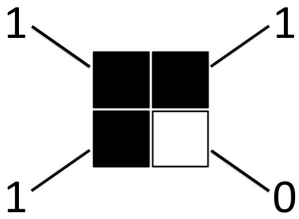


Zelluläre Blockautomaten mit Margolus-Nachbarschaft

Eine medienarchäologische Einladung

Humboldt-Universität zu Berlin
Institut für Medienwissenschaft
Masterstudiengang Medienwissenschaft
5. Fachsemester

Seminar: Schach dem Computer. Theorie, Geschichte und Implementierung des Computerschach (WS 2015/16)
bei Dr. Stefan Höltgen und Nikita Braguinski

Thomas Nückel
Käthe-Niederkirchner-Str. 28
10407 Berlin
thomas.nueckel@hu-berlin.de

Gliederung

1. Ziel der Arbeit

Zelluläre Automaten wurden in den Achtziger Jahren als revolutionäre Spezial- oder Universalwerkzeuge angepriesen, um dann wieder aus dem Mainstreamdiskurs zu verschwinden. Ob zu recht oder unrecht, darüber wurde schon zu jener Zeit gestritten. Heute kommen zelluläre Automaten immer noch in einigen Disziplinen zur Anwendung. Im Großen und Ganzen aber ist es still um sie geworden.

Diese Arbeit hat nicht zum Ziel, an diesem Umstand zu rütteln. Stattdessen soll sie Fragen nach dem epistemischen Status von zellulären Automaten aufwerfen. Dem Leser soll dabei keine endgültige Antwort, sondern die Möglichkeit gegeben werden, sich dieser Frage selbst zu widmen. Insofern hat diese Arbeit eine didaktische Ausrichtung:
Programme wie Golly umfassen eine ungeheure Menge an zellulären Automaten, verlieren aber irgendwann an Faszination, wenn die Begeisterung über die flimmernden Bilder in Überdruss umschlägt und die Motivation schwindet, sich die nächsten Beispiele anzusehen, ohne tiefer in die Materie einsteigen zu können. Diese Arbeit wurde nach umgekehrtem Vorzeichen konzipiert: Weit davon entfernt, so etwas wie eine Sammlung von zellulären Automaten zu sein, konzentriert sie sich auf eine einzige Familie von Zellularautomaten: Die elementaren Blockautomaten mit Margolus-Nachbarschaft. Anstatt diese Blockautomaten aber nur hermeneutisch zugänglich zu machen, wird hier ein technischer Apparat in Form eines Programms mitgeliefert, mit dem diese Automaten experimentell und in Echtzeit untersucht werden können. In guter medienarchäologischer Manier bildet den Kern dieser Arbeit daher ein ausführbares und vom Leser veränderbares Python-Programm, mit dem die Familie der elementaren Blockautomaten mit Margolus-Nachbarschaft implementiert werden kann. Da diese Familie immerhin 16^{16} Automaten umfasst, scheidet ein stumpfes Durchklicken der Automaten aus und es ist umso wichtiger, dem Leser in praktischer und theoretischer Hinsicht Zugang zum Programm zu geben. Im Sinne dieser Forderung wurde das Programm auf Verständlichkeit und Beherrschbarkeit auch und gerade für Neulinge in Sachen Programmierung ausgelegt. Auch solche Leser sollen in die Lage versetzt werden, das Programm selbst auf den Prüfstand zu stellen, zu analysieren und gegebenenfalls weiter zu entwickeln. Damit sieht sich diese Arbeit als eine Ergänzung zu Programmen wie Golly.

Auf Seiten des Autors entstand schon während der Arbeit die wahrscheinlich nur zu berechtigte Scheu, die eigene Tätigkeit als Programmieren *zu* bezeichnen. Zu offensichtlich wurde bei dieser Tätigkeit gegen zwei elementare Regeln zumindest jedes guten Programmierens verstoßen: Erstens wird es sich unnötig schwer gemacht. Es wurden umständlich Codezeilen geschrieben für Prozesse, die mit viel weniger Zeichen gestaltbar gewesen wären. Zweitens, und das ist gravierender, verstößt diese Arbeit gegen den Grundsatz, keinen Code zu übernehmen, den der Programmierer nicht verstanden hat. Genau das aber ist geschehen bei der Implementierung der graphischen Ausgabe.

Zumindest der erste dieser Verstöße war intendiert: Um das Programm möglichst verständlich zu halten, wurde nicht nur auf die Verwendung von kürzeren aber opaken Anweisungen verzichtet. Der Code wurde auch von Subroutinen freigehalten, so dass er linear gelesen werden kann. Die Lesbarkeit des Codes für Programmieranfänger hatte Vorrang vor jeder Eleganz.

Was den zweiten Verstoß betrifft: Vielleicht lässt sich jede Betrachtung eines hochtechnischen Mediums, wie es das hier vorgestellte Programm im Vollzug ist, nur auf Kosten von blinden Flecken an anderer Stelle erreichen. Nun ist der offensichtlichste blinde Fleck dieser Arbeit der genannte Vorgang der graphischen Ausgabe durch das Programm. Dieser Umstand wird in Kauf genommen, da eine Erhellung dieses Punktes keinen nennenswerten Gewinn für das Ziel dieser Arbeit mit sich gebracht hätte: Fragen nach dem epistemischen Status von zellulären Automaten zu exemplifizieren.

Die Experimente mit dem Programm und den Blockautomaten sollen im Sinne der oben genannten Forderung nicht im theorieleeren Raum stattfinden, sondern medien-theoretisch und medienhistorisch verortet werden. Dass diese Verortungen dabei nur Angebote und eine Auswahl darstellen können, liegt nicht zuletzt in der gebotenen Kürze dieser Arbeit begründet. Die vorgebrachten Beispiele und Überlegungen sind als mögliche Zündfunken für eine Auseinandersetzung mit Fragen nach dem epistemischen Status von zellulären Automaten zu verstehen, keinesfalls als erschöpfende Betrachtung oder hermetisch geschlossenes Ganzes. Brüche zwischen den einzelnen Kapiteln sind intendiert.

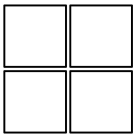
Zu Beginn wird in dieser Arbeit eine kurze Einführung in die Funktionsweise der Blockautomaten mit Margolus-Nachbarschaft gegeben, um zu verdeutlichen, worum es sich bei diesen Automaten handelt (Kapitel 2). Darauf folgt die Darstellung des historischen Versuchs, zelluläre Automaten als alternative Rechnerarchitekturen in Hardware zu implementieren. Daran wird sich zeigen lassen, welche Vorteile Blockautomaten gegenüber normalen Automaten aufweisen, und wie sich diese bei der Realisierung solcher Automaten als tatsächliche Hardware niederschlagen (Kapitel 3). Die damit angeschnittene Geschichte der zellulären Automaten ab der zweiten Hälfte der Achtziger Jahre wird im folgenden Abschnitt zur Grundlage einer medientheoretischen Überlegung zum Erfolg der zellulären Automaten als hochtechnische Medien. (Kapitel 4). Auf den so theoretisch gerüsteten Leser wartet dann ein Close-Reading des Programmcodes, das zur Handhabung des Programms befähigen soll und das Kernstück dieser Arbeit bildet (Kapitel 5). Damit die Handhabung des Automaten direkt an interessanten Beispielen vollzogen werden kann, wird der Beschreibung des Programms eine Sammlung von ausgewählten Blockautomaten nachgestellt (Kapitel 6). Vor diesem Hintergrund wird die Frage nach dem epistemischen Status dieser Automaten noch einmal explizit aufgegriffen, um die Einladung zur Entwicklung eigener Fragestellungen abzurunden (Kapitel 7). Zwei BASIC Listings mit Blockautomaten schließen als letztes Kapitel diese Arbeit ab (Kapitel 8).

Die Hoffnung ist, dass das Verständnis und die Handhabung des mitgelieferten Programms es dem Leser ermöglichen, eigene Schlüsse über den Zusammenhang von tatsächlichen physikalischen Vorgängen und deren Modellierung durch zelluläre Automaten zu ziehen. In diese Sinne ist diese Arbeit eine Einladung zur *Auseinandersetzung*, in dem Sinne, sich die Sache selbst *auseinander zu setzen*: Sie zu zerlegen, zu analysieren, und so – wenn auch mit Maus und Tastatur – buchstäblich begreifend nachzuvollziehen. Ob und inwiefern diese Arbeit den hier skizzierten Zielen auch nur nahe kommt, darüber darf und kann selbstredend nur der Leser urteilen.

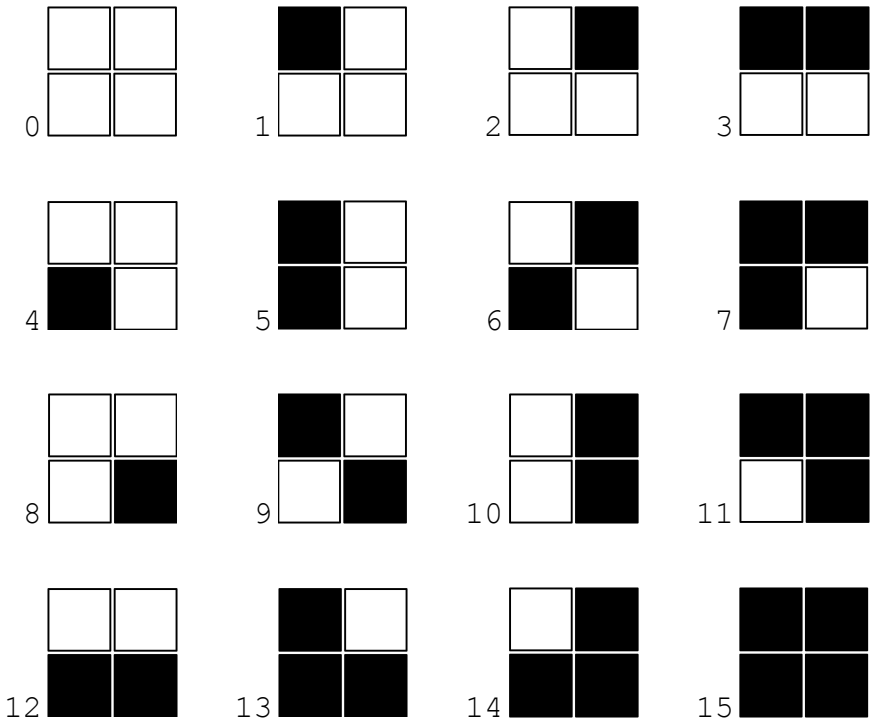
2. Aufbau und Funktionsweise

2.1 Blockstruktur

Wie andere zelluläre Automaten bestehen Blockautomaten aus einer Gitterstruktur, die das Spielfeld in diskrete Zellen unterteilt. Im Unterschied zu Automaten wie Game of Life aber werden in Blockautomaten pro Generationsschritt nicht einzelne Zellen, sondern jeweils ganze Zellblöcke berechnet:



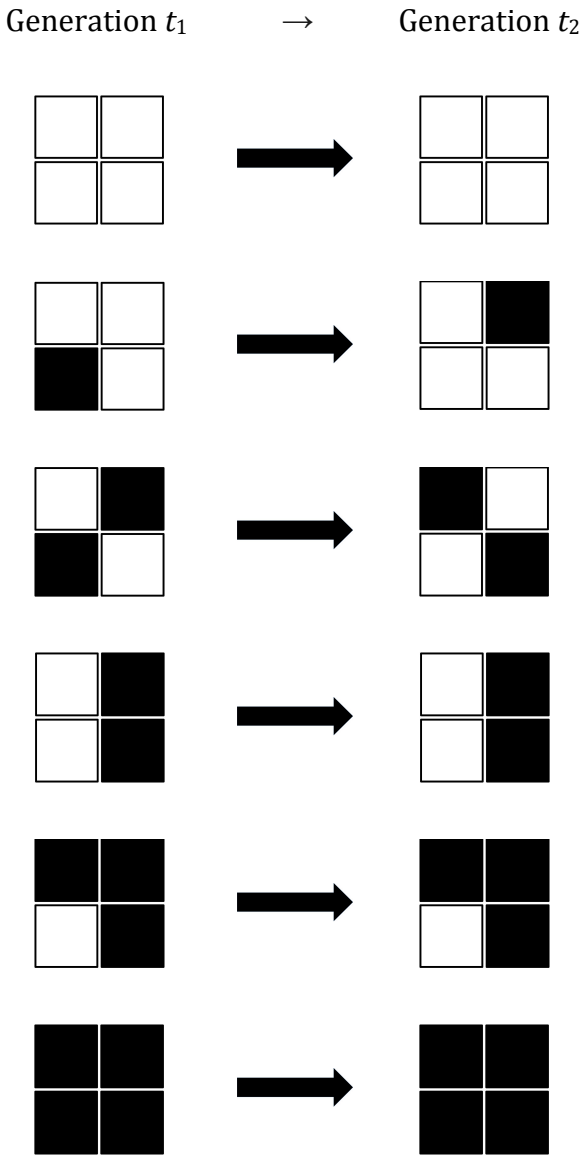
Dieses Blockschema bildet die Basis der Margolus-Nachbarschaft, benannt nach Norman H. Margolus, der als erster ihre Eigenschaften untersuchte.[1] Ein solcher Block enthält jeweils 4 binäre Zellen, also 4 bit Information. Unter Ausschluss der Punkt- und Achsensymmetrien zwischen den Blöcken gibt es daher in der Margolus-Nachbarschaft $2^4 = 16$ Möglichkeiten, diese mit Schwarz und Weiß, mit lebenden und toten Zellen zu füllen:



[1] Alternative Nachbarschaften für Blockautomaten sind bspw. die Q*Bert Neighbourhood und die Star of David Neighbourhood. Vgl. dazu bei Interesse die Verweise im Anhang auf die Internetpräsenz von Tyler.

2.2 Regel für den Generationswechsel

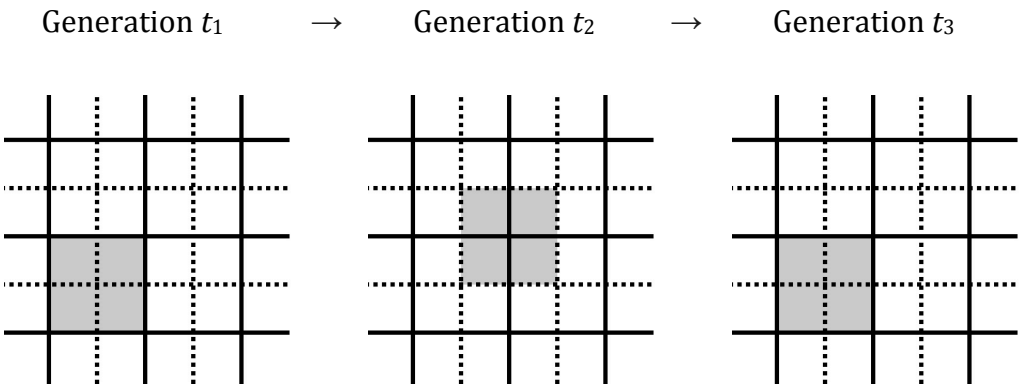
Bei jedem Generationswechsel ($t_1 \rightarrow t_2$) in einem vollständig determinierten zellulären Automaten geht der Zustand jeder Zelle in einen regeltechnisch exakt festgelegten nächsten Zustand über. Bei Blockautomaten heißt das dementsprechend, dass jeder Block in einen ganz bestimmten anderen Block übergeht:



Die als Beispiel abgebildete Regel nutzt den Umstand, dass bestimmte Blöcke zueinander rotationssymmetrisch sind. D. h. die Zuordnungen für die hier fehlenden zehn Blöcke lassen sich durch identische Rotation der Blöcke in t_1 und t_2 ablesen. Darum kann auf eine Auflistung aller 16 Blöcke verzichtet werden. Rotationssymmetrische Regeln bilden allerdings nur eine Untergruppe aller möglichen Regelsätze.

2.3 Das Alternieren der Gitter

Ein Programm, das nur die bisher beschriebenen Strukturen und Regeln beinhaltet, zeigt aus epistemologischem wie auch aus ästhetischem Blickwinkel kaum interessantes Verhalten. Noch fehlt denn auch die entscheidende Zuspitzung: Diese besteht darin, dass bei jedem Generationswechsel die Lage der berechneten Blöcke horizontal und vertikal um eine Zelle verschoben wird. In Abbildungen wird dies deutlich gemacht, indem die Linien des Gitters abwechselnd mit durchgezogenen und durchbrochenen Strichen ausgeführt werden. So ergeben sich zwei versetzt aufeinander liegende Gitter:[2]



Das durchgezogene Gitter ist aktiv

Das durchbrochene Gitter ist aktiv

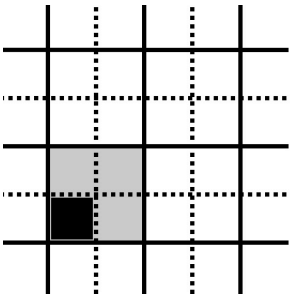
Das durchgezogene ist wieder aktiv, usw.

Wie die grauen Blöcke illustrieren, bilden sich je nach angelegtem Gitter verschiedene Blöcke der Margolus-Nachbarschaft: Einmal die Blöcke, die von den durchgezogenen Linien umgrenzt werden (t_1 und t_3), und das andere Mal die Blöcke, die von den durchbrochenen Linien umgrenzt werden (t_2). Welches Gitter gerade angesteuert wird, welche Blöcke also analysiert und berechnet werden, ist mit dem Wechsel der Generationen in der Zeit synchronisiert. Dieser zeitkritische Wechsel des Gitters ist das wesentliche Element zur Dynamisierung des Blockautomaten: Ohne ihn wäre die Information für immer in ihrem jeweiligen Block gefangen. Das Alternieren der Gitter ermöglicht erst den Informationsaustausch über das Spielfeld.

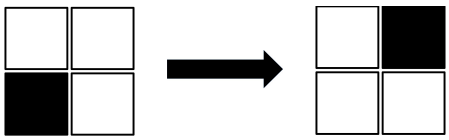
[2] So z. B. in Margolus 1984, 88-91, dort werden die Linien mit ‚solid' und ‚dotted' bezeichnet. Margolus/Toffoli 1990, 241 verwenden ‚thick' und ‚thin' als Bezeichnung und graphisch entsprechend dicke und dünne Linien. Tyler dagegen verwendet in seiner Erläuterung der Margolus-Nachbarschaft auf cell-auto.com zur Illustration wieder durchgezogene und durchbrochene Linien (s. Literaturverzeichnis).

2.4 Die Reise einer Zelle

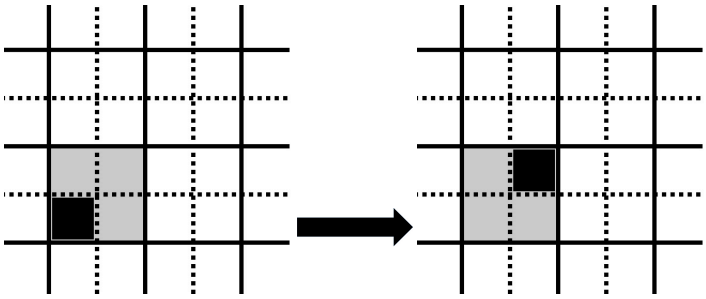
Dass die alternierende Verschiebung des Gitters eine Dynamisierung des Blockautomaten ermöglicht, soll am Beispiel einer Zelle greifbar gemacht werden, die über das Spielfeld reist. Zur Generation t_1 liegt im Gitter des zellulären Automaten eine einzelne Zelle:



In dieser Generation ist das durchgezogene Gitter aktiv. Nun wird der entsprechende Regelsatz angewandt. In diesem Beispiel soll das die in Kapitel 2.2 dargestellte rotationssymmetrische Regel sein. Demnach gilt folgende Zuordnung:

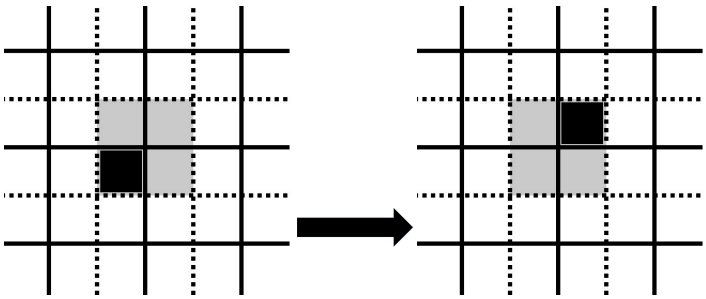


Da der linke Block der Margolus-Nachbarschaft durch den rechten ersetzt wird, wandert die Zelle von der Generation t_1 zur nächsten t_2 von links unten nach rechts oben:

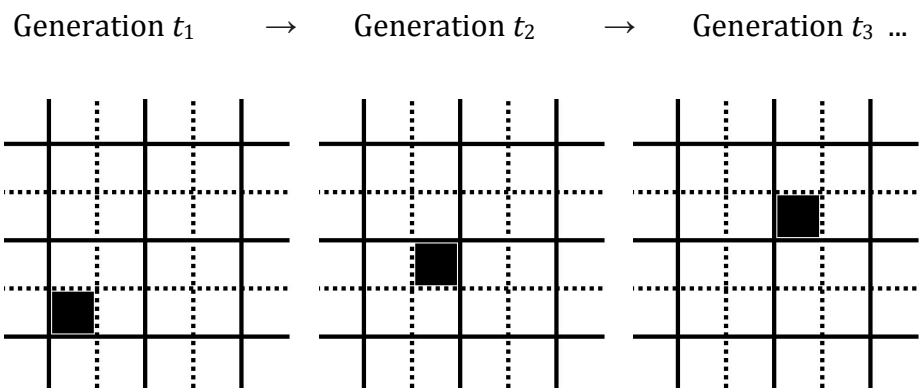


Würde nun im nächsten Generationsschritt wieder der selbe Block berechnet, so wanderte die eben nach rechts oben gelangte Zelle an die soeben verlassene Stelle zurück (aufgrund der Rotationssymmetrie). Für alle möglichen Regelsätze, nicht nur für die rotationssymmetrischen, würde dann gelten: Die Zelle könnte im Block wandern, sich vermehren, sterben – aber sie könnte ihn niemals verlassen.

