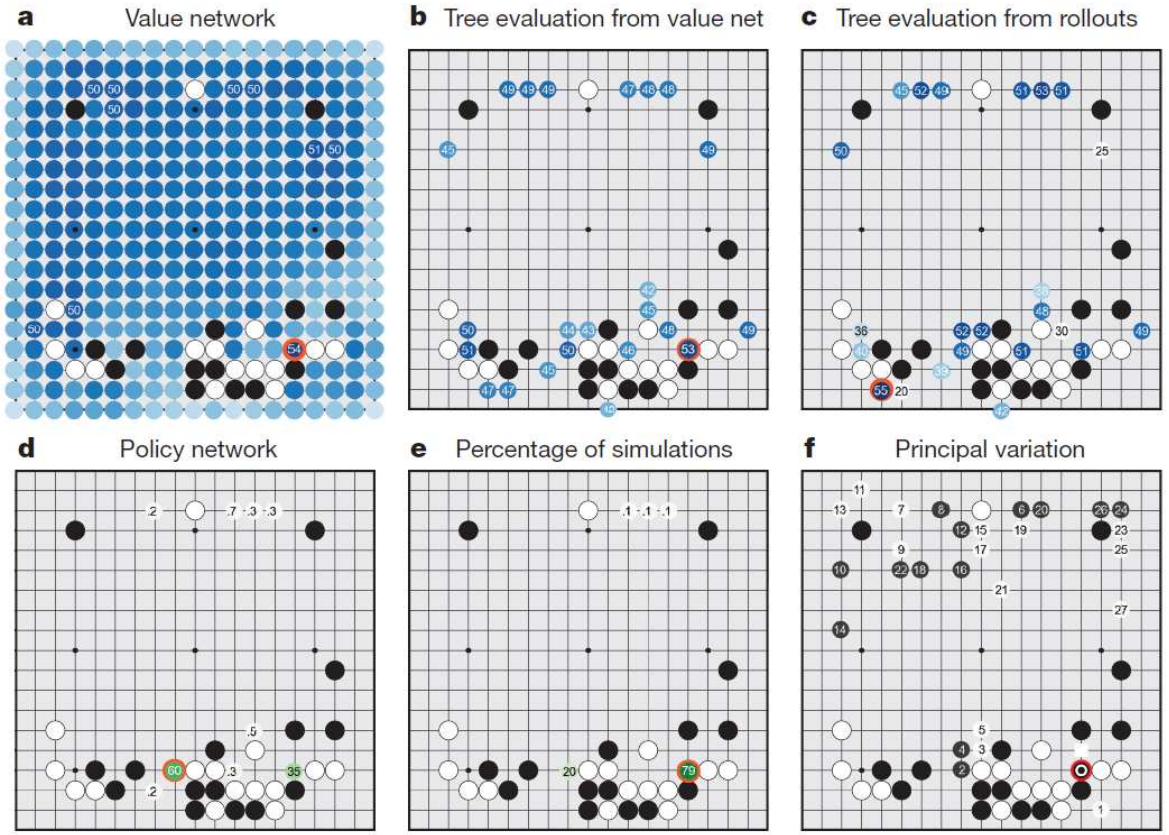
# Spellen: Van puzzels via tomografie naar backtracking

Universiteit Leiden

dr. Walter Kosters, Informatica

dinsdag 24 en 31 januari 2017; zaal 402 en 302

www.liacs.leidenuniv.nl/~kosterswa/




2016: computerprogramma verslaat Lee Sedol; 2017 . . .

Spellen en puzzels geven aanleiding tot complexe zoekproblemen, bijvoorbeeld bij schaken, go, vier-op-een-rij, sudoku, Japanse puzzels. Extra problemen zijn onzekerheden, kansen, . . .

We hebben een evaluatie-functie nodig om de niet-eindtoestanden te beoordelen, zoals bij schaken: pion 1, paard en loper 3, toren 5, koningin 9, veiligheid koning, . . .

We willen prunen: niet alles doorrekenen. We bekijken onder meer het minimax-algoritme van Von Neumann (1928) en het $\alpha$-$\beta$-algoritme uit 1956/58. Maar er zijn ook heel andere technieken . . .

# Spellen

Spellen kunnen als volgt worden ingedeeld:

| | deterministisch | kans |
|---|---|---|
| perfecte informatie | schaken, dammen, sudoku, checkers, go, othello | monopoly, backgammon |
| onvolledige informatie | zeeslag, mastermind | bridge, poker, scrabble |

Deterministisch: gevolgen van een zet liggen vast, geen kansen.
Perfecte informatie: spelers weten alles van de toestand.

Met name over schaken is veel gepubliceerd, waaronder allerlei anecdotes.
De ultieme uitdaging is/was het spel go.

Er zijn verschillende strategieën om (een benadering van) de "speltheoretische waarde" van een spel te bepalen. Shannon (1950) onderscheidt drie types:

**type A** reken alles tot en met zekere diepte door ("spelboom"), en gebruik daar een evaluatie-functie

**type B** reken soms verder door (als het onrustig is: "quiescence"); gebruik "heuristische" functie om dit te sturen

**type C** doelgericht menselijk zoeken

A en B zijn meer brute-force, C is meer "knowledge-based".

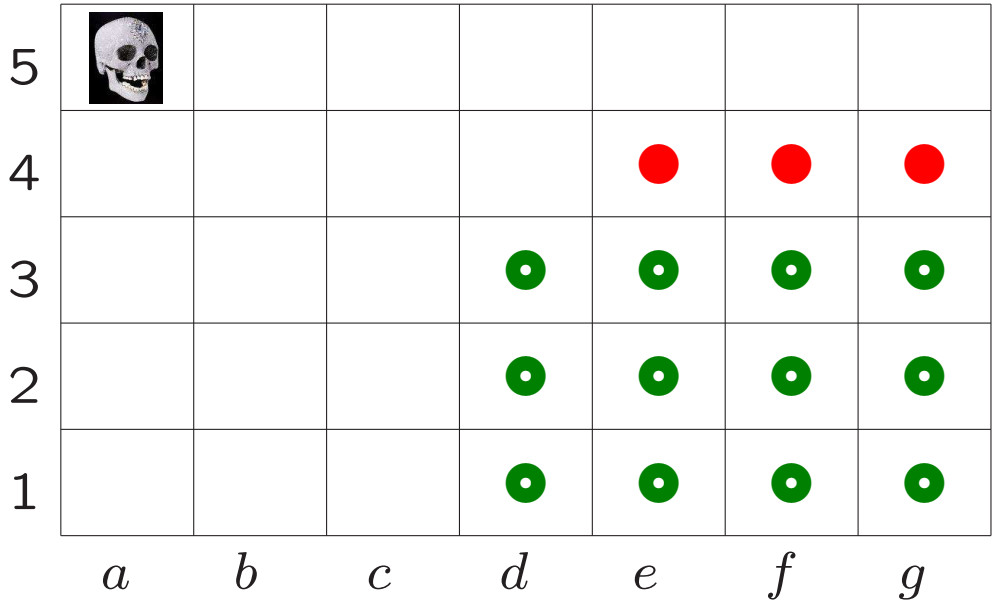
Er worden soms drie soorten oplossingen van spellen onderscheiden:

**ultra-zwak** de speltheoretische waarde van de beginstand is bekend: "je kunt vier-op-een-rij winnen"

**zwak** idem, en een optimale strategie is bekend (begin in middelste kolom, . . . , zie later)

**sterk** in elke legale positie is een optimale strategie bekend

Het spel Chomp wordt gespeeld met een rechthoekige reep chocola, waar de spelers om de beurt een stuk rechtsonder afhappen (een blokje en alles hier onder/rechts van). Wie het (vergiftigde) blokje linksboven eet, heeft verloren.




eerste hap: $d3$

tweede hap: $e4$

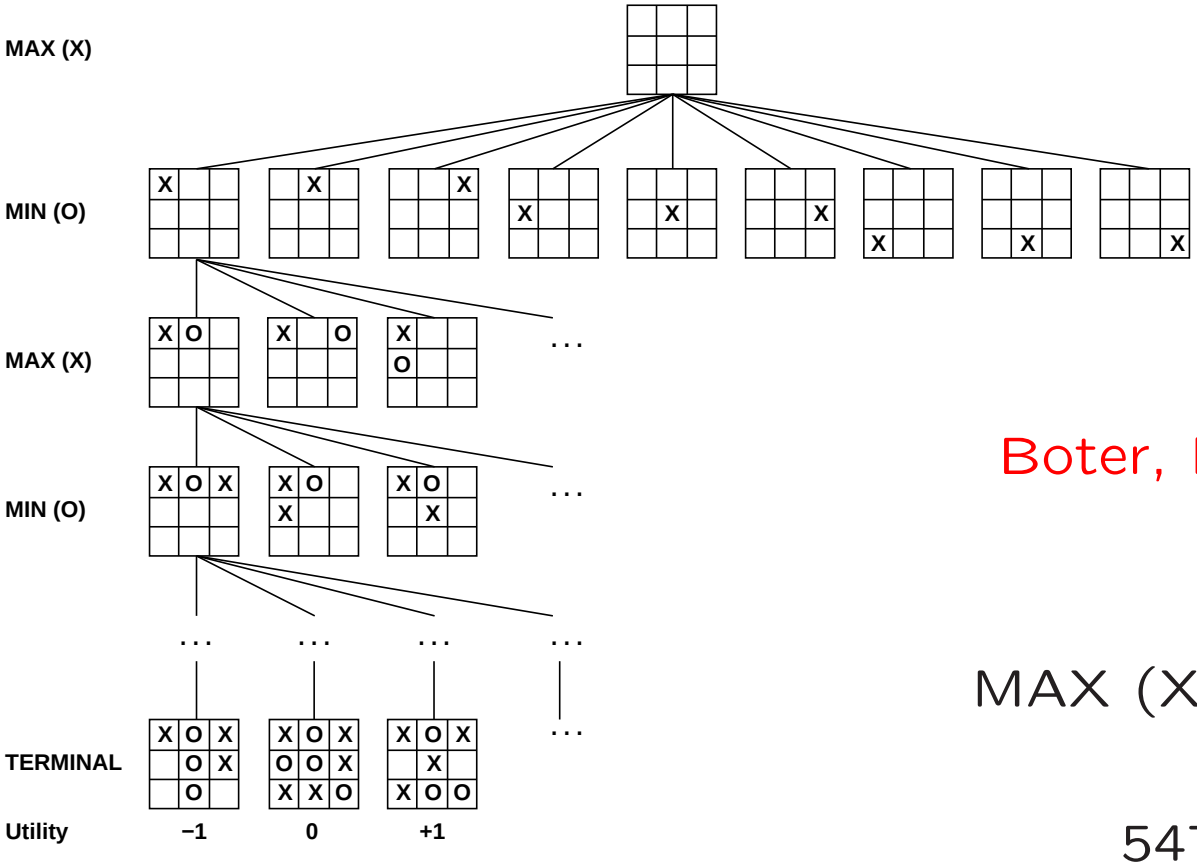
Bewering: de beginnende speler bij Chomp kan altijd winnen! Immers, als je door het blokje rechtsonder te nemen kunt winnen is het goed. Als dat niet zo is, heeft de tegenstander blijkbaar een voor hem winnende "tegen-hap". Die kun je dan zelf als eerste doen, en dus daarmee winnen! Dit argument heet strategy stealing.

