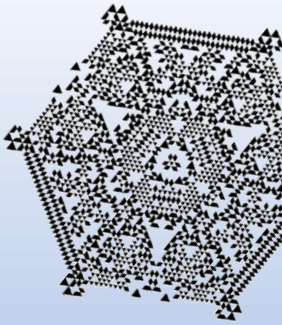
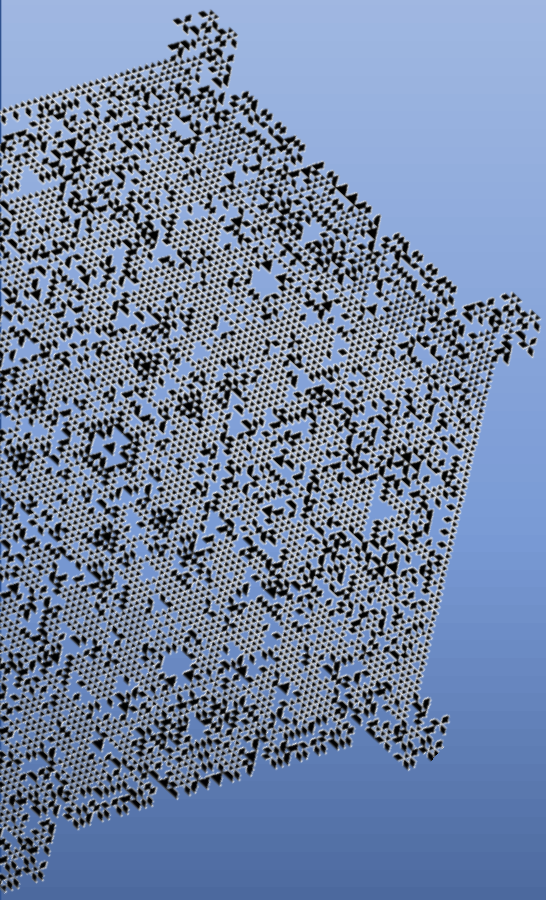


Zelluläre Automaten

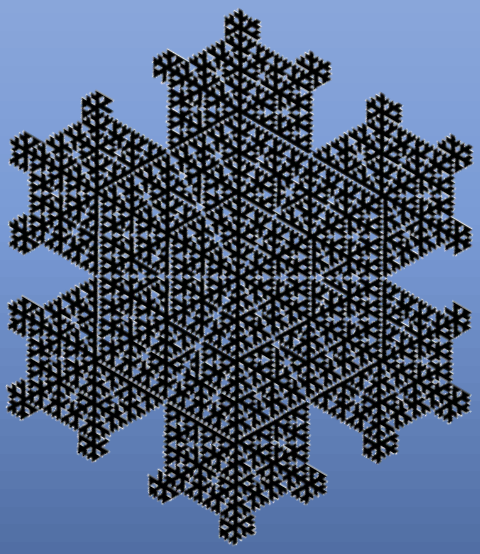
Wenn einfach komplex ist



Maturitätsarbeit 2018

Mathematik

Kantonsschule Zürcher Oberland Wetzikon



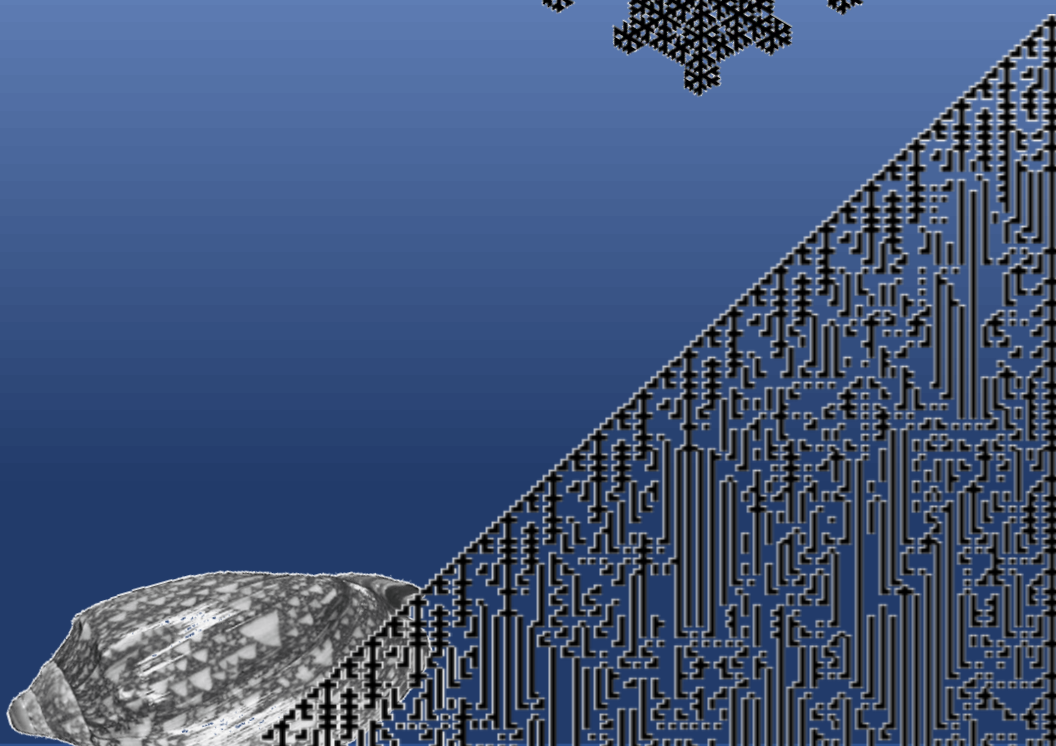
Pio Blieske

C6a
Bogenackerstrasse 5
8632 Tann
079 535 65 33

Betreut von Beat Trachsler

Mathematik

Herbstsemester 2017



Abstract

Zelluläre Automaten sind abstrakte mathematische Modelle, welche oft bei Simulationen in verschiedensten Gebieten der Naturwissenschaft zum Zuge kommen. In dieser Maturaarbeit soll es um die Untersuchung der Eigenschaften von zellulären Automaten gehen, wodurch die mathematischen Aspekte in den Vordergrund treten. Es werden Suchstrategien zum Finden von komplexen Automaten sowie bestimmten Objekten entwickelt. Ausserdem wird das Wachstum in speziellen Fällen sowie auch im Allgemeinen und der Zusammenhang zwischen den Regeln und dem «Verhalten» von zellulären Automaten untersucht. Zu den Resultaten gehören einige selbst gefundene Objekte sowie selbstgelieferte Erklärungen für bestimmte Auffälligkeiten. Zudem wurden alle Programme, welche für die Untersuchungen gebraucht werden, selbst entwickelt.

Inhaltsverzeichnis

1. Einleitung

Zelluläre Automaten - Was ist das? - Wie kommt man auf sowas?

Durch mein Interesse an mathematischen und naturwissenschaftlichen Fragen, stiess ich im Frühling 2016 auf den YouTube-Kanal «Numberphile». «Numberphile» gehört dem Videojournalisten und unabhängigen Filmemacher Brady Haran, der allgemeinverständliche Kurzvideos zu mathematischen Themen produziert. Dafür trifft er jeweils Mathematiker und auch Naturwissenschaftler, welche sich im entsprechenden Gebiet auskennen und oftmals Forschung dazu betreiben. [1] Das Video «Does John Conway hate his Game of Life», in welchem es um den zellulären Automaten namens "Game of Life" geht, faszinierte mich besonders. [2] Zelluläre Automaten sind abstrakte mathematische Modelle, welche oft mithilfe von Computern visuell dargestellt werden. Betrachtet man einen zellulären Automaten bei seinen «Handlungen», so kann der Eindruck entstehen, man sei in einem unbekannten Universum gelandet und beobachte nun dessen Bewohner und die Natur, denen mysteriöse Naturgesetze zugrunde liegen. Es sind Kolonien zu finden, welche gemeinsam Raumschiffe bauen und diese dann auf eine Reise quer durch das Universum schicken. Das verblüffende ist, dass solchen zellulären Automaten extrem einfache Regeln zugrunde liegen, welche sich in zwei, drei Zeilen formulieren lassen. Sie beinhalten nicht direkt ein Konzept für die Bewegung und Reproduktion von Figuren, ermöglichen dies aber in gewissen Fällen. Zelluläre Automaten zeigen auf, wie mit einfachen Regeln hoch komplexe Systeme wie das menschliche Hirn entstehen können.[1] Sie sind jedoch nicht nur interessant zu beobachten, sondern werden oft für Simulationen in den Naturwissenschaften gebraucht, werfen philosophische Fragestellungen auf und können ausserdem in gestalterischen und musikalischen Bereichen eingesetzt werden. In der Natur kann man ausserdem Muster von zellulären Automaten beispielsweise auf Schalen von Muscheln finden.

In dieser Arbeit wird zuerst erklärt, was zelluläre Automaten genau sind, sowie einige grundlegende Aspekte erläutert. Anschliessend werden einige zelluläre Automaten aus der Sicht der Mathematik betrachtet und untersucht. Theoretisch ist es möglich, sich unendlich viele verschiedene zelluläre Automaten auszudenken. Jedoch weisen einige davon ein äussert monotones und langweiliges «Verhalten» auf, man könnte sie sich als Planet ohne jede Spur von Leben, mit endloser Wüste vorstellen. Daher wird es um die Fragen gehen, wie man interessante Automaten, also Planeten mit Lebewesen wie die Erde, finden kann, ohne viele Regeln auszuprobieren. Wird ein solcher «Planet» gefunden, dann sollte es anschliessend das Ziel sein, möglichst interessante «Lebewesen» zu finden. Dafür sollen entsprechende Suchstrategien entwickelt werden. Weiters wird ausserdem das Wachstum bei bestimmten zellulären Automaten sowie auch im Allgemeinen untersucht. Dieses ähnelt oft der Entstehung von Eiskristallen, dessen Geschichte mit einem einfachen Kern beginnt und dann mit der Zeit symmetrisch nach aussen wächst. Um das Vorhaben etwas interessanter zu gestalten, werden zelluläre Automaten mit etwas ungewöhnlichen Regeln und Voraussetzungen gewählt, über welche noch nicht viel bekannt ist und welche noch nicht ausgiebig untersucht worden sind. Dabei wird uns ausserdem die Suche nach einem Zusammenhang zwischen den Regeln der Automaten (= den Naturgesetzen) und dem Verhalten (= Aussehen des «Planeten») begleiten.

[1] Eindrückliches Video zu einem «unbekannten Universum»: «The Meaning of Life» (https://www.youtube.com/watch?v=dJlNIWylQQ4)

Die Zielsetzung lässt sich somit in drei Teilziele aufteilen:

1. Zelluläre Automaten mit Zellularfeldern untersuchen, deren Zellen die Form von gleichseitigen Dreiecken und regelmässigen Sechsecken haben.
2. Suchstrategien für das Finden von interessanten Objekten zu entwickeln.
3. Einen Zusammenhang zwischen den Regeln und dem Verhalten von zellulären Automaten finden.

Aufgrund der ungewöhnlich gewählten Regeln, zu welchen sich nur mit sehr viel Aufwand Programme finden lassen würden, habe ich die entsprechende Software für die Untersuchung von zellulären Automaten selbst am Anfang der Arbeit programmiert, bevor dann anschliessend die Fragestellung untersucht wurde. Dadurch erhält die Arbeit zusätzlich einen kleineren praktischen Teil. Zudem liegt hinten eine CD-ROM bei, auf der die Programme, der Code dazu sowie auch die Arbeit in Form eines PDFs zu finden ist.

2. Grundlagen

2.1 Zelluläre Automaten

Was sind zelluläre Automaten?

Einen zellulären Automaten kann man sich als ein unendlich grosses Gitter, dass wie ein Schachbrett aussieht, vorstellen. Dessen Felder, genannt Zellen, können dabei zwei Zustände annehmen. Entweder sind sie schwarz, was als lebendig bezeichnet wird, oder weiss, was für tot steht. Oftmals braucht man auch die Zahlen 0 und 1, um anzugeben, in welchem Zustand eine Zelle sich befindet. Eine Zelle, im Bild links schwarz dargestellt, hat jeweils 8 Nachbarn, welche grau gekennzeichnet sind. Die Zellen des Automaten können im Verlauf der Zeit ihren Zustand ändern, abhängig von ihrem Zustand und dem der Nachbarn zuvor. Eine tote Zelle wird im nächsten Zeitschritt zum Leben erweckt, wenn sie genau drei Nachbarn hat. Hat eine lebende Zelle zwei oder drei Nachbarn, dann überlebt sie, ist im nächsten Zeitschritt also auch lebendig. Bei mehr oder weniger Nachbarn stirbt die Zelle an «Überbevölkerung» oder «Einsamkeit». Bei diesen einfachen Regeln vermutet man zunächst auch ein dementsprechend einfaches und vorhersagbares Verhalten. Dass dies jedoch nicht der Fall ist, wird im folgenden Abschnitt «Objekte» aufgezeigt. Das oben gemachte Beispiel trägt den Namen «Game of Life» und wurde von John Conway entworfen. Es gibt unzählige weitere zelluläre Automaten, theoretisch unendlich viele, die jedoch alle nach dem gleichen Prinzip aufgebaut sind. Jeder zelluläre Automat besteht aus einem Zellularfeld, dessen Zellen die Form eines Quadrats, gleichseitigen Dreiecks, regelmässigen Sechsecks oder auch anderer Polygone haben können. Dabei hat jede Zelle eine bestimmte Nachbarschaft. Die oben vorgestellte Art von Nachbarschaft, bei welcher die Zellen mit einer gemeinsamen Ecke als Nachbarn gezählt werden, wird «Moore-Nachbarschaft» genannt. Eine weitere Art von Nachbarschaft wäre die «Von-Neumann-Nachbarschaft», bei welcher eine Zelle nur die vier anliegenden Zellen, mit denen sie eine gemeinsame Kante hat, als Nachbarn betrachtet werden. Weiter ist es auch möglich, mehr als nur zwei Zustände für eine Zelle zuzulassen, sowie die Regeln, oft Überführungsfunktion genannt, beliebig zu wählen.

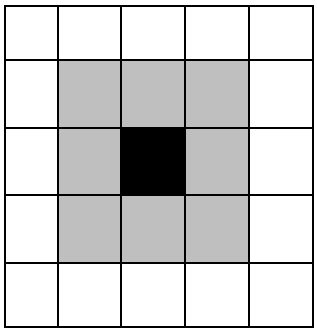


Abb. 1: Nachbarschaft

Objekte

In diesem Abschnitt werden die verschiedenen Arten von Objekten vorgestellt, welche in zellulären Automaten zu finden sind. Einige der Objekte werden im späteren Verlauf wichtig für die Bestimmung der Eigenschaften von Automaten sein. Die hier erwähnten Beispiele stammen alle aus «Game of Life», es lassen sich jedoch auch in anderen Automaten Objekte mit gleichen Eigenschaften finden.

Bei sogenannten «Still Lifes» handelt es sich um Objekte, die stabil sind. Das heisst, dass sie im nächsten Zeitschritt erhalten bleiben, wenn das Zellularfeld sonst leer ist, also die restlichen Zellen, die nicht zum Objekt gehören, weiss sind. Alle lebenden Zellen besitzen demzufolge bei den Beispielen von «Life» zwei oder drei Nachbarn und keine der umliegenden toten Zelle hat genau drei

Nachbarn. Viele der «Still Lifes» können aus anderen Konfigurationen hervorgehen, jedoch existieren auch sogenannte «Paradies», welche nicht aus anderen Mustern entstehen können.

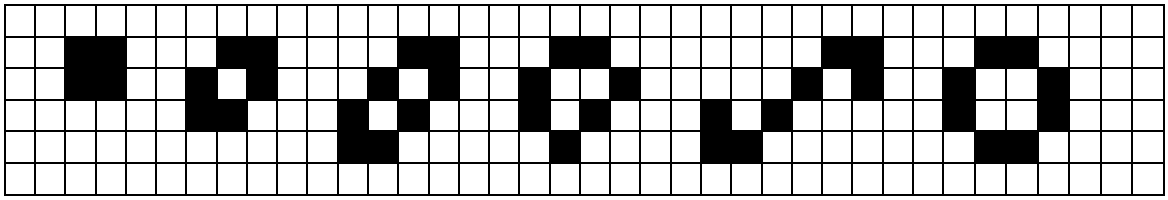


Abb. 2: Einige Beispiele von «Still Lifes» in «Life» [34]

Oszillatoren sind Objekte, welche am gleichen Ort bleiben und dabei ihr Muster verändern. Nach einer bestimmten Periodenlänge kehren sie dabei wieder zu ihrem Anfangszustand zurück.

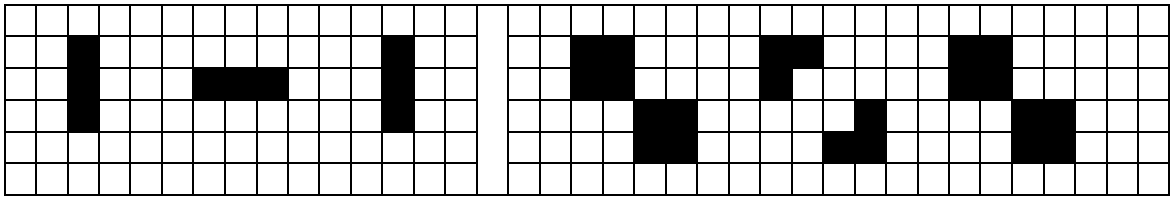


Abb. 3: Ein Blinker links und ein «Beacon» rechts [35], beide mit Periodenlänge 2

Bei Raumschiffen handelt es sich um Objekte, welche sich in eine bestimmte Richtung bewegen. Sie kehren wie Oszillatoren nach einer bestimmten Zeit wieder zum Anfangszustand zurück, sind dann jedoch um eine bestimmte Anzahl Zellen verschoben.

