

DM34 - 1. Obligatorisk opgave  
Dilemma spillet

Jacob Aae Mikkelsen 191076  
kok04

April 2005

# Kapitel 1

## Resumé

Denne rapport dokumenterer udviklingsforløbet og afprøvningen af et spil "Dilemma". Spillet går ud på at opbygge en tillid i byttehandel, hvor der også kan snydes. Der er implementeret forskellige spillere, ved hjælp af nedarving og polymorfi.

# Indhold

<b>1 Resumé</b>	<b>1</b>
<b>2 Indledning</b>	<b>3</b>
<b>3 Kravspecifikation</b>	<b>4</b>
<b>4 Design</b>	<b>5</b>
4.1 Udviklingsstrategi . . . . .	5
4.2 Klasseopbygning . . . . .	5
<b>5 Kok04 strategien</b>	<b>6</b>
<b>6 Implementation</b>	<b>7</b>
6.1 High score tabel . . . . .	7
6.2 Sortere tabellen . . . . .	7
6.3 Fejl afværgning . . . . .	8
<b>7 Afprøvning</b>	<b>9</b>
7.1 Test . . . . .	9
7.2 Mulig udvidelse . . . . .	10
7.3 Evaluering af kodens kvalitet . . . . .	10
<b>8 Konklusion</b>	<b>12</b>
<b>A Testkørsels udskrifter</b>	<b>14</b>
A.1 samarbejde . . . . .	15
A.2 udnytte . . . . .	16
<b>B Kilde kode</b>	<b>17</b>
B.1 Main.java . . . . .	17
B.2 Gui.java . . . . .	17
B.3 Tabel.java . . . . .	23
B.4 Spiller.java . . . . .	25
B.5 Moede.java . . . . .	27
B.6 RandomPlayer.java . . . . .	28
B.7 Udnytte.java . . . . .	29
B.8 Samarbejde.java . . . . .	29
B.9 Kok04.java . . . . .	30

## Kapitel 2

### Indledning

Spillet Dilemma, som denne rapport omhandler, går i alt sin enkelthed ud på at to spillere mødes og udveksler en vare, med mulighed for at aflevere varen, eller at snyde modstanderen. Hvis begge snyder, opnås ingen point, og da én spiller nok hurtigt bliver træt af at blive snydt, er det smarteste at opbygge en tillid, spillerne imellem, hvorved spillerens hukommelse kan benyttes. Rapporten her er udarbejdet af Jacob Aae Mikkelsen ("Kokken") i faget DM34, foråret 2005. Det er i forbindelse med den første afleveringsopgave, der tæller som en del af eksamen.

Denne udgave af Dilemma er lavet med en simpel grafisk brugerflade, seks spillere, hvoraf én er interaktiv, så dette program er et ét personers spil, og hver spiller mødes 10 til 40 gange. Der er i opgaven lagt op til en større ekstern turnering, der har spillet ind, ved udarbejdelsen af strategien. Der er endvidere kort beskrevet de umiddelbare muligheder for at forbedre spillet, og givet en vurdering af kodens kvalitet.

## Kapitel 3

# Kravspecifikation

Der er udtrykkeligt blevet stillet disse krav til opgaven

- Et Dilemma spil med 6 spillere
- De seks spiller skal være:
  - Interaktiv, brugeren bestemmer
  - Altid udnytte
  - Altid samarbejde
  - Random, tilfældigt udvalgt
  - Edmund (forelæserens) strategi
  - Vores egen, navngivet med vores IMADA brugernavn
- Designbeskrivelse af struktur og hovedprogram
- Specifikation og design af vores egen strategi
- Programudskrift
- Afprøvning
- Resultat og konklusion af afprøvningen

Ud over de i opgaven stillede krav er der udarbejdet en grafisk brugerflade.

# Kapitel 4

## Design

### 4.1 Udviklingsstrategi

Da der allerede var standardiserede klasser til spiller og møde klasserne, samt en ret udførlig beskrivelse af opgaven, valgtes vandfaldsmodellen som udviklingsstrategi. Der var enkelte småfejl, der blev fundet under testning, som så sendte udviklingen tilbage et skridt, men ellers er modellen fulgt med succes.

### 4.2 Klasseopbygning

De seks spillere nedarver alle fra Spiller klassen, ligesom de benytter Moede klassen. I kraft af den enumererede type "Handling" der er placeret i denne klasse, har den mange koblinger til de andre klasser, hvilket til dels kunne være undgået, hvis "Handling" havde været lagt i spiller. Den største klasse i programmet er uden tvivl Gui. Denne klasse står for den grafiske brugerflade, og selve afviklingen af spillet. Selve afviklingen af spillet kunne være placeret i en klasse for sig selv, men da den ikke er så omfattende endda, var den behagelig at have i samme klasse som de tekstfelter denne del hele tiden opdaterer, derfor er denne løsning foretrukket. Gui klassen gør brug af Tabel klassen, der laver de to evalueringsskemaer. Der er i opgaven implementeret en metode i Spiller klassen, der kan sammenligne to spillere, men da Tabel klassen løbende får informationerne om pointene, efter hvert møde, er det fortrukket at den benytter sig af sine egne oplysninger, frem for at koble til Spiller klassen. Der er desuden lavet en Main klasse, der kun har et formål, nemlig at lave og initialisere den første instans variabel, og dermed starte den grafiske brugerflade.

