Algoritmiek Assignment 1

Mirza Kuraesin (s3278395)
Tim Haasdijk (s3167445)

April 6, 2022

1 Introductie

In dit opdracht hebben we het spel vier op een rij nagemaakt en functies gemaakt om een paar experimenten uit te oefenen. Vier op een rij werkt als volgt:

- Er zijn twee spelers.
- Spelers moeten omstebeurt een schijf neerleggen in een van de kolommen.
- Het spel eindigt wanneer:
  - vier schijven van dezelfde kleur verticaal, horizontaal, diagonaal elkaar aanraken
  - het bord vol is en er dus geen zet in het bord meer kan worden gedaan.

2 besteScore

De besteScore functie berekent de 'bestekolom' en 'bestescore' aan de hand van recursie. Bovenin de functie wordt gekeken of er een 'eindstand' is, aan de hand van de functie `scoregeef(aanBeurt, uitkomst)`. Bij het gebruiken van deze 'scoregeef' functie wordt gelijk de 'uitkomst' berekend voor de speler 'aanBeurt'.

'besteScore' maakt gebruik van brute force, dat wil zeggen dat het alle mogelijke borden vanaf die situatie uittekend/berekend. De functie loopt alle mogelijke zetten af (na andere mogelijke zetten) door middel van de for-loop met integer column. Door dus alle mogelijke zetten af te lopen, en tussentijds te checken op eindstanden/'scoregeef', krijgen we elk mogelijke bord/situatie.

Voor het bepalen van de 'bestescore' kijken tussen alle uitgewerkte borden naar het bord met de hoogste score, deze 'score' vergelijken wij met de 'bestescore' (zie if-statement(score > bestescore)). Van dit bord kijken wij wat de allereerste zet was, dit is de 'bestekolom', deze 'bestekolom' kunnen wij gelijkstellen aan de integer 'column' van de for-loop omdat wij bij de recursieve aanroep gebruik maken van een locale variabele genaamd 'bkolom'.

In de if-statement (score > bestescore) staat ook nog een if-statement (bestescore == 1) return bestescore. Deze if-statement staat er om willen van de snelheid van het bepalen van de 'bestescore'. Wij kunnen de 'bestescore' namenlijk gelijk returnen als de 'bestescore' gelijk is aan 1, want 1 is de 'hoogstescore' mogelijk in een potje (je wint). Hierdoor checken we alle andere borden niet meer nadat we weten dat we winnen als beide spelers optimaal spelen.

3 Beredenering Toestanden

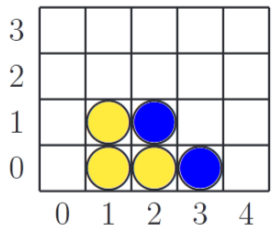

Figure 1: Toestand 1.

Bij toestand één kan Eva een zet spelen in kolom 1. Eva wilt ervoor zorgen dat in de eerste kolom een geel-blauw-geel-blauw situatie voorkomt zodat er een diagonale vier op een rij ontstaat. Dit kan ontstaan omdat er na de blauwe zet tien open vakjes zijn. Eva moet wachten totdat Iris in kolom zetten doet.

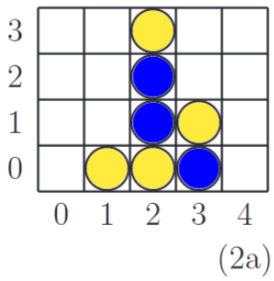

Figure 2: Toestand 2a.

Bij toestand 2a kan Eva een zet spelen in kolom 0. Eva heeft dan 2 manieren om diagonaal te winnen. Iris heeft een lose lose situatie in kolom 1, als Iris een zet doet in kolom 1 dan zal het de diagonale vier op een rij naar rechts van Eva blokkeren, maar dan kan eva daarna een zet doen in dezelfde kolom om één drie op een rij naar links te krijgen. Aan de andere kant als Iris niet een zet doet in kolom 1, dan kan Eva daar een zet doen om een drie op een rij naar rechts te krijgen. In ieder geval, Eva kan dan op wachten wanneer Iris een zet doet in kolom 0 of 3.

4 Experiment 1

In dit experiment hebben we een 5x6 bord vol gemaakt door goedezet tegen goedezet te spelen. Uiteindelijk kwam dit als eindstand:

Figure 3: Eindstand bord 5x6 met goedezet tegen elkaar.

Uit dit bord halen we steeds één zet uit om de volgende opdrachten uit te voeren:

- Bereken de beste score voor de speler A die in die toestand aan de beurt is.
- Speel het spel honderd keer uit (tot een eindstand), waarbij speler A steeds een 'goede zet' doet, terwijl de andere speler steeds een beste zet doet.
- Bereken van deze honderd spellen de gemiddelde score voor speler A, en vergelijk die met haar eerder berekende beste score.

Vevolgens moesten we het aantal standen uitprinten om beste score uit te rekenen, tijd dat duurt om honderd keer uit te spelen, de beste score en gemiddelde score van alle honderd spellen. Hieronder is de tabel:

	Aantal Standen	Tijd(in sec)	Beste Score	Gem.Score
1 undo	2	0.000696	1	1
2 undo	14	0.002474	-1	-1
3 undo	27	0.000777	1	1
4 undo	78	0.001812	-1	-1
5 undo	113	0.002299	1	-1
6 undo	195	0.000945	-1	-1
7 undo	250	0.002273	1	-1
8 undo	408	0.002382	1	-1
9 undo	567	0.002291	1	-1
10 undo	11289	0.04547	0	-0.59
11 undo	39960	0.11433	0	0
12 undo	353581	0.505988	0	-0.6
13 undo	1635240	2.55129	0	0
14 undo	19385691	56.262	0	0
15 undo	90796441	108.612	0	-0.34
16 undo	193218065	269.755	0	-0.6

Table 1: Tabel van alle waardes, elke keer dat er een zet wordt ongedaan.

Uit de tabel kunnen we concluderen dat naarmate het aantal zetten dat wordt ongedaan het aantal standen steeds meer groeit en ook de tijd om de honderd spellen te spelen. De reden waarom we niet een exponentiële groei hebben van het aantal standen is omdat we in de functie bepaalGoedeScore dit hebben staan:

```
if (bestescore == 1){
    return bestescore;
}
```

Hierdoor zal het niet alle standen over gaan omdat het gelijk returnt wanneer bestescore gelijk is aan 1. Ook is dit de reden waarom het sneller door de 100 spellen ging wanneer beste score 1 is.
Ook zien we dat bij lage aantal undo's de gemiddelde score 1 of -1 is, vaker -1. Dit komt omdat er bij weinig aantal vrije ruimte ook weinig aantal verschillende uitkomsten zijn, in dit geval alleen één uitkomst.
Bij meer undo's zien we ook dat de beste score 0 blijft, dit komt omdat goedezet met 100 simulaties nooit kan winnen van bestezet als er veel vrije ruimte is, alleen gelijk spelen.

