# Tron-spillet

Regulær eksamensopgave, OOPD 2010/2011

Daniel Hessellund Egeberg
L. Christian Thoudahl
Søren Dahlgaard

17. januar 2011

# Indhold

# 1 Resumé

Du skal designe, implementere og teste dele af et computerspil. Nogle dele af programmet har du fået udleveret, og nogle dele skal du selv lave. Du vil også blive bedt om at ændre noget af den udleverede kode.

Hver opgave består af en indledende beskrivelse efterfulgt af en række delopgaver. Det er kun i delopgaverne, du bliver bedt om at lave noget.

Projektets dokumentation kan findes i BlueJ fra menuen eller ved at trykke Ctrl+J. Det anbefales at bruge dette inden du går i gang med opgaverne til at få et overblik over projektet.

Opgavesættet er normeret til 20 timers arbejde og skal afleveres fredag den 21. januar, kl. 15:00 på Absalon (se kapitel 2).

# 2 Retningslinjer og formalia

Denne opgave skal løses individuelt. Der tillades ikke genaflevering. Under helt særlige omstændigheder (sygdom, dødsfald, m.v.) kan der gives en udsættelse efter dispensation fra studienævnet. En dispensationsansøgning skal være studienævnet i hænde inden afleveringsfristens udløb. Yderligere oplysninger kan findes i eksamensbestemmelserne for Det Naturvidenskabelige Fakultet samt studieordningen for Datalogiuddannelsen. Eksamenssnyd (herunder afskrift fra andre personer) behandles efter Københavns Universitets disciplinærbestemmelser og kan medføre annullering af samtlige eksamensresultater for alle kurser det pågældende studieår samt relegering. Eksamensopgaven udgøres af nærværende dokument samt eventuelle supplerende beskeder fra kursets undervisere, det måtte være nødvendigt at give på kursets hjemmeside på Absalon. Det er deltagerens ansvar løbende at holde sig orienteret på hjemmesiden.

## 2.1 Afleveringsformat

Der skal afleveres to filer i Absalon under menupunktet "Eksamen" > "Eksamensopgaven": En .jar fil med programmet og et besvarelsesdokument enten som .pdf eller .txt. Disse skal navngives som følger:

```
<efternavn>.<fornavn>.eksamen.jar
<efternavn>.<fornavn>.eksamen.pdf|txt
```

Hedder du Daniel Hessellund Egeberg skal filen således kaldes `Egeberg.DanielHessellund.eksamen.jar`. Mere specifikt skal jar-filen indeholde:

1. **Java-kildekode** forsynet med kommentarer — både Javadoc og ikke-Javadoc. Javadoc kommentarer skal skrives for hver udviklet klasse, metode og grænseflade. For hver metode skal parameter- og returværdier dokumenteres. Hvor der kastes undtagelser skal disse også dokumenteres. Ikke-Javadoc kommentarer forventes i kildekoden hvor det ellers ville være vanskeligt at forstå koden.
2. **JUnit-testklasser** hvor enkelte delopgaver kræver det.

Besvarelsesdokumentet skal indeholde besvarelser på opgaver der ikke skal besvares med Java-kildekode. Disse opgaver er markeret med stjerne (*).

*Du bedømmes ud fra det, du har afleveret! Hvis du ved en fejltagelse ikke har inkluderet fx din kildekode, kan denne ikke bedømmes, og karakteren bliver derefter!*

## 2.2 Afleveringssted

Besvarelsen skal afleveres elektronisk via Absalon senest fredag 21. januar 2011 kl. 15.00.

I tilfælde af, og *kun* i tilfælde af, at Absalon ikke fungerer i tidsrummet 21. januar, kl.13.00 – 15.00, kan besvarelser sendes til kaiip@diku.dk; der vil umiddelbart efter opgavens modtagelse på denne mailadresse blive sendt et tidsstemplet svar tilbage til den adresse, som opgaven er afsendt fra.

I tilfælde af, og *kun* i tilfælde af, at hverken Absalon eller DIKUs mailsystem fungerer 21. januar i tidsrummet 13.00 – 15.00, kan besvarelsen gemmes på blivende medium (f.eks. en USB-nøgle) og overdrages til den kursusansvarlige frem til 21. januar kl. 17.00 i DIKUs kantine på Universitetsparken 1.