## Kapitel 5

# Kok04 strategien

Personligt, tror jeg på det bedste i folk, også at man kan ændre sig til det bedre. Dette er grundidéen med kok04 strategien. Den er opbygget på de tre seneste møder, fordelt med halvdelen på det seneste, to sjettedele på det næstsidste og én sjettedel på det tredjesidste. Resten af historien er uden betydning. I dette spil med flere ukomplicerede strategier, klarer den sig fornuftigt, men taber selvfølgeligt hver gang mødet med strategien der kun udnytter. Det er til gengæld ikke med ret mange point, og derfor tror jeg kok04 har en bedre chance i en turnering, med mange spillere, der har implementeret forskellige strategier, men det må jo stå sin prøve senere.

# Kapitel 6

## Implementation

### 6.1 High score tabel

Der er forskellige måder at bygge tabeller op på, men her er valgt en enkel måde, der bygger på string-concatenation. Tabel klassen arbejder uafhængigt af spiller klassen, hvilket giver en bedre struktur med mindre kobling. Specielt den sorterede high-score tabel benytter et lidt naivt design, der dog virker uden problemer.

```
totalListe.add(formatNumber(total)+ " " + nameTable[i]);
```

Spillerne med point Forrest, bliver indsat i en Arrayliste, men med pointene Forrest med foranstillede nuller op til længden 7. Her ses tydeligt at denne tilgang ville få problemer med flere point end 9999999, men da dette svarer til maksimalt point udbytte i  $(99999999 / 4)$  møder 2500000 møder, og med 6spillere skal hver spiller så mødes 41666666 gange. Dette er ikke muligt, og derfor er der ingen problemer.

### 6.2 Sortere tabellen

Da der er foran stillede nuller til en fast længde, kan listen sorteres ganske enkelt efter simpel alfabetisk orden. Til dette formål benyttes javas egen funktion til at sortere liste. Da den så ender i modsat rækkefølge, concateneres strengen ved at tilføje nye dele før det eksisterende i strengen

```
public String highScoreTabel()
{
    Collections.sort(totalListe);
    String highScore = "";
    Iterator it = totalListe.iterator();
    for(int i=1; it.hasNext() ; i++) {
        highScore = "" + (7-i) + it.next() + "\n" + highScore;
    }

    highScore = "High Score Liste - Hall of Fame\nPlads Point Spiller\n" + highScore;
    return highScore;
}
```

### **6.3 Fejl afværgning**

Hvis spilleren fortsat trykker på samarbejde eller udnytte knapperne efter spillet er slut, udnyttes den ArrayOutOfBoundsException exception der opstår til at skrive en ny tekst i det lille tektfelt over knapperne ved de følgende catch:

```
catch(Exception e)
{
    count.setText("\n  Tryk på \n \"Nyt spil\"");
}
```

# Kapitel 7

## Afprøvning

### 7.1 Test

I kraft af at programmet er lavet med en grafisk brugerflade, er der kun tre knapper, og ingen indtastning fra brugeren, der jo kan modtage alverdens input, og derfor også skal grundigt testes. ”Start nyt spil” knappen, starter et nyt spil, også uanset om man ikke er færdig med det spil. Dette er der intet gjort for at forhindre, og kan derfor ikke betegnes som en bug, nærmere en feature, idet der jo heller ikke kommer nogen irriterende dialogbokse med ”Er du sikker” beskeder. Når spillet er startet, giver udnytte/samarbejde knapperne informationen videre til spillet, men under afprøvning, konstateredes det at når spillet var slut, udløste det en exception, at trykke på det. Denne fejl er rettet med en try/catch, og resulterer i stedet for med en meddeelse om at trykke på ”nyt spil”.

Med hensyn til de enkelte strategier, afprøves ved hjælp af den interaktive, ved to testkørsler, én der samarbejder med alle i 10 spil og en der udnytter mod alle i 10 spil.

#### 7.1.1 Samarbejde i 10 spil

Idet vi samarbejder i 10 spil, skulle vores resultater minde meget om det samarbejde spilleren opnår. Det vil sige at:

- Mod udnytte forventes -10
- Mod samarbejde forventes 20
- Mod Kok04 strategien/Kokkens forventes 20
- Mod Random forventes at samarbejde ca 5 gange og blive udnyttet ca 5 gange, men er ikke nem at teste på.
- Udnytte mod ”Dig selv” opnår 40
- Samarbejde mod ”Dig selv” opnår 20
- Kokken mod ”Dig selv” opnår 20

Disse resultater kan ifølge bilag A.1. bekræftes. Ud fra tabellen testes også total scoren, og konstaterer at:

## 7.2. MULIG UDVIDELSE

---

20-10+20+20+11 faktisk giver 61  
20-4+20+20+3 faktisk giver 59  
40+15+40+12+28 faktisk giver 136  
(Resten er også adderet og kontrolleret)

Slutteligt konstateres at de samlede totaler er de samme i begge skemaer, samt at Samarbejde og ”Dig selv” kun er adskilt af de tre point der er i forskel mod Random Denne del af testen er altså godkendt, uden konstaterede fejl.

