PT

Yvo Hu s2962802

February 2021

1 Introductie

1.1

Deze opdracht gaat over het spel Aapje Omino. Het spel wordt gespeelt met twee spelers.

Het spel begint met een N aantal stenen, die worden verdeeld over de handen van de twee spelers, de pot en 1 enkele op het bord.

Elke steen heeft 4 getallen aan elke zijde, deze zijn noord, oost, zuid en west.

Men mag alleen een steen plaatsen die aan een andere steen grenst en waarvan de getallen op de weerszijden van de stenen en de stenen er omheen overeenkomen met elkaar.

De beurt begint met speler1 aan zet. Indien er geen steen geplaatst kan worden, wordt er een steen uit de pot gehaald en als deze geplaatst kan worden, moet deze geplaatst worden, anders gaat de beurt over naar de volgende speler.

Het spel eindigt wanneer de hand van een speler leeg is, of wanneer de pot leeg is en de speler aan zet geen stenen kan plaatsen.

Je score wordt berekent door de het aantal overige stenen van de tegenstander van je eigen aantal overige stenen af te trekken. Je tegenstander berekent zijn score door precies hetzelfde te doen.

De opdracht is, maak een programma die dit spel Aapje Omino nabootst aan de hand van de gegeven instructies.

2 bepaalMogelijkeZetten

De bepaalMogelijkeZetten functie zoekt alle mogelijke zetten voor de huidige speler en werkt als volgt. Er worden eerst een paar variabelen en vectoren gedeclareert en geinitialiseerd.

Ten eerste declareren we een Zet nieuweZet die wordt gebruikt om een zet in op te slaan, en de vector zetten wordt gebruikt om deze nieuweZet instanties in op te slaan.

Vervolgens declareren we een x en y array die de offsets bevatten voor de aaliggende stenen in de richting noord, oost, zuid en west. De positie van de steen in het noorden van de huidige steen is 0,-1 ten opzichte van de huidige steen. Wat correspondeert met x[0] en y[0]. Een passende paar offsets gelden voor de andere stenen in de diverse richtingen. Deze offsets kunnen we trouwens dubbel op gebruiken om de positie te bepalen van bijvoorbeeld de steen in het noorden van de steen in het noorden van de huidige steen.

Vervolgens declareren we een array arr met de richting van het getal waarmee we op het eind moeten vergelijken. Bijvoorbeeld het noordelijke getal moet met het zuidelijke getal worden vergeleken, en zuid met noord etc. Na deze declaraties krijgen we te maken met een aantal geneste for loops en if statements.
1. for, Doorloopt alle stenen
2. for, Doorloopt de hoogte van het bord
3. for, Doorloopt de breedte van het bord
4. if, Checkt of er een steen ligt op de huidige positie op het bord
5. for, Doorloopt alle buurvakken van de huidige steen
6. if, Checkt of deze buurvakken binnen het bord zijn en of deze leeg zijn.
7. for, Doorloopt alle rotaties van de steen.
8. for, Doorloopt alle richtingen van de steen om te vergelijken

Als de positie van de nieuwe steen leeg is en binnen het bord is dan checkt het of de buurvakken van deze buurvakken leeg zijn of bezet. Indien bezet dan vergelijkt het met behulp van de arr array of de specifieke steen met een specifieke rotatie past op deze plek. Dit wordt vergeleken voor alle richtingen van de nieuwe steen.

Voor elke bezette buurvak die past met de positie van de nieuwe steen wordt er 1 toegevoegd aan de zetscore.

Een boolean flag wordt gebruikt om de condities te checken, en als een zet uiteindelijk geldig is, wordt het indien het al niet in de zetten vector zit, in de zetten vector geplaatst. Een stukje code die alle huidige zetten in de zetten vector doorloopt checkt of deze al niet bestaan in de zetten vector en voorkomt herhaalde zetten.

3 besteScore

De besteScore functie zoekt de hoogst mogelijke score voor de huidige speler en werkt als volgt.

Ten eerste worden er een paar variabelen en vectoren gedeclareerd.

score2 wordt gebruikt als tijdelijke opslagplaats voor de score om uiteindelijk te worden vergeleken met de huidige hoogste score.

uiPotGehaald dient als flag om te checken of er een steen uit de pot gehaald is.

De vector zetten bevat de huidige zetten voor de huidige speler.

Hierna wordt er gelijk gecheckt of er wordt voldaan aan de voorwaarden voor een eindstand. Indien wel wordt het aantal standen geinitialiseerd op 0, en wordt geincrementeerd met 1 voor elke keer dat het bij het einde van de functie komt(recursie).

De score wordt berekent door de grootte van de hand van speler 2 minus de grootte van de hand van speler 1 te nemen. Indien de besteScore functie voor het eerst wordt aangeroepen tijdens een beurt van speler2, wordt de score -score. De score wordt uitendelijk gereturned naar de calling function en geassigned naar score2. Deze wordt later vergeleken met de huidige hoogste score.

Indien er niet aan de eindstand wordt voldaan, bepalen we alle mogelijke zetten voor de huidige speler, als deze nul is, dan halen we een steen uit de pot en checken we nogmaals. Indien nog steeds nul wisselen we van beurt. Tijdens de backtrack wordt de steen weer terug in de pot geplaatst en als deze nog bestaat, uit de hand verwijderd.

Als we gelijk al een zet kunnen plaatsen, of een zet kunnen plaatsen na eerst een steen uit de pot gehaald te hebben, dan doorlopen we alle zetten via een for loop. We doen elke zet via de functie doeZet2 die de betreffende steen uit de hand verwijdert en op het bord plaatst. De zet die is gedaan wordt toegevoegd aan de membervariabele vector oudeZetten, en er wordt van beurt gewisselt.

Dit alles wordt herhaald tot er een eindstand is bereikt, waarna er wordt gebacktracked. Nadat er eenmaal is gebacktracked wort de huidige hooste score vergeleken met de score2 en indien hoger vervangen door score2. Indien hoger wordt tevens de besteZet geinitialiseerd op de eerste zet die is gespeeld in deze reeks met behulp van de oudeZetten vector.

Vervolgens wordt de unDoeZet functie aangeroepen die de steen van het bord veegt en vervolgens van beurt wisselt. Daarna zetten we de steen terug in de hand van de speler en verwijderen we de laatste zet in oudeZetten

Als we nog niet alle zetten van de huidige stand hebben doorlopen, wordt

dit alles herhaald, waarna weer wordt gebacktracked naar een vorige stand.

Hierna wordt, als er in de huidige stand een steen uit de pot is gehaald, de gepakte steen teruggezet in de pot en verwijdert uit de hand.

4 Goede zet of slechte zet

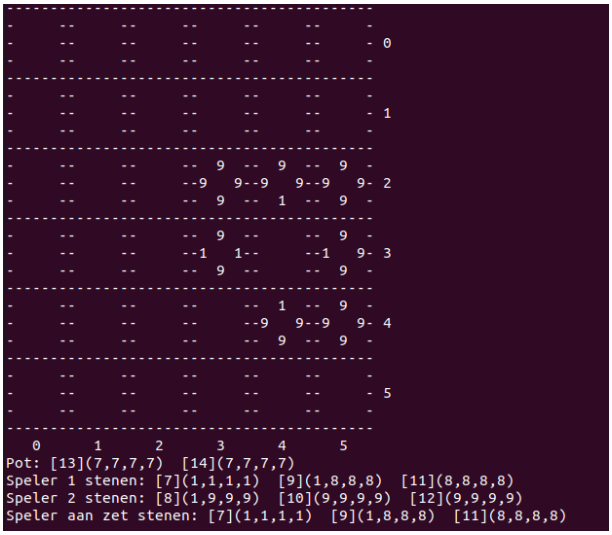


Figure 1: Beginsituatie

Speler 1 heeft 3 mogelijkheden:
zet 0; steen 7; rotatie 0-3; 3,4;

zet 1; steen 7; rotatie 0-3; 3,2;

zet 2; steen 9; rotatie 3 ; 3,4;

De goede zet zou in dit geval zet 0 zijn, want met deze zet komt de steen in aanraking met 4 andere stenen. Dit is echter niet de beste zet. De goede zet zal in dit geval leiden tot een verlies. Dit is te zien aan het volgende spelverloop

