Projektbeskrivning

Schack i Java

2021-05-28

Projektmedlemmar:
Benjamin Sundvall <bensu844@student.liu.se>
Anton Wegeström <antwe841@student.liu.se>

Handledare:
Jonathan Falk <jonfa001@student.liu.se>

Innehåll

Projektplan

1. Introduktion till projektet

Vi ska skapa schack med lokal multiplayer för två spelare. Vi siktar även på att implementera en schackklocka. Schack är ett brädspel som oftast spelas av två spelare. Varje spelare har 16 pjäser av olika typer. Dem olika typerna beter sig på olika sätt då dem kan förflytta sig och ta motståndare på olika sätt. Spelet går ut på att ta motspelarens kung.

2. Ytterligare bakgrundsinformation

Regler för hur pjäser går och tar i normala fall finns här.

En kung står i **schack** om den riskerar att bli tagen nästa drag. **Schackmatt** innebär att kungen står i schack och att det inte finns något drag som kan ta bort hotet (ex att flytta kungen eller ställa en pjäs mellan). Om en spelare inte kan flytta på någon av sina pjäser utan att ställa sin egen kung i schack är det **patt** vilket innebär att matchen blir oavgjord.

En bonde får sitt **första drag** antingen gå ett eller två steg framåt. Alla andra drag får dem endast gå ett steg.

Om varken kungen eller ett torn har utfört något drag kan dem utföra en **rockad**. Detta innebär att kungen och tornet möts i mitten av deras startpositioner och byter plats, till exempel om kungen står på E1 och ett torn står på H1 och **rockad** utförs så hamnar kungen på G1 och tornet på F1.

Om en bonde lyckas korsa hela brädet kan den **befordras**, vilket innebär att den kan bytas ut mot vilken pjäs som helst av spelarens val (oftast en dam, men det går även att byta mot andra pjäser).

Reglerna för **En passant** beskrivs mer utförligt i den länkade siden men kortfattat innebär det att en bonde kan slå ut en annan bonde ”i förbigående”.

Det finns även ett antal olika regler för fall där matchen slutar **oavgjort**, till exempel om ingen av spelarna har tillräckligt med pjäser för att kunna ställa motståndaren i schackmatt. Se denna artikel för fler fall.

I många mer tävlingsinriktade omgångar används en **klocka**. Klockan räknar ner och om ens tid tar slut förlorar man spelet. Klockan använder sig av **increments** vilket innebär att varje gång det blir ens tur läggs lite tid på för att motverka tiden det tar att till exempel klicka på klockknappen.

Mer om reglerna för schack finns här: https://en.wikipedia.org/wiki/Rules_of_chess

3. Milstolpar

Syftet med milstolparna är att få en överblick över vad som behöver göras i projektet. Dessa

#	Beskrivning
1	Det finns datatyper för spelplan och pjäser.
2	Det finns en grafisk komponent som visar spelplanen.
3	En testklass kan skapa en tom spelplan och visa den grafiskt i ett fönster.
4	Testklassen visar en spelplan med bokstäver som representerar schackpjäserna med hjälp av den grafiska komponenten.
5	Det går att stänga fönstret genom att klicka på stängknappen.
6	Samtliga pjästyper har en metod för att rita upp sig.
7	Den grafiska komponenten kan nu visa en spelplan med olika pjäser som placeras ”över” spelplanen på givna x/y-koordinater.
8	Skapa spelarobjekt som ska hålla koll på färg, (klocka och poäng).
9	Spelplanen innehåller nu fält för spelare.
10	Spelplanen håller även koll på vems tur det är.
11	Samtliga pjästyper har en metod för att beräkna alla möjliga drag som om det inte skulle finnas andra pjäser på brädet.
12	Dessa drag visas på spelplanen när pjäsen väljs.
13	Spelet förstår när man klickar på en pjäs.
14	Samtliga pjästyper tar nu hänsyn till andra pjäser på brädet när de beräknar möjliga drag.
15	Bönder kan nu ta två steg framåt första gången de rör sig.
16	Bönder kan göra ”en passant”.
17	Kungen kan nu göra rockad.
18	En bonde som kommer över till andra sidan av planen kan nu befordras till vilken annan typ av pjäs som helst. Spelaren får upp en ruta med alternativ för val av pjäs.
19	Vet när det är schack.
20	Vet när det är schackmatt/patt.

