TDDD78 / TDDE30 / 729A85 – Objektorienterad programmering och Java

# Projektbeskrivning

## Schack

## 2024-03-22


**Projektmedlemmar:**

Albin Ekman <albek052@student.liu.se>

**Handledare:**

Simon Hansson <simha158@student.liu.se>

# Innehåll

1. Introduktion till projektet 3

2. Ytterligare bakgrundsinformation 3

3. Milstolpar 4

4. Övriga implementationsförberedelser 6

5. Utveckling och samarbete 6

6. Implementationsbeskrivning 7

6.1. Milstolpar 7

6.2. Dokumentation för programstruktur, med UML-diagram 9

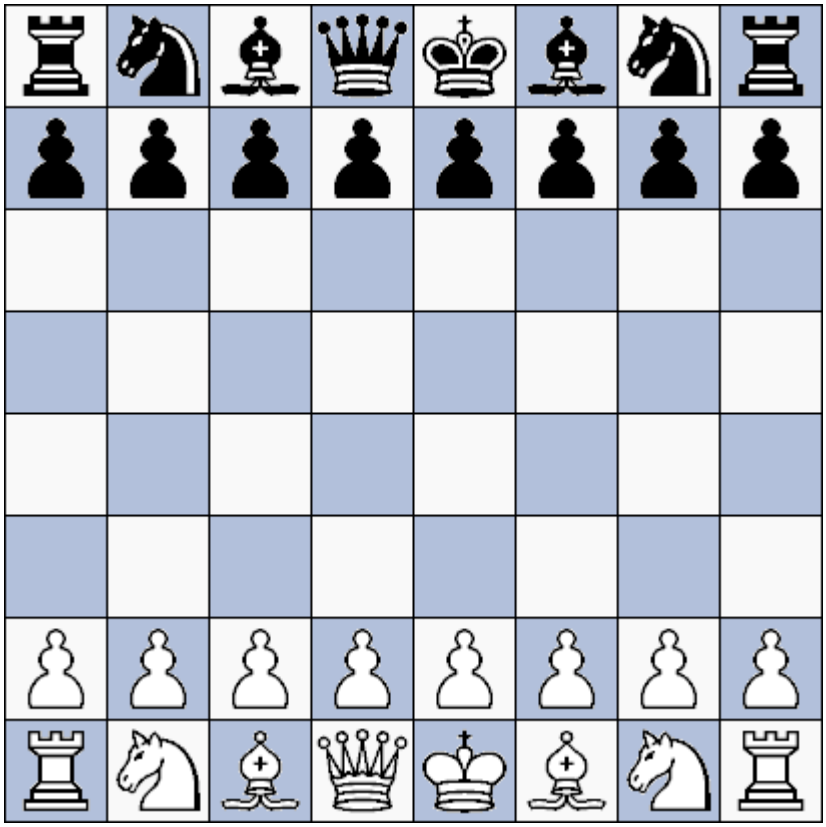
7. Användarmanual 11

# Projektplan

## 1. Introduktion till projektet

Under detta projekt ska ett schackprogram utvecklas där svarta och vita pjäser spelar mot varandra. Spelarna kommer spela på en kvadrat med åtta rutor på varje kant där båda kommer ha lika många pjäser placerade mittemot varandra.

Varje pjäs har ett unikt sätt att röra på sig och pjäser kan dö ifall spelaren/motståndaren kan placera sina pjäser på samma ruta som medspelarens pjäs under spelaren/motståndarens runda. Spelet går ut på att döda medspelarens kung och om detta sker har den som dödat kungen vunnit spelet.



## 2. Ytterligare bakgrundsinformation

Programmet kommer mestadels följa reglerna som presenterade enligt wikipedia. Dock kommer det grundläggande spelet inte att innehålla förmågan för passant eller rockad och spelet kommer enbart kunna ta slut då det blir schackmatt. Bönder kommer inte heller kunna krönas och därav bytas ut mot andra pjäser.