4.1 Spelverloop 1

Speler 1 speelt steen 7; rotatie 0; 3,4

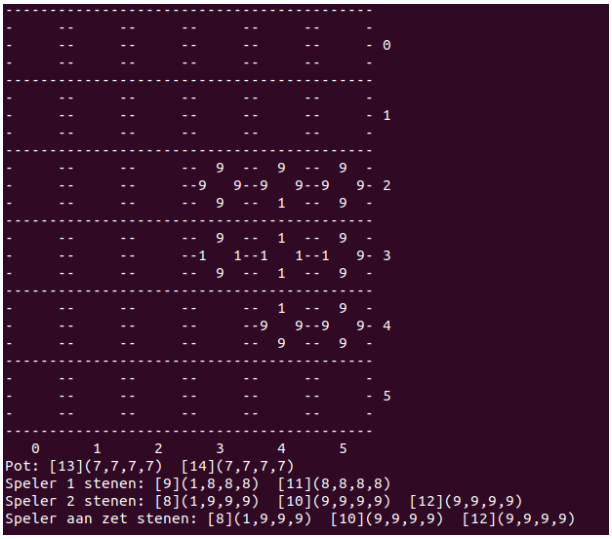
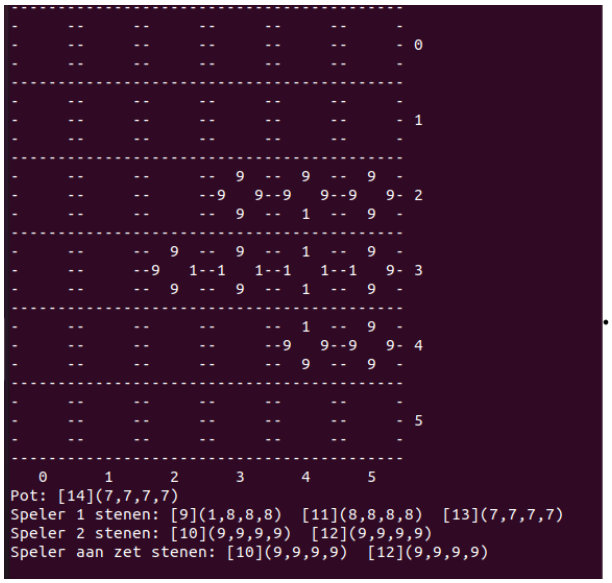


Figure 2: Beginsituatie

Speler 2 speelt steen 8; rotatie 3; 3,2

Figure 3: Speler 1 kan hierna geen steen meer zetten, en pakt een steen uit de pot maar kan nog steeds geen steen zetten. De beurt wisselt weer terug naar speler 2

Speler 2 is weer aan zet



Speler 2 speelt steen 10; rotatie 0; 1,3

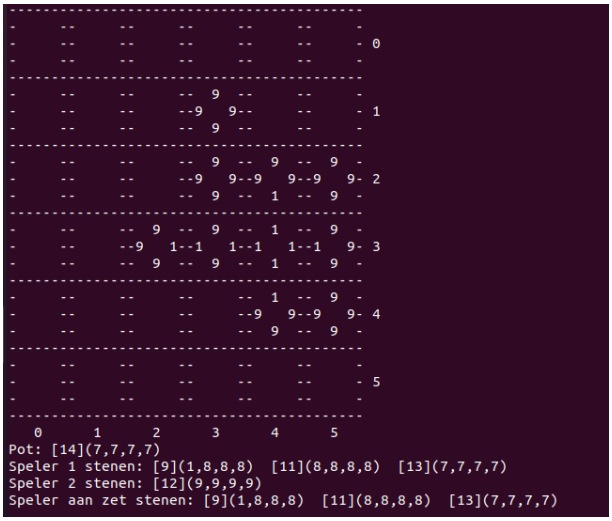
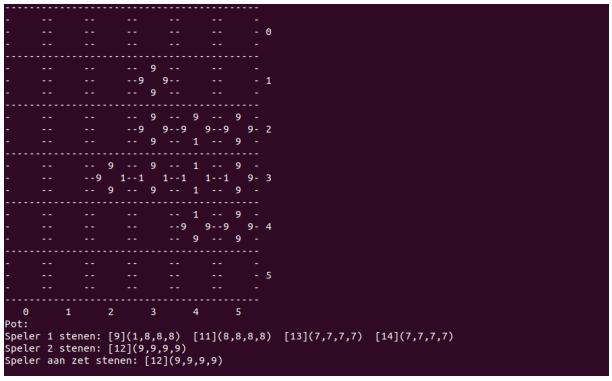


Figure 4: Speler 1 kan hierna geen steen meer zetten, en pakt een steen uit de pot maar kan nog steeds geen steen zetten. De beurt wisselt weer terug naar speler 2

Speler 2 is weer aan de beurt



Speler 2 speelt steen 12; rotatie 0; 0,3

Figure 5: Eindstand is bereikt Speler 2 heeft gewonnen

4.2 Spelverloop 2

Als in een alternatief universum, speler 1 anders had gehandeld, dan zou het anders hebben uitgepakt voor speler 2. Zoals in bijvoorbeeld het volgende scenario waarin speler 1 geen "goede" zet speelt.

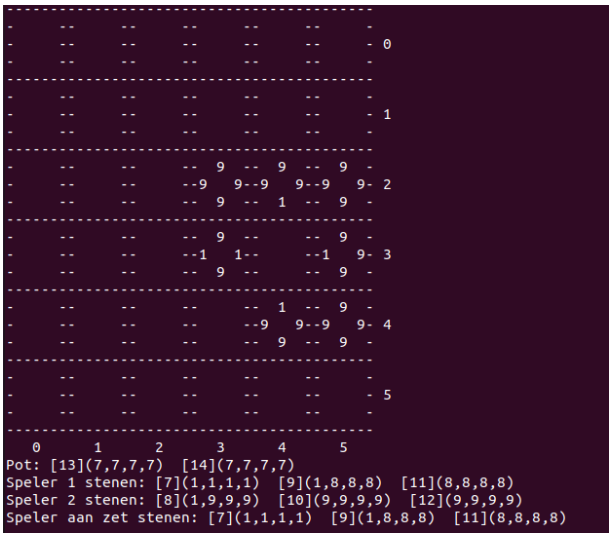
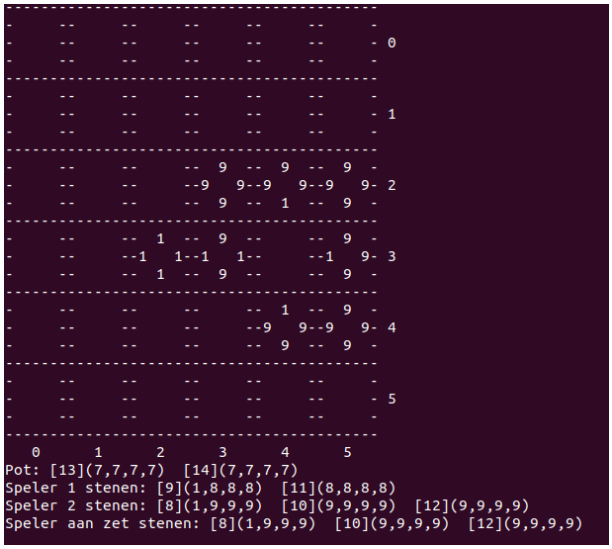
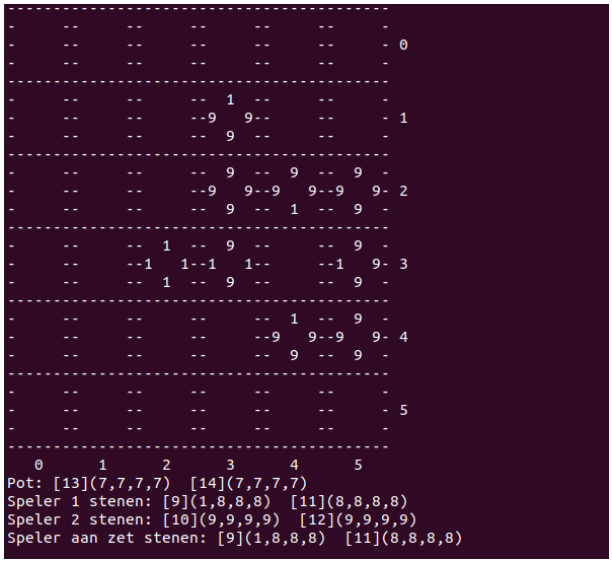