21	Game over pop-up som visar vem som vann och varför, samt möjlighet att starta om spelet.
22	Spelarobjekt har koll på sin klocka
23	Implementera regler för alla typer av draws.
24	Kan spara ett spel

4. Övriga implementationsförberedelser

Vi kommer ha en **Board**-klass som innehåller en array av **Pieces**. Boarden vet vilka pjäser som står vart. Den kommer även ha metoder för att flytta och få pjäser.

BoardViewer vet hur man ritar upp en board inklusive pjäserna på boarden och fungerar som ett mellansteg mellan datorn och spelaren.

Piece är ett interface för alla olika underklasser av olika pjästyper. Varje typ av pjäs kommer att ha en egen klass som vet hur den till exempel får förflytta sig. Dem vet även vilken spelare dem tillhör.

Player klassen håller koll på sin färg, hur mycket tid som finns kvar på dennes timer, vilken timer increment som valts samt vilken score den har.

GameManager bestämmer över hala spelet. Klassen skapar en board, har koll på spelarna, kontrollerar om någon står i schack, kan skapa nya spel och egentligen allt som behövs för att föra samman de olika delarna.

Klasser:

GameManager	**Piece (interface)**	**Player**
Board board BoardViewer boardViewer List<Player> players int activePlayerIndex boolean isCheck() boolean isMate() boolean isDraw()	Player owner boolean hasMoved	PlayerColor playerColor double timeLeft double timeIncrement int score
void startGame() void newGame()	List<String> getMoves(Board board)	int getTimeLeft() int getTimeIncrement()

Board	**BoardViewer**
Piece[][] pieces int width int height	Board board Color lightSquareColor Color darkSquareColor Map<Integer, Color> playerColors
void movePiece(String square1, String square2) int getWidth() int getHeight() Piece getPiece(String square) void setPiece(String square, Piece piece)	void show()

5. Utveckling och samarbete

Ambitionsnivå

Vi båda siktar på att få en femma i projektet och vill bli klara till deadline.

Arbetstider

Vi kommer att jobba ca tio timmar per vecka per person från och med vecka 13. Vi kommer ha avstämningar varje måndag och fredag för att gå igenom vad vi har gjort samt vad som behöver göras.

Tidslinje

Vi har satt ut ett antal gränser för när vi ska ha kommit till en viss punkt i milstolpslistan, dessa är:

- Punkt 8 den 4 april
- Punkt 13 den 11 april
- Punkt 14 den 18 april
- Punkt 17 den 25 april
- Punkt 18 den 2 maj
- Punkt 19 den 9 maj
- Punkt 20 den 16 maj

Resten av punkterna uppfylls i mån av tid.

Undantag

Vi kommer att utgå ifrån att man kommer på alla bestämda möten om det inte händer något utom ens kontroll som förhindrar det. Om så är fallet så ska den andra personen meddelas så snart som möjligt. Ett exempel på in icke giltig avledning är att man behöver plugga något annat ämne. En giltig anledning kan vara till exempel att man sitter fast på ett försenat tåg.

Samarbete

Till en början programmerar vi tillsammans, men allt eftersom delar vi upp arbetet mer och mer. Vi har även två avstämningsträffar per vecka där vi går igenom vad vi har gjort och vad som behöver göras.

Känsla inför projektet

En rolig utmaning. Allt har varit väldigt styrt fram tills nu. Mer ansvar och inte lika mycket stöd nu.

Projektrapport

6. Implementationsbeskrivning

Detta avsnitt har som syfte att beskriva implementationen för att läsaren ska få en förståelse för hur programmet är uppbyggt och fungerar.

6.1. Milstolpar

Projektet är uppdelat i milstolpar som finns tidigare i detta dokument. Vilka milstolpar som har implementerats kan ses i ***tabell 1***. Punkterna i tabellen motsvarar punkterna i milstolpslistan.