## 2.3 Bedømmelseskriterier og arbejdsfordeling

Fordelingen af arbejdsbyrden på opgaverne er estimeret til:

**Opgave 1: Spillepladen og omgivelserne i Tron** Ca. 1 time

**Opgave 2: Spillere i Tron** Ca. 6 timer

**Opgave 3: Fil I/O, spilkonfigurationer og spillets udførsel** Ca. 4 timer

**Opgave 4: GUI, Model-View-Controller og sammensætning af program** Ca. 8 timer

**Opgave 5: Spillets design og opbygning** Ca. 1 time

Den endelige karakter tager udgangspunkt i besvarelsen af de enkelte delopgaver og hvor godt besvarelsen som helhed opfylder læringsmålene, som kan ses på kursets SIS-hjemmeside[1]. Det er tilrådeligt selv at forholde sig til læringsmålene når opgaven besvares. Det bemærkes at vægtningen af karakteren generelt vil afvige en smule fra vægtningen af den normerede arbejdsbyrde, samt at karakteren også påvirkes af hvorvidt det færdige program fungerer, og hvor godt det fungerer.

Der lægges vægt på at der afleveres et kørende program, dvs. at et simpelt og lille men veldesignet, kørende program foretrækkes frem for et ambitiøst eller stort program, der ikke rigtig virker. Det er vigtigt, at individuelle klasser er veldesignet, specificeret, implementeret og afprøvet, selv hvis programmet som helhed ikke kan køre. Det anbefales stærkt at designe, implementere og afprøve et system med meget enkel funktionalitet, inden eventuelle udvidelser føjes til.

## 2.4 Tilladt samarbejde under eksamensugen

Bemærk følgende politik for samarbejde under eksamensugen:

[1] http://sis.ku.dk/kurser/viskursus.aspx?knr=120394

1. Eksaminanderne må stille spørgsmål til hinanden, som er uafhængig af opgaven, fx om de obligatoriske opgaver eller vedr. stof i en af lærebøgerne, Java API, vejledningerne eller andre almene ressourcer.

    - Tænk på denne eksamen som en "open book"-eksamen: det er tilladt at bruge medeksaminander, lige som man bruger en bog eller anden alment tilgængelig ressource (som altså ikke ved noget specifikt om denne eksamensopgave).
    - Det er altså tilladt at at bruge hinanden som kilde for fx hvordan `Observer/Observable` fungerer, hvor man kan finde noget læsbart om Swing `JButton`s, hvad der adskiller et `HashSet` fra an `TreeSet` m.m.
    - Det er *ikke* tilladt at spørge hinanden eller udenforkommende, hvordan en anden har tænkt sig at løse eksamensopgaven.

2. Hver linie kode og tekst, der afleveres skal være produceret af eksaminanden og af eksaminanden alene. Dette gælder ikke den udleverede kode.

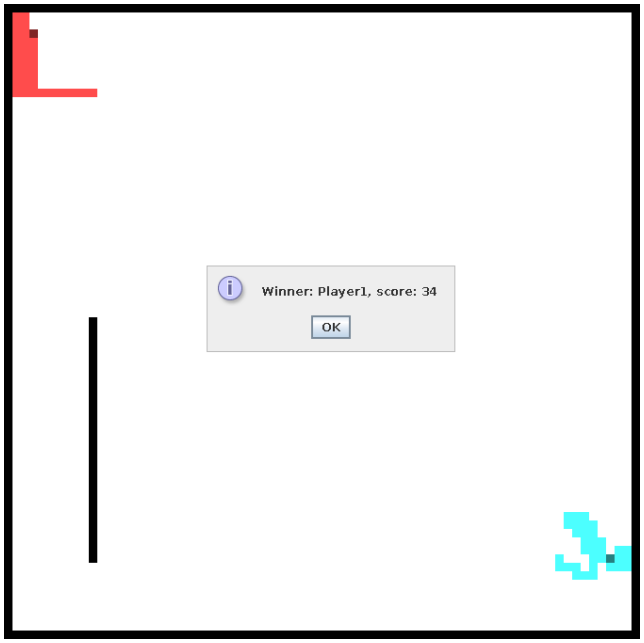
3. Alle kan stille spørgsmål om opgaven til instruktorvagterne eller på diskussionsforum. Generelle svar (uden specifik kode eller specifikke designs for opgaven) på disse, herunder af andre studerende, er tilladt, men skal i denne forbindelse gives på diskussionsforummet for at tilgå alle eksamensdeltagere og således ligestille alle indbyrdes.