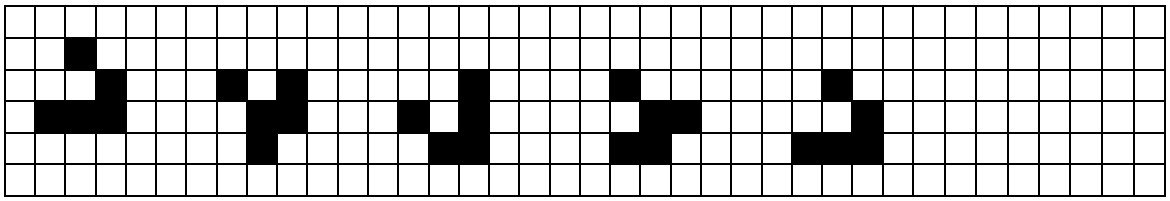


Abb. 4: Ein Gleiter

Weiters gibt es auch Raumschiffe, welche eine Spur hinterlassen, und «Puffer Train» genannt werden. Einige der Raumschiffe können zudem von anderen Konfigurationen «produziert» werden. Ein Beispiel dafür ist die «Gleiterkanone», welche alle 30 Zeitschritte einen Gleiter entsendet, und somit unendliches Wachstum ermöglicht. [3]

Regeln und Variationen

Grundsätzlich können zelluläre Automaten beliebig viele Dimensionen haben. Neben den zweidimensionalen spielen ausserdem die eindimensionalen Automaten eine wichtige Rolle. Bei einem eindimensionalen zellulären Automaten sind alle Zellen nebeneinander in einer Reihe angeordnet. Um etwas beobachten zu können, wird die Waagrechte als Dimension für den Raum gebraucht und die Senkrechte für die Zeit. Die Regeln von eindimensionalen Automaten können

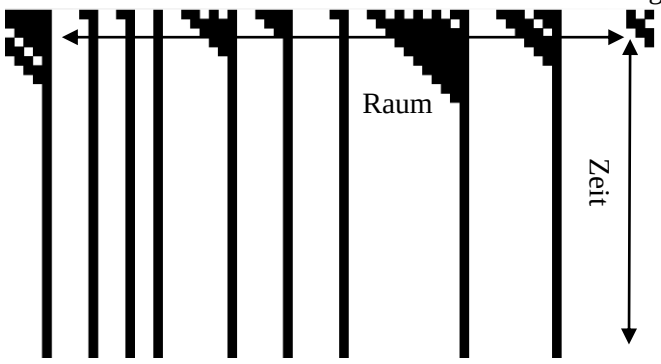
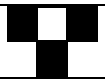
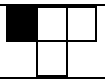
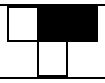
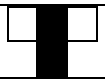
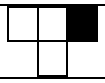
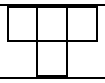


Abb. 5: Darstellung von Raum und Zeit

dabei mit Hilfe einer Tabelle dargestellt werden. Dabei wird in der obersten Zeile die Nachbarsituation angegeben und darunter der Zustand der Zelle beim nächsten Zeitschritt. Die Regeln lassen sich auch als binäre Zahl auffassen, wobei eine 1 für eine lebende (schwarze) und eine 0 für eine tote (weisse) Zelle steht. Wenn man diese so erhaltene binäre Zahl nun in eine dezimale Zahl umwandelt, bekommt man die Nummer der Regel. Diese gängige Methode wird in dieser Arbeit

								
1	1	1	0	0	1	0	0	
$1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 228$								

Tab. 1: Beispiel zur Angabe der Regel

bei eindimensionalen Automaten mit zwei Nachbarn (einer links und einer rechts) verwendet. Die Methode lässt sich auch auf eindimensionale Automaten mit beliebig vielen Nachbarn sowie auf zweidimensionale Automaten anwenden. Da jedoch die Umrechnungen dabei lang werden, wird darauf verzichtet. Deshalb werden die Regeln in der Form $s_1, s_2, \ldots / b_1, b_2, \ldots$ angegeben, wobei s_i angibt, bei welcher Anzahl Nachbarn eine Zelle überlebt, und b_i, bei welcher Anzahl eine Zelle geboren wird. «Life» hat somit die Regel 2, 3/ 3. Die Regeln so anzugeben, ist jedoch nur bei totalistischen Automaten möglich, also wenn die räumliche Verteilung der Nachbarn keine Rolle spielt. Dies ist bei «Life» der Fall, da es beispielsweise nicht darauf ankommt, ob sich drei lebende Zellen alle links oder alle rechts von einer Zelle befinden. Zudem sind bei «Life» die Übergangsregeln deterministisch, das heisst, wenn man mit einem bestimmten Anfangszustand startet, wird man nach einer bestimmten Anzahl Schritte immer das gleiche Resultat haben. Die Regeln erlauben nur einen möglichen nächsten Zustand einer Zelle. Es ist auch möglich, mit stochastischen Übergansregeln zu arbeiten, was bedeutet, dass eine Zelle mit einer bestimmten Nachbarschaftssituation mit einer gewissen Wahrscheinlichkeit einen Zustand annehmen wird, sodass man nach einer bestimmten Anzahl Schritte auf verschiedene Resultate kommen kann. Zudem können die Zellen eines zellulären Automaten nicht nur die Form eines Quadrats haben. Es sind auch gleichseitige Dreiecke, regelmässige Sechsecke oder auch eine Zusammensetzung aus verschiedenen Polygonen möglich.

Klassen

Die Einteilung der Automaten in 4 Klassen wurde von Stephen Wolfram vorgenommen, wobei die Zuteilung nicht immer ganz eindeutig ausfällt. Trotzdem gibt sie einen Überblick über das Verhalten von zellulären Automaten.

Klasse 1

Die Klasse 1 stellt hierbei die einfachste dar. Die Zellen von Automaten dieser Klasse erreichen nach einigen Zeitschritten alle den gleichen Zustand, unabhängig vom Startzustand des Automaten, wobei es einige wenige Ausnahmen geben kann. [4]

Abb. 6: Regel 96 links und 238 rechts als Vertreter der Klasse 1

Klasse 2

Bei Automaten der Klasse 2 entwickelt sich ein unsterbliches Muster, welches sich im Laufe der Zeit nicht mehr verändert oder periodisch ist. Verschiedene Startzustände führen hier nicht zum exakt gleichen Ergebnis, jedoch zu einem qualitativ ähnlichen Endzustand. Bei eindimensionalen Automaten mit zwei Zuständen besteht der Endzustand oftmals aus Blöcken von Einsen und Nullen, die sich periodisch wiederholen. [4]

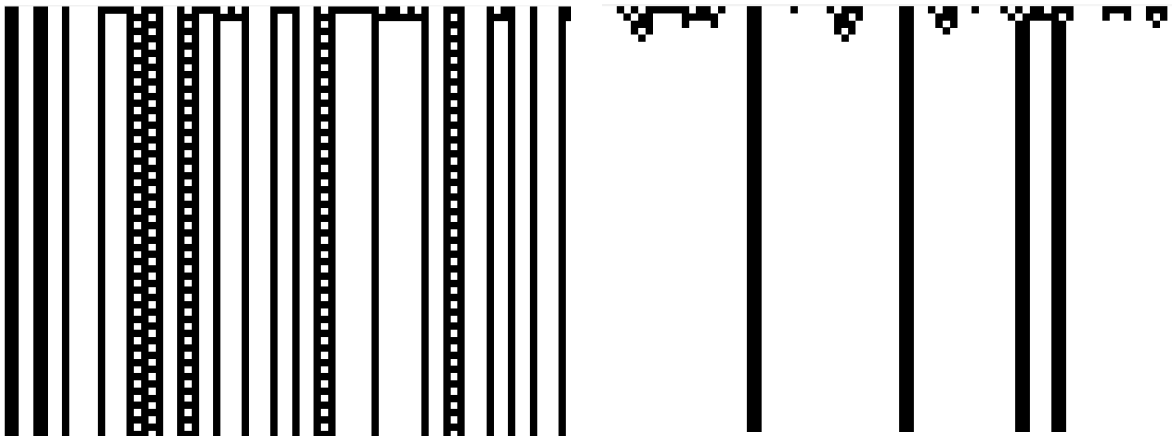


Abb. 7: Regel 108 links und 104 rechts als Vertreter der Klasse 2

Klasse 3

Bei der 3. Klasse lässt sich im Gegensatz zu den vorherigen Klassen eine Struktur nicht mehr leicht erkennen. Da die Muster dieser Automaten einer ständigen Veränderung unterworfen sind, werden sie oft als chaotisch bezeichnet. Meist sind bei eindimensionalen Automaten oft grössere und kleinere, weisse oder schwarze Dreiecke zu erkennen. Bei veränderten Startbedingungen ändern sich die Muster zwar im Detail, die Charakteristika bleibt jedoch wie bei der Klasse 2 erhalten. Jedoch kann sich eine kleine Änderung des Startzustandes oft über den ganzen Automaten ausbreiten. [4]

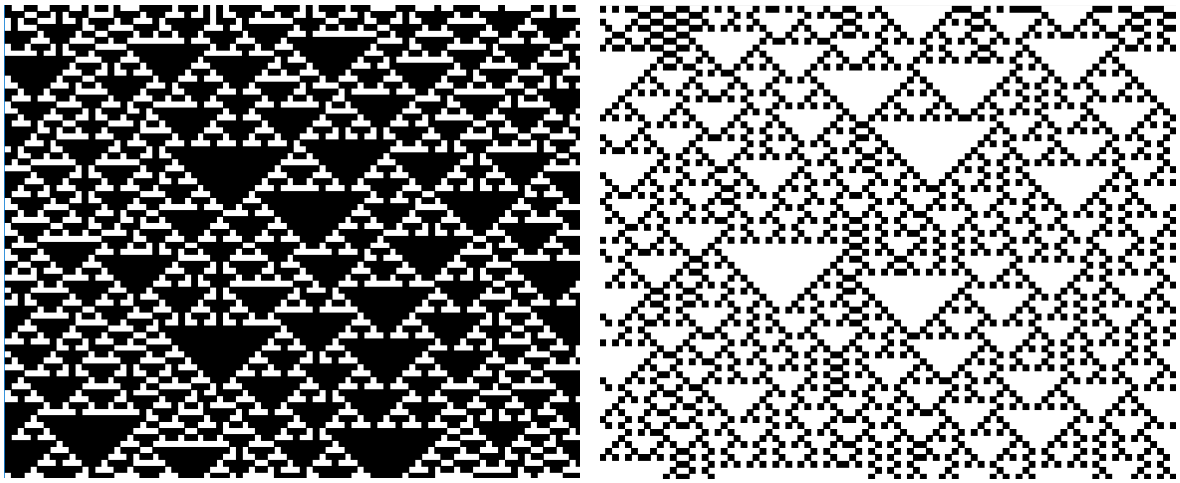


Abb. 8: Regel 151 links und Regel 18 rechts als Vertreter der Klasse 3

Klasse 4

Die Zuteilung zur Klasse 4 erweist sich als schwierig, da sie keine charakteristischen Muster aufweisen. Automaten der Klasse 4 sind gegenüber veränderten Startbedingungen nicht robust. Oft entwickeln sich komplexe Muster, welche ewig leben. Ausserdem kann man Arten «wandernder» Konfigurationen finden, welche nicht mehr stationär sind, sondern sich über das Spielfeld bewegen, also sogenannte Raumschiffe. Automaten der Klasse 4 sind oftmals besonders interessant, da sie Turing-mächtig sein können. Man kann damit also alles berechnen, was eine Turingmaschine berechnen kann, was so viel heisst wie universelle Programmierbarkeit. Beispiele dafür sind Programmiersprachen wie Java oder auch C#, welche Turing-vollständig sind. [4]

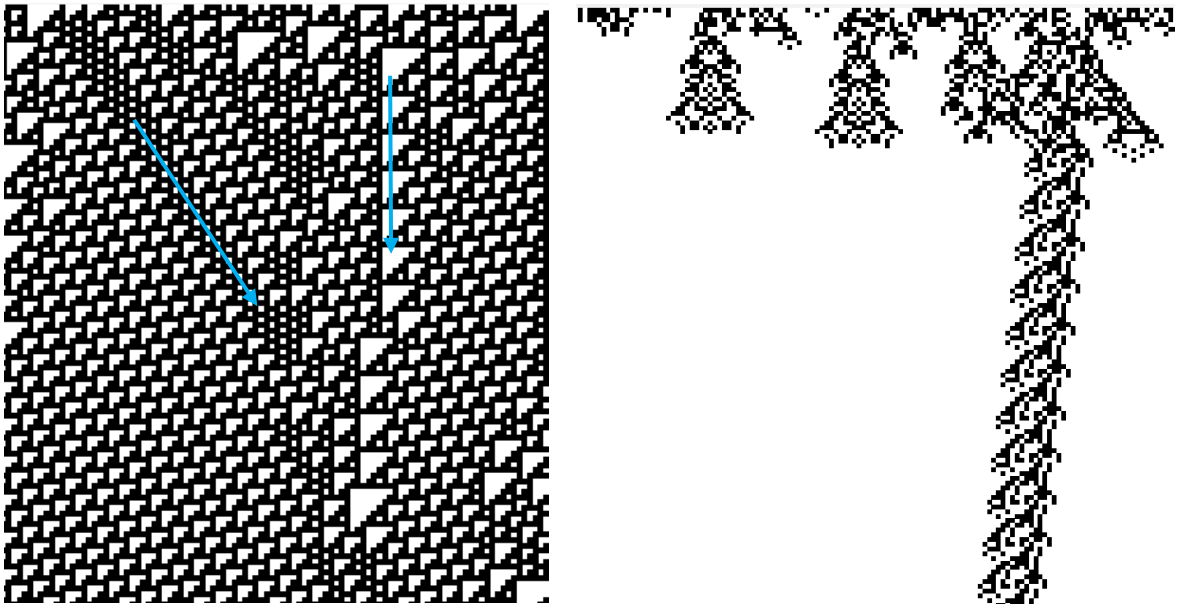


Abb. 9: Regel 110 links und Regel 1,3/2,4 mit 4 Nachbarn rechts als Vertreter der Klasse 4

Bei der Regel 110 ist eine Art Hintergrundmuster vorzufinden, welches sich immer wiederholt. Zwei Raumschiffe wurden bei dem oberen Beispiel jeweils mit einem Pfeil markiert. [5]

Langtons Parameter λ

Wird ein zellulärer Automat mit einem quadratischen Gitter, einer Moore-Nachbarschaft und zwei möglichen Zuständen gewählt und möchte man nun alle Regeln ausprobieren, stösst man auf ein Problem. Es existieren bei einem Automaten mit k Zuständen und n Nachbarn k^{n+1} verschiedene Ausgangssituationen zum Zeitpunkt t, wobei bei jeder Situation der Zustand der Zelle zum Zeitpunkt $t+1$ festgelegt werden muss. Daraus folgt, dass es $k^{k^{n+1}}$ verschiedene Regeln gibt und somit bei diesem Beispiel mit k = 2 und n = 8 ungefähr $1.341 \cdot 10^{154}$. [6] Daher ist es sinnvoll, nur die totalistischen Regeln zu betrachten, wobei jedoch immer noch 262'144 übrigbleiben. Man muss also einen Weg finden, um auf interessante Regeln zu stossen, ohne alle ausprobieren zu müssen.

Der Langtonparameter λ gibt an, wie gross die Wahrscheinlichkeit ist, dass eine Zelle im nächsten Zeitschritt nicht den Zustand 0 annimmt.

$$\lambda(\phi) = 1 - \frac{\aleph_0[\phi]}{k^N}$$

Hierbei handelt es sich um einen zellulären Automaten mit N Nachbarn, k mögliche Zuständen und der Übergangsregel ϕ. $\aleph_0[\phi]$ ist die Anzahl der Nachbarkonfigurationen, welche dazu führen, dass die Zelle den Zustand 0 annimmt. [7] Dabei wird die grösste Unordnung bei Automaten mit $\lambda = 1 - 1/k$ erreicht, während Automaten mit $\lambda = 0$ und $\lambda = 1$ komplett geordnet sind. Wenn man nun Automaten mit $0 \leq \lambda \leq 1 - 1/k$ betrachtet, lässt sich ein Übergang von geordneten Systemen in chaotische Systeme beobachten. Man stösst, wenn man den Parameter von 0 erhöht, zuerst auf die Klasse 1, dann auf die Klasse 2 und schlussendlich auf die Klasse 3. Automaten der Klasse 4 lassen sich vereinzelt im Übergangsbereich von der Klasse 2 in die Klasse 3 finden. Der Parameter gibt jedoch nicht exakt an, welcher Automat wo zu finden ist, es lässt sich bloss eine ungefähre Verteilung beobachten. [8] Wenn man also einen zellulären Automaten mit Raumschiffen suchen will, ist es sinnvoll diese im Übergangsbereich zu suchen.

Bei Life ist $\lambda = \frac{\binom{8}{2}+2\cdot\binom{8}{3}}{2^9} \cong 0.274$.