Tabell 1 *– Lista över vilka milstolpar som är genomförda.*

1	Helt genomförd	9	Helt genomförd	17	Helt genomförd
2	Helt genomförd	10	Helt genomförd	18	Helt genomförd
3	Helt genomförd	11	Helt genomförd	19	Helt genomförd
4	Helt genomförd	12	Helt genomförd	20	Helt genomförd
5	Helt genomförd	13	Helt genomförd	21	Helt genomförd
6	Helt genomförd	14	Helt genomförd	22	Helt genomförd
7	Helt genomförd	15	Helt genomförd	23	Ej genomförd
8	Helt genomförd	16	Helt genomförd	24	Delvis genomförd

6.2. Dokumentation för programstruktur, med UML-diagram

Spelet går i turer som passas mellan den vita och den svarta spelaren. I början av varje tur beräkar Board-klassen alla hot mot kungen och alla möjliga drag. Därefter låter ChessComponent-klassen den aktiva spelaren välja ett drag som sedan utförs av Board-klassen. Efter detta går turen över till nästa spelare. Implementationen är uppdelad i tre paket (packages): ”game” för all spellogik, ”GUI” för allt relaterat till hur spelet visas grafiskt och ”pieces” för alla olika pjästyper.

Spelet (Game)

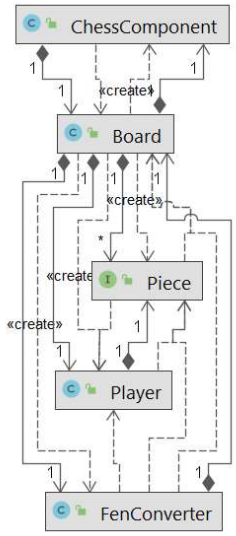
Spelets ”Game”-del består av ett antal klasser, hur dessa är relaterade kan ses i ***figur 2***.

För att beräkna schackmatt, pins och skyddade pjäser utgår vi från kungen och kollar längs alla rörelsevektorer (diagonalt, ortogonalt) och punktlistor (för hästar och bönder) för att se vilka potentiella hot som finns för kungen. Om en motståndarpjäs utgör ett direkt hot mot kungen (kan ta kungen nästa tur) skapas ett nytt ”direct threat” som innehåller alla rutor man kan gå till för att stoppa just det hotet (ställa en pjäs mellan eller ta pjäsen som utgör hotet). Om en allierad riskerar att öppna upp ett nytt hot mot sin egen kung genom att flyttas skapas i stället en ny ”pin” som innehåller alla rutor som den ”pinned” pjäsen kan gå till utan att sätta kungen i schack. Dessa ”direct

threats” och ”pins” används sedan för att begränsa pjäsers rörelse med metoderna limitMovesToThreatSquares() och limitMovesToPinSquares() i klassen AbstractPiece.

Board

Board-klassens uppgift är att hålla reda på all logik och information om spelets tillstånd som krävs för att man ska kunna spela. De två viktigaste metoderna är getMoves() och performMove(), som används för att göra drag i spelet. Metoden getMoves() returnerar en lista på alla möjliga drag som kan utföras i det nuvarande tillståndet. GUI:n använder sedan denna information för att visa spelaren vilka drag som kan göras. GUI:n skickar sedan tillbaka vilket drag spelaren har valt att utföra genom att kalla på metoden performMove(). Metoden performMove() tar ett Move och utför det i enlighet med spelreglerna. Board skapar även ett flertal andra objekt som beror på den. Se ***Figur 1***.



Figur 1 – Visar objekt som Board skapar.

Move

Move-klassen lagrar all information som krävs för att ett drag ska kunna utföras. Denna information inkluderar: vilken ruta pjäsen flyttas från, vilken ruta pjäsen flyttas till, vilken pjäs som flyttas och om draget har några undantag för vad som händer när draget utförs (se kommentarer i MoveCharacteristics-klassen).

MoveCharacteristics

MoveCharacteristics är en uppräkningstyp (enum) som innehåller de olika specialfallen ett drag kan ha.

FenConverter

FenConverter är en klass som används för att spara och ladda in spel sparade i så kallad

