7.6 Zweidimensionale Arrays (optional)

Und wieder kommen wir auf ein Beispiel aus der Biologie zurück, nachdem wir uns in der Aufgabe 7.11 schon mit Marienkäfern beschäftigt haben. Wir wollen diesmal - gemeinsam in einer Art Workshop - einen so genannten **Zellulären Automaten** realisieren. Am Ende des Workshops gibt es dann wieder eine Reihe von Aufgaben, die Sie lösen können, wenn Ihr Kurslehrer das für sinnvoll hält.

Allgemeines über Zelluläre Automaten

Ein Zellulärer Automat ist ein 4-Tupel Z = (R, N, Q, δ), mit

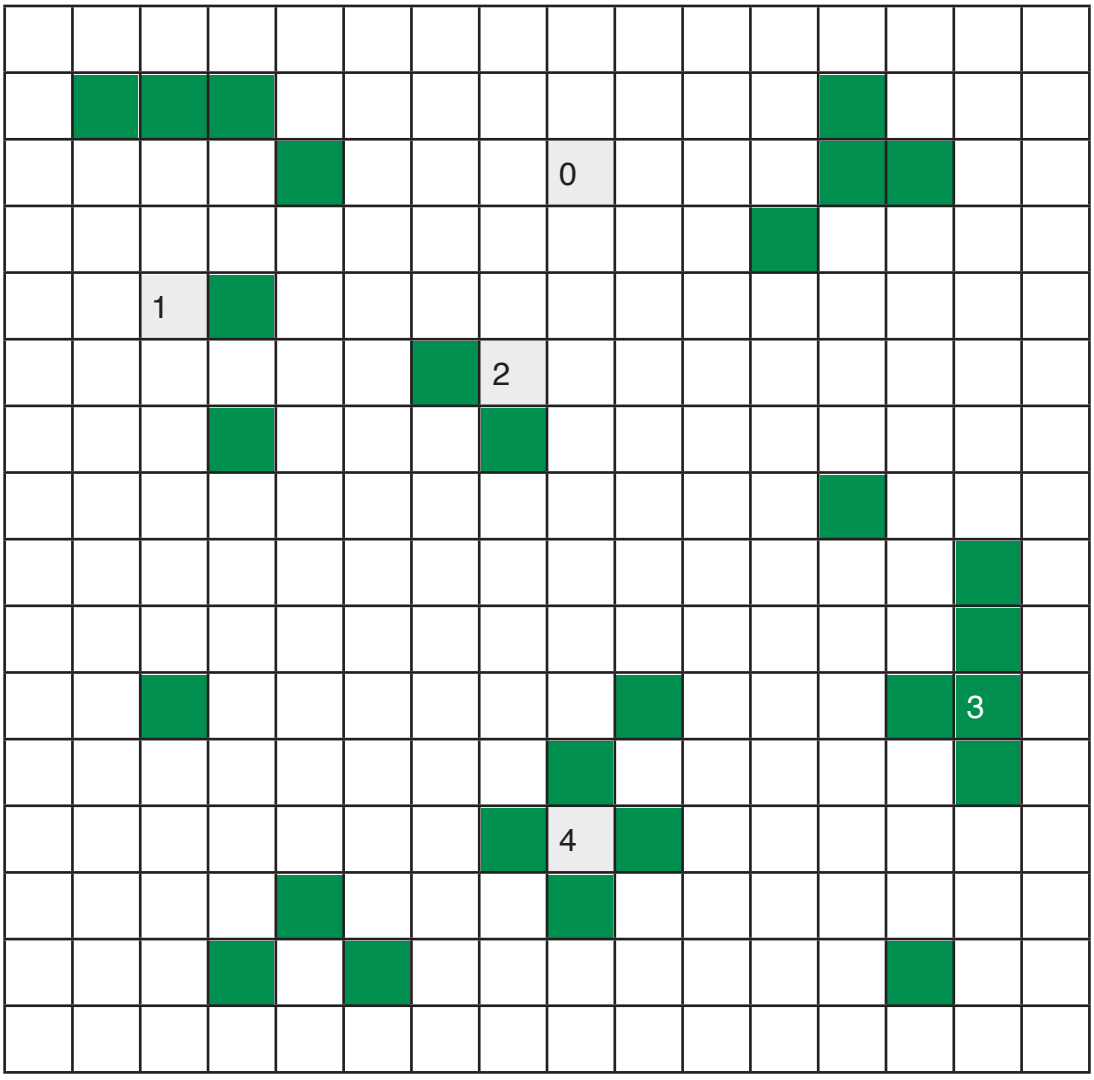
R = ein Raum (z.B. eine zweidimensionale Matrix)