Kortom: het spel Chomp is ultra-zwak opgelost, de echte winnende zet weten we niet . . .

Voor bijvoorbeeld $2 \times 2$ en $2 \times 3$ Chomp "wint" het blokje rechtsonder (algemener voor vierkanten: neem het vakje rechts onder het vergiftigde); bij $3 \times 4$ Chomp het blokje uit de middelste rij, derde kolom.

# Spellen — Informatica

# Voorbeeld spelboom




Boter, kaas en eieren

twee spelers:
MAX (X) en MIN (O)

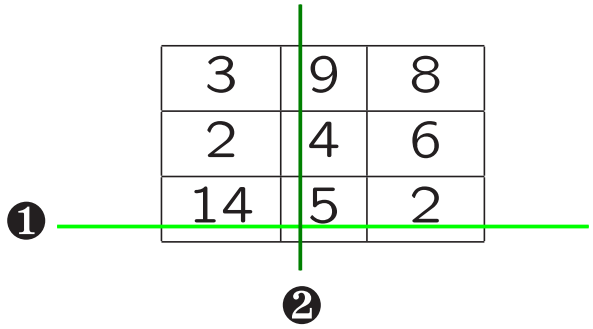
5478 toestanden

Boter, kaas en eieren is een voorbeeld van een tweepersoons deterministisch nulsom-spel met volledige informatie, waarbij de spelers om de beurt een "legale zet" doen. MAX moet een strategie vinden die tot een winnende eindtoestand leidt, ongeacht wat MIN doet. De strategie moet elke mogelijke zet van MIN correct beantwoorden.

Een utility-functie (= payoff-functie) geeft de waarde van eindtoestanden. Hier is dat: $-1/0/1$; bij backgammon is dat: $-192 \ldots +192$.

Uit symmetrie-overwegingen kunnen veel toestanden worden wegbezuinigd.

Maxi en Mini spelen het volgende eenvoudige spel: Maxi wijst eerst een horizontale rij aan, en daarna kiest Mini een (verticale) kolom:




Bijvoorbeeld: Maxi ❶ kiest rij 3, daarna kiest Mini ❷ kolom 2; dat levert einduitslag 5.

Maxi wil graag een zo groot mogelijk getal, Mini juist een zo klein mogelijk getal.

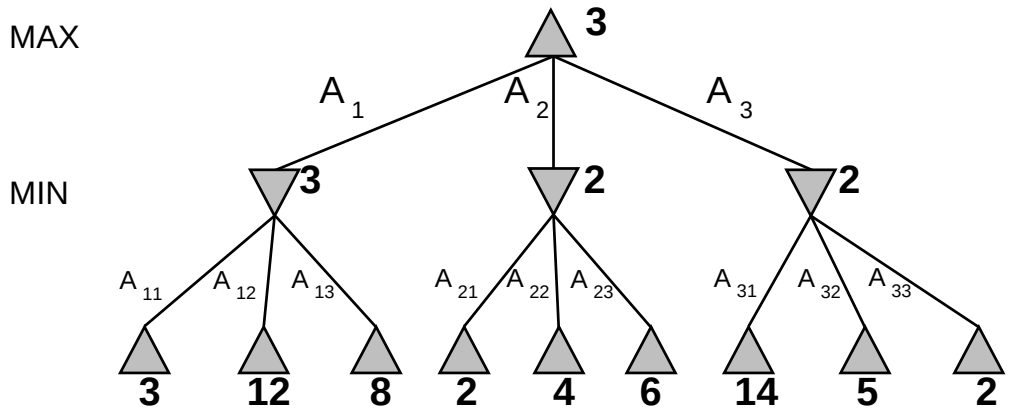
Hoe spelen we dit spel zo goed mogelijk?

Als Maxi rij 1 kiest, kiest Mini kolom 1 (levert 3); als Maxi rij 2 kiest, kiest Mini kolom 1 (levert 2); als Maxi rij 3 kiest, kiest Mini kolom 3 (levert 2). Dus kiest Maxi rij 1! Dit heet een brute force redenering: we hebben echt alles bekeken.

| | | |
|---|---|---|
| 3 | 12 | 8 |
| 2 | **?** | **?** |
| 14 | 5 | 2 |

Nu merken we op dat de analyse (het minimax-algoritme) hetzelfde verloopt als we niet eens weten wat onder de twee vraagtekens zit. Het $\alpha$-$\beta$-algoritme onthoudt als het ware de beste en slechtste mogelijkheden, en kijkt niet verder als dat toch nergens meer toe kan leiden.

In boomvorm:




Het minimax-algoritme is "recursief": neem in bladeren de evaluatie-functie, in MAX-knopen het maximum van de kinderen, in MIN-knopen het minimum van de kinderen. MAX- en MIN-knopen wisselen elkaar af.

Bovenstaande boom is één zet (= move) diep, oftewel twee ply.

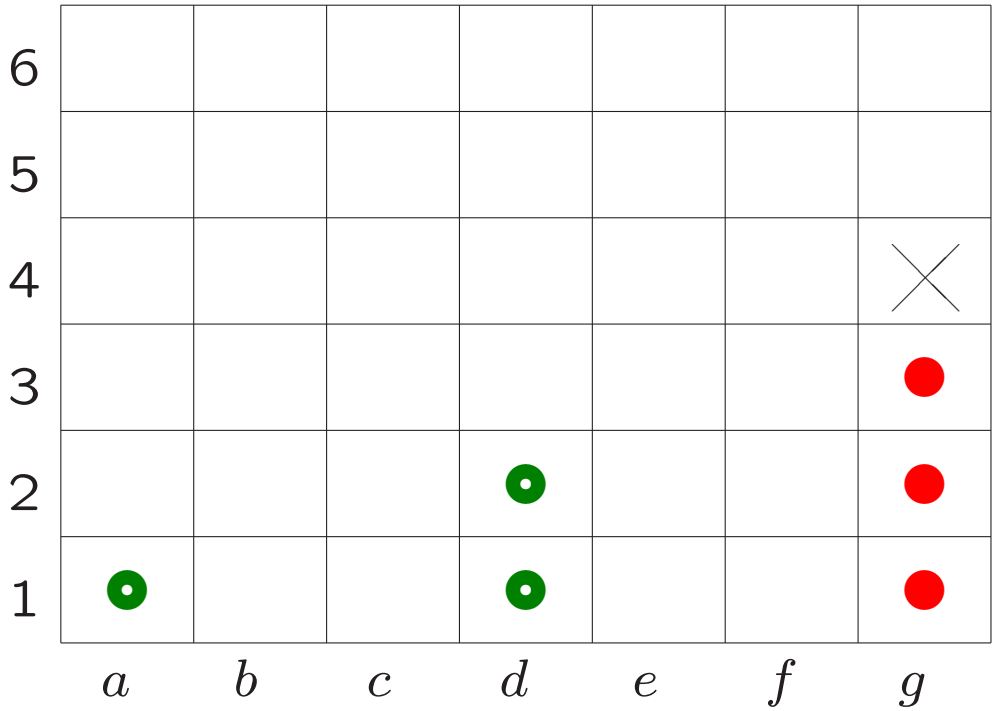
Het is van belang heuristieken (vuistregels) te hebben om je zetten te ordenen. Enkele voorbeelden:

**null-move** bekijk eerst voor de tegenstander goede zetten (sla je eigen zet in gedachten even over)

**killer** als een zet ergens een "snoeiing" teweeg brengt, doet hij dat elders wellicht ook

**conspiracy-number** $\approx$ aantal kinderen dat van waarde moet veranderen (samenzweren) om ouder van waarde te laten veranderen (voor stijgen MAX-knoop is maar één kind nodig, voor stijgen MIN-knoop zijn alle kinderen nodig)

**tabu-search** onthoud aantal (zeer) slechte zetten




Vier-op-een-rij

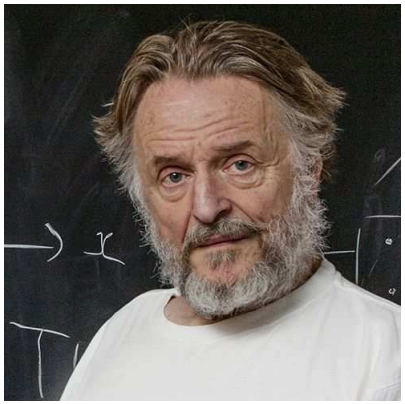
Met groen aan de beurt, is $g4$ zowel voor de null-move- als de killer-heuristiek de aangewezen zet.

