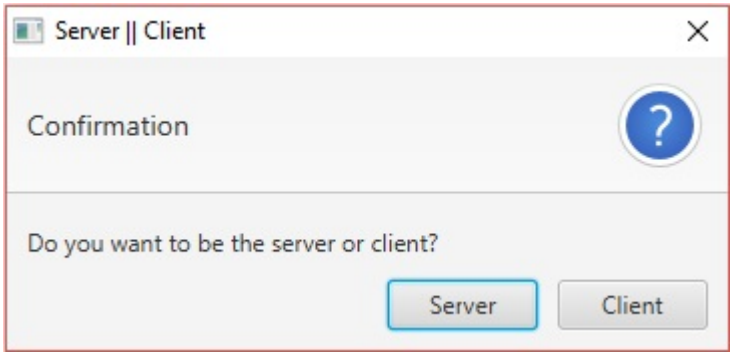
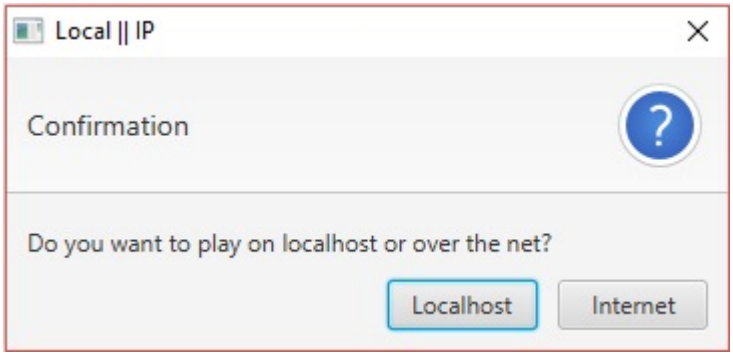
Spillmanual

1.) Oppstart

Spillet starter ved å høyreklikke på EntryPoint og velge Run As Java Application. Deretter får vi opp dialogbokser som guider oss gjennom oppstart.

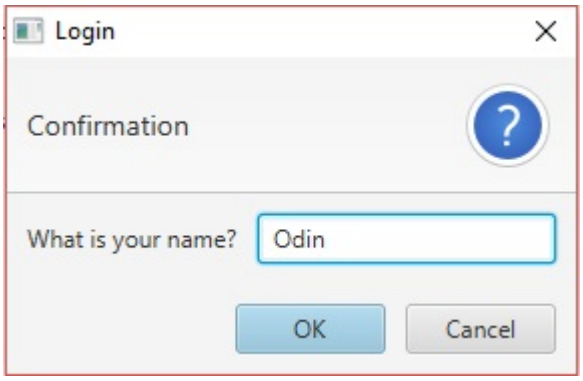


Først må du velge om du skal være server eller klient, for at spillet skal fungere må det først startes en server og så må en klient koble seg til serveren.

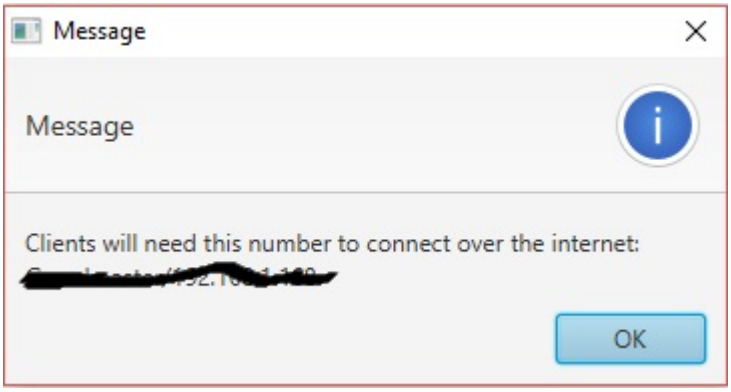


Så må du velge om du vil spille lokalt eller over nett med IP, det er ikke så farlig hvilken du velger men det er viktig at serveren og klienten velger