N = eine endliche Nachbarschaft (z.B. die vier Nachbarzellen)

Q = eine endliche Zustandsmenge (z.B. { gesund, krank, tot })

δ = eine Überführungs-Funktion

Nach dieser sehr formalen Definition wollen wir uns nun ein konkretes Beispiel anschauen:



7.13 Ein Zellulärer Automat mit 256 Zellen, 4er Nachbarschaft und zwei Zuständen

In der Abbildung 7.13 sehen wir einen zweidimensionalen Zellulären Automaten mit 16 x 16 Zellen, die in je zwei Zuständen vorkommen können, weiß und grün (was man z.B. als “tot” und “lebendig” interpretieren könnte). Jede Zelle hat genau

vier Nachbarzellen, nämlich links und rechts sowie oben und unten. Fünf der Zellen wurden besonders gekennzeichnet (grau), die Zahl der "lebenden" Nachbarzellen wurde in diesen Zellen vermerkt.

Damit hätten wir die Komponenten R, N und Q dieses Zellulären Automaten schon einmal näher beschrieben. Es bleibt nur noch die Übergangs-Funktion δ. Wir wollen eine solche Funktion δ erst einmal "umgangssprachlich" und anschließend formal definieren.

Die Übergangs-Funktion

Wenn eine Zelle keine lebenden Nachbarn hat, soll sie sterben (vor Einsamkeit oder weil sie keine Nahrung hat oder keinen Fortpflanzungspartner findet...). Hat eine Zelle einen oder zwei lebende Nachbarn, so soll sie selbst leben (egal, ob die Zelle vorher tot war oder schon gelebt hat). Bei drei oder vier lebenden Nachbarn soll eine Zelle sterben (vor Stress, Gedränge, zu viele Feinde oder Parasiten etc.). Eine tote Zelle bleibt natürlich tot.

Dies ist natürlich nur ein Beispiel für eine solche Übergangs-Funktion. Genau so gut hätte ich mir hier eine wesentlich komplexere Übergangsfunktion ausdenken können. Generell gilt aber, dass der neue Zustand einer Zelle von der Zahl der lebenden Zellen in der Nachbarschaft abhängt. Eine etwas formalere Darstellung wäre die folgende Tabellenform:

Zahl der lebenden Nachbarn	Neuer Zustand
0	0
1	1
2	1
3	0
4	0

7.14 Formale Darstellung einer Übergangs-Funktion

Soll der neue Zustand einer Zelle nicht nur von der Zahl der lebenden Nachbarn abhängig sein, sondern auch vom alten Zustand der Zelle, so wird die Übergangs-Funktion komplizierter, auch die tabellarische Darstellung sieht dann etwas komplizierter aus:

Zahl der lebenden Nachbarn	Alter Zustand = "tot"	Alter Zustand = "lebendig"
0	0	0
1	0	1
2	0	1
3	1	0
4	1	0

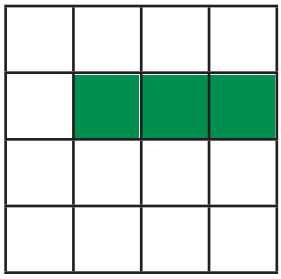
7.15 Formale Darstellung einer komplizierteren Übergangs-Funktion