Geschichte und Anwendungsbereiche

Als Vater der zellulären Automaten gilt John von Neumann, dessen Ziel es um 1940 war, eine künstliche Maschine zu bauen, die sich selbst reproduzieren kann. Diese Maschine sollte jedoch nicht aus Metallteilen und Drähten bestehen, sondern lediglich ein Gedankenexperiment sein. Stanislaw Ulam war Neumann beim Lösen dieses Problems behilflich und schlug ihm einen geeigneten Formalismus, eine bestimmte Art von Automaten, vor. Neumann nannte die Felder des Automaten «Zellen», parallel zu den Grundbausteinen des Lebens, womit der Name «zellulärer Automat» entstand. Die Zellen von Neumanns zellulärer Automaten konnten 29 verschiedene Zustände annehmen, womit dieser Automat ziemlich komplex ist. [9] 1968 entwickelte John Conway einen zellulären Automaten namens «Game of Life», dessen Regeln um einiges einfacher sind als von Neumanns Automaten. Conway und seine Kollegen experimentierten dafür fast zwei Jahre lang, wohlgemerkt von Hand, ohne Unterstützung eines Computers. Anschliessend nutzte Martin Gardner seine Spalte «Mathematical Games» im Scientific American um immer wieder neue Entwicklungen von «Life» vorzustellen, wodurch es schnell Berühmtheit erlangte. [10] Um 1980 begann sich dann

Stephen Wolfram mit den zellulären Automaten zu beschäftigen, veröffentlichte zahlreiche Studien und schliesslich 2002 sein Buch «A New Kind Of Science». Dabei untersuchte er vor allem die Anwendungsbereiche, besonders in der Physik, und nahm eine Klassifikation der eindimensionalen Automaten vor. [11] Zelluläre Automaten sind also nicht nur eine «mathematische Spielerei», sondern können in verschiedenen Gebieten der Biologie, Chemie und Physik bei Simulationen eingesetzt werden. Beispiele hierfür sind Verkehrssimulationen, Simulationen in der Strömungsmechanik, in der Ökologie (Populationen von Räuber und Beute), bei der Ausbreitung von Druckwellen, in der Soziologie oder bei der Musterbildung auf Fellen. Ein interessantes Beispiel dafür ist die Muschel Cymbiola innexa, deren Muster eine gewisse Ähnlichkeit mit dem eindimensionalen Automaten der Regel 18 aufweist. [12] Ausserdem ergibt sich mit der Regel 18,

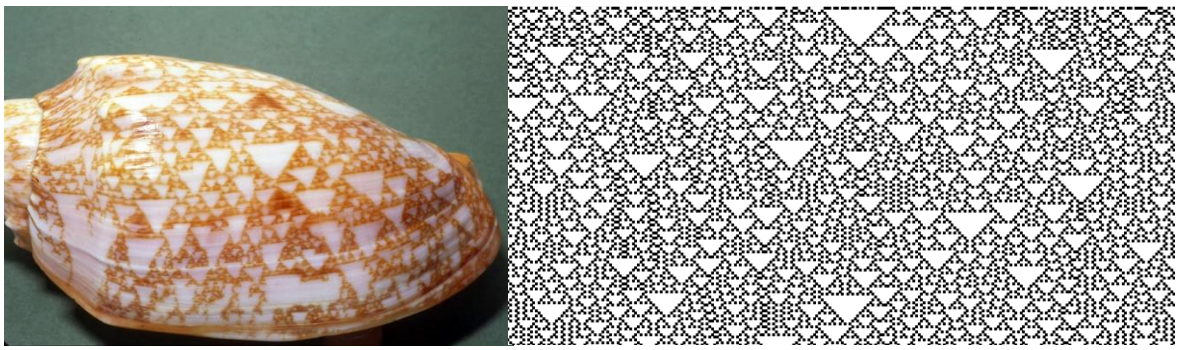


Abb. 10: Muschel Cymbiola innexa und Regel 18

wenn man mit einer lebendigen Zelle startet, ein Sierpinski-Dreieck. Ein Sierpinski-Dreieck ist ein Fraktal, also eine selbstähnliche Struktur. Zoomt man hinein, so wird man immer das gleiche Muster vorfinden. Weiters sind zelluläre Automaten auch Anschauungsobjekte für Bereiche der Komplexitätsforschung, zum Beispiel wenn es um den Schmetterlingseffekt geht. Dafür sind besonders Automaten der Klasse 3 interessant, die oftmals besonders anfällig gegenüber veränderte Startbedingungen sind und bei denen sich Störungen über weite Strecken ausbreiten können. Darüber hinaus ergeben sich auch kreative Anwendungsmöglichkeiten in der Kunst sowie auch in der Musik. Es ist beispielsweise möglich, harmonisch klingende Stücke mithilfe von zellulären Automaten zu schreiben. [13]

2.2 Programmieren

C#

C# ist eine Allzweck-Programmiersprache, welche von Microsoft unter der Leitung von Andres Hejlsberg entwickelt wurde und erstmals im Jahr 2000 erschienen ist. Zuvor war Microsoft damit beschäftigt, einen eigenen Java Compiler zu entwickeln, dessen erzeugter Code jedoch nur auf Windows lief. Daraufhin wurde Microsoft von Sun, welches Java entwickelt hatte, verklagt. Microsoft begann anschliessend mit der Entwicklung von C#. [14] Dabei wurde auf Konzepte von Java, C++, C und Delphi zurückgegriffen und versucht, jeweils das Beste dieser Sprachen zu kombinieren. Es wurde darauf geachtet, dass sich für Java- und C++-Programmierer ein einfacher Einstieg ergibt. [15] C# ist eine objektorientierte Programmiersprache, das heisst sie baut auf Klassen auf. Wird also eine neue Instanz einer bestimmten Klasse erstellt, so besitzt diese Instanz die typischen Eigenschaften seiner Klasse. Ein Teil der Attribute kann beim Programmieren und auch während der Laufzeit des Programmes geändert werden. Man kann es sich bildlich so vorstellen,

dass man beispielsweise zuerst bestimmt, dass die Instanz ein Schmetterling sein soll, also zur Klasse der Schmetterlinge gehört. Somit hat die Instanz nun die Eigenschaften eines Schmetterlings, besitzt also Flügel, kann fliegen, ernährt sich von Nektar etc. Anschliessend kann man dann einen Zitronenfalter daraus machen, indem man festlegt, dass die Flügel gelb sind und eine Spannweite von ca. 50 mm haben. [16] Dabei nutzt C# das .Net-Framework, eine Klassenbibliothek (darin sind die Informationen über die Klasse der Schmetterlinge gespeichert). Teile vom .Net-Framework können dabei mit «using» zu Beginn eines Programms eingebunden werden. [15]

Visual Studio

Visual Studio ist eine Entwicklungsumgebung, die sich für die Entwicklung von Desktop-, Web-, Smartphone- sowie Windows-Store-Apps-Software eignet. Visual Studio unterstützt jedoch nicht nur C#, sondern auch weitere Sprachen wie C, C++, Python, HTML, Java, CSS oder auch JavaScript. [17] Von Visual Studio gibt es drei Versionen, wobei die Express Edition, kostenlos erhältlich ist. [18] Die Express Edition weist zwar einige Nachteile gegenüber der Standard- und Professional-Edition auf, ist aber für den Beginn und eher einfache Programme völlig ausreichend.

Beim Erstellen einer Form-Anwendung lässt sich mithilfe einer Entwurfsansicht leicht die Benutzeroberfläche eines Programms gestalten. Dabei ist es möglich, Steuerelemente aus der Toolbox wie zum Beispiel Buttons, Panels, Labels oder auch Statusleisten mit Hilfe von «Drag-and-Drop» zu platzieren. Die Eigenschaften (Name der Instanz, Grösse, Farbe, Text etc.) können dabei über das Eigenschaftsfenster angepasst werden. Die Form1.Desinger.cs-Datei (hier «FrmMain.Designer.cs»), in welcher sich der Code befindet, mit dem die Eigenschaften der Instanzen festgelegt werden, wird dabei selbständig von Visual Studio entsprechend angepasst. Der eigentliche Teil des Programmierens findet dann in der Datei «Form1.cs» (hier «FrmMain.cs») statt, dort wird also der Code geschrieben und die Methoden aufgerufen. Zudem existiert noch die Datei Programm.cs (effektiv dann «GameOfLIfe.cs», «eindimensionalerZA.cs» etc.), die als Einstiegpunkt für das Programm dient und über welche gegebenenfalls Administratorechte etc. geprüft werden können. [19]

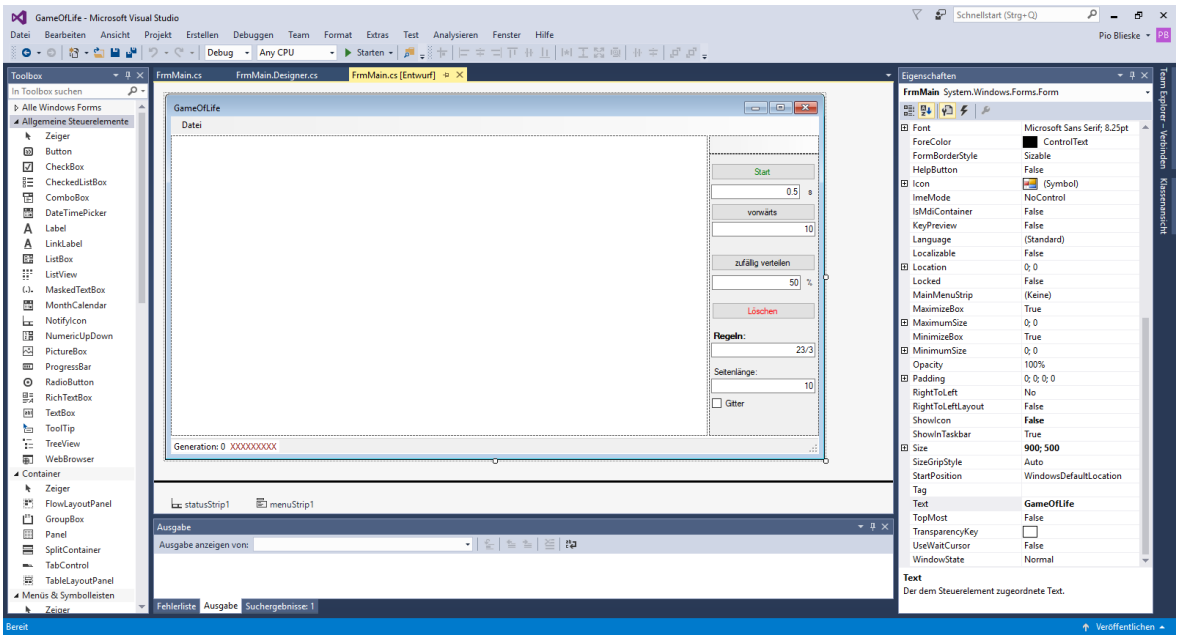


Abb. 11: Entwurfsansicht von Visual Studio mit Toolbox links und Eigenschaftsfenster rechts

3. Material und Methodik

3.1 Programmieren

In diesem Abschnitt geht es um die Programmierung der Automaten. Dabei soll es nicht das Ziel sein, eine besonders anwendungsfreundliche und optisch gutaussehende Benutzeroberfläche zu gestalten, sondern für die folgende mathematische Betrachtung geeignete Programme zu schreiben. Da es unter den verschiedenen Programmen viele Gemeinsamkeiten gibt, wird die Technik und Idee jeweils bei einem Programm erklärt und bei den dachfolgenden jeweils nicht nochmals aufgezeigt. Der komplette Code der FrmMain.cs-Dateien aller Programme ist auf der CD-ROM zu finden. Bevor ich angefangen habe zu programmieren, habe ich meine Kenntnisse mit der App «Learn C#» von «SoloLearn» aufgefrischt. [20] Während des Programmierens wurden die zellulären Automaten mit bereits bekannten oder selbst ausgedachten und von Hand überprüften Mustern getestet, um sicher zu gehen, dass die Programme richtig funktionieren. Dabei wurden jeweils besonders die Randregionen aufgrund der eher anspruchsvollen Berechnungen an diesen Stellen kontrolliert.

Eindimensionaler Automat

Für die Simulation eines eindimensionalen zellulären Automaten werden die momentanen Zustände (0 oder 1) in einer Arraylist namens «list» gespeichert, wobei das erste Element den Index 0 hat und die darauffolgenden Elemente durchlaufend nummeriert sind. Eine Arraylist wird gewählt, da deren Länge sich zur Laufzeit ändern kann, was bei einer normalen Liste nicht möglich ist. Da es nicht möglich ist, einen unendlich langen zellulären Automaten zu programmieren, wird jeweils ein endliches Zellularfeld simuliert. Den Zellen ausserhalb des Feldes wird jeweils der Zustand 0 zugordnet oder es werden die Ränder miteinander verbunden, sodass sich der Automat auf einer Schleife befindet.

Grafik

Die graphische Benutzeroberfläche setzt sich zusammen aus einem Panel «pnl», auf dem der zelluläre Automat abgebildet wird, einem Toolstripcontainer, in welchem sich die Buttons, Textboxen und Checkboxen befinden, einer Menü-Leiste, welche für das Öffnen und Speichern von Dateien gebraucht wird, und einer Statusleiste, wo die Fehler ausgegeben werden. Diese graphischen Elemente inklusive allen Buttons, Checkboxen etc. wurden mit Hilfe der Entwurfsansicht an Ort und Stelle gebracht.

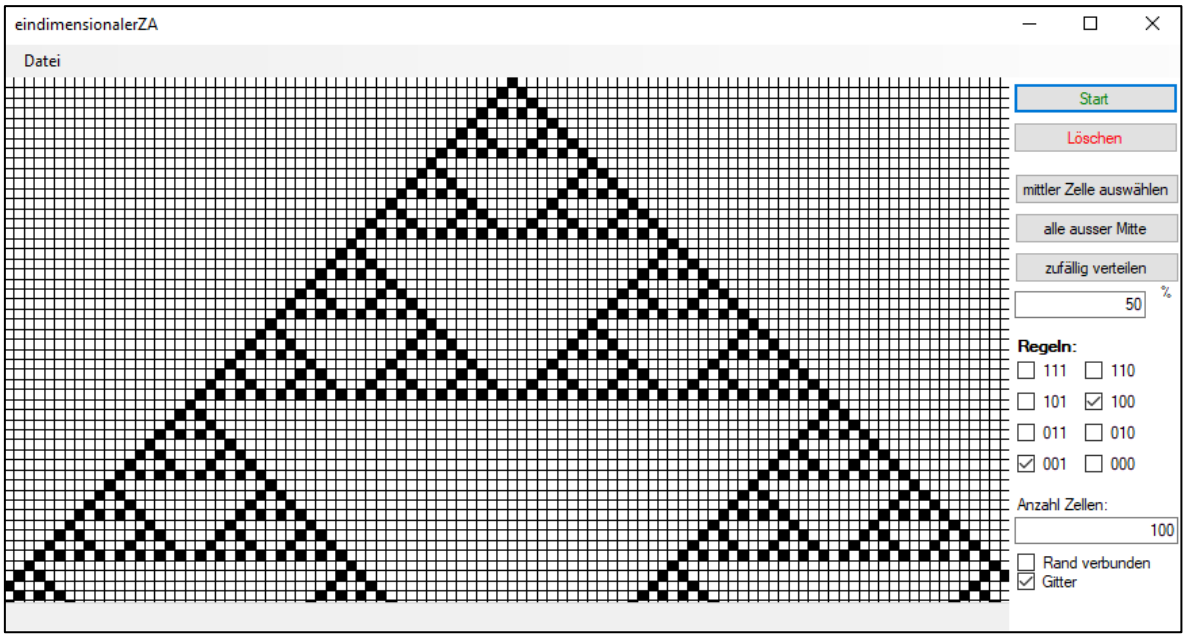


Abb. 12: Benutzeroberfläche des Programms für eindimensionale zelluläre Automaten

Um den zellulären Automaten auf das Panel zu zeichnen, wird eine Variable l für die Länge des Automaten, also der Anzahl Zellen, s für die Seitenlänge einer Zelle sowie x und y, welche die Breite beziehungsweise die Höhe des Panels angeben, gebraucht. Diese Variablen werden gleich beim Start des Programmes initialisiert und festgelegt.

Beim Aufrufen des Befehls «pnl_Paint» wird zuerst die Graphik «g» kreiert. Dann wird, falls ein Häkchen bei der Checkbox «cbxGitter» vorhanden ist, das Gitter gezeichnet. Dafür werden zwei For-Schleifen[2] gebraucht, eine für die vertikalen und eine für die horizontalen Linien. Anschliessend wird geprüft, ob die Liste «list» ein oder mehrere Elemente enthält. Ist dies nicht der Fall, was beim Starten des Programms oder auch nach dem Löschen des Zustandes sein kann, wird die Liste mit Nullen gefüllt. Die erste Zeile, welche den Startzustand aufzeigt, wird immer gezeichnet. Dabei wird die FillRectangle-Methode gebraucht. [21] Die Koordinaten werden in Abhängigkeit vom Index i und der Seitenlänge s einer Zelle angegeben. Dabei wird die Länge der Rechtecke mit der Addition 0.5 leicht erhöht, um zu verhindern, dass durch das Abrunden beim Umwandeln in eine Integer-Grösse Lücken zwischen den einzelnen Zellen entstehen.

```
private void pnl_Paint(object sender, PaintEventArgs e)
{
    Graphics g = pnl.CreateGraphics();
    Pen p = new Pen(Color.Black, 1);
    SolidBrush sb = new SolidBrush(Color.Black);
    //Gitter
    if (cbxGitter.Checked)
    {
        for (int i = 1; i <= l; i++)
        {
            g.DrawLine(p, Convert.ToInt32(i * s), 0, Convert.ToInt32(i * s), y);
        }
        for (int i = 1; i <= h; i++)
        {
```

[2] Eine For-Schleife wiederholt eine Anweisung. Dafür wird jeweils ein Integer i angegeben, welcher bei jedem Durchlauf verändert wird, hier beispielsweise mit i++ jeweils um 1 erhöht. Die For-Schleife wiederholt die Anweisung solange, wie die Bedingung erfüllt ist, z.B. i <= l.

```
            g.DrawLine(p, 0, Convert.ToInt32(i * s), x, Convert.ToInt32(i * s));
        }
    }
    //Liste vorbereiten
    if (list.Count == 0)
    {
        for (int i = 0; i < l; i++)
        {
            list.Add(0);
        }
    }
    //erste Reihe Zeichnen
    slist.Clear();
    for (int i = 0; i < list.Count; i++)
    {
        slist.Add(x);
        if (Convert.ToInt32(list[i]) == 1)
        {
            g.FillRectangle(sb, Convert.ToInt32(i * s), 0, Convert.ToInt32(s +
0.5), Convert.ToInt32(s + 0.5));
        }
    }
```

Tab. 2: Code für das Zeichnen eines zellulären Automaten

Bei Änderungen, zum Beispiel wenn die Anzahl der Zellen wechselt, muss die Grafik angepasst werden. Dazu werden über den Befehl «anpassen» die benötigten Variablen neu berechnet und anschliessend das Panel mit dem Befehl «pnl.Refresh» neu gezeichnet. Bei einer Änderung der Fenstergrösse werden zusätzlich die Objekte (Panel, ToolStripContainer etc.) neu platziert und das Panel anschliessend mit «anpassen()» angepasst. Dabei wird auf ein ständiges Neuzeichnen verzichtet, wenn schon ein zellulärer Automat komplett aufgezeichnet wurde (also schon Start gedrückt wurde), da es ansonsten zu einem «ruckelenden» Fenster beim Verändern der Grösse kommen würde. Jedoch lässt sich der komplette Automat leicht mit einem nochmaligen Klick auf Start wiederherstellen, falls die Grösse des Fensters geändert wurde.

Start-Button

Bei einem Klick auf Start wird zuerst die Arraylist «list», die momentan noch den Startzustand des Automaten enthält, auf die «slist» kopiert, welche später beim allfälligen Speichern des Automaten benötigt wird (siehe Öffnen und Speichern). Danach wird die Variabel «start» auf 1 gesetzt, wodurch beim anschliessenden Neuzeichnen des Panels alle Zustände des Automaten zu einem bestimmten Zeitpunkt gezeichnet werden, welche in dem Fenster Platz finden.

