De eerste programmeeropdracht — Brute Force
Algoritmiek voorjaar 2016, Universiteit Leiden

We bekijken een tweepersoonsspel dat wordt gespeeld op een 3 bij n (geheel, > 0) rechthoekig bord. Het bord heeft dus 3 rijen en n kolommen. Er zijn twee spelers, W (Wit) en Z (Zwart), die om de beurt een zet doen. De bedoeling is om een programma te schrijven dat, via het doorrekenen van alle mogelijkheden (*brute force*), berekent of het spel winnend is voor de beginnende speler en een winnende zet geeft. Tevens moet het spel tegen de computer gespeeld kunnen worden. De opdracht bestaat uit twee delen: een C++-programma en een verslag.

Het spel en de spelregels

Bij de start van het spel staan er n witte pionnen op de onderste rij en n zwarte pionnen op de bovenste rij. Wit speelt met de witte pionnen, Zwart met de zwarte pionnen.
Een *zet* met een witte pion is ofwel de pion 1 vakje naar voren verplaatsen (indien daar geen andere pion staat), ofwel 1 vakje diagonaal slaan (indien daar een zwarte pion staat; die wordt dus weggehaald als hij wordt geslagen). Voor een zwarte pion analoog, maar dan naar beneden verplaatsen/slaan. Belangrijk: *slaan is verplicht!* Het spel is afgelopen als de speler die aan de beurt is niet meer kan zetten. Hij/zij heeft dan gewonnen, ofwel: de speler die de laatste zet heeft gedaan verliest[1]. We spreken af dat Wit altijd begint.

Een voorbeeld van een mogelijk spelverloop, met $n = 7$:

```
Z Z Z Z Z Z Z    W     Z Z Z Z Z Z Z    Z     Z Z - Z Z Z Z    W
- - - - - - -   --->   - W - - - - -   --->   - Z - - - - -   --->
W W W W W W W          W - W W W W W          W - W W W W W
     (1)                    (2)                   (3)

Z Z - Z Z Z Z    Z     - Z - Z Z Z Z    W     - Z - Z Z Z Z    Z
- W - - - - -   --->   - Z - - - - -   --->   - W - - - - -   --->   ...
- - W W W W W          - - W W W W W          - - - W W W W
     (4)                    (5)                   (6)
```

In dit spelverloop is toestand (1) de begintoestand, dus W is aan de beurt. Hij kiest ervoor om de pion uit de tweede kolom te zetten. Vervolgens moet Z slaan, maar kan daarbij kiezen uit twee mogelijke slagzetten. Zij zet haar pion uit kolom 3. In toestand (3) moet W slaan; hij doet dat met zijn pion uit kolom 1. Vervolgens moet Z slaan, en daarna W. Deze twee zetten liggen volledig vast. Dit levert stand (6) op, waarbij Z aan de beurt is. Deze kan nu weer kiezen welke pion zij 1 positie vooruit verplaatst. De pionnen in de tweede kolom zijn beide geblokkeerd.

[1]Bij de meeste tweepersoonsspellen is de afspraak dat de speler die de laatste zet doet wint. Wanneer de regel is dat degene die de laatste zet doet juist verliest spreken we van misère-spel

Het C++-programma

Het programma moet voor zo veel mogelijk verschillende waarden van n kunnen bepalen of het spel winnend is voor de beginnende speler of niet. Tevens moet een winnende zet (indien die bestaat) worden berekend. Hiertoe moeten enige experimenten worden uitgevoerd. Verder moet het spel gespeeld kunnen worden tegen de computer. De gebruiker begint en speelt dus met de witte pionnen. Op `liacs.leidenuniv.nl/~graafjmde/ALGO/` staat een skeletprogramma dat de structuur van het te schrijven programma aangeeft.

- Het programma vraagt de gebruiker eerst om te kiezen of hij/zij het spel wil spelen of experimenten wil doen. Meer over de experimenten is verderop te vinden, onder het kopje "Verslag en experimenten".

- Er wordt eerst gevraagd om een waarde voor n in te voeren. Vervolgens wordt de beginstand getoond en kan de gebruiker herhaald kiezen om een zet te doen of te stoppen en een nieuw spel te beginnen. Voorlopig nemen we even aan dat n voldoende klein is (zie verderop voor het geval dat n groot is).

- Zetten van de gebruiker worden ingevoerd door het kolomnummer j te geven van de kolom waarin de te verplaatsen pion staat. Daarmee ligt een zet helemaal vast: als er een pion schuin voor staat wordt geslagen, zo niet dan wordt verticaal geschoven. Er moet worden gecontroleerd dat de zet correct is en dat de gebruiker inderdaad slaat als er geslagen kan worden.