Figure 6: Beginsituatie

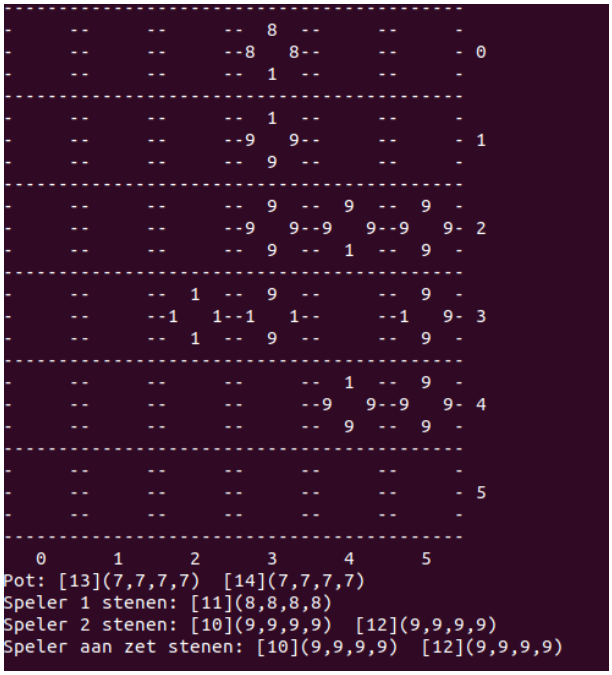
Speler 1 speelt steen 7; rotatie 0; 3,2



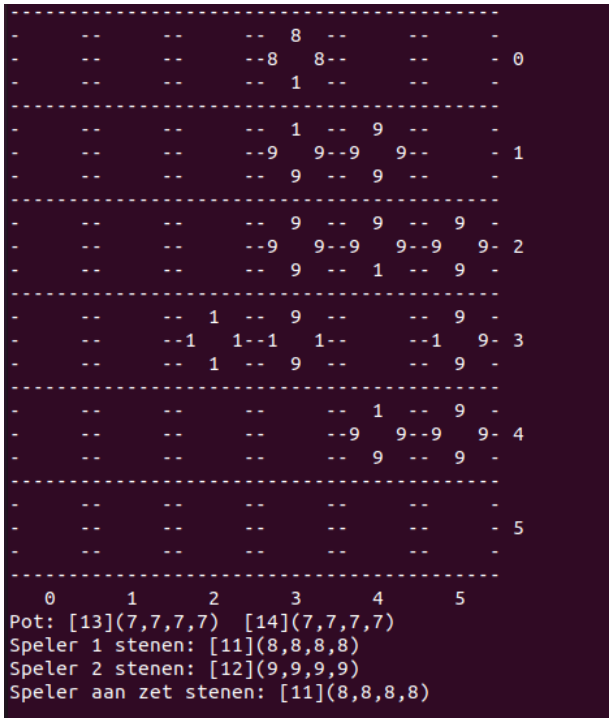
Speler 2 speelt steen 8; rotatie 0; 1,3



Speler 1 speelt steen 9; rotatie 1; 0,3



Speler 2 speelt steen 10; rotatie 0; 1,4



Speler 1 speelt steen 11; rotatie 0; 0,2

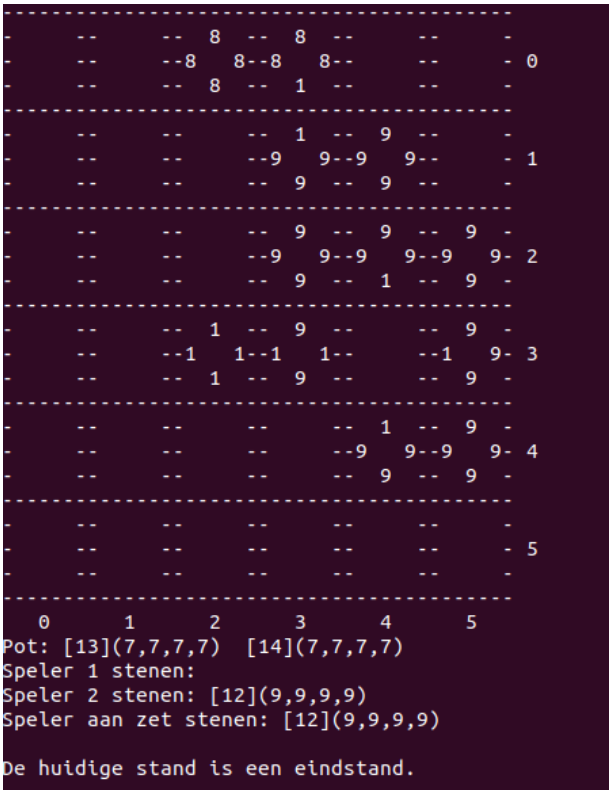


Figure 7: Eindstand bereikt

Speler 1 heeft gewonnen! Speler 1 heeft gewonnen zonder eerst de "goede" zet te spelen in de beginsituatie

5 Experiment

Dit experiment vergelijkt bij een toenemend aantal stenen de volgende waardes:
De eindscore voor speler1
Het aantal standen dat bekeken is om deze score te berekenen
De hoeveelheid tijd die daarvoor nodig is
De score voor speler 1 wanneer zij steeds een willekeurige 'goede zet' speelt, terwijl speler 2 steeds een beste zet speelt

De resultaten worden opgeslagen in het bestand experiment.txt
Een iteratie van deze functie geeft de volgende resultaten.

5.1 Resultaten

```
nr, besteScore, aantalStanden, tijd(s), scores2
8, 1, 156.6, 0.041403, 1
9, 0.9, 91.2, 0.023929, 1
10, 1, 103.6, 0.02537, 1.1
11, 0.7, 215.2, 0.124366, 0.7
12, 1.8, 2216, 1.29115, 1.9
13, 1.6, 2059, 1.50342, 2.29
14, 1.8, 1652, 1.0219, 1.8
15, 1.9, 1438.7, 0.946275, 2.29
16, 3.3, 117856, 121.113, 3
17, 3.4, 132674, 211.003, 3.8
18, 3.3, 36285.2, 60.7216, 3.7
19, 3.8, 148189, 474.81, 4
20, 5.09, 5.48088e+06, 14418.3, 5.2
21, 4.9, 3.14643e+06, 8625.63, 5.3
22, 5.3, 1.47546e+06, 5420.37, 5.2
```

We kunnen met behulp van deze gegevens uitgaan van de volgende beweringen. Des te meer beginstenen er zij. dan:
Kan de speler een hogere besteScore behalen
Neemt het aantal standen om de besteScore te berekenen exponentieel toe.
Neemt de tijd om de besteScore te berekenen exponentieel toe

Zaken die fout zijn gegaan bij het experiment.

De berekening voor het aantal beginstenen is int(i/4) Dit betekent dat we de getallen achter de komma weg moeten halen, en dat we naar beneden moeten afronden om deze te bereken. De functie round() in cmath had hier beter gebruikt kunnen worden, maar deze headerfile mocht niet worden gebruikt. Dit betekent dat het aantalStanden en de tijd na elk 4tal beginstenen een stuk begint te dalen.

De besteScore en scores2 zijn twee verschillende pseudorandom gegenereerde spellen en de waardes tussen elkaar lijken geen enkel verband te hebben, afgezien dat de waardes van score2 gemiddeld gezien lichtelijk hoger zijn. Aan de andere kant kan de berekening van besteScore en score2 verkeerd zijn in welk geval de waardes helemaal nergens op slaan.

