Projektbeskrivning

Objektorienterad programmering av schack
2021-03-11

Projektmedlemmar:
Christopher Wåtz <chrwa634@student.liu.se>

Handledare:
Robin Andersson <roban591@student.liu.se>

Innehåll

Projektplan

1. Introduktion till projektet

Målet med detta projekt är att programmera ett schackspel med hjälp av java och att lära sig mer om objektorienterad programmering. Schack är ett fritids- och tävlingsinriktat brädspel som spelas mellan två spelare. Schack är allmänt känt över hela världen och har funnits i åldrar. Det finns redan digitala adaptationer av spelet i flera plattformar. Idag är online-schack populärt på grund av dess tillgänglighet. Syftet med att skapa ett nytt schackprogram är att undersöka de mer obskyra reglerna i schack och tänka ut olika sätt att programmera spelet digitalt.

Projektet fokuserar främst på att använda objektorienterad programmering för att skapa schackprogrammet. Objektorienterad programmering är en designmodell som handlar om att skapa objekt som innehåller både data och metoder. Betydelsen av OOP (objektorienterad programmering) i programmeringsvärlden är enorm och det är anledningen till att försöka lära sig att använda denna designmodell under projektet.

2. Ytterligare bakgrundsinformation

2.1 Schack

Schack spelas mellan två motståndare som förflyttar sina pjäser på ett omväxlande turordning på ett schackbräde. Spelaren med de vita spelpjäserna börjar alltid spelet. Syftet i schack är att skapa en position där motståndarens kung är ”under attack” på ett sådant sätt att motståndaren inte har ett giltigt drag för att försvara kungen eller flytta kungen ifrån attacken. Spelaren som har uppnått målet sägs ha schackmatad motståndaren och har vunnit spelet. Att lämna sin kung under attack, utsätta den för en attack genom att flytta en annan pjäs och även ”döda” motståndarens kung är inte tillåten. Om en position uppstår där båda spelarna inte kan schackmatta motståndarens kung anses partit som oavgjort [1].

Schackbrädet består av ett 8 x 8 rutnät med 64 lika rutor växelvis ljusa och mörka. Brädet är placerat på ett sätt där kvadraten som är i den högra hörnen för varje spelare är vitt. De åtta vertikala kolumnerna med rutor kallas filer och de åtta horisontella raderna av rutor kallas rader. En linje med rutor i samma färg som går från en kant till en intilliggande kant kallas för en diagonal [1].

Figur 1:
Bilden visar ett schackspel i dess starttillstånd

I början av spelet finns 16 ljusa och 16 mörka schackpjäser på brädet. Det finns 6 olika typer av pjäser och de är kung, dam, löpare, torn, springare (eller häst) och bonde. Det mest signifikanta skillnaderna mellan dessa typer av pjäser är deras rörelse på schackbrädet. Figur 1 visar startpositionen för ett schackspel [1].

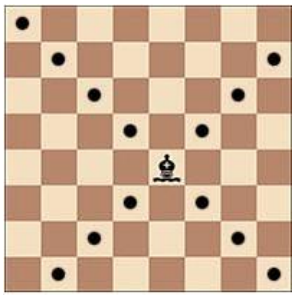
Det finns några allmänna lagar som styr rörelserna på schackpjäserna. En av de är att det inte är tillåtet att flytta en schackpjäs till en ruta som är redan upptagen av en pjäs av samma färg. En pjäs sägs attackera en motståndarens pjäs om pjäsen kan förflytta sig till rutan där motståndarens pjäs för närvarande befinner sig [1].

Löparen kan röra sig till rutor längs en diagonal som den står på. Figur 2 visar hur en löpare kan röra sig på en schackbräda [1].

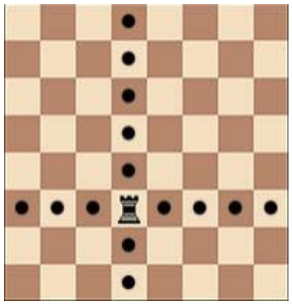
Tornet kan förflytta sig till kvadrater utmed en horisontell linje där den är satt. Figur 3 illustrerar hur en löpare kan förflytta sig [1].

Damen kan omplacera sig som en kombination mellan tornet och löpares, det vill säga både horisontell och diagonal. Figur 4 målar upp vilka rutor som en drottning kan förflytta sig till [1].

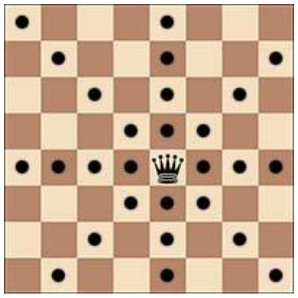
Alla tre pjäserna löparen, tornet och damen får inte flytta sig över någon annan schackpjäs [1].



Figur 2:
Bilden illustrerar de möjliga dragen en löpare kan utföra. De svarta punkterna visar rutorna som löparen kan förflytta sig till.



Figur 3:
Bilden visar hur ett torn kan förflytta sig. De markerade rutorna är var tornet kan förflytta sig till.



Figur 4:
Bilden målar upp hur en dam kan röra sig på ett schackbräde. De svarta markerade rutorna är de möjliga dragen som drottningen kan utföra.

Hästen eller också kallad springaren kan flytta till en av rutorna närmast den som är inte i samma rang, fil eller diagonal. Hästen kan också hoppa över andra pjäser. Figur 5 illustrerar hästen möjliga drag [1].

Figur 5:
Bilden visar de möjliga dragen som en häst kan utföra. Den visar också att det kan hoppa över pjäser på den övre högra hörnen. De punkterade rutorna är de som pjäsen kan röra sig till.

Bonden kan gå framåt till den obesatta rutan direkt framför den på samma fil. Den kan också förflytta sig två steg framåt om den har inte gjort ett drag. Horisontella drag av en bonde kan inte fånga en annan pjäs. Bonden kan röra sig diagonalt om det finns en motståndare pjäs på rutan och om bonden rör sig till det rutan fångar den motståndarens pjäs. Figur 6 visar hur en bonde kan röra sig. De runda markeringarna visar att bonden kan röra sig ditt om det inte finns någon pjäs i väggen. De med kryss är rutor där bonden kan endast röra sig om det finns ett motståndarens pjäs [1].