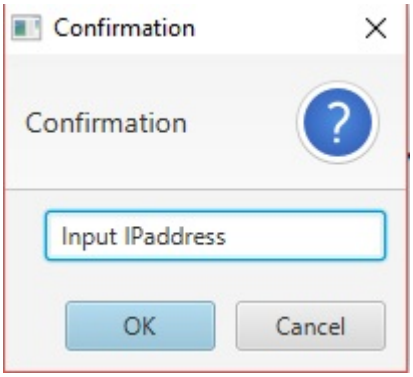
det samme for at de skal kobles opp mot hverandre.



til slutt ber programmet om et navn som skal representere deg som spiller.



Hvis du valgte å være server og du valgte å spille over internett får du en melding som sier hva ip addressen din er.

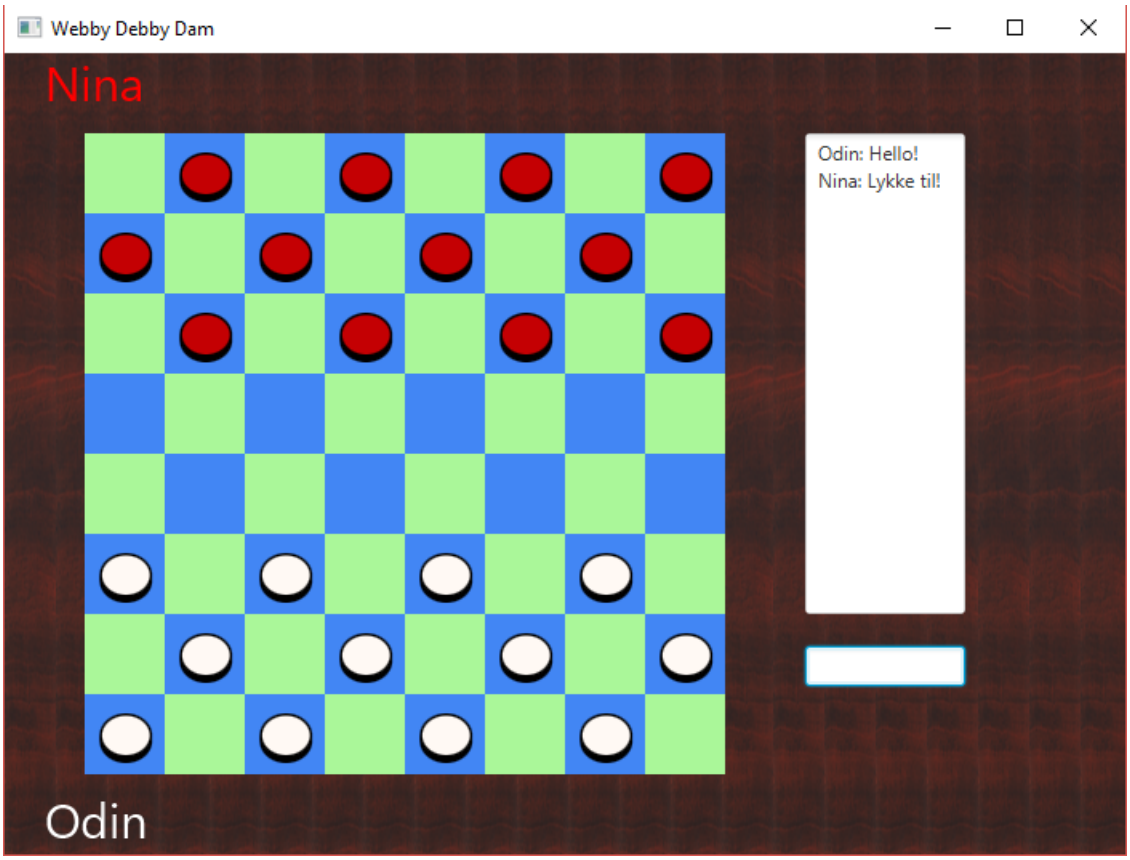


Hvis du valgte å være klient og du valgte å spille på internett må du skrive ip addressen til serveren inn og trykke "OK".

