## Del 3: Spørsmål

Her skal dere besvare noen spørsmål. Dere vil bli vurdert på hvorvidt dere evner å

reflektere rundt egne valg, forståelsen dere viser for objektorientert programmering og

pensum i emnet. Dere skal besvare følgende spørsmål:

1.Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (For

eksempel bruk av arv, interface, delegering osv.)

3.Hvordan forholder koden deres seg til Model-View-Controller-prinsippet? (Merk:

det er ikke nødvendig at koden er helt perfekt i forhold til Model-View-Controller

standarder for å få full uttelling på dette spørsmålet. Det er mulig (og bra) å

reflektere rundt svakheter i egen kode)

4.Hvordan har dere gått frem når dere skulle teste appen deres, og hvorfor har

dere valgt de testene dere har? Har dere testet alle deler av koden? Hvis ikke,

hvordan har dere prioritert hvilke deler som testes og ikke? (Her er tanken at

dere skal reflektere rundt egen bruk av tester)

5.Har dere møtt på noen utfordringer i løpet av prosjektet? Hva ville dere gjort

annerledes en annen gang?

## Del 1 - Beskrivelse

Appen vår består av et sjakkspill med grafisk brukergrensesnitt med mulighet for å lagre et pågående parti, som en senere kan laste inn og fortsette på. Flytting av brikker skjer ved å klikke på en brikke og klikke på feltet en vil flytte den til. Vi forhåndsberegner alle lovlige trekk, og de blir vist som sirkler på brettet etter en har klikket på en brikke. Algoritmen vår tar hensyn til alle regler, og den avslutter spillet både ved sjakk matt og uavgjort ved patt. Ved bondeforvandling skriver en inn den brikken en vil erstatte bonden med i et tekstfelt.

## Del 2

1.Hvilke deler av pensum i emnet dekkes i prosjektet, og på hvilken måte? (For

eksempel bruk av arv, interface, delegering osv.)

* Abstrakte klasser (Piece) -> gikk vekk fra arv
* Iterator (BoardTilesIterator)
* Comparator (lambda uttrykk)
* Lambda streams

I sjakk er det naturlig å uttrykke brikkene, brettet og rutene på brettet som objekter. Originalt hadde vi alle disse klassene, men vi valgte å droppe brettet på grunn av at vi støtte på problemer med innkapsling.

//**Abstrakte klasser**

Brikkene har en overordnet abstrakt klasse Piece som de individuelle brikkeklassene arver fra. Grunnen til at vi valgte å gjøre Piece til en abstrakt klasse, er fordi dette ikke er en egen brikke som eksisterer i spillet, men det generelle konseptet om en brikke. I tillegg til at subklassene arver metoder, så er en annen fordel med at en bruker ikke kan konstruere et Piece objekt at konstruktøren til Piece kan ha skjulte parametere som blir fylt ut med forhåndsbestemte verdier i konstruktøren til brikkeklassene. Piece klassen vår inneholder en ID som blir brukt i kontrolleren til å hente riktig bilde for å vise på brettet. IDen til en brikke kan dermed fylles inn anonymt i konstruktøren i brikkeklassene som sikrer at den alltid vil være korrekt.

**//Iterator**

I sjakk må en ha en måte å representere rutene for å kunne gjøre endringer som å flytte brikker fra og til dem. Vi endte opp med å bruke kartesiske koordinater i form av en int array. Vi lagde brettet vårt ved å lage en todimensjonal array av Tile-objekter, og kan dermed bruke koordinatene til å hente ut Tile-objektet vi ønsker å endre på.

I grafikken representerte vi brettet med et GridPane med ImageViews i hver celle. For å kunne flytte på bildene til brikkene, måtte vi ha en måte å korrespondere et objekt i Tile-arrayet til ImageViewet på den tilsvarende posisjonen i GridPanet. Løsningen vår er at IDen til et ImageView er koordinatene på GridPanet. Siden en ID er en streng, gjorde vi at Game-klassen (som inneholder Tile-arrayet) implementerer Iterable, og lagde en egen iterator som konverterer koordinatene til strengform. Vi bruker så denne til å hente ut ImageViews i GridPanet.

**//Comparator**

Siden vi forhåndsberegner de lovlige trekkene til alle brikkene må vi ha en datastruktur for å representere dette. Vi valgte et HashMap, hvor nøkkelen er koordinaten til en brikke og verdien er en ArrayList med koordinatene til alle rutene den kan flytte til.

I testingen måtte vi kunne sammenligne en stilling som er lagd på forhånd med en som blir generert av Game, og støtte da på problemer siden et HashMap ikke har innbyrdes ordning av elementene. Vi lagde derfor en Comparator i et lambdauttrykk for å sortere elementene etter koordinatene til nøklene.

//**Lambdastreams**

Det var få steder der det var aktuelt å bruke lambdastreams. I spill skjer endring av variabler sine verdier i stor grad gjennom kontrollflyten til programmet, som blir bestemt av algoritmer som er spesifikke til spillet sin funksjon. Det er dermed sjeldent at en får bruk for metodene lambdastreams tilbyr. Der vi fikk bruk for det var av filtrering av variabler som oppfylte visse krav.

**2.Dersom deler av pensum ikke er dekket i prosjektet deres, hvordan kunne dere brukt disse delene av pensum i appen?**