Stattdessen aber wird das Gitter gewechselt. Die durchbrochenen Linien sind nun aktiv. Deshalb wird auch ein anderer Block berechnet. Dieser liegt im Vergleich zum vorhergehenden Block vertikal und horizontal um eine Zelle verschoben:



Durch die Gitterverschiebung liegt die Zelle jetzt wieder links unten im Block, so dass die selbe Regel wie eben zum Tragen kommt: Die Zelle wandert wieder nach rechts oben. In diesem Schritt schon hat die Zelle den Bereich verlassen, in den sie ohne das alternierende Gitter eingesperrt geblieben wäre. Solange sie nun auf kein Hindernis trifft, wird sie ihre Richtung beibehalten und immer weiter ins Unendliche reisen:



Obwohl also in jeder Generation isolierte Blöcke analysiert und berechnet werden, findet in Blockautomaten ein globaler Informationsaustausch statt. Das Alternieren der versetzt liegenden Gitter transzendiert den Ereignishorizont einer Zelle dahingehend, dass sie diesen jedes mal neu festlegt und so Dynamik erst ermöglicht. Genau genommen bildet so erst das Zusammenkommen der drei genannten Bedingungen – die Viererblöcke, der Regelsatz und das zeitkritische Alternieren der Gitter – den Blockautomaten mit Margolus-Nachbarschaft.[3]

[3] Für eine weitergehende Beschreibung der Margolus-Nachbarschaft sei verwiesen auf Margolus/Toffoli 1987b, 119–124.

3. Blockautomaten als Hardware: Die CAM-7

3.1 Vorteile von Zellularautomaten

Norman H. Margolus, dem zu Ehren die Margolus-Nachbarschaft als solche benannt wurde, arbeitete von 1980 bis 1995 mit Tommaso Toffoli bei Edward Fredkin in der Information Mechanics Group am MIT Laboratory for Computer Science. Margolus und Toffolis Interesse galt der Untersuchung der physischen Basis von Rechenprozessen – insbesondere von reversiblen Prozessen – und der rechenbasierten Modellierung von physikähnlichen Systemen (Margolus/Toffoli 1987a, 967). Ihr Ziel war mithin die Erforschung der Verbindungen und Grenzen zwischen Physik und Berechnung. Für diesen Zweck schienen sich zelluläre Automaten in besonderer Weise anzubieten, haben sie doch mit dem Verhalten von physikalischen Systemen, beispielsweise der Bewegung der Partikel in einem Fluid drei grundlegende Eigenschaften gemein:
Parallelität: Alle Prozesse in zellulären Automaten laufen parallel ab, d. h. ideal gesehen werden alle Zellen gleichzeitig neu berechnet. Sie wirken also wie die Partikel eines realen Fluids gleichzeitig aufeinander ein und nicht nacheinander.
Lokalität: Entscheidend für die Berechnung des neuen Wertes einer Zelle sind nur die in einer festgelegten Reichweite liegenden Nachbarn, zumeist die sogenannten nächsten Nachbarn. Im Falle der Von-Neumann-Nachbarschaft wären das beispielsweise nur die vier orthogonal angrenzenden Nachbarn, im Falle der Moore-Nachbarschaft kommen noch die vier Nachbarzellen auf den Diagonalen hinzu. Lokalität gilt auch für die Partikel des Fluids, deren Verhalten – *ceteris paribus* – nur durch die Bewegungen der direkt kollidierenden Partikel verändert wird (Margolus 1987a, 968).
Homogenität bzw. Uniformität: Für alle Zellen gelten dieselben Gesetze, unabhängig von ihrer absoluten oder relativen Platzierung im Gitter (Margolus 1987b, 145).

> It is, of course, exactly this same property of being physics-like that makes CA a natural tool for physical modeling (e. g., fluid behavior). Von Neumann-architecture machines emulate the way we consciously think: a single processor that pays attention to one thing at a time. CA emulate the way nature works: local operations happening everywhere at once. (Margolus; Toffoli 1987, 968).

Die fragwürdige Auffassung von Denken und Natur sowie die Frage, ob es hier tatsächlich um ein Emulieren oder ein Simulieren geht, einmal dahingestellt, so bleibt es doch richtig, dass Computer der Von-Neumann-Architektur die drei genannten

Eigenschaften der Parallelität, der Lokalität und der Homogenität nicht von Haus aus aufweisen, sondern diese erst simulieren müssen – sofern sie nicht wie heute selbst mit mehreren Prozessoren ausgestattet werden. Gerade zur Simulation von Fluiden bieten sich diskrete zelluläre Automaten darum in besonderer Weise an, wie Yves Pomeau, ein Pionier in der Simulation von physikalischen Systemen mittels fester Gittermodelle, erklärte:

> [...] [T]wo fluids with quite different microscopic structures can have the same macroscopic behavior because the form of the macroscopic equations is entirely governed by the microscopic conservation laws and symmetries. [...] [S]uch observations have led to a new simulation strategy for fluid dynamics: fictitious microworld models obeying discrete cellular automata rules have been found, such that two- and three-dimensional fluid dynamics are recovered in the macroscopic limit. (Pomeau et al. 1987, 650)

Solche Ergebnisse bildeten das argumentative Fundament für die Relevanz von zellulären Automaten bei der Untersuchung physikalischer Prozesse – was in Kapitel 7 noch einmal zur Diskussion kommen wird. Margolus und Toffolis Ansatz ging darüber hinaus, auf Von-Neumann-Maschinen implementierte zelluläre Automaten für ein gut geeignetes Werkzeug neben anderen zu halten. Wie sie unmissverständlich darlegten, waren sie der Meinung, mit den zellulären Automaten eine alternative Rechnerarchitektur gefunden zu haben:

> The advantages of an architecture optimized for cellular automata (CA) simulations are so great that, for large-scale CA experiments, it becomes absurd to use any other kind of computer. (Margolus/Toffoli 1987a, 967)

Dementsprechend gingen Margolus und Toffoli daran, die oben genannten Vorteile der zellulären Automaten bei der Simulation von physikalischen Systemen – dass sie über die Eigenschaften der Parallelität, der Lokalität und der Homogenität verfügen –, in Hardware umzusetzen. Rechnerhardware also auf der Basis von Zellularautomaten, die, in gleicher Weise universal wie die Computer der Von-Neumann-Architektur, dennoch große Vorteile bei der Umsetzung physikalischer Modelle als berechenbare Vorgänge bieten sollte. Unter diese Prämisse wurde eine ganze Reihe von *Cellular Automata Machines* (CAMs) enworfen. Eine dieser CAMs, die CAM-7, war nun als Blockautomat mit Margolus-Nachbarschaft geplant. Diese CAM-7 verdient besonderes Interesse, da an ihr verdeutlicht werden kann, welche Vorteile Blockautomaten gegenüber gewöhnlichen zellulären Automaten haben, sobald es um eine Umsetzung in Hardware geht. Im Folgenden wird zuerst ein Blick auf das Vorgängermodell, die CAM-6, geworfen, bevor es um die CAM-7 und die Blockautomaten als Hardware geht.

3.2 Der Vorgänger: Die CAM-6

Die Celluar Automata Machine-6 wurde als Hardware gebaut und kam auch zur Serienreife. Die CAM-6 war ausgelegt für die Berechnung von traditionellen Zellularautomaten mit Von-Neumann- oder Moore-Nachbarschaft, nicht für Blockautomaten. Ihr Aufbau orientierte sich also an solchen zellulären Automaten, in denen jede Zelle durch eine Funktion über mehrere Zellen berechnet wird:

$$a_{x,y}^{(t)} = \mathbf{F}[a_{x,y}^{(t-1)}, a_{x-1,y}^{(t-1)}, a_{x+1,y}^{(t-1)}, a_{x,y-1}^{(t-1)}, a_{x,y+1}^{(t-1)}] \qquad (1)$$

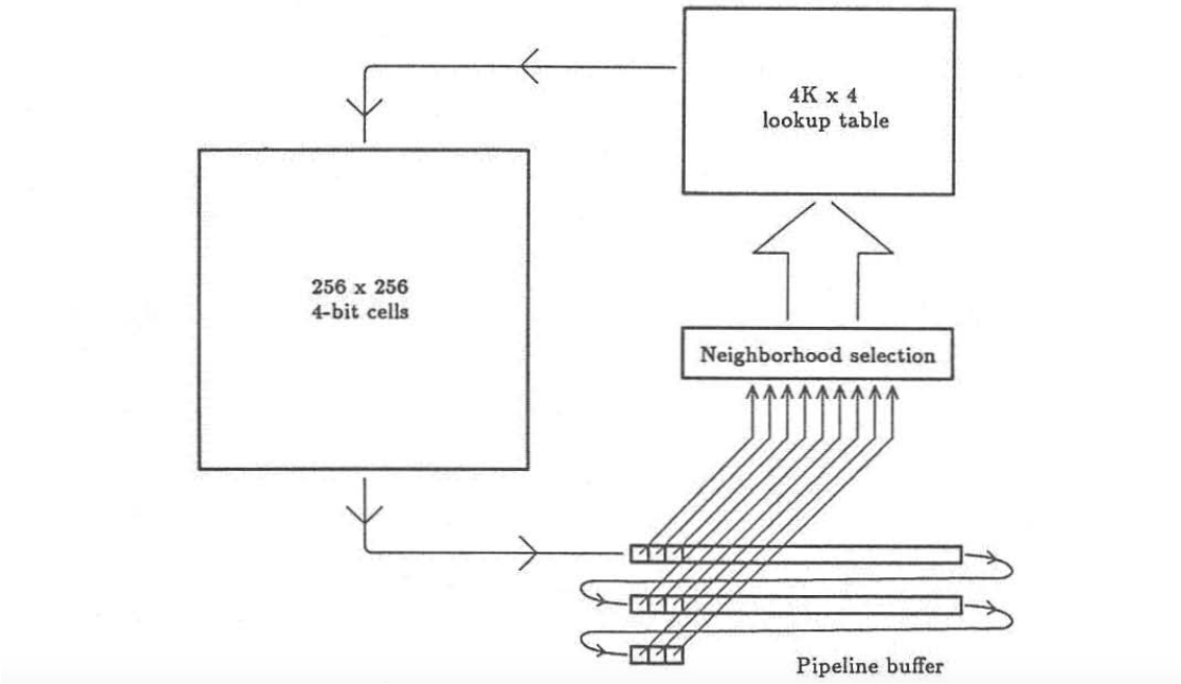
Diese Formel (1) weist der Zelle *a* an der Position *x*, *y* zum Zeitpunkt *t* (die linke Seite der Gleichung) eine Funktion **F** (die rechte Seite) zu. Die Funktion **F** bestimmt den neuen Wert der Zelle *a*. Dazu werden der Funktion **F** Werte aus der vorherigen Generation *t*-1 eingegeben, und zwar die Werte der Zelle *a* selbst, sowie ihres linken, rechten, oberen und unteren Nachbarn. Die abgebildete Formel deckt also die Von-Neumann-Nachbarschaft ab und wäre für die Moore-Nachbarschaft entsprechend zu erweitern.[4]

In der die CAM-6 mussten die Zellen – wie in den damaligen Computern der Von-Neumann Architektur – sequentiell abgearbeitet werden. Mehr noch: Um die nächste Generation von Zellen korrekt berechnen zu können, war eine Verdopplung des Zell-Datensatzes notwendig. Jeder, der sich mit der Implementierung eines Game of Life beschäftigt hat, kennt dieses Problem: Schreibt man die neuen Werte der Zellen direkt in das noch auszulesende Feld, so werden dabei auch Zellen verändert, deren Wert noch für die Berechnung anderer Zellen notwendig gewesen wäre – das Ergebnis wird verfälscht. Dies ist auch der Grund dafür, warum für Game of Life, selbst wenn es nur mit Spielsteinen auf Papier oder Boden gespielt wird, zumeist zusätzliche andersfarbige Marker gesetzt werden: Um nicht durch die Veränderung der Zellwerte noch benötigte „Spielstände“ zu überschreiben.

Während dieses Problem in einem echten Parallelrechner nicht auftreten würde, muss für jeden sequentiell arbeitenden Computer eine Lösung gefunden werden. Ganz so, wie auf dem Papier zusätzliche Marker gesetzt werden, besaß auch die CAM-6 einen speziellen Puffer, in dem die Zellwerte der alten Generation gespeichert werden konnten, während die neuen Zellen direkt in das 256 x 256 Zellen-Feld eingeschrieben

[4] Vgl. zu der in der Formel verwendeten Nomenklatur Wolfram 1984, 2

wurden. Die folgende Graphik zeigt die wesentlichen Elemente der CAM-6. Es wird deutlich sichtbar, wie sich das oben geschilderte Problem in die Hardware eingeschrieben hat:



Schematischer Aufbau der CAM-6 (Margolus/Toffoli 1987a, 971)

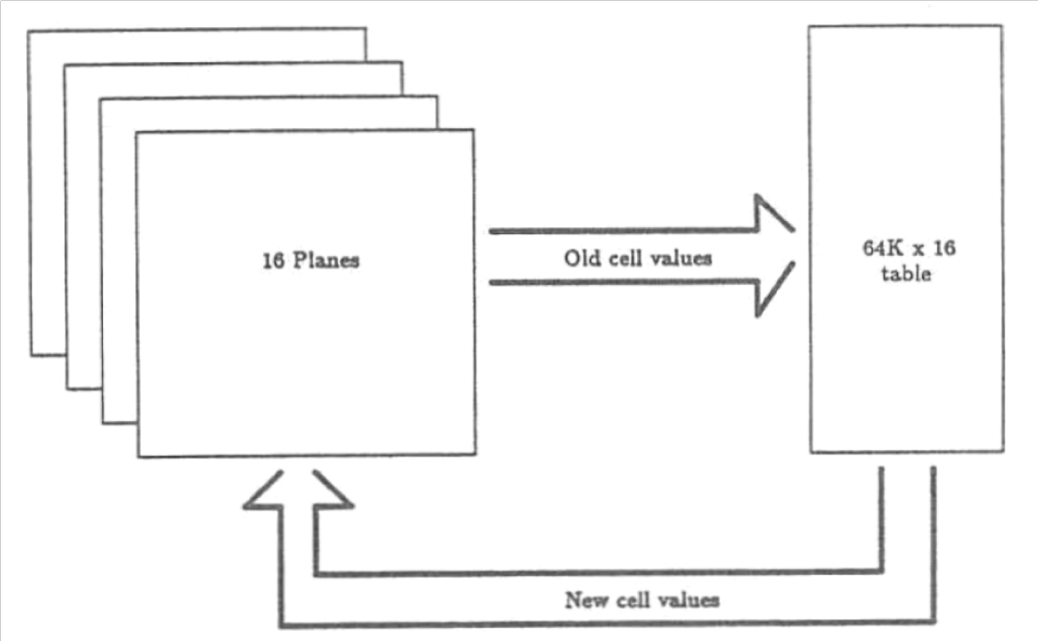
In jeder Generation gelangen die Zellwerte von dem 256 x 256 Zellen großen Spielfeld in den Pipeline buffer, wodurch das Spielfeld ganz wie bei Game of Life verdoppelt wird. Aufgrund dieser Verdopplung können die Zellwerte für die nächste Generation direkt in das Spielfeld eingeschrieben werden, indem die alten Werte aus dem Pipeline buffer gelesen werden, die Neighborhood selection durchlaufen, wo die Art der Nachbarschaft festgelegt wurde, um dann der Regel im lookup table entsprechend zur Berechnung des neuen Zellwertes zu dienen. Wie zu sehen ist, besteht die CAM-6 Hardware vor allem aus Elementen, die der Lösung dieses bei der Berechnung traditioneller zellulärer Automaten zwangsläufig auftretenden Problems des Überschreibens dienen. Eine hardwaretechnische Umsetzung, die dieses Problem auf geschicktere Weise lösen oder ganz ausklammern könnte, würde also auch reduzierten Aufwand in Sachen Hardware mit sich bringen.

3.3 Die CAM-7

Die Cellular Automata Machine-7, der Nachfolger der CAM-6, war im Unterschied zu dieser von vornherein als Blockautomat angelegt. Das Abweichen von traditionellen Zellularautomaten als Struktur der Hardware hin zu Blockautomaten mit Margolus-Nachbarschaft war keine arbiträre Entscheidung, sondern stellte die Lösung für das oben genannte Problem dar.

Da in Blockautomaten mit Margolus-Nachbarschaft jeweils ein isolierter Viererblock von Zellen ersetzt wird, ohne dass andere Nachbarzellen beachtet werden müssen, können die neuen Werte die alten sofort ersetzen und direkt in das Feld eingeschrieben werden. Daher entfiel für die CAM-7 die Notwendigkeit einer Verdopplung des Datensatzes und damit auch der Puffer, der bei der CAM-6 pro 3 x 3 Zellblock immerhin eine Länge von 515 bit gehabt hatte (Margolus/Toffoli 1987a, 982).

Das schlug sich im Aufbau der CAM-7 nieder:



Schematischer Aufbau der CAM-7 (Margolus/Toffoli 1987a, 981)

Für die CAM-7 gilt also schlicht:

$$b_{x,y}^{(t)} = \mathbf{F}[b_{x,y}^{(t-1)}] \qquad (2)$$

Der Wert des Blocks *b* an der Position *x, y* zum Zeitpunkt *t* wird durch eine Funktion **F** bestimmt, die als Parameter nur den Wert desselben Blocks *b* zum vorherigen Zeitpunkt *t*-1 benötigt. Das ist eine Zuweisung von solcher Art, wie sie in Kapitel 2.2 dargestellt wurde.
Das in dieser Formel noch nicht miteinbezogene zeitkritische Element, das Alternieren der Gitter, wurde von Margolus und Toffoli durch den Einsatz von Controller-Chips gehandhabt, je einer für jede der 16 Planes. Es sollten damit nicht die Bits in der Hardware verschoben werden – was zuviel Rechenzeit gekostet hätte –, sondern über einen 4-Bit-Bus in den Controller-Chips die Partitionierungen der Zellen durch veränderte Ansteuerung der Adressen horizontal und vertikal verschoben werden (Margolus/Toffoli 1987a, 983). Sehr zu vergleichen mit der Lösung, die in dem hier vorgestellten Programm gewählt wurde (vgl. Kapitel 5.4 und 5.6.1). Der Wegfall des Puffers bedeutete, dass die CAM-7 hardwaretechnisch simpler zu realisieren gewesen wäre als die CAM-6 (Margolus/Toffoli 1987a, 982).

Der entscheidende Vorteil der CAM-7 war mithin, dass die Zellen nicht länger sequentiell abgearbeitet werden mussten, sondern gleichzeitig berechnet werden konnten. Bei der CAM-7 handelte es sich also um einen echten Parallelrechner! Zwar sollte nicht jeder Block über einen eigenen Prozessor verfügen, sondern eine feste Anzahl von Blöcken sich einen Prozessor teilen. Dies hätte aber nichtsdestoweniger zu einem erstaunlichen Effekt geführt: Da jede Gruppe von Blöcken über einen eigenen Prozessor verfügt hätte, wäre die Rechenzeit in der CAM-7 nicht mehr von der Größe des Spielfelds abhängig gewesen. Egal, um wie viele real in Hardware hinzugefügte Zellen der Automat erweitert worden wäre, die benötigte Rechenzeit hätte sich nicht verändert. Nicht einmal die Lichtgeschwindigkeit der Signale hätte eine Bremse für die CAM-7 dargestellt:

> What is essential is that adding more cells does not increase the time needed to update the entire space – since you always add associated processors at a commensurate rate. For the foreseeable future, there are no practical technological limits on the maximum size of a simulation achievable with a fixed CAM[-7] architecture. (Margolus/Toffoli 1987a, 968)

Vom formalen Standpunkt der Berechenbarkeit aus wären CAM-6 und CAM-7 gleichmächtige Maschinen gewesen. Jeder Zellularautomat mit Margolus-Nachbarschaft lässt sich nämlich in einen traditionellen zellulären Automaten umformen, und umgekehrt (Margolus/Toffoli 1987b, 124–126). Da einige dieser Automaten turingvollständig sind – man denke nur an die von Wolfram okkupierte, tatsächlich aber von anderen maßgeblich untersuchte Regel 110 (Martínez 2004) oder an das bekanntere Game of Life (Adamatzky 2010) – sind CAM-6 und CAM-7, die solche Automaten berechnen können, zwangsläufig ebenfalls turingvollständig.
Davon unberührt bleibt ein Unterschied im formalen Zeit- und Raumverhalten zwischen Blockautomaten und traditionellen Automaten:

> Cellular automata can be seen as dynamical systems whose laws are *periodic* in space and time. In an ordinary cellular automaton, one temporal cycle corresponds to one step of the rule, and along either axis one spatial cycle corresponds to one cell. Partitioning cellular automata [d. h. Blockautomaten], though defined on the same spacetime grid, have laws that are periodic with a *coarser* pitch. Specifically, with the Margolus neighborhood it takes two steps to complete a cycle along the time axis (one step will use the even grid, the other the odd grid [das zeitkritische Alternieren der Gitter]), and it takes two cells (the length of a 2x2 block) to span a cycle along either spatial axis. (Margolus/Toffoli 1987b, 120)

Während die CAM-6 als Umsetzung traditioneller Zellularautomaten deren Parallelität nur simulieren kann, stattdessen sequentiell arbeitet und auf einen Puffer angewiesen ist, ist die CAM-7 als Blockautomat echt nebenläufig, mit allen Vorteilen, die das mit sich bringt. Auch hier zeigt sich – ganz wie an den realen Von-Neumann-Maschinen, die im Gegensatz zu der Turingmaschine von 1936 mit ihrer rein logisch-mathematischen Zeit einen implementierten Takt benötigen – der grundlegende Unterschied zwischen nur gedachten und als Hardware existierenden und prozessierenden Maschinen.
Interessanterweise ist es aber nicht das formale Zeitverhalten, sondern das formale Raumverhalten, also die Partitionierung des Spielfelds in Blöcke, sowie die Weise, in der diese gehandhabt werden, das dem wesentlichen Unterschied zwischen Blockautomaten und traditionellen Automaten zugrunde liegt. Blockautomaten als Hardware können ein anderes Zeitverhalten haben, nicht, weil sie formal eine längere Periode als traditionelle Zellularautomaten aufweisen, sondern weil in jeder Generation der Raum als aus isolierten Blöcken bestehend behandelt wird. Dieses formale Raumverhalten kann aber nur darum Einfluss auf das reale Zeitverhalten des Automaten haben, da es durch das Alternieren der Gitter selbst *zeitkritisch* gemacht wird (vgl. dazu Kapitel 2.3).

