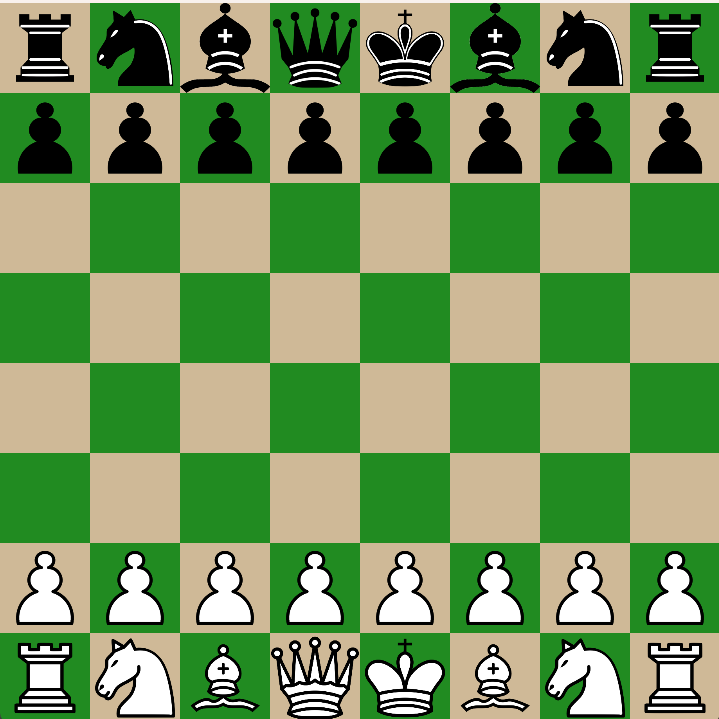
**Programmering B**

EUC Sjælland – HTX Næstved

Skakbot



Udleveret:2024-04-25 Afleveret: 2017-05-17

*Husk at ændre*

Alexander Frederiksen Klasse: 3.C 2023/24

Vejleder: Alexander Frederiksen

Antal sider: xx

Indholdsfortegnelse

[Abstract 2](#_Toc478890406)

[Problemformulering (eller Projektbeskrivelse) 3](#_Toc478890407)

[Problemformulering 3](#_Toc478890408)

[Analyse af børns behov for trafiksimulering 3](#_Toc478890409)

[Krav 3](#_Toc478890410)

[Testbeskrivelse 3](#_Toc478890411)

[Funktionalitet 4](#_Toc478890412)

[Skærmlayout 4](#_Toc478890413)

[Animation 4](#_Toc478890414)

[Funktionalitet i knapperne 4](#_Toc478890415)

[Indtastningsfelter 4](#_Toc478890416)

[Dokumentation 4](#_Toc478890417)

[Udvikling af programmet / beskrivelse af løsningen 4](#_Toc478890418)

[Skærmbilledet 5](#_Toc478890419)

[Detalje / Udformningen af timertick 5](#_Toc478890420)

[Variabler 6](#_Toc478890421)

[Begrænsninger 6](#_Toc478890422)

[Story-board 6](#_Toc478890423)

[Brugervejledning 6](#_Toc478890424)

[Test 6](#_Toc478890425)

[Konklusion 6](#_Toc478890426)

[Perspektivering 6](#_Toc478890427)

[Litteraturliste/kildeoversigt 7](#_Toc478890428)

[Bilag 8](#_Toc478890429)

Bilag 1: Tidsplan

Bilag 2: --

Bilag 3: --

Bilag 4: Logbog

Bilag 5: Programudskrift

# Abstract

Problemet med en skakbot løses i dette projekt med et program, som har skak implementeret samt algoritme som leder efter de bedste træk ledige på bordet.

Programmet kører automatisk uden bruger inputs, dvs. Der er ikke nogle virkelige spillere ud over skakbotten.

Algoritmen til at finde det bedste træk hedder Minimax med Alpha-Beta pruning.

Til test af programmet, sættes algoritmen op mod sig selv.

# Projektbeskrivelse

## Problemanalyse

Nogle ting at tænke på under udviklingen af skak og udviklingen af en skakbot skal man tænke på ydeevne en del, derfor kan man kigge på hvilket sprog man gerne ville bruge. Altså, ville man bruge Python eller C++? Med en skakbot kan man også kigge på hvordan skak kan spilles på det øverste niveau siden at den i teorien nærmest altid ville finde det bedste træk for spilleren. Man kan også kigge på hvordan forskellige algoritmer klarer det op mod hinanden. Måske er den ene algoritme bedre end den anden. For hurtigt at give svar på det første spørgsmål dog, kommer der i dette projekt til at blive programmeret i C++ og der bliver gjort brug af OpenGL. Dette er størst pga. den stærkt forøgede ydeevne af C++ ift. andre sprog som Python.

