# Projektbeskrivning

## Schack

## 2024-03-04

**Projektmedlemmar:**
Markus Svedenheim <marsv260@student.liu.se>

**Handledare:**
Simon Hansson <simha158@student.liu.se>

## Innehåll

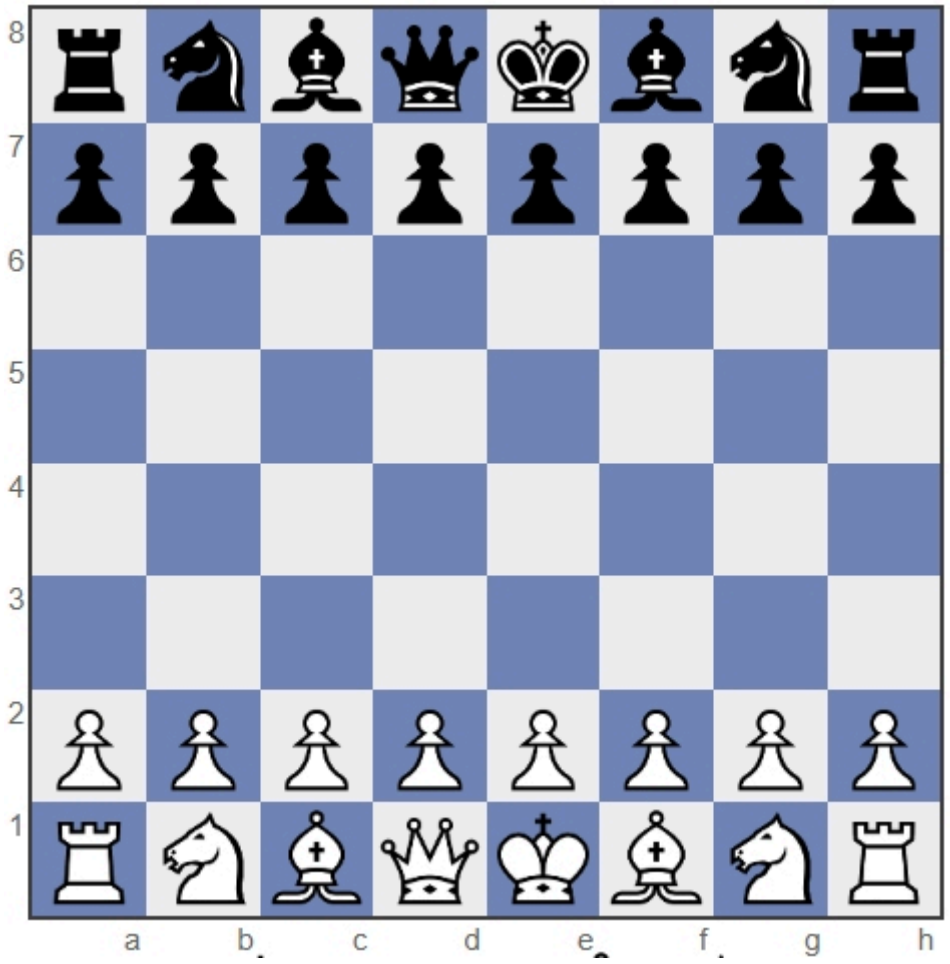
# Projektplan

## 1. Introduktion till projektet

Schack är ett mycket gammalt spel som består av att två olika spelare turas om att flytta en av sina 16 pjäser efter vissa förutsatta regler på ett 8x8 stort spelbräde. En spelare vinner först då dess motståndare inte längre kan försvara sin kung och spelet är då över. Det är också möjligt att ett parti schack slutar oavgjort ifall en av spelarna inte har några lagliga drag. Trots vad som kan verka som väldigt enkla regler finns det en enorm komplexitet bakom ett parti schack och det finns en snudd på oändlig mängd olika partier.

Ifall den standardvariant av schack inte längre lockar finns det också flera olika varianter på spelet som var och en bidrar med en unik twist. Exempelvis Fischer Random där startpositionen på alla huvudpjäser slumpas i början av spelet så att all normal teori inte längre är relevant.

Nedan finns en exempelbild på hur ett normalt parti schack ställs upp och kan se ut då det spelas på en dator.



# 2. Ytterligare bakgrundsinformation

Reglarna i schack kan ofta tyckas vara väldigt svåra, trots det finns det enbart 6 olika pjäser som rör sig på olika sätt. Några ytterligare regler som är viktiga att komma ihåg är att det inte är tillåtet att ta sina egna pjäser eller gå igenom sina egna pjäser. Däremot är det helt okej och oftast smart att ta motståndarens pjäser.

**Bonden** kan antingen ta en pjäs snett framåt eller gå ett steg framåt. Förutom dess första drag då bonden kan gå två steg framåt. Ifall en bonde når till andra sidan av spelplanen kan den bytas mot en annan pjäs i samma färg, utom kungen.

**Tornet** kan gå vertikalt eller horisontellt. Dessutom kan tornet och kungen göra en rockad ifall varken av dem har rört sig. Då flyttar sig kungen två steg mot tornet och tornet flyttas till utsidan av kungen.