3.4 Eine Spurensuche

Nicht nur vor dem Hintergrund der Differenz zwischen Papiermaschine und deren realen Implementierungen wäre eine Antwort auf die Frage wünschenswert, ob die CAM-7 jemals gebaut und in Vollzug gesetzt wurde. Die gegenwärtig zur Verfügung stehenden Quellen bieten dazu nur wenig Information:
Im Jahr 1987 schrieben Margolus und Toffoli von der CAM-7 als „work in progress“ (Margolus/Toffoli 1987a) und auch in der Doktorarbeit von Margolus steht die Maschine noch im Fokus (1988). Danach allerdings wird sie nicht mehr erwähnt. Am 27. Oktober 1992 wird Margolus und Toffoli dann das U.S. Patent Nr. 5,159,690 für ein „Multidimensional Cellular Data Array Processing System which Separately Permutes Stored Data Elements and Applies Transformation Rules to Permuted Elements“ zugesprochen. Dabei handelt es sich um die CAM-7, wie Beschreibungen, Skizzen und Verweise auf frühere Artikel zur CAM-7 klar belegen (Margolus/Toffoli 1992). Eingereicht wurde das Patent im September 1988. Was aber passierte dazwischen? Lässt sich eventuell eine andere Beschäftigung Margolus und Toffolis nachweisen, die eine Abwendung von der CAM-7 als Konzept nahe legt?
Im Folgenden soll die Geschichte des Vorgängers und die des Nachfolgers der CAM-7, also der CAM-6 und der CAM-8 skizziert werden, in der Hoffnung, etwas Licht auf den Verbleib der Cellular Automata Machine-7 werfen zu können.

Dass die CAM-6 Serienreife erreichte, heißt, dass man diesen Computer bestellen konnte und dann – in den Worten von Rudy Rucker – eine „chip-laden card you could plug into a slot in any DOS-based personal computer“ zugeschickt bekam (Rucker 1988). Nachdem Rucker Gelegenheit gehabt hatte, die von der CAM-6 in Echtzeit ausgegebenen „laufenden“ Bilder im Büro von Margolus und Toffoli zu bewundern, war er sich sicher, dass die CAM-6 ein Verkaufsschlager werden würde. Entgegen dieser Erwartung wurde die CAM-6 aber kommerziell ein Misserfolg.
Das hatte mehrere Gründe, unter denen an erster Stelle die lange Lieferzeit stand. Rucker, der von seinem Department, dem Department of Mathematics and Computer Science der San Jose State University, eine CAM-6 zugestanden bekam, wartete monatelang, bis die Firma System Concepts die Maschine zusandte:

> The packaging of the board was kind of incredible. It came naked, all by itself, in a plastic bag in a small box of Styrofoam peanuts. No cables, no software, no documentation. Just a three inch by twelve inch rectangle of plastic—actually two rectangles one on top of the other—completely covered with computer chips. There were two sockets at one end. I called Systems Concepts again, and they sent me a few pages of documentation. You were supposed to put a cable running your graphics card's output into the CAM-6 board, and then plug your monitor cable into the CAM-6's other socket. No, Systems Concepts didn't have any cables, they were waiting for a special kind of cable from Asia. So one of the SJSU Math and CS Department techs made me a cable. All I needed then was the software to drive the board, and as soon as I phoned Toffoli he sent me a copy (Rucker 1989).

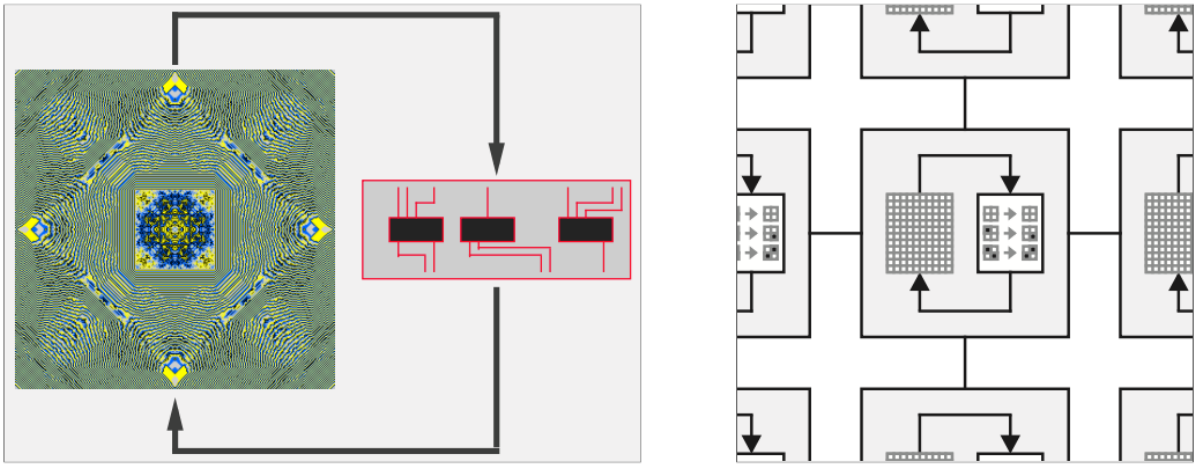
Die mangelnde Verpackung, die schlechte Dokumentation und die fehlende Soft- und Hardware, all das war angesichts des Preises von rund 1500$ für eine CAM-6 mehr als nur ein Wermutstropfen. Margolus und Toffoli hatten selbst noch angekündigt:

> It [the CAM-6] is currently produced by SYSTEMS CONCEPTS (San Francisco, CA), from which it was commissioned with the explicit intention that, after fulfilling MIT's internal needs, further output of the production line would be made available to the scientific community at large, as inexpensively as possible. (Margolus/Toffoli 1987b, 14)

Hardwaretechnisch war die CAM-6 aus Standardteilen zusammengesetzt. Es wurden keine besonderen Prozessoren verwendet, sondern nur einige schnelle RAM-Chips verdrahtet. Hinzu kam noch, dass die CAM-6 in Forth zu programmieren war, einer Programmiersprache mit umgekehrter polnischer Notation, die den Nutzer vor weitere Schwierigkeiten stellte: „Once you get used to it, Forth is very clean and nice, but it makes you worry about things you shouldn't really have to worry about." (Rucker 1989) Selbst ohne diese Probleme wäre der CAM-6 kein langfristiger Erfolg beschieden gewesen. Vor dem Hintergrund des Mooreschen Gesetz war wohl jedem Versuch, eine Lösung real in Hardware zu fixieren, nur eine kurze Halbwertszeit eingeschrieben. So überholten bald Software-Lösungen auf Von-Neumann-Maschinen die CAM-6 in Update-Rate und räumlicher Auflösung und die Produktion wurde eingestellt.

Im selben Jahr, in dem Margolus und Toffoli das Patent für die CAM-7 zugesprochen bekamen, ist von einer neuen geplanten CAM die Rede, der CAM-8 (Margolus 1992). Es scheint ganz so, als wäre es der langwierigen Bearbeitung des Patentantrags aus dem Jahr 1988 geschuldet, dass sich Patentschrift und Erwähnung dieser neuen CAM überschneiden. Die CAM-8 nämlich ist keine verbesserte CAM-7, sondern ein „careful compromise between parallel and serial processing." (Margolus/Toffoli 1990b, 266) Sie verwendet ein 3D mesh array aus SIMD (Single Instruction, Multiple Data) Prozessoren. Jeder Prozessor verarbeitet einen Teil des Gitters, und in diesem Aspekt ähnelt die

neuere CAM-8 der CAM-7. Der Prozessor alterniert dabei allerdings zwischen der Übertragung von Daten und dem sequentiellen Update der Bitgruppen in jeder Zelle, was die sequentielle Arbeitsweise der CAM-6 aufruft (Margolus 1999, 26). Die CAM-8 ist also eine Verbindung des Designs von CAM-6 und CAM-7.



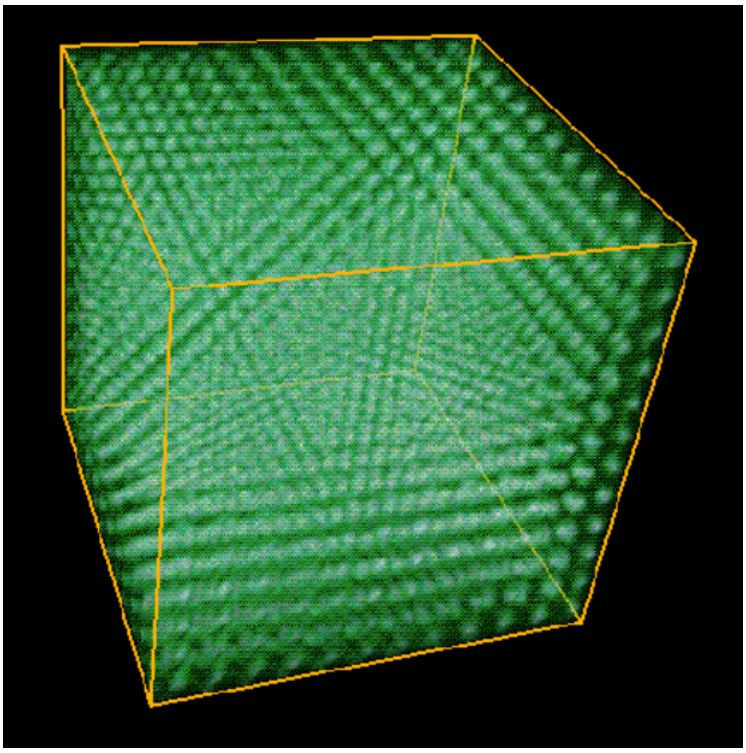
Links: Schema der frühen CAM-1 bis CAM-6. Rechts: Schema der CAM-8.
(Margolus 1999, 26 und 2002)

Die CAM-8 war größer angelegt als alle vorhergehenden CAMs. Sie sollte dazu dienen, ein bisher verborgenes Band des rechnerischen Spektrums freizulegen und vielfältigen wissenschaftlichen Disziplinen eine Plattform bieten, um ohne viel Aufwand effiziente Simulationen von physikalischen Phänomenen im großen Maßstab durchzuführen. Gefördert von der ARPA (wie die DARPA zwischen 1992 und 1996 noch einmal hieß) wurde ein verkleinerter Prototyp der CAM-8 gebaut:

CAM-8 Peripherie.
(Margolus c)

CBus backplane mit einem Modul der CAM-8.
(Margolus c)

Dieser Prototyp der CAM-8 wurde von der U.S. Air Force für einige Simulationen verwendet, wie für die Folgende mit einer Größe von 125 Millionen Zellen:



U. S. Air Force 3-D Simulation von elastischen und soliden Arealen bei der Gaskristallisation. Entwickelt mittels der CAM-8 (Margolus d).

Für die CAM-8 wurden noch weitere Programme entworfen. Unter ihnen finden sich neben physikalischen Modellen so unterschiedliche Ansätze wie die Simulation digitaler Schaltungen ähnlich Wireworld (vgl. Maibaum 2016), Verfahren zur Erhöhung des Kontrasts digitaler Bilder oder zum Erstellen stereoskopischer Bilder (Margolus a). Im Grunde zeichnet sich hier ab, dass die CAM-8 kein Spezialcomputer werden sollte, was ihr vielleicht eine längere Lebenszeit verliehen hätte, sondern dass sie genau das können würde, was die Computer mit Von-Neumann-Architektur, als die quasi-universellen Maschinen, die sie nun einmal sind, bald schneller und preisgünstiger erledigten. Auch der CAM-8 war kein Erfolg beschieden. Sie kam trotz jahrelanger Bemühungen Margolus nie über den Status eines Prototypen hinaus. Im Logbuch ihrer Entwicklungsgeschichte finden sich als letzter Eintrag die folgenden Zeilen vom 5. August 2001:

> Sigh... After having our machine abused by some mindless jerks the disk crashed and everything since Nov of 1998 has been lost. This is of course a major setback. Work on CAM8 software has since come to a halt. It is uncertain when it can be resumed. Hopefully we can offer something new this year. (Margolus b)

Schon lange zuvor ist die CAM-7 aus den Arbeiten Margolus und Toffolis verschwunden. Dieses Verschwinden ist auf eigenartige Weise vollständig: So führt Margolus in einem Vortrag auf der Computing Beyond Silicon Summer School 2002 alle Maschinen von CAM-1 bis CAM-8 an. Nicht aber die CAM-7 – sie wird ausgelassen. Selbst in Margolus und Toffolis Kabinett der zellulären Rechnerarchitekturen scheint sie ein weißer Elefant geblieben zu sein. Die Frage, ob sie tatsächlich gebaut wurde, lässt sich also nur aufgrund der hier angestellten Vermutungen, und das heißt, unter Vorbehalt, beantworten:

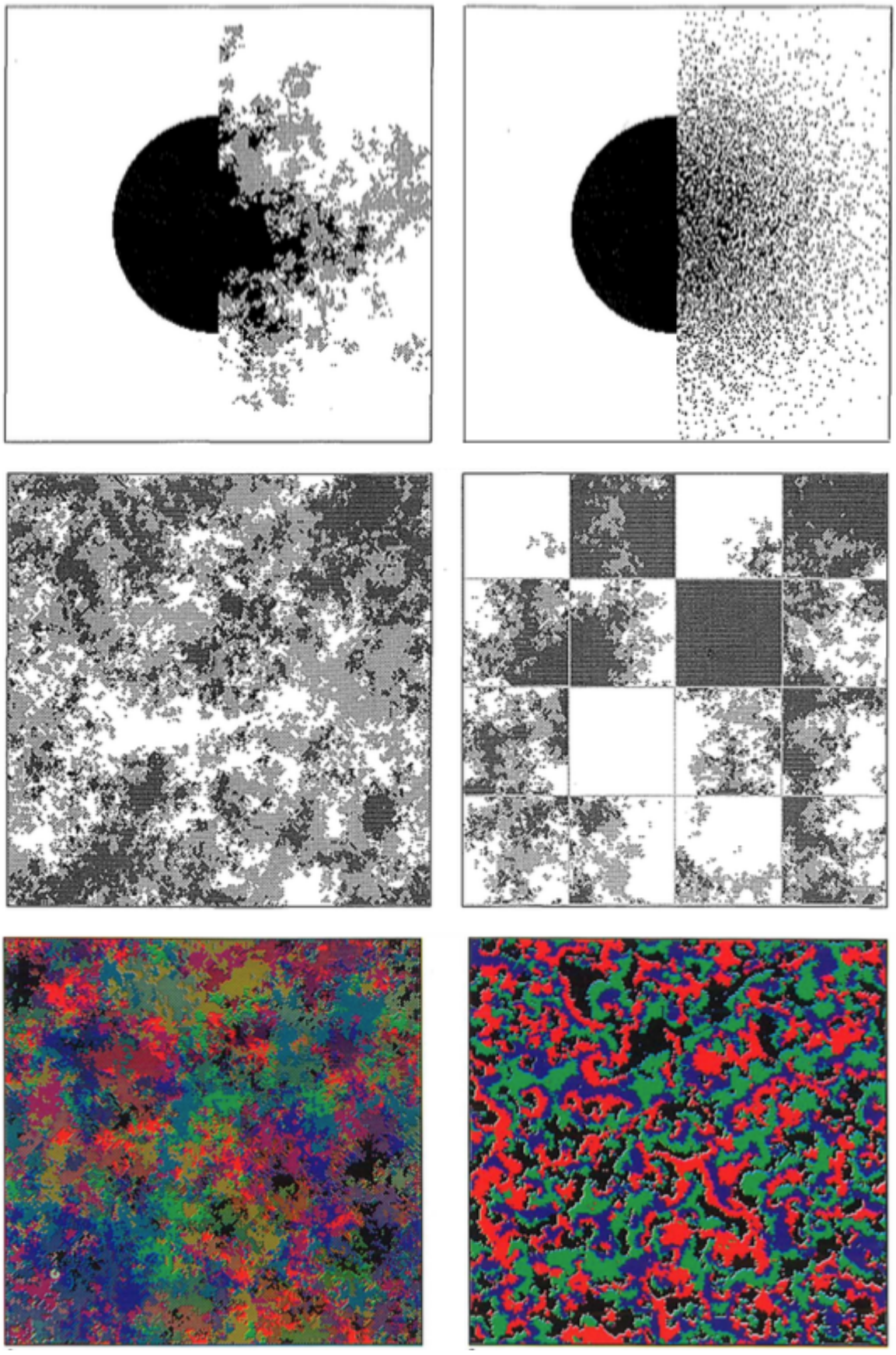
Die vorhandenen Dokumente, der kommerzielle Fehlschlag der CAM-6, das Auftauchen und der letztendliche Misserfolg der CAM-8 legen zusammen mit dem Verschwinden der CAM-7 aus den Arbeiten von Margolus und Toffoli den Schluss nahe, dass die CAM-7 nie gebaut wurde, sondern dass es bei einer Papiermaschine im Sinne der Turingmaschine von 1936 geblieben ist.

4. Der Erfolg der zellulären Automaten – ein Medieneffekt?

„Das Sehen als primärer Sinn wissenschaftlicher Erkenntnis […] gilt sowohl in den Naturwissenschaften selbst als auch in den mit ihnen befassten Metadisziplinen als offenbar ebenso selbstverständlich wie alternativlos." (Volmar 2015) Im Diskurs über die zellulären Automaten ab dem Beginn der Achtzige Jahre ist diese Vorherrschaft des Visuellen besonders auffällig. Hier werden allzu häufig visuelle Medien aufgerufen:
Der ungarische Computerwissenschaftler Peter Gacs wird von Rucker mit folgenden Worten zitiert: „With the cellular automaton simulator, we can *see* many very alien *scenes*. We have a new world to *look* at, and it may tell us a lot about our world. It is like *looking* first into a *microscope*. [Kursiv Th. N.]" (Rucker 1989) Wolfram meinte im selben Jahr rückblickend: „Computer scientists had done some fairly worthless clean-up work on Ulam and von Neumann's work. There were maybe a hundred papers. What I found outrageous was that none of the papers had any *pictures*. [Kursiv Th. N.]" (Rucker 1989) Margolus selbst äußerte sich folgendermaßen: „As we sit in front of our computer screens, watching to see what happens next, we never really know what new tricks our CA's may come up with. It is really an exploration of new worlds – *live television* from other universes. [Kursiv Th. N.]" (Margolus 1999, 32)
In Echtzeit auf dem Monitor zu sehen, was zuvor nur konzeptuell zugänglich gewesen war, das war für viele eine umwälzende Erfahrung. Als Ives Pomeau zum ersten Mal sein HPP-Gas als zellulären Automaten auf der CAM-6 im Vollzug sah, änderte das seine Einschätzung, was mit dem von ihm entwickelten Modell möglich war, grundlegend:

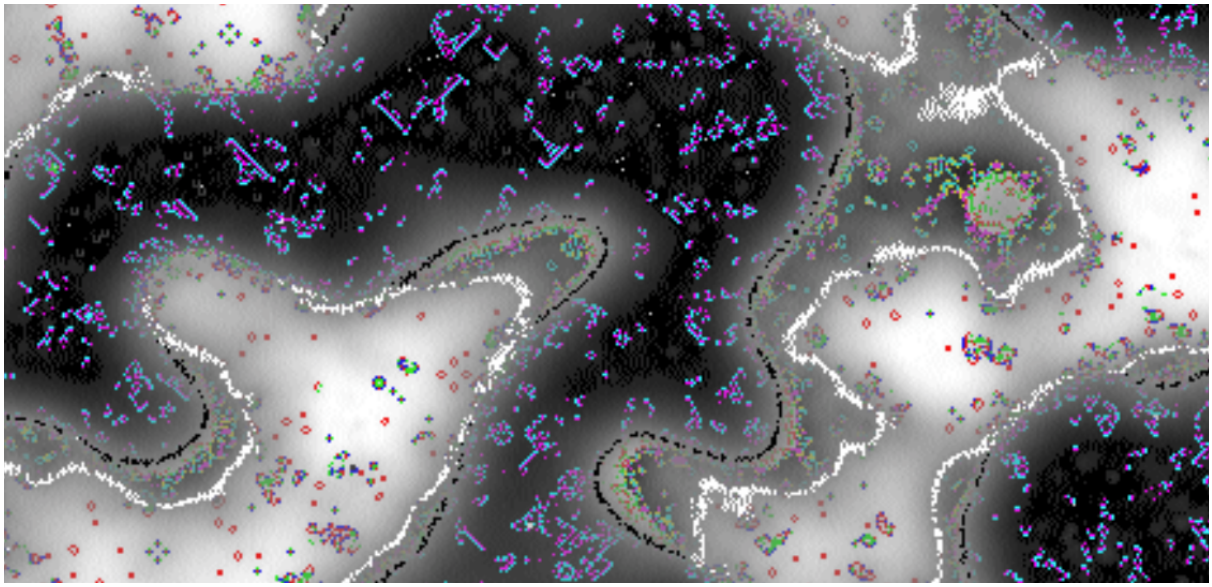
> According to Pomeau, seeing this model of his running on one of our early cellular automata machines made him realize that what had been conceived primarily as a conceptual model could indeed be turned, by using suitable hardware, into a computationally accessible model: this stimulated his interest in finding lattice-gas rules which would provide better models of fluids. (Margolus/Toffoli 1987a, 967f., vgl. auch Margolus/Toffoli 1990b, 265)

Das Verhalten dieses HPP-Gases lässt sich in Kapitel 6.2.1 und durch das Programm GASWELLE nachvollziehen. Um einen Eindruck von der Darbietung zu erhalten, wie sie Pomeau mit der CAM-6 erlebt haben mag, lohnt es sich, einen Blick auf einige der Bilder zu werfen, die Margolus und Toffoli in ihrem Buches *Cellular Automata Machines. A new environment for modelling* präsentieren:



Bilder oben: Pseudo-Diffusion vs. genuine Diffusion. (Margolus/Toffoli 1987b, 85)
Bilder Mitte: Genetische Drift. (Margolus/Toffoli 1987b, 87)
Bilder unten: Genetische Drift von vier Populationen. (Margolus/Toffoli 1987b, Farbseiten)

Dabei darf nicht vergessen werden, dass die CAM-6 eben keine Standbilder ausgab, sondern dass die Muster sich vor den Augen des Betrachters bewegten, zerflossen, aufeinander reagierten – kurz: sich im Vollzug befanden. Es ist angesichts dessen, was wir heute zu sehen gewohnt sind, sicherlich schwer, nachzuempfinden, wie neu dieses „Leben“ auf dem Bildschirm in diesem Kontext gewirkt haben muss. Eine Visualisierung von Rechenprozessen in dieser Größe und Geschwindigkeit war ein absolutes Novum, etwas bis dahin noch Ungesehenes. Wird die hier abgebildete Graphik mit den 60 Bildern pro Sekunde der CAM-6 im Vollzug kontrastiert (vgl. die Internetseite Hopkins: CAM6), so lässt sich vielleicht ein wenig einschätzen, welche Wirkung diese CAM auf Pomeau gehabt haben mag.