Durch den Befehl «pnl.Refresh()» und da die Variabel «start» jetzt den Wert 1 hat, werden beim Ausführen des Befehls «pnl_Paint» die restlichen Zeilen gezeichnet. Dabei werden in einer For-Schleife mit dem Befehl «nächsteReihe» die Zustände des Automaten für den nächsten Zeitschritt ermittelt und in der Arraylist «nlist» gespeichert. Nach dem Zeichnen der Zellen wird die Arraylist «list» geleert und die Elemente von «nlist» nach «list» kopiert. Zum Schluss werden dann die Zustände der «slist», welche noch die Anfangswerte enthalten, nach «list» übertragen, sodass nach dem Vorgang der Startzustand leicht variiert werden kann und beim Ändern der Fenstergrösse die Möglichkeit besteht, den Automaten bei Bedarf nochmals neu aufzuzeichnen.

```
//restliche Reihen
if (start == 1)
```

```
{
    for (int i = 1; i <= h; i++)
    {
        nächsteReihe();
        for (int ii = 0; ii < l; ii++)
        {
            if (Convert.ToInt32(nlist[ii]) == 1)
            {
                g.FillRectangle(sb, Convert.ToInt32(ii * s), Convert.ToInt32(i *
s), Convert.ToInt32(s + 0.5), Convert.ToInt32(s + 0.5));
            }
        }
        list.Clear();
        for (int ii = 0; ii < l; ii++)
        {
            list.Add(Convert.ToString(nlist[ii]));
        }
        nlist.Clear();
    }
    list.Clear();
    for(int i = 0; i < l; i++)
    {
        list.Add(slist[i]);
    }
    start = 0;
}
```

Tab. 3: Code für das Zeichnen des Automaten

Um den nächsten Zustand des zellulären Automaten mit dem Befehl «nächsteReihe» zu bekommen, werden für jede Zelle die Zustände der Nachbaren «a» und «c» ermittelt sowie der eigene Zustand «b» ermittelt. Mithilfe einer For-Schleife, welche l-mal durchlaufen wird (l ist gibt die Anzahl Zellen an), wird dies für jede einzelne Zelle getan. Hier ist i so gewählt, dass i dem Index der Zelle entspricht, deren Nachbaren ermittelt werden. Mit list[i] kann das Element mit Index i abgerufen werden. Für «a» braucht man also das Element mit Index i - 1, für «b» mit Index i und für «c» mit Index i + 1. Eine Ausnahme bilden dabei die Randregionen. Hier wird, wenn in der Checkbox «cbx.Randverbunden» ein Häkchen gesetzt ist, bei der Zelle mit Index 0 als Nachbar «a» die Zelle mit Index l genommen und bei der Zelle mit Index l als Nachbar «c» die Zelle mit Index 0, ansonsten wird dem Nachbarn der Zustand 0 zugeordnet.

```
private void nächsteReihe(Graphics g, SolidBrush sb, int t)
{
    for (int i = 0; i < l; i++)
    {
        //Zustände der Nachbarn
        int a, c;
        int b = Convert.ToInt32(list[i]);
        if (i + 1 < l)//rechter Nachbar
        {
            c = Convert.ToInt32(list[i + 1]);
        }
        else //rechts aussen
        {
            if (cbxRandverbunden.Checked)
            {
                c = Convert.ToInt32(list[0]);
            }
            else
            {
```

```
                c = 0;
            }
        }

        if (i - 1 >= 0) //linker Nachbar
        {
            a = Convert.ToInt32(list[i - 1]);
        }
        else //links aussen
        {
            if (cbxRandverbunden.Checked)
            {
                a = Convert.ToInt32(list[l]);
            }
            else
            {
                a = 0;
            }
        }
```

Tab. 4: Code für das Berechnen des nächsten Zustandes

Anschliessend wird geprüft, ob bei der Checkbox, welche für die entsprechende Situation steht, ein Häkchen gesetzt ist. Ist dies der Fall, wird die Zelle im nächsten Zeitschritt den Zustand 1 annehmen, ansonsten 0. Die neuen Zustände werden anschliessend in der List «nlist» gespeichert.

```
//Regeln
if (a == 0 && b == 0 && c == 0)
{
    if (cbx000.Checked)
    {
        nlist.Add(1);
    }
    else
    {
        nlist.Add(0);
    }
}
. . . //gekürzt
```

Tab. 5: Handhabung der Regeln

Buttons und Eingaben

Die Länge des zellulären Automaten kann in die Textbox «**tbx.Anzahl**» eingegeben werden. Wird der Text hier verändert, wird zuerst mit einer Try-catch Methode sichergestellt, dass es sich dabei um eine natürliche Zahl handelt. Ist dies nicht der Fall, wird dies über das Label «lbl.Error» dem Nutzer mitgeteilt. Anschliessend wird das Panel neu gezeichnet, so dass der Startzustand aller Zellen 0 ist.

Der Startzustand des Automaten kann zufällig gewählt werden. Dafür kann in der «tbxWahrscheinlichkeit» die Wahrscheinlichkeit angegeben werden, mit welcher eine Zelle den Zustand 1 haben soll. Wenn dann auf den Button «**btnZufall**» klickt wird, wird zuerst geprüft, ob es sich bei der Eingabe der Wahrscheinlichkeit um eine reelle Zahl handelt. Anschliessend wird in einer For-Schleife für jede Zelle mit der Methode «Random» und dem Befehl «rnd.NextDouble» eine reelle Zahl zwischen 0 und 1 «gezogen». Ist diese kleiner als die angegebene Wahrscheinlichkeit,

wird der Zelle der Zustand 1 zugeordnet. Anschliessend wird der neue Startzustand auf dem Panel aufgezeichnet.

```
private void btnZufall_Click(object sender, EventArgs e)
{
    list.Clear();
    lblError.Text = "";
    double ps = 50;
    try
    {
        ps = Convert.ToDouble(tbxWahrscheinlichkeit.Text);
        if(ps >= 100 || ps < 0)
        {
             throw new Exception();
        }
    }
    catch
    {
        lblError.Text = "Die Wahrscheinlichkeit muss eine positive Zahl kleiner
gleich 100 sein.;
    }
    Random rnd = new Random();
    for (int i = 0; i < l; i++)
    {
        double d = rnd.NextDouble();
        if (d >= ps / 100)
        {
            list.Add(0);
        }
        else
        {
            list.Add(1);
        }
    }
    pnl.Refresh();
}
```

Tab. 6: Bestimmen eines zufälligen Startzustandes

Der **Startzustand** einer Zelle kann geändert werden, indem mit der **Maus** daraufgeklickt wird. Der Index der Zelle (hier «p» für Position) wird berechnet, indem die X-Koordinate «e.X» der Maus durch die Länge «s» einer Zelle geteilt wird. Da bei der Konvertierung einer double-Grösse in eine integer-Grösse immer abgerundet wird, erhält man so automatisch die richtige Position. Anschliessend wird geprüft, ob die ausgewählte Zelle schwarz oder weiss gewesen ist, und das Element mit «RemoveAt(p)» entfernt. Danach wird ein neues Element hinzugefügt, so dass 1 zu 0 und 0 zu 1 wird, und anschliessend das Panel neu gezeichnet. Da es bei einer grossen Anzahl Zellen schwierig ist, nur in die obere Reihe zu klicken, wird darauf verzichtet, dies zu überprüfen und nur der Zustand der Zelle zu ändern, wenn «y.E» kleiner als «s» ist.

```
private void pnl_MouseClick(object sender, MouseEventArgs e)
{
    int p = Convert.ToInt32(e.X / s);
    list.RemoveAt(p);
    if (Convert.ToInt32(list[p]) == 1)
    {
        list.Insert(p, 0);
    }
    else
    {
```

```
            list.Insert(p, 1);
        }
        pnl.Refresh();
    }
```

Tab. 7: Abfangen eines Maus-Events

Wenn auf den Button «**btnMitte**» oder auch «**btnAlleAusserMitte**» geklickt wird, wird jeweils die Liste neu aufgefüllt und dabei mit der If-else-Methode beim entsprechenden Index das entsprechende Element hinzugefügt. Mit einem Klick auf den Button «**btnLöschen**» wird die Arraylist «list» geleert, anschliessend allen Zellen den Zustand 0 zugordnet und das Panel neu gezeichnet, so dass man wieder einen «leeren» zellulären Automaten vor sich hat. Hier wird ausgenutzt, dass, wenn beim Zeichnen eine leere Liste vorliegt, diese mit Nullen gefüllt wird.

Speichern und Öffnen

Der zelluläre Automat kann mit Hilfe eines Save-File-Dialog («sfd») und dem String «text» in Form einer Textdatei gespeichert werden. Mit den ersten 8 Ziffern des Strings «text» werden die Regeln des Automaten festgehalten, wobei eine 1 einem Häkchen («Checked») und eine 0 keinem Häkchen («Unchecked») entspricht. Danach wird die Arraylist «slist» an den String angehängt und der ganze String mit einem Stream-Writer in die Datei «kopiert». [22]

```
private void speichernToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Text files (.txt)|*.txt";
    sfd.Title = "Speichern...";
    if (sfd.ShowDialog() == DialogResult.OK)
    {
        text = "";
        if (cbx000.Checked)
            text += "1";
        else
            text += "0";
        . . . //gekürzt
        for (int i = 0; i < slist.Count; i++)
        {
            text += slist[i];
        }
        System.IO.StreamWriter sw = new System.IO.StreamWriter(sfd.FileName);
        sw.Write(text);
        sw.Close();
    }
}
```

Tab. 8: Speichern eines Automaten

Beim Öffnen einer Datei wird der Open-File-Dialog zusammen mit einem Stream-Reader benutzt, welcher den Text der Datei in den String «text» «kopiert». Anschliessend wird die Länge des zellulären Automaten ermittelt. Darauf werden die Status der Checkboxen angepasst, wobei mit «text[i]» der Buchstabe des Strings mit dem Index i ermittelt werden kann. Danach wird der Rest des Strings in die Arraylist «list» mit Hilfe einer For-Schleife kopiert, wobei der Index jeweils um 8 erhöht werden muss, da die ersten acht Ziffern für die Regeln stehen. [22] Bei diesem Vorgang werden mit einer Try-catch-Methode Fehler abgefangen, welche entstehen können, wenn man beispielsweise versucht eine Datei mit dem falschen Format zu öffnen.

```
private void öffnenToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
    lblError.Text = "";
    try
    {
        OpenFileDialog ofd = new OpenFileDialog();
        ofd.Filter = "Text files(.txt)|*.txt";
        ofd.Title = "Öffnen...";
        if (ofd.ShowDialog() == DialogResult.OK)
        {
            text = "";
            System.IO.StreamReader sr = new System.IO.StreamReader(ofd.FileName);
            text = sr.ReadToEnd();
            sr.Close();
            l = Convert.ToInt32(text.Length - 8);
            list.Clear();
            nlist.Clear();
            if (Convert.ToString(text[0]) == "1")
                cbx000.CheckState = CheckState.Checked;
            else
                cbx000.CheckState = CheckState.Unchecked;
            . . . //gekürzt
            for (int i = 0; i < l; i++)
            {
                list.Add(Convert.ToString(text[i + 8]));
            }
            pnl.Refresh();
        }
    }
    catch
    {
        lblError.Text = "Datei konnte nicht geöffnet werden.";
    }
}
```

Tab. 9: Öffnen eines Automaten

Game of Life

Mit diesem Programm soll die Simulation von «Game of Life» und ähnlichen Automaten, bei denen die Zellen ebenfalls die Form eines Quadrats haben, ermöglicht werden. Auch hier werden die Zustände der Zellen in einer Arraylist gespeichert. Dabei wird der Zelle mit dem Index 0 der Platz oben links zugewiesen und dann in Leserichtung von links nach rechts und von oben nach unten die restlichen Zellen platziert.

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

Abb. 13: Automaten mit Länge 6

Grafik

Die Benutzeroberfläche ist ähnlich wie bei einem eindimensionalen Automaten aufgebaut. In der Statusleiste wird ausser den Fehlermeldungen auch die momentan vorliegende Generation, also der Zeitpunkt t, sowie die Anzahl lebendige Zellen angegeben. Die Regeln werden dabei nicht mit der Hilfe von Checkboxen angegeben, sondern können in einem Textfeld in der Form $s_1, s_2 \ldots / b_1, b_2 \ldots$ eingegeben werden. Die Zeitdifferenz in Sekunden zwischen zwei Zuständen kann dabei über die Textbox «tbxGeschwindigkeit» festgelegt werden. Beim Zeichnen des Automaten wird ähnlich wie beim

eindimensionalen Automaten vorgegangen. Der Unterschied besteht darin, dass die Zustände der Arraylist nun auf einem quadratischen Feld, welches die Zellen enthalten, aufgezeichnet wird. Dadurch, dass die Länge der Höhe des Automaten entspricht oder anders gesagt, die Anzahl der Zellen in der Horizontalen die gleiche wie in der Vertikalen ist, wird der Platz auf dem Panel meistens nicht komplett ausgenutzt. Dies ist optisch und vom Benutzererlebnis nicht gerade ideal, macht es aber um einiges leichter mit wechselnden Fenstergrössen umzugehen und erleichtert zudem das Speichern des Automaten. Zusätzlich ist es möglich, die zeitliche Entwicklung einer Zeile des Automaten zu betrachten, indem man unter Ansicht «Querschnitt» auswählt.

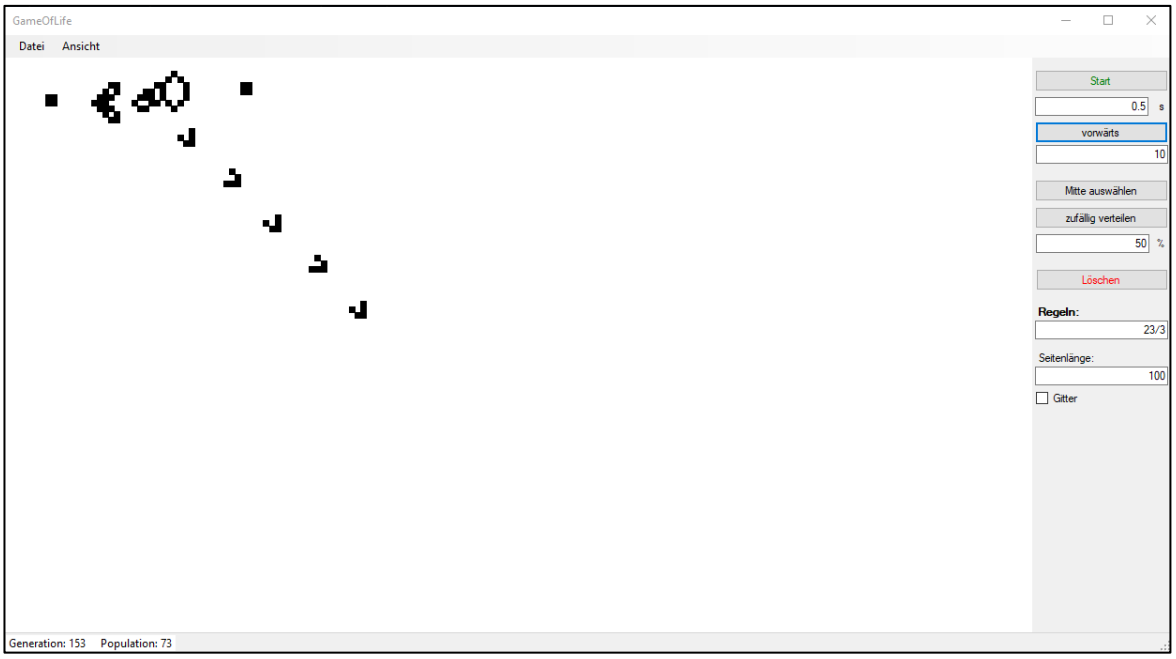


Abb. 14: Benutzeroberfläche von «GameOfLife»

Regeln

Mit dem Befehl «nächsterSchritt» wird der nächste Zustand des Automaten berechnet. Dabei wird in einer For-Schleife die Zelle zuerst auf spezielle Positionen untersucht, also ob sie sich in der obersten oder untersten Zeile oder in der Spalte ganz links oder rechts befindet. Anschliessend wird abhängig davon die Anzahl lebende Zellen unter den acht Nachbarn gezählt. Bei dem Automaten wird der rechte Rand mit dem linken und der obere mit dem unteren verbunden, wodurch der Automat sich auf der Oberfläche eines Torus befindet und dass Zellularfeld somit pseudo unendlich ist. Auch hier muss wieder darauf geachtet werden, ob die Zelle sich jeweils an einer speziellen Position befindet.

```
private void nächsterSchritt()
{
    for (int i = 0; i < l * l; i++)
    {
        int i2; //Zukünftiger Status
        string px = "n", py = "n"; //Position
        int n = 0; //Anzahl Nachbaren

        //Untersuchen auf spezielle Positionen
        if ((i + 1) % l == 0)
        {
            px = "r"; //rechts
```

```
        }
        . . . // gekürzt

        //Nachbaren zählen
        //oben rechts
        if (py != "u" && px != "r")
        {
            if (Convert.ToInt32(list[i + (l + 1)]) == 1)
                n += 1;
        }
        else if(py == "u" && px != "r")
        {
            if (Convert.ToInt32(list[i - l * (l - 1) + 1]) == 1)
                n += 1;
        }
        else if(py != "u" && px == "r")
        {
            if (Convert.ToInt32(list[i + 1]) == 1)
                n += 1;
        }
        else
        {
            if (Convert.ToInt32(list[0]) == 1)
                n += 1;
        }
        . . . // gekürzt
```

Tab. 10: Bestimmen der Anzahl Nachbarn

Da die Regeln nun über eine Textbox festgelegt werden, kann es zu Eingaben kommen, welche keine Regeln darstellen. Dadurch können Fehler entstehen, welche hier mit der Try-catch-Methode abgefangen werden. Um den Zustand einer Zelle beim nächsten Zeitpunkt zu ermitteln, wird zuerst die Position des Schrägstrichs im String «r» gesucht, welcher den Text mit den Regeln enthält. Anschliessend wird geprüft, falls eine lebende Zelle vorliegt, ob die Anzahl Nachbarn vor dem Slash vorzufinden ist. Wenn dem so ist, dann wird der Zustand im nächsten Zeitschritt 1 sein. Das Gleiche wird bei einer Zelle mit dem Zustand 0 getan. Dafür wird die Methode r.IndexOf(…) verwendet, die den Index des ersten Textstücks angibt, welche mit dem Gesuchten übereinstimmt. Wo die Suche beginnen soll, lässt sich mit einer zweiten Integer-Grösse angeben, welche mit einem Komma von dem gesuchten Textstück abgetrennt wird. [23]

```
//Regeln
string r = tbxRegeln.Text;
try
{
    int slash = r.IndexOf("/");
    if (Convert.ToInt32(list[i]) == 1 && r.Contains(Convert.ToString(n)))
    {
        if (r.IndexOf(Convert.ToString(n)) < slash)
            i2 = 1;
        else
            i2 = 0;
    }
    else if(Convert.ToInt32(list[i]) == 0 && r.Contains(Convert.ToString(n)))
    {
        if (r.IndexOf(Convert.ToString(n), slash) > slash)
            i2 = 1;
        else
            i2 = 0;
    }
    else
```

```
        i2 = 0;
    nlist.Add(i2);
}
catch
{
    lblError.Text = "Keine gültige Regel.";
}
```

Tab. 11: Lesen der Regeln und Bestimmen des nächsten Zustandes

Start

Der zelluläre Automat wird mit Hilfe eines Timers [24] regelmässig in den nächsten Zustand gebracht. Das Zeitintervall wird dabei über die TextBox «tbxGeschwindigkeit» festgelegt. Mit Hilfe des Befehles «timer_Tick» wird jedes Mal, wenn die festgelegte Zeit verstrichen und der Timer aktiviert ist, mit dem Befehl «nächsterSchritt» der nächste Zustand herbeigeführt. [25] Dabei kann der Timer mit einem Klick auf den entsprechenden Startbutton aktiviert werden, wodurch der Startbutton anschliessend den Text «Stop» zugewiesen bekommt und die Funktion des Stoppbuttons übernimmt. Zudem kann mit Hilfe des Buttons mit der Aufschrift «vorwärts» so viele Zeitschritte vorwärtsgegangen werden, wie man in der darunter stehenden Textbox eingibt.