### 7.1.2 Udnytte i 10 spil

Idet vi udnytter i alle spil, forventes resultater der skal ligne resultaterne fra Udnytte spilleren, det vil sige:

- Mod udnytte forventes 0
- Mod samarbejde forventes 40
- Mod Kok04 strategien/Kokkens forventes 12 (Kun point i de tre første runder)
- Mod Random forventes at samarbejde ca 5 gange og blive udnyttet ca 5 gange, men er ikke nem at teste på.
- Udnytte mod ”Dig selv” opnår 0
- Samarbejde mod ”Dig selv” opnår -10
- Kokken mod ”Dig selv” opnår -3

Disse resultater kan verificeres, se bilag A.2. Ud fra tabellen testes også total scoren, og konstaterer at:

16+40+12+24 giver 92  
-4-4+20+20+12 giver 44  
(Resten er også adderet og kontrolleret)

Også her konstateredes at de samlede totaler er identiske i de to skemaer, og at vi opnår de helt samme resultater som Udnytte spilleren. Vi finder i denne test heller ingen fejl.

## 7.2 Mulig udvidelse

Der kunne implementeres en brugsanvisning til spillet, der kunne hentes fra menulinien. Dette er udeladt, da reglerne er meget simple, og dem der stifter bekendskab med spillet kender dem.

## 7.3 Evaluering af kodens kvalitet

Som i ethvert projekt, er der jo positive og negative ting. Ting der er positive, er at programmet overholder de retningslinier for pæn kode (Coding Guidelines) som Michael Kölling beskriver[1]. Dette gør at koden er til at overskue, og der er lavet kommentarer til alle metoder, også i spiller klasssen, der var mangelfuld,

### *7.3. EVALUERING AF KODENS KVALITET*

---

i den udleverede version. Af negative ting kan nævnes at der er en del af koden i Spiller klassen der ikke benyttes, men der ikke kan ændres, grundet krav i opgaven om kompatibilitet.

# Kapitel 8

## Konklusion

Der er i processen løst og besvaret de stillede opgaver. Programmet virker efter hensigten, og de fejl der dukkede op under afprøvningen er rettet. Af personlige erfaringer, kan konkluderes at tidsplaner kan snyde; når en platform skal opgraderes som i dette tilfælde JAVA fra 1.4 til 5, kan det tage nogen tid at få compileren til at køre ordenligt. Der er ganske givet også klasser i JAVA, der er lavet til den slags skemaer, som voldte problemer i Tabel klassen, men denne fremgangsmåde virker, omend den er møjsommelig.

# Litteratur

- [1] “Objects First with JAVA - A practical introduction using BlueJ (2nd Edition)”, Michael Kölling, David J. Barnes, Pearson Educational Limited 2004