4. Alle eksaminander er underlagt KUs eksamensregler i hele perioden fra mandag, kl. 8:00, til efterfølgende fredag kl. 15:00. Det betyder, at snyd, forsøg på snyd og hjælp til at begå snyd (fx ved at hjælpe nogen konkret med løsningen) resulterer i bortvisning fra eksamen samt anmeldelse til studielederen mht. eventuel forelæggelse hos dekanen mhp. yderligere skridt (f.eks. bortvisning fra alle eksaminer i indeværende eksamensperiode). Det gælder for så vidt også ikke-OOPD/studerende, som hjælper en OOPD-eksaminand.

5. Desuden er alle eksaminander også underlagt citatloven og KUs ordensregler i øvrigt, herunder om videnskabelig etisk adfærd. Det betyder, at kilder til viden, der viderebringes, som er væsentlige og ikke har almenviden inden for faget, være det i form af personer, bøger, artikler, eller elektroniske resurser, skal angives, og citater skal markeres som sådan. Bemærk, at det at kopiere en stump kode således kræver citation.

# 3 Opgaver

Opgaven går ud på at færdiggøre et Tron spil. Tron handler om at bevæge sig rundt på en spilleplade uden at støde ind i murene. Hver gang en spiller besøger et nyt felt, bliver dette automatisk til en "mur", så man ikke kan besøge feltet igen. Den person der overlever i længst tid har vundet. Et eksempel på dette ses på Figur 3.1.




**Figur 3.1:** Turkis er kørt ind i sig selv, så rød tager sejren.

Et andet eksempel er Figur 3.2, hvor de to spillereer i tættere kamp og turkis er kommet til at lukke sig selv inde.

I denne version spilles der med mange spillere, hvoraf en kan styres af brugeren med piletasterne. De andre spillere styres af computeren. I den udleverede kode er der allerede én strategi som computeren kan bruge. Denne prøver altid at finde det nabofelt der er "mest åben" – dvs. det felt med flest åbne nabofelter. I løbet af opgaven skal du implementere flere strategier samt andre dele af programmet for at ende op med et færdigt spil.

**Figur 3.2:** Turkis er kommet til at lukke sig inde, og rød er garanteret en sejr.

## Navngivning og ændring af udleveret kode

Det anbefales på det kraftigste at følge opgavens navngivning - både det der er beskrevet i opgaveteksten og de funktioner der er lavet i den udleverede kode. Hvis ikke vil nogle af de udleverede klasser give fejl såfremt de ikke ændres for at tage højde for dette.

Det er tilladt at ændre i den udleverede kode udover opgavetekstens krav. I dette tilfælde skal ændringerne dokumenteres i dit besvarelsesdokument og i java koden.

## Oversigt over udleveret kode

**`control.TronLauncher`** starter spillet og indeholder `main`-metoden.

**`control.GUIController`** kører spillets hændelsesløkke og sørger for at lave en tur i spillet med et fast tidsinterval.

**`model.Game`** repræsenterer spillets nuværende tilstand.

**`model.Board`** repræsenterer spillepladen.

**`model.Direction`** repræsenterer en retning en spiller kan tage.

**`model.Position`** repræsenterer et punkt i $\mathbb{N}^2$ og en position i spillet.

**`model.Player`** er en abstrakt spiller.

**`model.HumanPlayer`** er en menneskelig spiller der er styret ved hjælp af tastaturet.

**`model.ComputerPlayer`** er en computerstyret spillet der bruger én eller flere AI-strategier til at vælge det næste træk.