Querschnitt

Zweidimensionale Automaten lassen sich jedoch nicht nur mit Aufnahmen zu verschiedenen Zeitpunkten betrachten. Wenn man die Zeit als Dimension noch dazu nimmt, erhält man ein dreidimensionales Gebilde. Von diesem lässt sich ein Querschnitt machen, so dass man die zeitliche Entwicklung einer Reihe von Zellen des Automaten verfolgen kann. [26] In der Menüleiste lässt sich unter Ansicht Querschnitt auswählen. Dadurch taucht ein zweiter Toolstripcontainer an der Position des gewöhnlichen auf, wobei der gewöhnliche mit .Hide() «versteckt» wird. In der Textbox «tbxReihe» kann man die Zeile eingeben, deren zeitliche Entwicklung man verfolgen möchte. Mit einem Klick wird dann berechnet, wie viele Zeitschritte benötigt werden, damit das Fenster optimal ausgenutzt wird. Der zelluläre Automat wird dann dementsprechend vielmal aktualisiert, wobei jeweils die Zustände der Zelle der entsprechenden Zeile in der Liste «schnitt» gespeichert werden.

Speichern

Bei diesem Programm werden die Regeln in der Form, in welcher sie in der Textbox eingegeben wurden, beim Speichern an den Anfang der Textdatei geschrieben. Die Regeln werden dann mit einem «a» vom Rest abgetrennt, welcher wie gewohnt den Zustand des Automaten enthält, also die Arraylist «list».

Zelluläre Automaten mit regelmässigem dreieckigen Mosaik

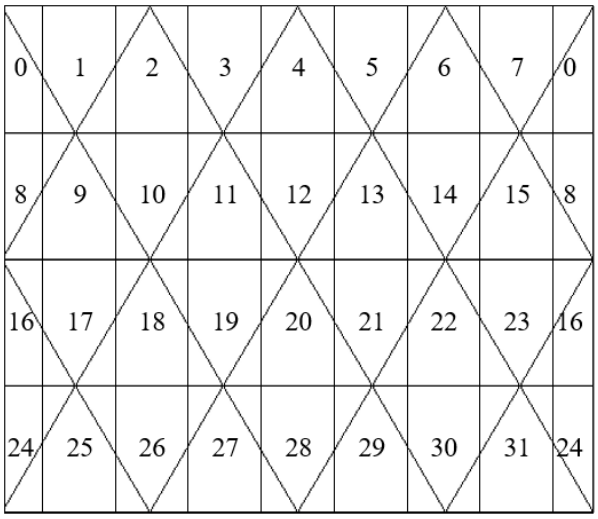


Abb. 15: Automat mit l = 4

Bei einem Automaten, bei welchem die Zellen die Form eines gleichseitigen Dreiecks haben, werden die Zellen wie bei «Life» durchnummeriert, wobei jeweils eine Zelle, die nach «oben» zeigt, mit einer, welche nach «unten» zeigt, abwechselt. Das Bild zeigt einen Automaten der Länge l = 4. Damit der Platz im Fenster einigermassen gut ausgenutzt wird, werden in der Horizontalen die doppelte Anzahl Zellen angebracht, hier in diesem Beispiel also 8. Um beim Auswählen einer Zelle mit der Maus zu bestimmen, um welche Zelle es sich handelt, wird geprüft, in welchem Rechteck sich die Maus befindet. Dadurch wird in ein paar Fällen eine falsche Zelle ausgewählt, was jedoch beim Benutzen des Programms nicht gross auffällt. Diese Technik hat den Vorteil, dass dadurch das Programmieren um einiges vereinfacht wird.

Zelluläre Automaten mit regelmässigem sechseckigen Mosaik

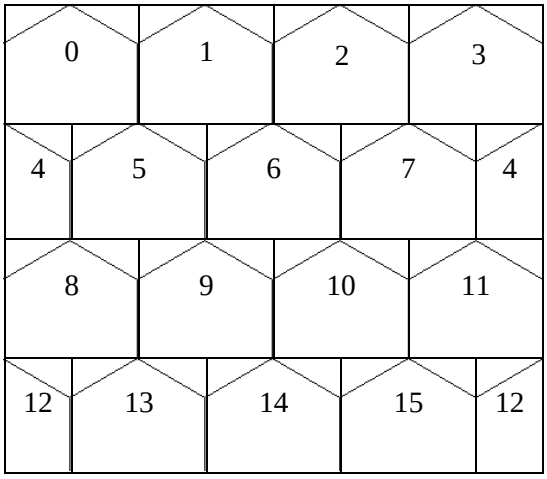


Abb. 16: Automat mit l = 4

Bei Automaten mit den sechseckigen Zellen werden wieder wie bei «Game of Life» gleich viele Zellen horizontal und vertikal angebracht. Dabei werden wie beim Automaten mit den Dreiecken wieder Rechtecke für das Bestimmen der Mausposition gebraucht und die Nummerierung wie im Bild links vorgenommen.

3.2 Mathematische Betrachtung

Wachstum und Zerfall

Das Untersuchen von Wachstum und Zerfall bei zellulären Automaten stellt eine Alternative zu Langtons Parameter bei der Suche nach Life ähnlichen Automaten dar, welche von David Eppstein entworfen wurde. Die Technik beruht darauf, dass es bei Life-ähnlichen Regeln Muster geben muss, die wachsen, und solche, die aussterben. Nur dann können Gleiter existieren, da Gleiter und Raumschiffe in eine bestimmte Richtung wachsen und in die übrigen Richtungen schrumpfen müssen. Aufgrund dessen lassen sich für Life-ähnliche Automaten einige Regeln ausschliessen. Bei diesem Kriterium besteht der Vorteil gegenüber dem Langtonparameter, dass man klar sagen kann, ob eine Regel Life-ähnlich sein könnte oder nicht. Um das Wachstum zu untersuchen wird dabei eine rechteckige Grenzbox definiert, welche ein beliebig gewähltes, endliches Muster enthält. Eine Regel wird als fruchtbar betrachtet, wenn ein endliches Muster existiert, welches alle seine Grenzboxen verlässt. Zudem wird ein endliches Muster als sterblich betrachtet, wenn eines der folgenden Muster keine lebenden Zellen mehr enthält. [27]

Suche in zufälligen Feldern

Bei der einfachsten Methode nach Objekten zu suchen startet man mit zufällig gewählten Konfigurationen und beobachtet deren Entwicklung. Oftmals werden auch systematisch alle Muster mit einer bestimmten Anzahl Zellen ausprobiert, zuerst also was mit einer lebenden Zelle, dann was mit allen möglichen Anordnungen von zwei, drei etc. passiert. Darunter befinden sich manchmal sogenannte Pentomino. Wie zum Beispiel das R-Pentomino in Life, welches aus nur fünf lebenden Zellen besteht, anfangs in alle Richtungen wächst und dabei auch mehrere Gleiter entstehen lässt. Das Vorgehen nach dieser Methode wird jedoch bei einer grösseren Anzahl lebender Zellen aufgrund der vielen Möglichkeiten schnell unmöglich. Eine andere Variante ist in einem Feld den Zustand der Zellen zufällig festzulegen und dann die Entwicklung zu beobachten. Dabei ist es sinnvoll, das Feld möglichst gross zu wählen, um möglichst viele interessante Objekte zu finden. Bei einigen Automaten der Klasse 4 sowie auch 2 nimmt die Anzahl der lebenden Zellen mit der Zeit ab. Um hier Rechnungszeit zu sparen und zu verhindern, dass sich Objekte gegenseitig vernichten, ist es sinnvoll, die Dichte der Startkonfiguration relativ tief (ca. 0.02 – 0.05) zu wählen. [28]

In der Forschung kam zuerst die Monte Carlo Methode auf, welche zufällig generierte Muster testet. Daraus wurde dann die Tabu-Methode entwickelt, welche bekannte Muster verändert und die besten davon auswählt. Dabei werden gefundene Objekte in einer Tabuliste gespeichert, um zu verhindern, dass bereits bekanntes getestet wird. Ausserdem liefern oftmals auch Kollisionen von Gleitern und Raumschiffen neue interessante Muster. Die momentan meist genutzte Methode scheint ein evolutionärer Algorithmus zu sein, welcher unterschiedliche Regeln untersucht und sich dabei mehr oder weniger an den Prinzipien der biologischen Evolution orientiert («survival oft the fittest»). Dieser Algorithmus besitzt eine sogenannte Fit-Funktion, welche unterschiedlich gewählte Kriterien enthält. Dabei wird von zufälligen Anfangszuständen mit verschiedenen Regeln gestartet und nach der Entwicklung des Automaten oftmals die Gleiterdichte ermittelt. Danach werden die «schlechten»

Regeln aussortiert, die guten behalten, anschliessend verändert und dann wieder getestet. Dieser Prozess wird mehrmals wiederholt, um die besten Regeln zu finden. [29]

Suche von Objekten

In dieser Arbeit werden zuerst Überlegungen zum Wachstum und Zerfall angestellt, um die Regeln dann so zu wählen, damit beides möglich ist. Mit dieser Methode lassen sich viele Regeln bei der Suche nach Gleiten sowie auch Oszillatoren ausschliessen, was einiges an nicht zielführender Arbeit erspart. Wird eine Regel ausgewählt, so wird die Methode gebraucht, bei welcher ein grosses Zellularfeld mit einer geringer Dichte betrachtet wird. Zudem wurden einige der gefundenen Objekte von Hand manipuliert, wodurch sich oft mit nicht übermässig grossem Aufwand neue Objekte mit ähnlichen Eigenschaften finden liessen. Das Suchen mit einem evolutionären Algorithmus würde dabei zwar effizienter sein und ausserdem erfolgreicher, jedoch müsste zuerst ein entsprechendes Programm für Automaten mit einem dreieckigen und sechseckigen Zellularfeld geschrieben oder zumindest angepasst werden, was der Rahmen dieser Maturaarbeit definitiv sprengen würde.

Untersuchung von wachsenden Strukturen

Bei der Untersuchung von wachsenden Strukturen wurde jeweils mit einfachen Regeln begonnen und anschliessend dann schwierigere und komplexere Muster betrachtet. Damit war eine Steigerung gut möglich und mir fiel es leichter, komplexere Strukturen zu beschreiben. Zudem konnten auch hier wieder Überlegungen zum Wachstum und Zerfall mit Hilfe der Grenzbox angestellt werden. Beim Betrachteten der Strukturen wurde jeweils besonders auf Regelmässigkeiten zwischen dem Aussehen und dem Zeitpunkt geachtet. Meistens begann es mit dem Bemerken einer Ausfälligkeit bei einer Regel, welche ich dann anschliessend bei anderen gezielt suchte oder auch zufällig wiederfand. Anschliessend wurde versucht mit Hilfe der Regeln eine Erklärung für das beobachtete Verhalten zu finden. Hier war es oftmals möglich aufgrund der diskreten Zeitschritte mit Folgen und Reihen zu arbeiten. Zudem war es einige Male möglich, eine Erklärung auf einer vorherigen aufzubauen. Um sich bei den gefundenen Formeln abzusichern, dass sich keine Fehler eingeschlichen haben, wurden sie jeweils mit Hilfe einer Simulation überprüft.

4. Resultate

4.1 Gefundene Objekte

Zum Testen der Methode und der angestellten Überlegungen wurden zelluläre Automaten mit einem sechseckigen Gitter und sechs Nachbarn gewählt. Alle Regeln mit B1 (siehe Seite 8) sind dabei unsterblich, da es unmöglich ist, ein endliches Muster zu konstruieren, bei dem keine tote, angrenzende Zelle eine lebende Nachbarzelle hat. Bei Regeln, welche nur B3, B4, B5 oder B6 enthalten und das Überleben der Zellen beliebig gewählt wird, ist kein Wachstum möglich. Wenn man eine Grenzbox festlegt, kann keine der toten Zellen ausserhalb mehr als zwei Nachbarn haben und somit nicht in den nächsten Zeitschritten lebendig werden. Eine Life-ähnliche Regel darf daher nicht B0 oder B1 enthalten und muss B2 beinhalten. Somit wird die Regel 3/2 mit $\lambda \cong 0.274$ gewählt.

Um nun die Regel 3/2 zu untersuchen, wird ein Feld mit der Länge von 400 Zellen und einer zufälligen Startkonfiguration gewählt, wobei jeweils eine Zelle mit einer Wahrscheinlichkeit zwischen 0.02 und 0.05 lebendig ist. Dabei stösst man zuerst auf einen Oszillator, welcher gewisse Ähnlichkeiten mit dem Blinker hat. Dieser «Blinker» kommt bei 512 verschiedenen Regeln vor, welche alle B2 enthalten sowie eine Kombination aus S2, S3, S4, S5, S6, B3, B4, B5 und B6. Ausserdem lassen sich drei weitere Oszillatoren mit Periode 2 finden. Weiters werden ein Oszillator mit Periode 3 und zwei mit Periode 4 gefunden. Bei den Oszillatoren mit Periode 4 ist jeweils der Zustand zum Zeitpunkt t achsen- und punktsymmetrisch zum Zustand des Zeitpunkts t + 2.

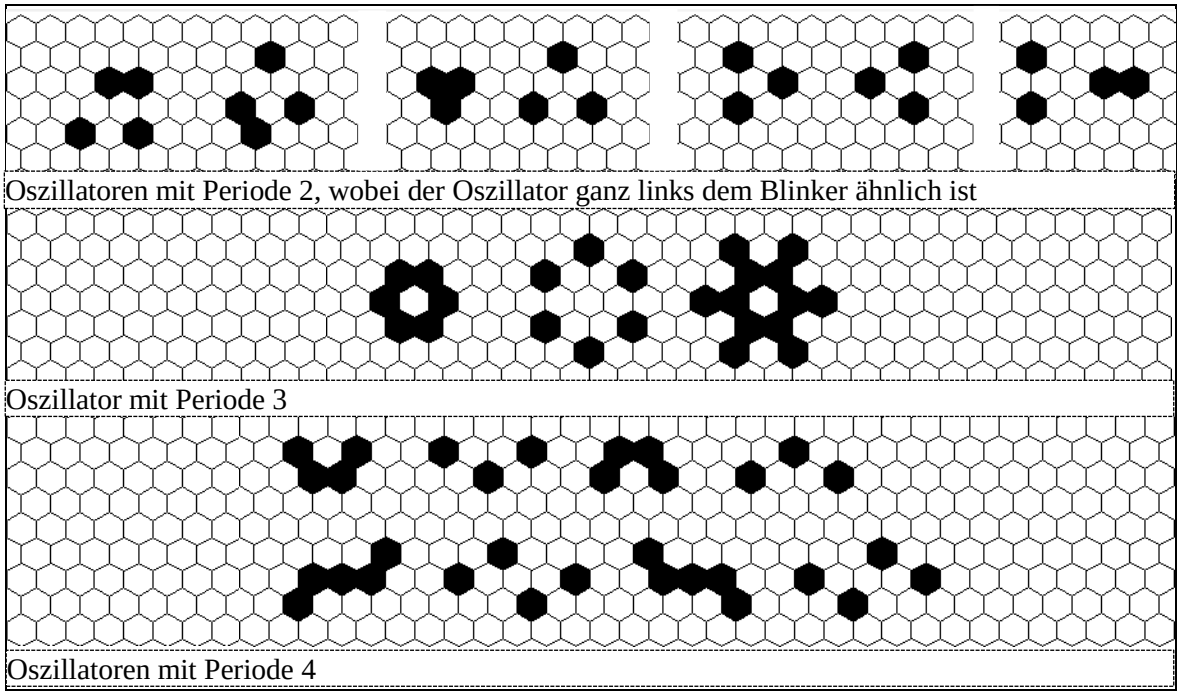


Abb. 17: Gefundene Oszillatoren der Regel 3/2

Dabei werden jedoch keine Still-Lifes gefunden. Wenn man versucht ein Objekt so zu konstruieren, dass jede lebende Zelle drei Nachbarn hat und keine tote Zelle zwei, ergibt sich am Rand des Objekts

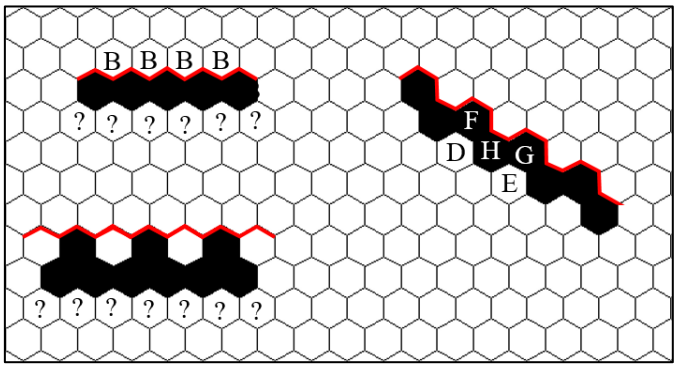


Abb. 18: Versuche der Konstruktion von Still-Lifes

ein Problem, welches unlösbar erscheint. Konstruiert man einen waagrechten Rand wie oben links, werden im nächsten Zeitschritt die mit B gekennzeichneten Zellen lebendig. Versucht man dieses Problem zu lösen, indem man bei der Randregion nur jede zweite Zelle auswählt (unten links), tritt das Problem auf, dass die oberen Zellen nicht mehr als zwei Nachbarn haben können und im nächsten Zeitschritt tot sein werden. Bei einem diagonalen Rand (rechts) müsste die Zelle D lebendig sein, damit F drei Nachbarn hat und demzufolge müsste auch E lebendig sein, damit G überlebt. Dann würde jedoch die Zelle H dazwischen vier Nachbarn haben und deswegen im nächsten Zeitschritt sterben.