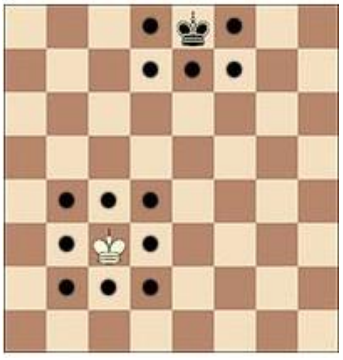
En bonde som attackerar en ruta som en av motståndarens bonde förflyttar sig över efter att ha avancerat två steg, kan fånga denna bonde som om den hade bara flyttats ett steg. Denna fångst är endast laglig på dragen efter detta framsteg och kallas för en passant. Figur 7 illustrerar detta drag. Det som syns i bilden är att om den svarta bonden förflyttar sig till den svarta punkten kommer den vita bonden ha möjligheten att röra sig till krysset och fånga den svarta bonden vid nästa drag [1].

Om en bonde når raden närmast motståndaren måste den vara utbytt av ett torn, löpare, häst eller dam av samma färg. Spelarens val är inte begränsat till bitar som har fångats tidigare [1].

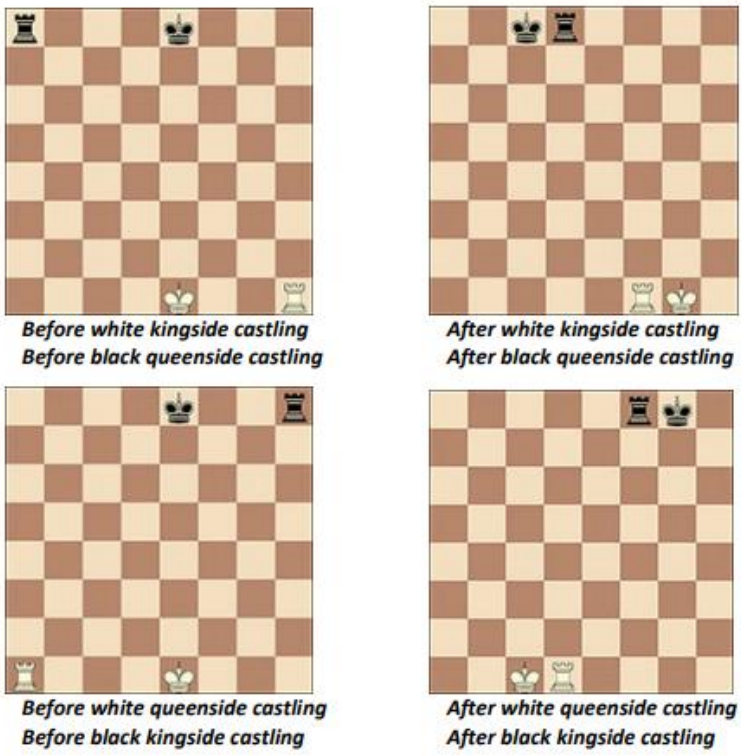
Figur 6:
Bilden illustrerar hur en bonde kan röra på sig.

Figur 7:
Bilden visar en passant draget.

Det finns två olika sätt att förflytta kungen. Det första är att genom att flytta till någon angränsande kvadrat som är inte attackerad av en motståndare pjäs. Denna rörelse är illustrerad på figur 8 där de markerade rutorna är rutor som kungen kan förflytta sig till. Det andra draget kallas för rockad. Detta är ett drag av kungen som också förflyttar ett torn med samma färg. Draget räknas som ett drag även om två pjäser förflyttar sig. Draget utförs genom att förflytta kungen två steg till antingen höger eller vänster och rutan som kungen förflytta sig över är rutan där tornet ska placeras. Rockad är endast giltig när kungen och tornet inte har förflyttat sig under matchens gång. Kvadraten som kungen kommer att flytta till och kvadraten som kungen passerar får inte vara attackerad. Det får hellre inte finnas några pjäser mellan kungen och tornet. Kungen får inte göra rockad om den är i schack. Kungen sägs vara ”i schack” om den attackeras av en eller flera av motståndarens bitar. Figur 9 visar de olika sättet kungen kan göra rockad på. Bilderna till vänster är innan de har gjort rockad och de till höger är efter att de har utfört draget [1].



Figur 8:
Bilden illustrerar hur kungen kan förflytta sig på schackbrädet



Figur 9:
Bilden visar de olika sättet kungen kan utföra ett rockad.

2.2 Forsyth-Edwards Notation

Forsyth-Edwars Notation eller förkortad FEN beskriver en schackposition genom en ASCII-sträng. FEN är baserat på ett system tillverkad av David Forsyth på 1800-talet. Steven Edwards specificerade senare FEN-standarden för datorschackapplikationer som en del av Portable Game Notation (PGN). En fen-sträng består av sex fält åtskilda av ett mellanslagstecken [2].

2.2.1 Pjäsplacering

Pjäsplaceringen bestäms utefter varje vertikal rad från den åttonde raden till det första. Varje rad är separerad av en ‘/’ symbol. För varje rad är den placerade från filen A till H. En decimalsiffra räknar tomma rutor i följd. Pjäserna identifieras av en enda bokstav från standard engelska namn för schackpjäser. Versaler är för det vita pjäserna medan gemener för de svarta pjäserna [2].

2.2.2 Aktivspelare

Det aktiva spelare bestäms med en bokstav. Det är ett ‘w’ om det är den vita spelarens tur på positionen medan ‘b’ om det är den svarta spelarens drag.

2.2.3 Rockadrättigheter

Nästa fält består av rockadrättigheter. Om ingen kan utföra ett rockad drag ska det

symboliseras med ett ‘-’. Om det går att genomföra rockad på kungens sida ska det symboliseras med tecknet ‘k’. Om det fungerar att utföra rockad drottningens sida ska detta betecknas med ‘q’. Versaler om det är den vita pjäserna annars är det gemener.

2.2.4 En passant ruta

I det här fältet ska det innehålla information om rutan som en bonde kan utföra en ”en passant” drag. Vid tillfälle där det inte finns bönder som kan utföra en passant vid den specifika rutan ska rutan även anges. Rutan ska vara givet i vanliga schackannotationer det vill säga en bokstav som beskriver den vågrättaraden medan ett nummer som beskriver det lodrätta raden.

2.2.5 ”Halfmove Clock”

“Halfmove Clock” är sätt att räkna för att bevaka för femtio drags regeln. Femtio drags regeln säger att ett schackspel anses vara oavgjort efter femtio på varandra följande drag utan någon fångst eller att någon bonde har förflyttats. Denna räknare återställs efter att en pjäs har fångat en annan pjäs eller att en spelare förflyttar en bonde. Om andra drag utförs kommer räknaren att räkna upp.

2.2.6 ”Fullmove counter”

”Fullmove counter” är en räknare som räknar antalet hela drag i ett spel. Den börjar 1 i början av spelet och räknar upp efter varje gång den svarta spelaren utför ett drag.