Schauen wir uns nun an, welche Auswirkungen diese Übergangs-Funktion (wir arbeiten mit der einfacheren Funktion weiter) auf den Zellraum, also die Anordnung der Zellen hat. Betrachten wir zunächst die mit der Ziffer "0" markierte Zelle in der Abbildung 7.13. Diese Zelle hat keine lebenden Nachbarn, also wird ihr Zustand in der nächsten Generation wieder "tot" sein. Die mit der Ziffer "1" gekennzeichnete Zelle wird in der nächsten Generation den Zustand 1 oder "lebendig" haben, ebenso die mit der Ziffer "2" gekennzeichnete Zelle. Die lebende Zelle mit der Ziffer "3" wird sterben, und die tote Zelle "4" wird wegen ihrer vier lebenden Nachbarn in dem Zustand 0 verbleiben.

Berechnung der nächsten Generationen

Der Zelluläre Automat berechnet jetzt für jede einzelne Zelle mithilfe der Übergangs-Funktion den Folgezustand in der nächsten Generation, realisiert diesen neuen Zustand aber erst dann, wenn er mit der letzten Zelle fertig ist. Somit hängt der Zustand einer Zelle ausschließlich vom aktuellen Zustand der Nachbarzellen ab, und nicht vom bereits errechneten Folgezustand. Schauen wir uns das mal für ein sehr kleines Beispiel an (ich habe nämlich keine Lust, für 256 Zellen per Hand den Folgezustand zu berechnen).

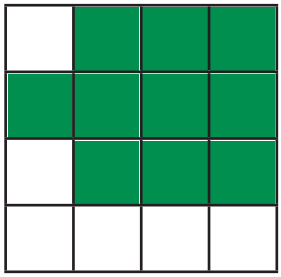
Ausgangskonfiguration (Generation 1):



Zahl der lebenden Nachbarn in Generation 1:

0	1	1	1
1	**1**	**2**	**1**
0	1	1	1
0	0	0	0

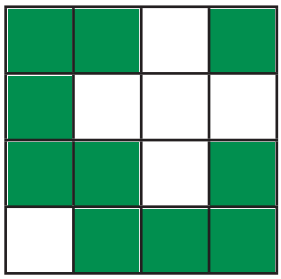
Folgezustände in Generation 2 laut Übergangs-Funktion:



Zahl der lebenden Nachbarn in Generation 2:

2	**2**	**3**	**2**
1	**4**	**4**	**3**
2	**2**	**3**	**2**
0	1	1	1

Folgezustände in Generation 3 laut Übergangsfunktion:



7.16 Ein kleines Beispiel, durchgerechnet für drei Generationen

Eine Torus-Welt

Bei der Berechnung der Nachbarschaft sind übrigens die Randzellen sowie die Eckzellen problematisch. Zwangsläufig haben Randzellen nur drei Nachbarn, während Eckzellen sogar nur zwei Nachbarn haben. Entweder muss dies bei der Übergangs-Funktion berücksichtigt werden, was diese natürlich enorm verkompliziert, oder es muss in die geometrische Trickkiste gegriffen werden, die da heißt: Torus-Welt.

Wir verkleben einfach den linken Rand unserer Welt mit dem rechten Rand, so dass wir einen Zylinder erhalten. So hat ein Feld des linken Randes einen rechten Nachbarn.

1,1	2,1	3,1	4,1
1,2	2,2	3,2	4,2
1,3	2,3	3,3	4,3
1,4	2,4	3,4	4,4

7.17 Nachbarschaft in einer Zylinder-Welt

Die Nachbarn des Feldes mit den Koordinaten (1,3) sind jetzt: (1,2), (2,3), (1,4) und (4,3).

Was machen wir aber mit den Feldern, die am oberen Rand oder am unteren Rand liegen? Nun, wir verfahren hier genau so und verkleben den oberen Rand mit dem unteren Rand. Wir biegen unseren Zylinder und erhalten einen Torus:

7.18 Ein Torus

Ein Torus kennen Sie sicherlich aus dem Alltag: Ein Fahrradschlauch oder ein Reifen ist im Grunde auch ein Torus. In unserer Torus-Welt hat jetzt jedes Feld vier Nachbarn, Randfelder und Eckfelder gibt es nicht mehr.