Simulation der CAM-6 (Hopkins: CAM6). Im Bild ist allerdings nicht das HPP-Gas zu sehen, sondern ein zellulärer Automat mit mehr als zwei Zuständen.

Fraglos spielt es für die Wirkung von implementierten zellulären Automaten eine Rolle, wie viele Zellen dargestellt werden. Das lässt sich leicht anhand des hier vorgestellten Programms nachvollziehen, indem größere und kleinere Spielfelder verglichen werden (vgl. die Variable *Seitenlänge* in Kapitel 5.2.2). Der wesentlichere Faktor ist aber die Ablaufgeschwindigkeit, da sie die Darstellung in Echtzeit und damit Margolus „live television“ erst ermöglicht.

Zwischen den in Echtzeit berechneten graphischen Ausgaben der Achtziger Jahre und den zellulären Automaten, die Conway und seine Mitstreiter in den Siebzigern mit Spielsteinen händisch umsetzten liegt demnach nicht nur eine historische Kluft.

Während Conways Spielsteine noch an die *psephoi*, die Rechensteine der Griechen anknüpfen, kommen mit der Implementierung in elektronischen Rechnern non-human agencies ins Spiel, welche die menschlichen Möglichkeiten und Wahrnehmungsschwellen grundlegend unterlaufen. Der Schritt vom manuellen Abarbeiten hin zur Übergabe des Prozesses an einen Automaten bringt eine Verschiebung des Zeitregimes vom Menschen auf die Maschine mit sich. Was zuvor nicht ohne Partizipation und die Benutzung von mehreren Sinnen vonstatten gehen konnte, geschieht nun ohne jedes Zutun und reduziert auf das Visuelle. Während Conways Spielsteine im Sinne McLuhans ein kaltes Medium bildeten, ist die graphische Ausgabe mittels Computer in hohem Maße heiß.[5] Die technisch verwaltete Zeit wird zum bestimmenden Faktor: Erst wenn das zeitliche Intervall zwischen den Refreshs des Bildschirms, zwischen den Generationen des zellulären Automaten also, klein genug wird, d. h. erst wenn die Einzelbilder quasi-kinematographisch zum Film verschmelzen und der Automat im Gegenwartsfenster des menschlichen Betrachters abläuft, bildet sich der Medieneffekt aus, der Margolus, Toffolis, Ruckers, Pomeaus und vieler anderer Faszination für zelluläre Automaten entfachte.

Der Medienbruch veränderte nicht nur den Werkzeugkasten oder die Meinung eines Wissenschaftlers wie Pomeau. Aus medientheoretischer Sicht ließe sich vielmehr konstatieren, dass dieses neue und durch und durch heiße Medium in die bestehenden Wahrnehmungsmuster eingriff: „The effects of technology do not occur at the level of opinions or concepts, but alter sense ratios or patterns of perception steadily and without any resistance." (McLuhan 1964, 19) Die Bedingungen der Wahrnehmung selbst werden durch die Wirkung des neuen Mediums unterlaufen und verändert, werden neu formiert und in-formiert:

> What we are considering here, however, are the psychic and social consequences of the designs or patterns as they amplify or accelerate existing processes. For the "message" of any medium or technology is the change of scale or pace or pattern that it introduces into human affairs. (McLuhan 1964, 8)

Die Veränderungen, die in Werkzeugkasten, Meinung oder Praxis eines Forschenden auftreten, sind nach dieser Lesart Sekundärphänomene, ausgelöst durch die vom Medium verursachten primären Veränderungen in den Wahrnehmungsstrukturen.

[5] In der Beschreibung seines Treffens mit Margolus und Toffoli bezeichnet Rucker die zellulären Automaten als ‚hot', und nicht etwa als ‚cool': „Cellular automata are hot." (Rucker 1989) Eine Koinzidenz, die McLuhan sicherlich gefallen hätte.

Interessant in diesem Zusammenhang bleibt, dass die Publikationen Margolus und Toffolis mit graphischen Darstellungen der Ergebnisse von zellulären Automaten nicht geizen, sich darin aber keine Bilder der verwendeten Hardware finden. Dieser Umstand mag für Texten zu Software, die auf herkömmlichen Von-Neumann-Maschinen läuft, nicht ungewöhnlich sein. Wenn aber, wie im Fall der CAMs, Spezialhardware erstellt wurde, wäre es da nicht angebracht, diese auch abzubilden? Von der CAM-8 aber ließen sich nur die Photographien einzelner Komponenten aufspüren (vgl. Kapitel 3.4). Bilder der CAM-6 als realer Maschine sucht man vergeblich. Weder in Margolus Doktorarbeit (Margolus 1988), noch in dem gemeinsam mit Toffoli verfassten Buch finden sich Bilder dieser Hardware (Margolus/Toffoli 1987b). An keiner Stelle geht es um das – wie Rucker berichte – eher abschreckende Design der Maschine, sondern stets nur um ihren graphischen Output.

Man muss kein Epigone McLuhans sein, um anzuerkennen, dass sich die Blütezeit der Zellularautomaten ohne Bezug auf die dabei verwendeten tatsächlichen Mediensysteme nicht erklären lässt. Margolus und Toffoli behaupteten zwar, dass es ihnen nicht um die Bilder gehe. Rucker aber berichtet demgegenüber über Toffolis Umgang mit der CAM-6:

> Toffoli pries Margolus away from the controls and takes over. “Now we do the square rock in the toroidal pond again, but this time we add a heat-bath, a cloud of random gas in the background.” The background fills with darting dots, and Toffoli keys another big red square into the center. This time the waves are smooth and roughly circular, much like real waves in a real pond. We try it with two squares and get interference patterns. Toffoli is pleased. He says that this shows how simple physics really is. (Rucker 1989)

Toffoli setzt hier die Erzeugung der Bilder mit der Erzeugung physikalischer Phänomene schlichtweg gleich, während jede medientheoretische Reflexion der verwendeten Instrumente und Sinne unterbleibt. Was aber stellt sicher, dass Margolus und Toffolis Idee, die physikalischen Vorgänge selbst für Berechnungen zu verwenden, nicht in weiten Teilen in den laufenden Bilder gründet, die ihnen von den technischen Apparaturen vorgespielt wurden? Was sagt, dass ihr Vorhaben nicht nachhaltig von der Wirkung ihres bildmächtigen Apparates befeuert wurde? Es stellt sich also die Frage, ob Margolus und Toffolis Wunsch, immer größere Spielfelder im Vollzug sehen zu wollen – „to let us *see* into the previously inaccessible "band of the computational spectrum" [Kursiv: Th. N.]“ (Margolus e) – nicht hauptsächlich die Folge eines Medieneffekts gewesen ist, ausgelöst durch das radikalisierte und damit neue Medium der implementierten zellulären Automaten.

Diese Frage soll hier nicht beantwortet, sondern nur gestellt werden. Stattdessen möchte diese Arbeit ein Werkzeug anbieten, das es dem Leser ermöglicht, dieser Frage selbst nachzugehen. Ganz in diesem Sinne wird im folgenden Kapitel das mitgelieferte Python-Programm, mit dem die Familie der Blockautomaten implementiert werden kann, einem erläuternden Close-Reading unterzogen. Dieses Close-Reading soll es dem Leser ermöglichen, selbstständig Veränderungen im Code vorzunehmen. Die Hoffnung ist dabei, dass dieses und die noch folgenden Kapitel 6 und Kapitel 7 einen Rahmen aufspannen können, in dem der Leser mithilfe des Programms eigenen Gedanken auf experimentelle Weise nachgehen kann.

5. Das Programm

Im Folgenden soll der Leser Zeile für Zeile in das Python-Programm eingeführt werden. Besondere Kenntnisse in Python sind dabei nicht erforderlich.

5.1 #____IMPORTE[6]

In diesem Abschnitt werden die für das Programm benötigten Bibliotheken importiert. Dazu gehören die graphische Ausgabe Tkinter,

```
from tkinter import *
```

die standardmäßig in Python verfügbar ist und nicht gesondert bezogen werden muss.

```
from random import *
```

Diese Zeile lädt die Bibliothek mit Routinen für die Erstellung von Zufallszahlen. Sie werden benötigt, um das Spielfeld des Automaten zufällig mit Zellen zu besetzen.

```
import time
```

Hier werden unter anderem Routinen importiert, um Pausen zwischen einzelnen Programmschritten einzufügen. Das kann z. B. dann notwendig werden, wenn die Ablaufgeschwindigkeit eines zellulären Automaten die menschlichen Wahrnehmungsschwellen unterläuft.

5.2 #____VARIABLEN

In diesem Teil des Programms kann der Benutzer verschiedene Parameter einstellen, um den zellulären Automaten je nach Bedarf einzurichten. Die den Variablen in den folgenden Programmzeilen zugewiesenen Ziffern haben stets nur beispielhaften Charakter. Sie können und sollen vom Benutzer verändert werden, um verschiedene Variationen der Blockautomaten zu kreieren.

---

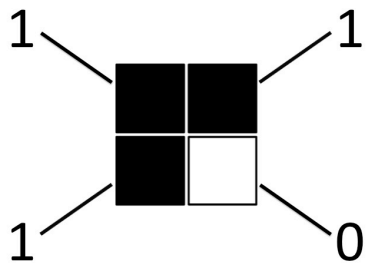
[6] Das Rautezeichen # wird in Python gesetzt, um eine Zeile oder einen Teil einer Zeile auszukommentieren. Jedes Zeichen und jede Zeichenfolge, die in einer physischen Zeile hinter dem Rautezeichen # folgt, wird vom Interpreter ignoriert und hat keinen Einfluss auf das laufende Programm.

5.2.1 Kennziffer

Der wichtigste Parameter des Automaten ist seine Kennziffer:

```
Kennziffer = ['0', '8', '4', '3', '2', '5', '9', '7', '1', '6', '10',\
              '11', '12', '13', '14', '15']
```

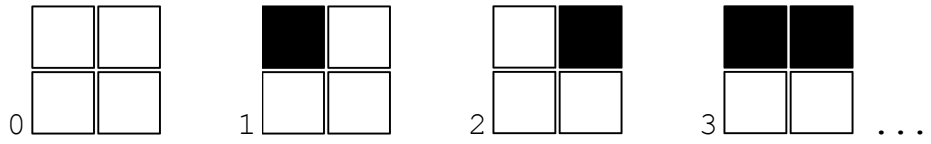
Die Kennziffer repräsentiert ein bestimmtes Regelset, also eine Zuordnung von Blöcken, wie sie in Kapitel 2.2 dargestellt wurde. Stellen wir uns die 16 möglichen Blöcke nacheinander aufgereiht vor, in der Reihenfolge, wie sie in Kapitel 2.1 abgebildet ist. Diese Reihenfolge ist keine beliebige. Dahinter stand vielmehr die folgende Idee:
Jeder der vier Zellen eines Blocks kann ein Wert zugewiesen werden. Wir entscheiden uns, denjenigen Zellen eine 1 zuzuschreiben, die lebendig bzw. schwarz ausgefüllt sind, und den restlichen Zellen, also denen, die tot bzw. leer oder weiß ausgefüllt sind, eine 0. Das sähe dann beispielsweise so aus:



Im abgebildeten Fall werden dem Block also die Werte 1, 1, 1 und 0 zugeordnet. Anhand dieser vier einzelnen Werte kann nun der ganze Block selbst als die Folge dieser Werte dargestellt werden. Das geschieht, indem die Werte der Zellen in einer festen Reihenfolge hintereinander geschrieben und zu einer einzigen Ziffernfolge konkateniert, d. h. zusammengeklebt werden. Dabei wollen wir folgendermaßen vorgehen: Wir beginnen mit der linken oberen Zelle, die wir in guter Programmierermanier nicht als die erste Zelle, sondern als *nullte* Zelle definieren. Der Wert dieser Zelle in dem oben dargestellten Block ist 1. Diesem Wert wollen wir nun den Wert der *ersten* Zelle, als die wir die rechte obere Zelle bezeichnen wollen, und die Werte der *zweiten* und *dritten* Zelle, das sind die Zelle links unten und die Zelle rechts unten, anheften. Dabei gehen wir so vor, dass wir den jeweils nächsten Wert *links* neben den alten Wert schreiben.

Fertig nebeneinander geschrieben sieht die konkatenierte Ziffernfolge also so aus:

Der letzte Wert, also die 0 aus der *dritten* Zelle rechts unten im Block, steht jetzt links in unserer Ziffernfolge. Das bringt einen höchst willkommenen Vorteil mit sich, der auch erklärt, warum wir uns für diese Vorgehensweise entschieden haben:
Jede der Ziffernfolgen, die auf diese Weise aus einem der 16 Blöcke konkateniert werden kann, wird immer nur aus 0 und 1 bestehen, da den Zellen des Blocks immer entweder eine 0 oder eine 1 zugewiesen wird. Das bedeutet, dass jede der 16 möglichen Ziffernfolgen als Binärzahl gelesen werden kann. Ein leerer Block mit vier toten Zellen würde also die kleinste Binärzahl, die Zahl 0000 zugeordnet bekommen, ein ganz schwarzer Block mit nur lebenden Zellen die Zahl 1111. Der entscheidende Kniff ist nun: Wenn die Binärzahlen, die wir auf diese Weise den Blöcken zugeordnet haben, in eine Dezimalzahl umgerechnet werden, dann zeigt sich, dass diese Dezimalzahlen genau die Zahlen sind, die in Kapitel 2.1 den Blöcken als Nummerierung an die Seite gestellt wurden.
Zum besseren Verständnis hier noch einmal die erste Reihe dieser Darstellung, wie sie in Kapitel 2.1 gezeigt wurde:



Nach der oben beschriebenen Vorgehensweise würde dem Block 0 die binäre Zahl 0000 zugeordnet, dem Block 1 die binäre Zahl 0001, dann 0010, 0011, usw. Das heißt: Die Zuordnung von Binärzahlen definiert die Blöcke nicht nur eindeutig, sondern *ordnet* diese Blöcke auch in einer bestimmten Art und Weise an. In dezimalen Zahlen ausgeschrieben sähe diese Ordnung – wenig überraschend – so aus:

```
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15'
```

Mithilfe dieser Zahlen lässt sich jetzt ein Regelset, also eine Zuordnung von Blöcken, numerisch ausdrücken. Für die in Kapitel 2.2 dargestellte Regel sähe das folgendermaßen aus:

```
 '0' ->  '0'
 '4' ->  '2'
 '6' ->  '9'
'10' -> '10'
'11' -> '11'
'15' -> '15'
```

Noch fehlen die rotationssymmetrischen Regeln, damit alle Blocknummern angeschrieben werden können. Berücksichtigt man diese, ergibt sich folgende Liste, welche die selbe Regel enthält, jetzt ausgeschrieben mit allen möglichen Kombinationen:

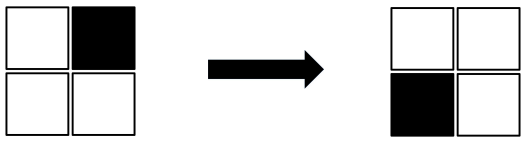
```
 '0' ->  '0'
 '1' ->  '8'
 '2' ->  '4'
 '3' ->  '3'
 '4' ->  '2'
 '5' ->  '5'
 '6' ->  '9'
 '7' ->  '7'
 '8' ->  '1'
 '9' ->  '6'
'10' -> '10'
'11' -> '11'
'12' -> '12'
'13' -> '13'
'14' -> '14'
'15' -> '15'
```

Nun sind alle 16 Blöcke von Block 0 bis Block 15 aufgelistet und einem anderen Block zugeordnet. Da in der rechten Spalte der Zuordnung wie in der linken jede Zahl nur einmal vorkommt, ist die Regel reversibel, was heißt: Der Automat könnte auch rückwärts ablaufen, und würde ohne das Auftreten von Ungewissheiten wieder in seinen Ursprungszustand zurückkehren.

Mithilfe der nun erstellten Liste mit Blocknummern ist ein Programm konstruierbar, das selbst herausfindet, welche Blöcke in der nächsten Generation für die gerade vorhandenen eingesetzt werden müssen. Das hier mitgelieferte Programm benötigt dazu sogar nur die rechte Spalte der obigen Liste als explizit ausgeschriebenen Code. Dies einfach deswegen, weil die Nummerierung der Blöcke, wie sie oben in der linken Spalte der Liste steht, exakt der Nummerierung entspricht, die Python den Elementen der Liste gibt. Diese ist damit schon implizit gegeben.

```
Kennziffer = ['0', '8', '4', '3', '2', '5', '9', '7', '1', '6', '10',\
              '11', '12', '13', '14', '15']
```

Die 0 in der hier abgebildeten Liste *Kennziffer* steht für Python also an der Stelle 0 der Liste, die 8 an Stelle 1, usw. Soll herausgefunden werden, was mit einem Block Nummer 2 geschehen soll, dann muss das Programm nur das zweite Element der Liste *Kennziffer* ansteuern, und findet dort die neue Blocknummer: Hier die Nummer 4. Es muss also der Block 2 durch einen Block 4 ersetzt werden:



Das heißt: In die Liste *Kennziffer* wird jeweils die rechte Spalte eines Regelsets in numerischer Form eingegeben, und so dem Programm mitgeteilt, welche Blöcke in der nächsten Generation die gerade vorhandenen ersetzen sollen. Die Werte in der Liste *Kennziffer* kann der Benutzer nach Belieben verändern und mit den verschiedensten Kombinationen experimentieren. Die Ergebnisse können sehr überraschend sein.

5.2.2 Seitenlänge

```
Seitenlaenge = 150
```

Hier wird die Seitenlänge des Spielfelds eingestellt. Der eingegebene Wert gibt die Anzahl der Zellen in Höhe und Breite an. Das Spielfeld ist somit immer quadratisch und beinhaltet (*Seitenlaenge*)^2 Zellen, in dem hier angeführten Fall also $150^2 = 22.500$ Zellen.

5.2.3 Zellauftrittswahrscheinlichkeit

Für die Bestückung des Spielfelds mit Zellen stehen zwei unabhängige Parameter zur Verfügung, und zwar

```
Zellauftrittswahrscheinlichkeit =  10
```

und

```
Verdichtung = 60
```

Das bietet die Möglichkeit, bestimmte Areale des Spielfelds mit Zellstrukturen von höherer Dichte zu füllen, was z. B. im Fall von simulierten Gasen von Interesse sein kann, wie in Kapitel 6 noch Thema sein wird. Die eingegebenen Werte bestimmen die Stärke der Verdichtung in Prozent und müssen darum im Intervall zwischen 0 und 100 liegen. Negative Werte und Werte > 100 führen zur Fehlermeldung oder unerwünschtem Programmverhalten. Das wäre durch eine passende Routine leicht zu ändern, widerspräche aber dem Geist, dem Leser des Verständnis des Programms nahezubringen – was auch heißt, ihn ein wenig in die Verantwortung zu nehmen.

5.2.4 Zellgröße und Zellfarbe

Bei bestimmten Regeln kann der ästhetisch motivierte Wunsch aufkommen, die Größe oder Farbe der Zellen ein wenig zu verändern.

```
Zellgroesse = 3
Farbe = "black"
```

Die empfohlenen Werte für die Größe der Zellen liegen zwischen 1 und 4. Der Farbwert wird als String zwischen "" eingegeben. Dieser kann dabei wie im Beispielcode den englischen Ausdruck der gewünschten Farbe enthalten, oder den HEX-Code des Farbtons.

5.2.5 Wartezeit

```
Wartezeit = 500
```

Dieser Parameter empfängt in Millisekunden die Zeit, die das Programm zwischen den Generationen pausieren soll. In diesem Fall wäre das eine halbe Sekunde.

5.3 #____SPIELFELD UND ZELLEN

In diesem Abschnitt des Programms wird das Spielfeld erzeugt und dann mit lebenden Zellen besetzt. Dabei kommen erstmals einige der unter *#____VARIABLEN* eingegeben Werte zum Einsatz. Eingriffe hier verändern nicht nur den implementierten Automaten, sondern zum Teil das Programm selbst.

5.3.1 Spielfeld

```
Spielfeld = [ (Seitenlaenge) * [0] for i in range(Seitenlaenge) ]
```

Hier wird das Spielfeld generiert. Python lässt es nicht zu, mehrdimensionale Arrays direkt zu erzeugen. Darum wird mit *(Seitenlaenge) * [0]* zuerst eine Liste von Nullen erzeugt mit der Länge, die als Zahl in der Variable *Seitenlaenge* gespeichert wurde. In dem oben genannten Beispiel war *Seitenlaenge = 150* gesetzt. Es entsteht also eine Liste mit 150 Nullen. Diese eindimensionale Liste wird durch den zweiten Teil des in eckigen Klammern stehenden Arguments *for i in range(Seitenlaenge)* erweitert. Die Liste mit 150 Nullen wird dadurch selbst 150 mal dupliziert. Es entsteht ein – für den Menschen zweidimensional denkbares – Spielfeld mit 150 * 150 = 22.500 Zellen. Entscheidend ist, dass die jetzt in der Liste *Spielfeld* abgelegten Werte gezielt angesteuert, gelesen und verändert werden können.
Hier zeigt sich im Code, dass für die Blockautomaten mit Margolus-Nachbarschaft tatsächlich nur ein Spielfeld erzeugt wird, und keine zwei, wie bei Conways Game of Life oder anderen traditionellen Zellularautomaten. Wie in der CAM-7 ist auch hier kein Puffer notwendig, da stets ein von seinen Nachbarn isolierter Block ausgelesen und ausgetauscht wird. Es besteht keine Gefahr, dass noch benötigte Werte im Spielfeld überschrieben werden.