De voor groen winnende (begin)serie: $d1! - d2 - d3! - d4 - d5! - b1 - b2$ (een ! betekent: unieke winnende zet).

# Wiskundige spellen

## Knuth en Conway

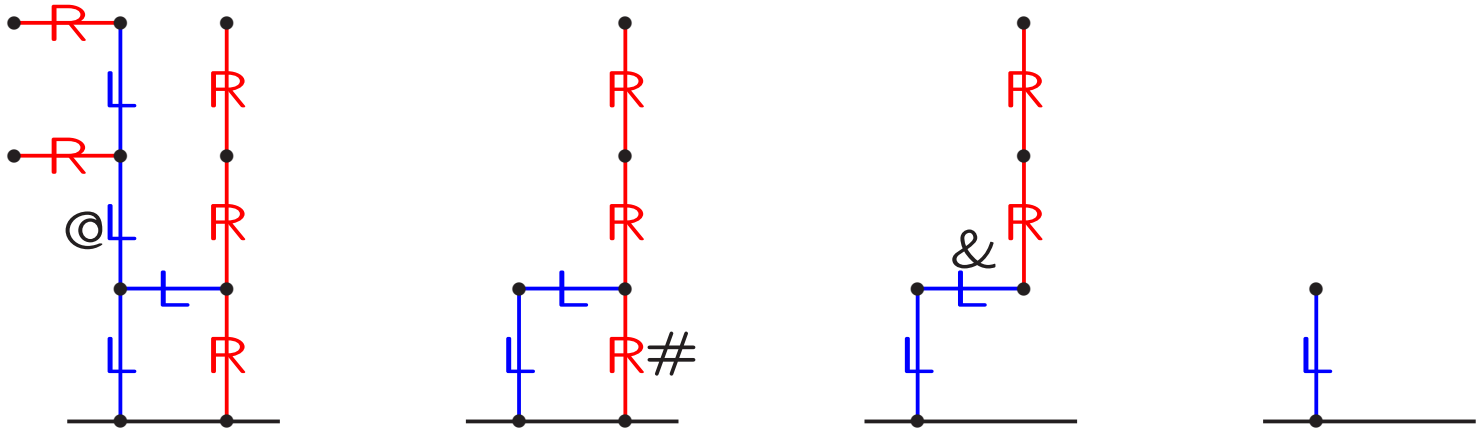
Donald E.(Ervin) Knuth
1938, US
NP; KMP
TEX
change-ringing; 3:16
The Art of Computer Programming



John H.(Horton) Conway
1937, UK $\rightarrow$ US
$Co_1$, $Co_2$, $Co_3$
Doomsday algoritme
game of Life; Angel problem
Winning Ways for your Mathematical Plays

**Surreal numbers**

Bij het spel Hackenbush verwijderen Links en Rechts om de beurt respectievelijk een bLauw of een Rood streepje, waarna alle streepjes die niet meer met de grond verbonden zijn ook worden verwijderd. *Wie niet kan, heeft verloren!*

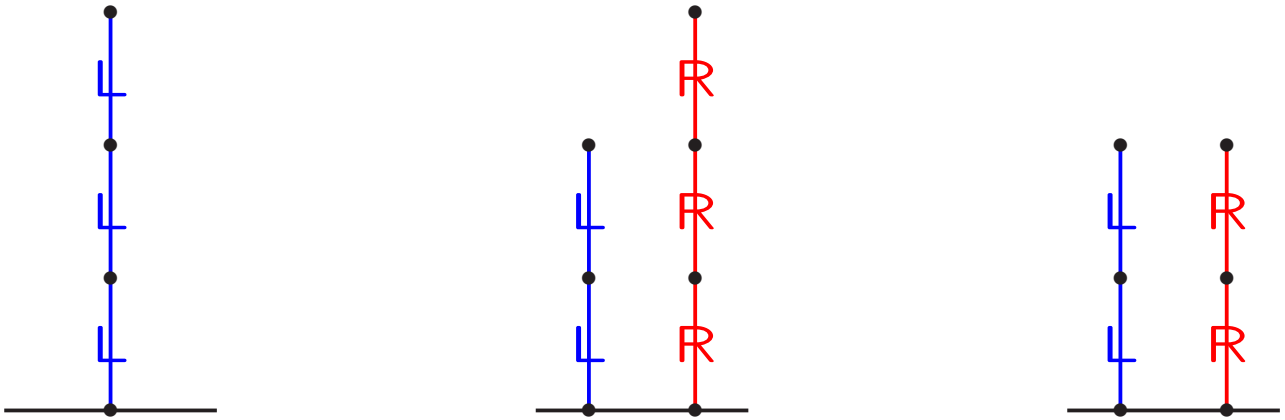



Links kiest @, Rechts kiest # (dom), Links kiest & en wint

Overigens: hier kan Rechts altijd winnen, wie er ook begint!

## Waarde van een positie

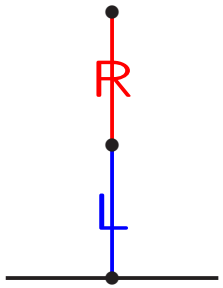
Wat is bij Hackenbush de waarde van een positie?




waarde 3 waarde $2-3=-1$ waarde $2-2=0$

Als de waarde positief ($>0$) is, *kan* Links altijd winnen (wie er ook begint; in het linker voorbeeld met voorsprong 3), als de waarde negatief ($<0$) is *kan* Rechts altijd winnen, en als de waarde 0 is verliest de beginspeler.

Maar wat is de waarde van deze positie?




Als Links begint, wint hij meteen; als Rechts begint, kan Links nog een keer, en wint hij ook. Dus Links wint altijd. De waarde is daarom $> 0$.

Vraag: is de waarde gelijk aan 1?

Als de waarde links 1 zou zijn, zou de waarde van de rechter positie $1 + (-1) = 0$ moeten zijn, en zou de beginspeler hier moeten verliezen. Is dat zo?


Nee: het is zo dat als Links begint, Links verliest, en als Rechts begint Rechts ook kan winnen. Dus Rechts wint altijd (= kan altijd winnen), en daarom is de rechter positie $< 0$, en de linker tussen 0 en 1.

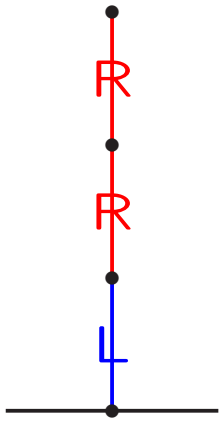
We noteren de waarde van de linker positie met $\{\, 0 \mid 1 \,\}$.

We merken op dat de rechter positie wél waarde 0 heeft: de beginspeler verliest. En dus geldt:

$$\{\, 0 \mid 1 \,\} + \{\, 0 \mid 1 \,\} + (-1) = 0,$$

en blijkbaar $\{\, 0 \mid 1 \,\} = 1/2$.

We noteren de waarde van een positie waarin Links kan spelen naar (waardes van) posities uit de verzameling $L$ en Rechts kan spelen naar (waardes van) posities uit de verzameling $R$ met $\{\ L \mid R\ \}$. Een voorbeeld:




De waarde is hier $\{\ 0 \mid \frac{1}{2}, 1\ \} = \frac{1}{4}$.

De waarde blijkt altijd het "eenvoudigste" getal dat tussen linker en rechter verzameling in zit.

Op deze manier definiëren we surreële getallen: het zijn "nette" paren van verzamelingen eerder gedefinieerde surreële getallen.

We beginnen met $0 = \{ \emptyset \mid \emptyset \} = \{ \text{NIKS} \mid \text{NIKS} \} = \{ \mid \}$: het spel waarbij de beginspeler geen enkele mogelijkheid heeft, en dus verliest.

En dan $1 = \{ 0 \mid \}$ en $-1 = \{ \mid 0 \}$.
En $42 = \{ 41 \mid \}$.
En $\frac{3}{8} = \{ \frac{1}{4} \mid \frac{1}{2} \}$.
En $\pi = \{ 3, 3\frac{1}{8}, 3\frac{9}{64}, \ldots \mid 4, 3\frac{1}{2}, 3\frac{1}{4}, 3\frac{3}{16}, 3\frac{5}{32}, \ldots \}$.

De reële getallen (de verzameling $\mathbf{R}$) zitten in de surreële getallen (de verzameling $\mathbf{S}$).

De zogeheten Dali-functie $\delta : \mathbf{R} \to \mathbf{S}$ verzorgt die "inbedding": $\delta(1) = \{\, 0 \mid \} = 1$. Maar er is meer ...

www.tondering.dk/download/sur16.pdf

We definiëren bijvoorbeeld:

$$\varepsilon = \{\, 0 \mid \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots \},$$

een "ongelooflijk klein positief getal", en

$$\omega = \{\, 0, 1, 2, 3, \ldots \mid \} = \{\, \mathbf{Z} \mid \emptyset \,\},$$

een "verschrikkelijk groot getal, een soort $\infty$".

