Zelluläre Automaten

Java

Alfred Nussbaumer

Zelluläre Automaten haben sich in den letzten Jahrzehnten zu einem beliebten Thema der EDV entwickelt. Das „Game of Life“ ***[1]*** beispielsweise ist allen Programmierern in vielen Programmiersprachen bekannt; es wurde vom Mathematiker John Horton Conway von der Universität Cambridge in den sechziger Jahren erfunden. Durch Stephen Wolframs neues Buch „A New Kind of Science“ ***[2]*** dürfte das Interesse an zellulären Automaten wieder sprunghaft angestiegen sein. Dieser Beitrag stellt eine einfache Java-Applikation vor, mit der die zeitliche Entwicklung eines solchen Automaten dargestellt wird.

1. Grundlagen

Je nach der Dimension eines zellulären Automaten wird ein Band, eine Fläche oder ein Raumbereich in lauter gleiche Zellbereiche unterteilt. Für jede Zelle gibt es verschiedene Zustände, die im einfachsten Fall „Leben“ oder „Tod“ bedeuten. Das Verhalten der Zellen wird nun in aufeinanderfolgenden Zeitschritten („Generationen“) untersucht: Der Zustand einer Zelle in der nächsten Generation wird durch ihre Nachbarschaft bedingt. In einem zellulären Automaten ist nun der Algorithmus verpackt, mit dem aus dem Zustand aller Zellen zu einem bestimmten Zeitpunkt t der Zustand aller Zellen zum nächsten Zeitpunkt t+1 berechnet werden kann. Nach jedem Rechenschritt wird der Zustand aller Zellen dargestellt.

Dies wird im *„Game of Life“* folgendermaßen realisiert: Jede Zelle befindet sich in einem von zwei Zuständen, nämlich „lebendig“ oder „tot“. Die Regeln sind einfach: Eine tote Zelle erwacht nach dem Ablauf eines Zeitschrittes zu neuem Leben, wenn sie davor genau drei lebendige Nachbarn besessen hat. Eine lebendige Zelle dagegen stirbt mit dem nächsten Zeitschritt, wenn sie von weniger als zwei oder mehr als drei Nachbarn umgeben ist. Diese zwei simplen Regeln verleihen dem Automaten ein Verhalten, das allein von der Anfangskonfiguration aus toten und lebendigen Zellen abhängt. Interessanterweise findet sich nach vielen Schritten stets eine Anordnung von Zellen, in der sich die Zellmuster periodisch wiederholen.

2. Ein eindimensionaler zellulärer Automat

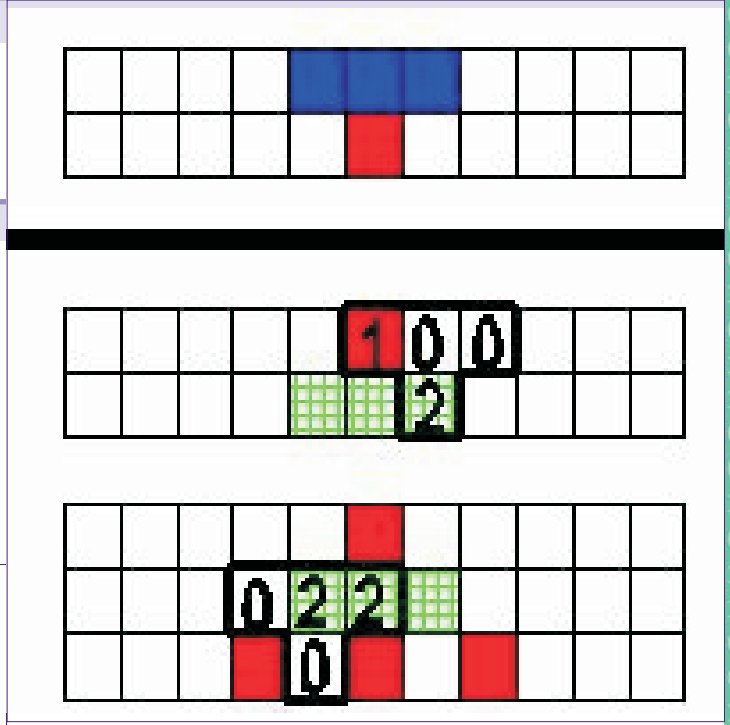
Wir betrachten an dieser Stelle einen eindimensionalen Automaten, weil für ihn der Algorithmus besonders einfach zu formulieren ist. Außerdem kann das „zeitliche Verhalten“ der Zellen in nacheinander angezeigten Reihen dargestellt werden. Untersuchungen des zeitlichen Verhaltens eines Systems sind damit besonders einfach duchzuführen.

