernandez) <a href="https://freecodecamp.org">freecodecamp.org</a>	2
2	En briks naboer	4
3	<a href="#"> findOwn()</a> — leder efter en indkapsling i nordvestlig retning (heltalskode 1).	5
4	Basic Reversi: overordnede metoder	7
5	AdvancedReversi: overordnede metoder	9
6	Første tidsplan (fra Projektplanen)	12
7	Mappestrukturen i JavaFX projekterne BasicReversi og AdvancedReversi	13

## TABELLER

1	Felter i <a href="#"> Board.java</a> —	3
2	Metoder i <a href="#"> Board.java</a> —	4
3	Datastrukturen <a href="#"> validMoves</a> — efter afsøgningen i Figur 3	5
4	Metoder brugt i <a href="#"> BasicReversi Controller.java</a> —	6
5	Overordnede metoder i AdvancedReversi	8
6	Metoder brugt til avancerede funktioner og i overordnede metoder AdvancedReversi <a href="#"> Controller.java</a> —	10
7	Forfatterskab i rapporten	17

# Appendices

## A AFSNITSFORFATTERE

Tabel 7 viser en oversigt over hvilke gruppemedlemmer der har skrevet hvilke afsnit i rapporten.

*Tabel 7: Forfatterskab i rapporten*

Navn	Afsnit
Abinav Aleti	Afsnit 3.2.1 til 3.2.3, 3.3.5, 6.2.5 og 6.3.4 og Tabeller 4 og 6
Yahya Alwan	
Aslan Behbahani	Afsnit 1.3, 1.5, 1.6, 2.1 til 2.4, 4.1, 4.2, 6.1, 6.2.4 og 6.3.1, Figurer 4 og 5 og Tabel 5
Rasmus Wiuff	Afsnit 1.1, 1.4, 3.1, 4.3 og 5, Tabeller 1 til 3 og Figurer 2, 3, 6 og 7