5 Experiment 2

5.1 Intro

In dit experiment hebben we met lege borden met dimensies 4x4, 5x4, 5x5, 6x5, ..., 10x10 spellen gespeeld waarbij Iris random zetten doet en Eva steeds met behulp van de functie bepaalGoedeZet het spel speelt. Per bord zal bepaalGoedeZet met 1, 2, 4, 8, 16, 32 en 64 aantal random simulaties het goedezet berekenen. Dit doen we honderd keer voor de verschillende aantal random simulaties. Iris krijgt bij winst score+1, verlies score-1 en gelijkspel score=0. Aan het eind wordt de score gedeeld door het aantal simulaties.

5.2 Uitkomsten

Hier onder is een tabel van de resultaten:

Aantal Simulaties								
		1	2	4	8	16	32	64
	4x4	-0.17	-0.23	-0.33	-0.65	-0.78	-0.85	-0.90
	5x4	0.22	0.15	-0.12	-0.19	-0.39	-0.53	-0.6
	5x5	0.14	-0.13	-0.2	-0.57	-0.7	-0.94	-0.98
	6x5	0.15	0.21	0.05	-0.05	-0.24	-0.58	-0.66
	6x6	0.55	0.33	0.03	-0.29	-0.47	-0.77	-0.95
Bord	7x6	0.47	0.47	0.09	-0.03	-0.07	-0.49	-0.49
Grootte	7x7	0.39	0.49	0.31	-0.19	-0.47	-0.69	-0.91
	8x7	0.53	0.37	0.47	0.05	0.05	-0.51	-0.41
	8x8	0.63	0.61	0.21	0.07	-0.25	-0.65	-0.85
	9x8	0.69	0.69	0.55	0.23	0.03	-0.27	-0.45
	9x9	0.55	0.55	0.47	0.29	-0.01	-0.57	-0.64
	10x9	0.73	0.67	0.63	0.39	0.01	-0.13	-0.19
	10x10	0.65	0.73	0.53	0.25	-0.09	-0.49	-0.61

Table 2: Tabel met de gemiddelde score bij elk bord grootte en aantal simulaties.

5.3 Conclusie

Uit de tabel kunnen we de volgende concluderen:

- Bij lage aantal random simulaties voor het goede zet bepalen, is de score van Iris boven nul bij elk bord behalve bij 4x4. Dit zal komen omdat bij lage aantal random simulaties bepaalGoedeZet zet niet genoeg samples heeft om daadwerkelijk een goede zet te berekenen. Het spel wat gespeeld wordt is eigenlijk dan random vs random. Omdat de speler die het eerste zet heeft gedaan altijd gelijk of meer schijven op het bord heeft dan de tweede speler, zal inderdaad Iris een positieve score hebben bij laag aantal simulaties.

- Iris heeft meer een negatieve scores bij kleinere borden dan bij grotere borden. Dit komt omdat bij kleinere borden bepaalGoedeZet met een kleiner aantal kolommen mee te maken heeft. Hierdoor zal er voor elk kolom meer samples zijn dan bij grotere borden, waardoor het een betere goede zet kan bepalen.