Ausserdem werden beim Versuch, die entdeckten Oszillatoren zu kombinieren und zu verändern, bereits bekannte Oszillatoren gefunden. Dies ist nicht weiter erstaunlich, da man in den oben abgebildeten Grafiken gut erkennen kann, dass es sich oftmals um Zusammensetzungen von «Blinkern» handelt. Bei der Veränderung des Oszillators mit Periode 3 entsteht jedoch ein neuer und grösserer als die bisherigen Oszillatoren mit Periode 15. In der nachfolgenden Grafik (Abb. 19) kann man oben links den manipulierten Periode-3-Oszillator sehen, dessen Zustand jedoch anschliessend nicht nochmals erreicht wird. Zudem wurden in kleineren Feldern mit der Grösse 12×12 drei weitere Oszillatoren gefunden (Vergleich Abb. 20). Dabei wurde jeweils wie zuvor mit einer zufälligen Startkonfiguration gestartet. Zwei der gefundenen Oszillatoren besitzen die Periode 4, wobei jeweils der Zustand zum Zeitpunkt t achsensymmetrisch zum Zustand des Zeitpunkts t + 2 ist. Ausserdem sind die Zustände des Oszillators zum Zeitpunkt t mit Periode 12 drehsymmetrisch zum Zustand zum Zeitpunkt t + 4. Durch die Modifikation eines Oszillators mit Periode 4 wurde zudem ein weiterer kleinerer Oszillator ebenfalls mit Periode 4 gefunden.

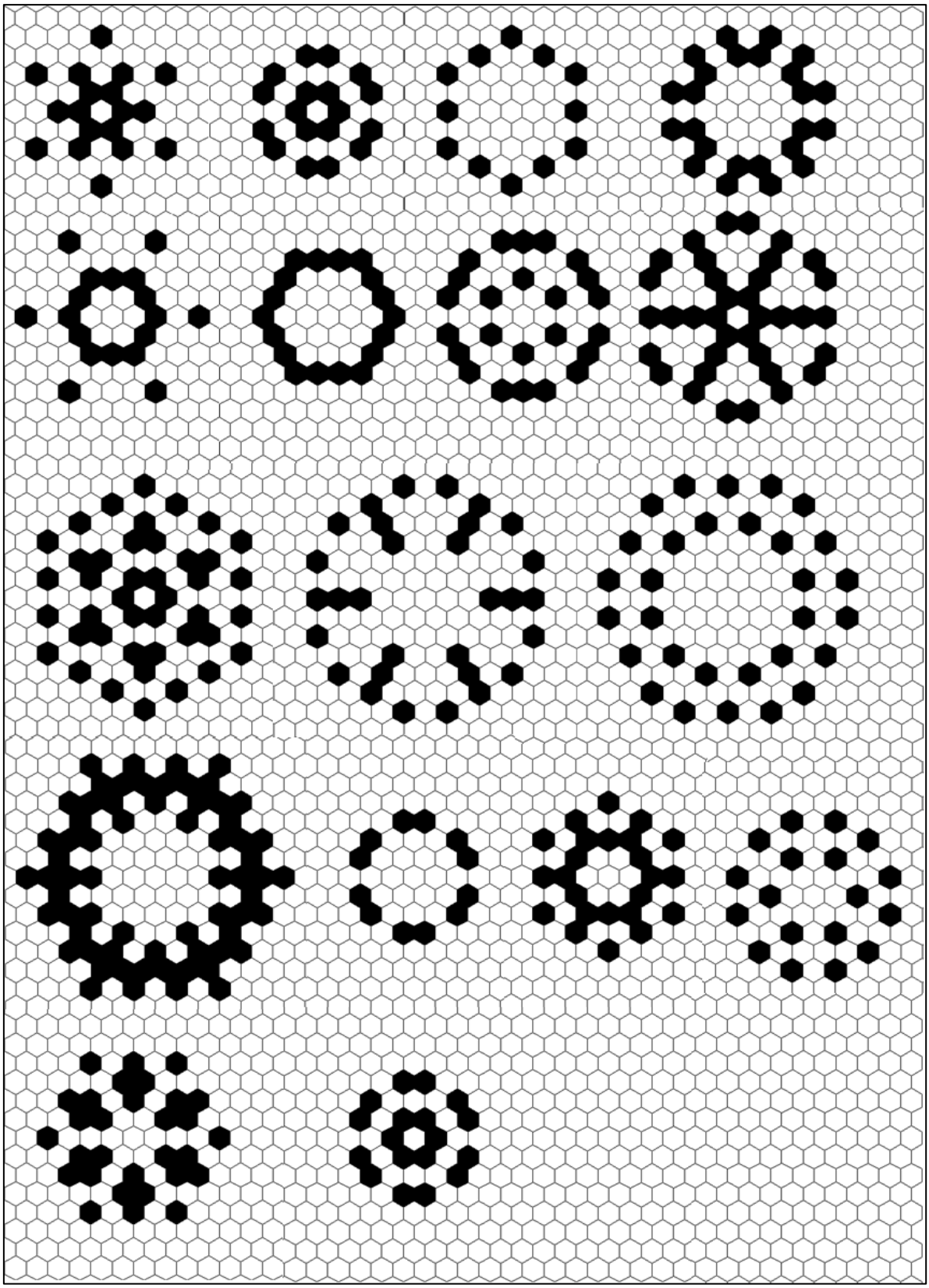
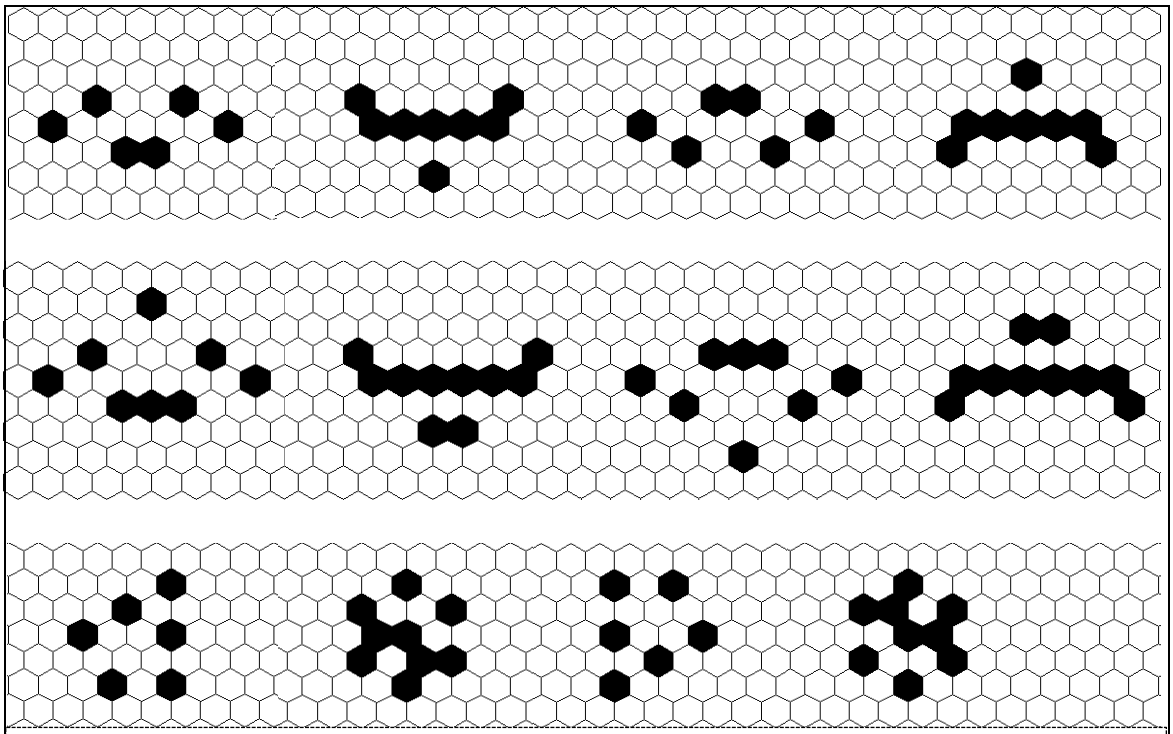
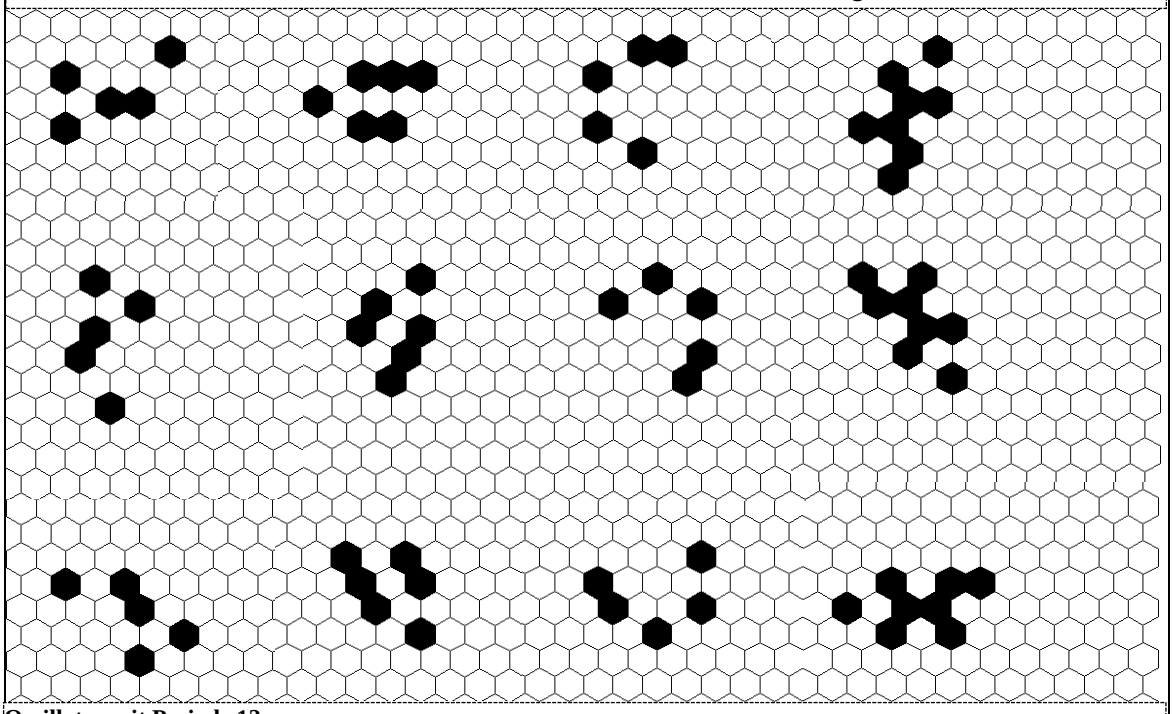


Abb. 19: Entwicklung des gefundenen Oszillators mit Periode 15



Oszillatoren mit Periode 4, wobei der obere durch eine Modifikation des mittleren gefunden wurde



Oszillator mit Periode 12

Abb. 20: Weitere gefundene Oszillatoren

4.2 Wachsende Strukturen

Unter wachsenden Strukturen oder der Entwicklung aus Keimen (engl. seeds) versteht man die Entwicklung aus kleinen Mustern, oftmals nur einer lebendigen Zelle, in einem sonst leeren Zellularfeld. Diese Strukturen erinnern zudem oft an Schneeflocken oder auch Ornamente.

Dafür schauen wir uns zelluläre Automaten in einem dreieckigen Mosaikgitter mit einer Von-Neumann-Nachbarschaft (3 Nachbarn) an. Wenn das Muster wachsen soll, dann muss in der Regel B1 enthalten sein, da Objekte von Regeln ohne B1 und mit B2 und/oder B3 nicht ihre Grenzboxen verlassen können. Regeln mit B1 sind ausserdem unsterblich, womit unter diesen Automaten wahrscheinlich keine Life-ähnlichen zu finden sind.

Regel 0,1,2,3/1,2

Betrachtet man für die Entwicklung aus einer lebenden Zelle als einfaches Beispiel die Regel 0,1,2,3/1,2 kommt die Frage auf, ob sich in diesem Fall die Anzahl lebender Zellen zu einem bestimmten Zeitpunkt t voraussagen lässt.

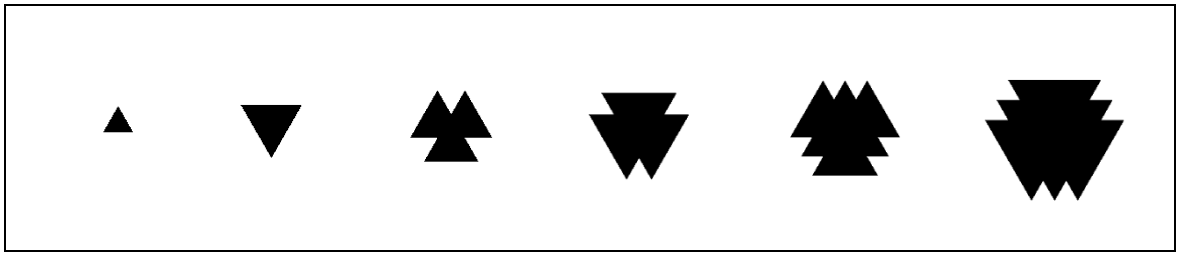


Abb. 21: Wachstum der Regel 0,1,2,3/1,2

Schreibt man die Anzahl lebender Zellen p in einer Exceltabelle auf und untersucht die Differenz zweier Generationen, so erhält man die Reihe 3, 6, 9, 12, 15, 18…

Vermutung:

$$p_t = 1 + \sum_{i=0}^{t} 3 \cdot i = 1 + 3 \cdot \sum_{i=0}^{t} i = 1 + 3 \cdot \frac{1}{2} t(t+1) = 1 + \frac{3}{2} t(t+1)$$

Wobei $p \in \mathbb{N}$ der Anzahl lebendiger Zellen, der Population, entspricht und $t \in \mathbb{N}_0$ dem Zeitpunkt, also der Generation. Es wird mit dem Zeitpunkt t = 0 gestartet.

Beweis:

Zu zeigen: Es kommen jeweils vom Zeitpunkt t – 1 zu t jeweils 3t Zellen dazu.

Betrachtet man das Muster als horizontale Zeilen von Zellen, so kann man sehen, dass von einem Zeitpunkt t – 1 zu t jeweils jede Zeile um zwei Zellen erweitert wird, also an der Seite links t sowie rechts t Zellen dazukommen. Somit kommen total jeweils 2t Zellen dazu. Weiters entsteht jeweils abwechselnd oben und unten eine neue Reihe, bei der nur jede zweite Zelle lebendig ist. Die Lücken werden anschliessend im nächsten Zeitschritt aufgefüllt. Für die Beschreibung der neu entstehenden Reihen wird der zu Null hin gerundete Quotient div verwendet, welcher einer ganzzahligen Division entspricht, die immer abgerundet wird (a div b = (a – (a mod b)) / b). [30] Dabei kommt bei jedem Zeitpunkt t jeweils oben oder unten $(t + 2)\,\mathrm{div}\,2$ Zellen dazu, während die zu Zeitpunkt t-1 entstandene Reihe mit $(t - 1)\mathrm{div}\,2$ Zellen aufgefüllt wird. Somit kommen insgesamt bei jedem Zeitschritt x Zellen dazu.

$$x = 2t + (t+2)\text{div } 2 + (t-1)\text{ div } 2$$

Fallunterscheidung:

1.Fall: t ist gerade (t mod 2 = 0)

$$(t-1)\text{div } 2 = \frac{1}{2}(t-1) - \frac{1}{2}\big((t-1)\text{mod } 2\big) = \frac{1}{2}(t-1-(t-1)\text{mod } 2) = \frac{1}{2}(t-1-1)$$
$$= \frac{1}{2}t - 1$$

$$(t+2)\text{div } 2 = \frac{1}{2}t + 1 - \frac{1}{2}\big((t+2)\text{mod } 2\big) = \frac{1}{2}t + 1 - 0 = \frac{1}{2}t + 1$$

$$(t+2)\text{div } 2 + (t-1)\text{div } 2 = \frac{1}{2}t - 1 + \frac{1}{2}t + 1 = t$$

2. Fall: t ist ungerade (t mod 2 = 1)

$$(t+2)\text{div } 2 = \frac{1}{2}(t+2) - \frac{1}{2}\big((t+2)\text{mod } 2\big) = \frac{1}{2}(t+2-(t+2)\text{mod } 2) = \frac{1}{2}(t+2-1)$$
$$= \frac{1}{2}(t+1)$$

$$(t-1)\text{div } 2 = \frac{1}{2}\Big(t-1-\big((t-1)\text{mod } 2\big)\Big) = \frac{1}{2}(t-1)$$

$$(t+2)\text{div } 2 + (t-1)\text{div } 2 = \frac{1}{2}(t+1) + \frac{1}{2}(t-1) = t$$

Womit in beiden Fällen gilt:

$(t+1)\text{div } 2 + t \text{ div } 2 = t$

$=> x = 3t$ q.e.d.