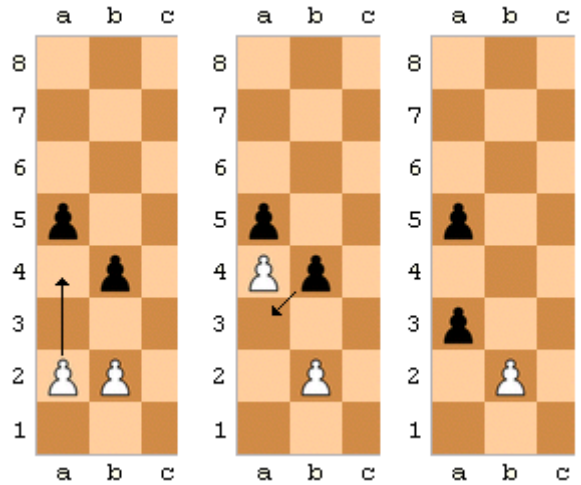
**Hästen** kan hoppa över pjäser i en L-form. Den hoppar då två steg horisontellt eller vertikalt plus ett till steg åt sidan.

**Löparen** går på diagonaler, detta innebär att den alltid är fast på samma färg.

**Damen** är den starkaste pjäsen. Denna kan röra sig både som en löpare eller som ett torn.

**Kungen** är den viktigaste pjäsen. Denne kan bara röra sig ett steg men ifall den är på väg att fångas av motståndaren är spelet över.

Några ytterligare regler är att ifall kungen är i schack, alltså hotas att tas av en annan pjäs på nästa drag, måste detta åtgärdas omgående. Dessutom får rockad ej genomföras då kungen är i schack. Ifall samma position upprepas tre gånger avslutas även spelet i remi (oavgjort). En ytterligare regel som är relativt nytt är också en passant: Ifall en bonde går två steg på sitt första drag och därmed går




förbi en annan bonde är det möjligt för denna att fångas som om att den endast hade gått ett steg, detta visas nedan.

Mer information finns att hitta men det garanteras inte att alla regler är

implementerade på samma vis. En enkel källa finns här:
https://sv.wikipedia.org/wiki/Schack

## 3. Milstolpar

| # | Beskrivning |
|---|---|
| 1 | Ett fönster som går att öppna och stänga som också visar ett schackbräde, det vill säga ett rutnät i skiftande färger. |
| 2 | Klass för torn implementeras och går att flytta horisontellt och vertikalt utan att kollidera med kanter eller andra pjäser. Att trycka på pjäser visar också tillåtna drag. |
| 3 | Fortsätt med att implementera klass för löpare och häst. |
| 4 | Klass för dam som möjligtvis ärver från torn och löpare samt kung. |
| 5 | Bönder som kan röra sig ett steg framåt eller ta ett steg diagonalt. |
| 6 | Pjäser kan nu tas av andra pjäser och bönder promoveras automatiskt till damer. |
| 7 | Implementera rockad, två steg bonde, en passant. |
| 8 | Implementera schack och schackmatt. |
| 9 | Pjäser ställs nu upp korrekt då spelet startar med svart och vit spelare. |
| 10 | Kontrollera att spelet går att köra spelare mot spelare. |
| 11 | Meny för att promovera bönder till valfri pjäs. |
| 12 | Implementera en startmeny med spela/sluta. |
| 13 | Visa en prompt då en spelare hamnar i schackmatt. |
| 14 | Visa vilka pjäser som blivit tagna och materiella skillnader. |
| 15 | Implementera regler för Fischer Random, undersök speciella regler för rockad. Lägg till detta som alternativ i menyn. |
| 16 | Lägg till en informations prompt för olika öppningar. https://github.com/lichess-org/lila-openingexplorer?tab=readme-ov-file#http-api |
| 17 | |

| | |
|---|---|
| 18 | |
| 19 | |
| 20 | |
| 21 | |
| 22 | |
| 23 | |
| ... | Lämna in! |

# 4. Övriga implementationsförberedelser

Lista av olika grundtankar i projektstruktur.

- Brädet och varje pjäs är en egen klass.
- Brädet kan representeras av en enum/class per ruta i en 2d 8x8 array.
- Varje pjäs håller koll på vilka drag den kan göra och dessa blir tillgängliga då den väljs.
- Möjliga drag kontrolleras genom att stegvis gå i pjäsens riktning till kollision.
- Dam, torn, löpare beter sig lika i att de går rakt till krock.
- Liknande med BoardViewer som i Tetris.

# 5. Utveckling och samarbete

N/A

# Projektrapport

Även om denna del inte ska lämnas in förrän projektet är klart, är det **viktigt att arbeta med den kontinuerligt** under projektets gång! Speciellt finns det några avsnitt där ni ska beskriva information som ni lätt kan glömma av när veckorna går (vilket flera tidigare studenter också har kommenterat).

Tänk på att ligga på lagom ambitionsnivå! En välskriven implementationsbeskrivning (avsnitt 6) hamnar normalt på **3-6 sidor** i det givna formatet och radavståndet, med ett par mindre UML-diagram och kanske ett par andra små illustrerande bilder vid behov. Hela projektrapporten (denna sista halva av dokumentet) behöver sällan mer än 10-12 sidor.