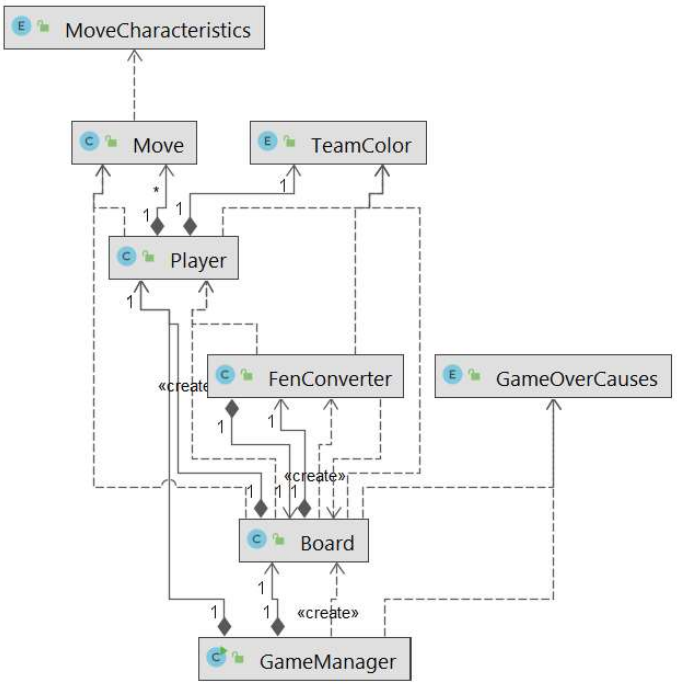
FEN-notation (läs mer på wikipedia). De två huvudsakliga funktionerna i FenConverter är createBoardFromFEN(), som läser av en textsträng och modifierar brädet enligt informationen i textsträngen, och convertBoardToFEN(), som läser av informationen i spelbrädet och genererar en textsträng som innehåller all nödvändig information för att kunna återskapa spelets tillstånd.

GameManager

GameManager:s uppgift är att initiera allt som behövs för ett nytt spel i början av matchen och att sedan räkna ned tiden som spelarna har kvar.

Player

Player-klassen lagrar vilka drag som kan utföras av spelaren, vilka rutor som är hotade av spelaren och ett antal spelarspecifika konstanter som till exempel vilken rad en pjäs kan bli befordrad på och vilka rutor spelarens torn börjar på (för att kontrollera om en spelare har rört på sina torn, vilket skulle förhindra rockad).

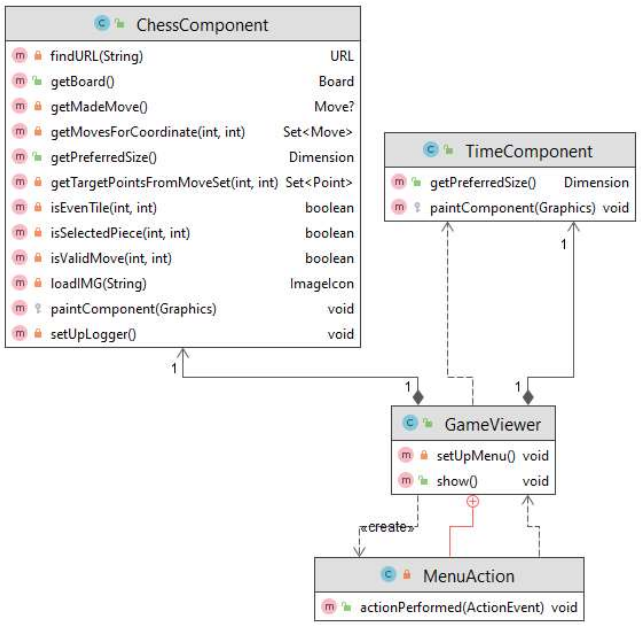


***Figur 2** - Visar relationen mellan alla "Game"-klasser.*

Grafiskt användargränssnitt (GUI)

Projektets grafiska användargränssnitt består av tre klasser: GameViewer, ChessComponent och TimeComponent. GameViewer är ansvarig för allt grafiskt och använder sig bland annat av ChessComponent och TimeComponent för att rita upp programmet i ett fönster. Hela GUI-delen av projektet bygger på Java Swing. GameViewer skapar ett fönster med hjälp av en JFrame och lägger till delar så som ChessComponent, TimeComponent och en menyrad. ChessComponent och

TimeComponent är förlängningar av JComponent. Dessa objekt har som huvudsakligt syfte att kunna rita upp sig själva samt att veta hur stora de vill vara. Implementation för hur de ritar upp sig själva skiljer sig åt markant. ChessComponent använder sig av for-loopar för att iterera över alla rutor (pieces) på brädet (Board). För att rita upp brädet bestäms färgen på alla rutor genom att undersöka om rutans position på brädet är jämn eller udda. För att rita upp pjäserna ritas en bild från resources-katalogen motsvarande pjäsen som står på rutan upp. TimeComponent frågar spelarobjekten hur mycket tid de har kvar och skriver sedan ut tiden. ChessComponent är även ansvarig för mänsklig interaktion med spelplanen. När en mänsklig spelare klickar på en ruta på spelplanen kallar ChessComponent på en metod från Board och frågar vad som ska hända och ritar sedan om bilden med ändringarna. Relationen mellan klasserna kan ses i ***figur 3***.