Code

Dit is het programma:

```
1 // Implementatie van klasse VierOpEenRij
2
3 #include "vieropeenrij.h"
4 #include "standaard.h"
5 #include <fstream>   // voor inlezen van bord
```

```
6  #include <iostream>
7
8  //
       ***************************************************************************
9
10 //Constructor voor een lege bord.
11 //Membervariabel 'breedte' & 'hoogte' zijn de afmetingen van
       het bord.
12 //Membervariabel 'aanBeurt' is 1 omdat speler 1 altijd begint.
13 VierOpEenRij::VierOpEenRij ()
14 {
15   for (int i = hoogte -1; i >= 0; i--){ // MOOI OF NIET?
16     for (int j = 0; j < breedte; j++){
17       bord[j][i] = 0;
18     }
19   }
20   aanBeurt = 1;
21 }  // default constructor
22
23 //
       ***************************************************************************
24
25 //Constructor voor de experimenten.
26 //De parameters 'nwBreedte' & 'nwHoogte' zijn voor de
       dimensies van het bord.
27 //Ze worden gelijk gesteld aan 'breedte' & 'hoogte'.
28 VierOpEenRij::VierOpEenRij (int nwBreedte, int nwHoogte)
29 {
30   for (int i = nwHoogte -1; i >= 0; i--){
31     for (int j = 0; j < nwBreedte; j++){
32       bord[j][i] = 0;
33     }
34   }
35   aanBeurt = 1;
36   breedte = nwBreedte;
37   hoogte = nwHoogte;
38 }  // constructor met parameters
39
40 //
       ***************************************************************************
41
42 //Leest het bord in van een .txt file.
43 //Parameter 'invoernaam' is de naam van de .txt file.
44 //'breedte' is gelijk aan de 1e gelezen char, 'hoogte' is
       gelijk aan de 2e gelezen char.
45 //Als 'breedte' en 'hoogte' geldige waardes hebben(dus meer
       dan 0 en kleiner dan MaxDimentie)
46 //en het aantal schijven kloppen, dan wordt een bord gemaakt.
47 bool VierOpEenRij::leesIn (const char* invoernaam)
48 {
49   ifstream fin(invoernaam);
```

```
50    if(fin.is_open()){
51      fin >> breedte;
52      cout << "Breedte is: " << breedte << endl;
53      fin >> hoogte;
54      cout << "Hoogte is: " << hoogte << endl;
55      if (breedte <= MaxDimensie && hoogte <= MaxDimensie &&
            breedte > 0 && hoogte > 0){
56        int teller1 = 0, teller2 = 0;//telt het aantal schijven
              van elk speler
57        for (int i = hoogte -1; i >= 0; i--){
58          for (int j = 0; j < breedte; j++){
59            fin >> bord[j][i];//plaatst de char op positie [j][i
                  ]
60            //checkt of alle stenen goed 'gevallen' zijn.
61            if ((bord[j][i+1] != 0 && bord[j][i] == 0 && i >= 0
                  && (i <= hoogte -2))){
62              return false;
63            }
64            if (bord[j][i] == 1){
65              teller1++;
66            }else if (bord[j][i] == 2){
67              teller2++;
68            }
69          }
70        }
71        //als het aantal schijven kloppen in verhouding met
              elkaar, bord wordt gemaakt.
72        if (teller2 >= 0 && teller1 >= teller2 && teller2 + 1 >=
              teller1){
73          return true;
74        }
75      }
76    }
77  return false;
78  }// leesIn
79
80  //
      **********************************************************************
81
82  //Checkt of er een eindstand is bereikt.
83  //Het checkt voor ieder schijf na elke zet of er een vier op
      een rij is.
84  //Membervariabelen 'hoogte' & 'breedte' zijn de dimensies van
      het bord.
85  bool VierOpEenRij::eindstand ()
86  {
87    bool bordvol = true;//variabel of bord vol is
88    for (int i = hoogte -1 ; i >= 0; i--){
89      for (int j = 0; j < breedte; j++){
90        if (bord[j][i] == 0){//als een schijf in het bord waarde
              0 heeft, bord niet vol.
91          bordvol = false;
92        }
```

```
93        //checkt 4 richtingen voor elke schijf of er vier op een
              rij is
94        for (int richtingen = -1; richtingen <= 1; richtingen++)
              {
95          for (int richtingen2 = -1; richtingen2 <= 1;
                richtingen2++){
96            if (richtingen != 0 || richtingen2  != 0){
97              if (telSchijvenInRichting(1, i, j, richtingen,
                    richtingen2) >= 4){
98                return true;
99              }
100             if (telSchijvenInRichting(2, i, j, richtingen,
                    richtingen2) >= 4){
101               return true;
102             }
103           }
104         }
105       }
106     }
107   }
108   if (bordvol == true){
109     return true;
110   }
111   return false;
112 } // eindstand
113
114 //
      ********************************************************************************
115
116 //Druk de hele stand (bord met schijven, speler aan beurt) af
      op het scherm.
117 //Membervariabel 'aanBeurt' zegt wie er aan de beurt is.
118 //Membervariabelen 'hoogte' & 'breedte' zijn de dimensies van
      het bord.
119 //
120 void VierOpEenRij::drukAf ()
121 {
122 int teller1 = 0, teller2 = 0;//teller voor aantal schijven van
       elk speler
123   for (int i = hoogte -1 ; i >= 0; i--){//onderaan is hoogte 0
124     cout << endl;
125     for (int j = 0; j < breedte; j++){
126      cout << bord[j][i] << " ";
127      if (bord[j][i] == 1){
128       teller1++;
129      }else if (bord[j][i] == 2){
130       teller2++;
131      }
132     }
133   }
134   cout << endl;
135   if(teller1 == teller2){//wie er aan de beurt is wordt
          geprint.
```

```
136       cout << "Speler 1 is aan de beurt";
137       aanBeurt = 1;
138     }else{
139       cout << "Speler 2 is aan de beurt";
140       aanBeurt = 2;
141     }
142
143   }  // drukAf
144
145   //
        **********************************************************************
146
147   //Functie om een zet te doen voor de huidige speler in de
        gekozen kolom.
148   //Als aanBeurt 1 is en een zet wordt gedaan, aanBeurt wordt 2.
149   //De laatste zet wordt in een vector achter in gepushed.
150   bool VierOpEenRij::doeZet (int kolom)
151   {
152     int hoogtecopy = hoogte -1;//variabel van de hoogte van het
          bord
153     if (bord[kolom][hoogtecopy] != 0){//als de kolom vol is,
          return false
154       return false;
155     }else{//totdat het geen 0 tegenkomt, hoogtecopy--
156       while (bord[kolom][hoogtecopy] == 0 && (hoogtecopy >=0)){
157         hoogtecopy--;
158       }
159       bord[kolom][hoogtecopy+1] = aanBeurt;//schijf van aanBeurt
            wordt geplaatst
160       zetten.push_back(kolom);//laatste kolom wordt in de vector
            gezet
161       if (aanBeurt == 1){//speler wordt van beurt gewisseld
162         aanBeurt = 2;
163       }else{
164         aanBeurt = 1;
165       }
166     return true;
167     }
168   }  // doeZet
169
170   //
        **********************************************************************
171   //Maakt laatst gedane zet ongedaan.
172   //Haalt de laatste kolom van de vector.
173   bool VierOpEenRij::unDoeZet ()