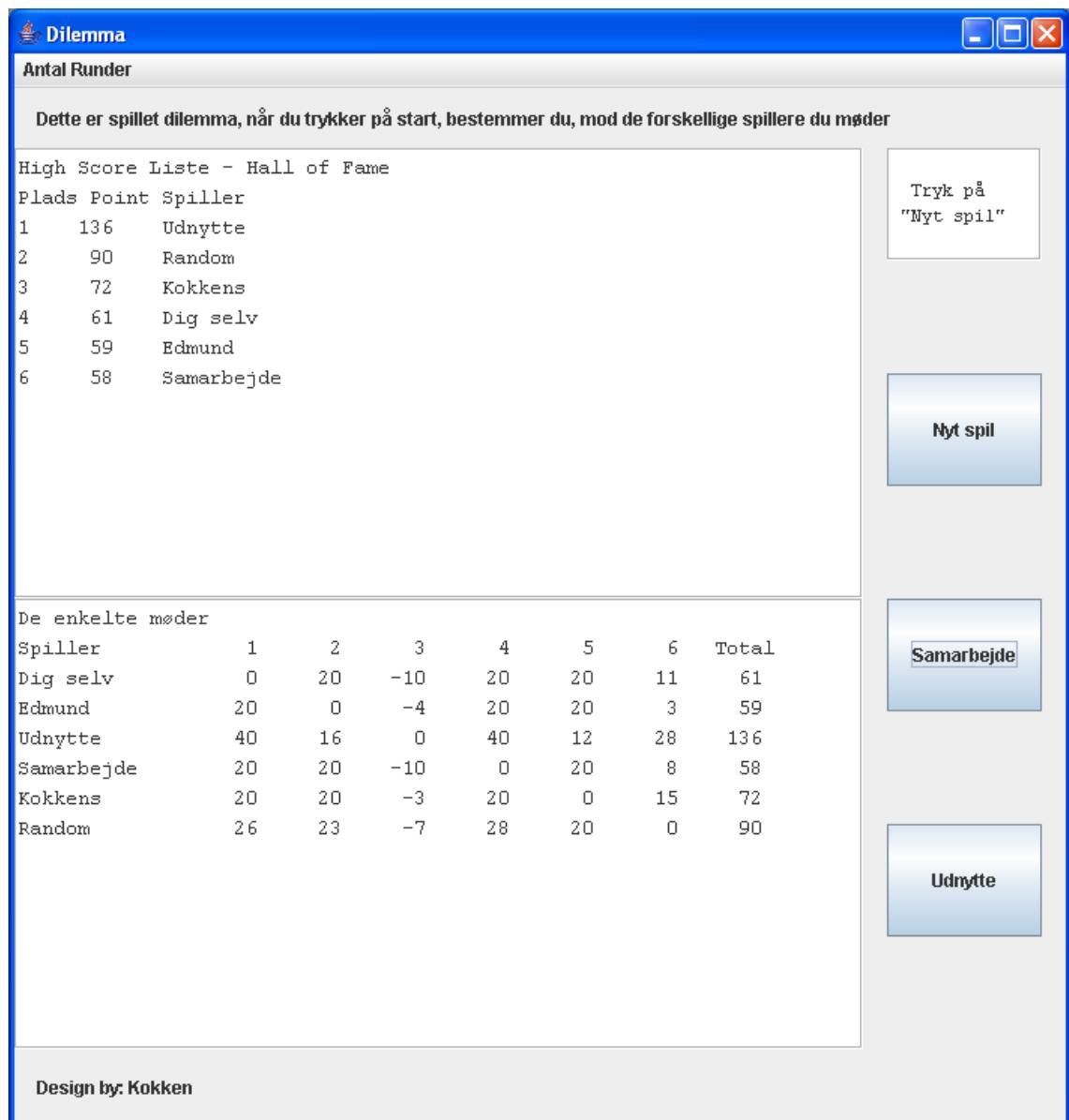
## Bilag A

### Testkørsels udskrifter

## A.1. SAMARBEJDE

### A.1 samarbejde

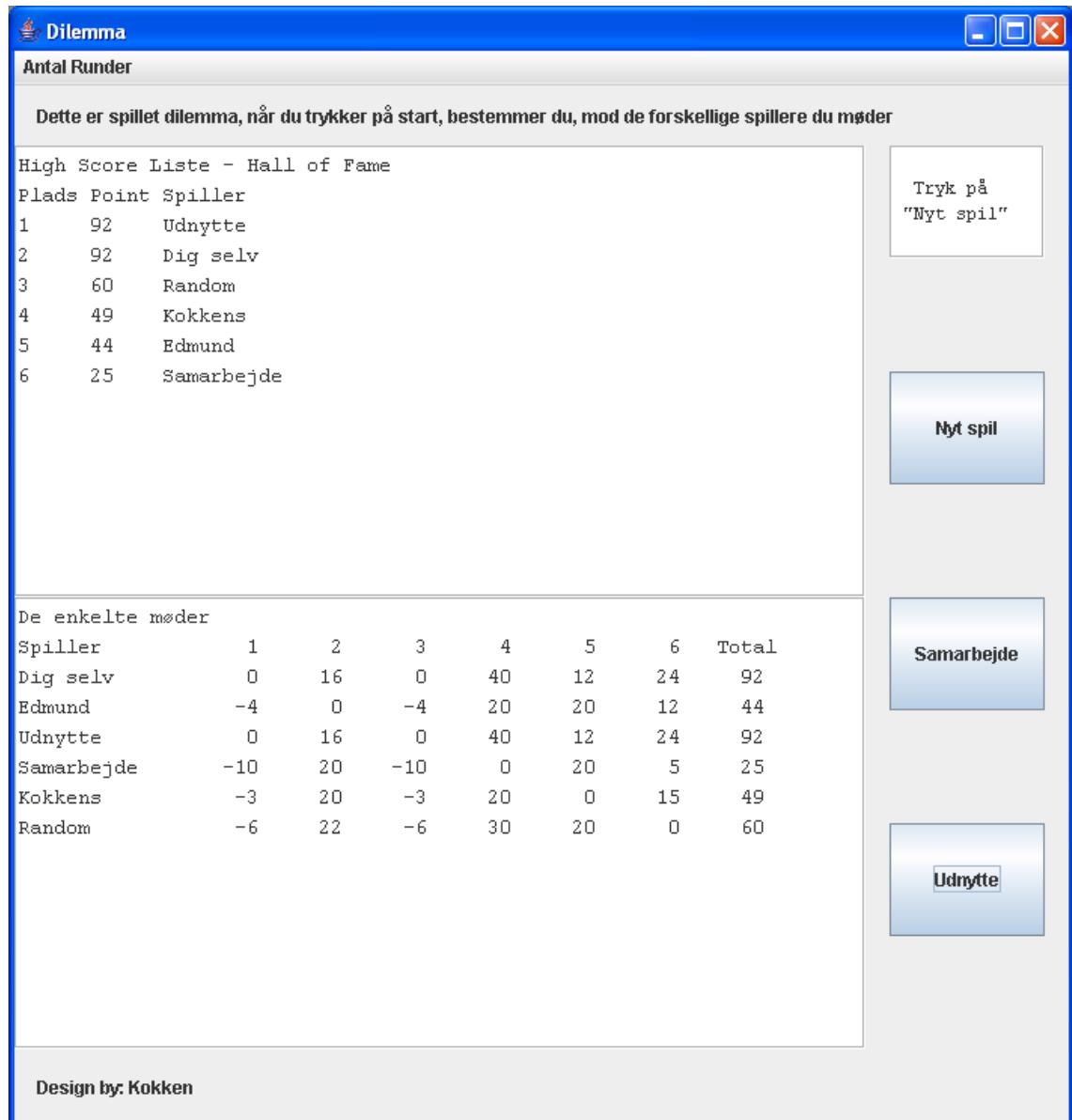
Figur A.1: 10 runder med kun samarbejde



## A.2. UDNYTTE

### A.2 udnytte

Figur A.2: 10 runder med kun udnytte



# Bilag B

## Kilde kode

### B.1 Main.java

```
1 public class Main {  
2     ***  
3     * Main class for the game Dilemma, initializes the  
4     * Graphical user interface  
5     *  
6     */  
7     public static void main(String [] args) {  
8         Gui gui = new Gui();  
9     }  
10 }  
11 }  
12 }
```

