Inhaltsverzeichnis

[Vorwort 2](#_Toc321930851)

[Definition: Zelluläre Automaten 3](#_Toc321930852)

[Geschichte 3](#_Toc321930853)

[Meine Simulation 4](#_Toc321930854)

[Das Programm 4](#_Toc321930855)

[Quellen & Links 6](#_Toc321930856)

# Vorwort

Bevor ich zu der eigentlichen Facharbeit komme, hier einige Worte vorweg; Diese Facharbeit hat nicht den Anspruch zelluläre Automaten von Grund auf neu zu erfinden und zu erforschen, sie ist lediglich ein Versuch Zelluläre Automaten an einem Bespiel zu untersuchen und zu visualisieren. Die durchgeführte Simulation, stellt die Ausbreitung eines Waldbrandes anhand einer in Java geschriebenen Simulation dar. Die Simulation steht zum Download unter [L1]. Mit dieser Angabe, komme ich schon zu der Quellenangabe. Eine Quelle ist durch eine in eckigen Klammern geschriebene Zahlen- und Buchstabenkombination (z.B. Q5 für „Quelle 5“) angegeben. Die exakte Quelle oder der Link steht im Abschnitt „Quellen & Links“.

Dieses Dokument beginnt mit der Definition der Zellulären Automaten, danach folgt eine Einführung in die Idee und die Arbeitsweise meiner Simulation. Schließlich folgt eine Auswertung der Ergebnisse des Verhaltens.

# Definition: Zelluläre Automaten

Der Wikipedia Definition nach sind Zelluläre Automaten eine Methode zur „Modellierung räumlich diskreter dynamischer Systeme, wobei die Entwicklung einzelner Zellen zum Zeitpunkt *t+1* primär von den Zellzuständen in einer vorgegebenen Nachbarschaft und vom eigenen Zustand zum Zeitpunkt *t* abhängt.“ [Q1] Ein zellulärer Automat besteht aus mehreren Zellen in Raum und einer oder mehreren Vorschriften die die Entwicklung bestimmten.

Jede Zelle befindet sich in einem aus einer bestimmten Anzahl von Zuständen (z.B. „An“ und „Aus“). Zusätzlich hat sie eine *Nachbarschaft* die häufig auch sie selbst einschließt und relativ von ihr besteht (z.B. „alle angrenzenden Zellen“). Jeder Zelle wird ein Anfangsstatus zugeordnet, der zum Zeitpunkt *t=0* gilt. Mit dem Fortschreiten der Zeit werden die Zellzustände den spezifizierten Regeln nach verändert.

Der Raum ist durch eine endliche Zahl von Dimensionen bestimmt und kann *endlich* oder *unendlich* sein.

## Geschichte

1940 arbeitete John von Neumann in Los Alamos an selbst-reproduzierenden Systemen. Zur gleichen Zeit untersuchte sein Kollege Stanislaw Ulam das Wachstum von Kristallen. Von Neumann’s beruhte auf der Idee, dass ein Roboter einen anderen baut. Als er daran arbeitete erkannte er die Schwierigkeit einen solchen Roboter zu bauen und ihn mit genügend Teilen zu versorgen. Ulam empfahl ihn sein System mathematisch zu abstrahieren so wie auch Ulam es tat. Der erste Zelluläre Automat war geboren. Der vorgestellte Automat hatte 29 Zustände und konnte sich immer wieder selbst reproduzieren. Dieses Design ist bekannt als „tesselation model“ (dt. Parkettierung).

1969 veröffentlichte Korad Zuse sein Buch „Rechnender Raum“. In diesem nimmt er an dass alle Naturgesetze bestimmten Regeln unterworfen ist und das gesamte Universum als Zellulärer Automat gesehen werden kann.

1970 wurde das von John Horton Conway entwickelte „Game of Life“ vorgestellt. Die Regeln lauteten: Hat eine Zelle zwei Nachbarn behält sie ihren Zustand. Hat sie drei Nachbarn wird sie schwarz. Liegen andere Zustände vor wird sie weiß. Trotz dieser einfachen Regeln ergeben sich viele verschiedene Verhaltensweisen zum Beispiel *Gleiter*, Anordnungen die sich von selbst durch den Raum bewegen.

1983 veröffentlichte Stephen Wolfram grundlegende Arbeiten zu „elementary cellular automata“ (dt. elementare Zellulären Automaten). Die Komplexität die diese simplen Regeln zeigen, ließen ihn annehmen, dass die Natur durch ähnliche Mechanismen gesteuert wird.

*Quelle: Wikipedia [Q1 & Q2]*

# Meine Simulation

Wie bereits erwähnt, stellt meine Simulation die Ausbreitung eines Brandes in einem Wald dar. Im Folgenden werde ich die Wirkungsweise des von mir entwickelten Programms und des dahintersteckenden Algorithmusses erläutern.

## Das Programm

Da ich das Programm mit der „lwjgl“ und „slick“ gebaut habe, waren die Grundstrucktur bereits gegeben. Die Klasse ist folgendermaßen definiert:

**public** **class** simulation **extends** BasicGame**{**

Es folgen die leeren Methodenbehälter:

**public** simulation**()** **{**

**}**

**public** **void** init**(**GameContainer gc**)** **throws** SlickException **{**

**}**

**public** **void** update**(**GameContainer gc**,** **int** delta**)** **throws** SlickException **{**

**}**

**public** **void** render**(**GameContainer gc**,** Graphics g**)** **throws** SlickException **{**

**}**

Wie die Namen erahnen lassen, ist das erste der Konstruktor. Die zweite Methode wird einmal beim Starten des Programms aufgerufen. Die anderen Beiden Methoden werden immer nacheinander aufgerufen. Erst updatezur Berechnung des Ablaufes und dann render zum Zeichnen auf den Bildschirm.