6 Appendix

Alle .h bestanden

6.1 aapjeomino

```
// Definitie van klasse AapjeOmino

#ifndef AapjeOminoHVar  // voorkom dat dit bestand meerdere keren
#define AapjeOminoHVar  // ge-include wordt

#include <vector>
#include "zet.h"
using namespace std;

const int SteenZijden = 4;   //Aantal zijdes van een steen
const int MaxDimensie = 10;  //Maximaal aantal rijen en maximaal aantal kolommen

class AapjeOmino
{ public:
    AapjeOmino ();                                  //Default constructor.

    bool leesIn (const char* invoernaam);           //Lees bestand in
```

```
    bool eindstand ();                          //Check of eindstand is bereikt

    void drukAf ();                             //Druk de hele stand (bord, sten
                                                //speler aan beurt) af op het sc

    vector<Zet> bepaalMogelijkeZetten ();       //Bepaal alle mogelijke zetten o

    int haalSteenUitPot ();                     //Haal steen uit pot

    void wisselSpeler ();                       //Wissel van beurt

    bool doeZet (Zet zet);                      //Doe een zet in zetten

    vector<Zet> bepaalGoedeZetten ();           //Bepaal de beste zetten op basi

    int besteScore (Zet &besteZet, long long &aantalStanden);
//Bepaal best mogelijke score op basis van de huidige stand

    int besteScore2 (Zet &besteZet, long long &aantalStanden);
//Doeexperimenten specificaties

    bool genereerRandomSpel (int hoogte0, int breedte0,
//Genereer random spel
            int nrStenen0, int nrStenenInHand0, int rij0, int kolom0,
            int minGetal, int maxGetal);
    bool aantalStandenFlag = true;
  private:
    int leesGetal (char& letter, ifstream& invoer);
//Lees getal uit bestand

    void drukAfStenen (vector<vector<int>> stenen);
//Druk alle stenen af in de vector stenen

    void bepaalMogelijkeZettenf (int rij, int kolom, int nr, int zijde);

    vector<int> schuif (vector<int> vec, int schuif);
//Verschuif de elementen in vec met schuif naar rechts en weer terug naar begin

    pair<int,int> bord[MaxDimensie][MaxDimensie];
// in een pair kunnen we een steennummer en een rotatie opslaan

    vector<vector<int>> stenen, speler1Stenen, speler2Stenen, potStenen;
//Totale aantal stenen, in hand van spelers, en de pot
```

```
    vector<vector<int>>* huidigStenen;
//Pointer naar de hand van de huidige speler

    vector<Zet> oudeZetten;
//Vector met alle zetten die gedaan zijn door beide spelers

    void wisselSpeler2 ();
//Wissel van beurt

    bool doeZet2 (Zet zet);
//Doe een zet in zetten

    bool undoeZet (Zet zet);
//Verwijder steen van bord


    int huidigStand;
//Counter met de huidige stand

    int hoogte, breedte,  // van het bord
        nrStenen,         // totaal aantal stenen in het spel
        aanBeurt,         // speler die aan de beurt is
        beginStenen,      // aantal beginstenen per speler
        rij, kolom,       // positie steen 0
        score;            // score initialiseren


};

#endif
```

6.2 zet

```
// Definitie van klasse Zet

#ifndef ZetHVar  // voorkom dat dit bestand meerdere keren
#define ZetHVar  // ge-include wordt

class Zet
{ public:

    // Default constructor
    Zet ();

    // Geef de vier velden waardes die in combinatie niet voor kunnen
    // komen, als default voor 'geen echte zet'.
```

```
    void setDefaultWaardes ();

    // Geef alle vier velden een waarde.
    // Controleer nog wel of het zinnige waardes zijn.
    void setWaardes (int i0, int r0, int rij0, int kolom0, int som0);

    // Vier getters:
    int getI ();
    int getR ();
    int getRij ();
    int getKolom ();
    int getSom ();

    // Druk de vier waardes van de zet af op het scherm
    void drukAf ();

  private:
    int i,  // nummer van de steen
        r,  // rotatie
        rij, kolom, // vakje op het bord
        som; //Aantal buurvakjes

};

#endif
```

6.3 standaard

```
// Defintie van standaard functies.

#ifndef StandaardHVar  // om te voorkomen dat dit .h bestand meerdere keren
#define StandaardHVar  // wordt ge-include

// Controleer of variabele met naam 'variabele' een waarde 'waarde' heeft
// die tussen (inclusief) minWaarde en maxWaarde in ligt.
// Zo nee, geef een passende foutmelding.
//
// Voorbeeld van aanroep:
//   if (integerInBereik ("teller", teller, 0, 1000))
//     ...
bool integerInBereik (const char *variabele, int waarde,
                      int minWaarde, int maxWaarde);

// Genereer een random geheel getal r waarvoor  min <= r <= max.
// Pre: min <= max;
int randomGetal (int min, int max);
```

```
#endif
```