### B.2 Gui.java

```
1 import java.awt.BorderLayout;  
2 import java.awt.Container;  
3 import java.awt.Dimension;  
4 import java.awt.Font;  
5 import java.awt.GridLayout;  
6 import java.awt.event.ActionEvent;  
7 import java.awt.event.ActionListener;  
8 import java.util.ArrayList;  
9  
10 import javax.swing.JButton;  
11 import javax.swing.JFrame;  
12 import javax.swing.JLabel;  
13 import javax.swing.JMenu;  
14 import javax.swing.JMenuBar;  
15 import javax.swing.JMenuItem;  
16 import javax.swing.JPanel;  
17 import javax.swing.JTextArea;  
18 import javax.swing.border.EmptyBorder;  
19 import javax.swing.border.EtchedBorder;  
20
```

## B.2. GUI.JAVA

---

```
21  /**
22  * @author Kokken
23  *
24  * This class creates the graphical user interface for the application
25  * And controls the game action
26  */
27 public class Gui {
28     private JFrame frame;
29     private JTextArea textArea;
30     private JTextArea resultArea;
31     private JTextArea count;
32     private ArrayList<Spiller> spillere;
33     private int antal;
34     private int spillerNummer;
35     private int rounds;
36     private Tabel tabel;
37
38     /**
39      * Constructor of the GUI
40      * Initializes the fields , and
41      * Calls the method makeFrame to display the GUI
42      */
43     public Gui() {
44         frame = new JFrame("Dilemma");
45         textArea = new JTextArea();
46         resultArea = new JTextArea();
47         count = new JTextArea();
48         spillere = new ArrayList<Spiller>();
49         rounds = 11;
50
51         makeFrame();
52         newGame();
53     }
54
55
56     /**
57      * This methods creates the GUI display ,
58      * and sets up a listener to the buttons
59      */
60     private void makeFrame() {
61
62         //The frame , and its properties
63         Container contentPane = frame.getContentPane();
64         contentPane.setLayout(new BorderLayout());
65         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
66
67         //The menu bar in the top op the frame , and its operations
68         JMenuBar menubar = new JMenuBar();
69         frame.setJMenuBar(menubar);
70         JMenu countMenu = new JMenu("Antal\u00c5Runder");
71         menubar.add(countMenu);
72
73         JMenuItem ten = new JMenuItem("10\u00c5runder");
74         countMenu.add(ten);
```

```
75     ten.addActionListener(new ActionListener() {
76         public void actionPerformed(ActionEvent e) {
77             setRounds(10);
78             newGame();
79         }
80     });
81
82     JMenuItem twenty = new JMenuItem("20_runder");
83     countMenu.add(twenty);
84     twenty.addActionListener(new ActionListener() {
85         public void actionPerformed(ActionEvent e) {
86             setRounds(20);
87             newGame();
88         }
89     });
90
91     JMenuItem thirty = new JMenuItem("30_runder");
92     countMenu.add(thirty);
93     thirty.addActionListener(new ActionListener() {
94         public void actionPerformed(ActionEvent e) {
95             setRounds(30);
96             newGame();
97         }
98     });
99
100    JMenuItem forty = new JMenuItem("40_runder");
101    countMenu.add(forty);
102    forty.addActionListener(new ActionListener() {
103        public void actionPerformed(ActionEvent e) {
104            setRounds(40);
105            newGame();
106        }
107    });
108
109 //The instructionin the top of the screen
110 JLabel label = new JLabel("_____Dette_er_spillet_dilemma,_når_du_trykker_
111     på_start,_bestemmer_du,_mod_de_forskellige_spillere_du_møder");
112 label.setPreferredSize(new Dimension(700,40));
113 contentPane.add(label, BorderLayout.NORTH);
114
115 //the large text areas to the left
116 Container textAreaContainer = new Container();
117 textAreaContainer.setLayout(new GridLayout(0,1));
118
119 textArea.setPreferredSize(new Dimension(500,300));
120 textArea.setBorder(new EtchedBorder());
121 textArea.setFont(new Font("Monospaced", 0, 14));
122 textAreaContainer.add(textArea);
123
124 resultArea.setPreferredSize(new Dimension(500,300));
125 resultArea.setBorder(new EtchedBorder());
126 resultArea.setFont(new Font("Monospaced", 0, 14));
127 textAreaContainer.add(resultArea);
```