5.3.2 Zellen

Die lebenden Zellen können einzeln und gezielt, aber auch mithilfe der unter #____IMPORTE geladenen Bibliothek zur Erzeugung von Zufallszahlen mit einer gewissen Wahrscheinlichkeit gesetzt oder eben nicht gesetzt werden.

5.3.2.1 Zellauftrittswahrscheinlichkeit

```
for Zeile in range(Seitenlaenge):
     for Spalte in range(Seitenlaenge):
          Spielfeld[Zeile][Spalte]  =  int  (  random()  +  (
          Zellauftrittswahrscheinlichkeit ) / 100 )
```

Jede Zeile des Spielfelds wird Zelle für Zelle durchgegangen, und diese Zellen werden mit der vom Benutzer unter *#____VARIABLEN* gesetzten Wahrscheinlichkeit als lebendig gesetzt oder tot gelassen. Die Entscheidung, ob die Zelle lebendig wird oder tot bleibt, fällt dabei folgendermaßen:

Das in der dritten Zeile stehende *random()* ist eine Funktion, die vom unter *#____IMPORTE* importierten Zufallszahlengenerator eine Zufallszahl erhält. Diese Zufallszahl wird als Gleitkommazahl übergeben und liegt zwischen 0,0 und 1,0. Python verwendet den Mersenne-Twister Pseudozufallszahlengenerator mit einer Periode von 2^{19937}-1 (Python 2015). Diese Zufallszahlen sind also vollständig determiniert und keineswegs im echten Sinne random.[7]

Die durch *Spielfeld[Zeile][Spalte]* jeweils angesteuerte Zelle im Spielfeld soll aber entweder den Wert 0 zugewiesen bekommen – was bedeutet, dass sie bleibt, wie sie ist – oder den Wert 1, was sie nach der hier verwendeten Nomenklatur lebendig werden lässt. Das heißt, dass nach Ablauf der hier im Programm stattfindenden Operation rechts vom Gleichheitszeichen entweder eine 0 oder eine 1 stehen soll, und keine Gleitkommazahl.

Das *int* sorgt nun schon einmal dafür, dass die Zahl rechts vom Gleichheitszeichen keine Gleitkommazahl, sondern eine ganze Zahl ist. Würde *int* aber nur auf die von *random()* übergebene Gleitkommzahl angewandt, erhielte die Zelle mit 99,9…% Wahrscheinlichkeit den Wert 0 – da alle Werte zwischen 0,0 und 0,9... auf 0 gerundet würden.

Aus diesem Grund und um die vom Benutzer eingestellte Wahrscheinlichkeit zur Grundlage der Berechnung zu nehmen, wird der in der Variable *Zellauftrittswahrscheinlichkeit* gespeicherte Wert durch 100 geteilt, so dass er – je nach Eingabe – ebenfalls zwischen 0,0 und 1,0 liegt. Dieser Wert wird dann zu dem in *random()* übergebenen Wert addiert.

[7] Es wäre interessant, die Ergebnisse dieses Generators einmal mit Hilfe der Ulam-Spirale zu visualisieren, wie es Nikita Brakuinski in seiner Arbeit „Die Spiraldarstellung – Ein experimentelles Visualisierungsverfahren“ vorgestellt hat (Braguinski 2015).

Im oben beispielhaft genannten Fall lag der Wert für die *Zellauftrittswahrscheinlichkeit* bei 10 Prozent. Das heißt, zu der von *random()* übergebenen Zufallszahl wird 10/100 = 0,1 addiert. Daraus folgt, dass die Summe aus beiden Werten zwischen 0,1 und 1,1 liegen muss. Daraus folgt wiederum, dass durch das *int* im Code 9/10 der möglichen Werte auf 0 gerundet werden und 1/10 auf 1. Da jeder Fall mit der gleichen Wahrscheinlichkeit eintritt, ergibt sich mit 90-prozentiger Wahrscheinlichkeit der Wert 0 (wenn der Wert der Summe zwischen 0,1 und 0,99… liegt) und mit 10-prozentiger Wahrscheinlichkeit der Wert 1 (wenn der Wert der Summe zwischen 1,0 und 1,1 liegt). Diese 10 Prozent Wahrscheinlichkeit, dass die Zelle auf 1 und damit als lebendig gesetzt wird, sind genau die 10 Prozent Wahrscheinlichkeit, die der Benutzer für die *Zellauftritts-wahrscheinlichkeit* eingegeben hat.

5.3.2.2 Verdichtete Bereiche

Selbiges gilt für die verdichteten Bereiche, nur dass hier statt der Variable *Zellauftrittswahrscheinlichkeit* die Variable *Verdichtung* aufgerufen wird. Es liegt dabei trotz der gewählten Namen beim Benutzer, in welche Relation er die beiden Variablen setzt.

```
for i in range(int(Seitenlaenge/3), int(Seitenlaenge/3*2)):
      for j in range(int(Seitenlaenge/3), int(Seitenlaenge/3*2)):
            Spielfeld[i][j] = int ( random() + (Verdichtung) / 100 )
```

In diesem Beispiel wird nicht mehr das ganze Spielfeld besetzt, sondern nur ein Quadrat in der Mitte. Das Quadrat hat 1/3 der Seitenlänge des Spielfelds. Welche Form, Lage und Größe das mit der Verdichtungsvariable erzeugte Objekt hat, lässt sich über die Parameter in den beiden for-Schleifen bestimmen.

5.3.2.3 Einzelzellen

Es können auch einzelne Zellen gesetzt werden. Dazu wird die Zelle direkt angesteuert:

```
Spielfeld[50][52] = 1
```

In diesem Fall wird die Zelle am Kreuzungspunkt von Zeile 50 und Spalte 52 als lebendig gesetzt. Die drei Arten, Zellen auf das Spielfeld zu setzen, können je nach Bedarf kombiniert und erweitert werden.

5.4 #____TRIGGER

Bis jetzt wurden die nötigen Bibliotheken importiert und die Parameter des Automaten durch den Benutzer eingegeben. Außerdem wurde das Spielfeld und die Ausgangssituation an Zellen erzeugt. Bis hierhin ist, abgesehen von der Kennziffer, nur ersichtlich, dass es sich um den Aufbau eines allgemeinen zellulären Automaten handeln könnte. Nun folgt das erste Element, das in seiner Funktion in jedem Margolus-Automaten auf die eine oder andere Weise implementiert sein muss:
Es wird eine weitere globale Variable definiert, und zwar mithin die einzige, die der Benutzer gar nicht oder nur mit Nachdenken verändern sollte: der Trigger.

```
Trigger = -1
```

Der Trigger wird zu Beginn auf -1 gesetzt. Vorgesehen ist er für Folgendes:
Damit das Programm zwischen den zwei Gittern wechseln kann, die ein Margolus-Automat alternierend verwendet, benötigt es einen Indikator dafür, welches der beiden Gitter gerade angesteuert werden muss. Dieser Indikator ist der Trigger. Er triggert das Programm dazu, sich für eines der beiden Gitter zu entscheiden. Der Trigger alterniert zwischen -1 und 1, indem er später im Programm vor jeder Generation mit (-1) multipliziert wird. Ist der Trigger gleich 1, wird das Programm das Gittermuster Nr. 1 ansteuern. Die nächste Generation wird berechnet, worauf der Trigger wieder mit (-1) multipliziert wird. Er hat nun den Wert -1, was das Programm dazu triggert, das Gittermuster Nr. 2 anzusteuern, und auf dessen Basis die nächste Generation zu berechnen. Schließlich wird der Trigger wieder mit (-1) multipliziert und das Ganze beginnt von vorn.[8]

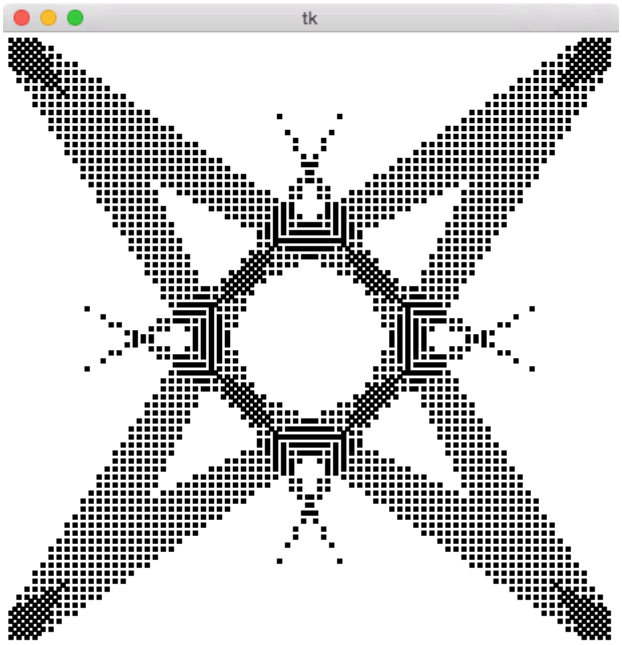
[8] Johannes Maibaum hat zurecht darauf hingewiesen, dass der Trigger eleganter mit Boolschen Variablen zu gestalten gewesen wäre. Da es dem Geist des Programms entspricht, möglichst wenig Vorwissen in Sachen Programmieren zu fordern, wird hier die umständlichere aber mathematisch leichter nachvollziehbarere Variante der Vorzeichenumkehr angewandt.

5.5 #____CANVAS

In diesem Abschnitt wird die graphische Ausgabe des zellulären Automaten initialisiert und der Canvas, d. h. die Leinwand, auf der die Zellen des Automaten sichtbar werden, erstellt. Außerdem wird eine while-Schleife geöffnet, in der Python die jeweils nächste Generation berechnet.

```
master = Tk()

w = Canvas(master, width=(4 + (Seitenlaenge * Zellgroesse) + 1),\
           height= (4 + (Seitenlaenge * Zellgroesse) + 1) )
w.pack()

done = 0

def finish(e):
    global done
    done = 1

master.bind("<Control-c>", finish)

while not done:
```

Da die graphische Ausgabe ganz eigene Prämissen und Probleme mit sich bringt, die nicht zum Verständnis des Margolus-Automaten beitragen, wird auf eine nähere Betrachtung des Codes hier verzichtet.



Die graphische Ausgabe Tkinter mit dem Programm MANDALA.

5.6 #____PROGRAMM

Nun sind alle notwendigen Parameter eingegeben. Das Spielfeld ist mit lebenden Zellen besetzt, die Triggervariable erstellt und auch die graphische Ausgabe ist initialisiert. Jetzt kann das eigentliche Programm ablaufen, das von der alten Generation zum Zeitpunkt *t* ausgehend die nächste Generation zum Zeitpunkt *t*+1 berechnet.

5.6.1 Auswahl des Gitters

Bevor das Lesen der alten Zellblöcke und die Berechnung und das Schreiben der neuen Zellblöcke erfolgen kann, müssen zwei Dinge geschehen: Zum Einen muss entschieden werden, welches Gitter angesteuert werden soll – zum Anderen müssen die entsprechenden Vorkehrungen getroffen werden, damit das entsprechende Gitter auch tatsächlich bearbeitet wird:

5.6.1.1 Trigger alterniert

Um dem Programm mitzuteilen, welche Gitterstruktur gerade angesteuert werden soll, d. h. ob die Viererblöcke verschoben liegen sollen oder nicht, wechselt der Trigger wie bereits beschrieben vor jeder Generation das Vorzeichen. Dazu wird er gleich zu Beginn der Schleife mit (-1) multipliziert.

```
Trigger = Trigger * -1
```

Der Trigger wechselt also beim ersten Durchlauf des Programms mit dieser Operation von -1 zu 1. Er wird, wenn die nächste Generation fertig berechnet und auf den Bildschirm gebracht ist, durch die selbe Operation wieder auf -1 wechseln und alterniert damit stets zwischen diesen beiden Werten.

5.6.1.2 Trigger wird abgefragt

Je nachdem, ob der Trigger nun 1 oder -1 ist, wird das eine oder das andere Gitter angesteuert. Das geschieht, indem die Variable *Gitterverschiebung* in den verschiedenen Fällen mit anderen Werten geladen wird:

```
if Trigger == 1:
     Gitterverschiebung = 0
else:
     Gitterverschiebung = 1
```

Ist der Trigger wie zu Beginn des Programms gleich 1, dann wird die Variable *Gitterverschiebung* auf 0 gesetzt. Ist der Trigger -1, wird diese auf 1 gesetzt. Welchen Sinn hat das?

5.6.2.3 Gittersteuerung

```
Zeile = Gitterverschiebung
Spalte = Gitterverschiebung
```

Im folgenden wird sich zeigen, dass die Variablen *Zeile* und *Spalte* – deren Wert bestimmt, welche Zellen gerade zu lesen oder zu schreiben sind – sukzessive um den Wert 2 erhöht werden, bis alle Zellen einbezogen wurden. Dadurch kommen alle Viererblöcke zur Analyse. Wenn nun die Variable *Gitterverschiebung* zum Startwert von *Zeile* und *Spalte* addiert wird, ändert sich nichts, solange *Gitterverschiebung* gleich 0 ist. Ist *Gitterverschiebung* aber gleich 1, dann beginnt die Analyse und das Schreiben der Viererblöcke nicht mehr bei Zeile und Spalte 0, sondern bei Zeile und Spalte 1, und schreitet von hier aus in 2er Schritten voran. Es werden dann also Viererblöcke untersucht, die gegenüber den zuerst untersuchten in Breite und Höhe um jeweils eine Zelle verschoben sind: Das andere Gitter wird angesteuert (vgl. Kapitel 5.6.2.7).

5.6.2 Hauptschleife

Jetzt, nachdem entschieden ist, welches Gitter angesteuert werden muss und die Variablen dafür korrekt gesetzt sind, kann das eigentliche Programm beginnen. In jedem Schleifendurchlauf wird nun der alte Block analysiert und so die Nummer dieses Blocks ermittelt. Mit dieser Nummer des alten Blocks wird aus der Kennziffer des jeweiligen Automaten die Nummer des neuen Blocks gelesen, der an die Stelle des alten Blocks geschrieben werden soll. Aus dieser Nummer des neuen Blocks wiederum wird der neue Block selbst synthetisiert und seine Zellen in das Spielfeld eingeschrieben. Schließlich werden entweder *Spalte* oder *Zeile* und *Spalte* um 2 erhöht, so dass beim nächsten Durchlauf der entsprechend nächste Block betrachtet wird. Das geschieht solange, bis alle Blöcke im gerade betrachteten Gitter bearbeitet wurden und die Hauptschleife für diese Generation ihr Ende erreicht hat. Im Einzelnen geschieht Folgendes:

5.6.2.1 Öffnen der Hauptschleife

Jetzt wird also die Hauptschleife mit den Durchlaufvariablen *Zeile* und *Spalte* geöffnet. Das gesamte folgende Programm zur Berechnung und zum Schreiben der nächsten Generation findet innerhalb der jetzt geöffneten und ineinander verschachtelten while-Schleifen statt:

```
while Zeile < Seitenlaenge - 1:
      Spalte = Gitterverschiebung
      while Spalte < Seitenlaenge - 1:
```

Die Zählvariablen *Zeile* und *Spalte* müssen kleiner sein als die *Seitenlaenge – 1* (in Worten: Seitenlaenge minus Eins). Andernfalls würde das Programm versuchen, Zellen zu lesen, die nicht mehr auf dem *Spielfeld* liegen und es käme zum Absturz des Programms. Die Bedeutung der zweiten Zeile ist nicht auf den ersten Blick erkennbar. Sie ist aber notwendig, da die Variable *Spalte*, wenn die innere while-Schleife komplett durchgelaufen ist – wenn also eine Zeile des Spielfelds bearbeitet wurde – wieder auf den durch die *Gitterverschiebung* übergebenen Wert gesetzt werden muss. Sonst wäre *Spalte* noch vom letzten Durchlauf der inneren while-Schleife >= *Seitenlaenge - 1*. Es würde dann gar keinen Durchlauf der inneren while-Schleife mehr geben, und alle anderen Zeilen des Spielfelds würden unbearbeitet bleiben.

5.6.2.2 LESEN des alten Blocks

Zuerst werden nun die vier Zellen des gerade angesteuerten alten Blocks eingelesen und ihre Werte für die weitere Bearbeitung in vier Variablen gespeichert:

```
Zelle0 = (Spielfeld[Zeile][Spalte])
Zelle1 = (Spielfeld[Zeile][Spalte+1])
Zelle2 = (Spielfeld[Zeile+1][Spalte])
Zelle3 = (Spielfeld[Zeile+1][Spalte+1])
```

5.6.2.3 NUMMER des alten Blocks

Fasst man die vier Zellzustände, die ein spezieller Block zu einem diskreten Zeitpunkt *t* in einem Margolus-Automaten hat – also das spezifische Muster der Nullen und Einsen in der Reihenfolge seiner vier Zellen – zu einer Binärzahl zusammen, dann entspricht

diese Zahl dem graphisch dargestellten Block. Diese Binärzahl gibt zugleich die Nummer des entsprechenden Blocks an (vgl. Kapitel 2.1 und vor allem 5.2.1). Wie bereits erläutert, werden die Zellzustände in ihrer Reihenfolge zu einer Binärzahl konkateniert und diese Binärzahl in eine Dezimalzahl umgewandelt. Diese Dezimalzahl ist die Nummer des analysierten alten Blocks. Diese beiden Operationen leistet die folgende Codezeile:

```
Alter_Block =  (Zelle3 * 8) + (Zelle2 * 4) + (Zelle1 * 2) + (Zelle0 * 1)
```

Programmtechnisch wird die zur Erläuterung des Vorgehens nötige Konkatenation der Zellzustände übersprungen. Stattdessen werden die einzelnen Stellen der Binärzahl sofort mit dem dezimalen Faktor multipliziert, den sie in der Binärzahl vertreten, und die erhaltenen Werte dann addiert. Mit dieser dann in der Variable *Alter_Block* gespeicherten dezimalen Nummer kann jetzt in der Kennziffer des Automaten nachgeschlagen werden, welcher neue Block an die Stelle des alten geschrieben werden muss (vgl. dazu Kapitel 5.2.1).

5.6.2.4 REGELANWENDUNG

Jetzt wird die Liste *Kennziffer* befragt. Aufgrund der in Kapitel 5.2.1 beschriebenen Zuordnung der Nummern der alten Blöcke zu den neuen Blöcken steht die Nummer des gesuchten neuen Blocks an der Stelle in der Liste *Kennziffer*, die von der Nummer des alten Blocks bezeichnet wird. Die Codezeile

```
Neuer_Block = int(Kennziffer[Alter_Block])
```

holt darum den Wert für den neuen Block von der durch die Nummer des alten Blocks bezeichneten Stelle in der Liste *Kennziffer* und speichert diesen Wert in der Variable *Neuer_Block*. Damit ist die Nummer des neuen Blocks zur weiteren Verwendung bereit.

5.6.2.5 NUMMER des neuen Blocks

So, wie die Nummer des alten Blocks aus dessen Zellzuständen bestimmt werden konnte (Kapitel 5.6.2.3), so kann umgekehrt auch die Nummer des neuen Blocks in die benötigten Zellzustände umgewandelt werden. Das leisten die folgenden Zeilen:

```
if Neuer_Block >= 8:
      Zelle3 = 1
      Neuer_Block = Neuer_Block - 8
       else:
      Zelle3 = 0

if Neuer_Block >= 4:
      Zelle2 = 1
      Neuer_Block = Neuer_Block - 4
else:
      Zelle2 = 0

if Neuer_Block >= 2:
      Zelle1 = 1
      Neuer_Block = Neuer_Block - 2
else:
      Zelle1 = 0

if Neuer_Block >= 1:
      Zelle0 = 1
else:
      Zelle0 = 0
```

Diese Routine bestimmt anhand des Wertes in *Neuer_Block* – der Nummer des noch zu schreibenden neuen Blocks – welche Zellen im Viererblock als lebendig gesetzt werden und welche tot sind. Dazu wird die dezimale Nummer wieder in ihre binäre Entsprechung umgewandelt. Für die vierte Zelle *Zelle3* im Viererblock wird dabei zu Beginn wie folgt vorgegangen:

Die Routine macht sich zu Nutze, dass die Dezimalzahl, wenn ihre binäre Entsprechung mit einer 1 beginnen soll, größer gleich dezimal 8 sein muss. Ist die Nummer also größer gleich 8, muss die erste Stelle der Binärzahl gleich 1 gesetzt werden.

Die erste Stelle der Binärzahl ist aber aufgrund der isomorphen Zuordnung nichts anderes als die vierte Zelle im Viererblock, die dort ebenfalls die dezimale 8 repräsentiert. Also wird die Variable *Zelle3*, die in unserer Zählweise die vierte Zelle repräsentiert, gleich 1 gesetzt, wenn *Neuer_Block* größer gleich 8 ist. Darauf muss nur noch die „verbrauchte“ 8 von der dezimalen Nummer abgezogen werden.

Wäre die dezimale Nummer kleiner als 8 gewesen, dann könnte die erste Stelle der Binärzahl nicht 1 sein. Die erste Zelle wäre dann gleich 0 gesetzt worden, und es würde nichts von der dezimalen Zahl abgezogen werden.

Dieser Test wird der Reihe nach für die vier Zellen und die ihnen entsprechenden Stellen der Binärzahl durchgeführt, bis alle Zellen entweder den Wert 1 oder den Wert 0 zugewiesen bekommen haben.