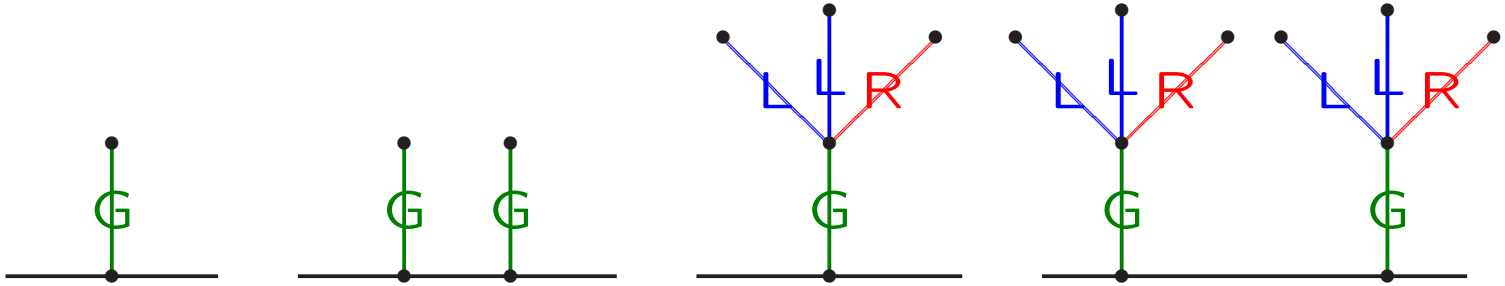
Dan blijkt te gelden:

$$\varepsilon \cdot \omega = 1$$

— mits je vermenigvuldiging goed gedefinieerd is . . .

En dan heb je ook $\omega + 1$, $\sqrt{\omega}$, $\omega^\omega$, $\varepsilon/2$, enzovoorts!

Bij Hutspot-Hackenbush zijn er ook nog Groene streepjes, die door *beide* spelers mogen worden weggepakt.

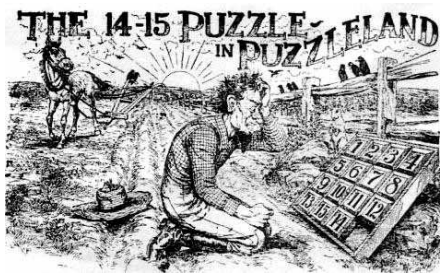


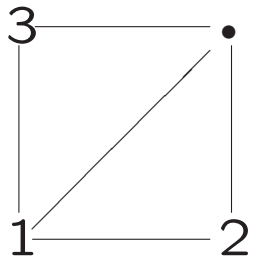

De meest linker positie heeft waarde $* = \{ 0 \mid 0 \}$ (dat blijkt *geen* surreëel getal te zijn), want de beginspeler kan winnen door het Groene streepje te pakken. De tweede van links is $* + * = 0$ (beginner verliest).

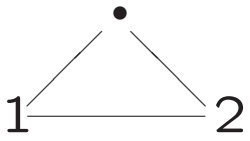
De tweede van rechts is weer gewonnen voor de beginspeler. De meest rechter positie is gewonnen voor Links (wie er ook begint), en is dus $> 0$.

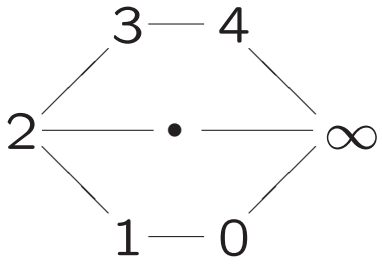
Bij een schuifpuzzel heb je een graaf waarbij alle knopen behalve één een uniek nummer hebben. Je mag een getal naar zijn buur schuiven als die buur geen nummer heeft. De vraag is: welke posities kun je vanuit een vaste bereiken?

Voorbeelden: de 15-puzzel, en onderstaande grafen. De meest rechtse is Wilson's Tricky Six Puzzle.

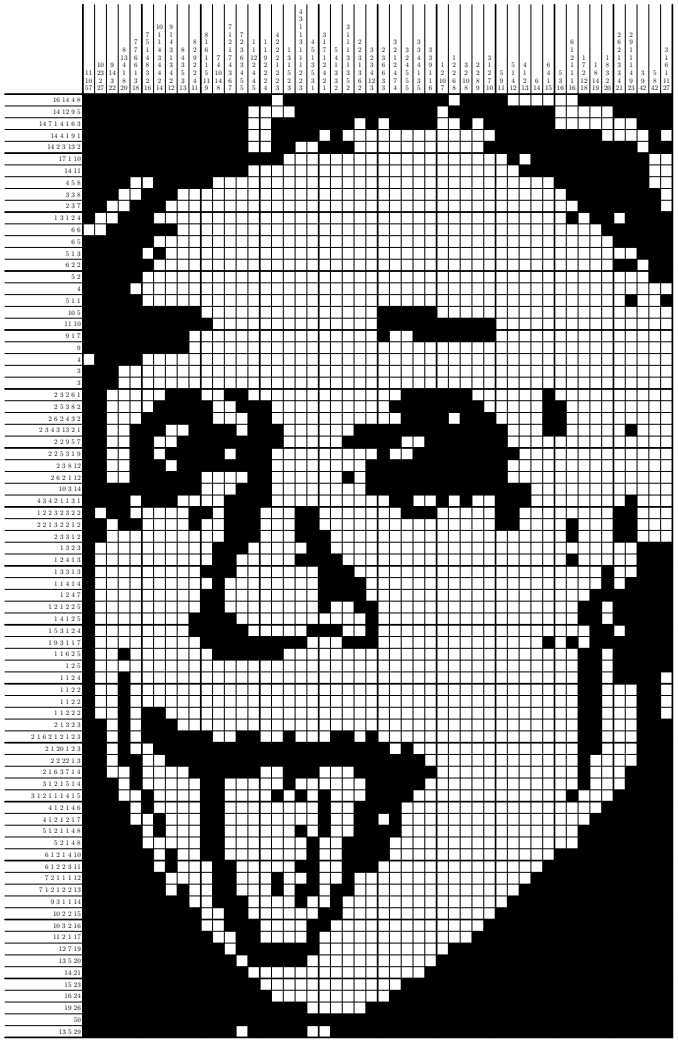


# Tomografie



Als je Japanse puzzels zegt, denkt iedereen aan Sudoku.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

Kort geleden is bewezen: “16 is genoeg” (Gary McGuire).

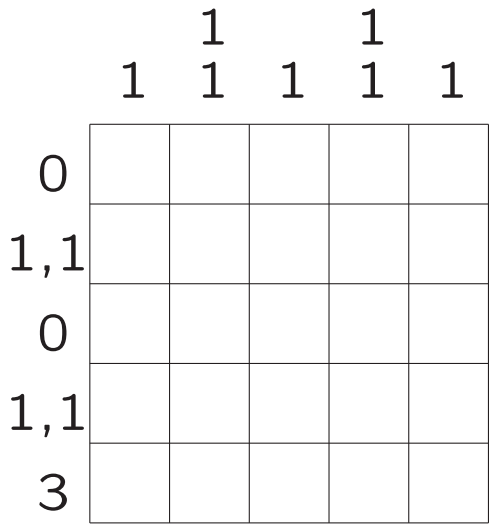
Als je Japanse puzzels zegt, denkt iedereen aan Sudoku.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 4 | 6 | 7 | 8 | 9 | 1 | 2 |
| 6 | 7 | 2 | 1 | 9 | 5 | 3 | 4 | 8 |
| 1 | 9 | 8 | 3 | 4 | 2 | 5 | 6 | 7 |
| 8 | 5 | 9 | 7 | 6 | 1 | 4 | 2 | 3 |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | 9 | 1 |
| 7 | 1 | 3 | 9 | 2 | 4 | 8 | 5 | 6 |
| 9 | 6 | 1 | 5 | 3 | 7 | 2 | 8 | 4 |
| 2 | 8 | 7 | 4 | 1 | 9 | 6 | 3 | 5 |
| 3 | 4 | 5 | 2 | 8 | 6 | 1 | 7 | 9 |

bron: Wikipedia

Maar wij gaan het hebben over Nonogrammen.

Een Nonogram is een puzzel; een klein voorbeeld:

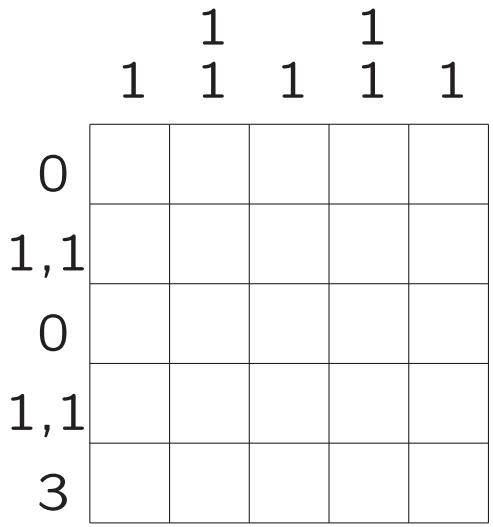


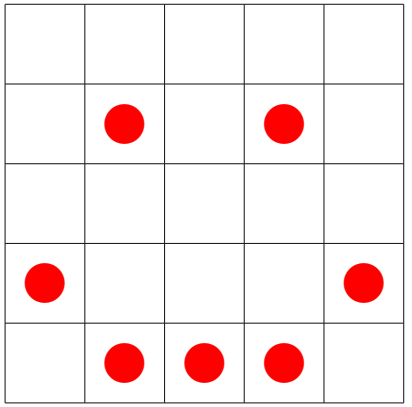

Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode (of zwarte) vakjes.

Waar moeten die rode vakjes komen?

## Voorbeeld — oplossing

De oplossing ziet er zo uit:



Naast iedere rij en boven iedere kolom staan in volgorde de lengtes van aaneengesloten series rode (of zwarte) vakjes.

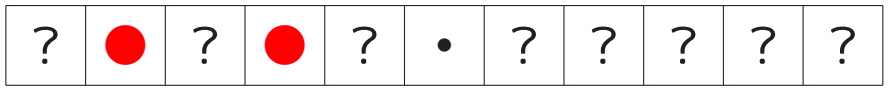
Hoe los je Nonogrammen op?

De meeste mensen gebruiken logische regels, en heuristieken = vuistregels zoals "redeneer eerst een keer via de rijen, en dan via de kolommen".