For å spille må man ha to spillere, så hvis du skal spille mot deg selv må du starte applikasjonen først en gang som server, så kjøre java applikasjonen en gang til som klient uten å lukke vinduet med serveren.

Hvis du har fulgt instruksene over burde spillet dukke opp som ett(eller to) vinduer med brukernavn på bunn og top med de fargene som tilhører brikkene spilleren skal bruke.

2.) Hvordan man spiller



Vi har valgt å bruke standard reglene for dam: Hvit starter, 'Men' brikker kan gå og kapre diagonalt forover mens 'konger' kan kapre forover- og bakover diagonalt. 'Men' brikker blir til 'konger' hvis de lander på den siste raden på motstanderens side av brettet. Hvis en brikke **kan** kapres så **må** den kapres, spillet tillater deg ikke å gjøre et ulovlig trekk.

Når en av spillerne har mistet alle brikkene sine så har den andre spilleren vunnet, da kommer det opp en melding til hver av spillerne og når den meldingen klikkes bort lukker spillet seg automatisk.

Chatten på siden kan brukes til å snakke med personen du spiller mot og gir et varsel hver gang spillet ser at en brikke kan kapres eller hvis en av spillerne frakobles forbindelsen underveis.

Teknisk manual

1.) Javafx over Swing:

Vår gruppe mener at Swing ikke var et godt verktøy for å løse denne oppgaven, Swing tegner grafiske elementer over hverandre på ett panel noe som gjør at den eneste måten å flytte på grafiske elementer er å ødelegge alt som er tegnet og så tegne det på nytt bare litt annerledes. Denne metoden er ikke egnet til et program som vårt hvor alle grafiske elementer er kjent på forhånd og vi mener også at denne metoden ikke er særlig objektorientert.

Vi valgte derfor å bruke Javafx for å håndtere grafiske elementer, det fungerer ved å ha containere slik som i Swing, men istedenfor å tegne rett inn i containerne så legger man til grafiske objekter (children nodes) med egne X,Y koordinater og andre attributter. Javafx lar deg så flytte de grafiske objektene ved hjelp av metoden relocate(), uten å trenge å tegne noe som helst på nytt eller slette noe som helst.
Derfor vil vi si at i Javafx er langt bedre egnet for vårt program og mindre utsatt for feil under runtime fordi det lar oss lett dra elementer på skjermen slik som brikkene i spillet uten at spillet må repaintes hundre ganger for å flytte brikken ett hakk forover.

Det eneste problemet med Javafx er at de grafiske elementene kan kun manipuleres av Javafx tråden og ikke av andre tråder vi instansierer. Men dette løses av metoden platform().runlater som syncer opp tråden vi vil bruke med Javafx tråden.

2.) Fordeling av ansvar:

Vi har valgt å dele klassene opp i tre pakker: Network, Model og Logic.

Pakken model inneholder alle de grafiske objektene som må lages for spillet. Det eneste disse klassene gjør er å instansiere grafiske objekter og fester relevant logikk fra logikk pakken til dem under instansieringsprosessen. Det er også her vi finner klassen Tile, den inneholder den statiske variabelen 'TILESIZE' som bestemmer størrelsen på spillet, alle andre grafiske objekters størrelser har blitt satt som en relativ størrelse ganget med TILESIZE så hvis du gjør tallet 50 prosent mindre blir hele spillet 50 prosent mindre.

Pakken logic inneholder to hovedklasser kalt MoveLogic og PieceLogic. PieceLogic er logikken som styrer brikkene og hvordan de beveger seg på brettet, mens MoveLogic er logikken som bestemmer om det trekket du prøver å gjennomføre er et lovlig trekk eller ikke. MoveLogic returnerer et enum kalt MoveType som bestemmer om PieceLogic flytter brikken dit du dro den eller setter den tilbake dersom trekket var ulovlig.