- Telkens wanneer de gebruiker aan de beurt is wordt eerst bepaald of de stand winnend of verliezend is voor de aan de beurt zijnde speler (de gebruiker dus) en wat een mogelijke winnende zet is (indien de stand winnend is). Deze winnende zet moet op het beeldscherm worden afgedrukt. Als een stand niet winnend is voor de gebruiker, moet dit ook gemeld worden.

- Het programma moet dus voor *elke* stand kunnen bepalen of deze winnend is voor de speler die aan de beurt is, of juist verliezend. Merk op dat een stand winnend is voor degene die aan de beurt is dan en slechts dan als een van zijn directe vervolgstanden niet winnend is voor de tegenstander.

- Er dient hiertoe een (member)functie `winst` te worden geschreven die met behulp van BRUTE FORCE en RECURSIE zo'n winnende zet bepaalt. Dat betekent dat vanuit elke stand steeds alle mogelijke zetten een voor een worden gedaan en dat van elk van de vervolgstanden *recursief* wordt nagegaan of deze winnend of verliezend is voor de tegenstander. Hieruit kun je dan een conclusie trekken over de oorspronkelijke stand en de eventuele winnende zet. Je moet stoppen zodra je zeker weet dat een stand winnend is.

- Laat de functie `winst` de winnende zet opleveren in de vorm van het kolomnummer van de kolom waarin de te verzetten pion staat. De waarde -1 correspondeert dan met het geval dat er geen winnende zet bestaat. De functie `winst` moet behalve een winnende zet tevens berekenen hoeveel (tussen)standen zijn bekeken om die zet te vinden.

- De computer speelt random (functie `doerandomzet`). Om een random zet te bepalen kun je eerst alle mogelijke zetten aflopen en zo tellen hoeveel zetten er in een zekere spelsituatie mogelijk zijn. Vervolgens kies je daar random een uit, zeg de i-de mogelijke zet. Gebruik hiertoe de random-generator zoals bij Programmeermethoden. Deze zet voer je vervolgens uit, bijvoorbeeld door nogmaals de zetten af te lopen en de i-de te doen. Let erop dat slaan verplicht is. In het geval dat geslagen moet worden heb je hooguit twee mogelijke zetten; anders heb je er in het algemeen meer.

- Druk na elke zet, zowel van de gebruiker als van de computer, de resulterende spelsituatie af op het scherm. Geef bij een computerzet ook expliciet aan welke zet de computer doet.

- De functie `winst` rekent recursief alle mogelijke zetten door (brute force), en werkt daardoor alleen goed voor niet al te grote n (zeg $n \leq$ `groot`; de constante `groot` bijvoorbeeld 25). We gaan daarom ook voor elke stand (beginstand en tussenstanden) zonder alles door te rekenen (dus snel) de best *lijkende* zet bepalen. Deze zet hoeft geen winnende zet te zijn.

- Schrijf hiertoe een (member)functie `bestezet`, die de best lijkende zet oplevert voor degene die aan de beurt is (we noemen die hier even X). Zo'n zet berekenen we als volgt. Laat voor elke directe vervolgstand van de huidige stand een vast aantal keren (een constante `vaak`, bijvoorbeeld 30) een random spelverloop gespeeld worden (spelers zetten om de beurt random). Houd bij hoe vaak X zo'n spelverloop wint. Kies vervolgens de zet die leidt tot een/de vervolgstand met het hoogste aantal winstpartijen voor X.

- Als $n \leq$ `groot` moet het programma bij het spelen van het spel de functie `winst` gebruiken, anders de functie `bestezet`.

- Kies een verstandige klasse-structuur, met in elk geval een aparte klasse `Stand`, die onder andere de memberfuncties bevat zoals in het skeletprogramma aangegeven. Representeer een stand met behulp van een tweedimensionaal array `bord`, dat gevuld is met `'W'`, `'Z'` en `'-'`. Ook degene die aan de beurt is moet als membervariabele worden opgenomen. Denk verder aan de constructor en de destructor (indien nodig). Splits je programma op in ten minste drie files, zoals in het skeletprogramma.

- Functies mogen niet te lang zijn (maximaal 35 regels).

- Gebruik constantes waar dat zinvol is, bijvoorbeeld `Nmax` voor de maximale grootte van het bord en `groot` voor de maximale bordgrootte waarvoor de winnende zet nog snel genoeg te berekenen is. Bepaal zelf de waarde van `groot` door de functie `winst` op verschillende bordgroottes uit te proberen.

- Het werkende programma mag er op het scherm eenvoudig uitzien, maar moet wel duidelijk zijn. De enige te gebruiken headerfiles zijn in principe iostream, cstdlib en ctime (voor de random-generator).

- Je mag aannemen dat de gebruiker een geheel getal invoert als daarom gevraagd wordt (er mag dus worden ingelezen met `cin`), maar er moet wel worden gecontroleerd of de ingelezen waarde tussen de juiste grenzen ligt.