Een voorbeeld van een logische regel is: "als het getal 3 naast een rij van breedte 5 staat, moet het middelste vakje wel rood zijn". Je kijkt dan eigenlijk naar één rij of kolom.

Een heuristiek uit het dagelijks leven: als je een bedrag (zeg 80 cent) moet betalen, kun je het beste de grootste kleinere munt (50 cent?) die je hebt eerst geven.

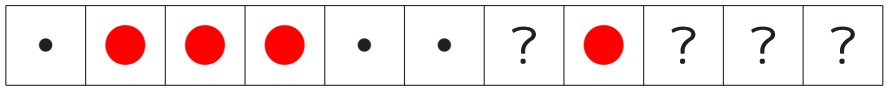
Stel dat je van een rij al weet:

3,2,1 


Een • betekent een zeker leeg vakje, een ● staat voor een zeker gevuld vakje. De rest is nog onbekend.

Wat kun je hier nu concluderen?

We concluderen dan dat voor deze rij geldt:

3,2,1 


Een • betekent een zeker leeg vakje, een ● staat voor een zeker gevuld vakje. De rest blijft nog onbekend.

Dus door naar een enkele rij of kolom te kijken kun je vooruitgang boeken. En dat gaat goed met dynamisch programmeren.

## Voorbeeld — oplossen

Hoe ver komen we als je alleen per rij/kolom kijkt? Een • betekent weer een zeker leeg vakje, een ● staat voor een zeker gevuld vakje.

| | 1 | 1 1 | 1 | 1 1 | 1 |
|---|---|---|---|---|---|
| 0 | | | | | |
| 1,1 | | | | | |
| 0 | | | | | |
| 1,1 | | | | | |
| 3 | | | | | |

| | | | | |
|---|---|---|---|---|
| • | • | • | • | • |
| • | ● | • | ● | • |
| • | • | • | • | • |
| ? | ? | • | ? | ? |
| ? | ? | ● | ? | ? |

Maar nu zitten we vast . . . tenzij we rijen en kolommen *samen* bekijken.

## Voorbeeld — verder oplossen

Dit hadden we:

| | | 1 1 | | 1 1 | |
|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 |
| 0 | | | | | |
| 1,1 | | | | | |
| 0 | | | | | |
| 1,1 | | | | | |
| 3 | | | | | |

| | | | | |
|---|---|---|---|---|
| • | • | • | • | • |
| • | ● | • | ● | • |
| • | • | • | • | • |
| $v$ | $w$ | • | ? | ? |
| $u$ | $x$ | ● | ? | ? |

Stel dat $u =$ ●, dan (kolom) moet $v$ leeg zijn, en dus (rij) $w =$ ●, en dus (kolom) moet $x$ leeg zijn. Tegenspraak (rij)! Dus $u$ moet leeg zijn.

Dat was een lastige logische redenering, ook voor een computer. Maar de rest is nu eenvoudig.

Soms heeft een probleem meer mogelijke (goede en foute) oplossingen dan je kunt doorrekenen. Zo heeft een $5 \times 5$ Nonogram al

$$2^{25} = 2^{10} \cdot 2^{10} \cdot 2^5 = 1024 \cdot 1024 \cdot 32 \approx 32 \text{ miljoen}$$

mogelijke invullingen! Er zijn immers $5 \times 5 = 25$ vakjes die elk 2 mogelijkheden hebben.

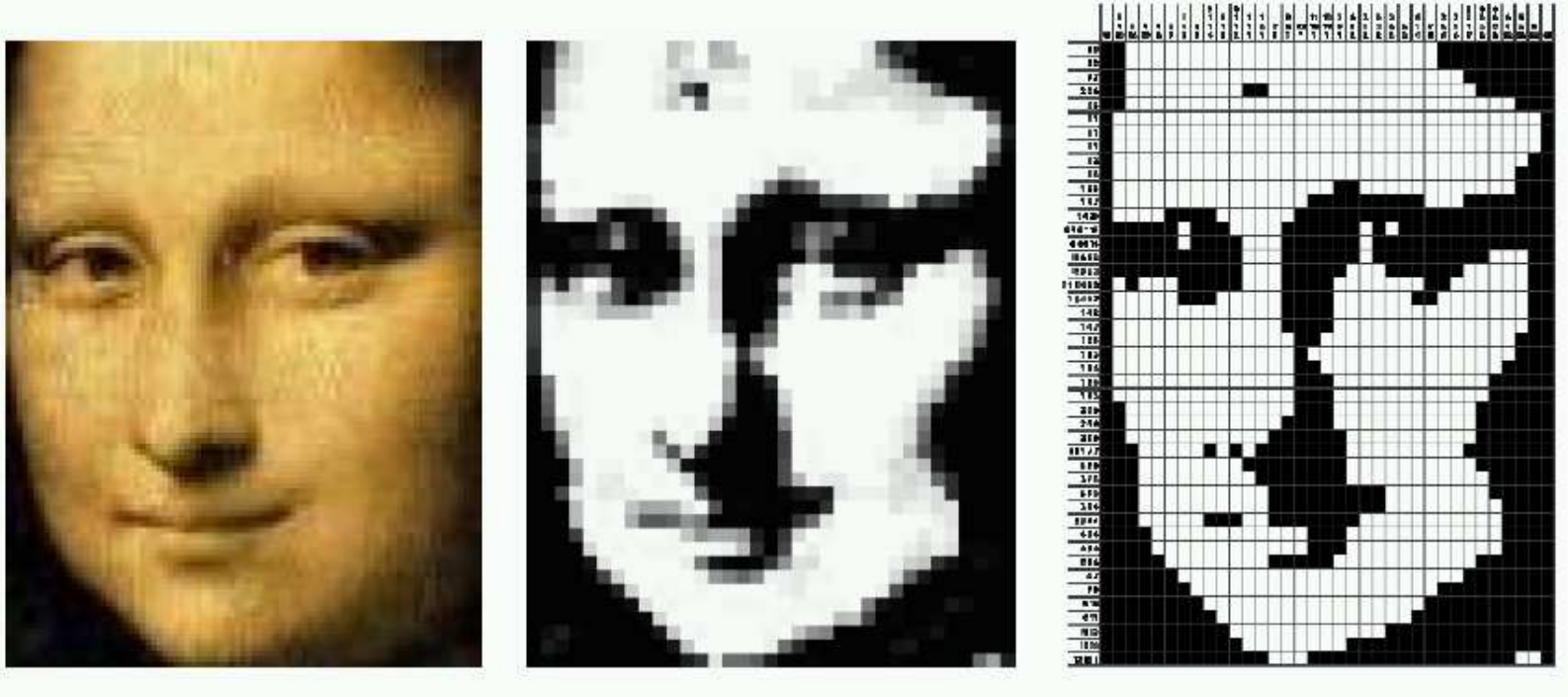
De "80 × 50 Einstein" heeft $2^{4000} \approx 10^{1200}$ mogelijkheden.

Dus brute-force, alles domweg proberen, lost een complete puzzel niet *snel* op . . .

Het oplossen van een Nonogram is een NP-volledig probleem, net als het maken van een schoolrooster. Dat kan eigenlijk alleen goed met brute-force, maar dan duurt het veel te lang. Van een mogelijke oplossing kun je wel snel zien of hij goed of fout is.

Het grootste open probleem in de informatica $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ gaat hierover. Je kunt een miljoen dollar verdienen als je dit oplost: de Clay Prize.
Het gaat daar niet om het oplossen van één puzzel of maken van één schoolrooster, maar om het vinden van een efficiënte algemene methode — als die al bestaat.

Hoe maak = construeer = ontwerp je zelf een Nonogram?

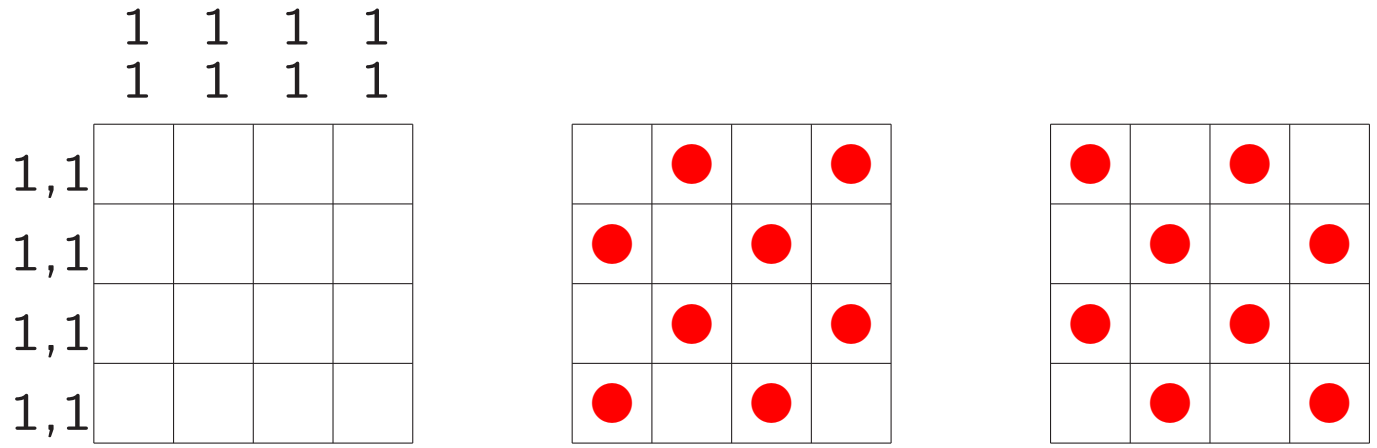


kleurenfoto grijswaarden-plaatje puzzel

www.liacs.leidenuniv.nl/~kosterswa/nono/

Denk eraan dat een *goed* Nonogram een unieke oplossing moet hebben.