Figur 3 *– Visar relationen mellan GUI-klasserna.*

Pjäser (Pieces)

Det finns sex olika typer av pjäser. Dessa typer har mycket gemensamt, men det finns viss funktionalitet som skiljer sig åt mellan typerna. Allt som går att implementera gemensamt har implementerats i abstrakta klasser och aspekter specifika till en viss pjästyp har implementerats i dess egna klass.

För att beräkna vilka drag en pjäs kan göra beräknas först alla drag som skulle kunna göras om det inte fanns en kung som riskerar att bli hotad. Alla drag som skulle resultera i att kungen blir hotad dras sedan bort från de genererade dragen. Hur dessa klasser relaterar till varandra kan ses i ***figur 4***.

Piece

Piece är ett gränssnitt (interface) för alla olika typer av pjäser. För att en klass ska få kalla sig en underklass till Piece behöver den implementera alla metoder definierade i

Piece. Dessa metoder är det minsta ett Piece-objekt behöver veta om sig själv för att fungera.

AbstractPiece

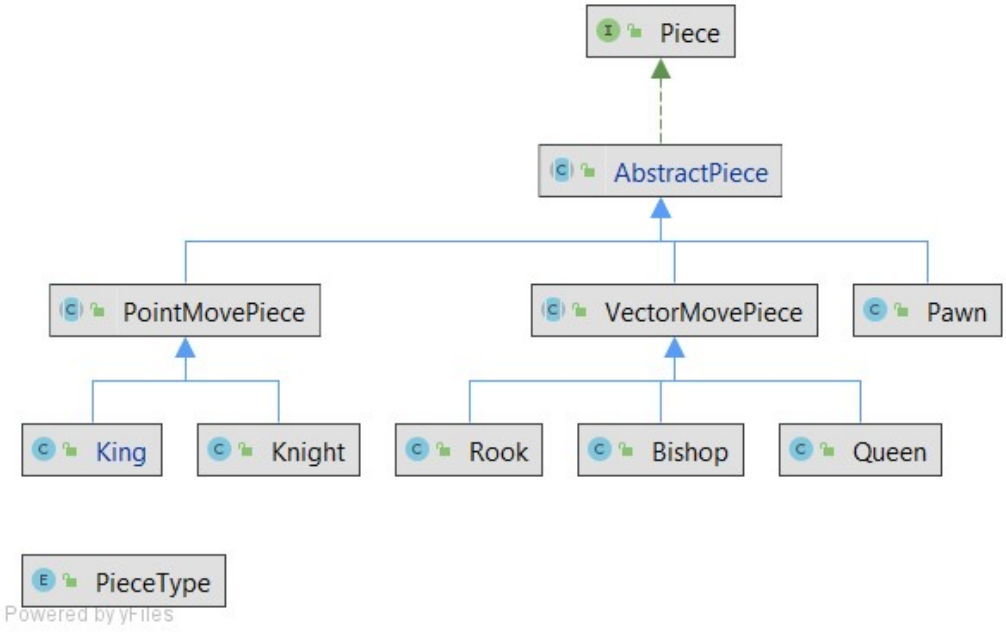
AbstractPiece är en abstrakt klass som implementerar gränssnittet Piece. Detta innebär att klassen innehåller implementationer till vissa metoder definierade i Piece som har en gemensam implementation för alla underklasser. Detta görs huvudsakligen för att inte upprepa kod.

PointMovePiece

PointMovePiece är en abstrakt klass som förlänger AbstractPiece. Syftet med klassen är att ge en gemensam implementation av metoden getMoves för pjäser som endast kan gå till ett fåtal punkter på brädet (King, Queen). Alla underklasser till PointMovePiece har ett fält med förflyttningar den skulle kunna göra utan att ta hänsyn till var på brädet pjäsen står eller vilka pjäser som står runt om den. Metoden getMoves tar dessa förflyttningar och undersöker om de fortfarande är möjliga på det aktiva brädet.