174   {
175     int hoogtecopy = hoogte -1;//variabel van de hoogte van het
          bord
176     int n = zetten.size();//aantal zetten in de vector
177     if (zetten.empty() == true){//als er geen zetten gedaan zijn
          , return false
178       return false;
```

```
179     }else{//hoogte van kolom wordt verlaagd
180       while (bord[zetten[n-1]][hoogtecopy] == 0 && (hoogtecopy
              >=0)){
181         hoogtecopy--;
182       }
183       bord[zetten[n-1]][hoogtecopy] = 0;//op deze positie wordt
              veranderd naar een 0
184       if (aanBeurt == 1){//speler wordt van beurt gewisseld
185         aanBeurt = 2;
186       }else{
187         aanBeurt = 1;
188       }
189       zetten.pop_back();
190     }
191     return true;
192   }  // unDoeZet
193
194   //
          ********************************************************************************
195   //Doormiddel van bruteforce en recursie wordt de eindscore van
          de huidige
196   //speler wanneer vanaf dat punt beide spelers optimaal spelen.
197   //Parameter besteKolom is de beste zet die de huidige speler
        kan zetten op
198   //dit punt. Het aantalStanden is het aantal borden gemaakt met
          recursie
199   //om de eindstand te berekenen.
200   int VierOpEenRij::besteScore (int &besteKolom, long long &
        aantalStanden)
201   {
202     int score;
203     aantalStanden++;
204     int bkolom = -1;//nodig voorrecursief aanroepen van
            bestescore
205     int uitkomst;//
206     int bestescore = -2;
207     if (scoregeef(aanBeurt,uitkomst)){
208       return uitkomst;
209      // laatste zet pakken van vector
210     }else{
211       for(int column = 0; column < breedte; column++){
212         if (bord[column][hoogte-1] == 0){
213           doeZet(column);
214           score = -besteScore(bkolom, aantalStanden);
215           unDoeZet();
216           if (score > bestescore){
217             bestescore = score;
218             besteKolom = column;
219             if (bestescore == 1){
220               return bestescore;
221             }
222           }
223         }
```

```
224
225       }
226     }
227     return bestescore;
228   }  // besteScore
229
230   //
        **************************************************************************
231   //Hier wordt de beste zet berekent voor de huidige speler door
          middel van
232   //het spelen van aantal nrSimulaties random uitspeelt het de
        kolom kiest
233   //met de hoogste gemiddelde score.
234   //nrSimulaties in de parameter is standaard 100.
235   //Voor het berekenen van score: +1 als gewonnen, 0 als remise
236   //-1 als verloren.
237   //
238   int VierOpEenRij::bepaalGoedeZet (int nrSimulaties)
239   {
240     int max = 0, punten[MaxDimensie] = {0};//hoogste score
241     int oudezetten = zetten.size();//aantal zetten voordat
          functie is aangeroepen
242     int speleraanbeurt = aanBeurt;//speleraanbeurt = speler die
          functie heeft geroepen
243     int random = randomGetal(0, breedte -1);//random kolom
244     int uitkomst; //integer voor uitkomst van het bord
245     if (!eindstand()){//als de kolom al vol is,...
246       for (int i = 0; i < breedte; i++){
247         if (bord[i][hoogte-1] != 0){
248           punten[i]= -10000;//kan sws niet gekozen als beste
                kolom
249         }
250       }
251       for(int kopties = 0; kopties < breedte; kopties++){
252         for(int simulaties = nrSimulaties; simulaties > 0;
              simulaties--){
253           if (bord[kopties][hoogte-1] == 0){
254             doeZet(kopties);//gaat alle kolommen na als begin
                  zet
255             while (!scoregeef(speleraanbeurt, uitkomst)){//als
                  nog geen eindstand, doe random zetten
256               random = randomGetal(0, breedte-1);
257               doeZet(random);
258             }
259             punten[kopties] += uitkomst;
260             while(zetten.size() - oudezetten > 0){
261               unDoeZet();
262             }
263           }
264         }
265       }
266     }
267     for (int j = 0; j < breedte; j++){
```

```
268       if (punten[j] > punten[max]){
269         max = j;//kiest de kolom met de hoogste score
270       }
271     }
272     return max;
273   }  // bepaalGoedeZet
274 
275   //
        **************************************************************************

276 
277   int VierOpEenRij::bepaalGoedeScore (int nrSimulaties)
278   {
279     int oudezetten = zetten.size(), speleraanbeurt = aanBeurt;
280     int uitkomst; //integer opslaan uitkomst
281     scoregeef(speleraanbeurt,uitkomst);//uitkomst bepalen als al
            eindstand
282     int returnwaarde= uitkomst;//returnwaarde is uitkomst
283     int bestekolom;//beste kolom voor beste zet
284     int goedekolom;//goede kolom voor goede zet
285     long long aantalStanden = 0;
286     if (!eindstand()){
287       while (!scoregeef(speleraanbeurt, uitkomst)){
288         if (speleraanbeurt == aanBeurt){
289           goedekolom = bepaalGoedeZet(nrSimulaties);
290           //bepalen goede zet
291           doeZet(goedekolom);//zet goede zet
292         }else{
293           besteScore(bestekolom, aantalStanden);
294           //bepalen beste kolom
295           doeZet(bestekolom);// zet beste zet
296         }
297       }
298       returnwaarde = uitkomst;
299       while(zetten.size() - oudezetten > 0){
300         unDoeZet();//plaats bord terug naar oude situatie
301       }
302     }
303 
304   // TODO: implementeer deze memberfunctie
305 
306     return returnwaarde;
307   // TODO: implementeer deze memberfunctie
308 
309   }  // bepaalGoedeScore
310 
311   //
        **************************************************************************

312   /* Deze member*/
313   int VierOpEenRij::bepaalRandomScore (int nrSimulaties)
314   {
315     int oudezetten = zetten.size();//aantal zetten voordat
          functie is aangeroepen
```

```
316   int speleraanbeurt = aanBeurt;//speleraanbeurt = speler die
          functie heeft geroepen
317   int uitkomst;//uitkomst van het spel
318   scoregeef(speleraanbeurt, uitkomst);//goede returnwaarde als
           eindstand
319   int returnwaarde = uitkomst;
320   int random = randomGetal(0, breedte -1);//random kolom
321   int goedezet;//kolom van goedezet
322   if (!eindstand()){
323     while (!scoregeef(speleraanbeurt, uitkomst)){
324       if (speleraanbeurt == aanBeurt){//als huidige speler,
325         while(bord[random][hoogte-1] != 0){//doe randomzet
326           random = randomGetal(0, breedte -1);
327         }
328         doeZet(random);
329       }else{
330         goedezet = bepaalGoedeZet(nrSimulaties);
331         doeZet(goedezet);//doe goede zet
332       }
333     }
334     returnwaarde = uitkomst;//score van eindstand
335     while(zetten.size() - oudezetten > 0){