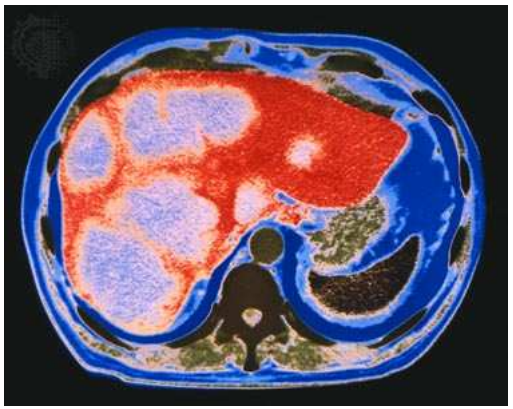
Ze hebben in het algemeen namelijk soms heel verschillende oplossingen:



Waarom doen wetenschappers Nonogrammen?

Tomografie houdt zich bezig met het volgende probleem:
Hoe reconstrueer je een object uit projecties?

Voorbeelden:



- Nonogrammen oplossen
- Hoe zien onze organen eruit, gegeven CT-scans?
- Waar zitten de “gaten” in een diamant?

## Evolutionaire algoritmen — algemeen

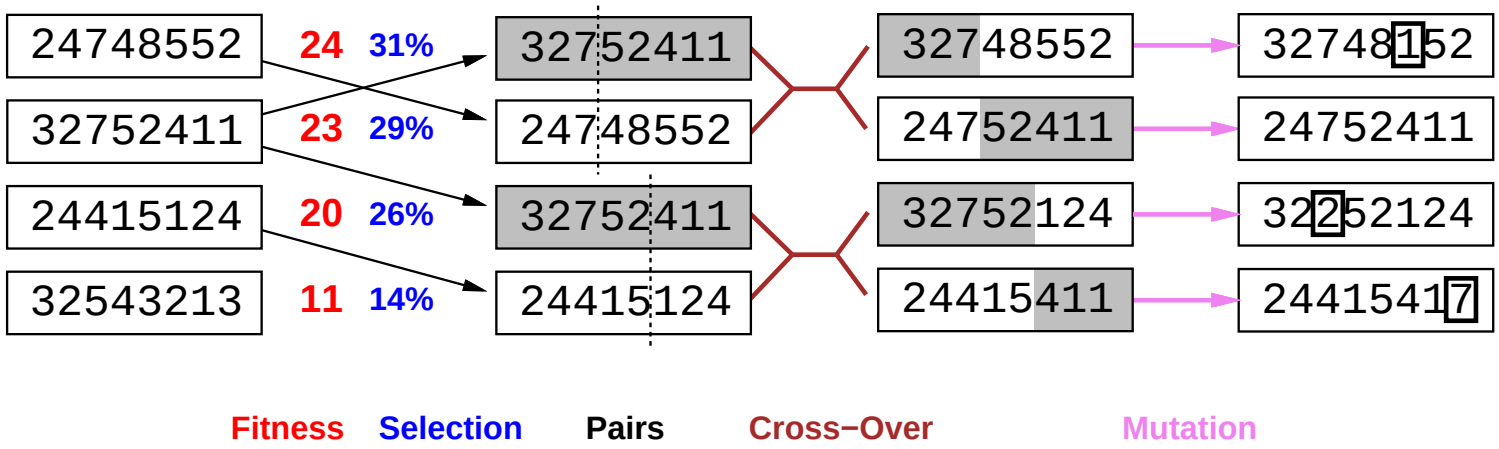
Hoe werkt in het algemeen een evolutionair (genetisch) algoritme?

We hebben steeds een stel kandidaat-oplossingen, individuen, die samen de populatie vormen. Je begint met een aantal willekeurige exemplaren.

Nu ga je die veranderen: muteren op willekeurige plaatsen, en combineren via crossover. Je bewaart steeds de beste individuen; om dat te beoordelen heb je een fitness-functie nodig: hoe hoger die is, des te beter is de oplossing.

Dit alles doe je net zo lang tot je er genoeg van hebt. En het op dat moment beste individu is je "oplossing".

Een voorbeeld-stap is:




Hier hebben we geen bits (0/1), maar getallen in de individuen. Je zou dit kunnen zien als genen. De string heet soms wel het genotype, dat wat hij voorstelt het phenotype.

## Oplossen van Nonogrammen

Evolutionaire (genetische) algoritmen kunnen gebruikt worden om Nonogrammen op te lossen.

Een individu, een kandidaat-oplossing, is een string met 25 bits (0000001010000001000101110; algemener, voor een $m$ bij $n$ puzzel, met $mn$ bits), waarbij een 1 rood en een 0 leeg voorstelt. Je kunt er voor kiezen om het aantal enen per rij altijd "goed" te houden.

Mutatie zou bijvoorbeeld in een rij een 1 en een 0 kunnen verwisselen. Als je handig muteert kun je "alles" bereiken!

De fitness-functie is een som over rijen en kolommen. Per rij/kolom geef je aan hoeveel je van de specificatie afwijkt — en dat is nog lastig precies te maken.

Ook aan een spel als Tetris kleven allerlei vragen:

- Hoe speel je het zo goed mogelijk? (AI)
- Hoe moeilijk is het?
- Wat kan er allemaal gebeuren?

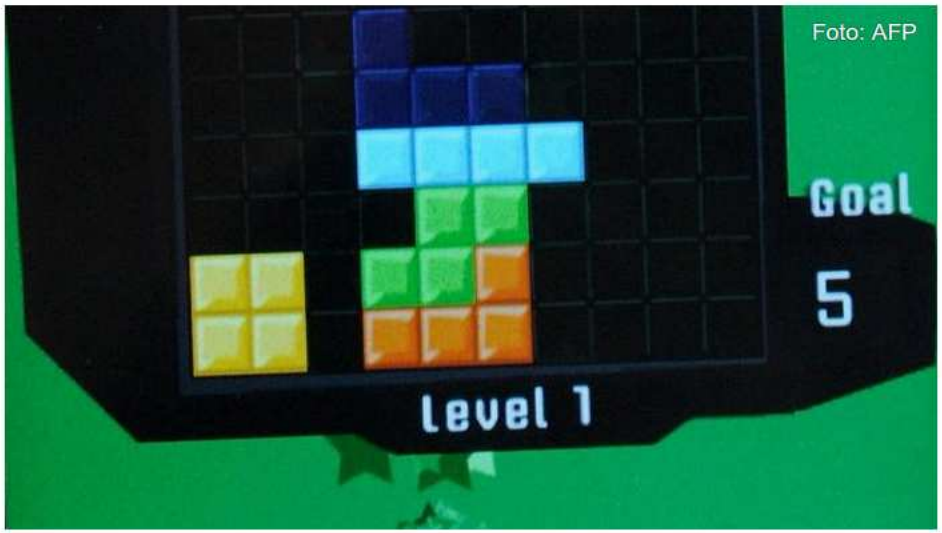
Zo is bijvoorbeeld bewezen dat sommige Tetris-problemen NP-volledig zijn, en dat je bijna alle configuraties kunt bereiken, zie:

www.liacs.leidenuniv.nl/~kosterswa/tetris/

Maandag 23 januari 2017 | Het laatste nieuws het eerst op NU.nl

NU.nl > Lifestyle > Werk en Privé

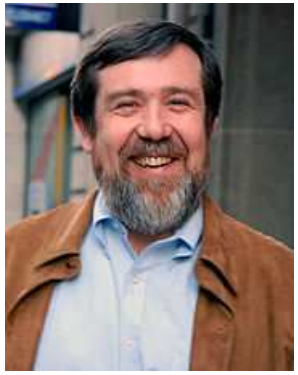
Voorpagina
Net binnen
Algemeen
Economie
Zakelijk
Sport
Tech
Entertainment
Lifestyle
Gezondheid
Eten & Drinken
Werk & Privé
Wonen
Reizen
Overig
Video's
Regionaal




'Tetris spelen helpt tegen verslavingen'

Gepubliceerd: 15 augustus 2015 18:17
Laatste update: 17 augustus 2015 09:05

Tetris is op 6 juni 1984 in Moskou gecreëerd door de Rus Alexey Pajitnov.



Het werd een groot succes, vooral dankzij de Game Boy.

Omstreeks 1989 was er juridisch getouwtrek, waarbij de Russische staat een belangrijke rol speelde.

Er zijn talloze varianten, zoals Tetris 4D, 1D, Blockout:

http://en.wikipedia.org/wiki/List_of_Tetris_variants

Maar er zijn ook subtiele varianten in de spelregels:

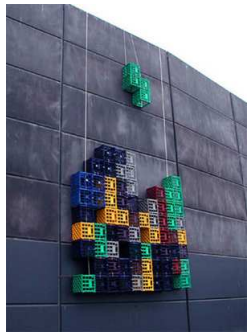
- zwaartekracht: zwevende tegels?
- rotatie: past het?