VectorMovePiece

VectorMovePiece är en abstrakt klass som förlänger AbstractPiece. Syftet med klassen är att ge en gemensam implementation av metoden getMoves för de pjäser som förflyttar sig i raka linjer (Queen, Rook och Bishop). En VectorMovePiece kan gå hur långt som helst i en rak linje, men den kan inte gå igenom andra pjäser. Metoden getVectorMoves() tar en array av riktningar (vektorer) som en pjäs kan gå i och beräknar alla drag som kan utföras. De drag som skulle resultera i att den egna kungen blir hotad filtreras sedan bort.



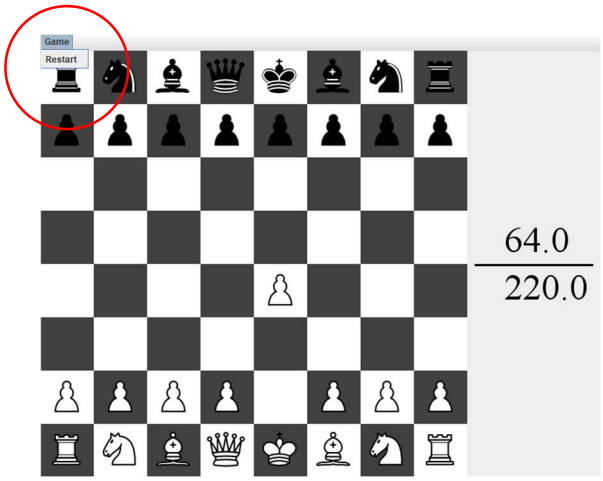
Figur 4 *– Visar relationen mellan alla Piece-klasser.*

7. Användarmanual

Användarmanualen går igenom allt som behövs för att komma i gång med spelet samt allt som är nödvändigt att veta.

Starta spelet

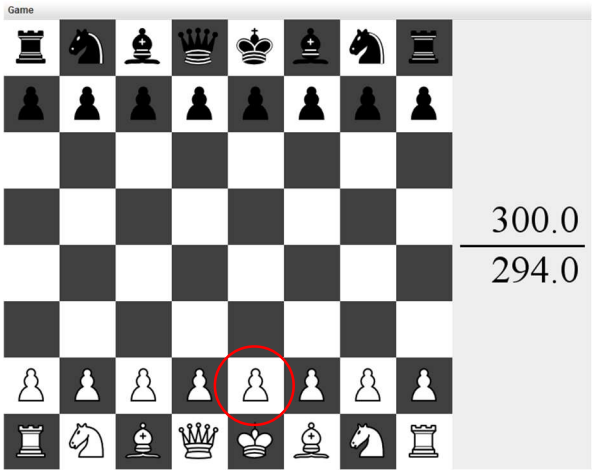
För att starta spelet kör man main-loopen i GameManager. För att starta om ett redan startat spel klickar man på Game > Restart (se figur 5).



***Figur 5** – Restart-knappen uppe i vänstra hörnet.*

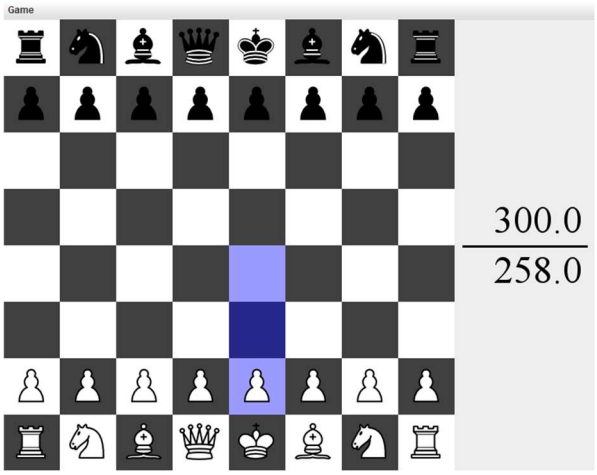
Att göra drag

1. Klicka på en pjäs som kan göra drag enligt reglerna för schack (se figur 6).



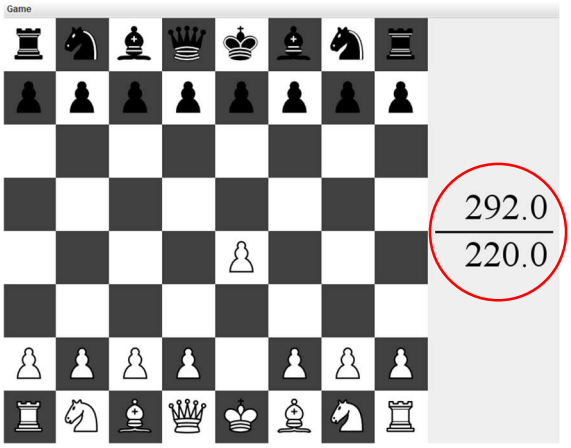
***Figur 6** – Bonden på E4 är en av pjäserna som kan röra sig.*