2.2.7 Exempel

Här visas några positioner i schack och deras fen-strängar.

```
FEN: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

```
FEN: rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq e3 0 1
```

```
FEN: rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR w KQkq c6 0 2
```

```
FEN: rnbqkbnr/pp1ppppp/8/2p5/4P3/5N2/PPPP1PPP/RNBQKB1R b KQkq - 1 2
```

3. Milstolpar

Projektet kommer vara indelad i milstolpar för att enklare vara medveten om de olika komponenterna som behövs utveckla och ordningen de ska vara implementerade. Programmeringen delades upp i mindre delar för att göra det enklare att fokusera på ett delmoment. Efter att man har genomfört alla uppgifterna i en milstolpe ska det testas för att vara övertygad att allt fungerar. De milstolparna som placeras efter att programmet anses vara funktionell existerar för att utföra om såvida extra tid finns på slutet.

#	Beskrivning
1	Det finns klasser för spelplan och ”kvadrater”. Spelplan ska ha en matris med kvadrater. Det kommer att finnas ett fält som heter isWhitesTurn som håller koll på vems tur det är. Det finns en grafisk komponent som kan visa tavlan.
2	Implementera ett ”Piece” abstraktklass som innehåller funktionen ”getValidMove” som returnerar alla möjliga drag som en pjäs kan utföra. Klassen ska innehålla fält för färg, bild, pjäs typ, och rutan som den är placerade på.
3	Implementera en klass ”players” om sparar spelarens namn och färg.
4	Implementera en klass ”pawn” som förlänger klassen ”piece”. Konstruktören ska ta emot en färg och URL till en bild. Metoden ”getValidMove” ska kunna ge tillbaka alla drag som bonden kan utföra. Klassen ska också ha ett ”promotePawn” funktion som ”promota” bonden till en annan pjäs. Metoden ska fråga användaren vad de vill ”promota” bonden till. Tillverka en test att det blir mycket enklare att testa de här funktionerna.
5	Implementera en klass ”Bishop” som förlänger abstraktklassen ”Piece”. Konstruktören ska ta emot en färg och en URL till en bild. Skapa någon klass som kan ”skanna” schackbrädet. Denna klass ska kunna returnera alla diagonala rutor från en specifik ruta. Det kommer att göra att ”getValidMoves” metoden i Bishop behöver endast fråga scannern om alla diagonala drag. Gör olika tester som testar dessa funktionalitet.
6	Implementera en klass ”Rook” som förlänger abstraktklassen ”Piece”. Konstruktören ska ta emot en färg och en URL till en bild. Lägg till metoden i scannern för att ge tillbaka alla horisontella rutor från en angiven ruta. Metoden ”getValidMoves” i ”Rook” ska då endast behöva kalla metoden i scannern för att returnera alla möjliga drag. Det behövs inte att implementera rockad ännu därför att klassen King måste också vara klart. Gör olika tester som testar dessa funktionalitet.

7	Implementera en klass “Queen” som förlänger abstraktklassen ”Piece”. Konstruktören ska ta emot en färg och en URL till en bild. Använd de diagonal och horisontell metod i scannern för att hitta alla möjliga drag. Tillverka olika tester som testar dessa funktioner.
8	Implementera en klass ”Knight” som förlänger abstraktklassen ”Piece”. Konstruktören ska ta emot en färg och en URL till en bild. Implementera de olika dragen som en häst kan göra. Gör olika tester som testar dessa funktionalitet.
9	Implementera en klass ”Kung” som förlänger abstraktklassen ”Piece”. Konstruktören ska ta emot en färg och en URL till en bild. Implementera is ”validMove” för kungen. Den får förflytta sig till attackerade rutor vid detta stadium.
10	Implementera en klass som kan ladda positioner på brädet med hjälp av FEN schackannotationer. FEN kommer att anges som en sträng. Denna klass ska också kunna ge tillbaka FEN position av en tavla.
11	Implementera ett map i board klassen där nyckeln är ett enum ”CastlingRights” som håller koll på vilka rockad drag som är möjliga. Implementera metoden ”updateCastlingRight” i klasserna ”King” och ”Rook” som uppdaterar mappen om dessa pjäser rör på sig.
12	Implementera metoden ”movePiece” i schackbrädet som kan röra pjäserna. Denna metod kan också hantera en passant och rockad. Lägg till kod i kung klassen som kan ge tillbaka de möjliga rockad dragen som kan utföra.
13	I scannern skriv en metod som kan undersöka om en ruta är attackerad av motståndarens pjäs.
14	Skriv en ”removeUnvalidMoves” metod i scannern som får möjliga drag som en pjäs kan utföra och ta bort de dragen som leder till att kungen ligger i schack.
15	Skriv en metod för att kontrollera oavgjort på grund av att en spelare har inga möjliga drag. Denna metod ska kallas för ”isStalemateNoValidMoves” som returnerar true om det är sant.
16	Skriv en metod som uppdaterar tillståndet på tavlan. Den ska inspektera om kungarna ligger i schack. Den ska undersöka om det är oavgjort (endast för inga möjliga drag just nu). Lägg till hantering för schackmatt genom att säga om kungen ligger i schack och

	”isStalemateNoValidMoves” metoden returnerar true är det schackmatt.
17	Skriv en metod som utreder om det är oavgjort på grund av att det inte finns tillräcklig många pjäser för både spelarna att schackmatta kungen.
18	Implementera en metod som kontrollerar om det blir oavgjort på grund av att en position har förekommit tre gånger under matchens gång.
19	Lägg till en metod som håller koll på 50 drags regeln.
20	Lägg till GUI som talar om på vilket sätt spelet är över. **Nu Anses spelet vara spelbart**