7 Alle .cc bestanden

7.1 aapjeomino

```
// Implementatie van klasse AapjeOmino

#include <iostream>
#include <fstream>
#include <vector>
#include "standaard.h"
#include "aapjeomino.h"
#include <unistd.h>


//***********************************************************************
//Default constructor
AapjeOmino::AapjeOmino ()
{
  huidigStand = 0;                  // Huidige stand van het spel
  aanBeurt = 1,                     // Speler die aan de beurt is in het begin
  huidigStenen = &speler1Stenen;    // Pointer naar de hand van de speler die aan
  score = -999;                     // score initialiseren
}  // default constructor

//***********************************************************************
//Leest een bestand in en initialiseert een spel op basis van deze gegevens
bool AapjeOmino::leesIn (const char* invoernaam)
{
  ifstream invoer;
  char letter;
  int som;
//Waarde voor de getallen

  invoer.open(invoernaam,ios::in);
//Check of het bestand kan worden geopend
  if(invoer.fail())
  {
    cout << "fail";
    return 0;
  }

  hoogte = leesGetal(letter, invoer);
//Lees het bestand in, en sla de gegevens op bij de bijbehorende variabelen
```

```
  breedte = leesGetal(letter, invoer);
  nrStenen = leesGetal(letter, invoer);
  beginStenen = leesGetal(letter, invoer);
  rij = leesGetal(letter, invoer);
  kolom = leesGetal(letter, invoer);

  som = leesGetal(letter,invoer);
//Lees iedere steen in en sla deze op
  for(int i = 0; i < nrStenen; i++){
    vector<int> steen;
    for(int j = 0; j < SteenZijden; j++)
    {
      steen.push_back(som);
      som = leesGetal(letter,invoer);
    }
    steen.push_back(i);
    stenen.push_back(steen);
    if(invoer.eof())
    {
      break;
    }
  }

  if(hoogte > MaxDimensie || breedte > MaxDimensie)
//Bord dimensies zijn binnen de limieten
    std::cout << "hoogte of breedte zijn buiten de toegestane limieten";
  else if(nrStenen != int(stenen.size()))
//Het aantal stenen komt overeen met het werkelijk aantal stenen die beschreven
    std::cout << "Aantal stenen komt niet overeen met de beschrijving";
  else if(rij >= hoogte || kolom >= breedte)
//Positie van de beginsteen is binnen het bord
    std::cout << "Positie van beginsteen is buiten het bord";
  else if(nrStenen < (beginStenen * 2) + 1)
//Er zijn genoeg stenen om te verdelen over het spel
    std::cout << "Het aantal stenen is niet genoeg om over het hele spel te vers
  else
  {
    for(int i = 0; i < hoogte; i++)
//Initialiseer het bord
    {
      for(int j = 0; j < breedte; j++)
      {
        bord[i][j].first = -1;
      }
    }
```

```
    for(int i = 0; i < nrStenen; i++)
//Verspreid de stenen over het bord, de spelers, en de pot
    {
      if(i == 0)
//Plaats eerste steen op bord
      {
        bord[rij][kolom].first = 0;
        bord[rij][kolom].second = 0;
      } else if(i <= beginStenen*2)
      {
        if(i & 1)
//Bitwise AND om de stenen te verdelen over even en oneven indexes
        {
          speler1Stenen.push_back(stenen[i]);
        } else
        {
          speler2Stenen.push_back(stenen[i]);
        }
      }else
      {
        potStenen.push_back(stenen[i]);
      }
    }
    return true;
  }
  return false;
}  // leesIn

//*********************************************************************
//Check of er aan de eindcondities wordt voldaan
bool AapjeOmino::eindstand ()
{
  if((speler1Stenen.size() == 0 || speler2Stenen.size() == 0) ||
//1 van beide spelers heeft geen stenen meer of
    (bepaalMogelijkeZetten().size() == 0 && potStenen.size() == 0)
//De pot is leeg, maar er kunnnen geen zetten meer gedaan worden
  )
  {
    return true;
  }
  return false;

}  // eindstand

//*********************************************************************
//Drukt de stand van het gehele spel af. Bord, speler handen, pot,
```

```
void AapjeOmino::drukAf()
{
  for(int j = 0; j < breedte; j++)
  {
    cout << "--------";
  }
  cout << "\n";
  for(int i = 0; i < hoogte; i++)
  {
    for(int j = 0; j < breedte; j++)
    {
      if(bord[i][j].first >= 0)
      {
        cout << "-  " << schuif(stenen[bord[i][j].first],bord[i][j].second)[0];
//Noord getal
        if(!(schuif(stenen[bord[i][j].first],bord[i][j].second)[0] > 9))
        {
          cout << " ";
        }
        cout << " -";
      }else
      {
        cout << "-      -";
      }
    }
    cout << "\n";
    for(int j = 0; j < breedte; j++)
    {
      if(bord[i][j].first >= 0)
      {
        cout << "-" << schuif(stenen[bord[i][j].first],bord[i][j].second)[3];
//West getal
        if(!(schuif(stenen[bord[i][j].first],bord[i][j].second)[3] > 9))
        {
          cout << " ";
        }
        cout << " ";
        if(!(schuif(stenen[bord[i][j].first],bord[i][j].second)[1] > 9))
        {
          cout << " ";
        }
        cout << schuif(stenen[bord[i][j].first],bord[i][j].second)[1];
//Oost getal
        cout << "-";
      }
      else
```

```
      {
        cout << "-      -";
      }
    }
    cout << " " << i << "\n";
    for(int j = 0; j < breedte; j++)
    {
      if(bord[i][j].first >= 0)
      {
        cout << "-  " << schuif(stenen[bord[i][j].first],bord[i][j].second)[2];
//Zuid getal
        if(!(schuif(stenen[bord[i][j].first],bord[i][j].second)[2] > 9))
        {
          cout << " ";
        }
        cout << " -";
      }else
      {
        cout << "-      -";
      }
    }
    cout << "\n";
   for(int j = 0; j < breedte; j++)
    {
      cout << "--------";
    }
    cout << "\n";
  }
  for(int j = 0; j < breedte; j++)
  {
    cout << "    " << j << "    ";
  }
  cout << "\n";
  cout << "Pot:";
  drukAfStenen(potStenen);
  cout << "Speler 1 stenen:";
  drukAfStenen(speler1Stenen);
  cout << "Speler 2 stenen:";
  drukAfStenen(speler2Stenen);
  cout << "Speler aan zet stenen:";
  drukAfStenen(*huidigStenen);



}  // drukAf
```

```
//**************************************************************************
//Haalt een steen uit de pot en doet het in de hand van de huidige speler die aa
int AapjeOmino::haalSteenUitPot ()
{
// wisselSpeler();
  if(int(potStenen.size()) >= 1)
//Haal steen uit pot als er nog stenen in de pot zitten
  {
    (*huidigStenen).push_back(potStenen[0]);
    potStenen.erase(potStenen.begin());
    return (*huidigStenen).back()[4];
//Return steennummer van steen uit de pot gehaald
  }else
  {
    return -1;
//Anders return -1
  }

}  // haalSteenUitPot

//**************************************************************************
//Wissel speler en verander de pointer van huidigStenen naar de hand van de ande
void AapjeOmino::wisselSpeler ()
{
  huidigStand++;                    //Incrementeer de huidige stand van het spe
  if(aanBeurt){
    huidigStenen = &speler2Stenen;
    aanBeurt = 0;
  }else{
    huidigStenen = &speler1Stenen;
    aanBeurt = 1;
  }
}  // wisselSpeler


void AapjeOmino::wisselSpeler2()    //Identiek aan wisselSpeler minus de huidig
{
  if(aanBeurt){                     //Wissel speler en verander der pointer van
    huidigStenen = &speler2Stenen;
    aanBeurt = 0;
  }else{
    huidigStenen = &speler1Stenen;
    aanBeurt = 1;
  }
}  // wisselSpeler
```

```
//*************************************************************************
 //Doezet implementatie voor handmatige zetten
bool AapjeOmino::doeZet (Zet zet)
{
  score = -999;
//Reset de score om besteScore later te berekenen
  for(int i = 0; i < int(huidigStenen->size()); i++)
//Herhaal voor iedere steen in hand
  {
    if((*huidigStenen)[i][4] == zet.getI())
//Check of de steennummer van de zet overeenkomt met steennummer in hand
    {
      (*huidigStenen).erase((*huidigStenen).begin() + i);
//Verwijder steen uit hand van speler
    }
  }
  bord[zet.getRij()][zet.getKolom()].first = zet.getI();
//Plaats steen op bord met rotatie
  bord[zet.getRij()][zet.getKolom()].second = zet.getR();
  oudeZetten.push_back(zet);
//Voeg een zet toe aan oudeZetten

  wisselSpeler();
  return true;
}

bool AapjeOmino::doeZet2 (Zet zet)
//Identiek aan doeZet minus score reset voor gebruik in besteScore,
{
  for(int i = 0; i < int(huidigStenen->size()); i++)
  {

    if((*huidigStenen)[i][4] == zet.getI())
    {
      (*huidigStenen).erase((*huidigStenen).begin() + i);
    }
  }
  bord[zet.getRij()][zet.getKolom()].first = zet.getI();
  bord[zet.getRij()][zet.getKolom()].second = zet.getR();
  oudeZetten.push_back(zet);

  wisselSpeler2();
  return true;
}
```

```
// undoeZet Undoe de zetten door stenen van het bord te vegen
bool AapjeOmino::undoeZet (Zet zet)
{
  bord[zet.getRij()][zet.getKolom()].first = -1;
//Steennummer op bord[rij][kolom] word -1
  wisselSpeler2();

  return true;

}  // undoeZet

//*********************************************************************

vector<Zet> AapjeOmino::bepaalGoedeZetten ()
{
  int buren = 0;                                //Aantal bezette buurvakjes
  vector<Zet> besteZetten;                      //Beste zetten
  vector<Zet> zetten;                           //Alle mogelijke zetten
  Zet goedeZet;                                 //Goede zet

  zetten = bepaalMogelijkeZetten();             //Neem alle mogelijke zetten
  for(int i = 0; i<int(zetten.size()); i++)     //Neem de het hoogst aantal bezett
  {
    if(zetten[i].getSom() > buren)
    {
      buren = zetten[i].getSom();
    }
  }

  for(int i = 0; i<int(zetten.size()); i++)     //Zet alle zetten waarvan de zetsc
  {
    if(zetten[i].getSom() == buren)
    {
      goedeZet.setWaardes(zetten[i].getI(),zetten[i].getR(),zetten[i].getRij(),z
      besteZetten.push_back(goedeZet);
    }
  }

  return besteZetten;                            //Retourneer de beste zetten

}  // bepaalGoedeZetten

//*********************************************************************
//Berekent de hoogst mogelijke score voor de speler aan zet in de huidige stand
```

```
int AapjeOmino::besteScore (Zet &besteZet, long long &aantalStanden)
{
  int score2 = -999;
//Tijdelijke variabele om de score mee te vergelijken
  int uitPotGehaald = -1;
//Status of een steen uit de pot gehaald is
  vector<Zet> zetten;
//Vector met de huidige zetten voor de huidige speler

  if(aantalStandenFlag){
//Initialiseer het aantal standen op 0 voor de eerste  iteratie
    aantalStandenFlag = false;
    aantalStanden = 0;
  }

  if(eindstand())
//Base case check of er wordt voldaan aan eindcondities
  {
    score2 = speler2Stenen.size() - speler1Stenen.size();
//Bereken score

    if(huidigStand & 1){
          score2 = -score2;
    }
    return score2;
  }else
  {
    zetten = bepaalMogelijkeZetten();
//Laat alle mogelijke zetten zien
    if(zetten.size() == 0)
//Check of er geen zetten mogelijk zijn
    {
      uitPotGehaald = haalSteenUitPot();
//Haal steen uit pot en laat de alle mogelijke zetten weer zien
      zetten = bepaalMogelijkeZetten();
    }
    if(zetten.size() == 0)
//Als er nog steeds geen zetten mogelijk zijn, wissel beurt
    {
      wisselSpeler2();
      score2 = besteScore(besteZet, aantalStanden);
      wisselSpeler2();
//Wissel beurt weer en backtrack
      if(score2 > score){
//Vergelijk nieuwe score met oude score
        score = score2;
```

```
        if(oudeZetten.size() > 0)
//Check of het programma al niet stopt voor de eerste zet
        {
          besteZet = oudeZetten[huidigStand];
        }
        else{
          besteZet.setDefaultWaardes();
//Initialiseer besteZet als er geen zet mogelijk is