In welchen Zustand eine Zelle nach einem Zeitschritt gelangt, hängt von der Zelle selbst und von ihren beiden Nachbarn ab. Dazu wird einfach die Summe der Zustände der drei Zellen berechnet. Eine extra angegebene „Reproduktionsregel“ legt fest, welchen Zustand die Zelle aufgrund der berechneten Summe annimmt. Dabei kann eine Zelle mehrere verschiedene Stadien, beispielsweise zwischen 0 und 4 annehmen.

Jede „Generation“ von Zellen wird in einer eigenen Reihe dargestellt. Daher sind für den Zustand einer Zelle drei Zellen der vorausgehenden Generation („Elterngeneration“) entscheidend:

Im Bild ***(Seite oben)*** wird zunächst gezeigt, dass der Wert einer Zelle aus den Eigenschaften der drei angrenzenden Zellen in der vorangegangenen Reihe („Elterngeneration“) berechnet wird. Die Farben der Felder entsprechen Zahlen: Weiß für 0, Rot für 1, Grün für 2, usw. Im unteren Teil des Bildes wird somit der Algorithmus dargestellt: Der Wert der neuen Zelle berechnet sich aus der Summe der drei benachbarten „Elternzellen“. Die Codezahl bestimmt schließlich den Zustand der „Tochterzelle“:

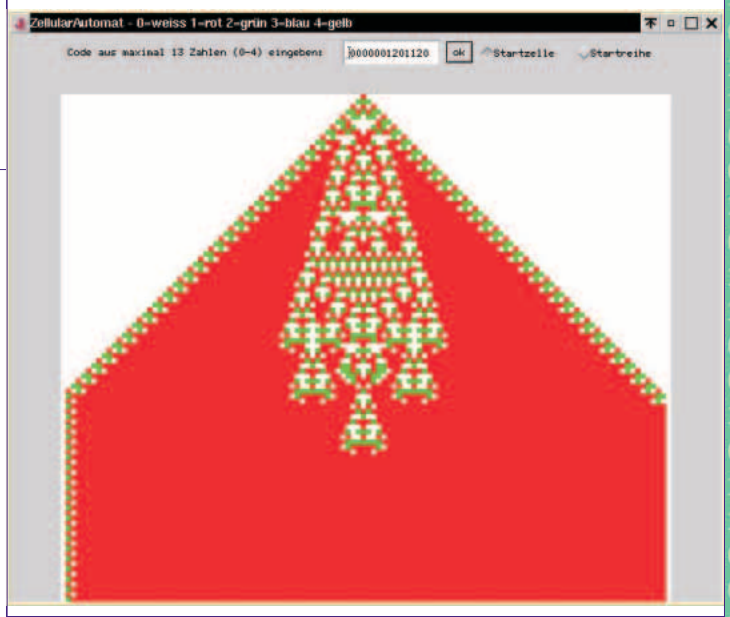
```
12 11 10 9 8 7 6 5 4 3 2 1 0
 0  0  0 0 0 0 1 2 0 1 1 2 0
```



Für dieses Beispiel wurde die Codezahl 0000001201120 verwendet (3 verschiedene Farben!). So ergibt die Summe 1 den neuen Zustand „2“ für die Tochterzelle - sie wird also grün eingefärbt. Die Summe 4 ergibt „0“ und somit eine weiß eingefärbte Tochterzelle. Aufgrund der festgelegten Regeln “entwickelt” sich eine komplizierte Struktur aus einer (oder mehreren) Ausgangszellen. Da jeder Zeitschritt einer neuen Reihe entspricht, kann aus dem erhaltenen Bild die zeitliche Entwicklung einer “Anfangspopulation” untersucht werden...