21	Lägg till en knapp för att ta tillbaka drag.
22	Lägga ett fönster på sidan av brädet som visar de olika dragen som spelarna har gjort. (Och att kunna trycka på de och brädet se hur det såg ut efter det draget.
23	Höger klicka och dra om man vill rita ett pill för att rita vad man vill göra nästa drag. Pilen gör inte något bara visualisera för spelaren.
24	Lägga till en meny för att ändra inställningar i spelet.

4. Övriga implementationsförberedelser

Klassen för spelbrädet har tänkt vara en klass som håller ihop alla andra klasser i anledningen av att de kan arbeta med varandra. Det som menas är att om en löpare vill veta alla diagonala drag då måste den be tavlan om detta och tavlan kommer utefter det fråga scannern om alla diagonala drag. Detta sätt att implementera valdes på grund av strukturen som blir mindre rörigt därför att alla klasser är endast beroende på tavlan och inte varandra. Tavlan har tänkt att också behöva hålla koll på rutorna och den ska också kunna flytta pjäser till olika rutor.

Scannern är en klass som ska ha uppgiften att som kan ge tillbaka information som man vill ha ifrån tavlan. Detta är för exempel alla de diagonala rutorna ifrån en angiven ruta. Den ska också skanna tavlan efter varje drag för att undersöka om det finns villkor som leder till ett avslutat parti.

Pjäserna ska för det mesta hålla koll på var den kan röra på sig. De ska också veta vilken färg de har och hur de ritar upp sig själv på tavlan.

Konverteraren ska ha uppgift att ladda in positionen på tavlan och ge tillbaka fenannotationer av en spelplan. Den ska ha ett objekt som kallas för ”PieceFactory” som har uppgifter att kunna skapa pjäser. Tavlan har inte en ”PieceFactory”-objekt därför att alla andra klasser inte behöver skapa nya pjäser.

5. Utveckling och samarbete

Författaren har tänkt att arbeta med projektet under laborationstiderna. Det kommer arbetas under författarens fritid förutsatt att det kräver extra tid för att utföra uppgifter som ska genomföras. Det kan också vara lämpligt att nämnas att författaren kommer

att fortsätta med projektet ifall att hen inte är nöjd med betyget. Författaren har tänkt projektet ska vara minst en fyra i betyg.

Projektet ska försöka ha ett fungerande schackspel innan den preliminära rapportinlämningen. En vecka innan slutinlämningen ska det vara en ”feature freeze”, vilket betyder en tid där inga nya funktioner får implementeras. Under denna vecka försöker författaren att hitta fel, fixa till kommentarerna på koden och läsa igenom rapporten.

Författaren tycker att detta projekt kommer att vara spännande därför att hen är insatt i både programmering och schack. Det finns delar som är oklara för författaren i hur hen ska genomföra de olika funktionerna. Som helhet längtar författaren att på börja projektet.

Projektrapport

Projektrapporten ska innehålla en implementationsbeskrivning där författaren framför sina tankar om hur utvecklingen har gått, hur programmet är implementerad och varför den implementationen har valts. Det behandlar också användarmanualen där författaren skriver användningen av programmet.

6. Implementationsbeskrivningen

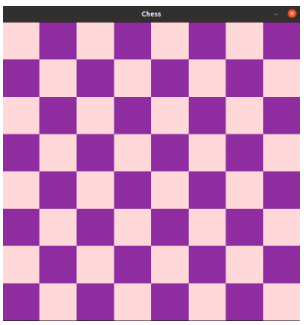
Implementationsbeskrivningen ska vara uppdelat i två delar. I det första delen ska författaren gå igenom varje milstolpe. Detta utförs för att framföra tankar i mer detalj om en specifik komponent. I den andra delen kommer författaren skriva mer allmänt om funktionaliteten i programmet. Det ska innehålla tankar som är lite mer allmänt som inte alltid passar i koden som kommentarer eller Javadoc.

6.1. Milstolpar

6.1.1 Milstolpe 1

Det skapades en klass som heter ”Board” som har en matris av rutor. Rutorna är representerad av klassen “Square” som har information om vilket färg den har och var på tavlan den är placerad. Klassen ”Square” har också metoder som kan rita upp sig själv på tavlan.

Det finns en klass som skapar ett ”JFrame” som visar spelet. I det här ”JFrame” objektet kommer det i framtiden kunna lägga flera ”JPanel” objekt som visar olika komponenter i programmet. Det finns en ”JPanel” som heter ”GamePanel” som är en container för spelplanen. ”JFrame” objektet innehåller nuvarande endast GamePanel för att visa tavlan. Figur 10 visar tavlans utseende.



Figur 10:
Bilden illustrerar schackbrädet som ritas upp av programmet

6.1.2 Milstolpe 2

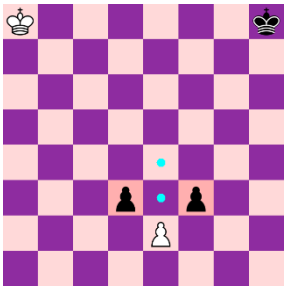
I denna milstolpe fanns inte något bekymmer och alla uppgifter är utförda. Det som är mindre bra är dock att det inte går att testa funktionaliteten i denna milstolpe på grund av att det inte är möjligt att skapa en abstraktklass.

6.1.3 Milstolpe 3

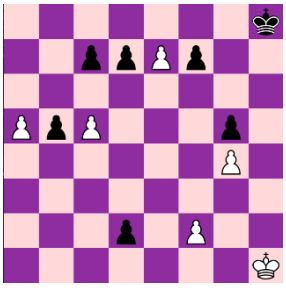
Denna milstolpe utfördes inte därför att författaren tänkte att spelare-klassen är onödig för att få ett fungerande schackspel, men i framtiden kan detta vara implementerad för att för exempel kunna spela över nätet.

6.1.4 Milstolpe 4

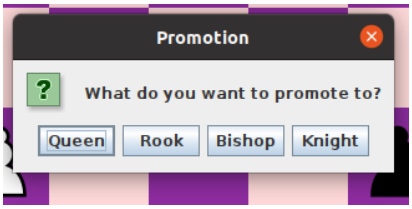
Alla uppgifter i denna milstolpe gjordes klart. Alla uppgifter är klara och med testet som skapades fungerar alla funktioner som de ska. Figur 11 demonstrerar att bonden kan röra sig som den ska. Figur 12 visar hur testet ser ut. Den är konstruerad på ett sätt som gör det möjligt att testa alla funktioner på bonden. Figur 13 demonstrerar att den kan ”promota” till andra pjäser.



Figur 11:
Bilden visar att bonden kan röra sig två steg och fånga fiende pjäser diagonalt.



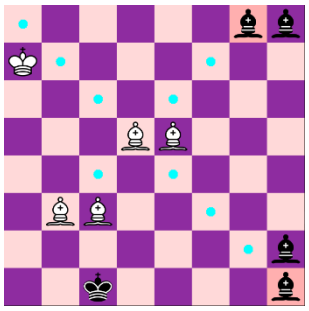
Figur 12:
Bilden illustrerar testet för bönder.



Figur 13:
Bilden visar att bonden kan ”promota” till andra pjäser.

6.1.5 Milstolpe 5

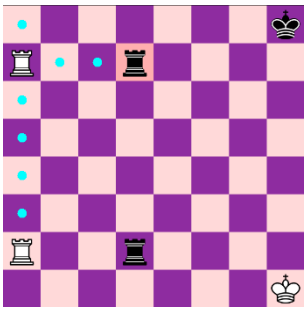
Det var inga konstigheter med denna milstolpe. Klassen gjordes och därefter en metod implementerades för att hitta alla diagonala rutor från en ruta som heter ”getDiagonalMoves”. Det enda ”getValidMoves” metoden gör i Bishop klassen är att bara anropa på ”getDiagonalMoves”-metoden på rutan den står på och få tillbaka alla möjliga drag. Det är inga konstigheter med testet då löparen har inga speciella regler som för exempel bonden. Det man kan göra i testet är att flytta runt löparna. Figur 14 visar att löparens funktionalitet fungerar och dess testposition.



Figur 14:
Bilden visar testpositionen för löparen.

6.1.6 Milstolpe 6

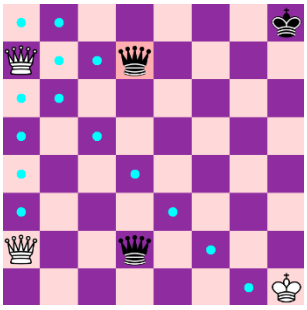
Det var inga svårigheter med denna milstolpe på grund av att metoden som kan undersöka vid ett specifikt håll är redan implementerad och genom att ändra riktningen kunde horisontella rutor fås. Testet är hellre inte speciellt konstruerad på grund av att författaren tänker att rockad går att testa senare med kungen. Figur 15 demonstrerar att tornet fungerar och testets utseende.



Figur 15:
Bilden illustrerar testpositionen för tornet.

6.1.7 Milstolpe 7

Det är inte mycket att prata om i implementationen av damen då författaren kombinerade bara ”Rook” och ”Bishop” klassen ”getValidMoves” för att få alla damens möjliga drag. Det var hellre inga konstigheter med testet på grund av att drottningen har inga konstiga regler. Figur 16 illustrerar att funktionaliteten på damen fungerar och visar testet som används för damen.

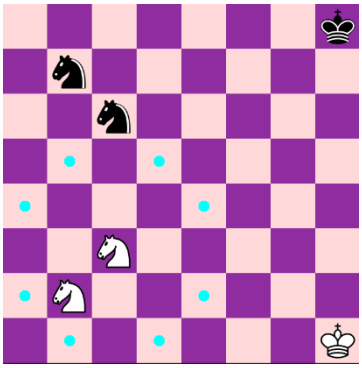


Figur 16:
Bilden demonstrerar testpositionen för damen.

6.1.8 Milstolpe 8

I denna milstolpe implementerades metoden ”getSquareMoves” som kunde undersöka om en ruta var antigen ledig eller ockuperad av en fiendepjäs. Om metoden returnerade true anses att pjäsen kunde förflytta sig till kvadraten.

I början kallades denna metod åtta gånger för de olika rutor som hästen kan förflytta sig till, men senare kom författaren på en algoritm för att kalla dessa rutor med bara en for-loop. For-loopen är lite mindre effektiv än att skriva alla åtta, men det bryter inte mot DRY (Dont Repeat Yourself) regeln i programmering.

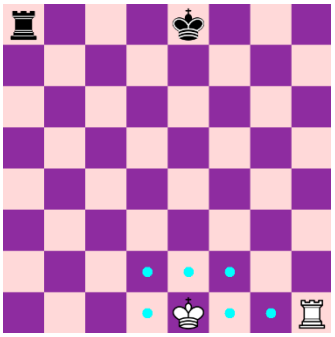


Figur 17:
Bilden demonstrerar testpositionen för hästen.

Det finns ingenting speciellt i testet därför att hästen har inga speciella regler som gör att den är endast konstruerad för att låta pjäserna röra sig fritt. Figur 17 visar testet och att hästen ”getValidMoves” fungerar

6.1.9 Milstolpe 9

Kungens ”getValidMoves” implementerades genom att använda metoden ”getSquareMoves” för alla rutor som är ett steg bort. Detta implementerades genom att använda en for-loop som kontrollerar alla rutor ett steg ifrån rutan som kungen befinner sig på. Under det här stadiet kan kungen förflytta sig till rutor som är attackerad och att den inte vet om den ligger i schack.



Figur 18:
Bilden illustrerar testpositionen för kungen.

Testet som gjordes är lite speciellt därför att i framtiden ska rockad testas och därför har författaren valt att konstruera testet som det ser ut på figur 18. Tanken med konstruktionen är att det kommer att inte behöva skapa ett till test för att testa rockad.

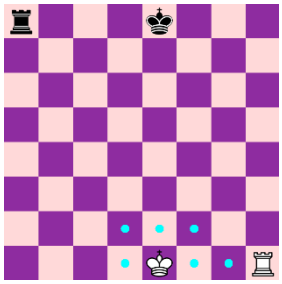
6.1.10 Milstolpe 10

I det här milstolpe skapades klassen ”FenConverter” som har uppgiften att ladda in en position på tavlan enligt fen strängen som anges. Klassen har också uppgiften att skapa fen strängen av en angivet ”Board”-objekt. Det var inga konstigheter med denna milstolpe. Klassen testades genom att ladda in positioner på tavlan och skriver ut fen strängar av tavlan efter varje drag.

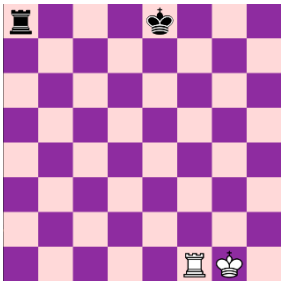
6.1.11 Milstolpe 11

Det var lätt att implementera ett map för rockadrättigheter. Det var också ganska enkel att lägga till metoden för att uppdatera rättigheterna för kungen. Dock uppstod några problem vid tornet därför att metoden hade ett fel. Det var att om en pjäs hade tagit tornet innan det tornet hade förflyttat sig uppdaterades inte rockadrättigheterna. Det var på grund av att det uppdaterades endast om ett torn hade flyttat sig ifrån ett av de fyra hörnen på schackbrädet. Nu är problemet löst genom att undersöka om en allmän pjäs förflyttar sig till eller ifrån ett hörn ska den motsvarande rockadrättigheter sättas till false.

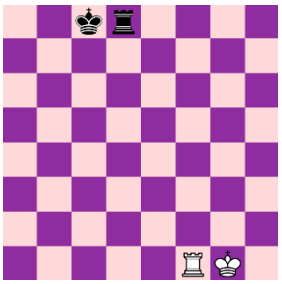
Figurerna 19, 20 och 21 visar att programmet nu kan utföra rockad.



Figur 19:
Bilden illustrerar innan en rockad.



Figur 20:
Bilden visar positionen efter att vit gjorde rockad



Figur 21:
Bilden visar schackbrädet efter att svart gjorde rockad

6.1.12 Milstolpe 12

Det som gjordes var att det skapades en klass som heter ”Move” där information kunde paketeras. Det som ”Move” sparade var startrutan och slutrutan. Genom att skicka in ett ”Move”-objekt till metoden ”movePiece” kunde en pjäs röra sig på tavlan.