Regel /1,2,3

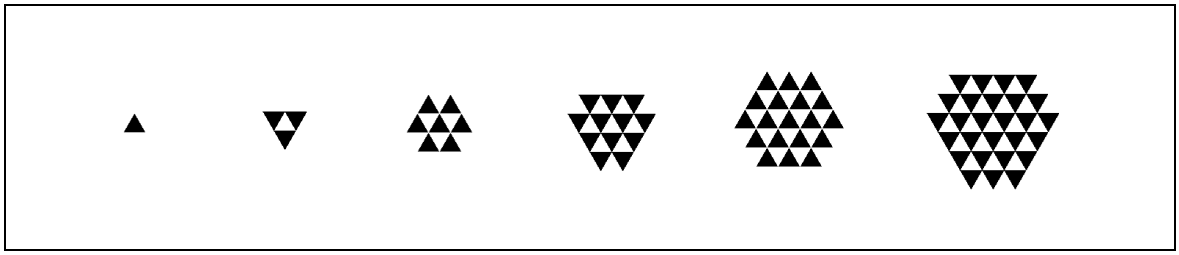


Abb. 22: Wachstum der Regel /1,2,3

Die Berechnung der Population der Regel /1,2,3 (gilt für alle 8 Regeln, die B1, B2 und B3 enthalten sowie nicht S0 und B0) ist ziemlich ähnlich. Es kann wieder in Reihen gedacht werden, wobei jeweils, da Zellen innerhalb des Musters abwechselnd lebendig und tot sind, nur eine anstelle von zwei Zellen pro Reihe dazukommt. Ausserdem wird die Reihe, welche im vorherigen Zeitschritt entsteht, dadurch auch nicht mehr aufgefüllt.

$$p_t = \sum_{i=0}^{t} i + \sum_{i=0}^{t} (i+2) \operatorname{div} 2 = \frac{1}{2}t(t+1) + \sum_{i=0}^{t} \frac{1}{2}i + 1 - \frac{1}{2}(i \bmod 2)$$

$$= \frac{1}{2}t(t+1) + \frac{1}{2}\sum_{i=0}^{t} i + 2 - i \bmod 2$$

$$= \frac{1}{2}\left(t(t+1) + \sum_{i=0}^{t} i + \sum_{i=0}^{t} 2 - \sum_{i=0}^{t} i \bmod 2\right)$$

Um weiterfahren zu können, muss hier zuerst die Summe von $i \, mod \, 2$ ohne Summenzeichen geschrieben werden. Um die Folge $a_n = n \, mod \, 2$ ohne Modulo zu schreiben, braucht man eine alternierende Folge. Hier bietet sich $a_n = (-1)^n$ an, wobei man jedoch möchte, dass jedes zweite Folgenglied 0 und nicht -1 ist. Dies erreicht man durch die Addition von 1, wodurch man die Folge (0, 2, 0, 2, 0, ...) erhält. Teilt man hier die Folgenglieder noch durch zwei, bekommt man das gleiche wie mit $a_n = n \, mod \, 2$.

$$\sum_{i=0}^{t} i \bmod 2 = \sum_{i=0}^{t} \frac{1}{2}((-1)^{i+1} + 1) = \frac{1}{2}(t+1) + \frac{1}{2}\sum_{i=0}^{t} (-1)^{i+1}$$

$$= \frac{1}{2}(t+1) + \frac{1}{2}\left(\frac{1}{2}((-1)^{t+1} - 1)\right) = \frac{1}{2}\left(\frac{1}{2}(-1)^{t+1} + t + 1 - \frac{1}{2}\right)$$

$$= \frac{1}{4}((-1)^{t+1} + 2t + 1)$$

Einsetzen:

$$\frac{1}{2}\left(t(t+1) + \sum_{i=0}^{t} i + \sum_{i=0}^{t} 2 - \sum_{i=0}^{t} i \bmod 2\right)$$

$$= \frac{1}{2}\left(t(t+1) + \frac{1}{2}t(t+1) + 2t + 2 - \left(\frac{1}{4}((-1)^{t+1} + 2t + 1)\right)\right)$$

$$= \frac{1}{2}\left(t^2 + t + \frac{1}{2}t^2 + \frac{1}{2}t + 2t + 2 - \frac{1}{4}(-1)^{t+1} - \frac{1}{2}t - \frac{1}{4}\right)$$

$$= \frac{1}{2}\left(\frac{3}{2}t^2 + 3t - \frac{1}{4}(-1)^{t+1} + \frac{7}{4}\right) = \frac{1}{8}(6t^2 - (-1)^{t+1} + 12t + 7)$$

$$= \frac{1}{8}(6t^2 + (-1)^t + 12t + 7)$$

Regel 0,1,2/1,2

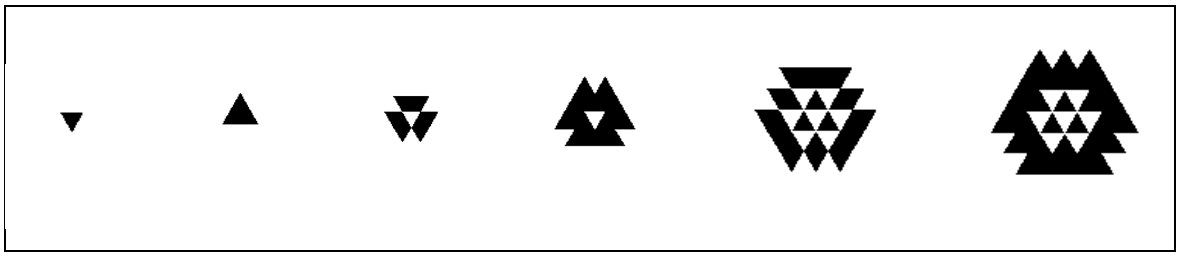


Abb. 23: Wachstum der Regel 0,1,2/1,2

Vermutung:

Untersucht man das Wachstum der Regel 0,1,2/1,2, so erhält man die Reihe 1, 4, 9, 18, 24, 39, 45, 66, 72, 99, 105, 138, 144, 183, 189, …, welcher die Folge 1, 3, 5, 9, **6**, 15, **6**, 21, **6**, 27, **6**, 33, **6**, 39,

6, … zugrunde liegt. Dabei fällt auf, dass jedes zweite Folgenglied ab dem Zeitpunkt t = 3 eine 6 ist und die restlichen Folgenglieder ein Vielfaches von 3. Dadurch kann man die Folge als Zusammensetzung zweier einzelnen Folgen betrachten, wobei man abwechselnd ein Glied der ersten und der zweiten Folge nimmt. Somit kann man die Reihe ab dem Zeitpunkt t = 3 als Addition der Folgen $a_t = 6((t+1) \bmod 2)$ und $b_t = 3(t-1)(t \bmod 2) + 3(t \bmod 2)$ auffassen.

Beweis:

Zu zeigen: Die Anzahl dazukommender Zellen können durch die Folgen $a_t = 6((t+1) mod\ 2)$ und $b_t = 3(t-1)(t\ mod\ 2) + 3(t mod 2)$ ab dem Zeitpunkt t = 3 beschrieben werden.

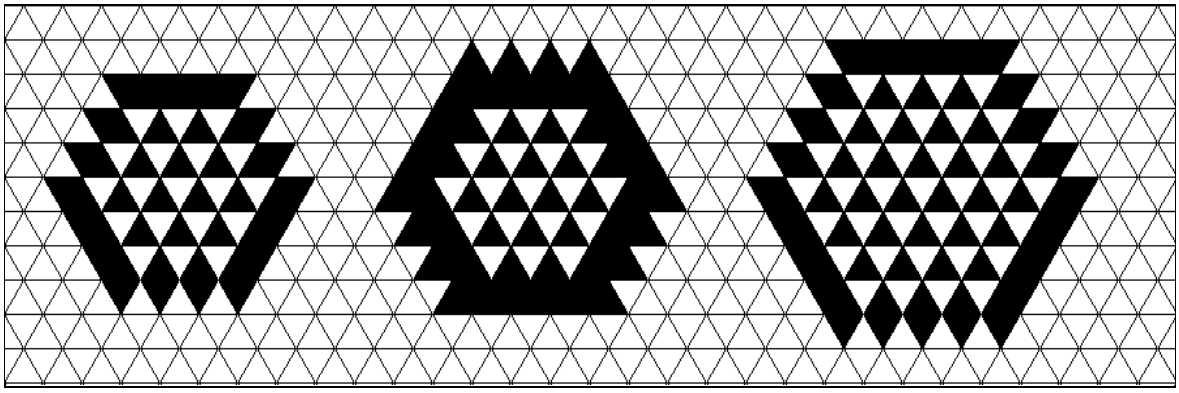


Abb. 24: Wachstum der Regel 0,1,2/1,2 von den Zeitpunkten t = 6 bis t = 8

Von einem geraden Zeitpunkt t - 1 zu t kommen auf der rechten sowie der linken Seite jeweils t Zellen, also insgesamt 2t Zellen dazu. Zudem wird eine neue Reihe mit $(t+1) \cdot \frac{1}{2}$ Zellen begonnen, und die zum Zeitpunkt t – 1 entstandene Reihe mit $(t-1) \cdot \frac{1}{2}$ Zellen aufgefüllt. Es kommen total x Zellen dazu:

$$x = 2t + \frac{1}{2}(t+1) + \frac{1}{2}(t-1) = 2t + \frac{1}{2}t + \frac{1}{2} + \frac{1}{2}t - \frac{1}{2} = 3t = 3(t-1) + 3$$

$$\rightarrow b_t = 3(t-1)(t \bmod 2) + 3(t \bmod 2)$$

Von einem ungeraden Zeitpunkt t - 1 zu t kommen links und rechts wieder total 2t Zellen dazu. Jedoch gehen innerhalb der Struktur links und rechts $2(t-1)$ Zellen verloren. Zudem sterben unten und oben $2\left(\frac{1}{2}t - 2\right)$ Zellen. Ausserdem entsteht zudem eine neue Reihe mit $\frac{1}{2}(t+2)$ Zellen und eine Reihe wird mit $\frac{1}{2}(t-2)$ Zellen aufgefüllt. Somit kommen insgesamt x Zellen dazu:

$$x = 2t + \frac{1}{2}(t+2) + \frac{1}{2}(t-2) - 2(t-1) - 2\left(\frac{1}{2}t - 2\right)$$

$$= 2t + \frac{1}{2}t + 1 + \frac{1}{2}t - 1 - 2t + 2 - t + 4 = 6$$

$$\rightarrow a_t = 6((t+1) \bmod 2)$$

q.e.d.

Nun muss die Reihe der addierten Folge gebildet werden, welche dann die Anzahl lebender Zellen zum Zeitpunkt t beschreibt. Dafür werden zuerst die Reihen der beiden Folgen einzeln gebildet und anschliessend addiert.

Reihe von a_t:

Die Reihe c_t der Folge a_t ist (0, 6, 6, 12, 12 18, 18, 24, …). Sie kann mit der Folge $f_t = 3t$ angenähert werden, wobei jeweils bei jedem zweiten Folgenglied noch 3 subtrahiert werden muss. Dies kann man mithilfe der Modulo-2-Folge tun.

$$c_t = 3t - 3(t \bmod 2) = 3t - \left(\frac{3}{2}((-1)^{t+1} + 1)\right) = 3\left(t - \frac{1}{2}(-1)^{t+1} - \frac{1}{2}\right)$$

Reihe von b_t:

Teilt man die Folge $g_t = 6t$ durch 6, so erhält man die Folge $h_t = t$, deren Reihe $k_t = \frac{1}{2}t(t+1)$ ist. Ein Glied der Reihe der Folge g_t sieht folgendermassen aus: $l_t = 6 \cdot 1 + 6 \cdot 2 + 6 \cdot 3 + \cdots + 6(t-1) + 6 \cdot t = 6(1 + 2 + 3 + \cdots + (t-1) + t) = 6 \cdot k_t$ Wenn man die Reihe der Folge b_t sucht, so muss diese wie l_t aussehen mit dem Unterschied, dass jedes Folgenglied zweimal hintereinander vorkommt, so dass man die Folge (6, 6, 18, 18, 36, 36, 60, …) erhält. Diese Reihe lässt sich mit $m_t = \frac{3}{2} \cdot t\left(\frac{t}{2} + 1\right)$ annähern. Hier muss jedoch bei jedem zweiten Glied noch etwas dazu addiert werden, um zweimal nacheinander auf das Gleiche zu kommen. Die Differenz beträgt dabei:

$$\begin{aligned} d_t &= \frac{3}{2} \cdot (t+1)\left(\frac{t+1}{2} + 1\right) - \frac{3}{2} \cdot t\left(\frac{t}{2} + 1\right) = \frac{3}{4}\big((t+1)(t+3) - t(t+2)\big) \\ &= \frac{3}{4}(t^2 + 4t + 3 - t^2 - 2t) = \frac{3}{4}(2t+3) \end{aligned}$$

Diese Differenz muss nur bei jedem zweiten Glied dazu addiert werden. Hierzu kann man die Differenz-Folge mit der Modulo-2-Folge multiplizieren.

$$d_t = \frac{3}{4}(2t+3)(t \bmod 2) = \frac{3}{8}(2t+3)((-1)^{t+1} + 1)$$

Zudem muss noch der Teil mit $3(t \bmod 2)$ beachtet werden, wodurch man zusätzlich die Folge $n_t = \frac{3}{2}t + \frac{3}{2}\left(\frac{1}{2}((-1)^{t+1} + 1\right)$ addieren muss. Somit ergibt sich die Reihe s_t zur Folge b_t.

$$\begin{aligned} s_t &= \frac{3}{4}t(t+2) + \frac{3}{2}t + \frac{3}{2}\left(\frac{1}{2}((-1)^{t+1} + 1\right) + \frac{3}{8}(2t+3)((-1)^{t+1} + 1) \\ &= \frac{3}{8}(2t^2 + 4t + 4t + 2(-1)^{t+1} + 2 + 2t(-1)^{t+1} + 3(-1)^{t+1} + 2t + 3) \\ &= \frac{3}{8}(2t^2 + 10t + 2t(-1)^{t+1} + 5(-1)^{t+1} + 5) \\ &= \frac{3}{8}(2t^2 + 10t + (-1)^{t+1}(2t+5) + 5) \end{aligned}$$

Nun lassen sich die Folgen s_t sowie c_t und der Startwert von 9 addieren. Dabei muss t durch t - 2 ersetzt werden, da die beiden Folgen erst ab dem Zeitpunkt t = 3 dazukommen. Die Schlussformel ist dann ab dem Zeitpunkt t = 2 gültig, wobei mit t = 0 begonnen wird, demzufolge zum Zeitpunkt t = 0 demzufolge der Startzustand herrscht.

$$p_t = 9 + 3\left((t-2) - \frac{1}{2}(-1)^{t-1} - \frac{1}{2}\right)$$
$$+ \frac{3}{8}(2(t-2)^2 + 10(t-2) + (-1)^{t-1}(2(t-2)+5) + 5)$$
$$= \frac{3}{8}(24 + 8t - 16 - 4(-1)^{t-1} - 4 + 2t^2 - 8t + 8 + 10t - 20 + 2t(-1)^{t-1}$$
$$+ (-1)^{t-1} + 5) = \frac{3}{8}(2t^2 + 10t + (-1)^{t-1}(2t-3) - 3)$$
$$= \frac{3}{8}(2t(t+5) + (-1)^{t-1}(2t-3) - 3)$$

Weitere Regeln

Bei vielen wachsenden Strukturen lässt sich jedoch nicht mehr so leicht eine Formel für die Population zu einem bestimmten Zeitpunkt finden. Dies liegt auch teilweise daran, dass es oftmals von einem Zeitschritt zum nächsten auch zu einer Abnahme der Anzahl Zellen kommt. Die Abbildungen zu den folgenden Regeln sind aus Platzgründen auf den nächsten Seiten zu finden.

Regel 0,1,2,3/1

Bei dieser Regel bleibt jede Zelle, welche einmal lebendig geworden ist, für immer lebendig. Dabei bildet sich von Zeit zu Zeit ein gezackter Rand, von dem aus kein Wachstum mehr möglich ist (Vergleich Abb. 25). Eine Ausnahme bilden hier jeweils die 6 Ecken, von denen die Struktur dann weiterwächst. Der gezackte Rand scheint sich jeweils zu den Zeitpunkten $2^n - 2$ und $2^n - 1$ zu bilden, wobei $n \in \mathbb{N}$. Da das Wachstum von den Ecken ausgeht, wird jeweils 2/3 eines Sechsecks an die Ecken des bereits bestehenden Randes hinzugefügt, womit man einen Rand bekommt, welcher an Fraktale erinnert. Es lässt sich zudem tatsächlich eine Struktur (blau eingezeichnet) finden, welche sich so wiederholt, dass sich ein Sierpinski-Dreieck ergibt. Die Teile, welche nicht zum Sierpinski-Dreieck gehören, wurden mit grün markiert und weisen ebenfalls eine fraktale Struktur auf. Dabei ist der oberste Teil bei allen identisch, sprich es handelt sich immer um das gleiche Dreieck, welches auf verschiedenen Höhen abgeschnitten wurde.

Regel 0,2,3/1

Diese wachsende Struktur besitzt einige Ähnlichkeit zur Regel 0,1,2,3/1. Es bildet sich auch zu den Zeitpunkten $2^n - 2$ und $2^n - 1$ ein Rand, von dem aus Wachstum nicht mehr möglich ist. Hier bleibt der Rand jedoch nicht in diesem Zustand erhalten, sondern fällt ein Stück weit weg und wächst anschliessend von dort an weiter. Zudem ist auch hier ein Wachstum, welches von den Ecken ausgeht, gut erkennbar (Vergleich Abb. 26). Macht man einen Querschnitt durch die mittlere horizontale Reihe der Regel 0,2,3/1, so lassen sich Muster entlang des Randes erkennen, welche sich in regelmässigen Abständen wiederholen. Einige davon wurden im Bild mit Farbe gekennzeichnet (Vergleich Abb. 27). In der Mitte sind jedoch Muster, welche sich mit gewissen Regelmässigkeiten wiederholen, gar nicht oder nur schwer vorzufinden. Interessant ist bei diesem Automaten zudem, dass nur bei jedem zweiten Zeitschritt Zellen vorzufinden sind, welche direkt nebeneinanderliegen. Bei den Zeitpunkten mit einem geraden t hat jeweils keine lebendige Zelle einen oder mehrere lebendige Nachbarn. Wenn eine neue Zelle geboren wird und überleben will, so darf sie nicht genau einen Nachbarn haben. Damit dies der Fall ist, muss entweder die Zelle, welche für die Geburt verantwortlich war, sterben, also genau einen Nachbarn haben, oder mindestens ein Nachbar der neugeborenen Zelle ebenfalls zum Leben erweckt werden. Dies scheint jedoch nicht möglich zu sein.

Das kann man damit begründen, dass jeweils bei den geraden Zeitpunkten in einer horizontalen Reihe alle lebendigen Zellen gleich ausgerichtet sind, sprich immer mit der Spitz nach oben zeigen.