## Tekniske problemstillinger

* Hvordan kan man render en png med OpenGL?
* Hvordan kan skaks diverse regler og gameplay blive implementeret?
* Hvilke søge-algoritmer ville give mening at bruge til at finde det bedste træk i skak og hvilken er bedst?

## Problemformulering

For at kunne lave en skakbot er der naturligvis forskellige metoder. Man kan bruge en algoritme som Monte Carlo Tree Search eller en anden algoritme som Minimax med Alpha-Beta pruning. En anden metode man kunne bruge er machine-learning hvor en AI kunne lærer at spille skak bedre og bedre over tid. I dette projekt bliver der dog kun kigget på søge-algoritmer.

Målet med dette projekt er derfor:

* Udvikle et program, som kan vise et skakbræt og have en skakbot som spiller spillet.

Der vil her fokuseres på hvordan man kan generere alle træk i skak og hvordan den valgte algoritme virker, for at kunne opnå det ses på:

* Andre personers implementeringer af skak og skakrobotter studeres og forstås.
* Hvilke algoritmer kan implementeres?
* Krav opstilles til projektet

## Projektafgrænsning

Der kommer ikke til at blive noget aktuelt spilning af skak fra en réel spiller. Dvs. At en bruger af programmet kun ville kunne se på hvad der sker i spillet. Evt. en brugerdefineret tilpasning af skakbrættet fra start kunne være en mulighed. En anden ting som tidligere er nævnt er hvordan machine learning ikke er fokuset her. Derfor ville machine learning ikke blive sat op mod søge algoritmer for at finde ud af hvilken er bedre. Der kommer nok heller ikke til at være en implementering af mere end én algoritme grundet mangel af tid.

## Kravspecifikationer

Krav:

* Generering af alle lovlige træk i skak
* Visualisering af skakbrættet og dens brikker i realtid
* En algoritme som kan spille spillet.

Udvide krav, der udføres hvis der bliver tid:

* Brugerdefineret skakbræt i starten af spillet.
* En anden algoritme som skal op mod den første algoritme

## Testbeskrivelse

Følgende skal virke, når minimumskravene er opfyldt:

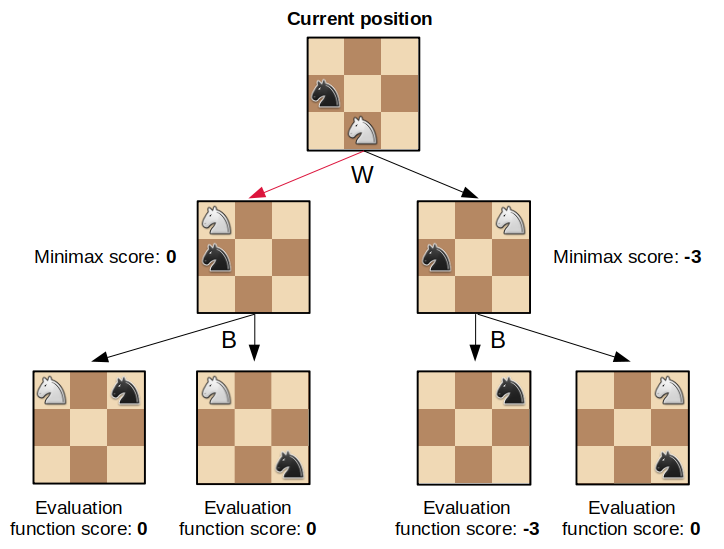
* Når spillet starter skal skakbotten gå igang med det samme, hvid trækker først.
* Med trækket så bliver brikken også rykket på skærmen.
* Sort svarer osv. Indtil spillet slutter

# Funktionalitet

## Skærmlayout

Skærmen skal bare være skakbrættet med skakbrikkerne på. Brikkerne bliver rykket på skærmen ift de træk som algoritmen finder.

## Algoritmen

Algoritmen der bliver brugt hedder Minimax med Alpha-Beta pruning. Algoritmen handler om at gå gennem alle lovlige træk og gå ned i dybden på hvert af dem. Man starter med max for at finde træk som går til fordel til spilleren og så min for at finde træk som går imod spillerens fordel osv. Hvor man til sidst evaluere bordet ved den laveste dybde for at finde ud af hvilken af trækkene leder til den position af bordet som giver den bedste evaluering tilbage. En visualisering af algoritmen kan ses på Figur 1. Alpha-Beta pruning er en måde at optimisere algoritmen på. Det handler om at formindske antallet af noder, eller i denne sag, træk som algoritmen evaluere ved at bruge to værdier, alpha og beta, under evalueringen.