336       unDoeZet();//terugzetten van oudesituatie van het bord
337     }
338   }
339
340 // TODO: implementeer deze memberfunctie
341
342   return returnwaarde;
343
344 }  // bepaalRandomScore
345
346 //
      ************************************************************************
347 /* Deze Memberfunctie telt het aantal schijven van een
      specifieke 'kleur' op een punt met 'kolom0',
348 'rij0'
349 in de richting 'deltakolom' en 'deltarij'. 'deltarij' is de
      delta richting van rij, 'deltarij' = 1
350 zou dus een verplaatsing naar boven. 'deltakolom' is hetzelfde
       maar dan voor de kolom.
351 Deze memberfunctie veranderd niets aan de membervariabelen. */
352 int VierOpEenRij::telSchijvenInRichting (int kleur, int kolom0
      , int rij0,
353           int deltakolom, int deltarij)
354 {
355   int kolom = kolom0, rij = rij0;//copy van kolom0 en rij0
          voor aanpassen
356   int inderichting = 0;// Teller voor aantalschijven in een
          richting
357   while (rij < hoogte && rij >= 0 && kolom < breedte && kolom
          >=0 && bord[kolom][rij] == kleur){
358     inderichting++;
```

```
359     rij = rij + deltarij;
360     kolom = kolom + deltakolom;
361   }
362   return inderichting;
363 }  // telSchijvenInRichting
364
365
366 /* Deze memberfunctie kijkt of er een zet is die al eerder in
       het programma is gedaan.
367  Zo ja dan gaat de functie voor deze kolom/zet kijken wat
       bijbehorende hoogte ervan is.
368  Aan de hand van deze bijbehorende hoogte gaat de functie in
       alle 8 de richtingen
369  kijken of er een 4 of meer op een rij is, met de functie
       telSchijvenInRichting.
370  Wij moeten beide richtingen checken want het kan het geval
       zijn dat de huidige zet net tussen een 1 links en een 2
       rechs wordt geplaats.  */
371
372 bool VierOpEenRij::scoregeef(int speleraanbeurt, int& uitkomst
      ){
373   int kolom0 = zetten[zetten.size()-1];// pakt vorige zet
374   int rij0 = hoogte - 1;//hoogst mogelijke rij
375   int kleur;//integer voor het bewaren van de kleur van de
        vorige zet
376   if (zetten.empty() == false){
377     while (bord[kolom0][rij0] == 0){
378       rij0--;
379     }
380     kleur = getVakje(kolom0, rij0);
381     for (int richtingen = 0; richtingen <= 1; richtingen++){
382       for (int richtingen2 = -1; richtingen2 <= 1; richtingen2
            ++){
383         if (richtingen != 0 || richtingen2  != 0){
384           if ((telSchijvenInRichting(kleur, kolom0, rij0,
                  richtingen, richtingen2) + telSchijvenInRichting
                  (kleur, kolom0, rij0, -richtingen, -richtingen2)
                   -1) >= 4){
385             if (speleraanbeurt == kleur){
386               uitkomst = 1; // de speler die de zet deed heeft
                      gewonnen
387               return true;
388             }else{
389               uitkomst = -1;// de speler die de zet deed heeft
                      verloren
390               return true;
391             }
392           }
393         }
394       }
395     }
396     //wel iets doen
397   }
```

```
398     for (int i = 0; i < breedte; i++){//kijken of het bord niet
            vol is
399       if (bord[i][hoogte-1] == 0){
400         uitkomst = -2;//uitkomst boeit niet, je speelt verder
401         return false;//het bord is nog niet vol
402       }
403     }
404     uitkomst = 0;//het bord heeft geen vier op een rij en is vol
            , remise
405     return true;//Eindstand is remise
406   }
```

```
1   // Implementatie van standaard functies.
2
3   #include <iostream>
4   #include <cstdlib>  // voor rand
5   #include "standaard.h"
6   using namespace std;
7
8   //
        ********************************************************************************
9
10  bool integerInBereik (const char *variabele, int waarde,
11                        int minWaarde, int maxWaarde)
12  {
13    if (waarde>=minWaarde && waarde<=maxWaarde)
14      return true;
15    else
16    { cout << variabele << "=" << waarde << ", maar moet in ["
17         << minWaarde
18           << "," << maxWaarde << "] liggen." << endl;
19      return false;
20    }
21
22  }  // integerInBereik
23
24  //
        ********************************************************************************
25
26  bool integerInBereik (int waarde, int minWaarde, int maxWaarde
        )
27  {
28    if (waarde>=minWaarde && waarde<=maxWaarde)
29      return true;
30    else
31      return false;
32
33  }  // integerInBereik
34
35  //
        ********************************************************************************
```

```
35
36 int randomGetal (int min, int max)
37 { int bereik,
38       r;
39
40   bereik = max - min + 1;
41
42   r = ((rand())%bereik) + min;
43   return r;
44
45 }  // randomGetal
```

```
1  // Defintie van standaard functies.
2
3  #ifndef StandaardHVar  // om te voorkomen dat dit .h bestand
       meerdere keren
4  #define StandaardHVar  // wordt ge-include
5
6  // Controleer of variabele met naam 'variabele' een waarde '
       waarde' heeft
7  // die tussen (inclusief) minWaarde en maxWaarde in ligt.
8  // Zo nee, geef een passende foutmelding.
9  //
10 // Voorbeeld van aanroep:
11 //   if (integerInBereik ("teller", teller, 0, 1000))
12 //     ...
13 bool integerInBereik (const char *variabele, int waarde,
14                        int minWaarde, int maxWaarde);
15
16 // Controleer of waarde 'waarde' tussen (inclusief) minWaarde
       en maxWaarde
17 // in ligt.
18 // Geef GEEN foutmelding als het niet zo is.
19 bool integerInBereik (int waarde, int minWaarde, int maxWaarde
       );
20
21 // Genereer een random geheel getal r waarvoor  min <= r <=
       max.
22 // Pre: min <= max;
23 int randomGetal (int min, int max);
24
25 #endif
```

```
1  // Enkele constantes voor implementatie Vier op een Rij
2
3  #ifndef ConstantesHVar  // voorkom dat dit bestand meerdere
       keren
4  #define ConstantesHVar  // ge-include wordt
5
6  const int MaxDimensie = 10;  // maximaal aantal rijen en
       maximaal aantal
7                               // kolommen in een spel
8  const int Leeg = 0;
9  const int Geel = 1;
```

```
10 const int Blauw = 2;
11 // TODO: zo nodig uw eigen constantes
12
13 #endif
```

```
1  // Definitie van klasse VierOpEenRij
2
3  #ifndef VierOpEenRijHVar // voorkom dat dit bestand meerdere
       keren
4  #define VierOpEenRijHVar // ge-include wordt
5
6  #include <vector>
7  #include "constantes.h"
8  using namespace std;
9
10 class VierOpEenRij
11 {
12 public:
13   // Default constructor.
14   VierOpEenRij();
15
16   // Constructor met parameters voor breedte en hoogte van het
         bord.
17   // Controleer eerst nog wel of deze parameters geldig zijn.
18   VierOpEenRij(int nwBreedte, int nwHoogte);
19
20   // Getter voor breedte
21   int getBreedte()
22   {
23     return breedte;
24   }
25
