

Institut für Geographie und angewandte Geoinformatik  
Universität Salzburg  
Hellbrunnerstraße 34, 5020 Salzburg

## **Geosimulation mittels Cellular Automata in einer Flash-Umgebung**

### **Projektarbeit (Modul 15)**

im Universitätslehrgang  
Geographical Information Science & Systems  
(UNIGIS MAS)

**Bearbeiter:** Dipl.-Ing. Dipl.-Wirt.Ing. Holger Meiß  
Anton Schosser-Str. 13, 4810 Gmunden  
Tel. 0664-4813517  
E-Mail: hmeiss@aon.at  
Lehrgangskennzahl: 734  
Matr.Nr.: 8355010

**Termin:** 10. Dezember 2001

## **Zusammenfassung**

Im Rahmen der vorliegenden Projektarbeit wird ein allgemeiner Simulator für zellulare Automaten (CA) in einer Flash 5/ActionScript-Umgebung entwickelt. Die Spezifikation eines CA soll mit Hilfe von XML erfolgen. Zunächst wird auf die programmtechnische Art und Weise, einen CA in einer Flash-Umgebung zu simulieren, eingegangen. Gleichzeitig werden die Anforderungen an die XML-Struktur erarbeitet, wobei auf das Game of Life Bezug genommen wird. Das so entwickelte ActionScript-Programm wird unter Anwendung dieser XML-Spezifikationsmethode für die Simulation von drei verschiedenen CA eingesetzt. Im Anhang wird auf das Abspielen von Flash-Dateien und auf Performanceprobleme eingegangen sowie die XML-Spezifikation eines CA zusammengefasst dargestellt.

# Inhaltsverzeichnis

<b>ZUSAMMENFASSUNG.....</b>	<b>I</b>
<b>INHALTSVERZEICHNIS .....</b>	<b>II</b>
<b>1. EINLEITUNG .....</b>	<b>1</b>
<b>2. AUFBAU EINES ZELLULAREN AUTOMATEN IN EINER FLASH 5-UMGEBUNG .....</b>	<b>1</b>
2.1. DATENSTRUKTUR.....	1
2.2. EIN EINFACHER ZELLULARER AUTOMAT .....	3
<b>3. DIE SPEZIFIKATION EINES CA MITTELS XML.....</b>	<b>6</b>
3.1. VORBEMERKUNGEN ZU XML .....	6
3.2. FESTLEGUNG DES INITIALZUSTANDES .....	6
3.3. GAME OF LIFE .....	8
3.4. DIE ZUSTANDSÜBERGANGSFUNKTION.....	12
3.5. KLASSIFIZIERUNG DER ZELLATTRIBUTWERTE .....	15
3.6. GRÖÖE DES RASTERS.....	17
<b>4. WEITERE ZELLULARE AUTOMATEN.....</b>	<b>18</b>
4.1. DER AUTOMAT VON GREENBERG UND HASTINGS .....	18
4.2. EIN CA-MODELL FÜR PFLANZENPOPULATIONEN.....	23
4.3. VERKEHRSSIMULATION .....	30
<b>5. ANHANG .....</b>	<b>39</b>
5.1. ABSPIELEN DER FLASH-DATEIEN .....	39
5.2. DIE XML-SPEZIFIKATION DES CA .....	39
5.3. LOGISCHE UND ARITHMETISCHE AUSDRÜCKE IN DEN WENN-DANN-REGELN .....	41
5.4. PERFORMANCEPROBLEME .....	43
<b>LITERATURVERZEICHNIS .....</b>	<b>44</b>

## 1. Einleitung

Mit zellularen Automaten (cellular automata – CA) können zeitabhängige, diskrete Prozesse simuliert werden. Sie basieren auf einer Idee von John von Neumann in den 1940er Jahren, um das Verhalten komplexer Systeme zu untersuchen. Im Bereich der quantitativen Geographie und der Geosimulation ist ihre Einsatzmöglichkeit sehr vielfältig: Simulation von Waldbränden, Analyse von Pflanzenpopulationsentwicklungen, Stadtausbreitungsmodelle etc.

Ein CA wird durch folgende Eigenschaften charakterisiert (vgl. RIEDL99):

- Ein CA besteht aus einem Raster von nicht überlappenden Zellen.
- Jede Zelle hat einen Zustand. Die Menge aller möglichen Zustände ist endlich.
- Die Zustandsänderung der Rasterzellen erfolgt in diskreten Zeitschritten  $t = 0, 1, 2, \dots$
- Der neue Zustand einer Zelle hängt in der Regel auch von deren Nachbarzellen ab.

Im Folgenden soll ein allgemeiner CA in einer Flash-Umgebung entwickelt werden. Die Spezifikation der Größe, des Initialzustandes, der Zustandsübergangsfunktion sowie der Farbsignatur erfolgt mittels XML.

## 2. Aufbau eines zellularen Automaten in einer Flash 5-Umgebung

### 2.1. Datenstruktur

Jede Zelle des CA wird durch ein aus Vektoren zusammengesetztes Quadrat repräsentiert. Die Quadrate sind jeweils Objekte vom Typ *MovieClip*. Der Aufbau des Rasters erfolgt automatisch: Ein Quadrat wird manuell angelegt und im Programm als Vorlage für jede Rasterzelle kopiert. Im folgenden Beispiel hat das manuell angelegte Quadrat den Namen *Z00*. Mit der Methode *duplicateMovieClip* wird *Z00* kopiert und die (x,y)-Position entsprechend verändert. Die neue Rasterzelle bekommt den Namen *XjYi* (i ... Zeile bzw. y-Koordinate, j ... Spalte bzw. x-Koordinate). Z. B. hat die Zelle mit der Position  $x = 10$  und  $y = 3$  die Bezeichnung *X10Y3*. Im zweidimensionalen Array *raster* werden korrespondierend dazu die Attributwerte der Zellen hinterlegt. In diesem Beispiel sind es Farben, die per Zufallsgenerator bestimmt werden.

Zur besseren Darstellung ist dem Raster ein Gitternetz unterlegt. Dieses Netz wird ebenso mit der Methode *duplicateMovieClip* erzeugt. Für die horizontalen Gitterlinien wird die Muster-

linie L0 kopiert und in Länge und Startposition geändert. Zur Erzeugung der vertikalen Gitterlinien wird sie noch zusätzlich um 90 Grad gedreht.

### ActionScript-Code zum Aufbau eines Rasters:

```
var zeilanz = 38; // Anzahl Zeilen
var spalanz = 50; // Anzahl Spalten
var cellsize = 10; // Zellgröße
var zelle; // Name der Zelle
var linie; // Name der Gitterlinie
var raster = new Array (zeilanz); // Raster
var depth = 1;
var ursprung = new Object ();

// Transformation in das Hauptkoordinatensystem
ursprung.x = 0;
ursprung.y = 0;
L0._width = cellsize;
L0._x = 0;
L0._y = 0;
L0.localToGlobal (ursprung);
Z00._height = cellsize;
Z00._width = cellsize;
Z00._x = 0;
Z00._y = 0;
Z00.localToGlobal (ursprung);

// Generierung der Rasterzellen
for (var i = 0; i < zeilanz; i++) {

    raster [i] = new Array (spalanz); // Neue Zeile

    for (var j = 0; j < spalanz; j++) {
        depth++;
        zelle = "X" + j + "Y" + i;
        Z00.duplicateMovieClip( zelle, depth );
        setProperty ( zelle, _x,
                      Z00._width * (j + 1) + Z00._x );
        setProperty ( zelle, _y,
                      Z00._width * (i + 1) + Z00._y );
        raster [i] [j] = new Color (zelle); // Attribut der Zelle
    }
}

// Generierung des Gitters
// - horizontal
for (i = 0; i <= zeilanz; i++) {
    depth++;
    linie = "H" + i;
    L0.duplicateMovieClip( linie, depth);
    setProperty ( linie, _x, Z00._x + Z00._width );
    setProperty ( linie, _y, Z00._width * (i+1) + Z00._y );
    setProperty ( linie, _width, Z00._width * spalanz );
}

// - vertikal
for (i = 0; i <= spalanz; i++) {
    depth++;
    linie = "V" + i;
    L0.duplicateMovieClip( linie, depth);
    setProperty ( linie, _x, Z00._width * (i+1) + Z00._x );
    setProperty ( linie, _y, Z00._y + Z00._width );
    setProperty ( linie, _width, Z00._width * zeilanz );
    setProperty ( linie, _rotation, 90 );
}

Z00.unloadMovie (); // werden nicht mehr benötigt
L0.unloadMovie ();

// Zufallsbestimmte Färbung der Zelle
for (var i = 0; i < zeilanz; i++) {
    for (var j = 0; j < spalanz; j++) {
```

```

        raster [i] [j].setRGB(Math.floor(Math.random() *
                                0xFFFFFFFF));
    }
}

```

In Abbildung 2-1 ist ein solchermaßen generiertes Raster dargestellt.

## 2.2. Ein einfacher zellularer Automat

Ausgehend von dem Raster aus Kap. 2.1. wird jetzt eine Simulation durchgeführt: Jedes Zellattribut, d.h. jede Farbe einer Zelle, wird bei einem Simulationsschritt zufallsabhängig festgelegt. Die Abfolge der Schritte in Form einer Endlosschleife sieht folgendermaßen aus:

- 1) Aufbau des Rasters und des Gitternetzes.
- 2) Zufallsabhängige Veränderung der Zellen.
- 3) Gehe zu Schritt 2.

Jede Flash-Animation setzt sich aus einzelnen, durchnummerierten Bildern zusammen, die sequenziell abgespielt werden. Ein Bild besteht aus einer oder mehreren wie transparente Blätter übereinanderliegenden Ebenen. Jede Ebene kann Grafiken, Töne, Filmsequenzen oder ActionScript-Programme beinhalten. Üblicherweise werden die Programme in einer Ebene konzentriert. Die durchzuführende Flash-Simulation besteht aus zwei Ebenen: Eine (*Scripts*) beinhaltet die Programme, und die andere (*Layer 2*) wird nur im ersten Bild verwendet (siehe Abbildung 2-2). Sie enthält das Quadrat *Z00*, das als Kopiervorlage für die Rasterzellen fungiert, sowie die Linie *L0* als Vorlage für die Gitterlinien. In den anderen Bildern ist diese Ebene leer.

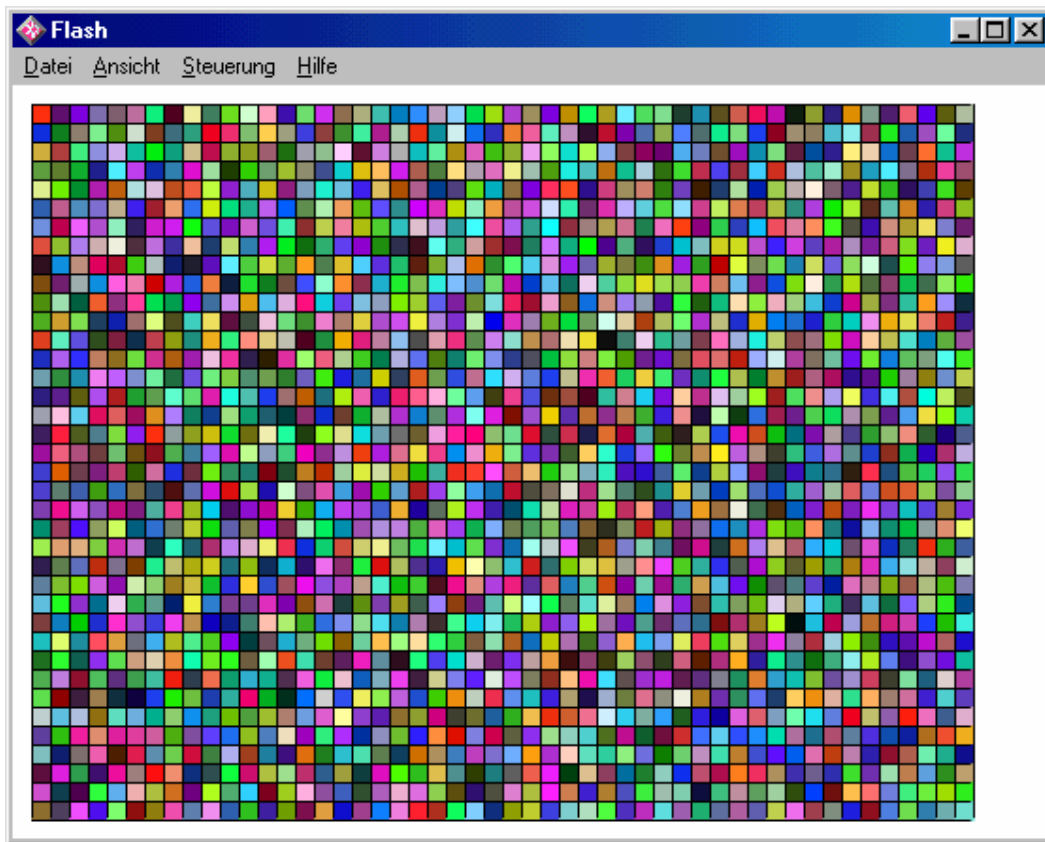
Jedem der o.a. Schritte ist ein Bild zugeordnet. Das erste Bild enthält das Programm, das das Raster aufbaut:

```

var zeilanz = 38; // Anzahl Zeilen
var spalanz = 50; // Anzahl Spalten
var cellsize = 10; // Zellgröße
var zelle; // Name der Zelle
var linie; // Name der Gitterlinie
var raster = new Array (zeilanz); // Raster
var depth = 1;
var ursprung = new Object ();
// Transformation in das Hauptkoordinatensystem
ursprung.x = 0;
ursprung.y = 0;
L0._width = cellsize;
L0._x = 0;
L0._y = 0;
L0.localToGlobal (ursprung);
Z00._height = cellsize;
Z00._width = cellsize;
Z00._x = 0;
Z00._y = 0;
Z00.localToGlobal (ursprung);

```

Abbildung 2-1: Zufallsbestimmtes Raster



```
// Generierung der Rasterzellen
for (var i = 0; i < zeilanz; i++) {

    raster [i] = new Array (spalanz); // Neue Zeile

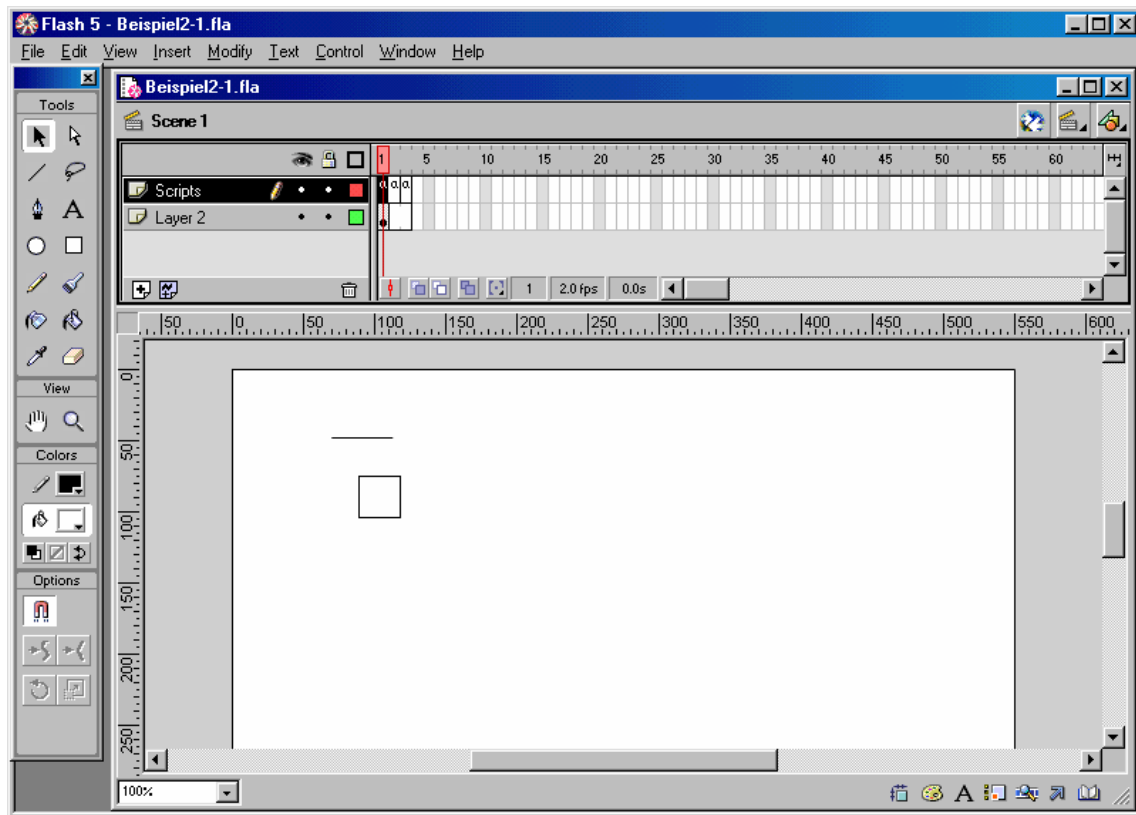
    for (var j = 0; j < spalanz; j++) {
        depth++;
        zelle = "X" + j + "Y" + i;
        Z00.duplicateMovieClip( zelle, depth );
        setProperty ( zelle, _x,
            Z00._width * (j + 1) + Z00._x );
        setProperty ( zelle, _y,
            Z00._width * (i + 1) + Z00._y );
        raster [i] [j] = new Color (zelle); // Attribut der Zelle
    }

}

// Generierung des Gitters
// - horizontal
for (i = 0; i <= zeilanz; i++) {
    depth++;
    linie = "H" + i;
    L0.duplicateMovieClip( linie, depth);
    setProperty ( linie, _x, Z00._x + Z00._width );
    setProperty ( linie, _y, Z00._width * (i+1) + Z00._y );
    setProperty ( linie, _width, Z00._width * spalanz );
}

// - vertikal
for (i = 0; i <= spalanz; i++) {
    depth++;
    linie = "V" + i;
    L0.duplicateMovieClip( linie, depth);
    setProperty ( linie, _x, Z00._width * (i+1) + Z00._x );
    setProperty ( linie, _y, Z00._y + Z00._width );
}
```

Abbildung 2-2: Die zwei Ebenen der durchzuführenden Flash-Simulation



```

setProperty ( linie, _width, Z00.width * zeilanz );
setProperty ( linie, _rotation, 90 );
}

```

```

Z00.unloadMovie (); // werden nicht mehr benötigt
L0.unloadMovie ();

```

Das Programm des zweiten Bildes füllt die Zellen zufallsabhängig:

```

// Zufallsbestimmte Färbung der Zelle
for (var i = 0; i < zeilanz; i++) {
  for (var j = 0; j < spalanz; j++) {
    raster [i] [j].setRGB(Math.floor(Math.random() *
      0xFFFFFF));
  }
}

```

Auf dem dritten Bild wird der Sprung zu Bild 2 durchgeführt:

```
gotoAndplay (2);
```

So wird ein einfacher CA simuliert, bei dem sich die einzelnen Zellen farblich auf Grund von Zufallszahlen in einer Endlosschleife verändern. Zwei wichtige Bestandteile eines CA, der Initialzustand und die Zustandsübergangsfunktion, werden hier nur implizit mit Hilfe des Zufallszahlengenerators festgelegt. Im folgenden Abschnitt sollen beide Bestandteile mittels XML festgelegt werden.