# 3. Milstolpar

| # | Beskrivning |
|---|---|
| 1 | Skapa ett bräde genom board-klassen och rita ut genom ett grafiskt gränssnitt. |
| 2 | Skapa pjäs-klassen med variabler för färg och rörelse och skapa sedan de sex olika pjäserna som ärver av den. |
| 3 | Lägg till pjäserna på brädet och ritas ut dem genom det grafiska gränssnittet. Implementera också kontroller för mus så att användaren kommer kunna röra på pjäser. |
| 4 | Skapa move-klass som innehåller typer av drag för alla pjäser och slutför kontroll med mus för att flytta pjäser. |
| 5 | Implementera unik rörelse för bonde då den är nära fiender. |
| 6 | Förhindra pjäser att placeras på samma ruta om samma färg och döda pjäs om olika. |
| 7 | Skapa game over om kung dör |
| 8 | Skapa spelar klasser och gör så att vit |

| | |
|---|---|
| | spelare rör vita pjäser och svart spelare för svarta pjäser. |
| **9** | Implementera rockad |
| **10** | Implementera passant. |
| **11** | Implementera kröning mot enbart drottning. |
| **12** | Skapa meny så att spelare kan avsluta inifrån spelet. |
| **13** | Lägg till alternativ för timer så att den kan aktiveras genom meny och avslutar om når noll. |
| **14** | Full kröning av bonde för alla typer. |
| **15** | Skapa hightscore för snabbaste vinst. Visa på meny. OBJ |
| **16** | Skapa alternativ för att köra flera matcher på rad. Visa även totala runder och vinster för varje spelare. |
| **17** | Skapa alternativ genom meny så att spelaren kan köra över nätverk. |

| | | |
|---|---|---|
| | **18** | Skapa AI alternativ genom meny så att spelaren kör mot AI istället för andra spelare. |

# 4. Övriga implementationsförberedelser

Spelet kommer innehålla en board-klass som kommer innehålla alla pjäser och deras positioner i form av en array. Varje pjäs kommer vara antingen en kung, dam, torn, löpare, springare eller bonde och kommer ges genom en enum för det grafiska gränssnittet.

Alla dessa pjäser kommer ärva från en pjäs-klass som kommer innehålla färg(enum), boolean för ifall de kan röra sig diagonalt och rakt och slutligen en move-funktion som kollar ifall ett drag är lagligt för att sedan gör det om sant. Bonden och springaren kommer dock att överrida denna check-funktion.

Om move-funktionen känner av en annan pjäs kommer den att checkas och möjligtvis tas bort under board. Kontroller kommer hanteras av en lyssnare som sedan kommer kommunicera med board-objektet för att röra på pjäser.

Spelarna kommer också behöva egna funktioner som kommer innehålla vilken färg de är vilket kommer avgöra vilka pjäser de kan styra.

# 5. Utveckling och samarbete

Målet med projektet kommer vara att avsluta projektet inom deadline innan periodens slut. För att försöka uppnå detta planerar man att komma till varje lektion och resurstillfälle. Helst vill man bli klar med kärnan av spelet inom första hälften av projektets planerade tid.

# Projektrapport

## 6. Implementationsbeskrivning

### *6.1. Milstolpar*

| # | Beskrivning | Implementation |
|---|---|---|
| 1 | Skapa ett bräde genom board-klassen och rita ut objektet genom ett grafiskt gränssnitt. | Implementerad |
| 2 | Skapa pjäs-klassen med variation för färg och rörelse och skapa sedan de sex olika pjäserna som ärver av den. | Implementerad |
| 3 | Lägg till pjäserna på brädet och ritas ut dem genom det grafiska gränssnittet. Implementera också kontroller för mus så att användaren kommer kunna röra på pjäser. | Implementerad |
| 4 | Skapa move-klass som innehåller typer av drag för alla pjäser och slutför kontroll med mus för att flytta pjäser. | Implementerad |
| 5 | Implementera unik rörelse för bonde då den är nära fiender. | Implementerad |
| 6 | | Implementerad |

| | | |
|---|---|---|
| | Förhindra pjäser att placeras på samma ruta om samma färg och döda pjäs om olika. | |
| 7 | Skapa game over om kung dör. | Implementerad |
| 8 | Skapa spelar klasser och gör så att vit spelare rör vita pjäser och svart spelare för svarta pjäser. | Implementerad |
| 9 | Implementera rockad | Implementerad |
| 10 | Implementera passant. | Implementerad |
| 11 | Implementera kröning mot enbart drottning. | Implementerad |
| 12 | Skapa meny så att spelare kan avsluta inifrån spelet. | Icke påbörjad |
| 13 | Lägg till alternativ för timer så att den kan aktiveras genom meny och avslutar om når noll. | Implementerad |
| 14 | Full kröning av bonde för alla typer. | Icke påbörjad |
| 15 | Skapa hightscore för snabbaste vinst. Visa på meny. | Icke påbörjad |

| | | |
|---|---|---|
| **16** | Skapa alternativ för att köra flera matcher på rad. Visa även totala runder och vinster för varje spelare. | Icke påbörjad |
| **17** | Skapa alternativ genom meny så att spelaren kan köra över nätverk. | Icke påbörjad |
| **18** | Skapa AI alternativ genom meny så att spelaren kör mot AI istället för andra spelare. | Icke påbörjad |