26   // Getter voor de inhoud van vakje (kolom,rij) op het bord.
27   int getVakje(int kolom, int rij)
28   {
29     return bord[kolom][rij];
30   }
31
32   // Lees een bord in vanuit tekstbestand invoernaam, mogelijk
         al (deels)
33   // gevuld met schijven.
34   // Controleer daarbij
35   // * of het bestand wel te openen is,
36   // * of breedte en hoogte binnen de grenzen vallen,
37   // * of 0 <= aantalBlauw <= aantalGeel <= aantalBlauw+1
38   // * of er geen lege vakjes onder gevulde vakjes liggen
39   // Retourneer:
40   // * true, als aan alle voorwaarden is voldaan
41   // * false, als niet aan alle voorwaarden is voldaan
42   // Post:
43   // * Als aan alle voorwaarden is voldaan, is het bord met de
         schijven
44   //   opgeslagen in membervariabelen, en staat ook aanBeurt
       goed.
```

```
45    bool leesIn(const char *invoernaam); // check
46
47    // Controleer of we een eindstand hebben bereikt:
48    // * een van de twee spelers heeft vier op een rij
49    // * of het bord is vol
50    // Retourneer:
51    // * true, als we een eindstand hebben bereikt
52    // * false, als we geen eindstand hebben bereikt
53    bool eindstand(); // check
54
55    // Druk de hele stand (bord met schijven, speler aan beurt)
          af op
56    // het scherm.
57    void drukAf(); // check
58
59    // Doe een zet voor de speler die aan de beurt is:
60    // een schijf laten vallen in kolom 'kolom'
61    // Controleer eerst of het wel een geldige zet is, dat wil
          zeggen,
62    // of het nog geen eindstand is, of kolom een geldig
          kolomnummer is,
63    // en die kolom nog niet vol is.
64    // Retourneer:
65    // * true, als dit een geldige zet is
66    // * false, als dit geen geldige zet is.
67    // Pre:
68    // * huidig bord is geldig
69    // Post:
70    // * als het een geldige zet is, is de zet uitgevoerd:
71    //   - de schijf ligt op het bord in de aangegeven kolom
72    //   - de speler aan beurt is gewisseld,
73    //   - de zet is toegevoegd aan de lijst met gedane zetten
74    // * als het geen geldige zet is, is de stand niet veranderd
          .
75    bool doeZet(int kolom); // check
76
77    // Maak de laatst gedane zet ongedaan.
78    // Controleer eerst of er wel een zet is om ongedaan te
          maken,
79    // opgeslagen in de lijst met zetten.
80    // Retourneer:
81    // * true, als er een zet was om ongedaan te maken
82    // * false, anders
83    // Post:
84    // * als returnwaarde true is, is de zet ongedaan gemaakt:
85    //   - de schijf is van het bord gehaald
86    //   - de zet is van de lijst met uitgevoerde zetten gehaald
87    //   - aanBeurt is teruggezet
88    // * als returnwaarde false is, is er niets veranderd
89    bool unDoeZet(); // check
90
91    // Bepaal met behulp van brute force en recursie de
          eindscore voor
92    // de speler die in de huidige stand (= de stand van de
```

```
        huidige
93      // recursieve aanroep) aan de beurt is, wanneer beide
            spelers vanaf
94      // dit punt optimaal verder spelen.
95      // De score is
96      // * 1 als de speler gaat winnen
97      // * 0 als het remise wordt
98      // * -1 als de speler gaat verliezen
99      // Pre:
100     // * de huidige stand is het resultaat van correct spel
101     // Post:
102     // * als de huidige stand geen eindstand was, bevat
            parameter
103     //   besteKolom het nummer van de kolom waarin de huidige
            speler
104     //   de volgende schijf moet spelen, om de beste score te
            bereiken
105     // * anders bevat parameter besteKolom een passende default
            waarde
106     // * aantalStanden is gelijk aan het aantal standen dat we
            hebben
107     //   bekeken bij het bepalen van de beste eindscore
108     // * de stand in het spel is nog onveranderd
109     int besteScore(int &besteKolom, long long &aantalStanden);
110
111     // Bepaal een 'goede zet' voor de speler die in de huidige
            stand aan
112     // aan de beurt is: de zet die ertoe leidt dat hij bij
            nrSimulaties keer
113     // random uitspelen een zo goed mogelijke gemiddelde score
            haalt.
114     // Controleer eerst
115     // * of het geen eindstand is
116     // Retourneer:
117     // * de gevonden kolom, als het geen eindstand is
118     // * een passende default waarde, als het al wel een
            eindstand is
119     int bepaalGoedeZet(int nrSimulaties); // check
120
121     // Speel het spel uit vanaf de huidige stand. Laat hierbij
            de speler
122     // die in de huidige stand aan de beurt is, steeds een '
            goede zet'
123     // (gevonden met bepaalGoedeZet) doen, terwijl de andere
            speler steeds
124     // een beste zet (gevonden met besteScore) doet.
125     // Parameter nrSimulaties wordt gebruikt voor aanroep
            bepaalGoedeZet.
126     // Retourneer:
127     // * de score aan het eind van het spel voor de speler die
            steeds
128     //   een 'goede zet' gedaan heeft
129     // Post:
130     // * de huidige stand is weer hetzelfde als aan het begin
```

```
        van de functie
131     //   (zetten zijn dus weer ongedaan gemaakt)
132     int bepaalGoedeScore(int nrSimulaties);
133
134     // Speel het spel uit vanaf de huidige stand. Laat hierbij
            de speler
135     // die in de huidige stand aan de beurt is, steeds een
            random
136     // (maar wel geldige) zet doen, terwijl de andere speler
            steeds
137     // een 'goede zet' (gevonden met bepaalGoedeZet) doet.
138     // Parameter nrSimulaties wordt gebruikt voor aanroep
            bepaalGoedeZet.
139     // Retourneer:
140     // * de score aan het eind van het spel voor de speler die
            steeds
141     //   een random zet gedaan heeft
142     // Post:
143     // * de huidige stand is weer hetzelfde als aan het begin
            van de functie
144     //   (zetten zijn dus weer ongedaan gemaakt)
145     int bepaalRandomScore(int nrSimulaties); // check
146   private:
147     // Tel het aantal aaneensluitende schijven van een bepaalde
            kleur,
148     // vanaf (en inclusief) positie (kolom0,rij0) op het bord,
149     // in richting (deltakolom,deltarij).
150     // Retourneer:
151     // * het gevonden aantal
152     int telSchijvenInRichting(int kleur, int kolom0, int rij0,
153                               int deltakolom, int deltarij); //
                                      check
154     bool scoregeef(int speleraanbeurt, int &uitkomst);       //
            check
155                                                              //
                                                                  membervariabelen

156     int bord[MaxDimensie][MaxDimensie]; // [kolom][rij]: inhoud
            van bord;
157                                         // kolommen genummerd