Det finns några if-satser i funktionen därför att det finns speciella fall som måste undersökas. Det första är att om det är en bonde ska metoden utreda om det är en ”en passant” drag. Det som är speciell med ”en passant” är att det är det enda dragen som fångar en annan pjäs utan att förflytta sig till det rutan som den fångade pjäsen befinner sig i. Detta är ett problem på grund av att den allmänna fångkoden fungerar endast om pjäsen förflyttar sig till rutan där fiendens pjäs befinner sig på. Det andra som är lite speciellt med bonden är att ”enPassantSquare”-fältet måste sättas varje gång en bonde förflyttar sig.

Det andra speciella fallen är kungen därför att pjäsen kan utföra rockad. Detta hanteras genom att köra en förflyttningskod för den speciella fallen av rockad. Koden aktiveras ifall att kungen ska förflytta sig två steg till vänster eller höger eftersom det betyder att kungen ska utföra rockad. I det här steget har inte metoden ”isSquareAttacked” implementerats och därför kan kungen utföra rockad även om rutorna är attackerade.

6.1.13 Milstolpe 13

Det förekommer inte några problem med implementation av denna metod. Metoden fungerar genom att undersöka alla riktningar. Om en fiendepjäs hittades på en riktning, utreda om pjäsen kan förflytta sig till rutan som undersöks, om den kan är rutan attackerad annars försätt att kontrollera de andra riktningarna. Om det inte finns några fiender i de olika riktningarna ta reda på om det finns hästar som attackerar rutan, om det inte finns är rutan inte attackerad.

6.1.14 Milstolpe 14

Denna metod var svårt att implementera därför det förekommer flera fel under implementationen. Metoden skulle radera drag som var ogiltiga på grund av att kungen ligger i schack. Problemet var att när metoden körs kommer den undersöka om kungen inte är attackerad genom att kontrollera om rutan den står på är attackerad. Problemet var att ”isSquareAttacked” funktionen anropade på ”getValidMoves” metoden för en fiendepjäs vilket leder till en evigloop där båda två hela tiden kallar på varandra. Lösningen som implementerades är att skicka ett boolean värde som utför metoden ”removeUnvalidMoves” endast om den är sant. Detta gjordes på grund av att i metodenen ”isSquareAttacked” behövs bara de möjliga dragen som en pjäs kan utföra

vilket betyder att man inte behöver radera dragen som är ogiltiga.

Funktionen testades genom att ladda in positioner där en schackpjäs inte borde kunna förflytta sig på grund av att den skyddar kungen. Det fanns också positioner där kungen ligger i schack och en pjäs hade alternativet att blockera attacken. Det sista är att utreda om kungen kan förflytta sig in i schack.

6.1.15 Milstolpe 15

Denna metod implementerades och det fungerar som den ska. Det som görs är att den kallar hjälpmetoden ”hasValidMove”. ”hasValidMove” itererar över alla pjäser som tillhör spelaren och kallar deras ”getValidMove” metod. Om det metoden ”getValidMove” returnerar minst ett drag retunerar metoden ”hasValidMove” true. Om metoden har itererat över alla pjäser och inga pjäser har returnerat ett drag kommer hasValidMove att returnera false.

6.1.16 Milstolpe 16

Denna milstolpe var relativt litet därför att uppdateringsmetoden endast kallade på andra metoder för att utföra sin uppgift. Det var enkel att implementera schackmatt därför att det behövdes bara kombinera koden för ”hasValidMove” och ”isChecked”.

6.1.17 Milstolpe 17

Metoden ”isInsufficientPieces” realiserades genom att undersöka spelarens pjäser. Den får tag i alla pjäser som en spelare har från ”Board” klassen. Med listan av båda spelarens pjäsers använder metoden en hjälpmetoden ”isInsufficientMaterial”. Om metoden returnerar true för både listorna är positionen oavgjort på grund av att båda spelarna inte kan schackmatta motståndarens kung.

Metoden ”isInsufficientMaterial” implementerades genom att använda listan av alla möjliga pjäser för att undersöka om båda spelarna inte kan utföra schackmatt. Om någon av spelarna har mindre än två pjäser, undersökt ifall att en av dessa pjäser returnerar true när man anropar deras ”canDeliverMateAlone” metod. Om en pjäs returnerar true kommer metoden ”isInsufficientMaterial” att returnera false. Om alla pjäser returnerar false kommer metoden att returnera true. Om en spelare har mer än 2 pjäser, kommer metoden att kontrollera att det finns en annan pjäs som inte är en löpare på en ruta av samma färg eller en kung. Om det inte finns är det oavgjort därför att det inte går att schackmatta motståndaren med endast en kung och flera löpare på lika färgade rutor.

6.1.18 Milstolpe 18

Metoden som implementerades heter ”isThreeFoldRepetition” och den använder sig av ”PositionCounter”. ”PositionCounter” är en klass som har uppgiften att räkna olika saker som behövs under spelet. Den har uppgiften att räkna hur många gånger en position har inträffat i ett spel. Den gör detta genom att använda positionsdelen av fen-strängen som en nyckel i en map av ”integers”. Om nyckel inte existerar (dvs att position aldrig har inträffat i spelet) kommer den att skapa en nyckel med positionen med värdet ett. Om den har redan inträffat räknar den i stället ut det nuvarande värdet med ett. Om en pjäs fångar en annan pjäs ska programmet rensa mappet därför att alla föregående positioner kan inte längre inträffa därför att det finns en mindre pjäs på tavlan. Mappen kommer också tömmas när en bonde förflyttas och att kungen utför rockad.
Det ”isThreeFoldRepetion” gör är att söka igenom mappen ifall att det finns ett värde som högre än två. Om det finns ett värde som är högre än två anses spelet vara oavgjort.