*Metoden ”Trinvis forbedring” kan med fordel bruges (se* [*https://programmering.systime.dk/?id=p147&L=0*](https://programmering.systime.dk/?id=p147&L=0)*).*

Figur 1 Visualisering af Minimax algoritmen

# Dokumentation

## Udvikling af programmet

Til at starte med ville jeg generere skakbordet. Til at gøre dette bruger jeg OpenGL til at tegne firkanter på skærmen med forskellige farver. Det jeg har gjort er at lave en klasse til selveste skakbordet og så en klasse til hvert brætplads. Med OpenGL kan du tegne Quads, altså firkanter, ved at beregne hvor hvert af de fire hjørner af firkanten skal være, og med OpenGL går koordinaterne på skærmen fra -1 til 1. Dvs. At vi kan opdele 2 med 8 for at finde ud af mellemrummet mellem alle punkterne. Med dette kører vi bare et for-loop for at lave brættet hvor vi beregner de fire hjørner for hvert skakplads. Til sidst tegner vi det til skærmen med skiftende farve hver gang for at lave det klassiske hvid/sort skak gitter.

For at kunne render skakbrikkerne har jeg også lavet en klasse til dem som indeholder skakbrikkens type, side og så en flags struct som indeholder information om hvis der er noget specielt ved brikken såsom at brikken kan rykke dobbelt som bonde. Hvert skakplads kommer til at indeholde en skakbrik klasse, selvom at hvis brikken har en tom side og en tom type så bliver den ikke vist på skærmen og algoritmen kommer til at ignorere den. For at kunne vise brikkerne på skærmen bruger jeg et eksternt bibliotek der hedder stb\_image til at putte png billeder på skærmen. Vi kører alle skakpladser igennem og kigger på dens skakbrik og ift. til dens type og side viser vi f.eks. en sort konge eller en hvid bonde.

Efter jeg havde lavet skakbrættet og skakbrikkerne skulle jeg nu begynde at lave algoritmen, men før jeg kan lave algoritmen ordentligt, skal jeg kunne generere alle lovlige træk på bordet. Som hjælp til dette kommer jeg til at forudberegne de træk der er mulige på hver position for hver skakbrik. Til det brikker der bliver kaldt ”sliding pieces”, altså dronningen, tårnet og løberen. For at gøre det finder vi først ud af offsettet på hver af de 8 retninger. Siden vi går fra nederst til venstre og går 1 index op hver gang vi går højre. Kommer offsetsne på hver af de 8 retninger være: 8, -8, -1, -1, 1, 7, -7, 9, -9Beregner jeg hvor mange pladser der er ud til enden af bordet i alle 8 retninger. Til bønderne skal der beregnes alle mulige træk

## Skærmbilledet

…

## Detalje / Udformningen af lysskift

*Se eksempel på rutediagram og pseudokode:* [*https://programmering.systime.dk/?id=c1002&L=0*](https://programmering.systime.dk/?id=c1002&L=0)[*https://programmering.systime.dk/?id=c1016&L=0*](https://programmering.systime.dk/?id=c1016&L=0)

## Variabler

…

## Begrænsninger

…

## Story-board

…

## Brugervejledning

…

*Se også eksempel på:* [*https://programmering.systime.dk/?id=c1132&L=0*](https://programmering.systime.dk/?id=c1132&L=0)

# Test

Der er løbende foretaget tests af programstumperne, undervejs.