                                               van 0..breedte-1,
                                               van links naar
                                               rechts;
158                                         // rijen genummerd van
                                               0..hoogte-1, van
                                               beneden naar boven
159     vector<int> zetten;                 // uitgevoerde zetten (
            kolomnummers) die tot
160                         // huidige stand hebben geleid (bij
                               ingelezen bord alleen de zetten
161                         // die na inlezen zijn uitgevoerd)
162     int breedte, hoogte, // van het bord
163         aanBeurt;        // speler die aan de beurt is
164   };
```

```
165
166 #endif
```

```
1  // Hoofdprogramma voor oplossing voor eerste
       programmeeropdracht Algoritmiek,
2  // voorjaar 2022: Vier op een Rij
3  //
4  // Biedt de gebruiker een menustructuur om
5  // * het spel te spelen
6  //   - vanaf een nieuw, leeg bord
7  //   - vanaf een in te lezen, en mogelijk (deels) gevuld bord
8  //    waarbij de gebruiker steeds
9  //    - een zet kan uitvoeren (een schijf in een kolom laten
       vallen)
10 //    - de laatste zet ongedaan kan maken
11 //    - kan vragen om de score voor de speler die aan de beurt
       is, als beide
12 //      spelers vanaf dit moment optimaal verder spelen
13 //    - kan vragen om een 'goede zet' voor de speler die aan de
        beurt is
14 //    - kan vragen om de eindscore voor de speler die aan de
       beurt is,
15 //      als hij vanaf nu een 'goede zet' doet, terwijl de
       andere speler
16 //      steeds een beste zet doet
17 //    - kan vragen om de eindscore voor de speler die aan de
       beurt is,
18 //      als hij vanaf nu een random zet doet, terwijl de andere
        speler
19 //      steeds een 'goede zet' doet
20 //
21 // * experiment 1 uit te voeren, waarbij een stand zo ver
       mogelijk vanaf
22 //    het einde wordt uitgespeeld met 'goede zetten' tegen
       beste zetten
23 // * experiment 2 uit te voeren, waarbij voor een reeks maten
       van borden
24 //    het spel vanaf het begin tot het einde wordt uitgespeeld
       met random
25 //    zetten tegen 'goede zetten' (waarbij ook het aantal
       simulaties wordt
26 //    gevarieerd dat wordt uitgevoerd bij het bepalen van een '
       goede zet')
27 //
28 // Namen + nummers studenten, datum
29
30 #include <iostream>
31 #include <ctime>  // voor clock() en clock_t
32 #include "vieropeenrij.h"
33 using namespace std;
34 const int MaxBestandsNaamLengte = 30; // maximale lengte van
       een bestandsnaam
35 const int nrSimulaties = 100;  // standaard aantal random
       simulaties per zet
```

```
36                                  // bij bepalen 'goede zet'
37  //const int nrTests = 100;  // aantal tests voor zelfde set
        parameters bij
38                              // experimenten
39
40  //
        ************************************************************************
41
42  // Schrijf het menu op het scherm en vraag een keuze van de
        gebruiker.
43  // Retourneer: de keuze van de gebruiker
44  int keuzeUitMenu ()
45  { int keuze;
46
47    cout << endl;
48    cout << "1. Een zet uitvoeren" << endl;
49    cout << "2. Laatste zet ongedaan maken" << endl;
50    cout << "3. Beste score (met beste zet) bepalen" << endl;
51    cout << "4. Een goede zet bepalen" << endl;
52    cout << "5. Bepaal score goed tegen best" << endl;
53    cout << "6. Bepaal score random tegen goed" << endl;
54    cout << "7. Stoppen met dit spel" << endl;
55    cout << endl;
56    cout << "Maak een keuze: ";
57    cin >> keuze;
58
59    return keuze;
60
61  }  // keuzeUitMenu
62
63  //
        ************************************************************************
64
65  // Roep vr1->besteScore aan, meet de benodigde tijd, en zet de
        relevante
66  // data op het scherm.
67  void roepBesteScoreAan (VierOpEenRij *vr1)
68  { clock_t t1, t2;
69    int kolom = 0,
70        score;
71    long long aantalStanden = 0;  // aantal bekeken standen bij
        aanroep besteScore
72
73    t1 = clock ();
74    score = vr1 -> besteScore (kolom, aantalStanden);
75    t2 = clock ();
76    cout << endl;
77    cout << "Beste score is: " << score << endl;
78    cout << "Een beste zet is: " << kolom << endl;
79    cout << "We hebben hiervoor " << aantalStanden
80         << " standen bekeken." << endl;
81    cout << "Dit kostte " << (t2-t1) << " clock ticks, ofwel "
```

```
82          << (((double)(t2-t1))/CLOCKS_PER_SEC) << " seconden."
                << endl;
83
84  }  // roepBesteScoreAan
85
86  //
        **************************************************************************

87
88  // Speel het spel op het bord van vr1.
89  // Hierbij krijgt de gebruiker herhaaldelijk de keuze om
90  // * een zet uit te voeren (een schijf in een kolom laten
        vallen)
91  // * te vragen om de score voor de speler die aan de beurt is,
         als beide
92  //   spelers vanaf dit moment optimaal verder spelen
93  // * te vragen om een 'goede zet'
94  // * te vragen om de 'goede score'
95  //
96  // Voor elke iteratie van het menu wordt de stand afgedrukt.
97  //
98  // Dit alles gaat door
99  // * totdat er een eindstand is bereikt (een van de spelers
        heeft vier op
100 //   een rij, of het bord is vol)
101 // * of totdat de gebruiker aangeeft dat hij wil stoppen met
        het spel
102 void doeSpel (VierOpEenRij *vr1)
103 { int breedte,
104       keuze,
105       kolom,
106       score;
107
108   breedte = vr1->getBreedte();
109
110   keuze = 0;
111   while (keuze!=7 && !vr1->eindstand())
112   {
113     vr1 -> drukAf ();
114
115     keuze = keuzeUitMenu ();
116
117     switch (keuze)
118     { case 1: cout << endl;
119               cout << "Geef het nummer van de kolom (0.." <<
                      breedte-1 << "): ";
120               cin >> kolom;
121               vr1->doeZet (kolom);
122               break;
123       case 2: if (!(vr1->unDoeZet ()))
124               { cout << endl;
125                 cout << "Er is geen zet ongedaan gemaakt." <<
                        endl;
126               }
```

```
127                 break;
128         case 3: roepBesteScoreAan (vr1);
129                 break;
130         case 4: kolom = vr1 -> bepaalGoedeZet (nrSimulaties);
131                 cout << endl;
132                 cout << "Een goede zet is: " << kolom << endl;
133                 break;
134         case 5: score = vr1 -> bepaalGoedeScore (nrSimulaties);
135                 cout << endl;
136                 cout << "Score goed tegen best is: " << score <<
                        endl;
137                 break;
138         case 6: score = vr1 -> bepaalRandomScore (nrSimulaties);
139                 cout << endl;