5.6.2.6 SCHREIBEN des neuen Blocks

Die Nummer des Neuen Blocks wurde erstellt und den vier Zellen des neuen Blocks entsprechend entweder 0 oder 1 zugewiesen. Die Werte dieser Zellen müssen nun noch ins Spielfeld geschrieben werden. Dazu werden wieder die in *Zeile* und *Spalte* gespeicherten Werte verwendet, um zu bestimmen, wo im Spielfeld die Zellen des neuen Block gespeichert werden sollen.

```
(Spielfeld[Zeile][Spalte]) = Zelle0
(Spielfeld[Zeile][Spalte+1]) = Zelle1
(Spielfeld[Zeile+1][Spalte]) = Zelle2
(Spielfeld[Zeile+1][Spalte+1]) = Zelle3
```

5.6.2.7 Inkrementieren und Schließen der Hauptschleife

Das Ende der Hauptschleife ist erreicht. Der alte Block wurde analysiert, die Nummer des neuen Blocks aus der *Kennziffer* des Automaten ausgelesen und der neue Block synthetisiert, so dass die Zellen des Blocks in das Spielfeld geschrieben werden konnten. Jetzt fehlt nur noch die Erhöhung der Zählvariablen *Zeile* und *Spalte* für den nächsten Durchlauf.

```
      Spalte = Spalte + 2
Zeile = Zeile + 2
```

Die Variablen werden um 2 inkrementiert, da der nächste zu bearbeitende Block 2 Zellen weiter zu finden ist. Zu beachten ist hierbei: Solange der Wert in *Spalte* den Wert von *Seitenlaenge -1* nicht erreicht oder überschritten hat, wird nur die Variable *Spalte* um 2 erhöht. Erst wenn der Wert von *Spalte* größer gleich *Seitenlaenge -1* ist, wird die innere while-Schleife beendet, und *Zeile* um 2 erhöht. Die Blöcke des Spielfelds werden also von links nach rechts und Zeile für Zeile von oben nach unten bearbeitet – zumindest in der hier gewählten Darstellungsweise.

5.6.3 Wartezeit

Nach der Berechnung aller Blöcke der nächsten Generation, die jetzt im *Spielfeld* bereit liegen, wird je nach Einstellung der Variable *Wartezeit* eine bestimmtes Zeitintervall pausiert.

```
time.sleep( float (Wartezeit) / 1000)
```

5.6.4 Graphische Ausgabe

Jetzt muss das Spielfeld auf den Canvas gebracht werden, damit es auf dem Bildschirm erscheint:

```
w.delete(ALL)

for Zeile in range(Seitenlaenge):
      for Spalte in range(Seitenlaenge):
            if Spielfeld[Zeile][Spalte] == 1:
                  w.create_rectangle((Spalte*Zellgroesse)   +   4,
                  (Zeile*Zellgroesse)+(Zellgroesse + 4), \
                  (Spalte*Zellgroesse)+(Zellgroesse + 4),
                  (Zeile*Zellgroesse) + 4, \
                  fill=Farbe, outline=Farbe)

master.update()
```

Zuerst wird die vorherige Generation vom Canvas gelöscht. Dann werden die Zählvariablen *Zeile* und *Spalte* verwendet, um auf Basis der vom Benutzer eingegebenen Werte das Spielfeld auf den Canvas zu übertragen.

6. Ausführbare Beispiele

6.1 Allgemeines

Die laufenden Programme werden mit der Tastenkombination <ctrl>+<c> beendet.

Im vorhergehenden Kapitel wurde der Code des Programms einem Close-Reading unterzogen, und so eine Möglichkeit vorgestellt, wie die Familie der zellulären Blockautomaten mit Margolus-Nachbarschaft implementiert werden kann. So überschaubar das System der Blockautomaten mit Margolus-Nachbarschaft formal erscheinen mag, so komplex und unvorhersehbar ist das Verhalten der tatsächlichen Implementierungen für die menschliche Wahrnehmung. Selbst Änderungen allein im Regelset oder nur in den Startbedingungen können zu erstaunlichen Abweichungen führen und rufen einen menschlichen Beobachter zu immer neuen Interpretationen auf. Es entsteht der Eindruck von Gasgemischen und Gaswellen, auf dem Bildschirm winden sich rechteckige Wucherungen oder mehr oder weniger ästhetisch schwingende Gebilde von amorpher Gestalt. Das Wachstum von Schneeflocken lässt sich genauso in das Verhalten dieser Zellularautomaten hineinlesen wie das Rieseln von Sand.

Die meisten der folgenden Programme bzw. die dieser Arbeit beiliegenden Dateien wurden nicht mit dem Namen beschriftet, den ihre Erfinder oder deren Epigonen den jeweiligen Regelsätzen gaben, sondern nach dem mehr oder weniger arbiträren Eindruck, den diese Automaten im Vollzug auf den Autor gemacht haben. Der Leser mag selbst entscheiden, inwieweit diese Titel dem Erscheinungsbild des jeweiligen Automaten angemessen sind. Eine Ausnahme von dieser Nomenklatur bilden lediglich die Dateien SAND und UNIVERSAL – aus Gründen, die noch verständlich werden.
Im Vergleich zu den Bildern von Margolus und Toffoli in Kapitel 4 fallen die hier vorgestellten Umsetzungen auffallend rudimentär aus. Sie sind schlicht zu klein, d. h. sie beinhalten zu wenige Zellen, um mehr als eine minimalistische Ästhetik zu liefern. Damit sind die folgenden Automaten auch für die Ansprüche von Margolus und Toffoli zu klein. Diese schrieben über ihre Beobachtungen an zellulären Automaten:

> As soon as the numbers involved become large enough for averages to be meaningful – say, averages over spacetime volume elements containing thousands of particles and involving thousands of collisions – a definite continuum dynamics emerges. (Margolus/Toffoli 1990b, 265)

Auch der Leser wird trotz der ihm zur Verfügung gestellten Variable *Seitenlaenge* (vgl. Kapitel 5.2.2) keine „thousands of particles“ ausgegeben erhalten. Das hat technische Gründe: In der hier verwendeten Art und Weise ist die graphische Ausgabe Tkinter schlicht zu langsam, als dass sie die für ein Kontinuum benötigte Anzahl von Zellen in ausreichender Geschwindigkeit darstellen könnte. Die Ausgaben der folgenden Programme bleiben weitgehend im Abstrakten.
Diese scheinbare Not hat aber auch ihre Tugend: Sie beugt der Gefahr vor, dass die allzu große Wirkmacht der technischen Bilder deren Technizität verschleiert. Es lassen sich zwar schnell Interpretationen finden, was diese Bilder zeigen könnten. Das Medium, so wie es hier verwendet wird, ist aber – wenn man so will – im Sinne McLuhans ein kaltes Medium. Dahinter steht die Hoffnung, dass dieser kalte Status dem medien-archäologischen Blick entgegenkommt, und dass, während ein heißes Medium nur noch bei einer Störung einen Hinweis auf sein technisches Sein gibt, ein kaltes, da es den Nutzer nicht vereinnahmt und aufheizt, Spielraum für eigene Deutungen und den kritischen analytischen Blick lässt.

Die im folgenden besprochenen Automaten liegen dieser Arbeit bei und können über Terminal oder Konsole wie bei Pythonprogrammen üblich mit dem Befehl *python* und dem Namen des Programms plus der Endung *.py* aufgerufen werden.

Liste der beiliegenden Automaten in alphabetischer Reihenfolge:

Amorph_block.py
Amorph_string.py
Gaswelle.py
Mandala.py
Muecken.py
Sand.py
Toothpick.py
Universal.py

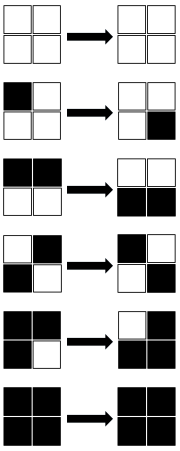
Die laufenden Programme werden mit der Tastenkombination <ctrl>+<c> beendet.

6.2 Ausgewählte Programme

6.2.1 GASWELLE – HPP-Gas

Kennziffer = ['0', '8', '4', '12', '2', '10', '9', '14', '1', '6', '5', '13', '3', '11', '7', '15']

Das Programm GASWELLE enthält eine Version des bereits erwähnten HPP-Gases. Dieses wurde entwickelt von Hardy, de Pazzis und Pomeau – daher das Akronym HPP. Es war das erste vollständig deterministische Gittergas mit diskreten Zeitschritten, diskreten Positionen und diskreter Geschwindigkeit (Pomeau et al. 1987, 650).



Regelset HPP-Gas

GASWELLE.PY

Was an früherer Stelle zur Eignung von zellulären Automaten für die Simulation von Fluiden gesagt wurde, lässt sich hier gut erkennen: Während die mikroskopischen Eigenschaften des HPP-Gases gegenüber dem Verhalten natürlicher Gaspartikel geradezu fahrlässig vereinfacht erscheinen – so haben alle Teilchen des simulierten Gases nicht nur dieselbe Masse und Geschwindigkeit und bewegen sich auf nur zwei Achsen, es tritt zusätzlich auch ein unnatürliches Verhalten der Teilchen auf, wenn diese auf den Rand treffen –, sind die makroskopischen Eigenschaften des HPP-Gases in vielerlei Hinsicht mit denen physischer Gase identisch (Margolus/Toffoli 1987b, 124).

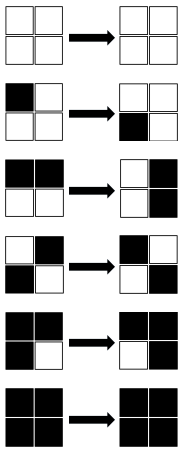
Da die Regel rotationssymmetrisch ist, wurden hier, entsprechend der bereits bekannten Zuordnungen der Blöcke in Kapitel 2.2, nur sechs Blöcke abgebildet. Die Abbildung lässt weiter sehen, dass in der rechten Spalte jeder Block nur jeweils einmal auftritt. Das HPP-Gas ist also reversibel. Das Gas ist auch bestandsstabil, d. h. die Anzahl der Zellen auf dem Spielfeld bleibt stets gleich. Das liegt ganz einfach daran, dass jeder neue Block genauso viele lebende Zellen enthält, wie der Block, den er ersetzt (vgl. hierzu Kari 1990, besonders 379).

Die Ausgangsbesetzung der ersten Generation im Programm GASWELLE ist ein eher lockeres Auftreten von Zellen im gesamten Spielfeld, während sich an der linken Seite zusätzlich ein Bereich erhöhter Zelldichte findet. Gut erkennbar ist das Fluktuieren der dichteren Welle über das Spielfeld, während sie sich nach und nach auflöst und die Entropie im System zunimmt (vgl. dazu auch das folgende Kapitel).

6.2.2 MUECKEN – RotationsIII

Kennziffer = ['0', '4', '1', '10', '8', '3', '9', '11', '2', '6', '12', '14', '5', '7', '13', '15']

RotationsIII ist eine Regel von Tim Tyler, der an anderer Stelle dieser Arbeit bereits Erwähnung fand. Auch das Verhalten von RotationsIII kann als das eines Gases interpretiert werden, nämlich als Diffusion. Ausgangslage ist in MUECKEN ein solider Block zentral in der Mitte des Spielfelds. Durch die Bewegung der Zellen erweckt dieser den Eindruck eines simplifizierten Mückenschwarms, wie man ihn vielleicht aus diversen Zeichentrickfilmen kennt. Im Laufe der Zeit verteilen sich die Zellen mehr und mehr über das Spielfeld, manche erreichen sogar den Rand und bewegen sich gegen den Uhrzeigersinn an diesem entlang.



Regelset RotationsIII

MUECKEN.PY

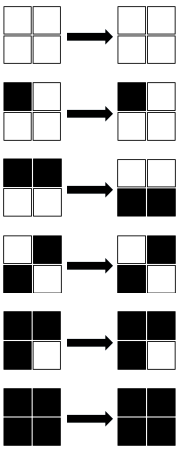
Ein Blick auf das Regelset legt nahe, dass sich im Verhalten des Automaten tatsächlich Rotationsbewegungen zeigen werden – was nicht mit der Rotationssymmetrie der Regel zu verwechseln ist! Ein einzelne Zelle auf dem Spielfeld ist es, die sich hier nicht wie im HPP-Gas linear auf der Diagonale weiterbewegt, sondern an Ort und Stelle *im* Uhrzeigersinn rotiert. Man könnte ausgehend von den Blöcken in der zweiten Zeile der Abb. links vermuten, es müsse eine Bewegung *gegen* den Uhrzeigersinn entstehen. Die Gitterverschiebung aber sorgt dafür, dass diese Bewegung umgekehrt abläuft. Eine Regel mit gegen den Uhrzeigersinn rotierenden Zellen ist allerdings leicht abzuleiten: Es reicht, die Regel umzukehren.

Die Regel RotationsIII wurde im Vergleich zum HPP-Gas an vielen Punkten verändert. Wie die Abbildungen zeigen, blieben nur der leere Block, die Zuweisung in der vierten Zeile und der gefüllte Block gleich. Die Veränderungen bewirken, dass jede Art von linearer Bewegung, wie sie im HPP-Gas ein wesentliches Charakteristikum war, ausgeschlossen wurde. Stattdessen zeigt dieser Automat lokale Bewegungen, die nur bei Kontakt der Einzelzellen untereinander – oder bei Kontakt mit dem Rand – globale Auswirkungen und wahrscheinlich eine Zunahme der Entropie zur Folge haben.[9]

[9] Die Frage nach der Entropie in diesen Automaten ist nicht trivial: Lokal nimmt sie tatsächlich zu, während sie global konstant bleibt. Da die Automaten HPP-Gas und RotationsIII reversibel sind, drücken sie ihre Ausgangslage nur auf immer kompliziertere Weise aus (vgl. Margolus 1984, 83).

6.2.3 AMORPH – StringThingII

Kennziffer = ['0', '1', '2', '12', '4', '10', '6', '7', '8', '9', '5', '11', '3', '13', '14', '15']



Regelset StringThingII

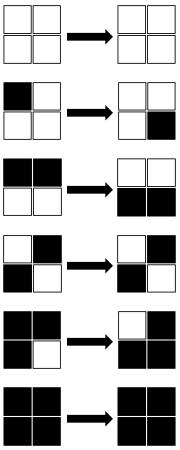
AMORPH_BLOCK.PY

AMORPH_STRING.PY

Auch die Regel StringThingII wurde von Tim Tyler entwickelt. Die Ausgangslage, die in AMORPH_BLOCK.PY implementiert wurde, ist dieselbe wie in MUECKEN, ein relativ hoch verdichteter Block zentral in der Mitte des Spielfelds. Beim Ausführen der Datei zeigt sich trotz der gleichen Ausgangslage ein ganz anderes Verhalten und die Interpretation, es sei das von Gasteilchen lässt sich hier nicht länger halten. Für eine weitere Überraschung mag das Ausführen der zweiten Datei AMORPH_STRING.PY sorgen. Hier zeigt sich die Bewegung, die für das Verhalten des Automaten charakteristisch ist, die sich so aber im komplexen Ablauf zuvor kaum erkennen ließ.

6.2.4 MANDALA – SwapOnDiag

Kennziffer = ['0', '8', '4', '12', '2', '10', '6', '14', '1', '9', '5', '13', '3', '11', '7', '15']



Regelset SwapOnDiag

MANDALA.PY

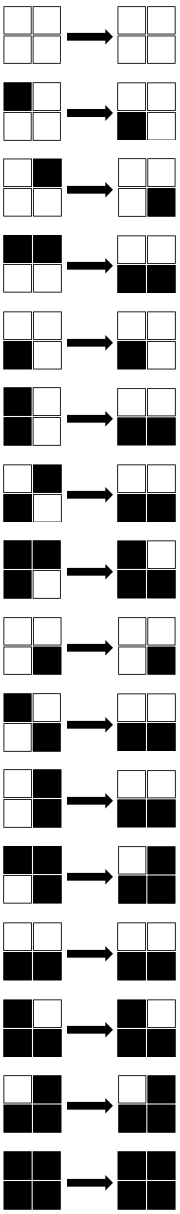
Die Regel SwapOnDiag wurde von Margolus und Toffoli entwickelt. Hier zeigt sich das, was man die Kraft der kleinen Veränderungen nennen könnte: SwapOnDiag ist nämlich regeltechnisch dem HPP-Gas aus dem ersten Beispiel GASWELLE nahezu identisch: In der Abbildung ist nur ein einziger Unterschied in der Zuordnung zu erkennen: Block 6 und Block 9 in der vierten Zeile werden nicht einander zugeordnet, sondern bleiben fest. Auch ein Vergleich der Kennziffer des Automaten zeigt: Nur die Zahlen 6 und 9 an ebendiesen Stellen wurden vertauscht.

Dieser kleine Unterschied zeigt sich als für die Interpretation bzw. die Simulationsmöglichkeiten des Automaten gravierend: Während im HPP-Gas die simulierten Partikel miteinander kollidieren, ist das in SwapOnDiag ausgeschlossen. Die Teilchen passieren einander ohne Interaktion.

6.2.5 SAND – Sand

Kennziffer = ['0', '4', '8', '12', '4', '12', '12', '13', '8', '12', '12', '14', '12', '13', '14', '15']

Wie das Ausführen der Datei zeigen wird, bleiben einem Betrachter im Falle der Regel Sand nur wenig interpretatorische Möglichkeiten. Aus diesem Grund trägt das Programm hier schlicht denselben Namen wie die Regel selbst: SAND.



Regelset Sand

SAND.PY

Sand ist keine rotationssymmetrische Regel, darum müssen alle 16 Blöcke angegeben werden – bestimmte Zuordnungen sind zwar zueinander achsensymmetrisch, aber das wird hier vernachlässigt. Sand kann nicht rotationssymmetrisch sein, da die Zellen innerhalb dieser Regel einer bestimmten Entwicklungstendenz folgen: Die Zellen sollen nach unten wandern. Folgerichtig gibt es in der Abbildung links auch keine Zuordnung, bei der die Zelle in einem Block nach oben steigen würde. Jede Zelle bleibt entweder an ihrem Platz – nämlich wenn sie auf der unteren Blockkante liegt oder zwei Nachbarzellen unter sich hat – oder sie fällt auf den direkt unter ihr liegenden leeren Platz bzw. schräg neben eine unter ihr liegende einzelne Zelle (vgl. Zeile 8 in der Abb. links).

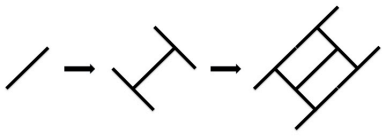
Die Wirkung von Sand ist selbst bei der hier implementierten minimalistischen Darstellung zumindest insofern verblüffend, als es einem menschlichen Beobachter schwer fällt, die eigene Mustererkennung auszuschalten und den Automaten ohne sofortige Zuschreibung von Eigenschaften wie Gravitation und Masse zu betrachten. Sand ist deswegen ein hervorragendes Beispiel, um der Frage nachzugehen, inwieweit zelluläre Automaten tatsächlich physikalischen Prozessen entsprechen, oder inwiefern sie diesen nur aufgrund mehr oder weniger arbiträrer Zuschreibungen in ausreichender Weise ähnlich sind.

SAND enthält als Ausgangslage einen nach rechts verschobenen Block. Dieser wurde durch zwei zufällig getaktete Zuflüsse am oberen Rand ergänzt, so dass stetig neue Zellen im Automaten erscheinen.

6.2.6 SCHNEEFLOCKE – Toothpick

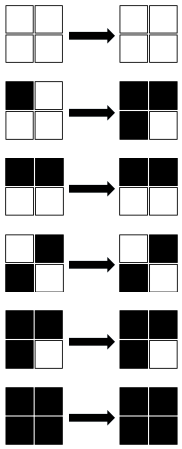
Kennziffer = ['0', '7', '11', '3', '13', '5', '6', '7', '14', '9', '10', '11', '12', '13', '14', '15']

Zugegeben, die im Programm SCHNEEFLOCKE entstehende Form erinnert nur entfernt an eine tatsächliche Schneeflocke. An einen Toothpick – einen Zahnstocher – aber ebenso wenig. Das liegt daran, dass der Automat Toothpick eigentlich anders dargestellt und implementiert wird als hier. Die originale Regel für Tootphick lautet: Zu Beginn befindet sich ein einzelner Zahnstocher auf dem sonst leeren Spielfeld. Im nächsten Schritt wird an jedes freie Ende des Zahnstochers orthogonal ein weiter Zahnstocher angebracht.



Diese Regel wird weiter befolgt, indem in jedem Schritt an jedes freie Ende eines Zahnstochers ein weiter Zahnstocher quer dazu angefügt wird.

Im Margolusautomaten wird ein Toothpick durch einen Block mit zwei diagonal zueinander stehenden Zellen repräsentiert, wie unten in der vierten Zeile abgebildet. Dass diese Regel so im Margolusautomaten anders dargestellt und anders implementiert wurde, tut der Äquivalenz zwischen den beiden Regelsätzen allerdings keinen Abbruch.



Regelset Toothpick

SCHNEEFLOCKE.PY

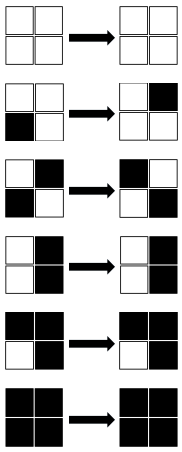
Auch die Regel Toothpick ist rotationssymmetrisch. Sie unterscheidet sich aber in einem wesentlichen Punkt von allen bisher beschriebenen Beispielen: Toothpick ist die erste der hier beschriebenen Regeln, die so etwas wie Wachstum ermöglicht, d. h. diese Regel ist nicht bestandsstabil. Wer TOOTPHICK.PY ablaufen lässt, muss denn auch feststellen, dass der Automat mit jedem Schritt langsamer wird, bis er fast steckenzubleiben scheint, zumindest für ungeduldige Geister. Das liegt an der Art und Weise, wie die Leinwand Tkinter hier verwendet wird: Anstatt nur die neu hinzukommenden Zellen auszugeben und die alten einfach stehenzulassen, werden alle Zellen jedes mal gelöscht und neu geschrieben (vgl. Kap. 5.6.4).

Die Ausgabe durch Tkinter ist also nicht zeitinvariant, sondern ihm höchsten Maße zeitkritisch. Warum ist das bisher nicht aufgefallen? Weil die bisherigen Programme alle bestandsstabil waren, und Tkinter zu jedem Zeitschritt gleich viele Zellen auszugeben hatte. Die alten Zellen stehenzulassen würde das Problem allerdings nicht ganz lösen, sondern nur eindämmen, da die Zahl der neuen Zellen variieren wird.