## *6.2. Dokumentation för programstruktur, med UML-diagram*

Board-klassen används för att spara positionen på pjäserna och förflytta dem. Klassen utför detta genom att skapa ett 8*8 bräde där varje cell representerar en ruta. InitiateMove-funktionen är den funktion som anropar pjäserna för kontroll av ett givet drag och anropar movePiece ifall draget är lagligt. Då selectedPiece-fältet är tomt kommer funktionen placera pjäsen spelaren klickar på i fältet. När selectedPiece inte är tom kommer programmet tillåta board att förändra värden i pieces och kontrollerar att draget är lagligt innan movePiece-funktionen anropas. I fallet då draget inte hade varit lagligt eller en pjäs var placerad mellan selectedPiece och punkten pjäsen skulle förflyttas till skulle programmet tömt selectedPiece och förbjudit förändringar i pieces. Behovet av en variabel som validerar förändring i pieces kommer från att drag som t.ex rockad där Moves-klassen flyttar på tornet. Eftersom draw-funktionen i board använder Moves-klassen för att måla ut gröna rutor kunde pjäser flyttas utan att användaren klickade. Därav får MovePiece-funktionen inte flytta pjäser vid varje anrop. Utöver rörelse sparar movePiece-funktionen undan pjäsen till Player-klassen för utritning av besegrade pjäser. Sist kontrollerar movePiece “game over”-fältet innan andra spelaren får kontrollera sina pjäser.

Moves-klassen är en hjälp-klass till board som innehåller boolean-funktioner för alla drag som pjäserna kan göra vilket är moveStraight-, moveDiagonal-, moveOne-funktionerna samt moveKing och movePawn. MoveKing använder moveOne samt isKingCastle och isRookCastle som kontrollerar när Kungen respektive Tornet kan göra en rockad. MovePawn är också uppdelad där MovePawn kontrollerar drag framåt och isPawnDiagonal kontrollerar diagonella drag. Slutligen innehåller klassen även isPathClear och isSquareOccupy för att avgöra då andra pjäser står mellan pjäsen och destinationen och när en ruta är möjlig att ockupera.

Pieces-klassen är en abstrakt superklass till alla pjäser i spelet. Klassen innehåller den abstrakta isValidMove-funktionen vilket gör att alla subklasser måste använda override på den. IsValidMove-funktionen kommer anropa de funktioner subklassen behöver från Moves-klassen för rörelse t.ex Rook anropar moveStraight. Ursprungligen skulle rörelsen ligga under varje pjäs men eftersom klasser enbart kan ärva från en klass kunde tornet inte få tillgång till firstMove-fältet och moveStraight-funktionen samtidig eftersom de låg i King- respektive Queen-klassen. Därmed flyttades koden till Moves-klassen.

Viktigt är också get/setFirstMove-funktionerna som inte har någon funktionalitet i Piece-klassen utan används som platshållare för FitstMove-klassen. Anledningen till att detta görs är för att introducera funktionerna till alla pjäser så att typkonroller kan undvikas då t.ex en Pawn gör sitt första drag. Piece-klassen innehåller också funktionerna draw och resize. Draw anropas av board och ritar ut den imageIcon som ges i konstruktorn. Resize anropas en gång av draw och förändrar storleken på pjäsen för att matcha brädets storlek. Anledningen till att resize anropas en gång är för att spelet skulle sakta ner då resize anropades med draw. Slutligen används isPathClear som anropar funktionen av samma namn i Moves-klassen.