2. Klicka på en av de markerade rutorna med möjliga drag för den valda pjäsen (se figur 7).



***Figur 7** – Rutorna den valda pjäsen kan gå till är markerade.*

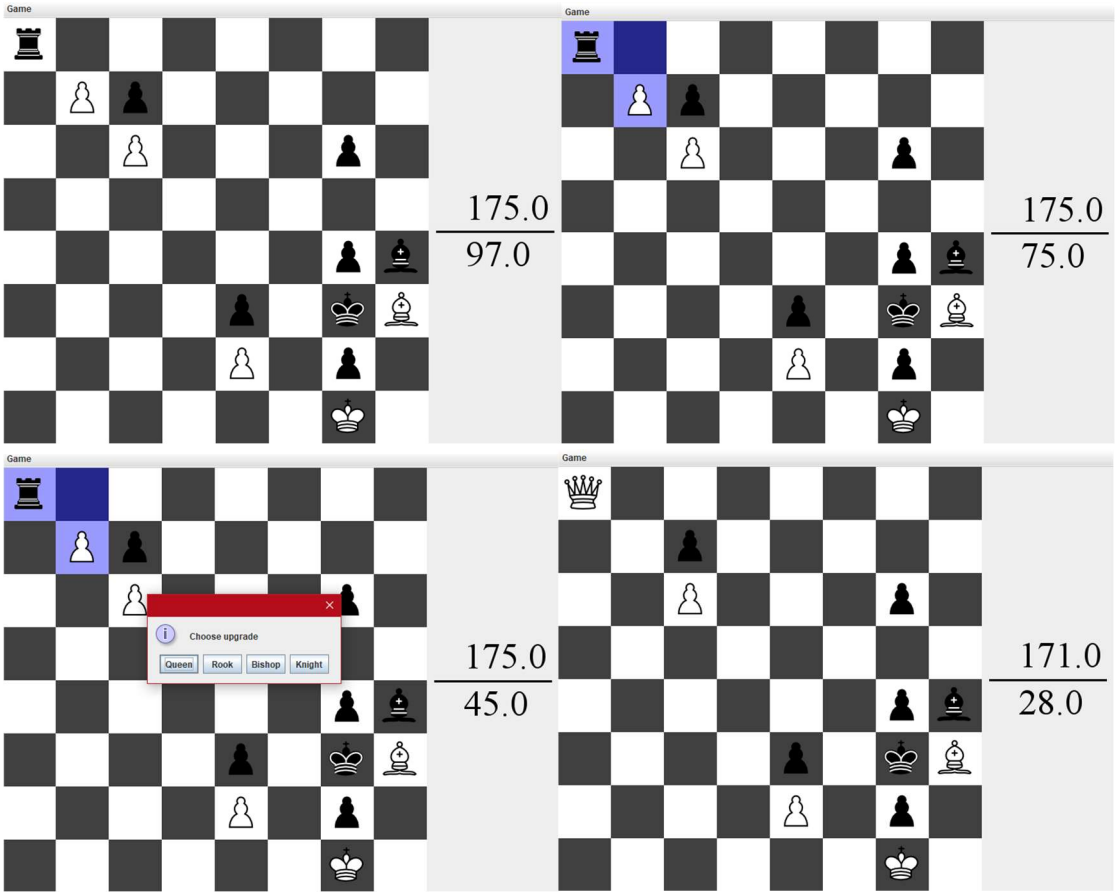
Draget är nu genomfört och turen har passats till motspelaren. För att se vilken spelare som står på tur kan man se vilken tidräknare som räknar ner (se figur 8).



***Figur 8** – Den återstående tiden för båda spelarna.*

Befordran

Om en bonde förflyttar sig till andra sidan spelplanen kan den befordras. När detta sker får man ett popup-fönster där man får välja vilken pjäs den befordrade bonden ska bli (se figur 9).



Figur 9 *– De steg som krävs för att befordra en bonde.*