        }
      }
      potStenen.push_back(stenen[uitPotGehaald]);
//Undoe de zetten door steen terug te zetten in de pot
      for(int j = 0; j < int((*huidigStenen).size()); j++)
//Verwijder de gepakte steen als hij nog in de hand zit
      {
        if((*huidigStenen)[j][4] == uitPotGehaald)
        {
          (*huidigStenen).erase((*huidigStenen).begin() + j);
        }
      }

    }
    else
    {
      for(int i = 0;  i < int(zetten.size()); i++)
      {
        doeZet2(zetten[i]);
//Doe Zet volgens de beschrijvingen in zetten[i]
        aantalStanden++;
//Verhoog Aantal standen met 1 na elke zet
        score2 = besteScore(besteZet, aantalStanden);

        if(score2 > score)
//Vergelijk nieuwe score met oude score
        {
          score = score2;
          if(oudeZetten.size() > 0)
//Check of het programma al niet stopt voor de eerste zet
          {
           besteZet = oudeZetten[huidigStand];
          }
          else{
              besteZet.setDefaultWaardes();
//Initialiseer besteZet als er geen zet mogelijk is
          }
        }
```

```
        undoeZet(oudeZetten.back());
//Verwijder steen van bord
        (*huidigStenen).push_back(stenen[oudeZetten.back().getI()]);
//Zet verwijderde steen terug in hand
        oudeZetten.pop_back();
//Verwijder laatse zet in oudeZetten
      }
      if(uitPotGehaald != -1)
//Check of er steen gepakt is en gelijk neergezet
      {
        potStenen.push_back(stenen[uitPotGehaald]);
//Undoe de zetten door steen terug te zetten in de pot
        for(int j = 0; j < int((*huidigStenen).size()); j++)
//Verwijder de gepakte steen als hij nog in de hand zit
        {
          if((*huidigStenen)[j][4] == uitPotGehaald)
          {
            (*huidigStenen).erase((*huidigStenen).begin() + j);
          }
        }
      }
    }
  }

  return score;

}  // besteScore
//Identiek aan besteScore, maar aangepast aan de eisen van de doeexperimenten op
int AapjeOmino::besteScore2 (Zet &besteZet, long long &aantalStanden)
{
  int score2 = -999;
//Tijdelijke variabele om de score mee te vergelijken
  int uitPotGehaald = -1;
//Status of een steen uit de pot gehaald is
  vector<Zet> zetten;
//Vector met de huidige zetten voor de huidige speler

  if(aantalStandenFlag){
//Initialiseer het aantal standen op 0 voor de eerste  iteratie
    aantalStandenFlag = false;
    aantalStanden = 0;
  }

  if(eindstand())
//Base case check of er wordt voldaan aan eindcondities
  {
```

```
    score2 = speler2Stenen.size() - speler1Stenen.size();
//Bereken score

    if(huidigStand & 1){
         score2 = -score2;
    }
    return score2;
  }else
  {
    if(aanBeurt)
    {
      zetten = bepaalGoedeZetten();
//Laat alle goede zetten zien
    }
    else
    {
      zetten = bepaalMogelijkeZetten();
//Laat alle mogelijke zetten zien
    }
    if(zetten.size() == 0)
//Check of er geen zetten mogelijk zijn
    {
      uitPotGehaald = haalSteenUitPot();
//Haal steen uit pot en laat de alle mogelijke zetten weer zien
      if(aanBeurt)
      {
        zetten = bepaalGoedeZetten();
//Laat alle goede zetten zien
      }
      else
      {
        zetten = bepaalMogelijkeZetten();
//Laat alle mogelijke zetten zien
      }
    }
    if(zetten.size() == 0)
//Als er nog steeds geen zetten mogelijk zijn, wissel beurt
    {
      wisselSpeler2();
      score2 = besteScore(besteZet, aantalStanden);
      wisselSpeler2();
//Wissel beurt weer en backtrack
      if(score2 > score){
//Vergelijk nieuwe score met oude score
        score = score2;
        if(oudeZetten.size() > 0)
```

```
//Check of het programma al niet stopt voor de eerste zet
        {
          besteZet = oudeZetten[huidigStand];
        }
        else{
          besteZet.setDefaultWaardes();
//Initialiseer besteZet als er geen zet mogelijk is
        }
      }
      potStenen.push_back(stenen[uitPotGehaald]);
//Undoe de zetten door steen terug te zetten in de pot
      for(int j = 0; j < int((*huidigStenen).size()); j++)
//Verwijder de gepakte steen als hij nog in de hand zit
      {
        if((*huidigStenen)[j][4] == uitPotGehaald)
        {
          (*huidigStenen).erase((*huidigStenen).begin() + j);
        }
      }

    }
    else
    {
      for(int i = 0;  i < int(zetten.size()); i++)
      {
        doeZet2(zetten[i]);
//Doe Zet volgens de beschrijvingen in zetten[i]
        aantalStanden++;
//Verhoog Aantal standen met 1 na elke zet
        score2 = besteScore(besteZet, aantalStanden);

        if(score2 > score)
//Vergelijk nieuwe score met oude score
        {
          score = score2;
          if(oudeZetten.size() > 0)
//Check of het programma al niet stopt voor de eerste zet
          {
           besteZet = oudeZetten[huidigStand];
          }
          else{
              besteZet.setDefaultWaardes();
//Initialiseer besteZet als er geen zet mogelijk is
          }
        }
        undoeZet(oudeZetten.back());
```

```
//Verwijder steen van bord
        (*huidigStenen).push_back(stenen[oudeZetten.back().getI()]);
//Zet verwijderde steen terug in hand
        oudeZetten.pop_back();
//Verwijder laatse zet in oudeZetten
      }
      if(uitPotGehaald != -1)
//Check of er steen gepakt is en gelijk neergezet
      {
        potStenen.push_back(stenen[uitPotGehaald]);
//Undoe de zetten door steen terug te zetten in de pot
        for(int j = 0; j < int((*huidigStenen).size()); j++)
//Verwijder de gepakte steen als hij nog in de hand zit
        {
          if((*huidigStenen)[j][4] == uitPotGehaald)
          {
            (*huidigStenen).erase((*huidigStenen).begin() + j);
          }
        }
      }
    }
  }

  return score;

}  // besteScore

//Genereert een random spel
//**********************************************************************

bool AapjeOmino::genereerRandomSpel (int hoogte0, int breedte0,
       int nrStenen0, int nrStenenInHand0, int rij0, int kolom0,
       int minGetal, int maxGetal)
{
  hoogte = hoogte0;
//Initialiseer variabelen
  breedte = breedte0;
  nrStenen = nrStenen0;
  beginStenen = nrStenenInHand0;
  rij = rij0;
  kolom = kolom0;

  for(int i = 0; i < nrStenen0; i++)
  {
    vector<int> steen;
    for(int j = 0; j < SteenZijden; j++)
```

```
    {
      steen.push_back(randomGetal(minGetal, maxGetal));
    }
    steen.push_back(i);
    stenen.push_back(steen);
  }

  for(int i = 0; i < hoogte; i++)
//Initialiseer het bord
  {
    for(int j = 0; j < breedte; j++)
    {
      bord[i][j].first = -1;
    }
  }

  for(int i = 0; i < nrStenen; i++)
//Verspreid de stenen over het bord, de spelers, en de pot
  {
    if(i == 0)
//Plaats eerste steen op bord
    {
      bord[rij][kolom].first = 0;
      bord[rij][kolom].second = 0;
    } else if(i <= beginStenen*2)
    {
      if(i & 1)
//Bitwise AND om de stenen te verdelen over even en oneven indexes
      {
        speler1Stenen.push_back(stenen[i]);
      } else
      {
        speler2Stenen.push_back(stenen[i]);
      }
    }else
    {
      potStenen.push_back(stenen[i]);
    }
  }

  return true;

}  // genereerRandomSpel

//*********************************************************************
//Leest een getal uit een bestand
```

```
int AapjeOmino::leesGetal(char& letter, ifstream& invoer)
{
  int sum = 0;
//Getal waarde
  letter = invoer.get();
  while(!isdigit(letter) && !invoer.eof())
  {
    letter = invoer.get();
//Pak getal uit invoer
  }
  while(isdigit(letter)){
//Indien meer getallen achter elkaar, vermenigvuldig met 10
      sum *= sum > 0 ? 10 : sum;
      sum += (letter - '0');
      letter = invoer.get();
  }
  return sum;
}

vector<int> AapjeOmino::schuif(vector<int> vec, int shift)
//Herordert een vector met een verschuiving
{
  vector<int> vec2 = vec;
  for(int i = 0; i < SteenZijden; i++){
  vec[i] = vec2[(4-shift + i)%SteenZijden ];
// shift vector elementen met n = shift naar rechts tot het einde en weer naar h
  }
  return vec;
}

//Print alle stenen in een set
void AapjeOmino::drukAfStenen(vector<vector<int>> stenen)
{
  for(int i = 0; i < int(stenen.size()); i++)
  {
    cout << " [" << stenen[i][4] << "]"<< "(" << stenen[i][0] << "," << stenen[i
    cout << stenen[i][2] << "," << stenen[i][3] << ") ";
  }
  cout << endl;
}

//Bepaalt alle mogelijke zetten voor de huidige speler
vector<Zet> AapjeOmino::bepaalMogelijkeZetten ()
{
  vector<Zet> zetten;                  // Vector met alle mogelijke zetten
  Zet nieuweZet;                       // Element voor in zetten
```

```
  int x[SteenZijden] = {0, 1, 0, -1}; // Kolom offset voor buurvak
  int y[SteenZijden] = {-1, 0, 1, 0}; // Rij    offset voor buurvak
  int arr[] = {2,3,0,1};              // Richting vergelijking op basis van de p

  for(int i = 0; i < int((*huidigStenen).size()); i++)
//Herhaal voor elke steen
  {
    for(int j = 0;j < hoogte; j++)
    {
      for(int k = 0;k < breedte; k++)
      {
        if(bord[j][k].first >= 0)                          //Check of er een steen
        {
          for(int l = 0;l < SteenZijden; l++)              //Check alle buurvakken
          {
            if(j+y[l] < 0 || k+x[l] < 0||j+y[l] >= hoogte ||k+x[l] >= breedte){}
//Check of de buurvak binnen het bord is
            else if(bord[j+y[l]][k+x[l]].first < 0)
//Check of de buurvak leeg is
            {
              for(int m = 0; m < SteenZijden; m++)
//Check alle rotaties van de buurvak
              {
                bool flag = true;
                int som = 0;
                for(int n = 0; n < SteenZijden; n++)
//Check alle buurvakken van de buurvak
                {
                  if(j+y[l]+y[n] < 0 || k+x[l]+x[n] < 0 || j+y[l]+y[n] >= hoogte
//Check of de buurvakken van de buurvak binnen het bord is
                  }
                  else if(bord[j+y[l]+y[n]][k+x[l]+x[n]].first >= 0)
//Check of de buurvakken van de buurvak niet leeg is
                  {
                    if(schuif(stenen[bord[j+y[l]+y[n]][k+x[l]+x[n]].first],bord[
//Vergelijk alle buurvakken van de buurvak die niet leeg zijn
                    {
                      flag = false;
                    }else if(schuif(stenen[bord[j+y[l]+y[n]][k+x[l]+x[n]].first]
                      som++;
                    }
                  }
                }
                if(flag)                        //Check of een zet geldig is
                {
                  bool flag2 = true;
```

```
//Flag om herhalingen te detecteren
                    for(int p = 0;  p < int(zetten.size()); p++)
//Detecteer herhalingen, en skip deze zetten
                    {
                       if(
                          zetten[p].getI() == (*huidigStenen)[i][4] &&
                          zetten[p].getR() == m &&
                          zetten[p].getRij() == j+y[l] &&
                          zetten[p].getKolom() == k+x[l])
                      {
                        flag2 = false;
                      }
                    }
                    if(flag2)                    //Check of er geen herhalende zetten
                    {                            //Stop de zet in de zetten vector
                      nieuweZet.setWaardes((*huidigStenen)[i][4],m,j+y[l],k+x[l],s
                      zetten.push_back(nieuweZet);
                    }
                  }
                }
              }
            }
          }
        }
      }
    return zetten;                       //Retourneer alle mogelijke zetten

}  // bepaalMogelijkeZetten
```