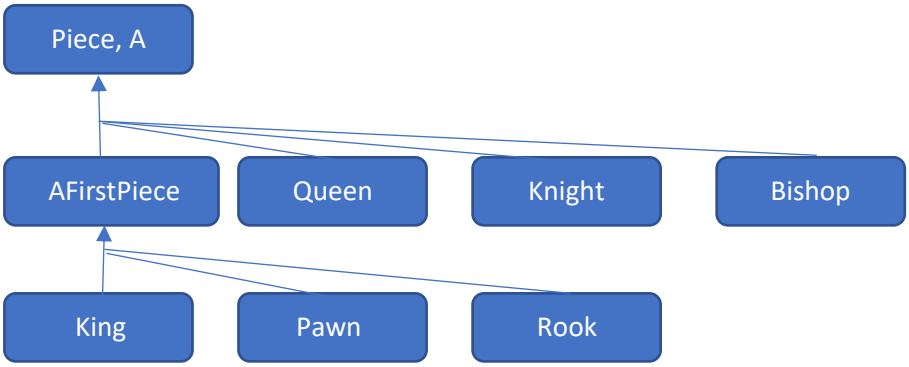



Fig 1. UML-diagram av pieces-hieraki

Piece-klassen har fyra klasser som ärver direkt från den, Bishop, Knight, Queen och FirstMove enligt figur. 1. Bishop och Queen använder override för att anropa moveDiagonal och moveDiagonal samt moveStraight. Knight använder override på validMove och isPathClear där isPathClear-funktionen enbart anropar isSquareOccupy för att tillåta pjäsen att hoppa över ockuperade rutor. Abstrakta FirstPiece-klassen är sist och intorducerar ett firstMove-fält med värdet true och använder override på funktionerna get/setFirstMove så de använder fältet.

FirstPiece-klassen har tre klasser som ärver direkt av den, King, Rook och Pawn. Rook-klasserna, se figur 1. FirstPiece klassen används för att introducera firsMove-fältet. Rook-klassen behöver firstMove variabeln för rockad och använder därför override på enbart isValidMove. Kungen använder override på både isValidMove och resize Resize förändras för att undvika typkontroll för “game over”-fältet vilket också är anledningen till att kungen tar in board i konstruktorn och “game over”-fältet har en set-funktion i board. Genom att använda en funktion som alla klasser redan använder behövs ingen onödig funktion implementeras i de andra klasserna på grund av att kungen. Eftersom resize redan använde en variabel i anropet kunde ett undantagsfall implementeras för kungen. Därmed anropas resize när en pjäs dör vilket stannar när det är en kung. Slutligen använder pawn override på

isValidMove, setFirstMove och getFirstMove för att förändra på twoSquare-fältet. Genom att förändra setFirstMove kommer twoSquare sättas true efter första draget när pawn flyttas två rutor i samband med koden från Moves-klassen. Med nuvarande struktur kan typkontroller undvikas men ett problem som skapdes med implementationen var att en passant kunde ske då en bonde inte rört sig och därav måste raderna kontrolleras i Moves.

Player-klassen skapas i board-konstruktorn och placeras i en array. Klassen innehåller listan med besegrade pjäser och ett tidtagarur. Klassen kan rita ut tidtagaruret och de döda pjäserna samt en funktion som räknar ner tiden. GameTimer-klassen som Player-klassen använder består av isTimerDone-funktionen som räknar ner varje gång den anropas och returnerar true om den når noll vilket spelaren sedan retuneras till tick-funktionen i board som ursprungligen anropade countDown.

För grafiken används ChessViewer-klassen som skapar fönstret där klasserna kan rita ut sig själva. Därefter används ChessComponent-klassen för att rita ut spelet genom att anropa alla draw-funktionr bland de andra klasserna. Genom att använda addBoardListener- och notifyListeners-funktonerna i board kan spelet uppdateras genom ChessComponent eftersom klassen implementerar BoardListener-interfacet och därav funktionen boardChanged.

# 7. Användarmanual

Programmet använder vanliga schackregler exkluderat promovering där bonden enbart kan bytas ut mot en drottning. Användaren får välja att köra med eller utan timer, se figur 2. när programmet startar. Då programmet startar enligt figur 4. visas två tidtagarur på femton minuter för båda spelarna och matchen slutar då något tidtagarur når noll. Alternativet är att spela utan tidtagarur, figur 3. Användaren flyttar pjäser genom att klicka på den önskade pjäsen och sedan en av de gröna rutorna som visas då en pjäs är vald, se figut. 3. För att byta pjäs kan användaren klicka på valfri ruta som inte är grön och sedan den nya pjäsen. Slutligen då användaren gör en rockad måste kungen väljas och inte ett torn.