3. Das Programmbeispiel

Im folgenden Bild sieht man die Entwicklung des oben angeführten zellulären Automaten über 100 Generationen; die Zeitachse



für die aufeinander folgenden Generationen verläuft von oben nach unten. Es fällt auf, dass von einer roten Zelle ausgehend bald nahezu alle Zellen diesen gleichen Zustand („1“ - rot) annehmen.

Im Programm kann ein bis zu 13 Stellen langer Reproduktionscode eingegeben werden, der die Ziffern von „0“ bis „4“ enthalten kann. Zu beachten ist dabei, wie die möglichen Ziffern (von „0“ bis „4“) und die Anzahl der Stellen der Codezahl (Index 0 bis 12) zusammenspielen. Außerdem kann gewählt werden, ob die Entwicklung von einer Zelle oder von einer ganzen Reihe von 120

alfred.nussbaumer@schule.at

Zellen beobachtet werden soll, deren Zustände zu Beginn der Simulation zufällig gesetzt werden.

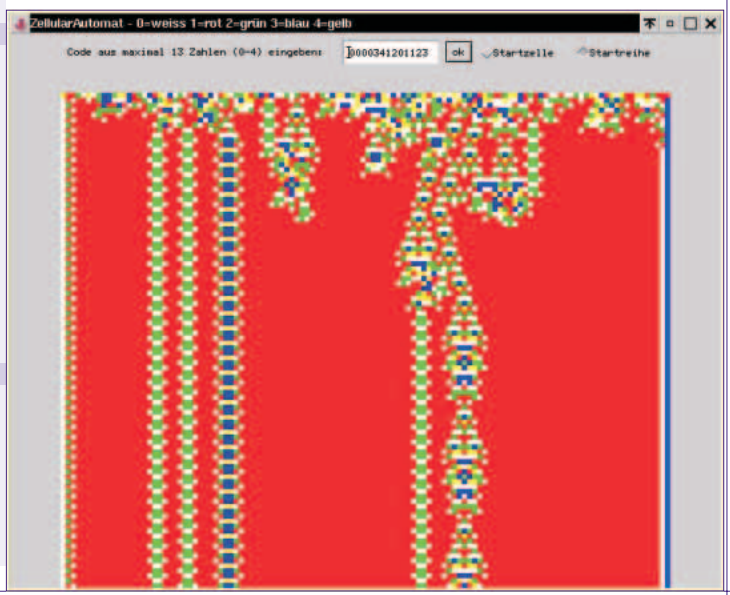
```
import java.awt.*;
import java.awt.event.*;
import java.util.Random;

public class zellular extends Frame implements ActionListener,
ItemListener {

  int code[] = new int [13];
  Button ok;
  TextField eingabe;
  CheckboxGroup gruppe;
  Checkbox starteins;
  Checkbox startreihe;
  boolean zufall;
  Random ergebnis;

  public static void main(String arguments[]) {
    zellular proggi = new zellular();
    WindowListener wl = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    proggi.addWindowListener(wl);
    proggi.setLocation(100,100);
    proggi.setSize(700,560);
    proggi.show();
  }

  zellular() {
    super("ZellularAutomat - 0=weiss 1=rot 2=grün 3=blau 4=gelb");
    setLayout(new FlowLayout());
    Label was =
      new Label("Code aus maximal 13 Zahlen (0-4) eingeben:");
    add(was);
    eingabe = new TextField("0000001201120",13);
    add(eingabe);
    ok = new Button("ok");
    add(ok);
    ok.addActionListener(this);
    gruppe = new CheckboxGroup();
    starteins = new Checkbox("Startzelle",gruppe,true);
    startreihe = new Checkbox("Startreihe",gruppe,false);
    add(starteins);
    starteins.addItemListener(this);
    add(startreihe);
    startreihe.addItemListener(this);
    ergebnis = new Random();
  }

  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == ok)
      codeeintragen();
  }

  public void itemStateChanged (ItemEvent e) {
    if (e.getItemSelectable() == starteins) {
      zufall = false;
    }
    if (e.getItemSelectable() == startreihe) {
      zufall = true;
    }
  }

  public void codeeintragen() {