6.1.19 Milstolpe 19

Metoden som implementerades heter ”isFiftyMoveDraw” och den använder sig också av en räknare i klassen ”BoardCounter”. I ”BoardCounter” finns det en räknare som räknar antalet drag som har pågått där inga bönder har rört sig eller ett fång. ”isFiftyMoveDraw” metoden utreder ifall att denna räknare är mer eller lika med femtio Om det är större än femtio kommer metoden returnera true annars är det false.

6.1.20 Milstolpe 20

Det gjordes en GUI som heter ”GameOverFrame” som visar ett fönster där användaren kan antigen välja att spela ingen eller att avsluta spelet. När användaren väljer att spela igen kommer tavlan att laddas med startposition i schack. Nu anses spelet vara spelbart.

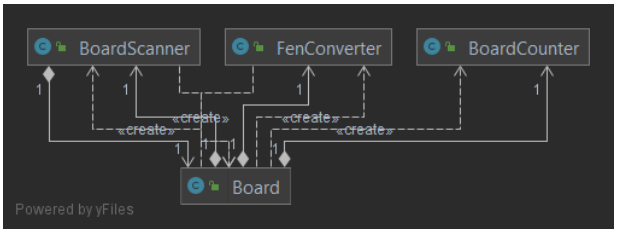
6.2. Dokumentation för programstruktur, med UML-diagram

6.2.1 Board

Denna klass representerar tavlan i schackprogrammet. Board innehåller en matris med rutor som ett sätt att representera tavlan. Matrisen är uppbyggd av åtta matriser med rutor där varje matris representerar en rad som innehåller åtta rutor. Denna struktur valdes på grund av att författaren tänker att det blir lättare att få tillbaka rätt ruta när den efterfrågas. Genom att först ange raden och sedan filen som rutan är placerad på kan rutan returneras snabbt.

Denna klass har uppgiften att spara tavlan och att utföra ändringar på tavlan. Det är därför denna klass har metoden för att förflytta pjäserna. Den har också uppgiften att kalla en uppdatering av tillstånd på tavlan efter varje drag.

Det kan se ut som att klassen har andra uppgifter än de som är nämnda ovan på grund av att funktioner som ”removeInvalidMove” förekommer, men detta är bara en komposition vilket betyder att den skickar vidare uppgiften till ett annat objekt. Detta sätt att designa ett program kallas för komposition och är fundamental i det objektorienterade designmodellen. Det finns många funktioner i board, men de flesta av funktionerna skickar endast vidare uppgiften till ett annat objekt. Komposition valdes på grund av att det minskar sannolikheten att koden blir spagettikod där alla klasser är beroende på varandra. Figur 22 visar att ”BoardCounter” är inte relaterad till ”BoardScanner”. Det är på grund av att om ”BoardScanner” behöver ha tillgång till fält eller metoder som är i ”BoardCounter” ber den istället board om att anropa funktionen i ”BoardCounter”. Koden är designat på detta sätt därför att det blir mycket enklare att förändra metoder i klassen som för exempel deras namn därför att anropet inträffar bara i klassen ”Board” och inte i flera klasser.



Figur 22:
Bilden visar ett UML diagram som visar komposition.

6.2.2 Square

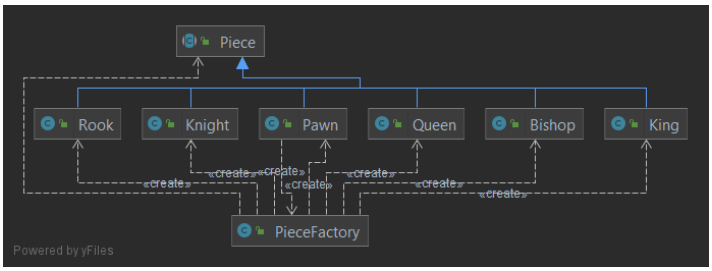
Klassen representerar en ruta på schackbrädet. Den har uppgiften att spara pjäsen som ockuperar rutan. Ifall att det inte finns en pjäs som står på rutan ska ”piece”-fältet vara null. Rutan håller reda på hur den ska rita av sig själv på tavlan. Om rutan finns i listan ”selectedSquares” i klassen ”BoardScanner”, ska den rita av sig själv med en rund blå cirkel i mitten. Detta representerar att pjäsen som användaren funderar på att förflytta kan flyttas till rutan. Om rutan är ockuperad av en pjäs och att den är i listan ”selectedSqures”, ritas rutan röd för att representera för användaren att hen kan fånga pjäsen som står på rutan.

6.2.3 FenCoverter

Klassen har uppgiften att ge tillbaka fen-strängen av ett givet schackbräde som den gör genom att anropa funktionen ”convertBoardToFen”. Metoden kan delas upp i små uppgifter som är att ladda in placering av pjäser, aktiv spelare, rockadrättigheter, en passant ruta, ”halvmove”-räknare och ”fullmove”-räknare. Det första uppgiften som ska utföras är att konvertera positionenen på tavlan till fenannotationer därför att det förekommer först i strängen. Nästa steg i metoden är att ladda in den aktiva spelaren. Efter det måste rockad rättigheterna laddas in. Nästföljande steg är att ladda in en passant rutan. Nästa steg i metoden är att lägga till halvdrags-räknaren.
I en vanlig fenannotation finns det en fullmove-räknare. I detta program finns inte det på grund av att programmet i detta tillstånd inte behöver detta värde. Dock finns det möjlighet att lägga till den i framtiden ifall att programmet ska vidare utvecklas.

Denna klass har också uppgiften att ladda in en position på tavlan med en given ”Position” objekt. Denna metod i klassen heter ”loadPosition” som tar in ett ”Position”-objekt och ett ”Board”-objekt. Det första som utförs i metoden är att hämta fensträngen ifrån ”Position”-objektet och skapa ett ”StringBuilder” med hjälp av strängen. Metoden använder ”StringBuilder” därför att det är mycket smidigare än att använda strängar eftersom det kommer att inträffa många ändringar i strängen.

Först kommer metoden att anropa en hjälpmetod ”loadPiecesToBoard” som har uppgiften att ladda in pjäserna i rätt position på tavlan. Metoden placerar de angivna pjäserna rad för rad enligt fenannotationens regler. Pjäsen placeras med hjälp av en metod ”getPiece” som använder objektet PieceFactory. PieceFactory klassen har uppgiften att returnera nya pjäser som ska placeras på tavlan. Detta är en designmetod som kallas för ”factory designmethod” som handlar om att skapa en klass som har uppgiften att skapa andra objekt.



Figur 23:
Bilden illustrerar “factory designmethod” med hjälp av ett UML diagram. Det som syns är att klassen ”PieceFactory” skapar pjäserna.

Efter att pjäser är placerade på tavlan ska den aktiva spelaren sättas genom att använda hjälpmetoden ”loadActivePlayer”. Nästa steg är att ladda rockadrättigheterna genom att kalla hjälpmetoden ”loadCastlingRights”. Efter det ska en-passant-rutan laddas in

genom metoden ”loadEnPassantSquare”. Det sista som laddas är halvmove-räknaren och detta görs genom hjälpmetoden ”loadFiftyMoveCounter”.

6.2.4 BoardScanner

”BoardScanner”-klassen har uppgiften att ta reda på information angående tavlan för exempel att returnera alla diagonala rutor från en given ruta. Klassen utför också uppdateringar på tavlan efter varje drag som utförs. De viktigaste uppdateringar som görs är att upptäcka villkor som leder till ett avslutat parti. Denna klass håller också reda på olika aspekter om tavlan som för exempel rockadrättigher, aktivaspelaren, en passant ruta och de ruta som är markerad.

”BoardScanner” har många metoder som används av pjäserna eftersom ”getValidMoves” metoderna behöver ha tillgång till olika information från tavlan såsom pjäsernas position, attackerade rutor och mer. En metod som används av alla typer av pjäser är ”removeUnvalidMove” i ”BoardScanner”. Denna metod har uppgiften att radera drag som är ogiltiga på grund av att dragen leder till att kungen ligger i schack. Denna funktion anropas i slutet av ”getValidMoves” om flaggan ”removeCheckMove” är true.

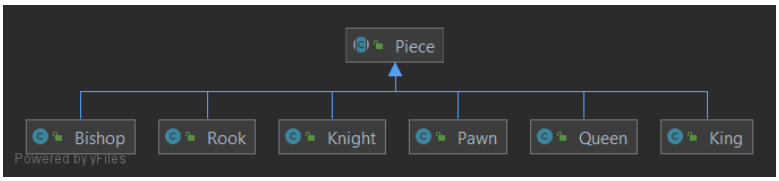
Klassen innehåller metoden ”isSquareAttacked” som kan undersöka om en given ruta är attackerad av en fiendepjäs. Metoden gör detta genom att undersöka de olika riktningar för en fiendepjäs. Om det finns en fiende vid en viss riktning undersöker metoden om denna pjäs kan förflytta sig till rutan. Om den kan är rutan attackerad. Om det inte finns några fiendepjäser som attackerar rutan ifrån de olika riktningarna kommer funktionen därefter söker efter hästar. Om det inte finns några fiendehästar som kan förflyta sig till rutan anses rutan inte vara attackerad.

Klassen kan undersöka om spelet är över. Denna metod heter ”getGameOver” som returnerar ett enum ”GameOverType”. ”GameOvertype” är ett enum som representerar de olika sättet spelet kan vara över. Det metoden gör är att anropa olika hjälpmetoder som undersöker olika fall där spelet anses vara över. Metoden kommer returnera olika typer av GameOverType beroende på vilket kriterium som är sant.

6.2.5 Piece

I denna abstrakta klass finns det metoder som är användbart för alla typer av pjäser men också metoder som måste implementeras om en klass ärver ifrån denna abstrakta klass. Det finns metoder för att rita pjäsen på tavlan och flera getters metoder för att hämta fält som ligger i klassen. Den har en ”getValidMoves” abstrakt metod därför att den är nödvändigt att ha för alla typer av pjäser. Detta gör att alla pjäser som ärver ifrån denna klass måste implementera och overrida ”getValidMoves”-metoden. Metoden valdes att vara abstrakt på grund av att det är omöjligt att skriva en allmän metod som fungerar för alla sub-typer av pjäser.

Alla pjäser ärver från denna klass eftersom de är en typ av en pjäs. Det valdes att använda ärvning i implementationen på pjäserna därför att författaren tycker att alla pjäsklasser är en sub-klass av ”Piece”-klassen och att det känns konstig att en bonde har ett ”Piece”-objekt som den kallar för att utföra olika uppgifter. Klassen har också en konstruktör som sätter olika fält på objektet och de flesta pjäserna använder denna konstruktör därför att det fungerar för alla olika typer av pjäser. Figur 24 visar ärvning av de olika pjäserna.



Figur 24:
Bilden illustrerar ärvning av pjäserna.

7. Användarmanual

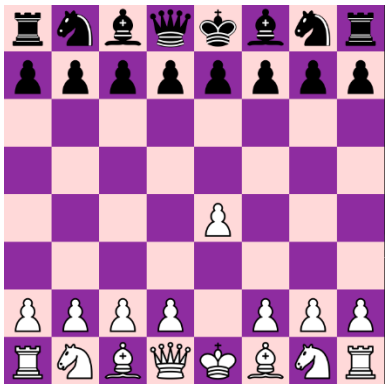
Detta är användarmanualen för schackprogrammet som utvecklas i detta projekt. Här ska det informeras om användningen av detta program.

7.1 Förflyttning av pjäser

Förflyttning i detta schackprogram är väldig enkel. Det första man ska göra är att trycka på en aktivpjäs. Efter trycket, kommer cirklar att ritas på rutorna eller att rutan blir röd. Dessa cirklar syns på figur 23. Detta indikeras att pjäsen som trycktes kan förflytta sig till dessa rutor. Genom att trycka på en av dessa markerade rutor kommer pjäsen att förflytta sig till rutan som figur 24 visar tydlig. Det går att ångra den valda pjäsen genom att trycka på en annan pjäs eller att trycka på en ruta som inte är markerad.

Figur 25:
Bilden visar startpositionen av schackbrädet

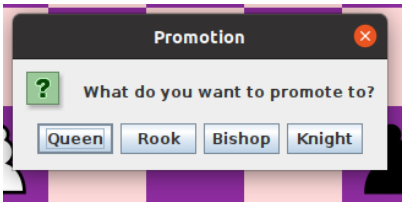
Figur 26:
Bilden illustrerar brädet efter att användaren trycker på bonden.



Figur 27:
Bilden visar att dragen är utförd.

7.2 Promovering

Vid situation där en bonde har nått den sista raden måste den bytas ut med en dam, torn, löpare eller häst. I detta program kommer det att visas ett fönster som frågar användaren vilken pjäs de vill byta mot. Detta fönster visas i figur 24. Om användaren tycker på krysset kommer det att automatisk väljas dam.



Figur 24:
Bilden illustrerar brädet efter att användaren trycker på bonden.

8. Källor

[1] FIDE, ”LawsOfChess,” [Online]. Available: https://www.fide.com/FIDE/handbook/LawsOfChess.pdf. [Använd 28 04 2021].

[2] ”chessprogramming.org,” 20 03 2021. [Online]. Available: https://www.chessprogramming.org/Forsyth-Edwards_Notation. [Använd 29 04 2021].