## B.2. GUI.JAVA

---

```
128     contentPane.add(textAreaContainer, BorderLayout.CENTER);
129
130     //The buttons and the small textarea to the right
131     JPanel buttonPanel = new JPanel();
132     buttonPanel.setBorder(new EmptyBorder(0,16,0,16));
133     buttonPanel.setLayout(new GridLayout( 0 , 1 ));
134
135     JButton startButton = new JButton("Nyt_spil");
136     startButton.addActionListener(new ActionListener() {
137         public void actionPerformed(ActionEvent e) {
138             newGame();
139         }
140     });
141
142     JButton samarbejdeButton = new JButton("Samarbejde");
143     samarbejdeButton.addActionListener(new ActionListener() {
144         public void actionPerformed(ActionEvent e) {
145             playGame(Moede.Handling.sam);
146         }
147     });
148
149     JButton udnytteButton = new JButton("Udnytte");
150     udnytteButton.addActionListener(new ActionListener() {
151         public void actionPerformed(ActionEvent e) {
152             playGame(Moede.Handling.udn);
153         }
154     });
155
156     count.setPreferredSize(new Dimension(50,50));
157     count.setBorder(new EtchedBorder());
158     count.setFont(new Font("Monospaced", 0, 12));
159     buttonPanel.add(count);
160
161     JLabel empty0 = new JLabel();
162     buttonPanel.add(empty0);
163     buttonPanel.add(startButton);
164     JLabel empty1 = new JLabel();
165     buttonPanel.add(empty1);
166     buttonPanel.add(samarbejdeButton);
167     JLabel empty2 = new JLabel();
168     buttonPanel.add(empty2);
169     buttonPanel.add(udnytteButton);
170     JLabel empty3 = new JLabel();
171     buttonPanel.add(empty3);
172
173     contentPane.add(buttonPanel, BorderLayout.EAST);
174
175     //The bottum textlabel
176     JLabel labeldown = new JLabel("Design by: Kokken");
177     labeldown.setPreferredSize(new Dimension(500,50));
178     contentPane.add(labeldown, BorderLayout.SOUTH);
179
180     frame.pack();
181     frame.setLocationRelativeTo( null );
```

```
182         frame.setVisible(true);
183     }
184 
185     /**
186      * This method initializes the Dilemma game for a
187      * new game, clearing all players memory, and loading them
188      */
189     private void newGame() {
190         antal = 1;
191         spillerNummer = 1;
192         tabel = new Tabel();
193 
194         spillere.clear();
195         spillere.add(new Kok04("Dig_selv"));
196         // "Dummy" player, to fill the
197         // space in the list
198         spillere.add(new Edmund("Edmund"));
199         spillere.add(new Udnytte("Udnytte"));
200         spillere.add(new Samarbejde("Samarbejde"));
201         spillere.add(new Kok04("Kokkens"));
202         spillere.add(new RandomPlayer("Random"));
203 
204         count.setText("\nDu kan nu starte");
205         textArea.setText("Vælg nu om du vil samarbejde eller udnytte");
206         resultArea.setText("Resultater");
207     }
208 
209     /**
210      * Controls the gameplay, once it has started
211      */
212     private void playGame(Moede.Handling A)
213     {
214         //The player namesis arranged in the table
215         for(int i=0; i < spillere.size(); i++) {
216             tabel.addPlayerName(i, spillere.get(i).getName());
217         }
218 
219         //This part controls the interactive player
220         if(antal % rounds != 0 && spillerNummer < 6)
221         {
222             count.setText("\nAntal Møder\n" + antal);
223             Moede.Handling B = spillere.get(spillerNummer).getAction();
224             Moede moedeA = new Moede(A,B);
225             Moede moedeB = new Moede(B,A);
226 
227             spillere.get(spillerNummer).saveMeet(moedeB);
228             spillere.get(0).saveMeet(moedeA);
229 
230             String text = "+spillere.get(spillerNummer).getName():"
231             "+spillere.get(spillerNummer).score()+"\n"+spillere.get(0).getName():"
232             "+spillere.get(0).score() + "\nSeneste møde: "
233             moedeA.toString();
```

```
232         resultArea.setText(text);
233
234         antal++;
235     }
236     else
237     {
238         antal=1;
239         try{
240             tabel.addResult(0,spillerNummer,spillere.get(0).score());
241             tabel.addResult(spillerNummer, 0 ,
242                             spillere.get(spillerNummer).score());
243             spillere.get(0).clearMemory();
244             spillere.get(spillerNummer).clearMemory();
245         }
246         catch(Exception e)
247         {
248             count.setText("\n\nTryk på\nNy spil");
249         }
250         spillerNummer++;
251         if(spillerNummer<6) {playGame(A);}
252     }
253
254 //Here it is the automatic players
255 if (spillerNummer == 6) {
256     int autoSpiller1 = 1;
257     int autoSpiller2 = 2;
258     while (autoSpiller1 < 5) {
259         for(int j = 1 ; j % rounds != 0; j++){
260             Moede.Handling first = spillere.get(autoSpiller1).getAction();
261             Moede.Handling second =
262                 spillere.get(autoSpiller2).getAction();
263             Moede moedeFirst = new Moede(first ,second);
264             Moede moedeSecond = new Moede(second ,first );
265
266             spillere.get(autoSpiller1).saveMeet(moedeFirst);
267             spillere.get(autoSpiller2).saveMeet(moedeSecond);
268         }
269         tabel.addResult(autoSpiller1,autoSpiller2 ,
270                         spillere.get(autoSpiller1).score());
271         tabel.addResult(autoSpiller2 , autoSpiller1 ,
272                         spillere.get(autoSpiller2).score());
273
274         spillere.get(autoSpiller1).clearMemory();
275         spillere.get(autoSpiller2).clearMemory();
276         autoSpiller2++;
277         if(autoSpiller2 > 5) {
278             autoSpiller1++;
279             autoSpiller2 = autoSpiller1 + 1;
280         }
281     }
282     resultArea.setText(tabel.lavTabel());
283     textArea.setText(tabel.highScoreTabel());
284 }
```

### B.3. TABEL.JAVA

---

```
282         }
283     }
284
285     /**
286      * Sets the number of rounds
287      */
288     private void setRounds(int numberOfRounds)
289     {
290         rounds = numberOfRounds + 1;
291     }
292 }
```