**`model.GameManager`** holder styr på spillets gang.

**`model.config.IParser`** er en grænseflade til indlæsning af konfiguration.

**`model.config.TextParser`** indlæser konfigurations filter i tekstformat fra filsystemet.

**`model.config.ParserException`** bruges af parsere i tilfælde af fejl.

**model.config.Configuration** indeholder en indlæst spilkonfiguration.

**model.config.LineSegment** beskriver et linjesegment, der bruges til vægge på spillepladen.

**model.strategies.RandomStrategy** er en spillerstrategi der tager et tilfældigt træk der, hvis muligt, ikke slår spilleren ihjel.

**model.strategies.WallHuggerStrategy** er en spillerstrategi der foretrækker at gå langs vægge.

**model.strategies.MostOpenStrategy** er en spillerstrategi der foretrækker at gå mod den position der har flest mulige ledige nabopositioner.

**model.strategies.StrategyFactory** er et factory til at få en liste over tilgængelige spillerstrategier samt en specifik strategi.

**model.strategies.StrategyFactory.Types** er en enum over alle spillerstrategier.

**view.MainWindowFrame** er hovedvinduet i programmet.

**view.MainWindowFrame.GameInput** er en klasse til at håndtere input fra piletasterne.

**view.TronDisplayPanel** er den grafiske repræsentation af spillepladen.

**view.TronColors** indeholder statiske hjælpemetoder til farverne i spillet.

**view.NewGameDialog** er dialogboksen til at indlæse og starte et nyt spil.

**view.NewGameDialog.PlayerPanel** er et panel i `NewGameDialog` der indeholder valgmuligheder for en enkelt spiller.

## Opgave 1: Spillepladen og omgivelserne i Tron

Spillepladen i Tron er formet som et rektangel med dimensioner $w \times h$, hvor visse felter er optaget af mure.

Hvert felt på brættet kan beskrives som et par af koordinater $(x, y)$, hvor felt $(0, 0)$ er det *øverste venstre* hjørne og $(w - 1, h - 1)$ er det nederste højre hjørne.

1. Implementér klassen `model.Position`, der repræsenterer et punkt på spillepladen. Denne skal have to koordinater med tilhørende get-metoder. Yderligere skal den have en metode `getNeighbour` der som input tager en `Direction` og returnerer punktets nabo i den retning.

2. Færdiggør klassen `model.Board` der repræsenterer selve spillepladen. Der ligger allerede et skelet, der specificerer metoderne: `occupyField` skal placere en *mur* på et felt og `isOccupied` skal returnere hvor vidt et felt er tomt eller ej.

## Opgave 2: Spillere i Tron

Den abstrakte klasse `model.Player` repræsenterer en spiller i Tron. En spiller har følgende egenskaber:

- Position og retning. Disse udgør spillerens tilstand på brættet.
- Navn, farve og score. Disse bruges til GUI'en.
- Et flag der angiver om spilleren er død eller ej.

Den abstrakte metode `nextMove` bruges til at informere spillet om hvilken retning denne spiller ønsker at bevæge sig i sit næste træk. Et eksempel på dette kan ses i den udleverede klasse `model.HumanPlayer`, der modellerer en spiller styret af brugerinput (eksempelvis piletasterne på tastaturet).

Vi ønsker også at have computerstyrede spillere i Tron-spillet. Til dette bruger vi strategimønstret.

### Delopgave 2.1: Strategimønster og grænseflade

1. Lav en grænseflade `model.strategies.IStrategy`. Denne skal have en metode `nextMove`, der returnerer en `Direction` og tager et `Game` objekt samt en `Player`.

   `Player`-objektet er den spiller strategien skal foretage træk for.

   `Game` indeholder den nuværende tilstand for spillet.

2. Implementér underklassen `model.ComputerPlayer` der udvider den abstrakte klasse `Player`. Denne skal have en strategi, der kan foretage træk for sig.

### Delopgave 2.2: Strategier

1. Implementér en strategi `model.strategies.RandomStrategy`, der udfører et tilfældigt træk, som *ikke* vil slå spilleren ihjel (dvs. ikke kører ind i en mur). Hvis et sådan træk ikke findes er det op til dig, hvilken retning strategien vil vælge.
2. Implementér en strategi `model.strategies.WallHuggerStrategy`, der altid prøver at kravle langs en mur. Hvis der ikke er en mur at kravle langs er det op til dig hvad strategien gør.
3. * Lav en testplan[1] for `WallHuggerStrategy`. Du behøver *ikke* at teste med modstandere.
4. Implementér testplanen med JUnit-tests.