    String eintrag = eingabe.getText();
    int el = eintrag.length();
    if (el>13) el=13;
    for (int s=0; s<eintrag.length();s++) {
      if (eintrag.charAt(s) == '0') code[12-s+el-13]=0;
      if (eintrag.charAt(s) == '1') code[12-s+el-13]=1;
      if (eintrag.charAt(s) == '2') code[12-s+el-13]=2;
      if (eintrag.charAt(s) == '3') code[12-s+el-13]=3;
      if (eintrag.charAt(s) == '4') code[12-s+el-13]=4;
    }
    for (int s=eintrag.length();s<13;s++) code[s]=0;
    repaint();
  }

  public void paint(Graphics bs) {
    int zeilealt[] = new int[120];
    int zeileneu[] = new int[120];
    int summe;
    int i;
    int zeile;

    if (zufall) {
      for (i=0;i<120;i++) {
        zeilealt[i]=zufallszahl(5);
      }
    } else {
      for (i=0;i<120;i++) {
        zeilealt[i]=0;
      }
      zeilealt[59]=zufallszahl(4)+1;
    }
    zeile=0;

    for (zeile=0;zeile<100;zeile++) {
      for (i=0;i<120;i++) {
        if (zeilealt[i]==0) bs.setColor(Color.white);
        if (zeilealt[i]==1) bs.setColor(Color.red);
        if (zeilealt[i]==2) bs.setColor(Color.green);
        if (zeilealt[i]==3) bs.setColor(Color.blue);
        if (zeilealt[i]==4) bs.setColor(Color.yellow);
        if (zeilealt[i]>4) bs.setColor(Color.black);
        bs.fillRect(i*5+50,zeile*5+60,5,5);

        summe=0;
        zeileneu[0]=0;
        zeileneu[119]=0;
        if ((i>0) && (i<119))
          summe=zeilealt[i-1]+zeilealt[i]+zeilealt[i+1];

        zeileneu[i] = code[summe];

      }
      for (i=0;i<120;i++) zeilealt[i]=zeileneu[i];
    }
  }

  public int zufallszahl(int grenze) {
    return (int)(ergebnis.nextDouble()*grenze);
  }
}
```

Im Programmbeispiel wird zusätzlich das Package „util“ zum Erzeugen von Zufallszahlen verwendet. Der Konstruktor erzeugt ein einfaches Flow-Layout mit einem Label, mit dem Texteingabeobjekt `eingabe` für die Eingabe des gewünschten Codes, mit den CheckButtons `starteins` und `startreihe` für die Wahl der Ausgangspopulation und mit der `ok`-Schaltfläche.

Sobald die Schaltfläche `ok` betätigt wurde, wird der im Texteingabefeld angegebene Reproduktionscode gelesen (Methode `codeeintragen()`) und die Ausgabe neu gezeichnet: Je nachdem, ob die Wahl `starteins` oder `startreihe` getroffen wurde, wird die erste Zeile entweder mit lauter Nullen bis auf die Zelle in der Mitte gefüllt, oder es werden in alle Zellen der ersten Zeile Zufallswerte zwischen 0 und 4 eingetragen.