## B.3 Tabel.java

```
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.Iterator;
4
5
6 /**
7  * Denne klasse står for at skabe tabellen med resultater
8  */
9 public class Tabel {
10     private int[][] playerTable;
11     private String[] nameTable;
12     private int total;
13     private ArrayList<String> totalListe;
14
15     public Tabel()
16     {
17         nameTable = new String[6];
18         playerTable = new int[6][6];
19         totalListe = new ArrayList<String>();
20         total = 0;
21
22         initialize();
23     }
24
25     /**
26      * Her tilføjes resultater til tabellen
27      */
28     public void addResult(int player, int opponent, int score)
29     {
30         playerTable[player][opponent] = score;
31     }
32
33
34     /**
35      * Adds a player name to the table
36      */
37     public void addPlayerName(int player, String name)
38     {
39         int length = name.length();
40         for(int i = 1; i < 14 - length; i++) {
```

### B.3. TABEL.JAVA

---

```
41         name = name + " " ;
42     }
43     nameTable[ player ] = name;
44 }
45
46 /**
47 * Her laves tabellen med de enkelte møder
48 * mellem spillerne
49 */
50 public String lavTabel()
51 {
52     String resultat = "De_enkelte_møder\nSpiller_____1_____2_____3_____
53 _____4_____5_____6___Total\n";
54     for( int i = 0; i<6 ; i++ ) {
55         resultat = resultat + nameTable[ i ];
56         for( int j = 0; j<6 ; j++ ) {
57             total += playerTable[ i ][ j ];
58             resultat = resultat + " " + formatNumber( playerTable[ i ][ j ]) + " ";
59         }
60         totalListe.add( formatNumber( total) + " " + nameTable[ i ]) ;
61         resultat = resultat + " " + formatNumber( total) + "\n";
62         total = 0;
63     }
64     return resultat;
65 }
66
67 /**
68 * Her laves highscore tabellen , sorteret efter point
69 */
70 public String highScoreTabel()
71 {
72     Collections.sort( totalListe );
73     String highScore = "";
74     Iterator it = totalListe.iterator();
75     for( int i=1; it.hasNext() ; i++ ) {
76         highScore = "" + (7-i) + it.next() + "\n" + highScore;
77     }
78     highScore = "High_Score_Liste--Hall_of_Fame\nPlads_Point_Spiller\n" +
79                 highScore;
80     return highScore;
81 }
82
83
84 /**
85 * Formatterer et nummer til en streng , med foran stillede
86 * mellemrum, til en samlet længde af 7
87 */
88 private String formatNumber( int number )
89 {
90     String badNumber = "" + number;
91     int length = badNumber.length();
92     String niceNumber = "";
```

#### B.4. SPILLER.JAVA

---

```
93     for(int i = 0 ; i < 7-length ; i++ ){
94         niceNumber = niceNumber + " ";
95     }
96     niceNumber = niceNumber + number;
97     return niceNumber;
98 }
99
100 /**
101 * Her fyldes dobbelt arrayet op med nuller, specielt til brug
102 * for de møder der ikke eksisterer/ når man ikke møder sig selv
103 *
104 */
105 private void initialize()
106 {
107     for(int j =0; j<6 ;j++) {
108         for(int i =0; i<6 ;i++) {
109             playerTable[i][j] = 0;
110         }
111     }
112 }
113 }
```

## B.4 Spiller.java

```
1 import java.util.ArrayList;
2 /**
3 * Dette er en abstrakt superklasse for alle spillerne
4 * i Dilemma spillet. Klassen definerer en generel spiller.
5 * Den konkrete spiller, dvs. med implementation af
6 * getAction, defineres i en afledt klasse.
7 * Klassen implementerer Comparable for at sortere spillerne.
8 */
9 abstract public class Spiller implements Comparable
10 {
11     protected String name;           // Spillerens navn
12     protected ArrayList<Moede> memory; // til tidligere møder
13     private int total = 0;           // spillerens totale score
14
15 /**
16 * Constructor til spilleren
17 *
18 * @param na Spillerens navn
19 */
20 public Spiller(String na)
21 {
22     name = na;
23     memory = new ArrayList<Moede>(); // initialiser hukommelsen
24 }
25
26 /**
27 * Returneret spillerens navn
28 * @return Navnet på spilleren
29 */
30 public String getName()
```

#### B.4. SPILLER.JAVA

---

```
31     {
32         return name;
33     }
34
35     /**
36      * Kan udskrive hukommelsen for denne spiller
37      */
38     protected void recall()
39     {
40         System.out.print("Hukommelse for " + name + ":");
41         if (memory.size() == 0) {
42             System.out.print("tom");
43         }
44         for (int i = 0; i < memory.size(); i++) {
45             System.out.print(" " + memory.get(i));
46             System.out.println();
47         }
48     }
49
50     // Denne metode implementeres i den afledte klasse
51     abstract public Moede.Handling getAction();
52
53     /**
54      * følgende metode indsætter et møde i hukommelsen.
55      */
56     public void saveMeet(Moede m)
57     {
58         memory.add(m);
59     }
60
61     /**
62      * følgende metode sletter hukommelsen før ny modspiller.
63      */
64     public void clearMemory()
65     {
66         memory.clear();
67     }
68
69     /**
70      * Denne metode beregner scoren ud fra hukommelsen
71      * @return Spillerens score
72      */
73     public int score()
74     {
75         int sum = 0; // sum er scoren mod een spiller
76         for (int i = 0; i < memory.size(); i++) {
77             sum += (memory.get(i)).value();
78         }
79         return sum;
80     }
81
82     /**
83      * Denne metode lægger et tal til scoren
84      * @param nyscore Det der skal lægges til i point
```

```
85     */
86     public void addToTotal(int nyscore)
87     {
88         total += nyscore;
89     }
90
91 /**
92 * Denne metode returnerer spillerens totale score
93 * @return den samlede score
94 */
95 public int getTotal()
96 {
97     return total;
98 }
99
100 //Her implementeres compareTo fra interfacelet Comparable.
101 //Værdien er positiv, hvis this player kommer efter efter
102 //B i ordningen, dvs. hvis B har den største score.
103 //Dette er modsat normal ordning af talværdier!
104 public int compareTo(Object B)
105 {
106     return ((Spiller) B).getTotal() - getTotal();
107 }
108 }
```