140                 cout << "Score random tegen goed is: " << score
                       << endl;
141                 break;
142         case 7: break;
143         default: cout << endl;
144                  cout << "Voer een goede keuze in!" << endl;
145       }  // switch
146
147     }  // while
148
149     if (vr1->eindstand())
150     { vr1 -> drukAf ();
151       cout << endl;
152       cout << "De huidige stand is een eindstand.\n";
153     }
154
155 }  // doeSpel
156
157 //
        ************************************************************************
158
159 // Voert experiment 1 uit zoals beschreven in de opdracht.
160 void doeExperiment1 ()
161 {
162   clock_t t1, t2;//stopwatch
163   int breedte, hoogte;
164   double score = 0;//score
165   double gemmscore;//gemmiddelde score
166   int bscore; //opslaan bestescore
167   char aantalundoezet;
168   int bestekolom;//integer voor bepalen beste kolom
169   long long aantalstand = 0;
170   cout << "Geef de breedte van het bord (1.." << MaxDimensie
          << "): ";
171   cin >> breedte;//invoeren breedte
172   cout << "Geef de hoogte van het bord (1.." << MaxDimensie <<
          "): ";
173   cin >> hoogte;// invoeren hoogte
174   VierOpEenRij *vr2;
```

```
175     vr2 = new VierOpEenRij (breedte, hoogte);//aanmaken van een
            bord
176     while (!vr2 -> eindstand()){
177       //maken van een bijna vol bord met bepaalgoede zet
178       vr2 -> doeZet(vr2-> bepaalGoedeZet(nrSimulaties));
179     }
180     while (vr2->unDoeZet()){
181       cin >> aantalundoezet;//zorgt ervoor dat je per undoe de
              resultaten netjes kan zien
182       t1 = clock ();
183       bscore = vr2-> besteScore(bestekolom, aantalstand);
184       for (int loep = 100; loep > 0; loep--){
185         score += vr2-> bepaalGoedeScore(bestekolom);
186       }
187       gemmscore = score/100;
188       t2 = clock ();
189       cout << "Het totale aantal standen dat bekeken is is: " <<
              aantalstand << endl;
190       cout << "Dit kostte " << (t2-t1) << " clock ticks, ofwel "
191          << (((double)(t2-t1))/CLOCKS_PER_SEC) << " seconden."
                 << endl;
192       cout << "De beste score was: " << bscore << endl;
193       cout << "De gemmiddelde score is: " << gemmscore << endl;
194       gemmscore = 0;
195       score = 0;
196     }
197   // TODO: implementeer deze functie verder, zodat het
          experiment met zo min
198   // mogelijk input van de gebruiker wordt uitgevoerd
199   }  // doeExperiment1
200
201   //
          **********************************************************************
202
203   // Voert experiment 2 uit zoals beschreven in de opdracht.
204   void doeExperiment2 ()
205   {
206     double score = 0;//opslaan van score
207     double gemscore;//opslaan van gemmscore
208     for (int bordgrootte = 4; bordgrootte <= MaxDimensie;
            bordgrootte++){
209       for (int aantalsimulaties = 1; aantalsimulaties <=64;
              aantalsimulaties = aantalsimulaties * 2){
210         for (int loop = 100; loop > 0; loop--){//loop over alle
                100 potjes
211           VierOpEenRij *vr2;//maakt bord aan
212           vr2 = new VierOpEenRij (bordgrootte, bordgrootte);//
                  aanmaken van bord met afmetingen
213           score += vr2 -> bepaalRandomScore(aantalsimulaties);//
                  bepalen van score voor dat potje
214           delete vr2;//verwijderen van het bord
215         }
216         gemscore = score/100;
```

```
217         cout << bordgrootte << " x " << bordgrootte << "met
                aantalsimulaties" <<aantalsimulaties << " heeft
                gemmiddelde score: " << gemscore << endl;
218         score = 0;//resetten van score voor hergebruik
219         if (bordgrootte + 1 < MaxDimensie + 1){
220           for (int loop = 100; loop >=0; loop--){
221             VierOpEenRij *vr2;
222             vr2 = new VierOpEenRij (bordgrootte + 1, bordgrootte
                    );//aanmaken van bord met afmetingen (+1)
223             score += vr2 -> bepaalRandomScore(aantalsimulaties);
224             delete vr2;
225           }
226           gemscore = score/100;//bepalen van de gemmiddelde
                  score
227           cout << bordgrootte + 1 << " x " << bordgrootte << "
                  met aantalsimulaties" <<aantalsimulaties << "
                  heeft gemmiddelde score: " << gemscore << endl;
228         }
229         score = 0;//resetten van score voor hergebruik
230       }
231     }
232   // TODO: implementeer deze functie, zodat het experiment met
        zo min
233   // mogelijk input van de gebruiker wordt uitgevoerd
234
235   }  // doeExperiment2
236
237   //
        ************************************************************************

238
239   void hoofdmenu ()
240   { VierOpEenRij *vr1;  // pointer, om makkeijk nieuwe objecten
        te kunnen maken
241                         // en weer weg te gooien
242     int keuze,
243         breedte, hoogte;
244     char invoernaam[MaxBestandsNaamLengte+1];
245
246     do
247     {
248       cout << endl;
249       cout << "1. Een nieuw spel starten" << endl;
250       cout << "2. Een spel inlezen" << endl;
251       cout << "3. Experiment 1 uitvoeren" << endl;
252       cout << "4. Experiment 2 uitvoeren" << endl;
253       cout << "5. Stoppen" << endl;
254       cout << endl;
255       cout << "Maak een keuze: ";
256       cin >> keuze;
257       switch (keuze)
258       { case 1: cout << "Geef de breedte van het bord (1.." <<
              MaxDimensie
259                     << "): ";
```

```
260                 cin >> breedte;
261                 cout << "Geef de hoogte van het bord (1.." <<
                        MaxDimensie
262                      << "): ";
263                 cin >> hoogte;
264                 vr1 = new VierOpEenRij (breedte, hoogte);
265                 doeSpel (vr1);
266                 delete vr1;  // netjes opruimen
267                 break;
268         case 2: vr1 = new VierOpEenRij ();
269                 cout << "Geef de naam van het tekstbestand met
                        het spel: ";
270                 cin >> invoernaam;
271                 if (vr1 -> leesIn (invoernaam))
272                   doeSpel (vr1);
273                 delete vr1;  // netjes opruimen
274                 break;
275         case 3: doeExperiment1 ();
276                 break;
277         case 4: doeExperiment2 ();
278                 break;
279         case 5: break;
280         default: cout << endl;
281                  cout << "Voer een goede keuze in!" << endl;
282       }
283
284     } while (keuze!=5);
285
286   }  // hoofdmenu
287
288   //
        ************************************************************************
289
290   int main ()
291   {
292     hoofdmenu ();
293
294     return 0;
295
296   }
```