Je kunt wonen in Tetris:

Ljubljana

## Tetris is Life!

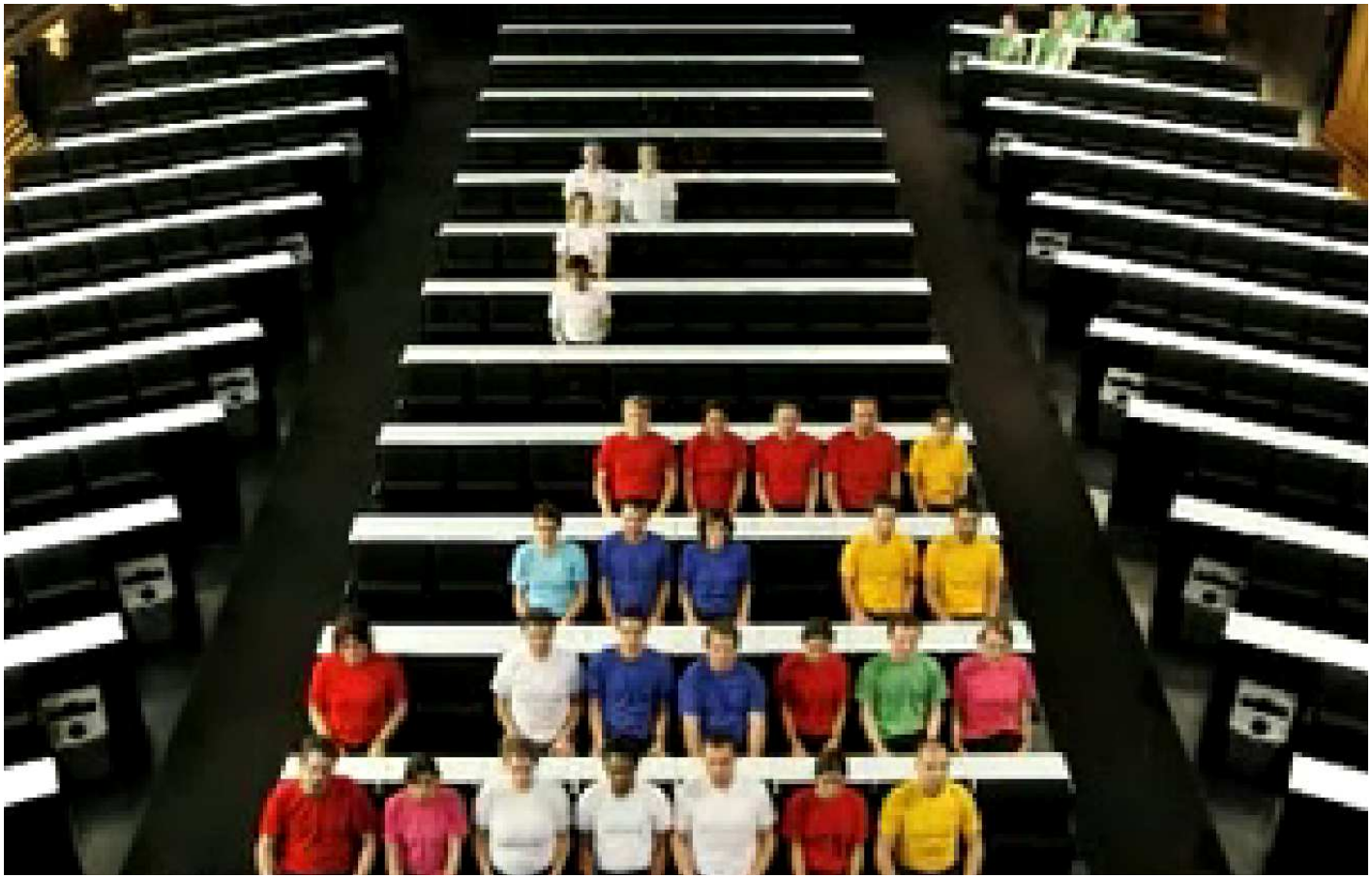
En je kunt je leven verder inrichten met Tetris:






Zie www.google.com/

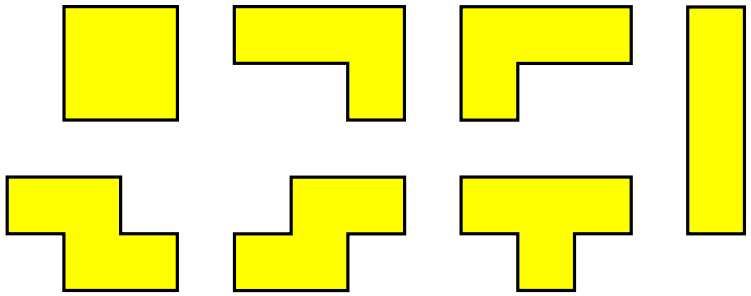
Tijd voor Tetris op de film.



YouTube

## Tetris: NP-volledig?

De 7 Tetris-stukken:

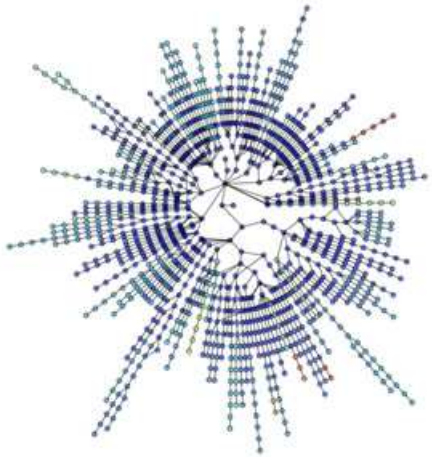


Stukken vallen random; volle regels worden verwijderd.

De vraag "Kun je met een gegeven serie (inclusief volgorde) van deze stukken een bord helemaal leeg spelen?" is heel erg moeilijk te beantwoorden. Een brute-force berekening lost het op . . . maar duurt erg lang.

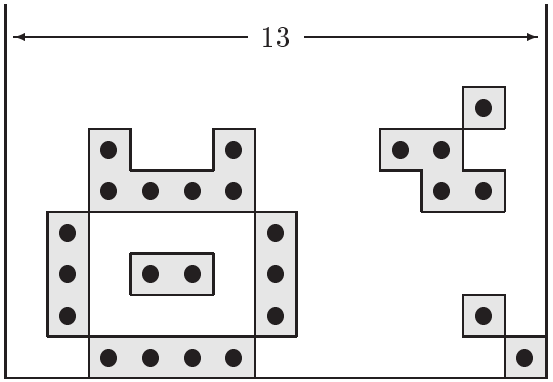
De vraag is NP-volledig, net als de vraag of een graaf een Hamilton-circuit heeft ("kun je rondwandelen?"), en net als de vraag of een Nonogram (uniek) oplosbaar is.

- (Sorteer een rij getallen met Shellsort. —ja)
- (Druk $n$ enen af. —nee: $n = e^{\log n}$)
- Is een gegeven graaf samenhangend? —ja
- Is $p$ een priemgetal? —ja (2002)
- Heeft een graaf een Hamilton-circuit? —???
- Is een kaart te "kleuren" met 2/3/4 kleuren —ja/???/JA



## Tetris: bereikbaar?

Een "willekeurige" configuratie:




Deze kan gemaakt worden door 276 *geschikte* Tetris-stukken op de juiste plaats te laten vallen.
Let op: alleen geheel gevulde regels verdwijnen, alles daarboven zakt *één rij*.

Claim: op een bord van oneven breedte kan elke configuratie bereikt worden!

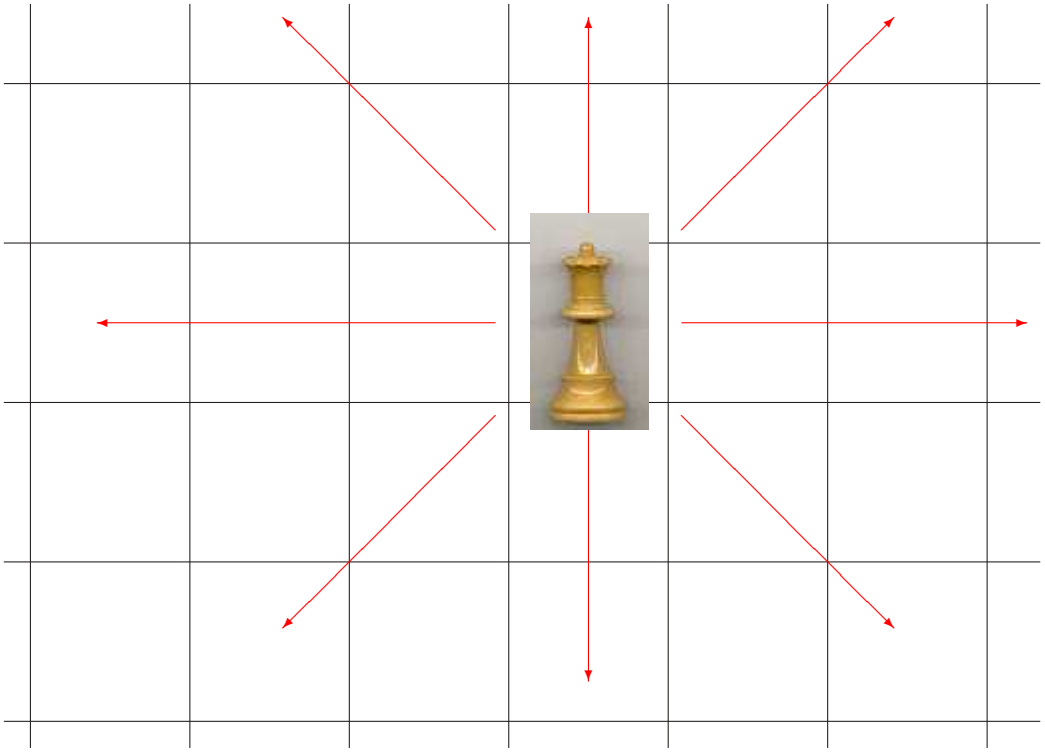
# Backtracking

Backtracking is een techniek die je helpt om uit doolhoven te komen — en soortgelijke problemen op te lossen.

Er zijn veel algoritmen (rekenvoorschriften, stappenplannen) die met backtracking werken, onder meer in de robotica, en ook bij het programmeren van spellen.
Het basisidee is: als je bij het zoeken vastloopt, ga je terug op je pad om het laatste "open" alternatief te proberen.

## Probleem

We willen nu het volgende probleem oplossen: zet 8 dames op een 8×8 schaakbord, zodanig dat geen dame een andere in één keer kan slaan.