## 6. Implementationsbeskrivning

I det här avsnittet, och dess underavsnitt (6.x), beskriver ni olika aspekter av själva *implementationen*, under förutsättning att läsaren redan förstår vad *syftet* med projektet är (det har ju beskrivits tidigare).

Tänk er att någon ska vidareutveckla projektet, kanske genom att fixa eventuella buggar eller skapa utökningar. Då finns det en hel del som den personen kan behöva förstå så att man vet *var* funktionaliteten finns, *hur* den är uppdelad, och så vidare. Algoritmer och övergripande design passar också in i det här kapitlet.

Bilder, flödesdiagram, osv. är starkt rekommenderat!

Skapa gärna egna delkapitel för enskilda delar, om det underlättar. **Ta inte bort några rubriker!**

**Även detta är en del av examinationen som visar att ni förstår vad ni gör!**

### *6.1. Milstolpar*

Ange för varje milstolpe om ni har genomfört den helt, delvis eller inte alls.

Detta är till för att labbhandledaren ska veta vilken funktionalitet man kan ”leta efter” i koden. Själva bedömningen beror *inte* på antalet milstolpar i sig, och inte heller på om man ”hann med” milstolparna eller inte!

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

Programkod behöver dokumenteras för att man ska förstå hur den fungerar och hur allt hänger ihop. Vissa typer av dokumentation är direkt relaterad till ett enda fält, en enda metod eller en enda klass och placeras då lämpligast vid fältet, metoden eller klassen i en Javadoc-kommentar, *inte här*. Då är det både enklare att hitta dokumentationen och större chans att den faktiskt uppdateras när det sker ändringar. Annan dokumentation är mer övergripande och saknar en naturlig plats i koden. Då kan den placeras här. Det kan gälla till exempel:

- **Övergripande programstruktur**, t.ex. att man har implementerat ett

spel som styrs av timer-tick n gånger per sekund där man vid varje sådant tick först tar hand om input och gör eventuella förflyttningar för objekt av typ X, Y och Z, därefter kontrollerar kollisioner vilket sker med hjälp av klass W, och till slut uppdaterar skärmen.

- **Översikter över relaterade klasser** och hur de hänger ihop.
  - Här kan det ofta vara bra att använda **UML-diagram** för att illustrera – det finns även i betygskraven. Fundera då först på vilka grupper av klasser det är ni vill beskriva, och skapa sedan ett UML-diagram för varje grupp av klasser.
  - Notera att det sällan är särskilt användbart att lägga in hela projektet i ett enda gigantiskt diagram (vad är det då man fokuserar på?). Hitta intressanta delstrukturer och visa dem. Ni behöver normalt inte ha med fält eller metoder i diagrammen.
  - **Skriv sedan en textbeskrivning av vad det är ni illustrerar med UML-diagrammet**. Texten är den huvudsakliga dokumentationen medan UML-diagrammet hjälper läsaren att förstå texten och få en översikt.
  - IDEA kan hjälpa till att göra klassdiagram som ni sedan kan klippa och klistra in i dokumentet. Högerklicka i en editor och välj Diagrams / Show Diagram. Ni kan sedan lägga till och ta bort klasser med högerklicksmenyn. Exportera till bildfil med högerklick / Export to File.

I det här avsnittet har ni också en möjlighet att visa upp era kunskapen genom att diskutera koden i objektorienterade termer. Ni kan till exempel diskutera hur ni använder och har nytta av (åtminstone en del av) objekt/klasser, konstruktorer, typhierarkier, interface, ärvning, overriding, abstrakta klasser, subtypspolymorfism, och inkapsling (accessnivåer).

Labbhandledaren och examinatorn kommer bland annat att använda dokumentationen i det här avsnittet för att förstå programmet vid bedömningen. Ni kan också tänka er att ni själva ska vidareutveckla projektet efter att en annan grupp har utvecklat grunden. Vad skulle ni själva vilja veta i det läget?

**När ni pratar om klasser och metoder ska deras namn anges tydligt (inte bara ”vår timerklass” eller ”utritningsmetoden”).**

Framhäv gärna det ni själva tycker är **bra/intressanta lösningar** eller annat som handledaren borde titta på vid den senare genomgången av programkoden.

Vi räknar med att de flesta projekt behöver runt **3-6 sidor** för det här avsnittet.

# 7. Användarmanual

När ni har implementerat ett program krävs det också en manual som förklarar hur programmet fungerar. Ni ska beskriva programmet tillräckligt mycket för att en labbhandledare själv ska kunna *starta det, testa det och förstå hur det används*.

Inkludera flera (**minst 3**) **skärmdumpar** som visar hur programmet ser ut! Dessa ska vara ”inline” i detta dokument, inte i separata filer. Sikta på att visa de relevanta delarna av programmet för någon som *inte* startar det själv, utan bara läser manualen!

(Glöm inte att ta bort våra instruktioner, och exportera till PDF-format med korrekt namn enligt websidorna, innan ni skickar in!)