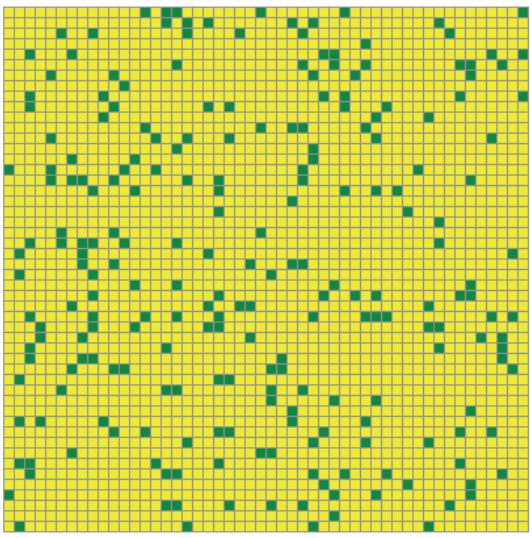
Wir wollen jetzt einen solchen Zellulären Automaten mithilfe eines Java-Applets simulieren. Dabei wollen wir eine Matrix aus 50 x 50 Zellen mit zwei Zuständen zu einer Torus-Welt zusammenfügen, die Zellen mit dem Zufallsgenerator initialisieren und dann mithilfe eines Timers die Folgegenerationen automatisch erzeugen und zeichnen lassen.

Schritt 1 - Ein Spielfeld

Wir beginnen mit zwei einfachen Java-Klassen, dem eigentlichen Spielfeld und dem Applet zum Anzeigen des Spielfeldes (und später der Buttons und der anderen Bedienelemente). Hier der komplette Quelltext der Klasse **Spielfeld**:

```
import java.util.Random;
import java.awt.*;

public class Spielfeld
{
   int[][] feld;
   Random rand = new Random();

   public Spielfeld()
   {
      feld = new int [50][50];
      for (int y = 0; y < 50; y++)
         for (int x = 0; x < 50; x++)
         {
            int zufall = rand.nextInt(10);
            if (zufall < 9)
                feld[x][y] = 0;
            else
                feld[x][y] = 1;
         }
    }

   public void anzeigen(Graphics g)
   {
      for (int y = 0; y < 50; y++)
         for (int x = 0; x < 50; x++)
         {
            if (feld[x][y] == 1)
               g.setColor(new Color(0,127,0));
            else if (feld[x][y] == 0)
               g.setColor(Color.YELLOW);

            g.fillRect(50+x*8, 20+y*8,8,8);
            g.setColor(new Color(127,127,127));
            g.drawRect(50+x*8, 20+y*8,8,8);
         }
   }
}
```

```
import java.awt.*;
import javax.swing.*;

public class Anzeige extends JApplet
{
   Spielfeld welt;

   public void init()
   {
       welt = new Spielfeld();
   }

   public void paint(Graphics g)
   {
       welt.anzeigen(g);
   }
}
```



7.19 Die erzeugte Welt

Hier werden die lebendigen Felder durch die Farbe Dunkelgrün und die toten Felder durch die Farbe Grau repräsentiert. Es steht Ihnen frei, dies zu ändern. Übrigens sehen Sie hier auch, was ein zweidimensionaler Array ist und wie man einen solchen zweidimensionalen Array deklariert und initialisiert. Der Quelltext des Applets ist kurz, Sie finden ihn rechts oben auf dieser Seite. Eine Beispiel-Ausgabe des Applets sehen Sie in der Abbildung 7.19.

Übung 7.12 (1 Punkt)

Ihre erste Übung in diesem Workshop ist es, die beiden Klassen zum Laufen zu bringen. Experimentieren Sie mit der Größe und der Farbe der Felder herum und manipulieren Sie den Zufallsgenerator, so dass zum Beispiel mehr lebende Zellen erzeugt werden.