

Informatica: Java reeks 4: Conway's Game of Life

Ruben De Smet

2^{de} semestre 2021

Samenvatting

Dit is een “spelletje” zonder spelers dat werd uitgevonden door Conway in 1970. Het bestaat uit een 2-dimensionaal speelveld, waarin elke cel “uit” of “aan”, **False** of **True**, ofwel “dood” of “levend” kan zijn; zie bijvoorbeeld fig. 1a.

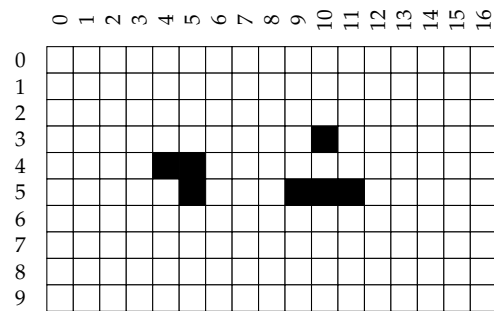
Gegeven een initieel “beginveld”, evolueert het veld verder zonder extra input.

We maken vandaag Conway's Game of Life, op een object-geïntende manier en *met* een grafische interface.

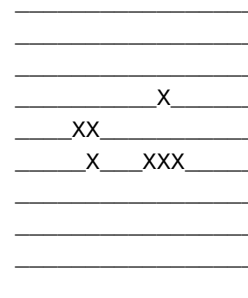
Nieuwe cellen worden geboren, en oude cellen sterven af. Hiertoe bestaan vier eenvoudige regels:

- 1) Elke levende cel met *minder dan twee* levende buren gaat dood, als door onderbevolking.
- 2) Elke levende cel met *twee of drie* levende buren blijft in leven.
- 3) Elke levende cel met meer dan drie levende buren sterft, als door overbevolking.
- 4) Elke dode cel met exact drie levende buren komt tot leven, als door reproductie.

Een cel heeft acht burenen; ze worden zowel links, recht, boven, onder *als* diagonaal geteld.



(a) Zwarte cellen zijn "levend", witte cellen zijn "dood"



(b) Resultaat van de printveld functie bij aanroepen met het beginveld.

Figuur 1 – Conway's game of life

Oefening 18 (Conway logica). De `class ConwayField` is verantwoordelijk voor de logica van Conway's Game of Life.

- (a) Maak een `enum` `CellState`. Een `enum` is een *type* dat een discreet aantal benoemde waarden kan aannemen. Noem de *varianten* van de `enum` `DEAD` en `ALIVE`. **Hint:** Een `enum` variant in Java wordt in HOOFDLETTERS geschreven. Je kan trouwens ook methodes toevoegen aan `enums`!
- (b) Maak de `class ConwayField`. Voeg een `public static void main(...)` toe om je veld te kunnen testen.
- (c) Voeg een geneste array (geen `ArrayList<>`¹) toe van `CellState`'s.
Maak een constructor voor `class ConwayField` die een leeg veld maakt. Geef de grootte van het lege veld mee aan deze constructor als twee `integers`.
- (d) Implementeer de `public String toString()` methode zodanig dat de output overeenkomt met fig. 1b. Deze kan je nu gebruiken om alles wat volgt te testen.
- (e) Maak een tweede constructor die — behalve de grootte van het veld — ook een `List<Point>` van levende velden neemt. Maak `class Point` als nieuwe klasse, met twee publieke attributen (`int x, y`;) een constructor voor de attributen.
Hint: Je kan je een andere constructor oproepen met `this(...)`!
- (f) Zorg dat `class Point` een nette `public String toString()` heeft (e.g. `"(x, y)"`).
Hint: Om bovenstaande methode te testen, kan je `Arrays.asList` gebruiken:

```
1 ConwayField testVeld = new ConwayField(17, 10, Arrays.asList(new Point[]
    ↪ {
2     new Point(4, 4),
3     new Point(5, 4),
4     ...
5     new Point(11, 5),
6 }));
7 System.out.println(testVeld);
```

- (g) Maak een `private int countNeighboursOf(int x, int y)` die de *levende* burens van een cel telt.
- (h) Maak een methode `public ConwayField evolve()` die de evolutie van een speelveld teruggeeft.
Doe dit door een nieuw veld te maken en voor elke cel regels (1)–(4) toe te passen. Gebruik `countNeighboursOf` om het aantal burens te berekenen.

Oefening 19 (GUI). Maak een grafische interface die elke twee seconden het veld doet evolueren. Je kan je baseren op stukken code van oefeningen 13 en 14 van vorige reeks.

- (a) Maak een `class GameOfLifePanel` die van `JPanel` overerft. Gebruik dezelfde code als in oefeningen 13 en 14 om je paneel met een `JFrame` te starten.
- (b) Voeg een attribuut toe aan je `class GameOfLifePanel` om het de game state bij te houden.

¹Een `ArrayList<>` is vooral nuttig als je een lijst wil van een veranderlijk *aantal* elementen. De grootte van het speelveld staat vast bij het aanmaken van de `ConwayField`, en enkel de inhoud verandert nog.