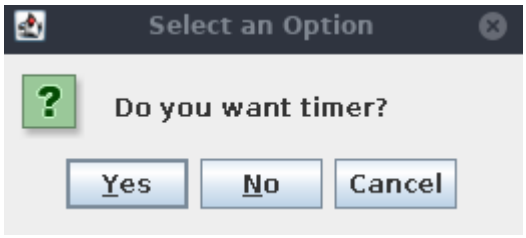



Fig 2. Alternativ för timer eller inte

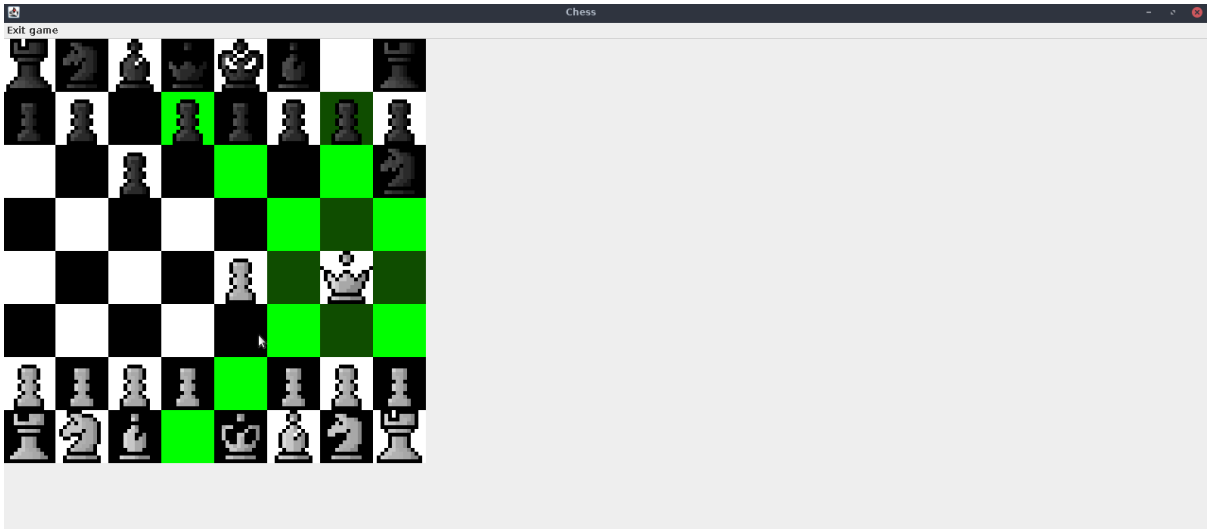



Fig 3. Spel utan timer. Lagliga drag för drottning visade i grön

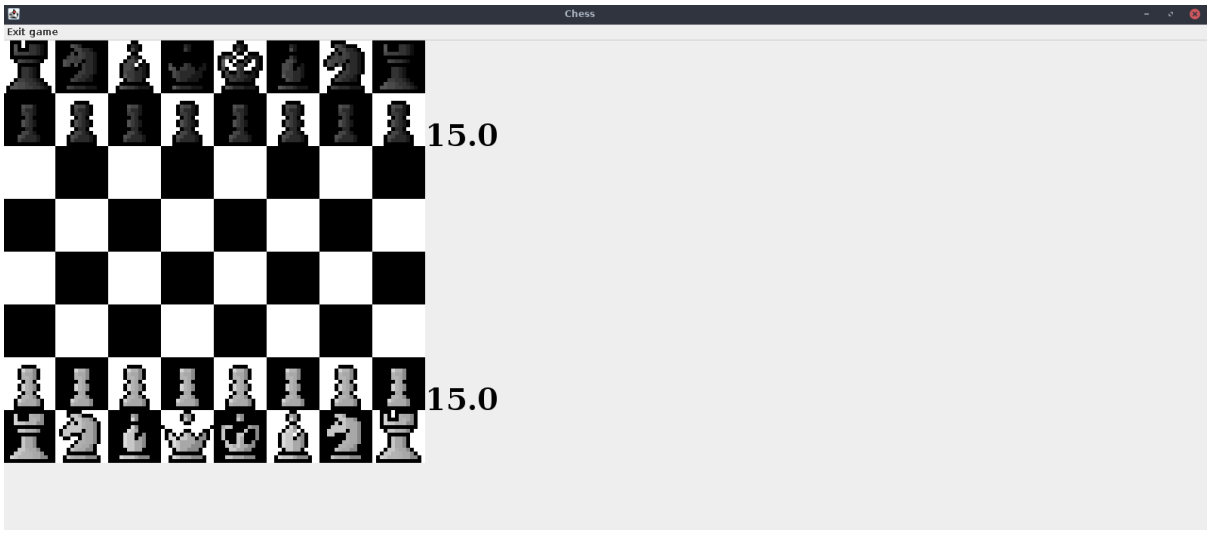



Fig 4. Spel med timer.