7.2 zet

```
// Implementatie van klasse Zet

#include "standaard.h"
#include "zet.h"
#include <iostream>
using namespace std;

//*************************************************************************

Zet::Zet ()
{
  i = -1;
  r = -1;
  rij = -1;
```

```
  kolom = -1;
  som = -1;

}  // default constructor

//*************************************************************************

void Zet::setDefaultWaardes ()
{
  i = -1;
  r = -1;
  rij = -1;
  kolom = -1;
  som = -1;

}  // setDefaultWaardes

//*************************************************************************

void Zet::setWaardes (int i0, int r0, int rij0, int kolom0, int som0)
{
  if (i0>=0)
    i = i0;
  else
  { i = 0;
    cout << endl;
    cout << "i0 mag niet negatief zijn, maar is " << i0 << endl;
  }

  if (integerInBereik ("r0", r0, 0, 3))
    r = r0;
  else
    r = 0;

  if (rij0>=0)
    rij = rij0;
  else
  { rij = 0;
    cout << endl;
    cout << "rij0 mag niet negatief zijn, maar is " << rij0 << endl;
  }

  if (kolom0>=0)
    kolom = kolom0;
  else
  { kolom = 0;
```

```
    cout << endl;
    cout << "kolom0 mag niet negatief zijn, maar is " << kolom0 << endl;
  }

  if (som0>=0)
    som = som0;
  else
  { som = 0;
    cout << endl;
    cout << "som0 mag niet negatief zijn, maar is " << som0 << endl;
  }

}  // setWaardes

//*************************************************************************

int Zet::getI ()
{
  return i;

}  // getI

//*************************************************************************

int Zet::getR ()
{
  return r;

}  // getR

//*************************************************************************

int Zet::getRij ()
{
  return rij;

}  // getRij

//*************************************************************************

int Zet::getKolom ()
{
  return kolom;

}  // getKolom
```

```
//*********************************************************************
int Zet::getSom ()
{
  return som;

}  // getKolom

//*********************************************************************

void Zet::drukAf ()
{
  cout << "(steen " << i << ", rotatie " << r << ", vakje ["
       << rij << "," << kolom << "])" << endl;

}  // drukAf
```