## Den endelige egentest

# Konklusion

# Perspektivering

# Litteraturliste/kildeoversigt

# Bilag

Bilag 1: Tidsplan

Bilag 2: --

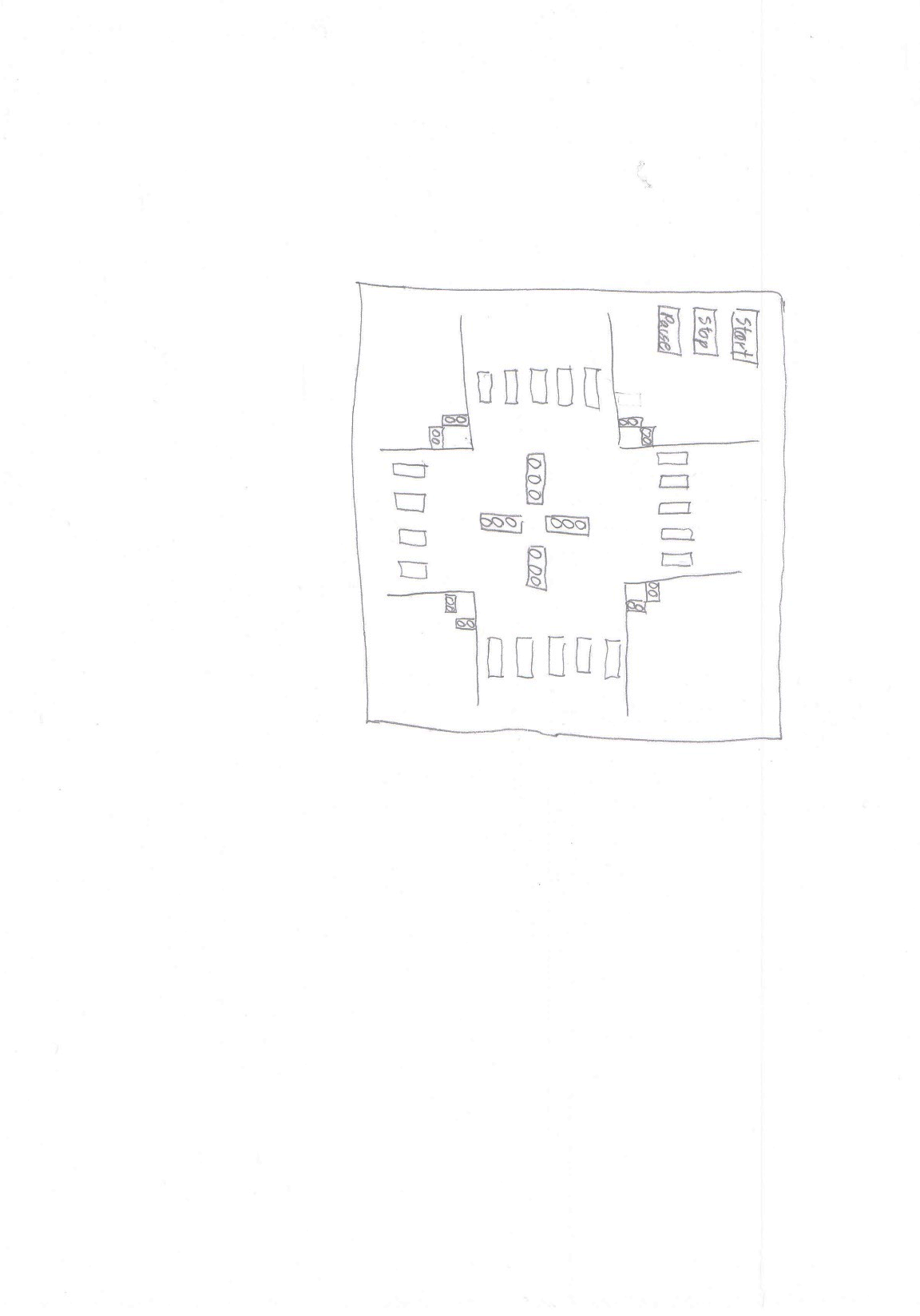
Bilag 3: --

Bilag 4: Logbog

Bilag 5: Programudskrift

**Bilag 1: Logbog**

Dag 1:

Brainstormede over hvilket problem, der kunne løses.

Skrev problemformuleringen.

Tegnede de første skitser over lyskryds.

…

Dag 2:

…

*Note: Se detaljer I template for logbog*

**Bilag 2: Programkode**

// Need G4P library

import g4p\_controls.\*;

Boolean doRun = false;

int state = 0;

final int maxState = 6;

float cirCarNSRedAlpha = 20; //rødt lys slukket

float cirCarNSYellowAlpha = 20; //gult lys slukket

float cirCarNSGreenAlpha = 255; //grønt lys tændt

…

…

public void **setup**(){

  size(480, 320, JAVA2D);

…

  frameRate(1); // draw kører en gang i sekundet

  …

}

public void **draw**(){

  background(230);

  if (doRun) {

    lightShift();

  }

…

}

…

public void lightShift() {

  stroke(50);

  switch(state) {

    case 0:

      cirCarNSGreenAlpha = 255;  //grønt lys tændes

      cirMannNSGreenAlpha = 255; //grønt lys tændes

      cirCarOWYellowAlpha = 20;  //gult lys slukkes

      cirCarOWRedAlpha = 20;     //rødt lys slukkes

      cirMannNSRedAlpha = 20;    //rødt lys slukkes

      break;

    case 1:

     …

      break;

    case 2:

     …

      break;

  }

  if (state < maxState) {

    state += 1;

  } else {

    state = 0;;

  }

  //BilNordSyd

  fill(255,0,0,cirCarNSRedAlpha);

  circle(200, 50, 20);

  fill(255,255,0,cirCarNSYellowAlpha);

  circle(200, 75, 20);

  fill(0,255,0,cirCarNSGreenAlpha);

  circle(200, 100, 20);

…

}

*Hvis man vil undgå de røde understregninger, så marker hele koden i Word og vælg:* ***Gennemse*** *>* ***Sprog*** *>* ***Angiv korektursprog*** *>   
Sæt tjek ved* ***Kontroller ikke stavning eller grammatik***

*Hvis man vil have en baggrundsfarve for koden, så marker hele koden og vælg:* ***Hjem*** *>* ***Afsnit*** *>* ***Skygge*** *Vælg en passende farve.*