Pakken network er hvor vi håndterer alt som håndterer kommunikasjon mellom server og klient, det er også der vi instansierer Game klassen som er ett slags kontrollsenter som inneholder mesteparten av variablene som blir brukt til å holde styr på hvem sin tur det er til å gjøre ett trekk, om du er serveren eller klienten, og mye annet.

Ett unntak verdt å nevne er 'Dialogs' som er en klasse som instansierer dialoger som kan betraktes som grafiske elementer men vi valgte å ikke legge denne klassen i 'model' fordi den blir kun brukt av klassen 'Game' i pakken 'network'(så vi kan innkapsle som package-private), dessuten er alle dialogene relatert til nettverket og vi ser ikke nødvendigvis på disse dialogboksene som del av modellen fordi de er ikke del av selve spillet.

Vi mener at vi har fått til en relativt god struktur og det kan man se på hvor mange klasser og metoder som vi har innkapslet som private eller package-private.

3.) Java 8

Vi har brukt lambda, metodereferanser og streams for det det er verdt. En ting vi bør nevne er hvordan streams ble notert i koden. Vi brukte samme notasjon som prof. Venkat Subramaniam som vi mener gjør det på en måte som er veldig enkel å forstå. Her er ett eksempel:

$$\begin{aligned} canCapture = Board.getBoard().getPieceGroup().getChildren() \\ .stream() \\ .filter(e \longrightarrow MoveLogic.getML().canCaptureAdapter(e)) \\ .count() != 0; \end{aligned}$$

Her ser man for seg at .stream er som en vannkran som åpnes og den skyller data ned som en strøm av vann, hver .metode som følger under må påvirke all data som skyller inn fra denne strømmen. I dette relativt enkle eksempelet så har vi en samling brikker over og når vi setter i gang strømmen så blir alle disse brikkene skylt ned gjennom filteret vårt som sjekker om brikkene kan eller ikke kan kapre noen brikker og deretter kommer .count inn og teller de brikkene som kom seg gjennom filteret og derfor kan kapre.

4.) Innkapsling

Vi har innkapslet slik at alle variabler er private (unntatt 'messages' som er public final static fordi den bare instansieres en gang men blir skrevet til ofte og fra flere kilder.) Vi har også passet på at alt som blir definert kun under instansiering har blitt merket final".

Klassene og metodene våre er så private som mulig, vi har gjordt mange av dem package-private fordi de kun brukes av andre klasser i samme pakke og vi har prøvd å ha pakker som inneholder klasser som bruker hverandre.

For å unngå å ha for lite innkapsling har vi tatt mye nytte av Singleton designbruksmønsteret. Den lar oss gjøre om klasser som Board til en statisk instans som kan kalles fra overalt i koden, og da er det mye lettere å skrive kode hvor for eksempel en klasse i nettverket skal gjøre endringer på variabler som befinner seg på brettet slik som i denne koden her:

```
void capturePiece(){
    PieceLogic.getPL().movePiece(
        Board.getBoard().getBoardTiles()[x0][y0].getPiece(),
        Board.getBoard().getBoardTiles(),
        newX,
        newY,
        Board.getBoard().getBoardTiles()[x0][y0],
    );
}
```

Her ser vi at ved å bruke Board som singleton kan vi få tak i alle variablene vi trenger som argument for en metode som tilhører singleton klassen PieceLogic og hele denne prosessen skjer i klassen MovePacket som tilhører network pakken. Vanligvis måtte vi ha gravet oss ned til Board ved å la alle klassene stå public og jobbe oss nedover hierarkiet

Game $\longrightarrow$ *root* $\longrightarrow$ *board*

men fordi board er en singleton kan vi kalle den direkte. Dette gjør koden vår lettere å forstå men også mye bedre innkapslet.