7.3 standaard

```
// Implementatie van standaard functies.

#include <iostream>
#include <cstdlib>  // voor rand
#include "standaard.h"
using namespace std;

//*********************************************************************

bool integerInBereik (const char *variabele, int waarde,
                      int minWaarde, int maxWaarde)
{
  if (waarde>=minWaarde && waarde<=maxWaarde)
    return true;
  else
  { cout << variabele << "=" << waarde << ", maar moet in [" << minWaarde
         << "," << maxWaarde << "] liggen." << endl;
    return false;
  }

}  // integerInBereik

//*********************************************************************

int randomGetal (int min, int max)
{ int bereik,
      r;
```

```
  bereik = max - min + 1;

  r = ((rand())%bereik) + min;
  return r;

}  // randomGetal
```

7.4 main

```
// Hoofdprogramma voor oplossing voor eerste programmeeropdracht Algoritmiek,
// voorjaar 2021: Aapje Omino.
//
// Biedt de gebruiker een menustructuur om
// * een instantie van Aapje Omino in te lezen, en daarmee het spel te spelen,
//   waarbij de gebruiker steeds
//   - (zo nodig) een steen uit de pot kan halen
//   - een zet kan uitvoeren (een steen op het bord leggen)
//   - kan vragen om de 'goede' zetten
//   - kan vragen om de score voor de speler die aan de beurt is, als beide
//     spelers vanaf dit moment optimaal verder spelen
// * experimenten te doen, waarbij een gretige strategie met een optimale
//   strategie vergeleken wordt.
//
// Yvo Hu s2962802

#include <iostream>
#include <fstream>
#include <string>
#include <ctime>  // voor clock() en clock_t
#include "standaard.h"
#include "zet.h"
#include "aapjeomino.h"
using namespace std;
const int MaxBestandsNaamLengte = 30; // maximale lengte van een bestandsnaam

//*********************************************************************

// Schrijf de zetten in vector zetten met een passende kop naar het scherm.
void schrijfZetten (string kop, vector<Zet> zetten)
{ size_t nrZetten;

  cout << endl;
  cout << kop << endl;

  nrZetten = zetten.size();
  if (nrZetten>0)
```

```
  { for (size_t i=0;i<nrZetten;i++)
    { cout << i << ": ";
      zetten[i].drukAf();
    }
  }
  else
    cout << "Helaas, 0 zetten." << endl;

} // schrijfZetten

//*********************************************************************

// Schrijf het menu op het scherm (afhankelijk van nrZetten),
// en vraag een keuze van de gebruiker.
// Retourneer: de keuze van de gebruiker
int keuzeUitMenu (size_t nrZetten)
{ int keuze;

  cout << endl;
  if (nrZetten==0)
    cout << "1. Een steen uit de pot halen" << endl;
  else
    cout << "1. Een zet uitvoeren" << endl;
  cout << "2. De 'goede' zetten bepalen" << endl;
  cout << "3. Optimale score bepalen" << endl;
  cout << "4. Stoppen met dit spel" << endl;
  cout << endl;
  cout << "Maak een keuze: ";
  cin >> keuze;

  return keuze;

} // keuzeUitMenu

//*********************************************************************

// Speel het spel op het bord van ao1.
// Hierbij krijgt de gebruiker herhaaldelijk de keuze om
// * (zo nodig) een steen uit de pot te halen
// * een zet uit te voeren (een steen op het bord leggen)
// * te vragen om de 'goede' zetten
// * te vragen om de score voor de speler die aan de beurt is, als beide
//   spelers vanaf dit moment optimaal verder spelen
//
// Voor elke iteratie van het menu wordt de stand afgedrukt, met de mogelijke
// zetten voor de huidige speler aan de beurt.
```

```
//
// Dit alles gaat door
// * totdat er een eindstand is bereikt (een van de spelers heeft geen
//   stenen meer en/of de huidige speler kan niets meer doen)
// * of totdat de gebruiker aangeeft dat hij wil stoppen met het spel
void doeSpel (AapjeOmino *ao1)
{
  int keuze,
      stnr,             // steen die uit pot gehaald wordt
      zetNr,            // nummer van de zet die gedaan moet worden
      score;            // returnwaarde van besteScore
  long long aantalStanden;  // aantal bekeken standen bij aanroep besteScore
  vector<Zet> zetten;
  size_t nrZetten;
  Zet besteZet;
  clock_t t1, t2;

  keuze = 0;
  while (keuze!=4 && !ao1->eindstand())
  {
    ao1 -> drukAf();
    zetten = ao1 -> bepaalMogelijkeZetten ();
    schrijfZetten ("Mogelijke zetten voor speler aan beurt zijn:", zetten);
    nrZetten = zetten.size();

    keuze = keuzeUitMenu (nrZetten);
    switch (keuze)
    { case 1: if (nrZetten==0)
              { stnr = ao1 -> haalSteenUitPot ();
                if (stnr!=-1)  // gelukt om een steen uit de pot te halen
                { zetten = ao1 -> bepaalMogelijkeZetten ();
                  if (zetten.size()==0)
                    ao1 -> wisselSpeler ();
                }
              }
              else
              { cout << endl;
                cout << "Geef het nummer van een mogelijke zet (0.."
                     << nrZetten-1 << "): ";
                // ook als je hiervoor net de 'goede' zetten hebt
                // opgevraagd, moet je kiezen uit alle mogelijke zetten
                cin >> zetNr;
                if (integerInBereik ("zetNr", zetNr, 0, nrZetten-1))
                  ao1->doeZet (zetten[zetNr]);
              }
              break;
```

```
      case 2: zetten = ao1 -> bepaalGoedeZetten ();
              schrijfZetten ("'Goede' zetten voor speler aan beurt zijn:",
                             zetten);
              break;
      case 3: t1 = clock ();
              score = ao1 -> besteScore (besteZet, aantalStanden);
              t2 = clock ();
              cout << endl;
              cout << "Optimale score = " << score << endl;
              cout << "Een beste zet is: ";
              besteZet.drukAf();
              cout << "We hebben hiervoor " << aantalStanden
                   << " standen bekeken." << endl;
              cout << "Dit kostte " << (t2-t1) << " clock ticks, ofwel "
               << (((double)(t2-t1))/CLOCKS_PER_SEC) << " seconden." << endl;
              break;
      case 4: break;
      default: cout << endl;
               cout << "Voer een goede keuze in!" << endl;
    }  // switch

  }  // while

  if (ao1->eindstand())
  { ao1 -> drukAf ();
    cout << endl;
    cout << "De huidige stand is een eindstand.\n";
  }

}  // doeSpel

//*****************************************************************

// Voert de experimenten uit zoals beschreven in de opdracht.
void doeExperimenten ()
{
  ofstream bestand("experiment.txt");
  bestand << "nr, besteScore, aantalStanden, tijd, scores2 " << endl;

  for(int i = 8; i <= 30; i++)
//Doorloop aantal beginstenen
  {
    int besteScores[10];
//Arrays for de waardes van de resultaten
    int standen[10];
    double tijd[10];
```

```
    int besteScores2[10];

    double besteScoresSom = 0;
//Variabelen om de waardes in op te slaan
    double standenSom = 0;
    double tijdSom = 0;
    double besteScores2Som = 0;

    for(int j = 0; j < 10; j++)
//Doorloop het programma 10 keer
    {
      long long aantalStanden = 0;
//Initialiseer aantalstanden
      Zet besteZet;
      clock_t t1, t2;
      AapjeOmino *ao1;
//Pointers naar een AapjeOmino object
      AapjeOmino *ao2;

      ao1 = new AapjeOmino ();
//Wijs naar een AapjeOmino object
      ao2 = new AapjeOmino ();

      ao1->genereerRandomSpel(7,7,i,i/4,3,3,1,i);
//Genereer spellen
      ao2->genereerRandomSpel(7,7,i,i/4,3,3,1,i);


      t1 = clock();
//Check de tijd die het duurt om het te bereken
      besteScores[j] = ao1->besteScore(besteZet, aantalStanden);
//Bereken besteScore
      t2 = clock();
      standen[j] = aantalStanden;
//Bereken aantalStanden

      besteScores2[j] = ao2->besteScore2(besteZet, aantalStanden);
//Bereken besteScore2

      tijd[j] = (((double)(t2-t1))/CLOCKS_PER_SEC);

      delete ao1;
      delete ao2;
    }

    for(int j = 0; j < 10; j++)
```

```
    {
      besteScoresSom += besteScores[j];
      standenSom     += standen[j];
      tijdSom        += tijd[j];
      besteScores2Som     += besteScores2[j];
    }
    besteScoresSom  = int((besteScoresSom/10)*100)/double(100);
    standenSom      = int((standenSom/10)*100)/double(100);
    tijdSom         = tijdSom;
    besteScores2Som      = int((besteScores2Som/10)*100)/double(100);
    bestand << i              << ", ";
    bestand << besteScoresSom << ", ";
    bestand << standenSom     << ", ";
    bestand << tijdSom        << ", ";
    bestand << besteScores2Som     << endl;

  }
  bestand.close();

}  // doeExperimenten

//*********************************************************************

void hoofdmenu ()
{ AapjeOmino *ao1;  // pointer, om makkeijk nieuwe objecten te kunnen maken
                    // en weer weg te gooien
  int keuze;
  char invoernaam[MaxBestandsNaamLengte+1];

  do
  {
    cout << endl;
    cout << "1. Een spel spelen" << endl;
    cout << "2. Experimenten doen" << endl;
    cout << "3. Stoppen" << endl;
    cout << endl;
    cout << "Maak een keuze: ";
    cin >> keuze;
    switch (keuze)
    { case 1: ao1 = new AapjeOmino ();
              cout << "Geef de naam van het tekstbestand met het spel: ";
              cin >> invoernaam;
              if (ao1 -> leesIn (invoernaam))
                doeSpel (ao1);
                // netjes opruimen
              break;
```

```
      case 2: doeExperimenten ();
              break;
      case 3: break;
      default: cout << endl;
               cout << "Voer een goede keuze in!" << endl;
    }

  } while (keuze!=3);

}  // hoofdmenu

//*************************************************************************

int main ()
{
  hoofdmenu ();

  return 0;

}
```

8 makefile

```
CC = g++

CompileParms = -c -Wall -std=c++11

OBJS = standaard.o zet.o aapjeomino.o main.o

Opdr: $(OBJS)
        $(CC) $(OBJS) -o AapjeOmino

standaard.o: standaard.cc standaard.h
        $(CC) $(CompileParms)  standaard.cc

zet.o: zet.cc standaard.h zet.h
        $(CC) $(CompileParms)  zet.cc

aapjeomino.o: aapjeomino.cc standaard.h zet.h aapjeomino.h
        $(CC) $(CompileParms)  aapjeomino.cc

main.o: main.cc standaard.h zet.h aapjeomino.h
        $(CC) $(CompileParms)  main.cc
```