6.2.7 UNIVERSAL – BBMCA

Kennziffer = ['0', '8', '4', '3', '2', '5', '9', '7', '1', '6', '10', '11', '12', '13', '14', '15']

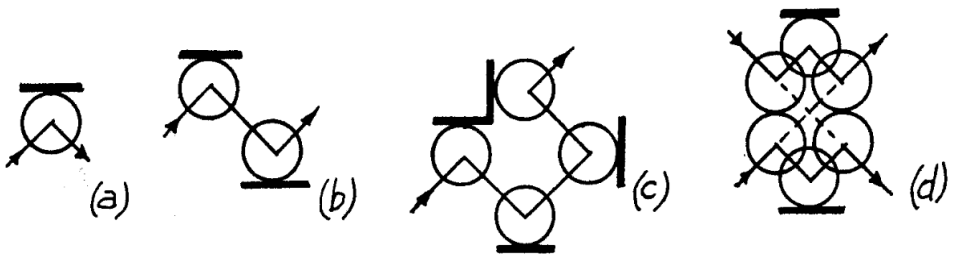
Die Datei UNIVERSAL.PY enthält die Kennziffer der Regel, die schon in Kapitel 2.2 und 2.4 als Beispiel und in Kapitel 5.2.1 bei der Beschreibung des Programms begegnet ist. Diese Regel ist das BBM, das *billiard ball model*, oder genaugenommen – und auch etwas umständlicher – der BBMCA, der *billiard ball model cellular automaton*.



Regelset BBMCA
UNIVERSAL.PY

Die Regel BBMCA ist rotationssymmetrisch und auch reversibel, wie an dem Auftreten der Blöcke in der rechten Spalte zu sehen ist. Wieder ist jeder Block nur einmal vorhanden. Diese Zuordnung, in der jeder Block der linken Seite nur auf einen Block der rechten Seite zielt und nie auf zwei oder mehr, ist eine *injektive* Funktion. Diese Injektivität ist hier hinreichende Bedingung dafür, dass der Automat reversibel ist. Die Blöcke auf der linken Seite wurden hier allerdings nicht wie in den vorhergehenden Beispielen in der Reihenfolge ihres Auftretens in der Kennziffer aufgeführt. Das liegt einfach daran, dass die Darstellung gewählt wurde, wie sie Margolus selbst verwendet hat (vgl. Margolus 1984, 86).

Der BBMCA geht auf das genannte BBM zurück, das Edward Fredkin und Tommaso Toffoli 1982 beschrieben haben. Die Quintessenz dieses Modells ist einfach: Ideal gedachte Billiardkugeln, die miteinander oder mit festen Hindernissen – mit sogenannten Spiegeln (mirrors) – kollidieren, und in beiden Fällen ihre Richtung, aber nicht ihre Geschwindigkeit ändern, können als Grundelemente herangezogen werden, um logische Gates und letztendlich einen turingvollständigen Computer zu konstruieren.



Verschiedene Konstellationen von Spiegeln und abprallenden Billiard-Bällen. (Fredkin/Toffoli 1982, 16)

Solche Spiegel bzw. Reflektoren sind auch in der Ausgabe der Datei UNIVERSAL.PY zu sehen. Vier Signale bewegen sich in einer Endlosschleife über das Spielfeld, prallen an den Spiegeln ab und kollidieren miteinander. Die Signale bestehen notwendig aus zwei Zellen. Eine einzelne Zelle würde sich zwar wie die Signale diagonal über den Bildschirm bewegen, sie würde aber an Hindernissen wie dem Rand nicht das Gesetz Einfallswinkel gleich Ausfallswinkel erfüllen, sondern nach der Kollision auf dem selben Weg zurückwandern, auf dem sie auch gekommen ist. Zu sehen ist ein solches Verhalten in MANDALA.PY und GASWELLE.PY, wenn einzelne Zellen auf den Rand treffen.

Die Regel BBMCA ist, wie auch Game of Life oder die Regel 110 für eindimensionale Automaten, turingvollständig. Wie mit dieser Regel logische Gates und ganze Schaltungen aufgebaut werden, verdient eine eigenständige Arbeit. Die Ausgabe von UNIVERSAL.PY muss an dieser Stelle zum Verständnis genügen (vgl. Fredkin/Toffoli 1982 und Margolus 1984). Stattdessen soll Folgendes hervorgehoben werden:

Für alle 16^{16} Regelsätze, die mit dem in dieser Arbeit vorgestellten Programm implementiert werden können, gilt die in Kapitel 3.1 beschriebene Homogenität: Die Bedingung, dass keine Zelle und kein Block des Spielfelds sich gegenüber den anderen in irgendeiner Weise besonders auszeichnet. D. h. dass an jedem Ort des Spielfelds die Regeln in gleicher Weise gelten. Diese Bedingung trifft natürlich auch im Falle des BBMCA zu, was bedeutet, dass die stabilen Elemente, die Spiegel oder Reflektoren, genau den gleichen Regeln unterworfen sind wie die sich bewegenden Signale. Das Prinzip der Homogenität wurde auch hier nicht verletzt. Tatsächlich nämlich wird jeder Block – wie sich im Code des Programms nachvollziehen lässt –, und damit auch jede Zelle dem gleichen Instruktionsset zugeführt und berechnet. Es ist allein die Lage der Spiegel, die verhindert, dass diese Bereiche zerfallen und über das Spielfeld diffundieren. Wäre ein solcher Reflektor nur um eine Zelle verschoben, wäre er nicht länger komplett stabil. Die menschliche Wahrnehmung wird also grundlegend getäuscht: Der Unterschied, den wir auf den ersten und auch zweiten Blick zwischen den stabilen und beweglichen Elementen machen, ist illusorisch. Aber ist die Annahme, dass es sich bei diesen Objekten um im Kern verschiedene Elemente handelt, oder dass sie zumindest verschiedenen Kräften und Gesetzmäßigkeiten unterliegen, nur das Ergebnis menschlicher Wahrnehmungseffekte, oder sind eben auch Zeitpunkt und Lage der Objekte diese konstituierende Bedingungen? Generell lässt sich so fragen, inwieweit auch andere Eigenschaften zellulärer Automaten nicht objektiv, sondern nur Produkt menschlicher Zuschreibung sein könnten, und was sich hier im Kern vor uns vollzieht.

7. Der epistemische Status von zellulären Automaten

Die vorgestellten Beispiele haben verschiedene Interpretations- und Nutzungsräume eröffnet, von eher künstlerischen bis zu physikalischen, biologischen und technischen, ja sogar rechnenden Strukturen. Dabei wurden hier nur 7 der 16^{16} elementaren Blockautomaten mit Margolus-Nachbarschaft vorgestellt.

Über die Bilder, die von zellulären Automaten erzeugt werden, und dem Eindruck, den diese auf den Betrachter machen, wurde bereits in Kapitel 4 gesprochen. Der visuelle Effekt ist, wie gezeigt wurde, ein wesentlicher Faktor in der Wirkungsgeschichte der zellulären Automaten und einer, von dem aus diese Wirkungsgeschichte einer kritischen Reflexion unterzogen werden kann. Der visuelle Eindruck und seine Überzeugungskraft, wie er sie beispielsweise auf den erwähnten Pomeau ausübte, sind dabei die eine Seite der Medaille.

Auf der anderen, und man könnte sagen, wissenschaftlichen Seite lieg die Verwendung der zellulären Automaten als physikalisch-mathematische Modelle und Simulationen, also der Versuch, anhand der Eigenschaften und des Verhaltens dieser Automaten etwas über die Vorgänge in der Natur zu erfahren. Die zahlreichen Diskussionen, die in der zweiten Hälfte der Achtziger Jahre, der Entwicklungszeit von CAM-6 und CAM-7, darüber stattfanden, welche Automaten welchen physikalisch-mathematischen Formeln entsprächen und ob die Annahme, dass Zellularautomaten als experimentelle Systeme genutzt werden können, um physikalische Eigenschaften zu simulieren und zu studieren, überhaupt gerechtfertigt ist, geben Einblick in eine historische Phase, in der zelluläre Automaten ihren Teil zur ubiquitären Verbreitung des digitalen Computers beitrugen. Vor allem aber bieten sie Ansatzpunkte für die medientheoretische Frage, was bei den Zuschreibungen und Interpretationen der Ausgaben von zellulären Automaten nur menschliche Interpretation ist und inwieweit die Maschine diesen Interpretationen objektiv zuordenbar ist. So wurde in diesen Diskussionen der Zweifel an der Eignung von zellulären Automaten für wissenschaftliche Fragestellungen in zum Teil sehr deutlichen Worten ausgedrückt:

> In work on cellular automata, one is usually not trying to model a physical phenomenon; the rules of the game are freely chosen, the only criterion being that they produce an interesting evolution. The best example is the archetypal game of Life [sic]: its fascinating phenomena, in spite of their colorful names, bear only the faintest resemblance to actual living organisms. I do not know of a single instance where something useful for the work on lattice gases has been borrowed from the cellular automata field. (Hénon 1989, 161)

Während den zellulären Automaten für die einen aufgrund der Arbitrarität ihrer Regeln und der Tatsache, dass ihr mikroskopisches Verhalten sich so grundlegend von dem Verhalten von physikalischen oder biologischen Entitäten unterscheidet, keinerlei Berechtigung in anderen wissenschaftlichen Feldern zugesprochen werden kann, sehen andere gerade in letzterem Punkt einen wichtigen Hinweis auf die Eigenschaften realer physikalischer Systeme:

> Computer simulations on cellular automata show that they reproduce patterns observed in real physical fluids evolving according to the solutions of the hydrodynamic equations. This is remarkable, even astonishing, since at a microscopic-particle level the dynamics seems at best a caricature of the interactions between real molecules. It shows that the macroscopic behavior of a fluid does not depend on the detailed features of the particle interactions: systems which microscopically look completely different may give rise to the same type of macroscopic equations. (DeMasi et al. 1989, 93)

Diese diametral entgegen gesetzten Überzeugungen spannen zwei Extreme auf, nämlich dass entweder anhand von zellulären Automaten gar nichts über die reale Welt erkannt werden kann, oder dass nicht nur die Ergebnisse zellulärer Automaten adäquate Beobachtungen realer Vorgänge ermöglichen, sondern dass zelluläre Automaten als solche Rückschlüsse auf die Beschaffenheit der Welt zulassen – was bis zu der Ansicht führen kann, dass das Universum selbst nichts als ein einfacher zellulärer Automat sei, dessen Komplexität erst auf der makroskopischen Ebene entsteht, auf der auch wir Menschen leben. Die Idee, dass größtmögliche Komplexität im Innersten von einfachsten Regeln zusammengehalten werden könnte, hat für viele auf der Suche nach des Pudels Kern bis heute ihren Reiz behalten.

Einer weiterer wesentlicher Streitpunkt, wenn es um die Frage geht, inwieweit zelluläre Automaten sich als physikalische Modelle eignen, ist der Bruch zwischen dem Kontinuierlichen und dem Diskreten, der bei der Nutzung von zellulären Automaten zwangsläufig bei der Simulation der allermeisten physikalischen Prozesse auftritt. Gerade in der Kinetik zeigte sich von Anfang an die Dichotomie zwischen kontinuierlich und diskret: In Maxwells Arbeit von 1866 „On the Dynamical Theory of Gases“, die als das Gründungsdokument der klassischen kinetischen Theorie gilt, wird zwar die Materie diskretisiert, die Positionen der Gaspartikel und ihre Geschwindigkeit aber kontinuierlich bestimmt. Erst in den Sechziger Jahren des 20. Jahrhunderts wurden Ansätze entwickelt, auch Position und Geschwindigkeit der Partikel zu diskretisieren,

vor allem um alternative Modelle zur komplexen Boltzmanngleichung von 1872 zu erhalten (vgl. Cabannes 1989, 1f.).

Der in Kapitel 6.2.7 vorgestellte BBMCA (billiard ball model cellular automaton) ist bei diesen alternativen diskreten Modelle zu situieren, auch wenn sein Zweck letztendlich ein anderer sein sollte. Die ursprüngliche BBM ist ein klassisches mechanisches System, das nicht diskret, sondern kontinuierlich arbeitet. Während in Zellularautomaten ganze Zahlen verwendet werden, sind es im BBM von Fredkin reelle Zahlen. Wird die BBM Fredkins und Toffolis zu einem zellulären Automaten umgewandelt, dem BBMCA, dann findet hier ein Umbruch zwischen Diskretem und Kontinuierlichem statt, wie er der Streitpunkt für zahlreiche Diskussionen über die Eignung zellulärer Automaten war.

Margolus schreibt nun in seiner Arbeit über den BBMCA, dass es kein Problem sei, ein kontinuierliches System in ein digitales, d. h. auch diskretes umzuwandeln:

> In order to make it [the BBM] perform a digital computation, we make use of the fact that integers are also real numbers. By suitably restricting the initial conditions we allow the system to have, and by only looking at the system at regularly spaced time intervals, we can make a continuous dynamics perform a digital process. (Margolus 1984, 86)

Margolus, und nicht er allein, vertritt also die Auffassung, dass der Schritt vom Kontinuierlichen zum Diskreten ohne Verlust an Information oder der Veränderung wesentlicher Eigenschaften des modellierten Systems vollzogen werden kann.[10] Es liegt auf der Hand, dass jeder, der zelluläre Automaten als adäquates Mittel zur Modellierung physikalischer Phänomene favorisieren will, den Schritt vom Kontinuierlichen zum Diskreten als einen neutralen ansehen muss. Sieht man aber einmal von tiefer liegenden zeitkritischen Änderungen bei diesem Schritt ab, so liegt die wesentliche Formulierung, an der sich das eigentliche Problem zeigt, in Margolus Worten „By *suitably* restricting …“ Denn welche zellulären Automaten als *suitable* zu betrachten sind und welche nicht, ist Ende der Achtziger Jahre Gegenstand zahlreicher Debatten, wie in der bereits Erwähnten aus dem Feld der Gastheorie. Die Diskretisierung, die in diesem Bereich eigentlich keine Neuheit mehr darstellt, wird gerade im Kontext der zellulären Automaten kritisch betrachtet, zumindest solange sie, wie eben in Zellularautomaten, als einzige Methode verwendet werden soll.

[10] Es ist verblüffend, wie sehr die Beschreibung von Margolus der *time of non-reality* ähnelt, wie sie Wiener als die Grundbedingung des Digitalcomputers vorstellt (vgl. Pias 2003, 158). Die mögliche Tragweite der hier im Haupttext folgenden Zitate für den Computer als epistemisches Werkzeug soll damit nur angedeutet sein.

Im 1989 erschienen Tagungsbericht des Workshops on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics vom September 1988 in Turin, aus dem bereits zitiert wurde, heißt es:

> If we combine space and velocity discretization, the theory of existence and uniqueness for discrete models appears to be easier than the corresponding theory for the Boltzmann equation with continuous variables. Thus one might suggest that there is no need of the latter, since, after all, when we solve the equations on a computer, we always discretize the variables. This remark would, however, be short-sighted. In fact, we would not be satisfied by numerical solutions which strongly depend upon the details of the discretization, even for very fine grids; and proving existence for continuous variables is, essentially, proving that there is a solution of the equation, which underlies the discrete models that we solve. (Cercignani 1989, 71)

Während für die einen der Bruch zwischen Kontinuierlichem und Diskretem einer ist, der die Eignung rein diskreter Modelle zur Untersuchung physikalischer Prozesse zweifelhaft erscheinen lässt, drehen Toffoli und vor allem Margolus den Spieß um: Nicht die Modelle müssen sich gegenüber der Physik als adäquat erweisen. Vielmehr müssen die physikalischen Vorgänge zeigen, dass mit ihnen buchstäblich *gerechnet* werden kann. Was Margolus und Toffoli unter den Schlagworten *Programmable Matter* und *Crystalline Computation* entwerfen, ist die Idee, „[...] to build computing devices directly out of the laws of *microscopic* physics, rather than out of some phenomenologic properties of bulk matter […]." (Toffoli 1989, 1). Schon Fredkin und Toffoli schrieben: „[The billard-ball] model constitutes a substantial breakthrough in establishing a correspondence between computation and physics." (Fredkin/Toffoli 1982) So ist im BBM und vielleicht noch stärker im BBMCA schon die ganze Idee, die Physik selbst als Mittel für Berechnungen heranzuziehen, *in nuce* enthalten. Der weiße Elefant unter den Cellular Automata Machines, die wahrscheinlich nur als Papiermaschine existente CAM-7, wird vor allem vor dem Hintergrund dieses Programms verständlich: Als ein Versuch, die grundlegenden Eigenschaften physikalischer Vorgänge – Parallelität, Lokalität und Uniformität – in Hardware zu überführen, um so einen Schritt weiter hin zu Berechnungen über Materie mit der Materie selbst zu machen. Was Margolus und Toffoli dabei vorschwebt, ist nichts weniger als die Kontrolle über Zeit und Raum:

> Indeed, the most natural way of using the newly-tamed computational resources of physics will be to simulate at our leisure the phenomena of physics itself. In our simulations, a wisp of smoke will be perhaps a billion times bigger and a thousand times *slower* than the real thing, but unlike the real thing we'll be able to *stop* it and *restart* it, to *run it in reverse*, to *set breakpoints*, to measure any details of it, and to correlate it with an identical copy of itself *shifted in space or time*. In other words, we'll be doing experimental theoretical physics with total control of initial and boundary conditions. [Kursiv Th. N.] (Toffoli 1989, 1)

Die stärkste Lesart dieses Bemühens wäre die Annahme, dass die Welt selbst ein Rechner sei und im Kern so funktioniere wie die zellulären Automaten. „Erst in jüngster Zeit ziehen Chaostheorie und und informatische Komplexitätstheorie genau diese Hypothese in Zweifel. Wenn Wolken regnen oder Wellen entstehen, haust in ihren Molekülen wohl kaum ein digitaler Computer, der, indem er rechnet, das Wetter von heute und morgen machen würde." (Kittler 1996, 132)
Aber Kittler schließt das Digitale mit der von Neumann-Maschine kurz und übersieht die Möglichkeit von Rechnern, die nicht aus „some phenomenologic properties of bulk matter" gebaut sind. Insofern ist Margolus und Toffolis Projekt eine Antwort auf die folgenden Gedanken Kittlers, und zwar noch vor deren Entstehen: „Schon sind die ersten mathematischen Schritte des Nachweises getan, daß die rasante Vermehrung von Transistoren pro Chipfläche, wie sie seit zwei Jahrzehnten als Allheilmittel läuft, grundsätzlich außerstande bleibt, mit der Komplexität rückgekoppelter Naturphänomene mitzuhalten. Der digitale Ansatz selber zieht Grenzen, die für analoge Computer einer allerdings noch vollkommen hypothetischen Architektur nicht gleichermaßen gelten würden." (Kittler 1996, 132f.)

Die CAM-7 wäre die in Hardware gegossene Familie der Blockautomaten mit Margolus-Nachbarschaft gewesen, ein echt nebenläufiger Rechner. Die Maschine selbst mag in Margolus späteren Schriften verschwunden sein, die mit ihrer Konzeptualisierung verbundenen Ideen sind es dagegen nicht. Die Überzeugung, dass Parallelität, Lokalität und Homogenität als gemeinsame Eigenschaften von physikalischen Abläufen und zellulären Automaten die Letzteren zur Berechnung der Ersteren prädestinieren – eine Überzeugung die auch Stephen Wolfram teilt – hat Margolus nicht aufgegeben. Während Wolfram dabei aber auf Software-Systeme wie das von ihm entwickelte Mathematica setzte, verfolgt Margolus immer noch den Plan, alternative Rechnerarchitekturen zu kreieren, ja, die Materie selbst als Rechner zu verwenden, ganz so, wie er es in dem mit Toffoli verfassten Manifest „Programmable Matter. Concepts and Realization" (1991) dargelegt hatte. Das bereits zur Erwähnung gekommene *Crystalline Computation* ist dabei eines der neueren Schlagworte, unter denen Margolus seinen Ansatz zusammenfasst:

> Computer algorithms and computer hardware evolve together. What we mean by a good algorithm is that we have found some sort of efficient mapping between the computation and the hardware. For example, CA and other lattice algorithms are sometimes "efficiently" coded on conventional machines by mapping the parallelism and uniformity of the lattice model onto the limited parallelism and uniformity of word-wide logical operations—so-called "multi-spin coding." This is a rather grotesque physical realization for models that directly mimic the structure and locality of physics: we first build a computer that hides the underlying spatial structure of nature, and then we try to find the best way to contort our spatial computation to fit into that mold!
> Ultimately all physical computations have to fit into a spatial mold, and so our most efficient computers and algorithms will eventually have to evolve toward the underlying spatial "hardware" of nature (Margolus 1999, 24f.).

Es wäre aber zu einfach, auf die Positivitäten der Rechnerentwicklung zu zeigen und die Ideen und Konzepte von Margolus und Toffoli als anachronistisch unter den Teppich kehren zu wollen. Nicht nur zeigen diese Ideen Alternativen und Möglichkeiten auf, die uns Zeugnis für die Kontingenzen in der technologischen Entwicklung sein können. Sie liefern auch das Material für eine in die Zukunft gerichtete Archäologie, für Unternehmungen, die nicht in der Vergangenheit graben nur um der Vergangenheit willen, sondern ein darin vorhandenes Potential zu aktivieren und zu vergegenwärtigen suchen. So sind auch Margolus und Toffolis Konzepte nicht einfach ins Leere gelaufen. Ganz im Gegenteil, sie fanden und finden ihren Wiederaufruf in der Projektierung des streng parallel arbeitenden Quantenrechners. Das ist kein Zufall, schließlich haben Margolus und Toffoli einigen Anteil an der Entwicklung dieser Rechnerarchitektur gehabt (vgl. Margolus 1986/1989 und auch das Toffoli-Gate).

An kleinen diskursiven Realitäten (ein Begriff von Paul Veyne 2009, 92) wie sie die CAM-7 geblieben ist, lässt sich mehr ablesen, als nur die Geschichte einer abgebrochenen Entwicklung, aber auch mehr als eine kontinuierliche Fortschrittsgeschichte von den ersten Versuchen mit Parallelcomputern bis zum Quantenrechner. Vergegenwärtigen heißt, die Brüche und Diskontinuitäten zu beachten, die ein solches Objekt auszeichnen. Mit dem hier mitgelieferten Programm und den implementierten elementaren Blockautomaten wird denn auch die Geschichte der CAM-7 auf eine andere und diskontinuierlichere Weise fortgesponnen, als es ein Nachbau oder eine Emulation vermöchten, deren Vektor doch immer eher auf die Vergangenheit gerichtet wäre und nicht, wie es die Hoffnung dieser Arbeit ist, auf die Aufgaben der Zukunft.

5. BASIC Listings

Die beiden folgenden Listings enthalten das schon bekannte Programm SAND und das noch nicht beschriebene Programm TRON. Die Listings können, komplett so wie sie sind, auf folgender Seite per copy and paste eingefügt und in Vollzug gesetzt werden:

http://www.quitebasic.com

Es empfiehlt sich, dabei den Canvas auf 25x25 Zellen einzustellen.