*Vink: Metoden `Direction.shuffledValues()` returnerer en samling af de forskellige retninger i tilfældig rækkefølge.*

[1] Se den supplerende litteratur.

## Opgave 3: Fil I/O, spilkonfigurationer og spillets udførsel

I Tron repræsenterer vi en konfiguration af en spilleplade med et `Configuration`-objekt. Dette består af:

- Spillepladens bredde
- Spillepladens højde
- En skaleringsfaktor[2]
- En samling af mure på spillepladen
- En samling af mulige startpositioner på spillepladen

Pakken `model.config` indeholder flere klasser til at hjælpe med repræsenteringen og indlæsning af konfigurationer. Bl.a. bruges klassen `LineSegment` til at repræsentere en mur på spillepladen. Klassen `IParser` giver en grænseflade for hvad en parser skal kunne. Dette kunne bruges til en parser, der laver en spilleplade ud fra et billede eller en tekstfil.

En tekstkonfiguration for Tron er udformet således:

```
<width> <height> <scaleFactor>
line: <x1> <y1> <x2> <y2>
player: <x> <y>
```

Først en linje med bredde, højde og skaleringsfaktor som positive heltal. Dernæst en eller flere linjer startende med teksten `line:` og efterfulgt af fire heltal repræsenterende start- og slutpunktet for en mur inden for spillepladen. Til sidst en eller flere linjer med teksten `player:` efterfulgt af $x$ og $y$ koordinat for en mulig startposition på brættet. Bemærk yderligere, at der kan være et vilkårligt antal blanktegn mellem de forskellige ord i inputfilen.

Et eksempel på en konfigurationsfil kan ses herunder:

```
100 100 6
line:   10  10 90 30
line: 10 70   90 90
line:  80 40 70  70
line: 30 30 20 60
player: 25    25
player:   75 75
player:    25  75
player: 75 25
```

### Delopgave 3.1: Indlæsning af konfiguration

1. Lav en klasse `model.config.TextParser`, der implementerer `IParser` og som kan analysere en tekstfil med det specificerede format og skabe et konfigurationsobjekt.

[2]Denne bruges til at tegne spillepladen på skærmen i en opskaleret størrelse. Ved skaleringsfaktor 1 optager hvert felt netop én pixel.

### Delopgave 3.2: Spiloprettelse

1. Lav metoden `createBoard` i `model.GameManager`. Ud fra denne skal der konstrueres et `Board` og spillerne skal tildeles startpositioner. Er der flere startpositioner end spillere, kan der frit vælges hvilke positioner der skal anvendes.
   Husk at spillepladen skal omgives af mure.

*Vink: Metoden `getPositions` i klassen `LineSegment` returnerer samtlige punkter på objektets linje.*

### Delopgave 3.3: Udførelse af spillet

1. Implementér metoden `takeTurn` i klassen `model.GameManager`. Denne skal for hver levende spiller finde det ønskede næste træk og afgøre om det er tilladt. Hvis trækket er tilladt[3], skal den udføre trækket og opdatere brættet. Hvis trækket ikke er tilladt skal spilleren blot fortsætte i sin nuværende retning. Vær opmærksom på at selvom et træk er tilladt kan det godt slå spilleren ihjel.
   Metoden skal også give besked til diverse observers, så de kan opdatere brugergrænsefladen.

## Opgave 4: GUI, Model-View-Controller og sammensætning af program

Den grafiske grænseflade for Tron består af to dele:

- Et "hoved" vindue, `view.MainWindowFrame`, der viser selve spillet. Til dette bruges den udleverede klasse `view.TronDisplayPanel`, som er i stand til at tegne selve spillepladen med de forskellige spillere.
- Et popupvindue, `view.NewGameDialog`, der lader brugeren vælge konfigurationsfil samt hvilke spillere der skal spille. Bemærk at denne kun behøver at kunne håndtere én `HumanPlayer`. Det skal være muligt at vælge færre spillere end konfigurationen har startpositioner.