## B.5 Moede.java

```
1 /**
2 * This class creates the table of results
3 * for the dilemma game
4 */
5 public class Moede
6 {
7     enum Handling {sam, udn};
8     private final Handling SAM = Handling.sam;
9     private final Handling UDN = Handling.udn;
10    private Handling avalg;
11    private Handling bvalg;
12
13 /**
14 * Constructor, takes the two players actions
15 * as parameters
16 */
17 public Moede(Handling a, Handling b)
18 {
19     avalg=a;
20     bvalg=b;
21 }
22
23 /**
24 * Returns player a's action
25 */
26 public Handling getA()
27 {
```

## B.6. RANDOMPLAYER.JAVA

---

```
28         return avalg;
29     }
30
31     /**
32      * Returns player b's action
33      */
34     public Handling getB()
35     {
36         return bvalg;
37     }
38
39     /**
40      * Returns the point score of player a
41      */
42     public int value()
43     {
44         int result = -100000; // ændres med sikkerhed
45         if (avalg==SAM && bvalg==SAM) result= 2;
46         if (avalg==SAM && bvalg==UDN) result=-1;
47         if (avalg==UDN && bvalg==SAM) result= 4;
48         if (avalg==UDN && bvalg==UDN) result= 0;
49         return result;
50     }
51
52     /**
53      * Overwrites the method from object , is
54      * used in System.out.print lines , where
55      * the method is implicitly called
56      */
57     public String toString()
58     {
59         return "(" + avalg + "," + bvalg + ")";
60     }
61
62     /**
63      * This method compares two meetings (Not used in this project)
64      */
65     public boolean equals(Object m)
66     {
67         return ((Moede)m).getA() == avalg && ((Moede)m).getB() == bvalg;
68     }
69 }
```

## B.6 RandomPlayer.java

```
1 import java.util.Random;
2
3 /**
4  * A player strategi for the Dilemma game
5  * Returns a random action
6  */
7 public class RandomPlayer extends Spiller
8 {
9     Random random;
```

## B.7 UDNYTTE.JAVA

---

```
10    public RandomPlayer( String na )
11    {
12        super( na );
13        random = new Random();
14    }
15
16
17    // Implements the abstract method getAction:
18    public Moede.Handling getAction()
19    {
20        int i = random.nextInt(10);
21        if (i < 5) { // 50% chance each
22            return Moede.Handling.sam;
23        }
24        else {
25            return Moede.Handling.udn;
26        }
27    }
28 }
```

## B.7 Udnytte.java

```
1 /**
2 * Dette er en spillertype til Dilemma spillet
3 *
4 * Denne spiller udnytter hver gang
5 */
6 public class Udnytte extends Spiller
7 {
8     public Udnytte( String na )
9     {
10         super( na ); // bruger Constructor fra Player
11     }
12
13     // Implementationen af den abstrakte metode getAction:
14     public Moede.Handling getAction()
15     {
16         return Moede.Handling.udn;
17     }
18 }
```

## B.8 Samarbejde.java

```
1 /**
2 * A player strategi for the Dilemma game
3 * the player will allways coorperate
4 */
5 public class Samarbejde extends Spiller
6 {
7     public Samarbejde( String na )
8     {
9         super( na );
10    }
```

## B.9. KOK04.JAVA

---

```
11 // Implementats the abstrakt method getAction:  
12 public Moede.Handling getAction()  
13 {  
14     return Moede.Handling.sam;  
15 }  
16 }  
17 }
```

## B.9 Kok04.java

```
1  /**
2  * This is a player type
3  * for the Dilemma game,
4  * Depending on the last tree moves
5  */
6 public class Kok04 extends Spiller
7 {
8
9     public Kok04(String na) {
10         super(na);           // Constructor from spiller
11     }
12
13 // Implementation of the abstrakt method getAction:
14 public Moede.Handling getAction()
15 {
16     int score = 0;
17     int length = super.memory.size();
18     if(length > 2) {
19         Moede.Handling thirdLast = super.memory.get(length-2).getB();
20         Moede.Handling secondLast = super.memory.get(length-1).getB();
21         Moede.Handling last = super.memory.get(length-1).getB();
22         if (thirdLast.equals(Moede.Handling.sam)) {score += 1;};
23         if (secondLast.equals(Moede.Handling.sam)) {score += 2;};
24         if (last.equals(Moede.Handling.sam)) {score += 3;};
25     }
26     if (score < 3 && length > 2) {
27         return Moede.Handling.udn;
28     }
29     else {
30         return Moede.Handling.sam;
31     }
32 }
33 }
34 }
```