Die Flash-Datei zu diesem Kapitel ist auf der beiliegenden CD unter *Beispiel2* zu finden.

### 3. Die Spezifikation eines CA mittels XML

#### 3.1. Vorbemerkungen zu XML

XML (Extended Markup Language) wurde vom World Wide Web Consortium (W3C) herausgegeben und ist ein Standard, der sich für den Austausch von Informationen aller Art eignet. Inhalt und Bedeutung der Informationen sind im Gegensatz zu HTML von der Darstellung getrennt.

Da ActionScript eine eigene Klasse für XML-Dokumente aufweist und das Document Object Model (DOM) unterstützt, wurde der Weg gewählt, einen CA mit Hilfe eines XML-Dokuments zu spezifizieren. In den folgenden Kapiteln wird eine mögliche XML-Struktur hierfür entwickelt.

#### 3.2. Festlegung des Initialzustandes

Der Aufbau des XML-Dokuments sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="windows-1252"?>
<CA>
  <NAME>Beispiel 3-1</NAME>
  <INITIAL_STATE>
    <CELL><X>0</X> <Y>0</Y> <VAL>1</VAL></CELL>
    <CELL><X>0</X> <Y>1</Y> <VAL>1</VAL></CELL>
    <CELL><X>0</X> <Y>2</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>0</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>1</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>2</Y> <VAL>1</VAL></CELL>
  </INITIAL_STATE>
</CA>
```

Das Dokument beginnt mit einem Knoten CA. Dieser umfasst je einen Knoten für den Namen des CA (NAME) sowie für den Initialzustand (INITIAL\_STATE). Der Initialzustand beinhaltet mehrere Knoten für Zellen (CELL). Je Zelle werden die x- und y-Koordinaten sowie der Attributwert (VAL) festgelegt. Alle Zellen, die in dem XML-Dokument nicht spezifiziert sind, erhalten 0 als Initialwert.

Das Erzeugen eines XML-Objektes erfolgt in ActionScript folgendermaßen:

```
var CADoc = new XML ();
```

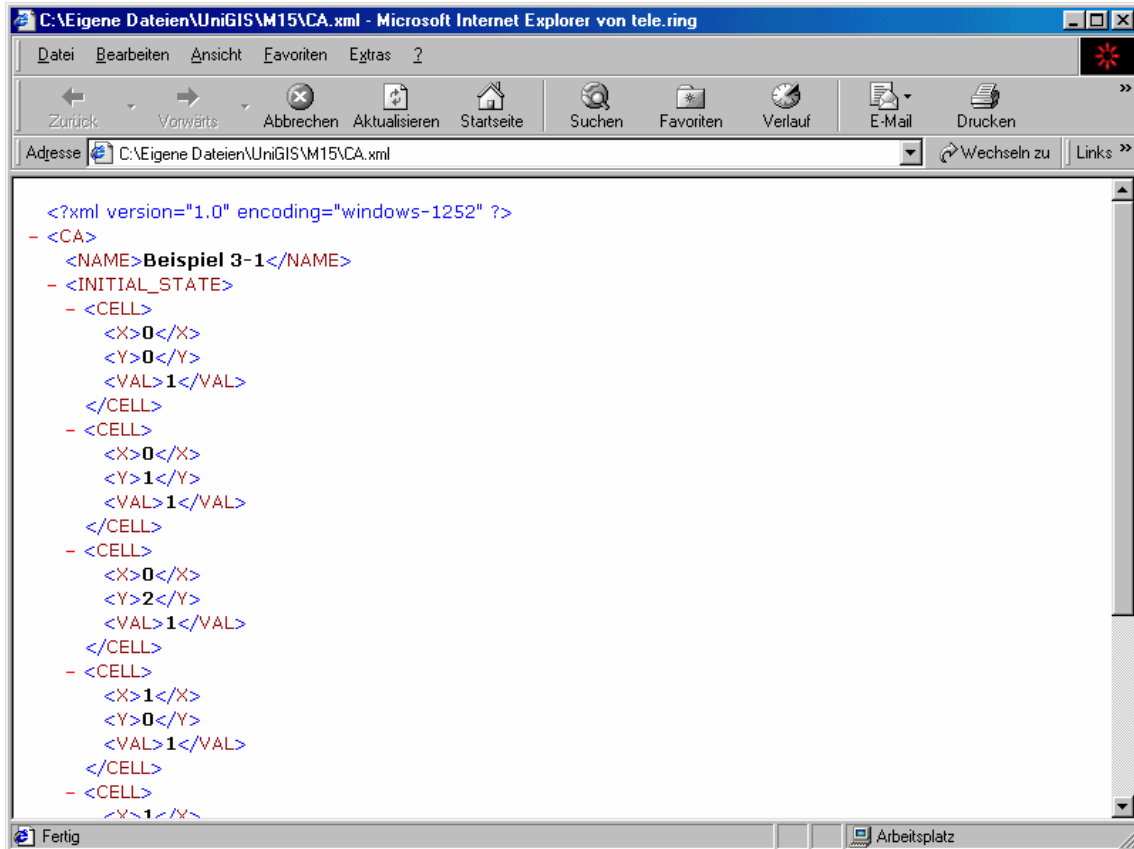
Diesem Objekt muss aber noch die XML-Datei mit dem Initialzustand zugewiesen werden.

Sie hat den Namen CA.xml und liegt im selben Verzeichnis wie die Flash-Datei. Die Anwei-

sung lautet:

```
CADoc.load (_url.substring (0, _url.lastIndexOf ("/")) + "/CA.xml");
```

Abbildung 3-1: Darstellung des XML-Files im Internet-Explorer



*url* beinhaltet den Pfad und Namen der Flash-Datei. Mit der String-Methode *substring* wird der Pfad extrahiert und mit dem Namen der zu ladenden XML-Datei verkettet. Bevor die XML-Datei verarbeitet werden kann, muss sichergestellt sein, dass der Ladevorgang beendet ist:

```
if (!CADoc.loaded) {
  // Warten bis CA.xml geladen
}

else {
  // CA.xml verarbeiten
  ....
}
```

Dann ist in dem XML-Baum der Knoten CA zu suchen und der Variablen *CANode* zuzuweisen:

```
// Suchen des Knotens CA
i = 0;
CANode = CADoc.childNodes [i];
while (i < CADoc.childNodes.length &&
  CANode.nodeName != "CA") {
  CANode = CADoc.childNodes [i];
  i++;
}
```

```
}
```

Die Methode *childNodes* liefert ein Array, das alle Unterknoten beinhaltet. In diesem Fall handelt es sich um die Knoten auf der ersten Ebene. Ist der Knoten CA gefunden, dann kann dessen Unterknoten INITIAL\_STATE gesucht werden:

```
// Suchen des Knotens INITIAL_STATE (Unterknoten von CA)
i = 0;
IStatNode = CANode.childNodes [i];
while (i < CANode.childNodes.length &&
      IStatNode.nodeName != "INITIAL_STATE") {
    IStatNode = CANode.childNodes [i];
    i++;
}
```

Der Knoten INITIAL\_STATE umfasst eine variable Anzahl von Unterknoten des Typs CELL, die zu einer (x,y)-Position den Initialwert angeben. Beim Verarbeiten der CELL-Knoten werden jeweils der x-, y- und Attributwert extrahiert. Dem zweidimensionalen Array *raster*, das die Attributwerte jeder Zelle beinhaltet, wird in diesem Beispiel das Farbattribut für schwarz zugewiesen, falls der Attributwert ungleich 0 ist:

```
// Verarbeiten der CELL-Knoten
if (IStatNode != null) {
    for (var i = 0; i < IStatNode.childNodes.length; i++) {
        cellNode = IStatNode.childNodes [i];
        if (cellNode.nodeName != "CELL") {continue;
        }
        // Extrahieren von x, y und dem Attributwert der Zelle
        for (var j = 0; j < cellNode.childNodes.length; j++) {
            if (cellNode.childNodes [j].nodeName == "X") {
                x = cellNode.childNodes [j].childNodes.toString ();
            }
            else if (cellNode.childNodes [j].nodeName == "Y") {
                y = cellNode.childNodes [j].childNodes.toString ();
            }
            else if (cellNode.childNodes [j].nodeName == "VAL") {
                val = cellNode.childNodes [j].childNodes.toString ();
            }
        }
        if (val != 0) {
            raster [y] [x].setRGB(0x000000); // auf schwarz setzen
        }
    }
}
```

In der Beispiel-XML-Datei sind die Zellen mit den Positionen (0,0), (0,1), (0,2), (1,0), (1,1) und (1,2) mit 1 (schwarz) zu initialisieren. In Abbildung 3-2 ist dieser Anfangszustand dargestellt. In Flash ist der Ursprung links oben. Die y-Achse verläuft von oben nach unten.

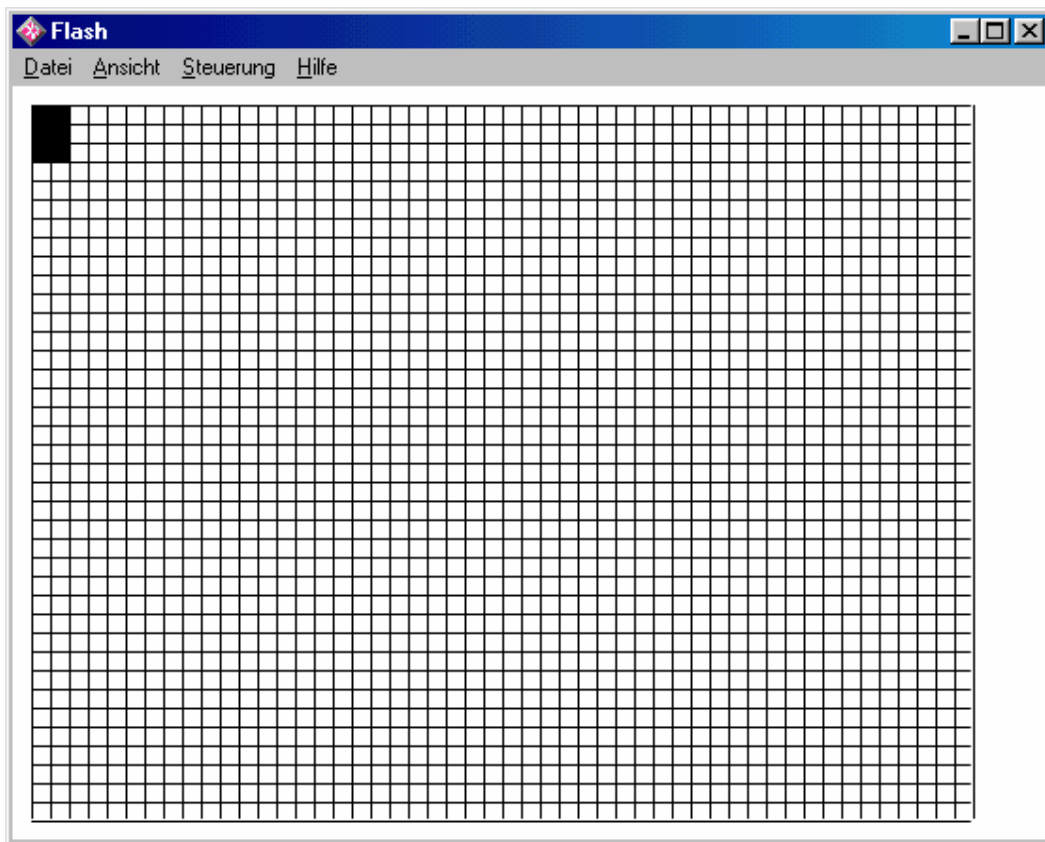
### 3.3. Game of Life

Das Game of Life (vgl. RIEDL99) ist ein klassisches CA, bei dem die Zellen die Zustände tot (Farbe weiß) oder lebend (Farbe schwarz) haben. Zu jeder Zelle wird die Zahl der lebenden unter den 8 Nachbarzellen zum Zeitpunkt t bestimmt. Bei 2 oder 3 lebenden Nachbarn bleibt eine bereits lebende Zelle auch zum Zeitpunkt t+1 in diesem Zustand. Eine tote Zelle wird zum Zeitpunkt t+1 lebendig, wenn sie genau 3 lebende Nachbarn hat.

Der Initialzustand wird wie in Kapitel 3.2 eingelesen. Die Zustandsübergangsfunktion wird ausprogrammiert. Hierbei werden zunächst die Attributwerte der Zellen, die im Array *raster* gespeichert sind, auf das Array *raster\_alt* kopiert:

```
// raster auf raster_alt kopieren
for (i = 0; i < zeilanz; i++) {
    for (j= 0; j < spalanz; j++) {
        raster_alt [i] [j] = raster [i] [j].getRGB ();
    }
}
```

Abbildung 3-2: Initialisiertes CA



Für die Bestimmung des Zustandes in  $t+1$  wird für jede Zelle die Zahl der lebenden Nachbarzellen bestimmt und das Farbattribut entsprechend gesetzt:

```
for (i = 0; i < zeilanz; i++) {
    for (j = 0; j < spalanz; j++) {

        // Lebende Nachbarzellen zählen
        lebend = 0;
        for (k = i - 1; k <= i + 1 && lebend <= 4; k++) {
            for (l = j - 1; l <= j + 1 && lebend <= 4; l++) {
                if (k >= 0 && k < zeilanz && l >= 0 && l < spalanz &&
                    !(k == i && l == j) && raster_alt [k] [l] == 0x000000) {
                    lebend++;
                }
            }
        }
    }
}
```

```

raster [i] [j].setRGB(0xFFFFFFFF); // mit weiß = tote Zelle vorbelegen
// Zellen beleben
if (raster_alt [i] [j] == 0x000000 && lebend >= 2 && lebend <= 3) {
    raster [i] [j].setRGB(0x000000);
}
else if (raster_alt [i] [j] == 0xFFFFFFFF && lebend == 3) {
    raster [i] [j].setRGB(0x000000);
}
}
}

```

Aus Performancegründen wird bei 4 lebenden Nachbarn die weitere Suche abgebrochen, da zur Entscheidungsfindung genau 3 bzw. 2 bis 3 lebende Nachbarzellen erforderlich sind.

Als Beispiel wird ein Game of Life mit 10 x 10 Zellen ausgeführt. Die XML-Datei für den Anfangszustand hat folgendes Aussehen:

```

<?xml version="1.0" encoding="windows-1252"?>
<CA>
  <NAME>Beispiel 3-1</NAME>
  <INITIAL_STATE>
    <CELL><X>0</X> <Y>0</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>1</Y> <VAL>1</VAL></CELL>
    <CELL><X>0</X> <Y>2</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>2</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>1</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>2</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>2</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>4</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>3</Y> <VAL>1</VAL></CELL>
    <CELL><X>4</X> <Y>3</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>3</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>6</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>4</Y> <VAL>1</VAL></CELL>
    <CELL><X>4</X> <Y>4</Y> <VAL>1</VAL></CELL>
    <CELL><X>2</X> <Y>7</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>8</Y> <VAL>1</VAL></CELL>
    <CELL><X>2</X> <Y>8</Y> <VAL>1</VAL></CELL>
    <CELL><X>4</X> <Y>6</Y> <VAL>1</VAL></CELL>
    <CELL><X>4</X> <Y>7</Y> <VAL>1</VAL></CELL>
    <CELL><X>4</X> <Y>8</Y> <VAL>1</VAL></CELL>
    <CELL><X>5</X> <Y>8</Y> <VAL>1</VAL></CELL>
    <CELL><X>7</X> <Y>8</Y> <VAL>1</VAL></CELL>
    <CELL><X>8</X> <Y>8</Y> <VAL>1</VAL></CELL>
    <CELL><X>7</X> <Y>6</Y> <VAL>1</VAL></CELL>
    <CELL><X>8</X> <Y>6</Y> <VAL>1</VAL></CELL>
    <CELL><X>8</X> <Y>3</Y> <VAL>1</VAL></CELL>
    <CELL><X>8</X> <Y>4</Y> <VAL>1</VAL></CELL>
  </INITIAL_STATE>
</CA>

```

Abbildung 3-3: Der Anfangszustand

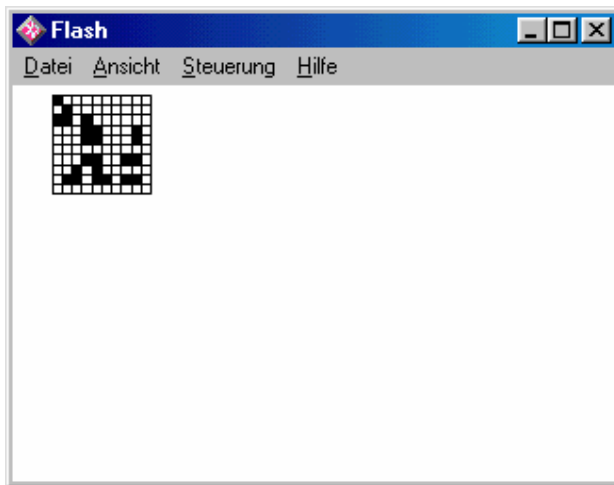


Abbildung 3-4: Der zweite Schritt

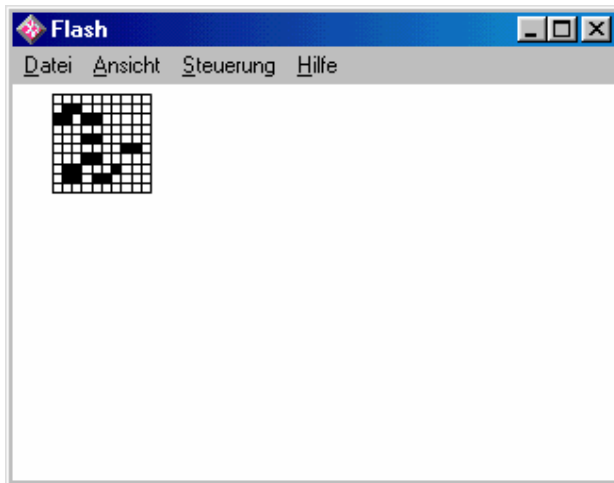


Abbildung 3-5: Der dritte Schritt

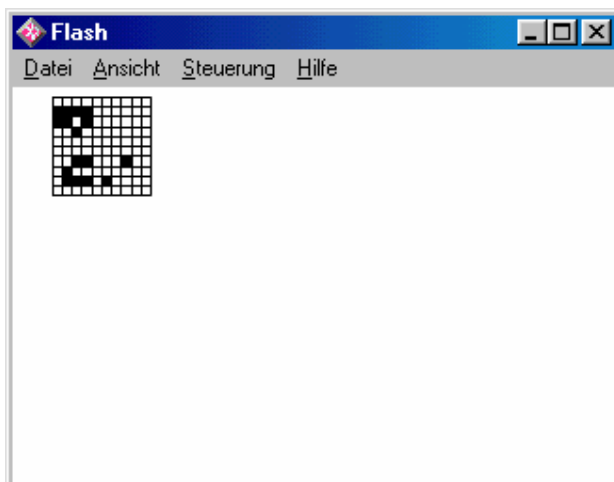


Abbildung 3-6: Der vierte Schritt

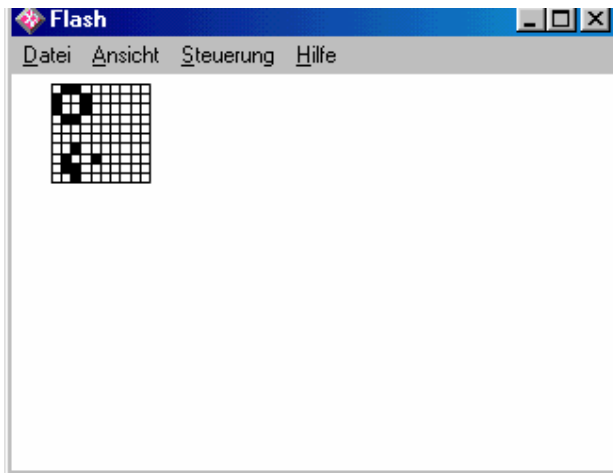
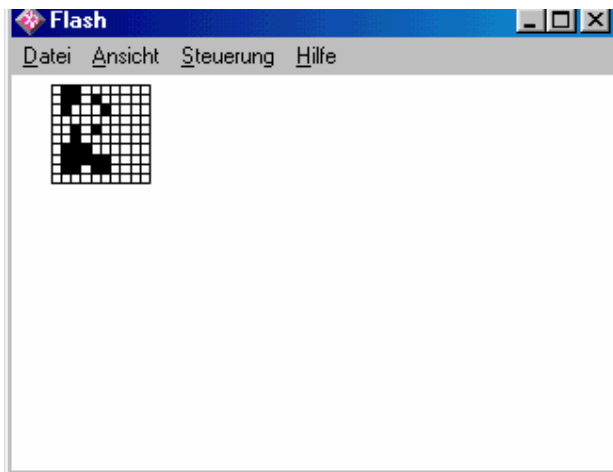


Abbildung 3-7: Nach einigen weiteren Schritten



Die Flash-Datei und das zugehörige XML-File sind auf der beiliegenden CD unter *Beispiel3-1* zu finden. Allfällige Performanceprobleme werden im Kap. 5.4. erläutert.

### 3.4. Die Zustandsübergangsfunktion

Mit Hilfe der Zustandsübergangsfunktion wird der Zustand der Zelle  $(i,j)$  zum Zeitpunkt  $t+1$  bestimmt. In der Regel ergibt sich dieser aus den Zuständen benachbarter Zellen zum Zeitpunkt  $t$ .

Die Zustandsübergangsfunktion wird mittels Wenn-Dann-Regeln spezifiziert. Z.B. beim Game of Life:

WENN  $z(i,j) = \text{lebend}$  &  $(i,j)$  hat 2 oder 3 lebende Nachbarn DANN  $z(i,j) = \text{lebend}$

WENN  $z(i,j) = \text{tot}$  &  $(i,j)$  hat 3 lebende Nachbarn DANN  $z(i,j) = \text{lebend}$

SONST  $z(i,j) = \text{tot}$

Für jede Zelle  $(i,j)$  werden diese Regeln der Reihe nach angewandt bis eine Bedingung zutrifft. Im zugehörigen DANN-Teil wird der Zellzustand zum Zeitpunkt  $t+1$  angegeben. Falls keine Bedingung zutrifft wird der SONST-Teil ausgeführt.

Die Definition der Zustandsübergangsfunktion in XML erfolgt anhand des folgenden Schemas:

```
...
</INITIAL_STATE>
<TRANSITION_RULES>
  <RULE>
    <IF> ..... </IF>
    <THEN> .... </THEN>
  </RULE>
  <RULE>
    <IF> ..... </IF>
    <THEN> .... </THEN>
  </RULE>
  ...
</TRANSITION_RULES>
```

Die Zustandsübergangsfunktion beim Game of Life hat dann folgendes Aussehen:

```
...
<TRANSITION_RULES>
  <RULE>
    <IF> ( ( Z (I-1, J-1) + Z (I-1, J) + Z (I-1, J+1) +
            Z (I, J-1) + Z (I, J) + Z (I, J+1) +
            Z (I+1, J-1) + Z (I+1, J) + Z (I+1, J+1) ) = 2 |
            ( Z (I-1, J-1) + Z (I-1, J) + Z (I-1, J+1) +
            Z (I, J-1) + Z (I, J) + Z (I, J+1) +
            Z (I+1, J-1) + Z (I+1, J) + Z (I+1, J+1) ) = 3 ) &
            Z (I, J) = 1 </IF>
    <THEN> 1 </THEN>
  </RULE>
  <RULE>
    <IF> ( ( Z (I-1, J-1) + Z (I-1, J) + Z (I-1, J+1) +
            Z (I, J-1) + Z (I, J) + Z (I, J+1) +
            Z (I+1, J-1) + Z (I+1, J) + Z (I+1, J+1) ) = 3 ) &
            Z (I, J) = 0 </IF>
    <THEN> 1 </THEN>
  </RULE>
  <RULE>
    <IF> </IF>
    <THEN> 0 </THEN>
  </RULE>
</TRANSITION_RULES>
```

Mit den vordefinierten Variablen I und J werden die Zeile (y-Position) bzw. die Spalte (x-Position) der aktuellen Zelle angegeben. Mit Z wird die aktuelle Zustandsmatrix bezeichnet.  $Z(I+1, J+1)$  spezifiziert also den Zustand des „südöstlichen“ Nachbarn. Die Zustände einer einzelnen Zelle sind entweder 1 (lebend) oder 0 (tot). In der zweiten Regel werden beispielsweise die Zustände aller acht Nachbarn addiert. Die Summe muss 3 sein, d.h. es gibt genau 3 lebende Nachbarn. Weiters muss der Zustand der aktuellen Zelle gleich 0 (tot) sein. Wenn diese Bedingung zutrifft, erhält die aktuelle Zelle den Zustand 1 (lebend) gemäß des zugehörigen THEN-Teils, d.h.  $Z(I, J) := 1$ .

Die Regeln werden der Reihe nach solange abgearbeitet bis eine Bedingung zutrifft. Der in der zugehörigen THEN-Klausel spezifizierte Ausdruck ergibt dann den Zustand der Zelle zum nächsten Zeitpunkt. Die dritte Regel hat eine leere Bedingung. Eine solche Bedingung ist immer wahr. Damit wird der o.a. SONST-Fall realisiert. Falls also die Bedingungen der ersten beiden Regeln falsch sind, wird die Zelle gemäß des THEN-Teils der Regel 3 auf 0 (tot) gesetzt.

Die arithmetischen Ausdrücke können neben den Grundrechnungsarten (+, -, (), /, \*) auch gängige Funktionen wie Quadratwurzel (SQRT), natürlicher Logarithmus (LOG), Sinus (SIN) etc. verwenden. Die Funktion RANDOM liefert gleichverteilte Zufallszahlen zwischen 0 und 1. Dadurch können komplexere Ausdrücke wie z.B.  $3 * \text{SQRT} ( \text{LOG} ( Z (\text{RANDOM} * 10, J) ) )$  gebildet werden. Ein Verzeichnis der möglichen Funktionen ist im Kap. 5.3. angegeben.

Die logischen Ausdrücke verwenden die gängigen Vergleichssymbole (<, <=, >, >=, =, !=) und können mittels „&“ (logisches Und) und „|“ (logisches Oder) verknüpft bzw. mittels „^“ (logisches Nicht) negiert werden.

Die Analyse und Auswertung erfolgen mit einem in ActionScript programmierten Formelinterpreter. Hierbei werden die im Compilerbau gebräuchlichen Methoden der attributierten Grammatik (vgl. RECHENBERG85) und des rekursiven Abstiegs (vgl. WIRTH84) verwendet.

Die Zustandsübergangsfunktion beim Game of Life kann unter Zuhilfenahme der Funktion FOCSE (focal sum eight), die die Summe der Attributwerte der acht Nachbarn liefert, folgendermaßen definiert werden:

```
<TRANSITION_RULES>
  <RULE>
    <IF> FOCSE < 2 </IF>
    <THEN> 0 </THEN>
  </RULE>
  <RULE>
    <IF> FOCSE = 2 & Z (I,J) = 1 </IF>
    <THEN> 1 </THEN>
  </RULE>
  <RULE>
    <IF> FOCSE = 3 & Z (I,J) = 1 </IF>
    <THEN> 1 </THEN>
  </RULE>
  <RULE>
    <IF> FOCSE = 3 & Z (I,J) = 0 </IF>
    <THEN> 1 </THEN>
  </RULE>
  <RULE>
    <IF>
      </IF>
```

```

    <THEN> 0 </THEN>
  </RULE>
</TRANSITION_RULES>

```

Bedingt durch die XML-Syntax steht „&lt;“ für das Symbol „<“ und „&amp;“ für „&“. Die erste Regel dient der Beschleunigung der Abarbeitung: Da die meisten Zellen häufig weniger als zwei lebende Nachbarn haben, trifft die erste Regel bereits zu und die weitere Abarbeitung entfällt.

Das Suchen des Knotens TRANSITION\_RULES, der ein Nachbarknoten von INITIAL\_STATE ist, erfolgt in folgender Weise:

```

// Suchen des Knotens TRANSITION_RULES (Unterknoten von CA)
i = 0;
TrRNode = CANode.childNodes [i];
while (i < CANode.childNodes.length &&
      TrRNode.nodeName != "TRANSITION_RULES") {
  TrRNode = CANode.childNodes [i];
  i++;
}

```

Der Knoten TRANSITION\_RULES umfasst eine variable Anzahl von Unterknoten des Typs RULE, die jeweils aus einem IF- und einem THEN-Teil bestehen. Beide Teile werden als Zeichenketten in separaten Arrays gespeichert:

```

// Verarbeiten der RULE-Knoten
if (TrRNode != null) {
  for (var i = 0; i < TrRNode.childNodes.length; i++) {
    ruleNode = TrRNode.childNodes [i];
    if (ruleNode.nodeName != "RULE") {continue;}
  }
  // Extrahieren von IF und THEN
  for (var j = 0; j < ruleNode.childNodes.length; j++) {
    if (ruleNode.childNodes [j].nodeName == "IF") {
      ifpart.push (ruleNode.childNodes [j].childNodes.toString ());
    }
    else if (ruleNode.childNodes [j].nodeName == "THEN") {
      thenpart.push (ruleNode.childNodes [j].childNodes.toString ());
    }
  }
}
}

```

Die Flash-Datei und das zugehörige XML-File zu diesem Beispiel sind auf der beiliegenden CD unter *Beispiel3-2* zu finden.

### 3.5. Klassifizierung der Zellattributwerte

Die Attributwerte der Rasterzellen werden durch Farbflächen signiert. Die Werte werden in aufeinander folgende Klassen eingeteilt, denen eine bestimmte Farbe zugeordnet wird. Für die Definition der Zuordnung wird folgendes XML-Schema gewählt:

```

<SCALE>
  <VAL> 0 </VAL>   <COL> FFFFFFFF </COL>
  <VAL> 1 </VAL>   <COL> 000000 </COL>
</SCALE>

```

Der Knoten SCALE ist ein Nachbar von INITIAL\_STATE und TRANSITION\_RULES. Im Unterknoten VAL wird der Anfangswert der Klasse angegeben. COL enthält den zugeordneten Farbcode. In diesem Beispiel gibt es zwei Klassen. Allen Werten größer oder gleich 0 und kleiner 1 wird die Farbe FFFFFFFF (weiß) und allen Werten größer oder gleich 1 die Farbe 000000 (schwarz) zugewiesen. Ein nicht klassifizierter Wert erhält immer die Farbe weiß.

Der Farbcode wird hexadezimal in der in ActionScript gebräuchlichen Form RRGGBB angegeben. RR, GG und BB sind 1-Byte-Zahlen zwischen 0 und 255 (hexadezimal zwischen 00 und FF) und repräsentieren den Rot-, Grün- bzw. Blauanteil. Z.B. FF0000 ist rot, FFFF00 ist gelb, 0000FF ist blau, 000000 ist schwarz und FFFFFFFF ist weiß.

Das Suchen des SCALE-Knotens erfolgt in folgender Weise:

```
// Suchen des Knotens SCALE (Unterknoten von CA)
i = 0;
ScNode = CANode.childNodes [i];
while (i < CANode.childNodes.length &&
      ScNode.nodeName != "SCALE") {
    ScNode = CANode.childNodes [i];
    i++;
}

```

Die VAL- und COL-Knotenwerte werden in den Tabellen *valtab* und *coltab* abgelegt:

```
if (ScNode != null) {
    // Extrahieren von VAL und COL
    for (var i = 0; i < ScNode.childNodes.length; i++) {
        if (ScNode.childNodes [i].nodeName == "VAL") {
            var val = parseFloat (ScNode.childNodes [i].childNodes.toString ());
            valtab.push (0);
            coltab.push (0xFFFFFFFF);
            for (j = valtab.length - 2; j >= 0 && valtab [j] > val; j--) {
                valtab [j + 1] = valtab [j];
                coltab [j + 1] = coltab [j];
            }
            valtab [j + 1] = val;
        }
        else if (ScNode.childNodes [i].nodeName == "COL") {
            var col = parseInt (ScNode.childNodes [i].childNodes.toString (), 16);
            coltab [j + 1] = col;
        }
    }
}

```

Durch diesen Aufbau sind beide Tabellen nach dem VAL-Knotenwert sortiert. Die Ermittlung einer Farbe zu einem Wert erfolgt mit der Funktion *ColZuord*:

```
// Zuordnung der Farbe zu einem Wert bestimmen
function ColZuord (pval) {

    var farbw = 0xFFFFFFFF; // weiß als Default

    for (var k = 0; k < valtab.length && valtab [k] <= pval; k++) {
    }
    if (k > 0) {
        farbw = coltab [k - 1];
    }
    return farbw;
}

```

Die Farben der Zellen werden – korrespondierend zum Attributwertarray *raster* – in dem Array *farbe* gespeichert.

Die Flash-Datei und das zugehörige XML-File sind auf der beiliegenden CD unter *Beispiel3-3* zu finden.

### 3.6. Größe des Rasters

Die Größe wird durch die Zahl der Zeilen und Spalten spezifiziert. Das zugehörige XML-Schema lautet:

```
<SIZE>
  <ROWS> 10 </ROWS>
  <COLS> 10 </COLS>
</SIZE>
```

In diesem Beispiel wird ein Raster der Größe 10x10 festgelegt. Der SIZE-Knoten ist ein Nachbar von INITIAL\_STATE, TRANSITION\_RULES und SCALE. Das Suchen des Knotens erfolgt in ähnlicher Weise wie bei den anderen:

```
// Suchen des Knotens SIZE (Unterknoten von CA)
i = 0;
SzNode = CANode.childNodes [i];
while (i < CANode.childNodes.length &&
      SzNode.nodeName != "SIZE") {
  SzNode = CANode.childNodes [i];
  i++;
}

if (SzNode != null) {
  // Extrahieren von ROWS und COLS
  for (var i = 0; i < SzNode.childNodes.length; i++) {
    if (SzNode.childNodes [i].nodeName == "ROWS") {
      zeilanz = parseInt (SzNode.childNodes [i].childNodes.toString ());
    }
    else if (SzNode.childNodes [i].nodeName == "COLS") {
      spalanz = parseInt (SzNode.childNodes [i].childNodes.toString ());
    }
  }
}
```

Die Variablen *zeilanz* und *spalanz* werden mit den Werten aus dem XML-File gefüllt.

Da sich erst bei der Verarbeitung der XML-Datei die Größe des CA ergibt, können das Raster und die Gitterlinien nur danach generiert werden. Das Programm wurde daher entsprechend umgestaltet. Die Flash-Datei und das zugehörige XML-File sind auf der beiliegenden CD unter *Beispiel3-4* zu finden.

Die XML-Spezifikation eines CA ist hiermit fertig entwickelt. Im folgenden Abschnitt erfolgt die Anwendung auf weitere Beispiele.

## 4. Weitere zellulare Automaten

### 4.1. Der Automat von Greenberg und Hastings

Jede Zelle kann einen ganzzahligen Zustand zwischen 0 und N-1 annehmen (vgl. ALLOUCHE00). Die Zustände zwischen 1 und e ( $0 < e < N-1$ ) heißen Erregungszustände (excited states). Die Zustandsübergangsfunktion beinhaltet folgende Fälle:

- Eine Zelle im Zustand k ( $1 \leq k < N-1$ ) erhält zum Folgezeitpunkt den Zustand k+1.
- Eine Zelle im Zustand N-1 erhält den Folgezustand 0.
- Eine Zelle im Zustand 0
  - erhält den Zustand 1, falls sich mindestens eine Nachbarzelle in einem Erregungszustand befindet,
  - bleibt im Zustand 0, falls keine Nachbarzelle erregt ist.

Als Nachbarzellen werden die nördliche, östliche, südliche und westliche Zelle (von Neumann-Nachbarschaft) betrachtet.

Mit dem Greenberg-Hastings-CA soll die Ausbreitung eines Waldbrandes simuliert werden. Ein Waldstück (=Zelle) kann die Zustände brennend (1 = Erregungszustand), verbrannt (2 bis 4), wachsend (5) und brandgefährdet (0 = normaler Zustand) haben. Nach 3 Zeiteinheiten beginnt also auf einem verbrannten Waldstück wieder etwas zu wachsen. Die XML-Datei hat folgendes Aussehen:

```
<?xml version="1.0" encoding="windows-1252"?>
<CA>
  <NAME>Beispiel 4-1</NAME>
  <SIZE>
    <ROWS> 10 </ROWS>
    <COLS> 10 </COLS>
  </SIZE>
  <INITIAL_STATE>
    <CELL><X>0</X> <Y>0</Y> <VAL>1</VAL></CELL>
  </INITIAL_STATE>
  <TRANSITION_RULES>
    <RULE>
      <IF> ZIJ = 1 </IF>
      <THEN> 2 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 2 </IF>
      <THEN> 3 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 3 </IF>
      <THEN> 4 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 4 </IF>
      <THEN> 5 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 5 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> FOCSN = 0 </IF>
      <THEN> 0 </THEN>
    </RULE>
  </TRANSITION_RULES>
</CA>
```

```

</RULE>
<RULE>
  <IF> Z (I+1,J) = 1 </IF>
  <THEN> 1 </THEN>
</RULE>
<RULE>
  <IF> Z (I-1,J) = 1 </IF>
  <THEN> 1 </THEN>
</RULE>
<RULE>
  <IF> Z (I,J+1) = 1 </IF>
  <THEN> 1 </THEN>
</RULE>
<RULE>
  <IF> Z (I,J-1) = 1 </IF>
  <THEN> 1 </THEN>
</RULE>
<RULE>
  <IF>          </IF>
  <THEN> 0 </THEN>
</RULE>
</TRANSITION_RULES>
<SCALE>
  <VAL> 0 </VAL>   <COL> 009900 </COL>
  <VAL> 1 </VAL>   <COL> FF0000 </COL>
  <VAL> 2 </VAL>   <COL> 000000 </COL>
  <VAL> 3 </VAL>   <COL> 000000 </COL>
  <VAL> 4 </VAL>   <COL> 000000 </COL>
  <VAL> 5 </VAL>   <COL> 99FF99 </COL>
</SCALE>
</CA>

```

Als Initialzustand beginnt der Brand in Zelle (0,0). Ein brennendes (= 1) Waldstück hat die Farbe rot, ein verbranntes (= 2 bis 4) ist schwarz, ein wachsendes (= 5) hellgrün und ein brandgefährdetes normales (= 0) Waldstück ist grün. Das in den Regeln verwendete Symbol ZIJ stellt den Wert der aktuellen Zelle und ist gleichbedeutend mit Z (I,J). Die Auswertung ist allerdings performanter. Die Funktion FOCSN (Focal Sum Neumann) liefert die Summe der Zellattributwerte in der von Neumann-Nachbarschaft. Die Regel mit dieser Funktion ist redundant und ist nur aus Performancegründen angegeben.

Abbildung 4-1: Am Anfang brennt nur die Zelle links oben.

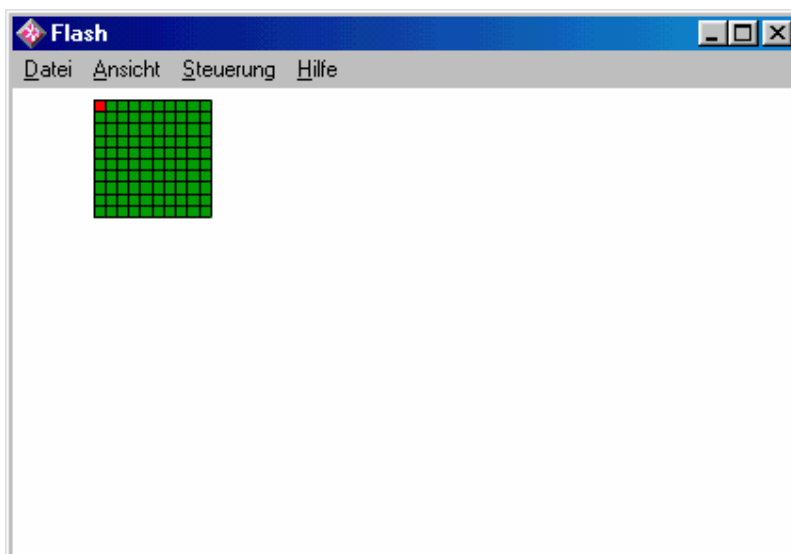


Abbildung 4-2: Der Zeitpunkt  $t = 1$

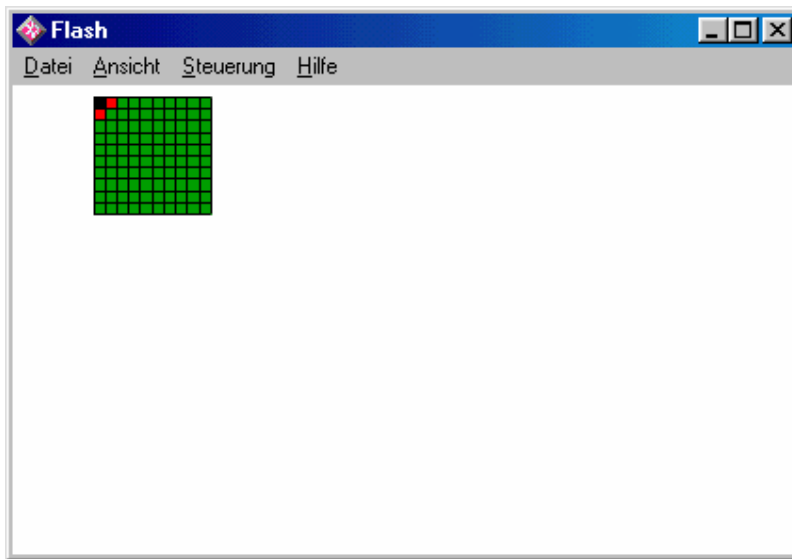


Abbildung 4-3: Der Zeitpunkt  $t = 2$

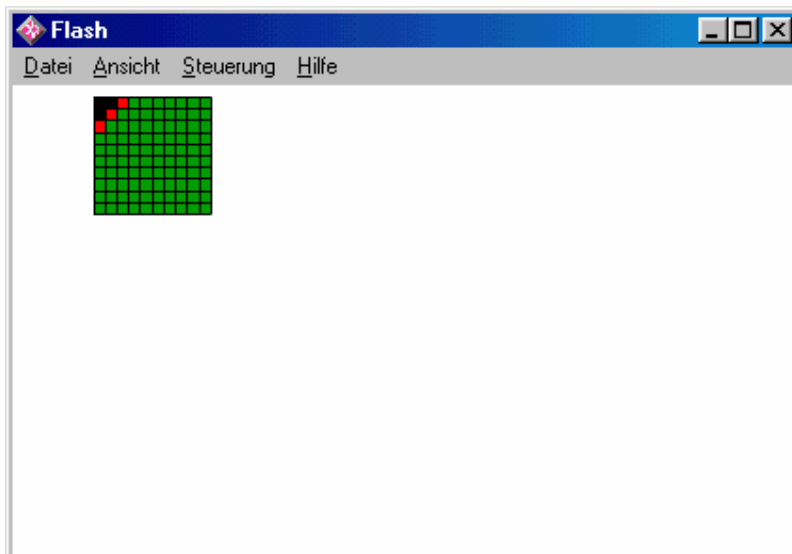


Abbildung 4-4: Der Zeitpunkt  $t = 3$

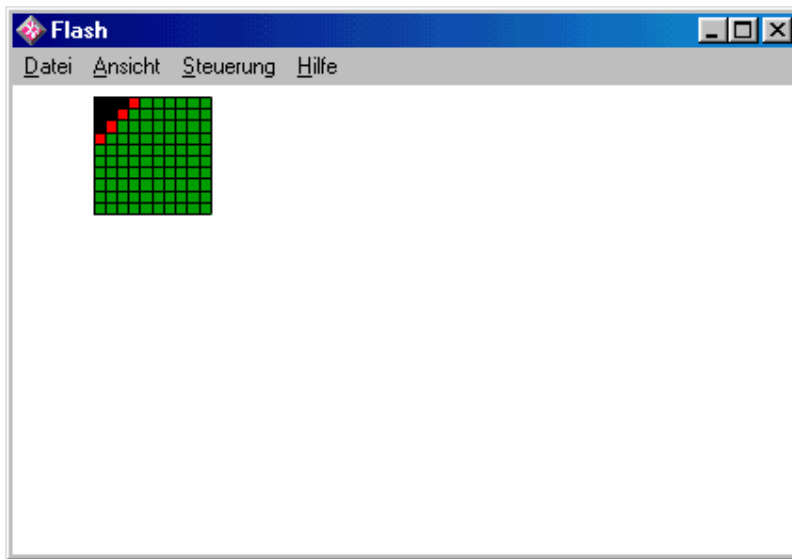


Abbildung 4-5: Links oben beginnt es wieder zu wachsen.

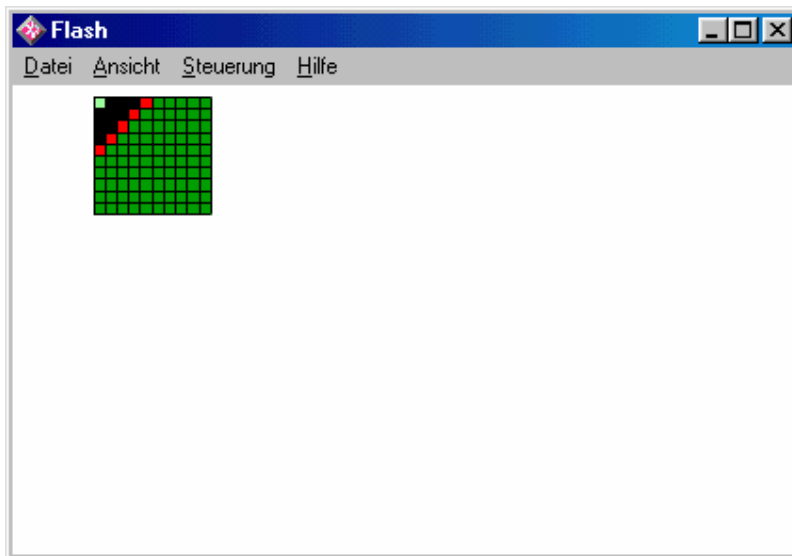


Abbildung 4-6: Der Zeitpunkt  $t = 5$

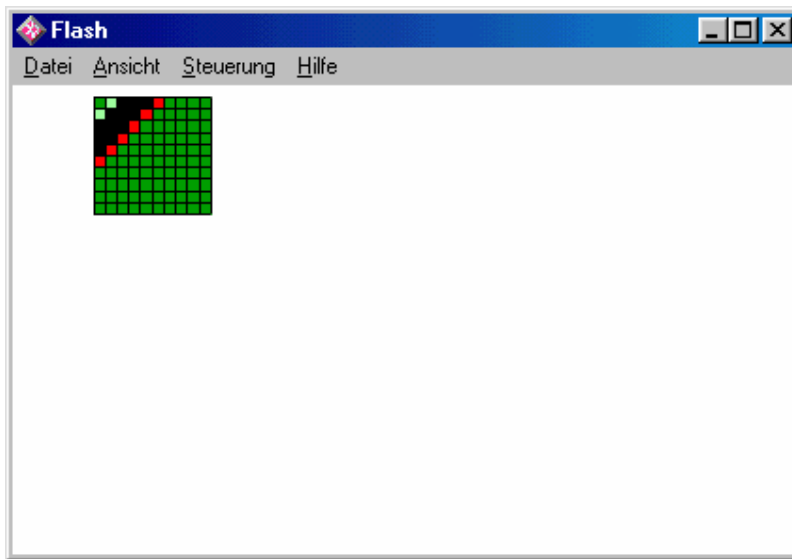


Abbildung 4-7: Nach einigen weiteren Zeitpunkten.

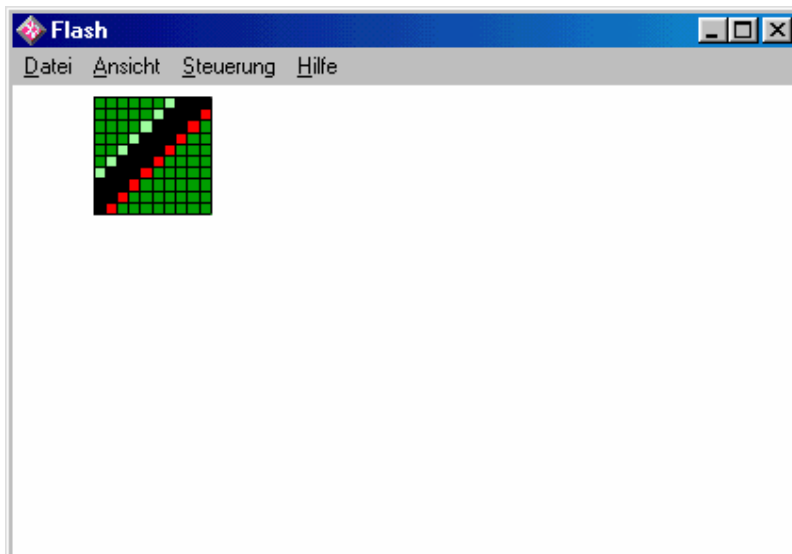
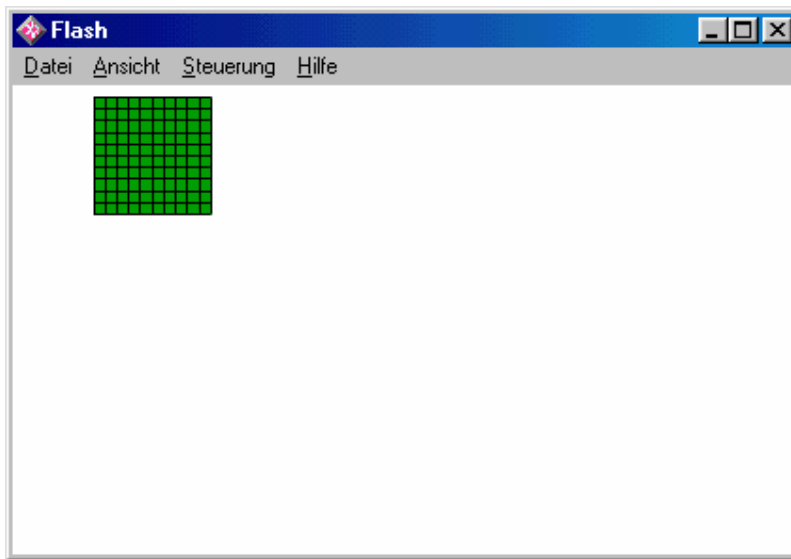


Abbildung 4-8: Der letzte Zeitpunkt: Alles ist wieder normal.



Das Modell lässt sich natürlich realitätsnaher gestalten. Z. B. kann ein brandgefährdetes Waldstück auch von sich aus mit einer gewissen Wahrscheinlichkeit zum Brennen anfangen, d.h. bei der Zustandsübergangsfunktion kann die Zufallszahlenfunktion verwendet werden. Ebenso könnte eine brennende Zelle noch mehrere Perioden lang weiter brennen, d.h. es gäbe, wie beim Zustand verbrannt, mehrere Zustände für brennend.

Die Flash-Datei und das zugehörige XML-File sind auf der beiliegenden CD unter *Beispiel4-1* zu finden.

#### **4.2. Ein CA-Modell für Pflanzenpopulationen**

Betrachtet werden zwei Pflanzenarten (vgl. WEIMAR96). Eine Art (= S) gedeiht eher unter feuchten Bedingungen (z.B. Schilf), die andere (= L) unter trockenen Bedingungen (z.B. Löwenzahn).

Je Zelle sind vier Zustände möglich:

- 0 – Keine der beiden Arten kommt vor.
- 1 – Nur L kommt vor.
- 2 – Nur S kommt vor.
- 3 – Vorkommen von S und L.

Für trockene Bedingungen gilt folgendes Regelwerk für Zustandsübergänge:

0	->	1	mit Wahrscheinlichkeit $p_2$
	->	0	sonst
1	->	1	
2	->	1	mit Wahrscheinlichkeit $p_1 p_2$
	->	0	mit Wahrscheinlichkeit $p_1 (1 - p_2)$
	->	3	mit Wahrscheinlichkeit $(1 - p_1) p_2$
	->	2	sonst
3	->	1	mit Wahrscheinlichkeit $p_1$
	->	3	sonst

Das zugehörige XML-File für ein 4 x 4-Raster mit  $p_1 = 50\%$  und  $p_2 = 20\%$  weist folgendes

Aussehen auf:

```
<?xml version="1.0" encoding="windows-1252"?>
<CA>
  <NAME>Beispiel 4-2</NAME>
  <SIZE>
    <ROWS> 4 </ROWS>
    <COLS> 4 </COLS>
  </SIZE>
  <INITIAL_STATE>
    <CELL><X>0</X> <Y>0</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>0</Y> <VAL>1</VAL></CELL>
    <CELL><X>2</X> <Y>0</Y> <VAL>2</VAL></CELL>
    <CELL><X>3</X> <Y>0</Y> <VAL>2</VAL></CELL>
    <CELL><X>0</X> <Y>1</Y> <VAL>3</VAL></CELL>
    <CELL><X>1</X> <Y>1</Y> <VAL>3</VAL></CELL>
    <CELL><X>2</X> <Y>1</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>1</Y> <VAL>2</VAL></CELL>
    <CELL><X>0</X> <Y>2</Y> <VAL>2</VAL></CELL>
    <CELL><X>1</X> <Y>2</Y> <VAL>1</VAL></CELL>
    <CELL><X>2</X> <Y>2</Y> <VAL>2</VAL></CELL>
    <CELL><X>3</X> <Y>2</Y> <VAL>2</VAL></CELL>
    <CELL><X>0</X> <Y>3</Y> <VAL>1</VAL></CELL>
    <CELL><X>1</X> <Y>3</Y> <VAL>1</VAL></CELL>
    <CELL><X>2</X> <Y>3</Y> <VAL>1</VAL></CELL>
    <CELL><X>3</X> <Y>3</Y> <VAL>3</VAL></CELL>
  </INITIAL_STATE>
  <TRANSITION_RULES>
    <RULE>
      <IF> ZIJ = 1 </IF>
      <THEN> 1 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 & RANDOM &lt;= 0.2 </IF>
      <THEN> 1 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 2 & RANDA &lt;= 0.1 </IF>
      <THEN> 1 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 2 & RANDA &lt;= 0.5 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 2 & RANDA &lt;= 0.6 </IF>
      <THEN> 3 </THEN>
    </RULE>
  </TRANSITION_RULES>
</CA>
```

```

    <IF> ZIJ = 2 </IF>
    <THEN> 2 </THEN>
</RULE>
<RULE>
    <IF> RANDOM &lt;= 0.5 </IF>
    <THEN> 1 </THEN>
</RULE>
<RULE>
    <IF> </IF>
    <THEN> 3 </THEN>
</RULE>
</TRANSITION_RULES>
<SCALE>
    <VAL> 0 </VAL> <COL> FFFFFF </COL>
    <VAL> 1 </VAL> <COL> FFFF00 </COL>
    <VAL> 2 </VAL> <COL> 00FF00 </COL>
    <VAL> 3 </VAL> <COL> 99FF00 </COL>
</SCALE>
</CA>

```

Die Funktion RANDA ermittelt zu Beginn des Regelwerks für eine bestimmte Zelle eine Zufallszahl und speichert diese in das Register A. Jeder weitere Aufruf in den Regeln für die gleiche Zelle zum gleichen Zeitpunkt liest die Zahl aus dem Register A. Bei der nächsten Zelle bzw. beim nächsten Zeitpunkt wird A mit einer neuen Zufallszahl befüllt. In gleicher Weise funktionieren RANDB bis RANDE (Register B bis E). Im Gegensatz dazu liefert RANDOM bei jedem Aufruf eine neue Zufallszahl.

Die Zustände haben folgende Farben:

- 0 – weiß
- 1 – grün (nur S)
- 2 – gelb (nur L)
- 3 – gelbgrün (S und L)

Abbildung 4-9: Der Anfangszustand ( $t = 0$ ).

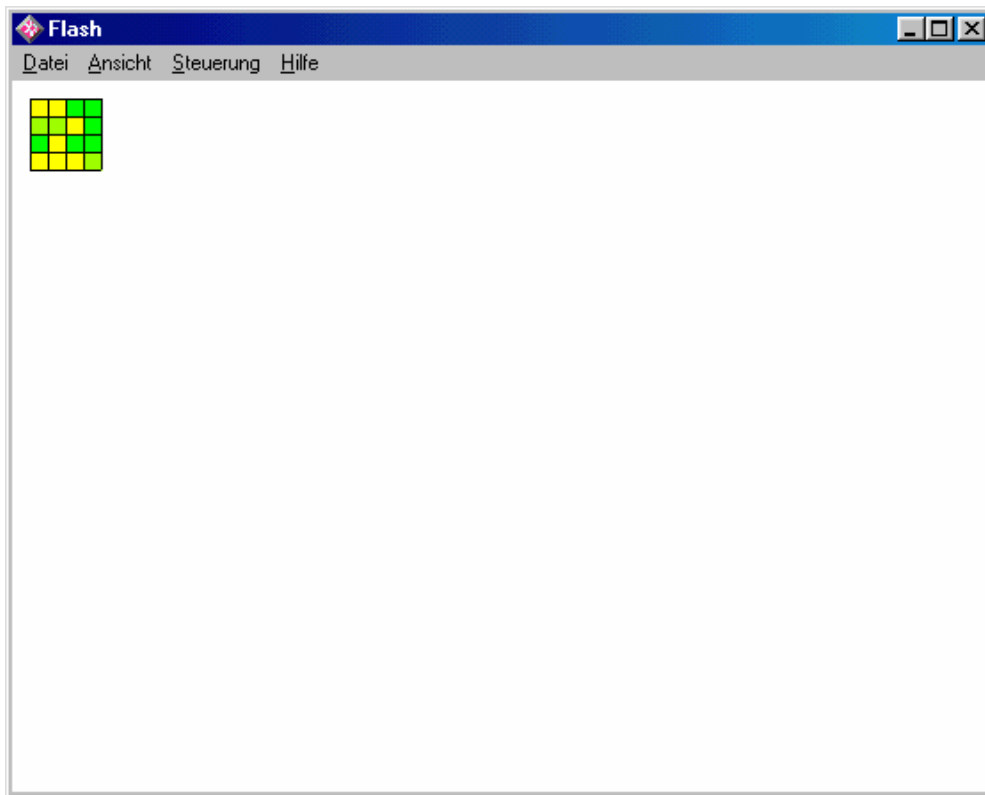


Abbildung 4-10: Zum Zeitpunkt  $t = 1$ .

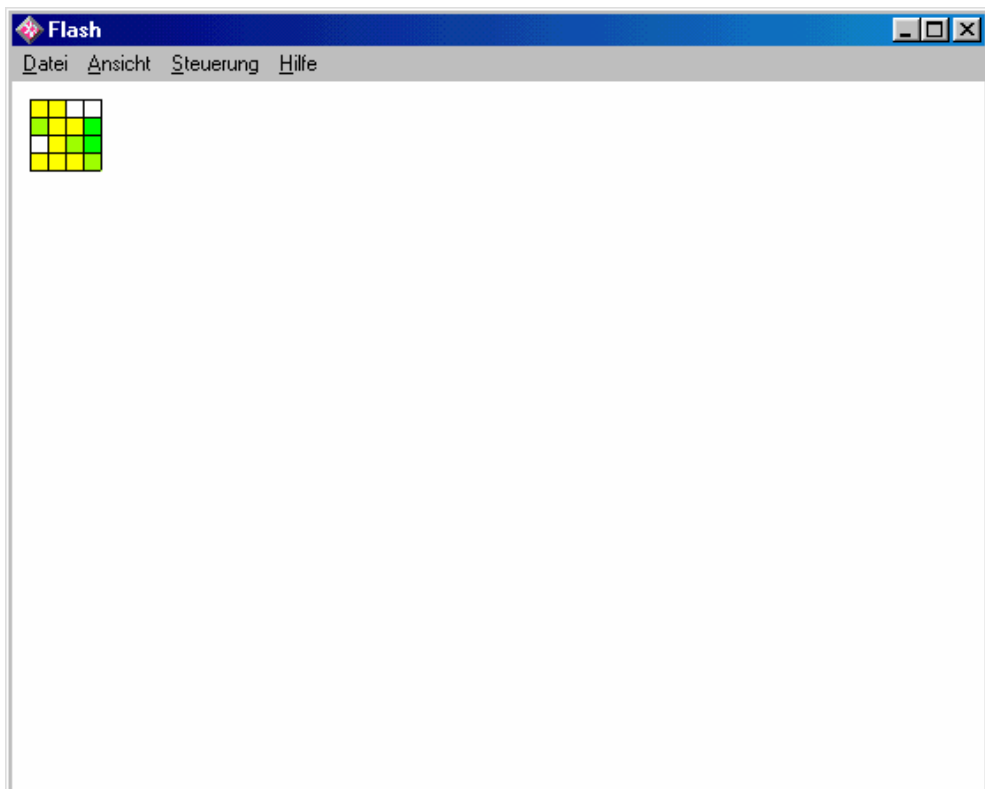


Abbildung 4-11: Zum Zeitpunkt  $t = 2$ .

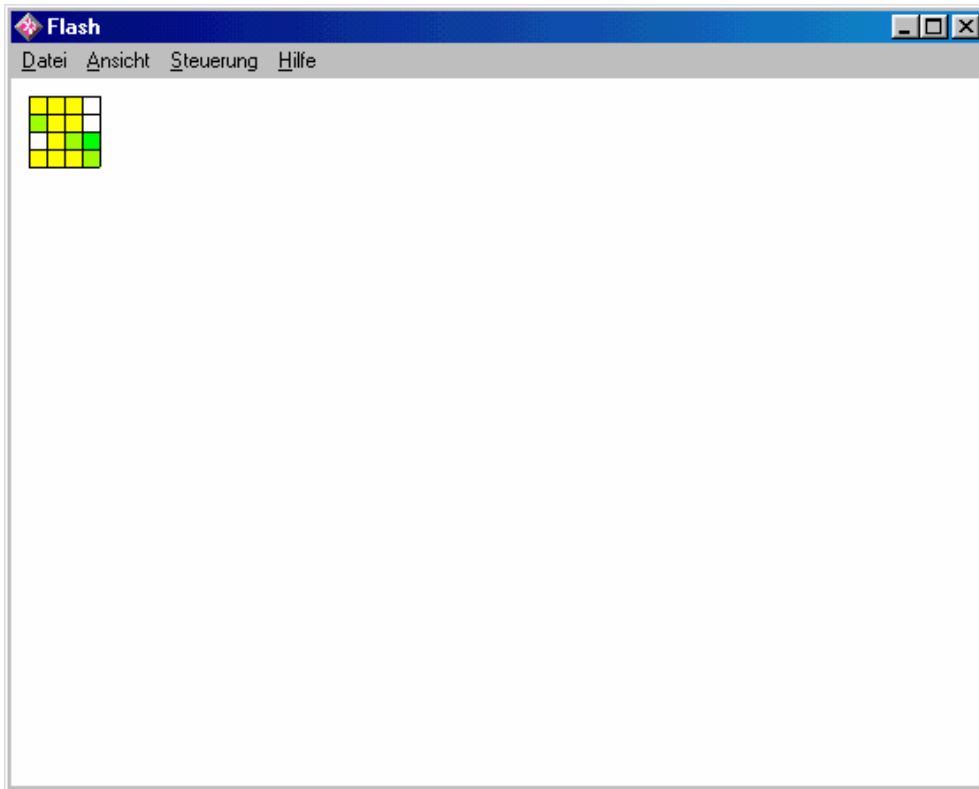


Abbildung 4-12: Zum Zeitpunkt  $t = 3$ .

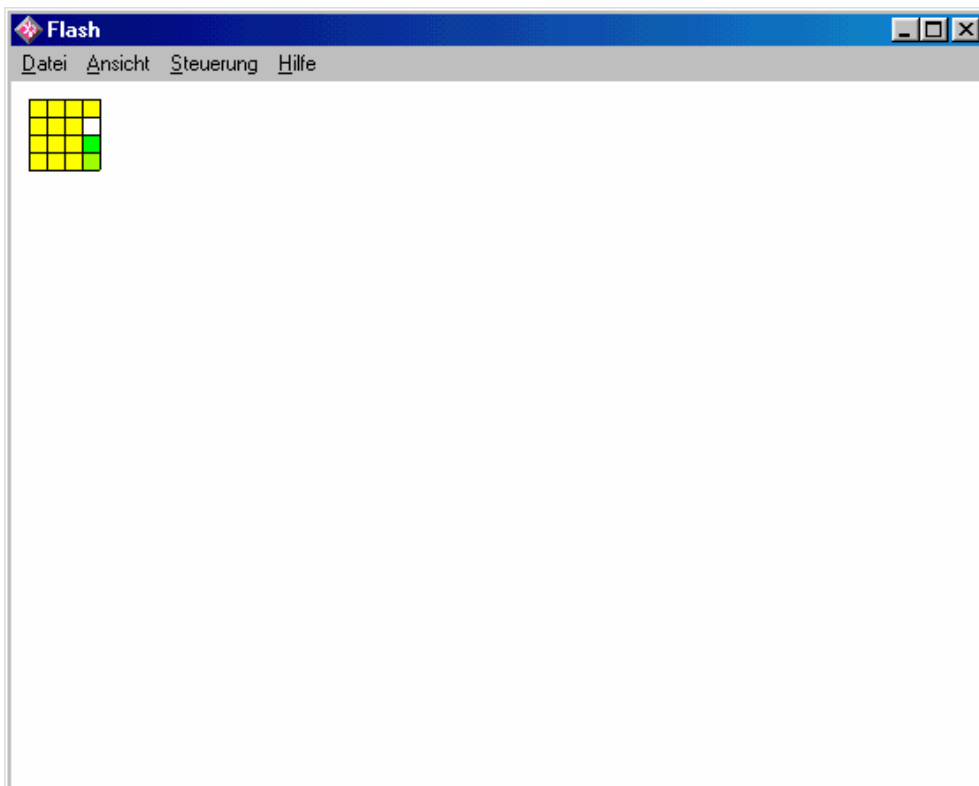


Abbildung 4-13: Zum Zeitpunkt  $t = 4$ .

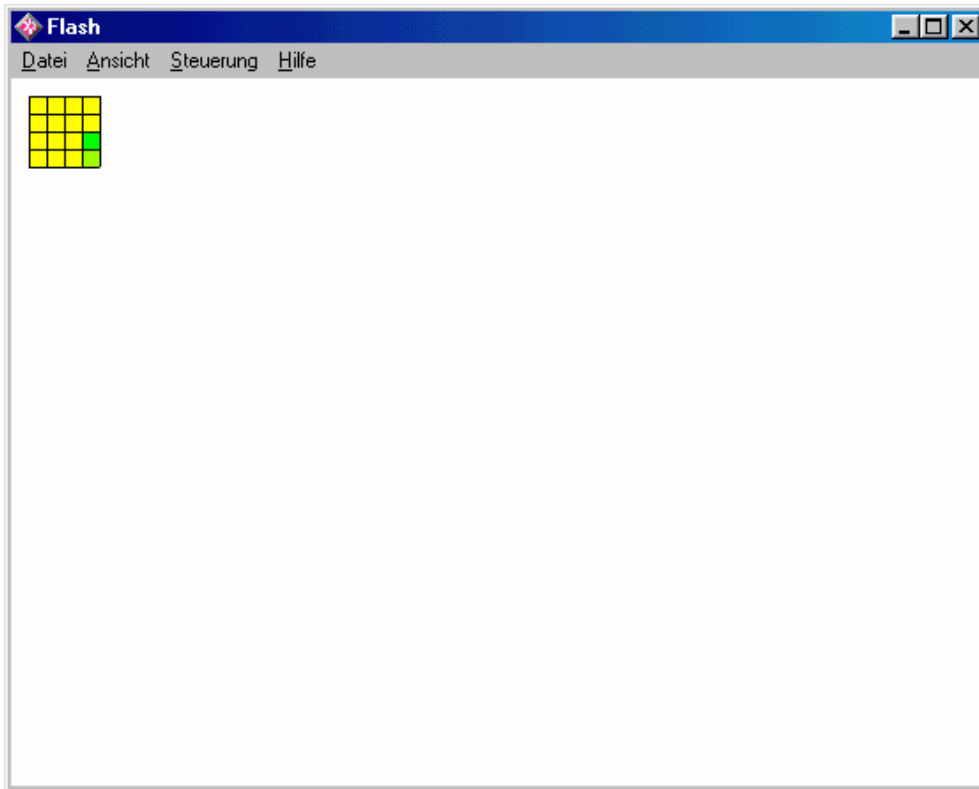


Abbildung 4-14: Zum Zeitpunkt  $t = 5$ .

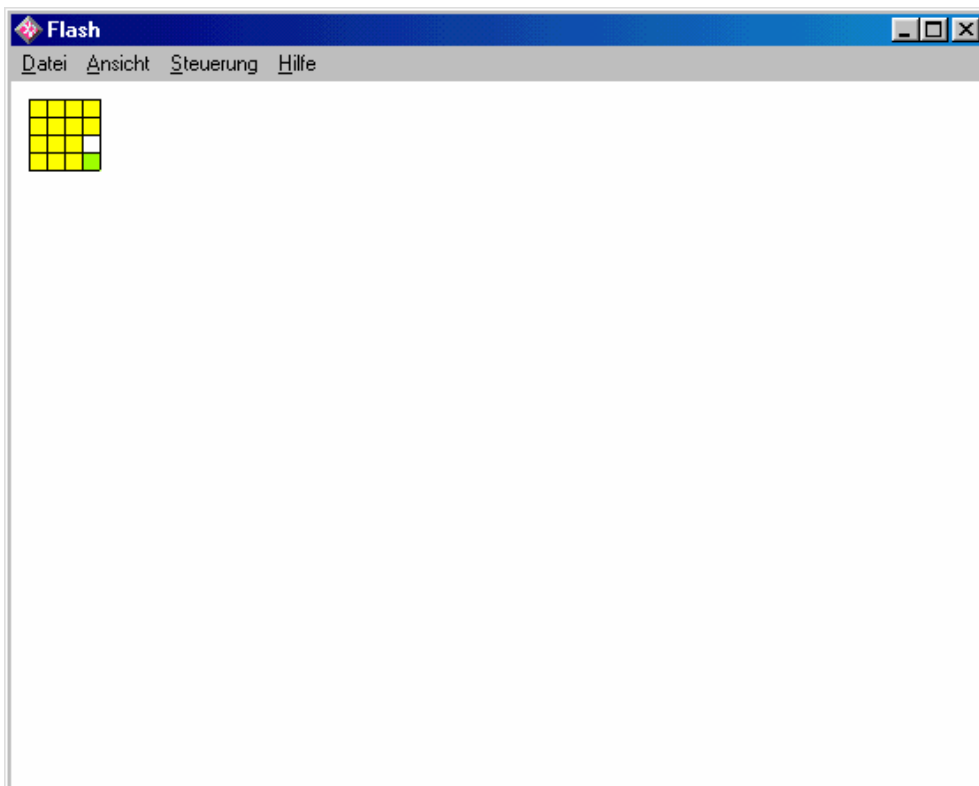


Abbildung 4-15: Zum Zeitpunkt  $t = 6$ .

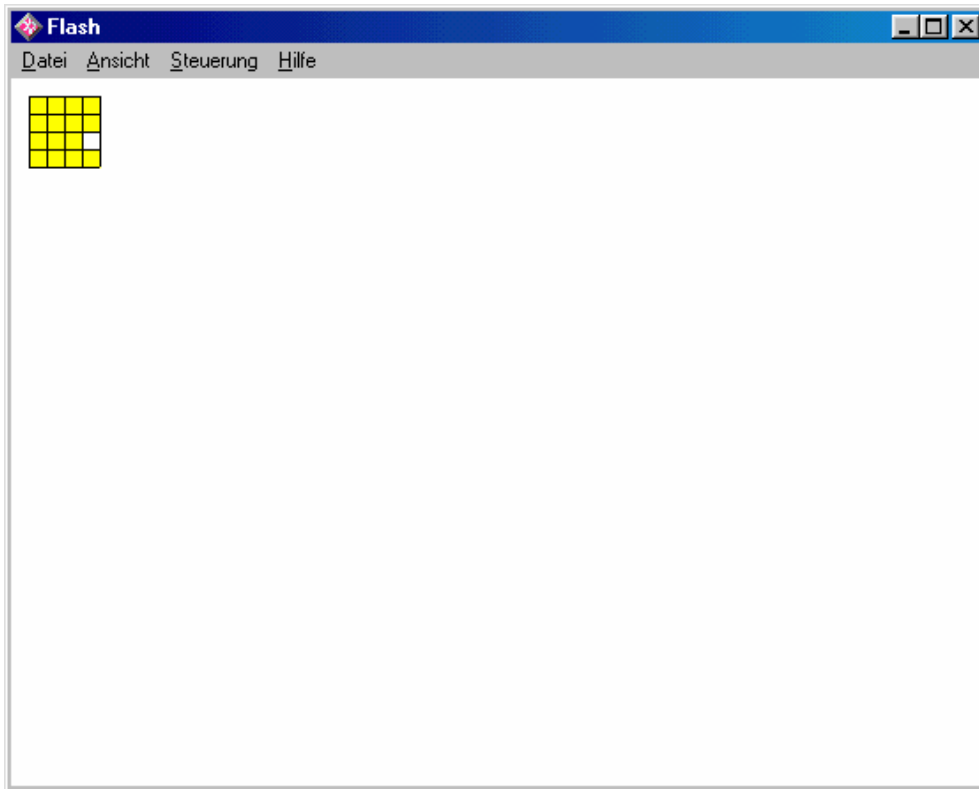
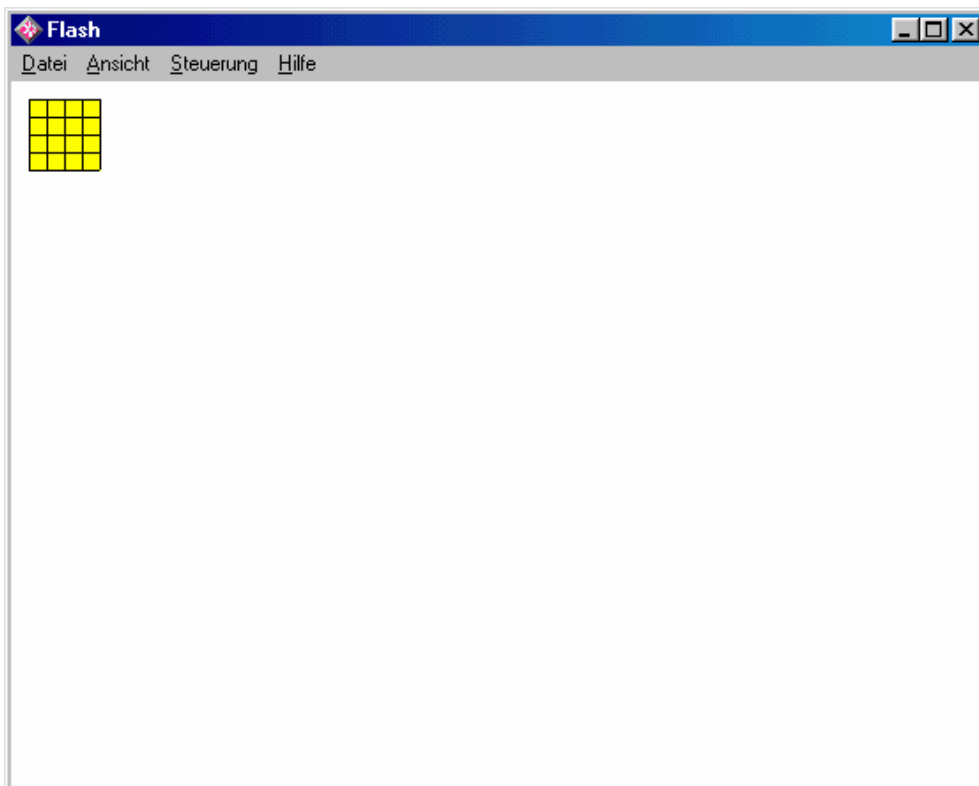


Abbildung 4-16: Am Ende existiert nur mehr L(öwenzahn).



Für feuchte Bedingungen gibt es ein anderes Regelwerk. Die Abfolge trockener und feuchter Bedingungen könnte zufallsgesteuert erfolgen. Andere Erweiterungen erlauben kein gleichzeitiges Vorkommen von S und L in einer Zelle. Anstatt trockener oder feuchter Bedingungen für das gesamte Raster könnte jede Zelle ihre eigene zufallsbestimmte Durchschnittsfeuchtigkeit haben.

Die Flash-Datei und das zugehörige XML-File sind auf der beiliegenden CD unter *Beispiel4-2* zu finden.

### 4.3. Verkehrssimulation

Der Verkehr auf einer Straße soll mit Hilfe eines eindimensionalen CA simuliert werden (vgl. WEIMAR96). Das Modell besteht nur aus einer Zeile (bzw. einer Spalte), die die Straße darstellt. Der Verkehr bewegt sich nur in einer Richtung. Überholen ist nicht möglich. Auf einer Zelle hat genau ein Kraftwagen Platz. Folgende Zustände einer Zelle sind möglich:

- 0 – Die Zelle ist leer (= weiß).
- 1 – Auf der Zelle befindet sich ein haltender Kraftwagen (= rot).
- 2 – Der Kraftwagen auf der Zelle bewegt sich fort (= grün).

Bei der Zustandsübergangsfunktion werden die vordere und die hintere Nachbarzelle berücksichtigt. Für den Zustandsübergang der mittleren Zelle gelten folgende Regeln:

0 0 0	->	0	
0 0 1	->	0	
0 0 2	->	1	mit Wahrscheinlichkeit p
	->	2	sonst
0 1 x	->	2	
0 2 x	->	0	
1 0 0	->	0	
1 0 1	->	0	
1 0 2	->	1	
1 1 x	->	1	
1 2 x	->	0	
2 0 0	->	0	
2 0 1	->	0	
2 0 2	->	1	mit Wahrscheinlichkeit p

-> 2    sonst

2 1 x -> 1

2 2 x -> 0

(x ... 0, 1, 2)

Das zugehörige XML-File hat folgendes Aussehen:

```
<?xml version="1.0" encoding="windows-1252"?>
<CA>
  <NAME>Beispiel 4-3</NAME>
  <SIZE>
    <ROWS> 10 </ROWS>
    <COLS> 1 </COLS>
  </SIZE>
  <INITIAL_STATE>
    <CELL><X>0</X> <Y>0</Y> <VAL>0</VAL></CELL>
    <CELL><X>0</X> <Y>1</Y> <VAL>2</VAL></CELL>
    <CELL><X>0</X> <Y>2</Y> <VAL>1</VAL></CELL>
    <CELL><X>0</X> <Y>3</Y> <VAL>0</VAL></CELL>
    <CELL><X>0</X> <Y>4</Y> <VAL>0</VAL></CELL>
    <CELL><X>0</X> <Y>5</Y> <VAL>0</VAL></CELL>
    <CELL><X>0</X> <Y>6</Y> <VAL>2</VAL></CELL>
    <CELL><X>0</X> <Y>7</Y> <VAL>0</VAL></CELL>
    <CELL><X>0</X> <Y>8</Y> <VAL>2</VAL></CELL>
    <CELL><X>0</X> <Y>9</Y> <VAL>0</VAL></CELL>
  </INITIAL_STATE>
  <TRANSITION_RULES>
    <RULE>
      <IF> ZIJ = 0 &amp; Z(I-1,0) = 0 &amp; Z(I+1,0) = 0 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 &amp; Z(I-1,0) = 0 &amp; Z(I+1,0) = 1 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 &amp; Z(I-1,0) = 0 &amp; Z(I+1,0) = 2 &amp; RANDA &lt;= 0.4 </IF>
      <THEN> 1 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 &amp; Z(I-1,0) = 0 &amp; Z(I+1,0) = 2 </IF>
      <THEN> 2 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 1 &amp; Z(I-1,0) = 0 </IF>
      <THEN> 2 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 2 &amp; Z(I-1,0) = 0 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 &amp; Z(I-1,0) = 1 &amp; Z(I+1,0) = 0 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 &amp; Z(I-1,0) = 1 &amp; Z(I+1,0) = 1 </IF>
      <THEN> 0 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 0 &amp; Z(I-1,0) = 1 &amp; Z(I+1,0) = 2 </IF>
      <THEN> 1 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 1 &amp; Z(I-1,0) = 1 </IF>
      <THEN> 1 </THEN>
    </RULE>
    <RULE>
      <IF> ZIJ = 2 &amp; Z(I-1,0) = 1 </IF>
      <THEN> 0 </THEN>
    </RULE>
  </TRANSITION_RULES>
</CA>
```