Zuerst werden globale Variablen initialisiert.

**public** Image white**,**red**,**green**;**

**public** **int[][]** field**;**

**public** **int** width**,**height**,**scale**,**spread**;**

**public** **float** prob**;**

**public** TextField outputField**;**

Da die Simulation mit verschiedenen Parametern aufrufbar sein soll, ist dem Konstruktor eine Reihe von Parametern gegeben.

**public** simulation**(int** width**,** **int** height**,** **int** scale**,** **int** spread**,** **float** prob**)**

**{**

**super(**"simulation"**);**

**this.**width **=** width**;**

**this.**height **=** height**;**

**this.**scale **=** scale**;**

**this.**spread **=** spread**;**

**this.**prob **=** prob**;**

**this.**field **=** **new** **int[**width**][**height**];**

**}**

Hier wird der Konstruktor der Klasse BasicGame aufgerufen und die anfangs erstellten Variablen mit Werten aus den Parametern gefüllt. Zuletzt wird ein 2-Dimensionales Array erstellt, welches das Feld darstellt.

Danach wird die Methode init() bestimmt. Diese wird beim Start einmal aufgerufen, sie ist also ideal um die Bilder die später als kleine Punkte dargestellt werden zu laden und das Feld Array mit Zahlen zu füllen. Hier ist anzumerken, dass die Bilder nur 20x20 große, einfarbige Flächen sind. In „slick“ ist es schneller Bilder als Bitmaps in den Speicher zu legen und von dort aus zu zeichnen, anstatt die Farben selbst zu zeichnen.

**public** **void** init**(**GameContainer gc**)** **throws** SlickException **{**

**this.**white **=** **new** Image**(**"img/white.png"**);**

**this.**red **=** **new** Image**(**"img/red.png"**);**

**this.**green **=** **new** Image**(**"img/green.png"**);**

**int** i**,**j**;**

**for** **(**i **=** 0**;** i **<** **this.**width**;** i**++)**

**{**

**for** **(**j **=** 0**;** j **<** **this.**height**;** j**++)**

**{**

**if** **(**Math**.**random**()** **>** 1 **-** **this.**prob**)** **{**

**this.**field**[**i**][**j**]** **=** 2**;**

**}**

**else** **{**

**this.**field**[**i**][**j**]** **=** 1**;**

**}**

**}**

**}**

**}**

Zunächst werden drei weitere globale Variablen mit ihrem Namen entsprechenden Bildern/ Farben gefüllt. Darunter werden zwei Zählvariablen für die for Loops initialisiert. Die Loops an sich füllen dann das Feld Array. Mit der gegeben Wahrscheinlichkeit starten die Zellen entweder im Zustand 2 oder 1. An dieser Stelle wäre eine Übersicht über die Zustände angebracht.

|  |  |
| --- | --- |
| 0 | Es steht kein Baum an dieser Stelle. Die Zelle ist weiß. |
| 1 | Es steht ein Baum an dieser Stelle. Die Zelle ist grün. |
| 2 | Der Baum in dieser Position steht in Flammen. Im nächsten Durchlauf wird er angebrannt sein. Die Zelle ist rot. |

Die Wahrscheinlichkeit wird folgendermaßen berechnet: Die Funktion Math.random() gibt eine zufällige Zahl zwischen 0 und 1 zurück. Die Variable this.prob enthält die angegebene Wahrscheinlichkeit. Es wird überprüft ob Math.random() größer ist als 1 - this.prob. Somit lässt sich die Wahrscheinlichkeit präzise festlegen. Trifft diese Wahrscheinlichkeit nicht zu, wird der Zustand 1 in das Feld geschrieben.

Nach der Methode init() folgt die Methode update(). Diese ist dafür zuständig die Berechnungen bzw den Algorithmus auszuführen. Diese wird im Abschnitt „Der Algorithmus“ weiter erklärt. Nach der Methode update() folgt render(). Diese Methode ist dafür zuständig, das Feld auf den Bildschirm zu zeichnen.

**public** **void** render**(**GameContainer gc**,** Graphics g**)** **throws** SlickException **{**

**int** i**,**j**;**

**for** **(**i **=** 0**;** i **<** **this.**width**;** i**++)**

**{**

**for** **(**j **=** 0**;** j **<** **this.**height**;** j**++)**

**{**

**switch** **(this.**field**[**i**][**j**])**

**{**

**case** 1**:**

**this.**green**.**draw**(**i **\*** **this.**scale**,**j **\*** **this.**scale**,this.**scale**);**

**break;**

**case** 2**:**

**this.**red**.**draw**(**i **\*** **this.**scale**,**j **\*** **this.**scale**,this.**scale**);**

**break;**

**case** 0**:**

**this.**white**.**draw**(**i **\*** **this.**scale**,**j **\*** **this.**scale**,this.**scale**);**

**break;**

**}**

**}**

**}**

**}**

Quellen & Links

Q1: http://de.wikipedia.org/wiki/Zellul%C3%A4rer\_Automat (10.04.2012)

Q2: http://en.wikipedia.org/wiki/Cellular\_automaton (10.04.2012)

L1: http://download.jangxx.com/Random\_Files/facharbeit/simulation/