- Boven elke functie moet een commmentaarblokje komen met daarin een (zeer) korte beschrijving van wat de functie doet. Noem daarin tevens de gebruikte parameters: geef hun betekenis en geef aan hoe ze eventueel veranderd worden door de functie. Geef bij memberfuncties ook aan wat deze met de membervariabelen van het object doen. Let verder op de layout (consequent inspringen) en op het overige commentaar bij de programmacode (zinvol en kort).

- Het programma moet onder Linux bij LIACS getest zijn en werken.

Verslag en experimenten

Het **verslag** moet getypt zijn in LaTeX, en moet een introductie bevatten, de probleemstelling, een duidelijk antwoord op / uitwerking van onderstaande vier vragen en een hoofdstukje waarin je uitlegt hoe de recursieve functie werkt. Ten slotte bevat het verslag nog een klein hoofdstukje over de experimenten (zie verderop). Neem ook je programma op in het verslag.

Bij het tekenen van toestand-actie-ruimtes, zoals hieronder gevraagd, mag je rekening houden met de symmetrie van het probleem. Bijvoorbeeld: vanuit de beginstand (1) in het eerder gegeven voorbeeld zijn zeven zetten van W mogelijk. Echter uit symmetrie-overwegingen zijn het er maar vier: het verplaatsen van de pion uit kolom $1, 2, 3$ of 4.

- Te beantwoorden **vragen**:

1. Teken de (op symmetrie na) volledige toestand-actie-ruimte voor de gevallen $n = 2$, $n = 3$ en $n = 4$. Geef bij elke toestand aan of deze winnend is voor W of voor Z. Bepaal zo een winnende zet en een winnende strategie.
2. Teken de toestand-actie-ruimte voor de gevallen $n = 5$ en $n = 6$. Geef bij elke toestand aan of deze winnend is voor W of voor Z. Bepaal zo een winnende zet en een winnende strategie. Je hoeft hier een toestand waarvan je al gezien hebt of die winnend is voor W danwel voor Z, niet meer helemaal uit te werken. Zie de opmerkingen hieronder.
3. Beredeneer dat (i) er bij elke zet altijd maar hooguit één pion geslagen kan worden en (ii) een zet (zowel slaan als gewoon) altijd vanuit de onderste (voor W) of de bovenste (voor Z) rij gebeurt, maar nooit vanuit de middelste rij.
4. Bewijs de volgende bewering: als het spel voor $n = K$ verliezend is voor de beginnende speler, dan is het spel met $n = K + 2$ winnend voor de beginnende speler.

Opmerkingen bij vraag 2: - Een voorbeeld: toestand (6) is eigenlijk het geval $n = 4$, met Z aan de beurt. Je weet dus al uit vraag 1 of (6) winnend is voor degene die aan de beurt is of niet. - Als er een serie slagzetten gedaan moet worden met verschillende mogelijke volgordes, resulteert dat telkens in dezelfde stand (zie vraag 1). Daarom mag je bij vraag 2 al die mogelijke volgordes samenvatten met één toestandsovergang naar die resulterende stand.

- **Experimenten**:

1. Ga voor zoveel mogelijk waarden van n na of de beginstand winnend of verliezend is voor de beginnende speler of niet. Bepaal ook een winnende beginzet (-1 als die niet bestaat). Voor deze tabel kan n groter gekozen worden dan de maximale grootte `groot` die je bij het spelen gebruikt. Zet deze resultaten in een tabel. Zet daarin ook de tijd die het kost om de/een winnende zet te vinden (of te constateren dat die niet bestaat) en de aantallen bekeken tussenstanden.
2. Bepaal voor alle waarden van n waarvoor je de winnende zet nog kon bepalen ook de best lijkende zet en neem beide op in een tweede tabel.
3. Geef behalve de tabellen met resultaten enige uitleg over je experiment en conclusie(s).

Zie: `liacs.leidenuniv.nl/~graafjmde/ALGO/` voor eventuele aanvullingen of tips bij de programmeeropdracht. De behaalde cijfers komen t.z.t. op blackboard te staan.

Uiterste (!) inleverdatum: maandag 21 maart 2016, uiterlijk 12:00 uur.
Het programma en `Makefile` per e-mail sturen naar: `j.m.de.graaf@liacs.leidenuniv.nl`. Zorg dat het onderwerp van je mail begint met "[ALGO]", dat scheelt de docent een hoop gezoek. Het verslag (inclusief het programma) moet op papier worden ingeleverd in de daartoe bestemde doos met opschrift Algoritmiek in de postkamer van Informatica, kamer 156. Voor elke week te laat inleveren gaat er een punt van het cijfer af. Vermeld overal duidelijk de namen van de makers.

Normering: verslag 3; commentaar en layout 1; modulaire opbouw en OOP: 1; werking 5