Der Algorithmus sieht nun vor, dass in beide Randzellen „0“ eingetragen wird. Die Zustände aller anderen Zellen werden jeweils aus der Summe der drei Nachbarzellen ermittelt. Je nach diesem Ergebnis wird der neue Zustand aus dem Reproduktionscode gesetzt (`zeileneu[i] = code[summe];`).

Ist der Kode fehlerfrei compiliert, so steht einem weiten Feld von „Experimenten“ nichts im Weg. Dabei wird vorwiegend die Frage gestellt, welcher Reproduktionscode zu welchem Langzeitverhalten führt. Das Verhalten des eindimensionalen zellulären Automaten hängt wesentlich von der Codezahl und von der Aus-



gangsposition, also von der Anzahl der Zellen und von ihrer „Farbe“ (also von ihren „Zuständen“) ab. In diesem Sinn können zelluläre Automaten biologische Systeme oder physikalische Modelle darstellen.

Interessant ist, wie sich eine Population aus einer Reihe von Zellen mit zufällig vergebenen Zuständen entwickelt:

Bei bestimmten Reproduktionscodes und bei bestimmten Elterngenerationen entstehen Ketten: Diese „Populationen“ sterben offenbar nicht aus. Nur die Eigenschaften einer endlichen Kette (also einer Kette, die nach endlich vielen Schritten abbricht) können genau studiert werden. Im Allgemeinen kann das endgültige Verhalten des zellulären Automaten - etwa, ob eine Kette unregelmäßig bleibt oder sich wiederholt - nicht aus einer endlichen Anzahl von Schritten bestimmt werden. Weitere Informationen dazu finden sich in der Literatur und im Internet.

4. Erweiterungen

Das angegebene Beispiel kann im Rahmen der Unterrichtsarbeit, für Haus- oder Projektarbeiten in vielfältiger Weise erweitert werden:

- Der Programmcode soll hinsichtlich des Layouts der Java-Applikation verbessert werden.
- Für die Eingabe des Reproduktionscodes soll eine Eingabe-Fehlerbehandlung dafür sorgen, dass nur Ziffern zwischen „0“ und „4“ eingegeben werden.
- Der Benutzer soll die Anfangspopulation selbst festlegen bzw. editieren können.
- Der Zeitraum soll auf mehrere hundert Generationen ausgedehnt werden . Dazu sollen anstelle der Quadrate, die die einzelnen Zellen darstellen, nur Pixel in den entsprechenden Farben gesetzt werden. Das bedeutet, dass auch wesentlich mehr Zellen in einer Reihe bearbeitet werden müssen.
- Die Nachbarschaft soll auch auf die übernächsten Nachbarn ausgedehnt werden; dabei soll der Zustand dieser übernächsten Nachbarn mit einem Gewichtungsfaktor kleiner 1 (z.B. 0,5) berücksichtigt werden. Das Ergebnis ist jedenfalls auf ganze Zahlen zu runden.
- Andere zelluläre Automaten sind mit Hilfe von Internet-Publikationen zu recherchieren und darzustellen.
- ...

5. Literatur, Weblinks

[1] xlife; dieses Programm liegt aktuellen Linux-Distributionen bei.

[2] Wofram Stephen, „A New Kind of Science“, ISBN 1579550088

[3] Linux-Magazin 12/2002, „Spiele des Lebens“ (M. Mathys), S. 74 ff.

[4] Spektrum der Wissenschaft: Verständliche Forschung, „Chaos und Fraktale“ (1989), Artikel „Software für Mathematik und Naturwissenschaften“ (St. Wolfram), S. 194 ff

[5] http://cell-auto.com (Zelluläre Automaten)

[6] http://www.gymmelk.ac.at/~nus/informatik/wpf/JAVA/index.php?kat=appt&teil=zell (Applet zur obigen Java-Applikation)