- (c) Overschrijf de `paintComponent(Graphics g)` en teken het speelveld. Gebruik rechthoeken van 20×20 pixels².
Hou je code hiertoe zo simpel mogelijk.
- (d) Maak — zoals in oefeningen 13 en 14 — een `UpdateTimerTask` en een `Timer` die elke 2000 ms je speelveld updatet.

Oefening 20 (Interactie). *We maken een tweede versie van de GUI, waarin we manueel het evolutieproces beheren. Met de Enter toets zorg je voor een evolutiestap. Met de + toets wordt je veld groter en met de - toets wordt je veld kleiner.*

- (a) Om te vermijden dat we tweemaal de code moeten schrijven om een veld te tekenen, verplaatsen de relevante code naar een nieuwe draw methode in `class ConwayField`, net zoals in opgave 14.(d) van de vorige reeks.
- (b) Dupliceer de `class GameOfLifePanel`; noem de nieuwe versie `class InteractiveGameOfLifePanel`. Verwijder de timergerelateerde code.
- (c) Implementeer de `KeyListener` interface op je `InteractiveGameOfLifePanel`. Voeg `this` toe als `KeyListener` m.b.v. de `addKeyListener` methode van het `JPanel`.
Om toetsen te kunnen afvangen, moet je ook aan Java vragen om ons panel de “focus” van het geheel te laten zijn.
Hint: Je kan ook Eclipse de nodige methodes laten genereren, wanneer je `implements` `KeyListener` hebt toegevoegd.

```

1  public class InteractiveGameOfLifePanel extends JPanel implements
    ↳ KeyListener {
2      BallPanel() {
3          this.addKeyListener(this);
4      }
5
6      @Override
7      public void addNotify() {
8          super.addNotify();
9          this.requestFocusInWindow();
10     }
11
12     @Override
13     void keyPressed(KeyEvent e) { ... }
14 }

```

Oefening 21 (Inheritance). `class GameOfLifePanel` en `class InteractiveGameOfLifePanel` hebben gemeenschappelijke code. Het verschil is erg klein en de rest moet je kunnen hergebruiken. Java doet dit aan de hand van inheritance: overerving.

- (a) Hernoem `class GameOfLifePanel` naar `class AutomaticGameOfLifePanel`.
- (b) Maak een nieuwe `class GameOfLifePanel` waarin je de gemeenschappelijke code zet, en zorg dat beide varianten van de `class GameOfLifePanel` overerven van `class AutomaticGameOfLifePanel`.

²Als je een HiDPI scherm hebt, gebruik je best grotere rechthoeken.

```

1  public class AutomaticGameOfLifePanel extends GameOfLifePanel {
2      ...
3  }
4  public class InteractiveGameOfLifePanel extends GameOfLifePanel {
5      ...
6  }

```

Extra oefening 22 (Verbeterde `class ConwayField` constructor). Een beginveld definiëren zoals in opgave 18.(f) is nogal omslachtig. Wanneer je andere begincondities wil testen, moet je alle coördinaten aflezen en moet je relatief veel code produceren.

- (a) Maak een nieuwe constructor voor `class ConwayField` die een lijst van Strings neemt. Elke String stelt een rij voor en ze moeten dus even lang zijn. Wanneer je een 'X' tegenkomt plaats je een levende cel, andere cellen zijn per definitie dood.
- (b) Maak een nieuwe constructor voor `class ConwayField` die een enkele String neemt. Splits de enkele string op "\n" en hergebruik de vorige constructor.

Nu kan je makkelijk een nieuw veld maken:

```

1  new ConwayField(Arrays.asList(
2      "----",
3      "-XX-",
4      "-XX-",
5      "----"
6  ));

```

Je kan nu ook een veld inlezen van een bestand!

Extra oefening 23 (Muis input). Een nuttige manier om het veld te kunnen instellen is met de muis: wanneer je klikt op een cel verandert die van dood naar levend of omgekeerd.

Hiermee kan je enkele beginsituaties van het internet (bvb. https://en.wikipedia.org/wiki/Conway's_Game_of_Life#Examples_of_patterns) uittesten.

- (a) Implementeer de `MouseListener` interface op je `InteractiveGameOfLifePanel`, analoog aan opgave 20.(c).
- (b) Bereken aan de hand van de muiscoördinaten de coördinaten van de cel waarop geklikt werd. Hou rekening met de grenzen van je speelveld! Verander een dode cel naar een levende of omgekeerd, wanneer er geklikt werd.

Hint: Op <https://docs.oracle.com/javase/tutorial/uiswing/events/mouselistener.html> staat er een tutorial voor de `MouseListener`; op <https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseEvent.html> vind je de documentatie van de `MouseEvent`.

Extra oefening 24 (Fraaiier maken). In de eenvoudigste versie teken je het veld beginnend in de linkerbovenhoek. Probeer je veld gecentreerd in het midden van je `JFrame` te tekenen.

Extra oefening 25 (Volgende en vorige). Gebruik kleuren of extra vormen om te tonen wat de volgende en vorige stap in de evolutie was. Terugrekenen kan je niet, maar je kan wel de historiek bijhouden!