Et eksempel på hvordan en `NewGameDialog` kunne se ud kan ses på figur 3.3.

For at kunne lave en fornuftig `NewGameDialog` er der implementeret en `PlayerPanel` klasse, der kan bruges til at vælge en strategi for en bestemt spiller. Den kan også bruges til at få et `IStrategy` objekt, der er af den valgte strategi. Denne er en privat inder klasse som ligger i `NewGameDialog` klassen.

### Delopgave 4.1: Dialog til start af nyt spil

1. Færdiggør `view.NewGameDialog`s constructor ved at gøre følgende:

[3] Det eneste ikke tilladte træk, er at går baglæns.


**Figur 3.3:** Eksempel på en `NewGameDialog`.

a) Lav en `JFileChooser` til at vælge konfigurationsfilen. Der skal tages højde for at der vælges en ugyldig fil eller slet ikke vælges en fil.

b) Tilføj en måde at vælge om man vil have en `HumanPlayer` eller ej.

c) Ud fra antal startpositioner i konfigurationen, tilføj et antal `PlayerPanels`[4] til dialogen.

d) Tilføj en *OK* knap til dialogen.

2. Implementér funktionen `actionPerformed` så klassen implementerer `ActionListener` korrekt. Tilføj klassen som listener for ok knappen. Følgende skal ske når der trykkes på ok knappen:

   a) Det afgøres om brugerens input er validt. (bl.a. skal der være mindst 2 spillere).

   b) Spillerobjekterne oprettes ud fra brugerens valg.

   c) Dialogen lukkes.

   Husk, at det skal være muligt at tilgå konfigurationen og spillerne efter dialogen er lukket!

*Vink: Klassen `TronColors` indeholder en hjælpemetode `getColors` der returnerer n farver til brug i spillet. Disse farver laves på en måde der er særligt hensigtsmæssig for GUI'en.*

### Delopgave 4.2: Hovedvindue

1. Tilføj et `TronDisplayPanel` til klassen `MainWindowFrame`.

2. Tilføj et `GameInput`-objekt som `keyListener`.

3. Gør det muligt, at tilføje en `HumanPlayer` som observer[5] for vinduets keyListener (`GameInput` objektet).

[4] `PlayerPanel` er en privat indre klasse i `view.NewGameDialog`

[5] Brug `java.util.Observer` og `java.util.Observable`.

### Delopgave 4.3: Sammensætning af det hele

1. Sæt alting sammen i `control.TronLauncher`. Følgende skal ske for at spillet kan køre:
   a) Lav en `NewGameDialog` og hent konfiguration og spiller ud af den.
   b) Lav en `GameManager`, `TronDisplayPanel`, `MainWindowFrame` og `GUIController`. Husk at kalde `TronDisplayPanels init` funktion for at initialisere billedet.
   c) Initialiser klasserne rigtigt og sæt diverse observers og listeners.
   d) Kald `GUIController.run` for at starte spillet.

## Opgave 5: Spillets design og opbygning

### Delopgave 5.1: Argumentation for designet

1. * Lav CRC kort for `Game`-klassen og argumentér for dens brug. Kunne man have brugt `Board` til det samme?
2. * Argumentér for dit valg af datastruktur til at implementere `model.Board`.

*(Opgavesættet slut)*

# A Litteratur

## A.1 Pensum


- David Barnes og Michael Kölling. *Objects First With Java — A Practical Introduction Using BlueJ*. Pearson Education, 2008.


## A.2 Supplerende litteratur


- Daniel Egeberg. Writing documentation using Javadoc.
- Jaime Niño og Frederick A. Hosch. *Introduction to Programming and Object-Oriented Design Using Java*, side 262–277. Wiley, 2008.
- Tim Ottinger. Meaningful Names. I Greg Wilson og Andy Oram, redaktører, *Beautiful Code*, side 17–30. O'Reilly Media, 2007.
- Alberto Savoia. Beautiful Tests. I Greg Wilson og Andy Oram, redaktører, *Beautiful Code*, side 85–103. O'Reilly Media, 2007.
- Peter Sestoft. *Java Precisely*. MIT Press, 2005.