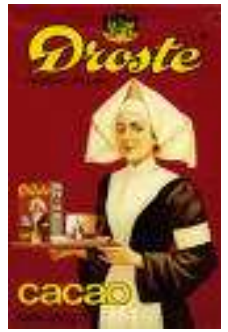


Het probleem kan systematisch met behulp van het volgende idee aangepakt worden:

als er nog een lege rij is

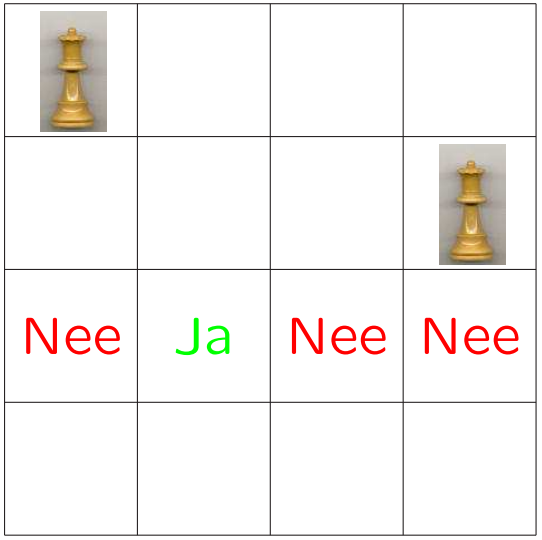
probeer in elke kolom een dame te zetten

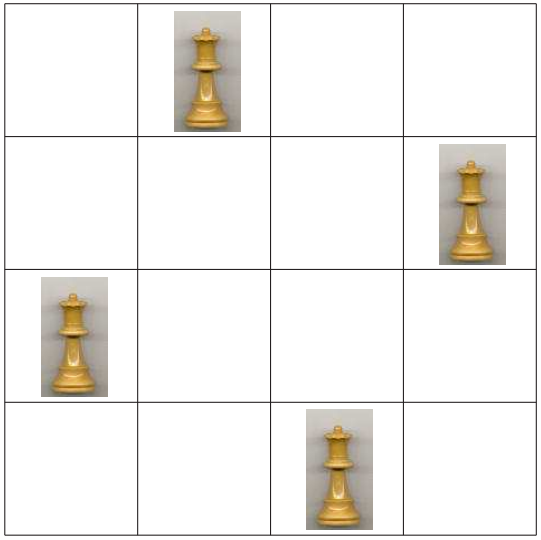
als dat mag: ga door . . .




Samengevat: je zoekt steeds verder zolang het nog kan ("depth first search") en gaat — als het mis loopt — terug ("backtracking") naar de laatste plek waar je nog een keuze open hebt staan.
Een heuristiek kan helpen bij de keuze van de volgende rij.

Op deze manier worden alleen “redelijke” kandidaten bekeken, allemaal één keer, en komt alles aan de beurt.



www.liacs.leidenuniv.nl/~kosterswa/gastlessen/dame.html

Vaak wordt backtracking uitgeprogrammeerd met behulp van recursie. Het is namelijk zo dat je soms het oorspronkelijke probleem als een deelprobleem herkent.

Een eenvoudig voorbeeld van recursie vind je bij het sorteren van een rijtje getallen. Zoek het kleinste getal op, zet dit vooraan, en (recursie!) sorteer de rest. Bijvoorbeeld: $7\ 5\ 9\ 2\ 6\ 4\ 4 \rightarrow 2 \cdot 7\ 5\ 9\ 6\ 4\ 4 \rightarrow 2\ 4 \cdot 7\ 5\ 9\ 6\ 4 \rightarrow \ldots$

Let er wel op dat de recursie een keer moet afbreken: in een eenvoudig basisgeval. Bij sorteren is dat wanneer je nog maar één getal over hebt. En bij de Russische poppetjes bij het kleinste poppetje.

De wiskunde maakt ook gebruik van recursieve definities. Zo wordt $n!$ ($n$ faculteit) vaak netjes gedefinieerd als:

$$n! = \begin{cases} 1 & \text{als } n = 0 \\ n \times (n-1)! & \text{als } n > 0 \end{cases}$$

in plaats van

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$$

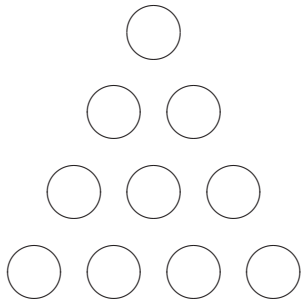
Er geldt namelijk zeker niet

$$4! = 4 \times 3 \times 2 \times \cdots \times 3 \times 2 \times 1$$

Hoe programmeer je recursie? Laten we een voorbeeld bekijken in de programmeertaal C++.

De som van de getallen 1 tot en met 100 is gelijk aan 100 plus de som van de getallen 1 tot en met 99, en die laatste som is weer 99 plus de som van de getallen 1 tot en met 98, . . . enzovoort. In C++:

```
int som (int n) {
   if ( n == 1 ) return 1;  // basisgeval
   else return ( n + som (n-1) );
}  // som
```



C++ gebruikt functies, bijvoorbeeld een functie `geenaanval` die controleert of de dame op de zoveelste rij geen problemen heeft met dames op eerdere rijen.

De belangrijkste functie is `zetdames`. De recursieve functie probeert een dame in de `rij`-de rij te zetten. Het idee is:

- Als het de 9-de rij is (van een $8 \times 8$ bord) hebben we blijkbaar een nieuwe oplossing gevonden: succes! We drukken deze stand af — en turven hem.

- Anders: probeer één voor één alle kolommen. Als we ergens een dame neer mogen zetten (gebruik `geenaanval`) doen we dat, en gaan op dezelfde manier door: recursie!

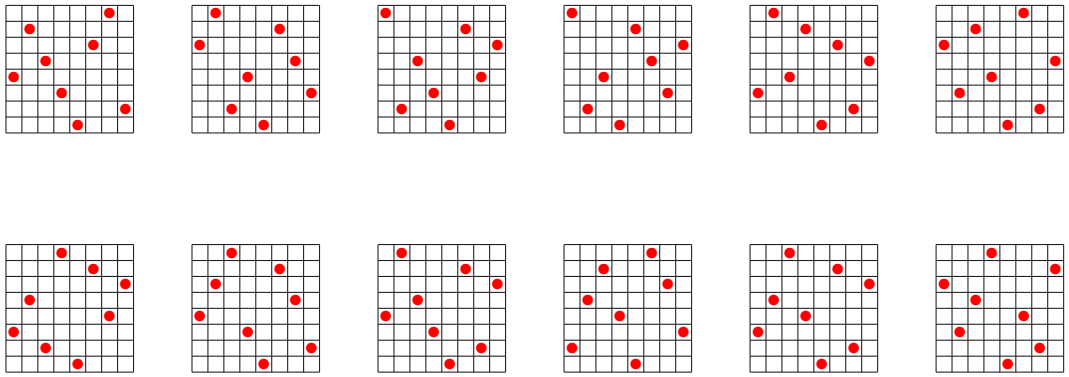
## De functie zetdames in C++

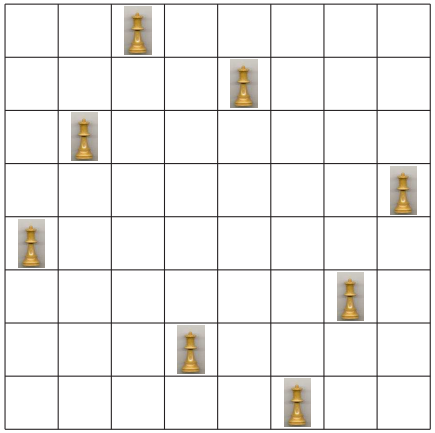
```
void zetdames (int grootte, int rij, int S[ ], int & teller) {
  int kolom;
  if ( rij == grootte + 1 ) {
    drukaf (grootte, S);
    teller++;
  }  // if
  else
    for ( kolom = 1; kolom <= grootte; kolom++ ) {
      S[rij] = kolom;
      if ( geenaanval (rij, S) )
        zetdames (grootte, rij+1, S, teller);
    }  // for
}  // zetdames
```

## Het gehele C++-programma

```
#include <iostream>
using namespace std;
const int MAX = 20;
bool geenaanval (int rij, int S[ ]) {
  bool veilig = true; int hulprij = 1;
  while ( veilig && ( hulprij < rij ) ) {
    veilig = ( ( S[rij] != S[hulprij] ) && ( S[rij] - S[hulprij] != rij - hulprij )
             && ( S[rij] - S[hulprij] != hulprij - rij ) );
    hulprij++; }//while
  return veilig; }//geenaanval
void drukaf (int grootte, int S[ ]) {
  for ( int i = 1; i <= grootte; i++ ) cout << S[i] << " ";
  cout << endl; }//drukaf
void zetdames (int grootte, int rij, int S[ ], int & teller) {
  int kolom;
  if ( rij == grootte + 1 ) { drukaf (grootte, S); teller++; }//if
  else
    for ( kolom = 1; kolom <= grootte; kolom++ ) { S[rij] = kolom;
      if ( geenaanval (rij, S) ) zetdames (grootte, rij+1, S, teller); }//for
}//zetdames
int main ( ) {
  int S[MAX]; int grootte; int teller = 0;
  do {
    cout << "Grootte schaakbord ( < " << MAX << " ) .. "; cin >> grootte;
  } while ( grootte < 1 || grootte >= MAX );
  zetdames (grootte, 1, S, teller);
  cout << endl << "Aantal: " << teller << endl << endl; return 0;
}//main
```

Op het $8 \times 8$ schaakbord vind je zo 92 oplossingen. In essentie zijn er 12 verschillende, waaruit je door draaien en spiegelen (8 mogelijkheden) ze allemaal kunt maken. Er is één wat meer symmetrische oplossing.





# Vragen?