5.1 BASIC Listing: SAND

-- ab hier kopieren--

```
0  CLS
10  ARRAY A
20  DIM B(24,24)
30  FOR i = 0 TO 24
40    FOR j = 0 TO 24
50       LET B(i,j) = 0
60    NEXT j
70  NEXT i
80  LET Z = -1

90  REM Zellen können besetzt werden auf dem Arbeitsfeld C (25,25)
100 REM FOR i = 0 TO 9
110 REM LET B(i+13,5) = 1
120 REM NEXT i
130 REM LET B(2,3) = 1
140 REM LET B(3,3) = 1
150 REM LET B(9,9) = 1
160 REM LET B(10,10) = 1
170 REM LET B(14,13) = 1

200 REM_____PROGRAMMKOPF

240 REM Zu Kopierroutine Spielfeld auf Canvas
250 GOSUB 6400

260 REM Zu SUB INJECTION
270 REM lässt neue Zellen erscheinen
280 GOSUB 6000

300 REM Zähler Z für den Margolus Wechsel, wird mit -1 multipliziert um zu alternieren. So wird bei jedem zweiten Durchlauf das selbe Unterprogramm angesprungen.
310 LET Z = Z * -1
320 PRINT Z

350 IF Z = 1 THEN GOTO 600
360 REM Sets 1 werden analysiert
370 IF Z = -1 THEN GOTO 700
380 REM Sets 2 werden analysiert
390 REM PAUSE 10
400 GOTO 210
410 REM 210 Geht zurück zu PROGRAMM

580 REM_____MARGOLUS
590 REM hier werden Parameter START und ENDE für i und j als x (START) und y (ENDE) vordefiniert für Sets 1
600 LET x = 0
610 LET y = 22
620 GOTO 1000
```

```
630 REM 1000 Geht zu REGELN
640 GOTO 450

690 REM hier werden Parameter START und ENDE für i und j als x (START) und y (ENDE) vordefiniert für Sets 2
700 LET x = 1
710 LET y = 23
720 GOTO 1000
730 REM 1000 Geht zu REGELN
740 GOTO 450

990 REM_____ANALYSE
1000 FOR i = x TO y STEP 2
1010    FOR j = x TO y STEP 2

2000 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5000
2010 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5040
2020 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5080
2030 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5120
2040 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5040
2050 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5120
2060 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5120
2070 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5130
2080 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5080
2090 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5120
2100 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5120
2110 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5140
2120 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5120
2130 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5130
2140 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5140
2150 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5150

3100    NEXT j
3110 NEXT i
3120 GOTO 200

4990 REM_____TRANSFORMATION

5000 LET B(i,j)    = 0
5001 LET B(i+1,j)  = 0
5002 LET B(i,j+1)  = 0
5003 LET B(i+1,j+1) = 0
5004 GOTO 3100

5010 LET B(i,j)    = 0
5011 LET B(i+1,j)  = 0
5012 LET B(i,j+1)  = 1
5013 LET B(i+1,j+1) = 0
5014 GOTO 3100

5020 LET B(i,j)    = 0
5021 LET B(i+1,j)  = 0
5022 LET B(i,j+1)  = 0
5023 LET B(i+1,j+1) = 1
5024 GOTO 3100
```

```
5030 LET B(i,j)   = 0
5031 LET B(i+1,j)  = 0
5032 LET B(i,j+1)  = 1
5033 LET B(i+1,j+1) = 1
5034 GOTO 3100

5040 LET B(i,j)   = 1
5041 LET B(i+1,j)  = 0
5042 LET B(i,j+1)  = 0
5043 LET B(i+1,j+1) = 0
5044 GOTO 3100

5050 LET B(i,j)   = 1
5051 LET B(i+1,j)  = 0
5052 LET B(i,j+1)  = 1
5053 LET B(i+1,j+1) = 0
5054 GOTO 3100

5060 LET B(i,j)   = 1
5061 LET B(i+1,j)  = 0
5062 LET B(i,j+1)  = 0
5063 LET B(i+1,j+1) = 1
5064 GOTO 3100

5070 LET B(i,j)   = 1
5071 LET B(i+1,j)  = 0
5072 LET B(i,j+1)  = 1
5073 LET B(i+1,j+1) = 1
5074 GOTO 3100

5080 LET B(i,j)   = 0
5081 LET B(i+1,j)  = 1
5082 LET B(i,j+1)  = 0
5083 LET B(i+1,j+1) = 0
5084 GOTO 3100

5090 LET B(i,j)   = 0
5091 LET B(i+1,j)  = 1
5092 LET B(i,j+1)  = 1
5093 LET B(i+1,j+1) = 0
5094 GOTO 3100

5100 LET B(i,j)   = 0
5101 LET B(i+1,j)  = 1
5102 LET B(i,j+1)  = 0
5103 LET B(i+1,j+1) = 1
5104 GOTO 3100

5110 LET B(i,j)   = 0
5111 LET B(i+1,j)  = 1
5112 LET B(i,j+1)  = 1
5113 LET B(i+1,j+1) = 1
5114 GOTO 3100
```

```
5120 LET B(i,j)    = 1
5121 LET B(i+1,j)  = 1
5122 LET B(i,j+1)  = 0
5123 LET B(i+1,j+1) = 0
5124 GOTO 3100

5130 LET B(i,j)    = 1
5131 LET B(i+1,j)  = 1
5132 LET B(i,j+1)  = 1
5133 LET B(i+1,j+1) = 0
5134 GOTO 3100

5140 LET B(i,j)    = 1
5141 LET B(i+1,j)  = 1
5142 LET B(i,j+1)  = 0
5143 LET B(i+1,j+1) = 1
5144 GOTO 3100

5150 LET B(i,j)    = 1
5151 LET B(i+1,j)  = 1
5152 LET B(i,j+1)  = 1
5153 LET B(i+1,j+1) = 1
5154 GOTO 3100

9990 END

5990 REM_____________________________SUBROUTINEN

6000 REM_____INJECTION
6010 LET B(11,24) = 1
6020 LET B(12,24) = 1
6021 LET B(5,24)  = 1

6050 RETURN

6400 REM Kopierroutine von Spielfeld B auf Canvas
6410   FOR i = 0 TO 24
6420     FOR j = 0 TO 24
6450     IF B(i,j) = 1 THEN PLOT i,j, "yellow" ELSE PLOT i,j, "brown"
6470     NEXT j
6480   NEXT i
6480 RETURN
```

5.2 BASIC Listing: TRON

-- ab hier kopieren--

```
0  CLS
10  ARRAY A
20  DIM B(24,24)
30  FOR i = 0 TO 24
40    FOR j = 0 TO 24
50       LET B(i,j) = 0
60    NEXT j
70  NEXT i
80  LET Z = -1

90  REM Zellen können besetzt werden auf dem Arbeitsfeld C (25,25)
100 FOR i = 0 TO 9
110 LET B(i+13,5) = 1
120 NEXT i
130 LET B(2,3) = 1
140 LET B(3,3) = 1
150 LET B(9,9) = 1
160 LET B(10,10) = 1
170 LET B(14,13) = 1

200 REM_____PROGRAMMKOPF

240 REM Zu Kopierroutine Spielfeld auf Canvas
250 GOSUB 3400

300 REM Zähler Z für den Margolus Wechsel, wird mit -1 multipliziert um zu alternieren. So
wird bei jedem zweiten Durchlauf das selbe Unterprogramm angesprungen.
310 LET Z = Z * -1
320 PRINT Z

350 IF Z = 1 THEN GOTO 600
360 REM Sets 1 werden analysiert
370 IF Z = -1 THEN GOTO 700
380 REM Sets 2 werden analysiert
390 REM PAUSE 10
400 GOTO 210
410 REM 210 Geht zurück zu PROGRAMM

580 REM_____MARGOLUS
590 REM hier werden Parameter START und ENDE für i und j als x (START) und y (ENDE)
vordefiniert für Sets 1
600 LET x = 0
610 LET y = 22
620 GOTO 1000
630 REM 1000 Geht zu REGELN
640 GOTO 450
```

```
690 REM hier werden Parameter START und ENDE für i und j als x (START) und y (ENDE) vordefiniert für Sets 2
700 LET x = 1
710 LET y = 23
720 GOTO 1000
730 REM 1000 Geht zu REGELN
740 GOTO 450

990 REM_____ANALYSE
1000 FOR i = x TO y STEP 2
1010    FOR j = x TO y STEP 2

2000 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5150
2010 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5010
2020 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5020
2030 IF B(i,j) = 0 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5030
2040 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5040
2050 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5050
2060 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5060
2070 IF B(i,j) = 1 AND B(i+1,j) = 0 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5070
2080 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5080
2090 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5090
2100 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5100
2110 IF B(i,j) = 0 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5110
2120 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 0 THEN 5120
2130 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 0 THEN 5130
2140 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 0 AND B(i+1,j+1) = 1 THEN 5140
2150 IF B(i,j) = 1 AND B(i+1,j) = 1 AND B(i,j+1) = 1 AND B(i+1,j+1) = 1 THEN 5000

3100    NEXT j
3110 NEXT i
3120 GOTO 200

4990 REM_____TRANSFORMATION

5000 LET B(i,j)    = 0
5001 LET B(i+1,j)  = 0
5002 LET B(i,j+1)  = 0
5003 LET B(i+1,j+1) = 0
5004 GOTO 3100

5010 LET B(i,j)    = 0
5011 LET B(i+1,j)  = 0
5012 LET B(i,j+1)  = 1
5013 LET B(i+1,j+1) = 0
5014 GOTO 3100

5020 LET B(i,j)    = 0
5021 LET B(i+1,j)  = 0
5022 LET B(i,j+1)  = 0
5023 LET B(i+1,j+1) = 1
5024 GOTO 3100

5030 LET B(i,j)    = 0
5031 LET B(i+1,j)  = 0
```

```
5032 LET B(i,j+1)   = 1
5033 LET B(i+1,j+1) = 1
5034 GOTO 3100

5040 LET B(i,j)     = 1
5041 LET B(i+1,j)   = 0
5042 LET B(i,j+1)   = 0
5043 LET B(i+1,j+1) = 0
5044 GOTO 3100

5050 LET B(i,j)     = 1
5051 LET B(i+1,j)   = 0
5052 LET B(i,j+1)   = 1
5053 LET B(i+1,j+1) = 0
5054 GOTO 3100

5060 LET B(i,j)     = 1
5061 LET B(i+1,j)   = 0
5062 LET B(i,j+1)   = 0
5063 LET B(i+1,j+1) = 1
5064 GOTO 3100

5070 LET B(i,j)     = 1
5071 LET B(i+1,j)   = 0
5072 LET B(i,j+1)   = 1
5073 LET B(i+1,j+1) = 1
5074 GOTO 3100

5080 LET B(i,j)     = 0
5081 LET B(i+1,j)   = 1
5082 LET B(i,j+1)   = 0
5083 LET B(i+1,j+1) = 0
5084 GOTO 3100

5090 LET B(i,j)     = 0
5091 LET B(i+1,j)   = 1
5092 LET B(i,j+1)   = 1
5093 LET B(i+1,j+1) = 0
5094 GOTO 3100

5100 LET B(i,j)     = 0
5101 LET B(i+1,j)   = 1
5102 LET B(i,j+1)   = 0
5103 LET B(i+1,j+1) = 1
5104 GOTO 3100

5110 LET B(i,j)     = 0
5111 LET B(i+1,j)   = 1
5112 LET B(i,j+1)   = 1
5113 LET B(i+1,j+1) = 1
5114 GOTO 3100

5120 LET B(i,j)     = 1
5121 LET B(i+1,j)   = 1
5122 LET B(i,j+1)   = 0
```

```
5123 LET B(i+1,j+1) = 0
5124 GOTO 3100

5130 LET B(i,j)     = 1
5131 LET B(i+1,j)   = 1
5132 LET B(i,j+1)   = 1
5133 LET B(i+1,j+1) = 0
5134 GOTO 3100

5140 LET B(i,j)     = 1
5141 LET B(i+1,j)   = 1
5142 LET B(i,j+1)   = 0
5143 LET B(i+1,j+1) = 1
5144 GOTO 3100

5150 LET B(i,j)     = 1
5151 LET B(i+1,j)   = 1
5152 LET B(i,j+1)   = 1
5153 LET B(i+1,j+1) = 1
5154 GOTO 3100

9990 END

2990 REM_____SUBROUTINEN

3400 REM Kopierroutine von Spielfeld B auf Canvas
3410    FOR i = 0 TO 24
3420      FOR j = 0 TO 24
3430        REM LET x = i+1
3440        REM LET y = j+1
3450      IF B(i,j) = 1 THEN PLOT i,j ELSE PLOT i,j, "yellow"
3470      NEXT j
3480    NEXT i
3480 RETURN
```

9. Literaturverzeichnis

Adamatzky, Andrew (Hg.) (2010): Game of Life Cellular Automata. London: Springer.

Braguinski, Nikita (2015): Die Spiraldarstellung – Ein experimentelles Visualisierungsverfahren. Download am 7.3.15.
https://www.academia.edu/7380819/Die_Spiraldarstellung_-_ein_experimentelles_Visualisierungsverfahren

Cabannes, Henri: Workshop on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics. Opening Lecture. In: : Monaco R. (Hg.): Proceedings of the Workshop on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics. September 20–24, 1988. Singapore/New Jersey/London/Hong Kong: World Scientific, 1–5.

Cercignani, Carlo (1989): Discrete Models in Kinetic Theory. In: Monaco R. (Hg.): Proceedings of the Workshop on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics. September 20–24, 1988. Singapore/New Jersey/London/Hong Kong: World Scientific, 62–73.

DeMasi, A./Esposito, R./Lebowitz, L./Presutti, E. (1989): Rigorous results on some stochastic cellular automata. In: Monaco R. (Hg.): Proceedings of the Workshop on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics. September 20–24, 1988. Singapore/New Jersey/London/Hong Kong: World Scientific, 93–101.

Fredkin, Edward/Toffoli, Tommaso (1982): Conservative Logic. In: In. J. of Theo. Phys. 21 (1982), 219–253, als Download 1–26. Download am 7.11.15.
http://www.cs.princeton.edu/courses/archive/fall06/cos576/papers/fredkin_toffoli82.pdf

Hénon, M. (1989): On the Relation between Lattice Gases and Cellular Automata. In: Monaco R. (Hg.): Proceedings of the Workshop on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics. September 20–24, 1988. Singapore/New Jersey/London/Hong Kong: World Scientific, 160–161.

Kari, Jarkko (1990): Reversibility of 2D Cellular Automata is Undecidable. In: Physica D 45 (1990), 379–385.

Kittler, Friedrich (1996): Computeranalphabetismus. In: Ders. (2002): Short Cuts. Frankfurt a. M.: Zweitausendeins, 109–133.

Margolus, Norman (1984): Physic-like Models of Computation. In: Physica 10D (1984), 81–95.

Margolus, Norman (1986): Quantum Computation. Downlaod am 24.10.15.
http://people.csail.mit.edu/nhm/qcomp.pdf

Margolus, Norman (1988): Physics and Computation. Ph.D. Thesis. Tech. Rep. MIT/LCS/TR-415. MIT Laboratory for Computer Science.

Margolus, Norman (1989): Parallel Quantum Computation. Download am 24.10.15.
http://people.csail.mit.edu/nhm/pqc.pdf

Margolus, Norman (1992): A Bridge of Bits. Download am 24.10.15.
http://people.csail.mit.edu/nhm/bridge.pdf

Margolus, Norman (1999): Crystalline Computation. Download am 24.10.15.
http://people.csail.mit.edu/nhm/cc.pdf

Margolus, Norman (2002): Spatial Computers: Architectures and algorithms for large-scale spatial computations (Ein Vortrag, gehalten anlässlich der Computing Beyond Silicon Summer School, CalTech, 26. Juni 2002). Download 24.10.15.
http://people.csail.mit.edu/nhm/spatial_computers.pdf

Margolus, Norman/Toffoli, Tommaso (1987a): Cellular Automata Machines. In: Complex Systems 1 (1987), 967–993.

Margolus, Norman/Toffoli, Tommaso (1987b): Cellular Automata Machines. A new environment for modelling. Cambridge, London: MIT Press.

Margolus, Norman/Toffoli, Tommaso (1990a): Invertible Cellular Automata: A Review. In: Physica D 45 (1990), 229–253.

Margolus, Norman/Toffoli, Tommaso (1990b): Programmable Matter. In: Physica D47 (1991), 263–272.

Margolus, Norman/Toffoli, Tommaso (1992): U.S. Patent Nr. 5,159,690. Multidimensional Cellular Data Array Processing System which Separately Permutes Stored Data Elements and Applies Transformation Rules to Permuted Elements. Download am 22.10.15.
http://www.patents.com/pdf-5159690.html

McLuhan, Marshall (1964): Understanding Media. The Extensions of Man. London/New York: Ark Paperbacks.

Martínez, Genaro Juárez (2004): Introduction to Rule 110. Download am 13.10.15.
http://www.rule110.org/amhso/results/rule110-intro/introRule110.html

Maxwell, J. Clerk: On the Dynamical Theory of Gases. In: Philosophical Transactions of the Royal Society of London, Vol. 157 (1867), 49–88. Download am 21.11.15.
http://www.jstor.org/stable/108968?seq=1#page_scan_tab_contents

Pias, Claus (Hg.) (2003): Cybernetics - Kybernetik. The Macy-Conferences 1946-1953. Band 1: Transactions - Protokolle. Zürich/Berlin: Diaphanes.

Pomeau, Yves et al. (1987): Lattice Gas Hydrodynamics in Two and Three Dimensions. In: Complex Systems 1 (1987), 649–707.

Rucker, Rudy (1988): Welcome to Silicon Valley. Download 23.10.15.
http://www.rudyrucker.com/transrealbooks/collectedessays/#_Toc19

Rucker, Rudy (1989): Weird Screens. Download 23.10.15.
http://www.rudyrucker.com/transrealbooks/collectedessays/#_Toc19

Toffoli, Tommaso (1989): Frontiers in Computing. In: Ritter, G. X. (Hg): Information Processing 89. Proceedings of the IFIP 11th World Computer Congress San Francisco, U.S.A. August 28-September 1, 1989. Amsterdam et al.: North-Holland, 1.

Veyne, Paul (2009): Foucault. Der Philosoph als Samurai. Stuttgart: Reclam.

Volmar, Axel (2015): Klang-Experimente. Die auditive Kultur der Naturwissenschaften 1761–1961. Frankfurt a. M.: Campus.

Wolfram, Stephen (1984): Universality and Complexity in Cellular Automata. In: Physica 10D (1984), 1–35.

Internetseiten

Hopkins, Donald Edward: CAM6. Download am 29.11.15.
http://www.donhopkins.com/home/CAM6/

Maibaum, Johannes (2016):

Margolus, Norman (a): An Early Sampler of CAM-8 Applications. Downlad am 24.10.15.
http://www.ai.mit.edu/projects/im/broch/

Margolus, Norman (b): CAM8: Latest Developments. Downlad am 24.10.15.
http://www.ai.mit.edu/projects/im/cam8/latest.html

Margolus, Norman (c): CAM8: a Parallel, Uniform, Scalable Architecture for Cellular Automata Experimentation. Download am 24.10.15.
http://www.ai.mit.edu/projects/im/cam8/

Margolus, Norman (d): Lattice-Gas Models of Crystalization and Elastic Solids. Download am 24.10.15.
http://www.ai.mit.edu/projects/im/arpa/cam8bb.html

Margolus, Norman (e): Norm Margolus. Download 20.11.15.
http://people.csail.mit.edu/nhm/

Python Software Foundation (2015): 9.6 Generate pseudo-random numbers. Download am 7.3.15.
https://docs.python.org/3.0/library/random.html

Rokicki, Tom/ Trevorrow, Andrew (2013): Golly. Download 10.1.16.
http://golly.sourceforge.net

Tyler, Tim: The Margolus neighbourhood. Download am 5.9.15.
http://cell-auto.com/neighbourhood/margolus/
Tyler, Tim: The Q*Bert neighbourhood. Download am 7.3.15.
http://cell-auto.com/neighbourhood/qbert/index.html

Tyler, Tim: The Star of David neighbourhood. Download am 7.3.15.
http://cell-auto.com/neighbourhood/sod/

Wójtowicz, Mirek (2001): Cellular Automata rules lexicon. Family: Margolus. Download am 4.11.15.
http://psoup.math.wisc.edu/mcell/rullex_marg.html

Humboldt-Universität zu Berlin
Philosophische Fakultät III

Name: Nückel Vorname: Thomas

Matrikelnummer: 553745

Eidesstattliche Erklärung zur

☒ **Hausarbeit** ☐ **Magisterarbeit**

☐ **Bachelorarbeit**

☐ **Masterarbeit**

Ich erkläre ausdrücklich, dass es sich bei der von mir eingereichten schriftlichen Arbeit mit dem Titel

Zelluläre Blockautomaten mit Margolus-Nachbarschaft. Eine medienarchäologische Einladung

um eine von mir erstmalig, selbstständig und ohne fremde Hilfe verfasste Arbeit handelt.

Ich erkläre ausdrücklich, dass ich *sämtliche* in der oben genannten Arbeit verwendeten fremden Quellen, auch aus dem Internet (einschließlich Tabellen, Grafiken u. Ä.) als solche kenntlich gemacht habe. Insbesondere bestätige ich, dass ich ausnahmslos sowohl bei wörtlich übernommenen Aussagen bzw. unverändert übernommenen Tabellen, Grafiken u. Ä. (Zitaten) als auch bei in eigenen Worten wiedergegebenen Aussagen bzw. von mir abgewandelten Tabellen, Grafiken u. Ä. anderer Autorinnen und Autoren (Paraphrasen) die Quelle angegeben habe.

Mir ist bewusst, dass Verstöße gegen die Grundsätze der Selbstständigkeit als Täuschung betrachtet und entsprechend der fachspezifischen Prüfungsordnung und/oder der Allgemeinen Satzung für Studien- und Prüfungsangelegenheiten der HU (ASSP) bzw. der Fächerübergreifenden Satzung zur Regelung von Zulassung, Studium und Prüfung der Humboldt-Universität (ZSP-HU) geahndet werden.

Datum 19.3.16 Unterschrift