```

<IF> ZIJ = 0 & Z(I-1,0) = 2 & Z(I+1,0) = 0 </IF>
<THEN> 0 </THEN>
</RULE>
<RULE>
<IF> ZIJ = 0 & Z(I-1,0) = 2 & Z(I+1,0) = 1 </IF>
<THEN> 0 </THEN>
</RULE>
<RULE>
<IF> ZIJ = 0 & Z(I-1,0) = 2 & Z(I+1,0) = 2 & RANDA <= 0.4 </IF>
<THEN> 1 </THEN>
</RULE>
<RULE>
<IF> ZIJ = 0 & Z(I-1,0) = 2 & Z(I+1,0) = 2 </IF>
<THEN> 2 </THEN>
</RULE>
<RULE>
<IF> ZIJ = 1 & Z(I-1,0) = 2 </IF>
<THEN> 1 </THEN>
</RULE>
<RULE>
<IF> ZIJ = 2 & Z(I-1,0) = 2 </IF>
<THEN> 0 </THEN>
</RULE>
</TRANSITION_RULES>
<SCALE>
<VAL> 0 </VAL> <COL> FFFFFFFF </COL>
<VAL> 1 </VAL> <COL> FF0000 </COL>
<VAL> 2 </VAL> <COL> 00FF00 </COL>
</SCALE>
</CA>

```

In diesem Fall besteht der Automat aus einer Spalte. Der Verkehr fließt von unten nach oben.  
Die Wahrscheinlichkeit  $p$  wird mit 40% angenommen.

Abbildung 4-17: Der Anfangszustand ( $t = 0$ ).

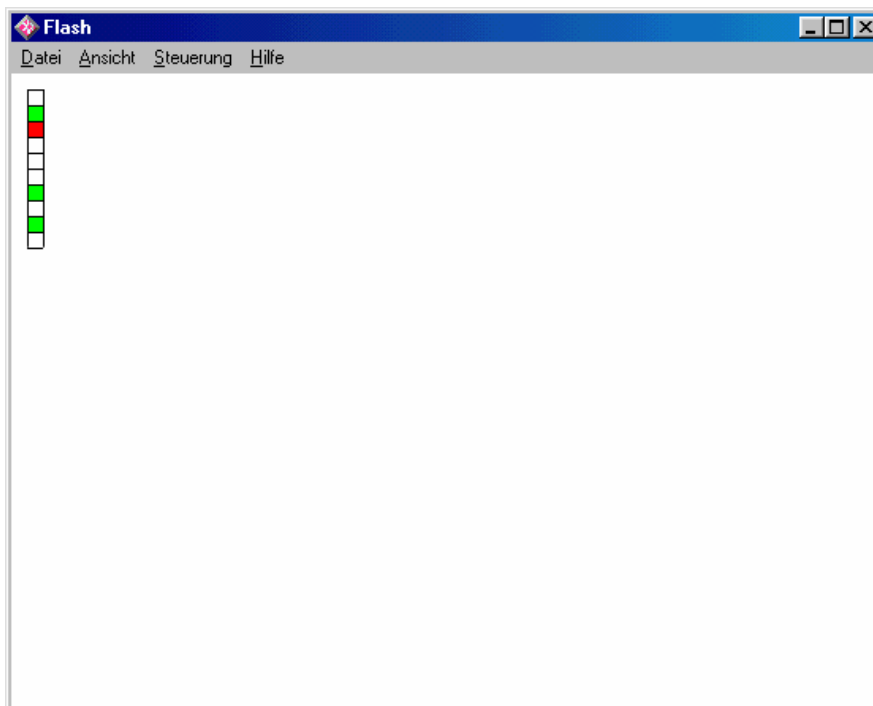


Abbildung 4-18: Der Zeitpunkt  $t = 1$ .

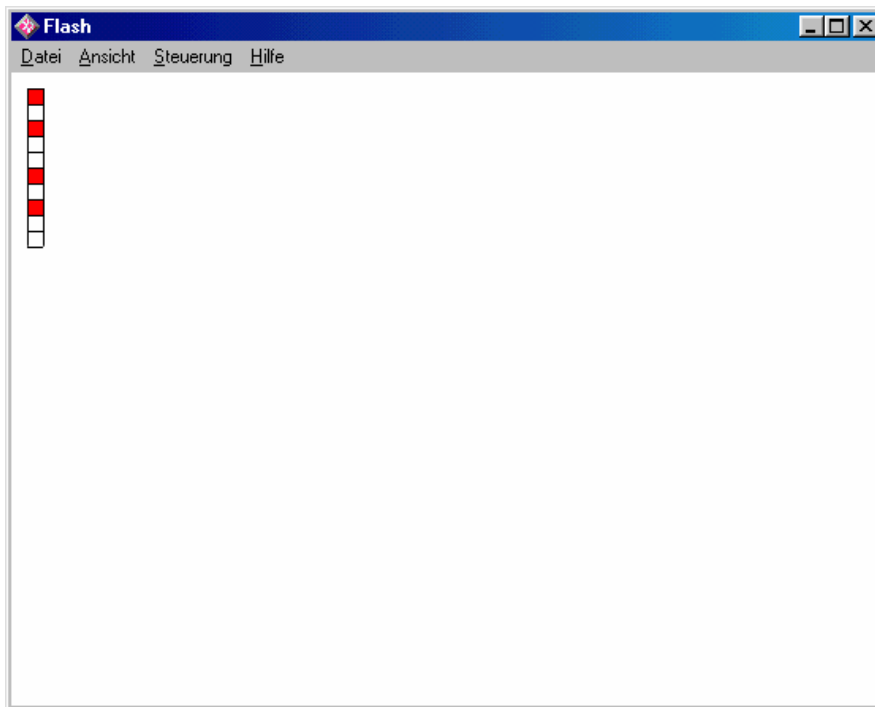


Abbildung 4-19: Der Zeitpunkt  $t = 2$ .

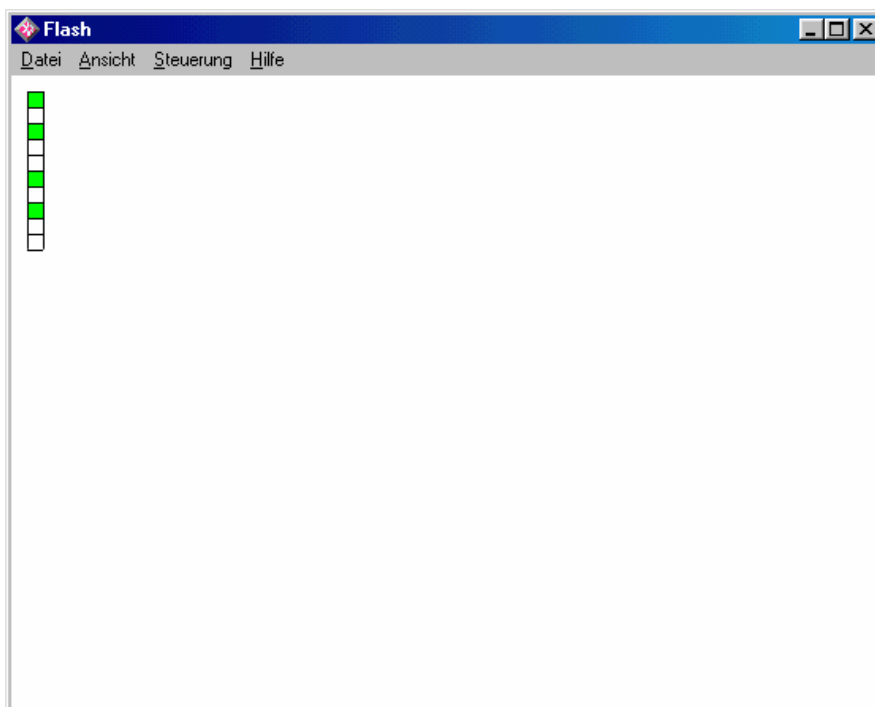


Abbildung 4-20: Der Zeitpunkt  $t = 3$ .

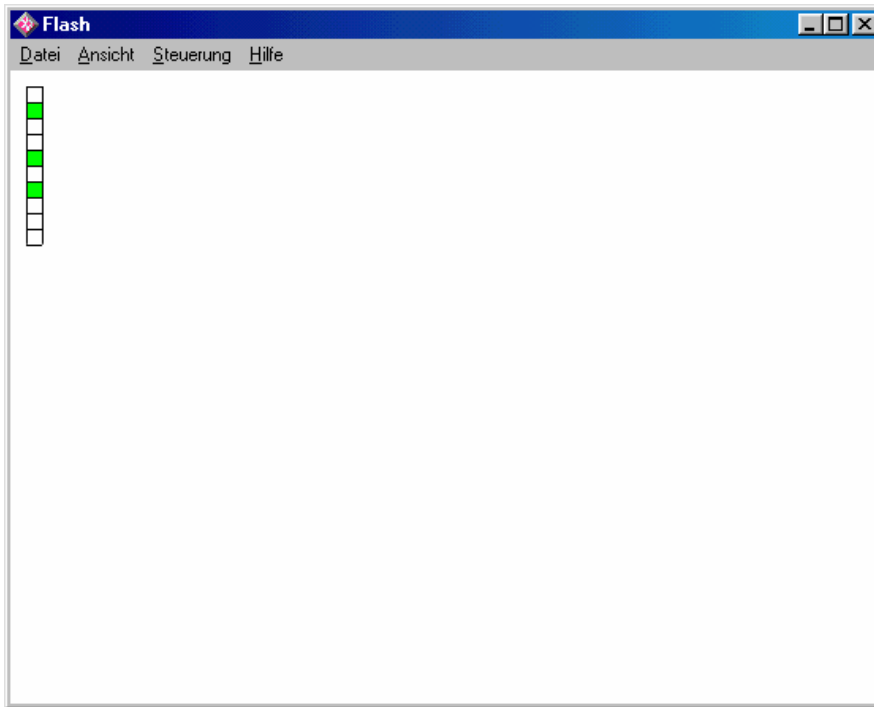


Abbildung 4-21: Der Zeitpunkt  $t = 4$ .

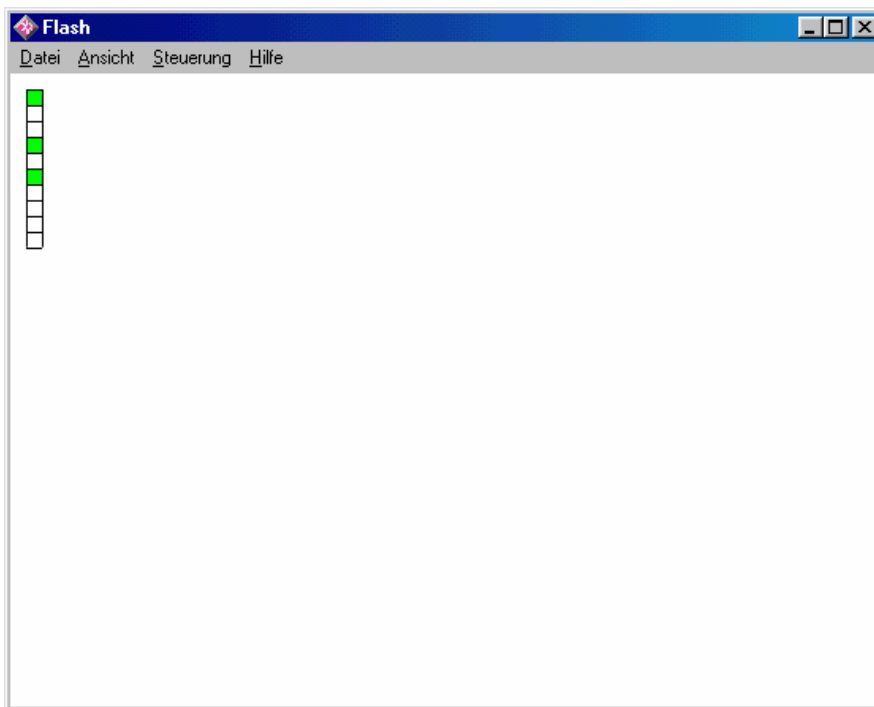


Abbildung 4-22: Der Zeitpunkt  $t = 5$ .

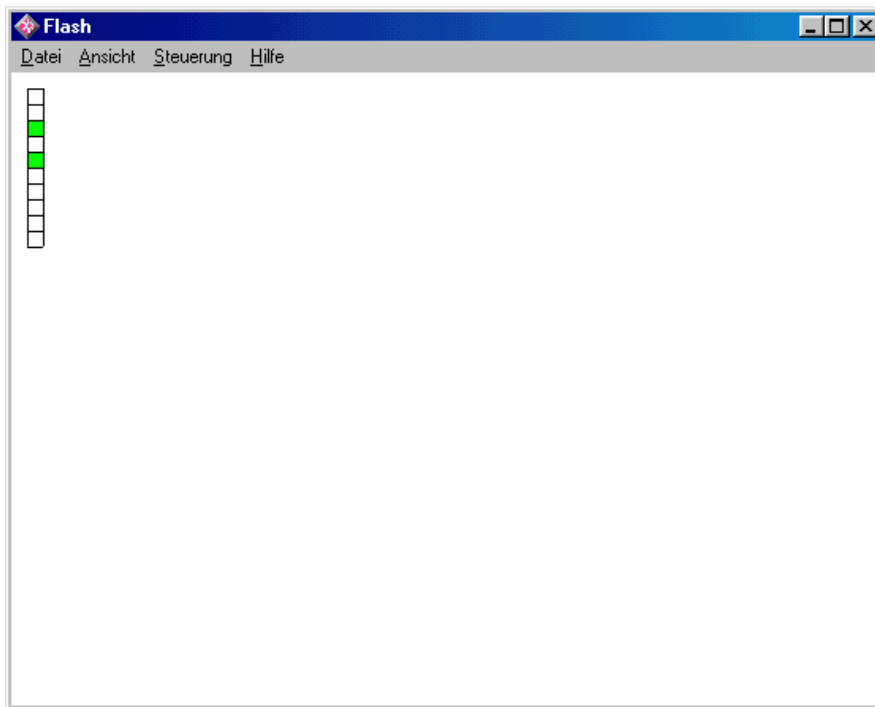


Abbildung 4-23: Der Zeitpunkt  $t = 6$ .

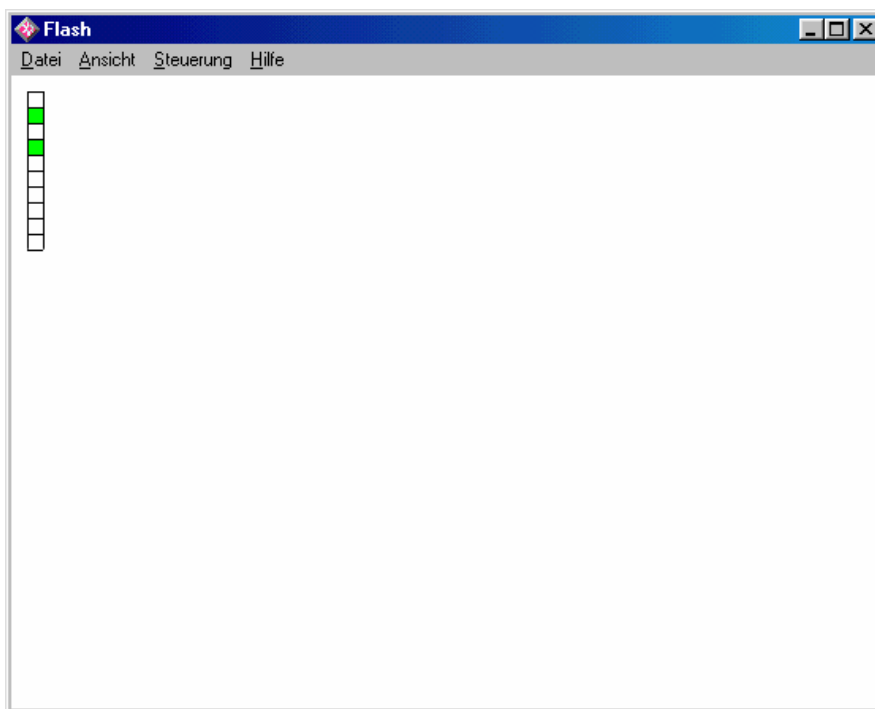


Abbildung 4-24: Der Zeitpunkt  $t = 7$ .

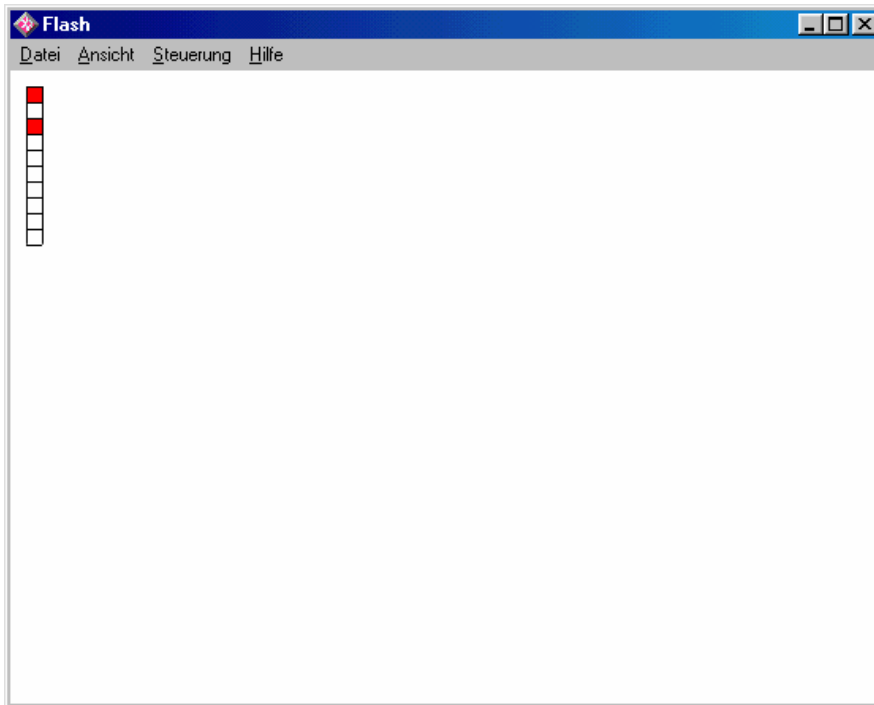


Abbildung 4-25: Der Zeitpunkt  $t = 8$ .

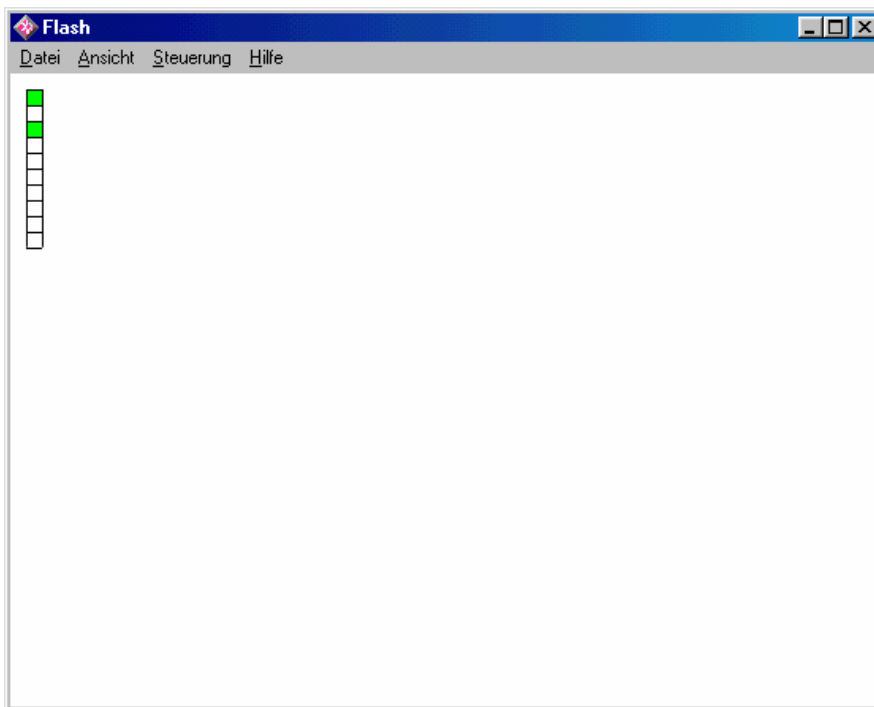


Abbildung 4-26: Der Zeitpunkt  $t = 9$ .

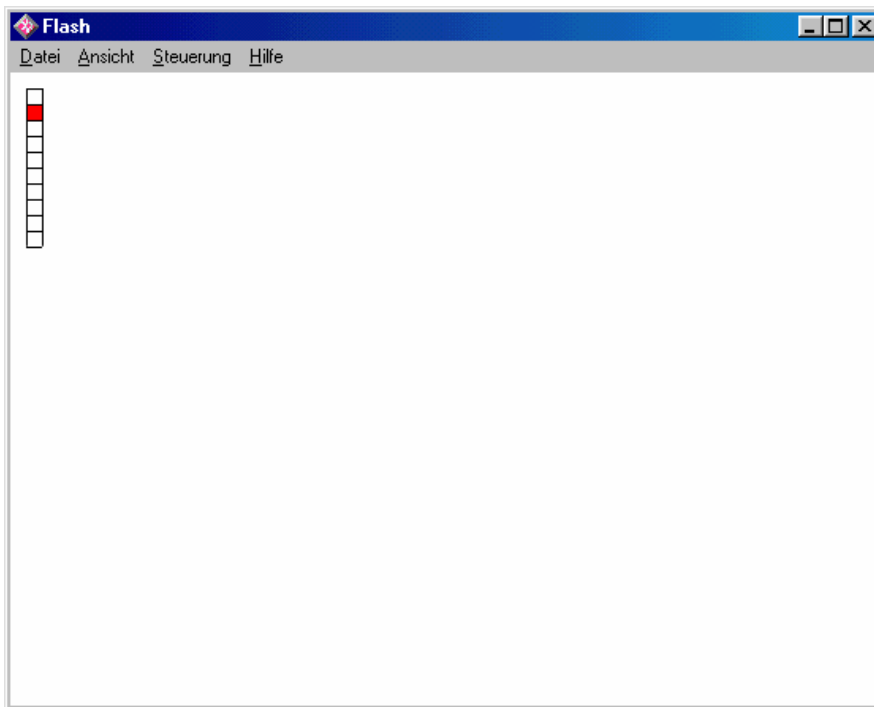


Abbildung 4-27: Der Zeitpunkt  $t = 10$ .

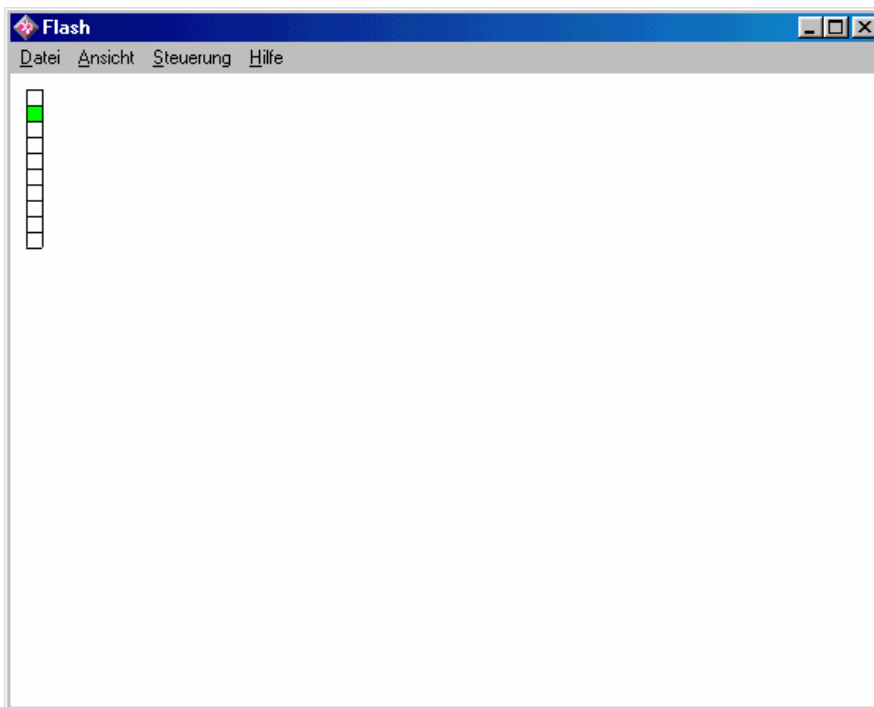


Abbildung 4-28: Der Zeitpunkt  $t = 11$ .

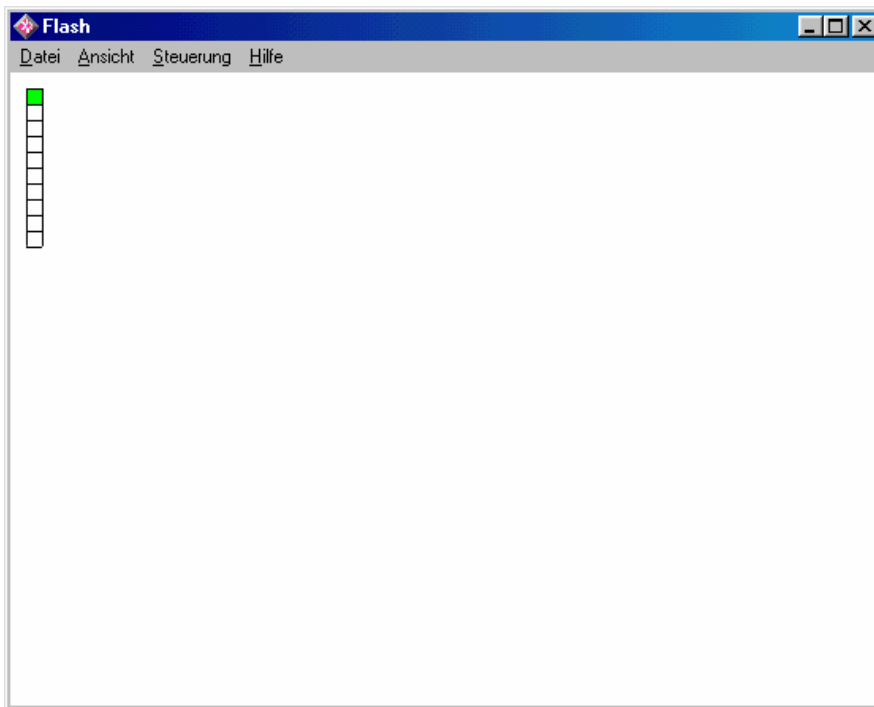
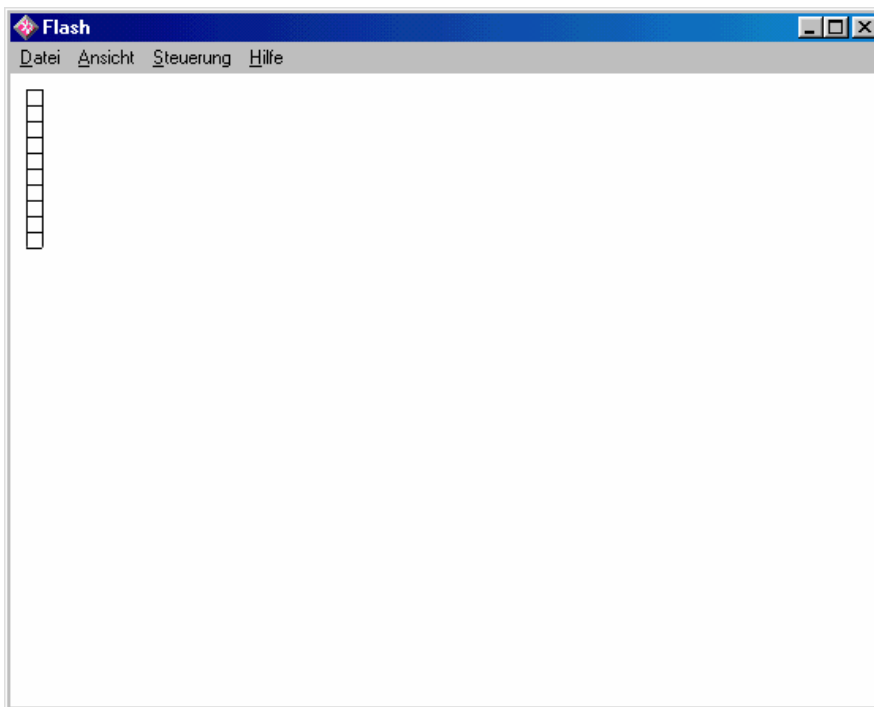


Abbildung 4-29: Die Straße ist leer ( $t = 12$ ).



Eine Erweiterungsmöglichkeit des Modells wäre die Berücksichtigung mehrerer Geschwindigkeiten. Die Flash-Datei und das zugehörige XML-File sind auf der beiliegenden CD unter *Beispiel4-3* zu finden.

## 5. Anhang

### 5.1. Abspielen der Flash-Dateien

Die Beispiele und der CA-Geosimulator auf der beiliegenden CD sind jeweils in den Formaten *.fla*, *.swf* und *.exe* gespeichert. Die *.fla*-Dateien können direkt in der Flash-Umgebung abgespielt und bearbeitet werden.

Für Dateien mit der Endung *.swf* ist der Standalone Player manuell zu starten und die Datei zu öffnen. Das Abspielen beginnt automatisch und kann mit dem Menübefehl *Steuerung-> Abspielen* bzw. mit der Tastenkombination *Strg + Eingabe* angehalten und wieder gestartet werden.

Bei den direkt ausführbaren Flash-Dateien (*.exe*) wird der Standalone Player automatisch gestartet und der Flash-Film sofort ausgeführt. Das Abspielen kann wie oben angeführt unterbrochen werden.

### 5.2. Die XML-Spezifikation des CA

Die XML-Datei eines CA hat denselben Namen wie die Flash-Datei und die Namenserverweiterung *.xml* (z.B. *CAGEOSIM.xml*). Sie weist folgenden Aufbau auf:

```
<?xml version="1.0" encoding="windows-1252"?>
<CA>
  <NAME> ... </NAME>
  <SIZE>
    <ROWS> ... </ROWS>
    <COLS> ... </COLS>
  </SIZE>
  <INITIAL_STATE>
    <CELL><X> ... </X> <Y> ... </Y> <VAL> ... </VAL></CELL>
    <CELL><X> ... </X> <Y> ... </Y> <VAL> ... </VAL></CELL>
    <CELL><X> ... </X> <Y> ... </Y> <VAL> ... </VAL></CELL>
    ...
  </INITIAL_STATE>
  <TRANSITION_RULES>
    <RULE>
      <IF> ... </IF>
      <THEN> ... </THEN>
    </RULE>
    <RULE>
      <IF> ... </IF>
      <THEN> ... </THEN>
    </RULE>
    ...
  </TRANSITION_RULES>
  <SCALE>
    <VAL> ... </VAL> <COL> ... </COL>
    <VAL> ... </VAL> <COL> ... </COL>
    ...
  </SCALE>
</CA>
```

Der Knoten NAME beinhaltet den Namen des CA und dient nur zur Information.

Z.B.: <NAME>Beispiel 4-3</NAME>

SIZE legt die Ausmaße des CA fest. Die Anzahl der Zeilen werden im Unterknoten ROWS, die der Spalten im Unterknoten COLS festgelegt.

```
Z.B.: <SIZE>
      <ROWS> 10 </ROWS>
      <COLS> 1 </COLS>
</SIZE>
```

INITIAL\_STATE legt den Anfangszustand des CA fest. Es werden nur die Zellen (Unterknoten CELL) angegeben, die mit einem Wert ungleich 0 zu initialisieren sind. Alle anderen Zellen werden automatisch mit 0 vorbelegt. Je (x,y)-Position (Unterknoten X bzw. Y) wird der Attributwert der Zelle (Unterknoten VAL) festgelegt. Die x-Koordinate ist gleichzeitig die Spalte, die y-Koordinate die Zeile der Zelle. Der Ursprung (0,0) liegt also links oben.

```
Z.B.: <INITIAL_STATE>
      <CELL><X>0</X> <Y>0</Y> <VAL>0</VAL></CELL>
      <CELL><X>0</X> <Y>1</Y> <VAL>2</VAL></CELL>
      <CELL><X>0</X> <Y>2</Y> <VAL>1</VAL></CELL>
      <CELL><X>0</X> <Y>3</Y> <VAL>0</VAL></CELL>
      <CELL><X>0</X> <Y>4</Y> <VAL>0</VAL></CELL>
      <CELL><X>0</X> <Y>5</Y> <VAL>0</VAL></CELL>
      <CELL><X>0</X> <Y>6</Y> <VAL>2</VAL></CELL>
      <CELL><X>0</X> <Y>7</Y> <VAL>0</VAL></CELL>
      <CELL><X>0</X> <Y>8</Y> <VAL>2</VAL></CELL>
      <CELL><X>0</X> <Y>9</Y> <VAL>0</VAL></CELL>
</INITIAL_STATE>
```

Der Knoten TRANSITION\_RULES legt die Zustandsübergangsfunktion in Form von Wenn-Dann-Regeln fest. Die Regeln (Unterknoten RULE) werden sequentiell für die aktuelle Zelle abgearbeitet bis eine Bedingung (Unterknoten IF) zutrifft. Der zugehörige Dann-Teil (Unterknoten THEN) gibt dann den Attributwert an. Falls keine Bedingung zutrifft, ist der resultierende Attributwert stets 0.

```
Z.B.: <TRANSITION_RULES>
      <RULE>
        <IF> ZIJ = 0 &amp; Z(I-1,0) = 0 &amp; Z(I+1,0) = 0 </IF>
        <THEN> 0 </THEN>
      </RULE>
      <RULE>
        <IF> ZIJ = 0 &amp; Z(I-1,0) = 0 &amp; Z(I+1,0) = 1 </IF>
        <THEN> 0 </THEN>
      </RULE>
      <RULE>
        <IF> ZIJ = 0 &amp; Z(I-1,0) = 0 &amp; Z(I+1,0) = 2 &amp; RANDA &lt;= 0.4 </IF>
        <THEN> 1 </THEN>
      </RULE>
</TRANSITION_RULES>
```

Der Aufbau der logischen bzw. arithmetischen Ausdrücke ist im nächsten Kapitel erläutert.

Der Knoten SCALE spezifiziert die farbliche Darstellung der Attributwerte. Der im Unterknoten VAL spezifizierte Wert, ist der Anfangsattributwert eines Intervalls, das bis zum

nächsten VAL-Wert reicht. Diesem Intervall wird der im Unterknoten COL angegebene Farbcode zugewiesen. Der Farbcode wird hexadezimal in der Form RRGGBB angegeben. RR, GG und BB sind 1-Byte-Zahlen zwischen 0 und 255 (hexadezimal zwischen 00 und FF) und repräsentieren den Rot-, Grün- bzw. Blauanteil. FF0000 z.B. ist rot, FFFF00 ist gelb, 0000FF ist blau, 000000 ist schwarz und FFFFFFFF ist weiß.

```
Z.B.: <SCALE>
      <VAL> 0 </VAL>   <COL> FFFFFFFF </COL>
      <VAL> 1 </VAL>   <COL> FF0000 </COL>
      <VAL> 2 </VAL>   <COL> 00FF00 </COL>
</SCALE>
```

Attributwerte aus dem Intervall [0,1) erhalten in diesem Beispiel die Farbe weiß zugewiesen. Das Intervall [1,2) wird rot und [2,∞) grün (00FF00) dargestellt.

Falls für einen Attributwert kein zugehöriges Intervall ermittelt werden kann, wird er generell in weiß dargestellt.

### 5.3. Logische und arithmetische Ausdrücke in den Wenn-Dann-Regeln

Die Bildung der Ausdrücke erfolgt mit den üblichen Regeln und Symbolen. Beispiele für arithmetische Ausdrücke sind:

```
Z(I+1,0)
- 3.14 + I
0.5 * SIN (MAX (I,J))
3 + 2 * I + ROUND (3/J)
```

Das Ergebnis arithmetischer Ausdrücke ist immer eine Ganz- oder Dezimalzahl. Neben den Grundrechnungsarten (+, -, \*, /) stehen folgende Funktionen zur Verfügung:

<b>Funktion</b>	<b>Bedeutung</b>	<b>Beispiel</b>
ABS	Absolutbetrag	ABS (3.4)
ACOS	Arcuscosinus	ACOS (1)
ASIN	Arcussinus	ASIN (1)
ATAN	Arcustangens	ATAN (1)
CEIL	Aufrunden	CEIL (1.2) = 2
COS	Cosinus	COS (0)
EXP	Exponentialfunktion	EXP (2) = e "hoch" 2
FOCSE	Summe der Attributwerte aller 8 Nachbarn	FOCSE / 8
FOCSN	Summe der Attributwerte der 4 Nachbarn oben, unten, links, rechts	FOCSN / 4
FLOOR	Abrunden	FLOOR (2.7) = 2
I	Zeilennr. bzw. y-Koordinate	I * 2
J	Spaltennr. bzw. x-Koordinate	J + 2
LOG	Logarithmus zur Basis e	LOG (2.7)
MAX	Maximum zweier Werte	MAX (1,1)
MIN	Minimum zweier Werte	MIN (J,2)
POW	Potenz	POW (2,3) = 2 "hoch" 3
RANDOM	Gleichverteilte Zufallszahlen zwischen 0 und < 1	2 * RANDOM
RANDA	Wie RANDOM. Die Zufallszahl wird aber für die Bearbeitung der aktuellen Zelle gespeichert. Siehe Kap. 4.2.	RANDA
RANDB		
RANDC		
RANDD		
RANDE		
ROUND	Runden eines Werts	ROUND (3.1) = 3 ROUND (3.5) = 4
SIN	Sinus	SIN (3)
SQRT	Quadratwurzel	SQRT (2)
TAN	Tangens	TAN (3.14)
Z	Attributwert einer Zelle	Z (1,2) = Zeile 1, Spalte 2
ZIJ	Attributwert der aktuellen Zelle	3 * ZIJ

In logischen Ausdrücken werden die Symbole < (kleiner), <= (kleiner oder gleich), > (größer), >= (größer oder gleich), = (gleich), != (ungleich), ^ (nicht), | (oder) sowie & (und) verwendet. Das Ergebnis logischer Ausdrücke ist entweder wahr oder falsch. Beispiele für logische Ausdrücke sind:

```
Z(I+1,0) <= 0 & Z(0,J-1) >= 0
(I < 7 | J < 7) & (I > 0 | J > 0)
^ (I > 7 & J > 3)
```

Die Zeichen <, > und & müssen in XML als „&lt;“, „&gt;“ und „&amp;“ codiert werden.

Z.B. „3 <= I & J > 2“ wird zu: „3 &lt;= I &amp; J &gt; 2“

Im Prinzip können logische und arithmetische Ausdrücke gemischt werden. Ein arithmetischer Ausdruck mit einem Wert ungleich 0 wird in einem logischen Kontext als wahr interpretiert. Ist er gleich 0, dann hat er den logischen Wert falsch. Umgekehrt wird der logische Wert wahr als arithmetisch 1, der Wert falsch als arithmetisch 0 interpretiert.

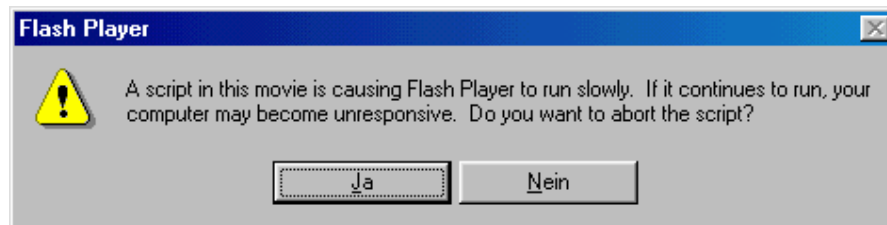
#### 5.4. Performanceprobleme

Die logischen und arithmetischen Formeln werden mittels ActionScript, das seinerseits wieder eine interpretative Sprache ist, interpretiert. Das führt auch bei Verwendung von wenigen Rasterzellen zu einem erheblichen Bedarf an Prozessorleistung, sodass sich der Übergang des Rasters von einem Zeitpunkt zum nächsten nur langsam vollzieht.

Im Folgenden wird die Übergangszeit zwischen zwei Zeitpunkten von den im Abschnitt 4 angeführten CA ausgewiesen. Als CPU stand ein AMD K6-III-Prozessor mit 450 MHz zur Verfügung. Der Hauptspeicher betrug 128 MB.

<b>CA</b>	<b>Rastergröße</b>	<b>Dauer in sec</b>
4.1. Waldbrand	10 x 10	26
4.2. Pflanzenpopulation	4 x 4	3 - 6
4.3. Verkehrssimulation	10 x 1	8 - 15

Gelegentlich erscheint im Rahmen einer zu hohen CPU-Belastung folgende Meldung, die mit NEIN zu quittieren ist:



## Literaturverzeichnis

### ALLOUCHE00

J.-P. Allouche, M. Courbage, G. Skordev: Notes on cellular automata, Institut für Dynamische Systeme, Universität Bremen, Report Nr. 458, 2000,  
<http://www.cevis.uni-bremen.de/CeVis/publications/2000.html>

### BEHME98

H. Behme, S. Mintert: XML in der Praxis, Bonn, 1998

### MOOCK01

C. Moock: ActionScript – The Definitive Guide, Sebastopol, 2001

### RECHENBERG85

P.Rechenberg, H. Mössenböck: Ein Compiler-Generator für Mikrocomputer, München, Wien, 1985

### RIEDL99

L. Riedl: Possible Cities – Simulation von Siedlungsentwicklung mit zellularen Automaten, Institut für Stadt- und Regionalforschung, TU Wien, 1999,  
[http://www.corp.at/beitraege/50\\_riedel\\_leop.pdf](http://www.corp.at/beitraege/50_riedel_leop.pdf)

### WEIMAR96

J. R. Weimar: Simulation with Cellular Automata, Lecture Notes, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, 1996,  
<http://www.tu-bs.de/institute/WiR/weimar/ZAscript/ZAscript.html>

### WIRTH84

N. Wirth: Compilerbau, Stuttgart, 1984

### WOLTER00

S.Wolter: Flash 5 – Mit ActionScript und Generator, Bonn, 2000