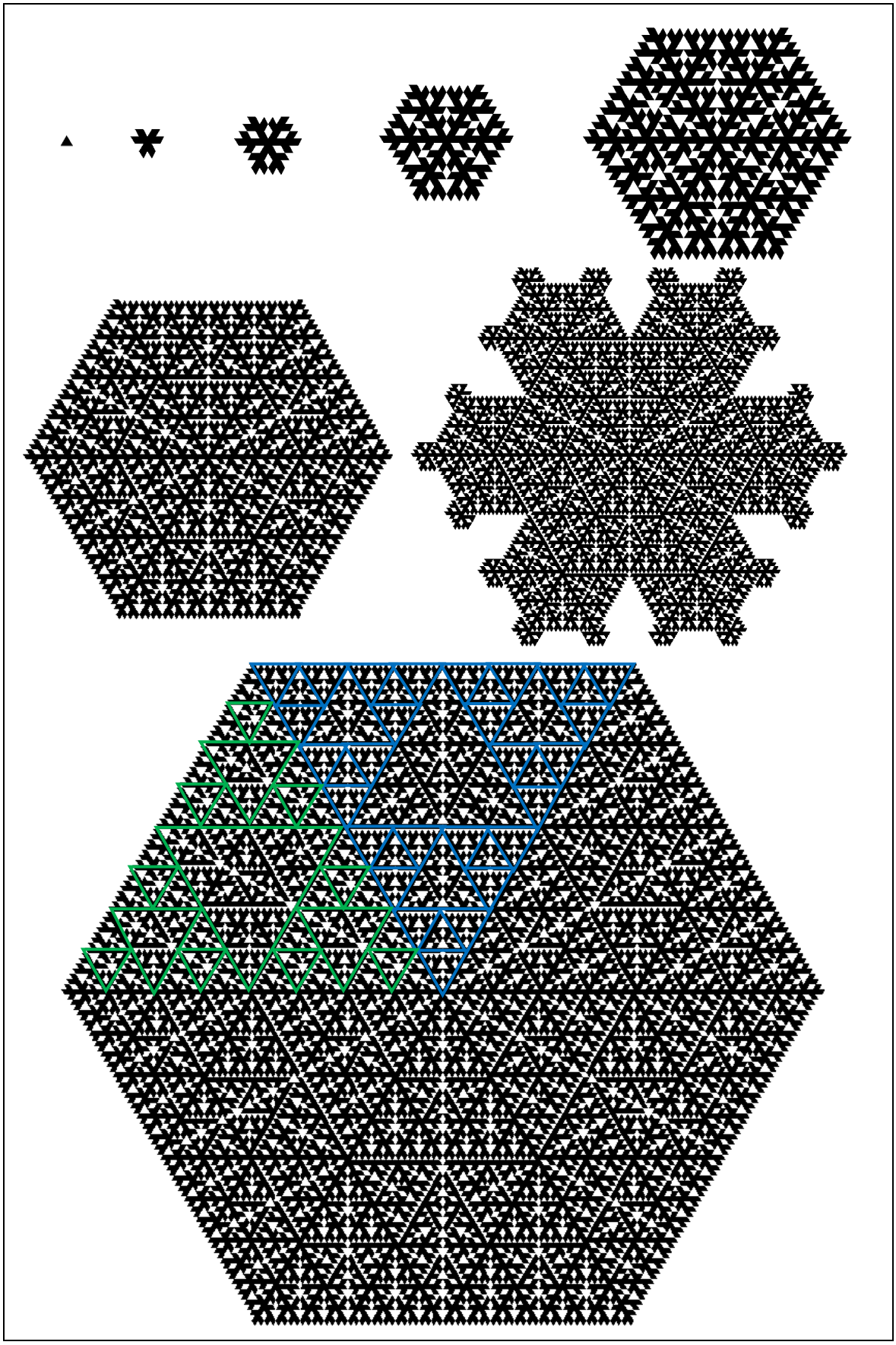


Abb. 25: Wachstum der Regel 0,1,2,3/1 zu den Zeitpunkten t = 0, t = 3, t = 7, t = 15, t = 31, t = 63, t = 103 und t = 127

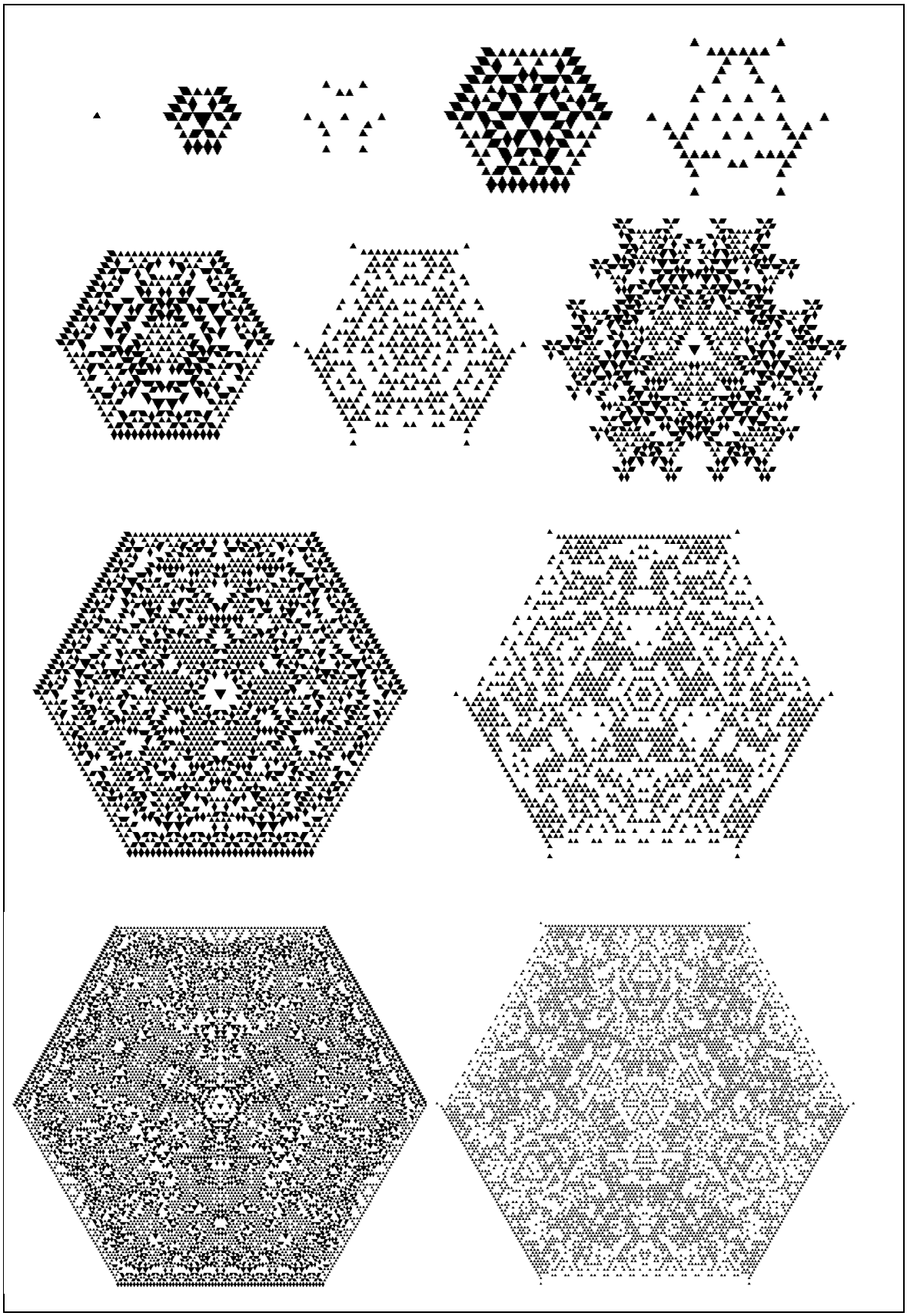


Abb. 26: Wachstum der Regel 0,2,3/1 zu den Zeitpunkten t = 0, t = 7, t = 8, t = 15, t = 16, t = 31, t = 32, t = 51, t = 63, t = 64, t = 127 und t = 128

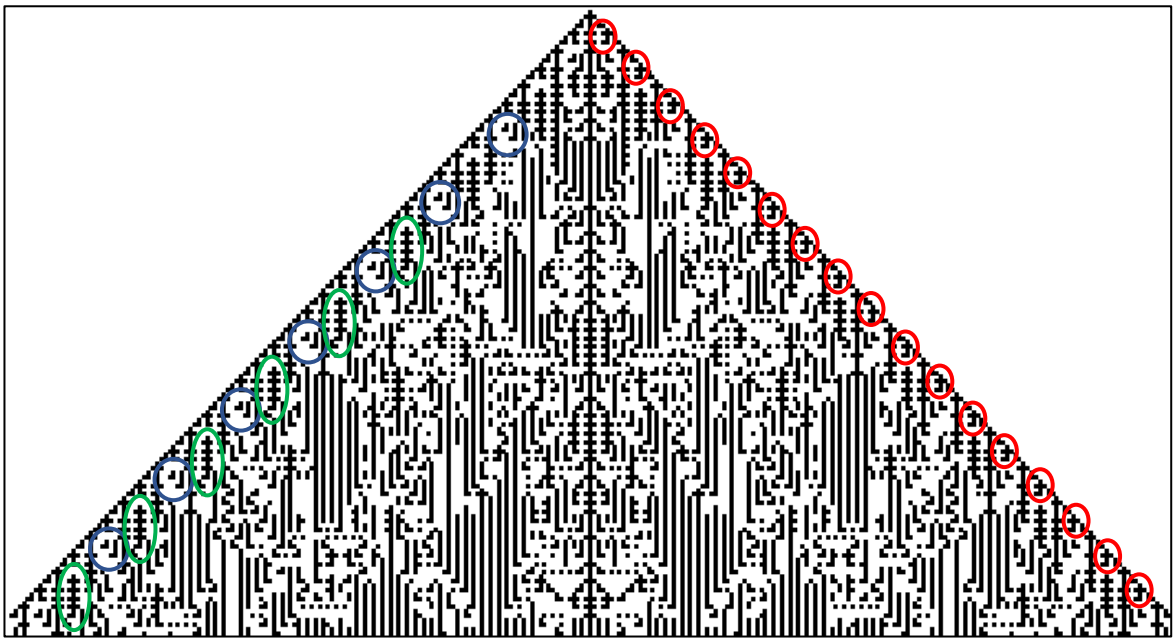


Abb. 27: Querschnitt durch die mittlere horizontale Reihe der Regel 0,2,3/1

Gemeinsamkeiten

Es scheint so, als würde es bei jeder wachsenden Struktur, bei der mit einer Zelle gestartet wird, jeweils 3 Diagonalen geben, an deren Enden im nächsten Zeitschritt immer eine neue Zelle entsteht. Schaut man beispielsweise eine Diagonale (blau eingezeichnet) stellvertretend für alle Diagonalen an, so lässt sich festhalten, dass eine Zelle in jedem Fall zum Leben erweckt wird, wenn sie genau einen Nachbarn hat, da alle wachsende Strukturen in der Regel B1 (siehe S. 8) enthalten müssen. Um dies zu verhindern, müsste eine Zelle auf der Diagonalen 2 oder 3 lebendige Nachbarn haben, wenn sie tot ist. In der Grafik sind die zwei Wege mit rot und grün eingezeichnet, welche eine der zwei anderen der insgesamt drei Nachbarn zu Beginn für ihre Route wählen. Die Zahlen geben an, wie viele Zeitschritte benötigt werden, damit eine Zelle lebendig wird. Eine Zelle der Diagonalen wird jeweils zum Zeitpunkt t lebendig, wenn sie einen Nachbarn auf der Diagonalen hat, welcher zum Zeitpunkt t – 1 lebendig geworden ist. Damit sie nicht lebendig wird, bräuchte sie einen Nachbarn, welcher nicht auf der Diagonalen liegt und somit spätestens zum Zeitpunkt t – 1 lebendig wurde. Auf der Grafik ist jedoch ersichtlich, dass dies nicht möglich ist. Somit wird bei jeder wachsenden Struktur an den sechs Enden der drei Diagonalen immer eine Zelle dazukommen. Dies ist auch der Fall, wenn die Zellen der Diagonalen im Laufe der Zeit wieder sterben. Was danach auf den Diagonalen passiert, lässt sich hiermit jedoch nicht sagen.

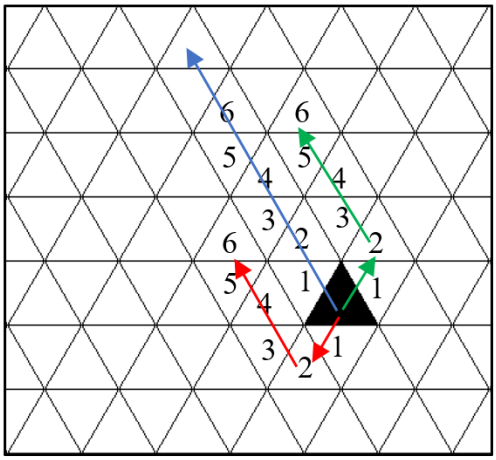


Abb. 28: Wachstum entlang den Diagonalen

Mithilfe dieser Überlegungen kann man nun eine Erklärung für das von den Ecken ausgehende Wachstum finden.

Vermutung: Der gezackte Rand, von dem aus kein Wachstum möglich ist, tritt jeweils zu den Zeitpunkten auf, welche durch sich die Folgen $a_n = 2^n - 2$ und $b_n = 2^n - 1$ beschreiben lassen.

Die hier gemachten Überlegungen werden für den gezackten Rand, welcher zu den Zeitpunkten $a_n = 2^n - 2$ entsteht gemacht und können analog für $b_n = 2^n - 1$ durchgedacht werden. Ein gezackter Rand entsteht bei allen wachsenden Strukturen, deren Regel nicht B2 enthalten, was ermöglichen würde, über die gezackte Randregion hinaus zu wachsen. Bei jedem dieser Automaten findet ein Wachstum entlang der blauen Diagonalen statt. Der gezackte Rand wächst jeweils von diesen

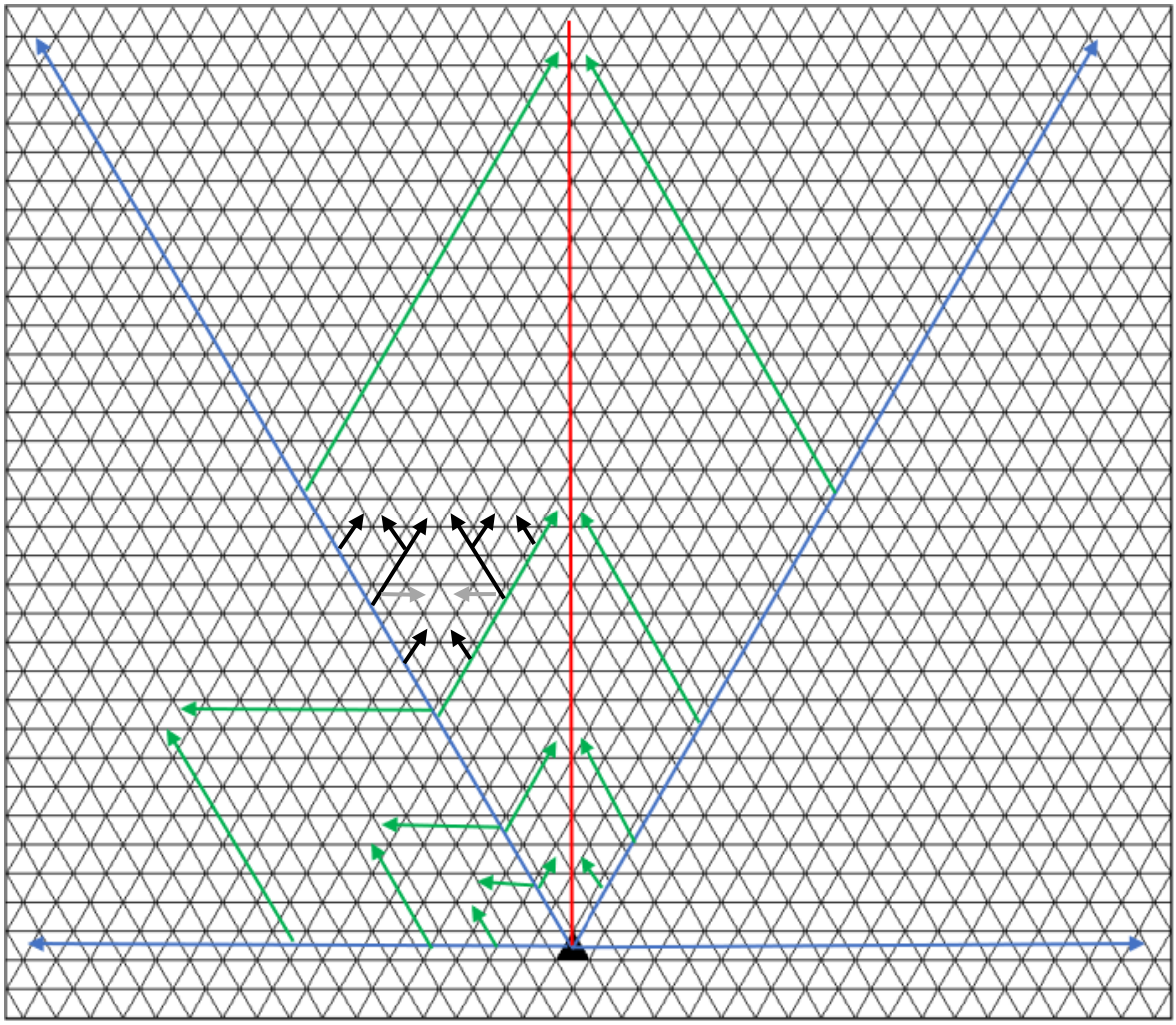


Abb. 29: Entstehung des gezackten Randes von den Diagonalen aus

Diagonalen aus, wobei aus Symmetriegründen von einer Diagonalen aus ein Wachstum bis zur roten Mittellinie stattfinden muss, damit die Randregion entsteht. Dies wurde mit den grünen Pfeilen eingezeichnet. Ein Wachstum entlang der grünen Pfeile findet immer statt, da der gezackte Rand verhindert, dass ein kürzerer Weg zum nächsten gezackten Rand möglich ist. Zwischen der Mittelline und der Diagonalen findet dabei ein Wachstum nach dem gleichen Prinzip statt (schwarz eingezeichnet) und bildet den restlichen Teil des Randes. Somit lässt sich auch erklären, warum sich bei der Regel 1,2,3/1 ein Sierpinski-Dreieck finden lässt. Die blaue Diagonale hat jeweils einen Abstand von $(t-2)\mathrm{div}2$ zur Mittellinie, da die Zellen entlang der Diagonalen abwechselnd jeweils eine Zelle nach links oder rechts und oben oder unten wachsen. Somit wird der Abstand zur Mittellinie alle zwei Zeitschritte um eine Zelle grösser. Um nun den Rand zu bilden, muss ein Wachstum von der Diagonalen zur Mittellinie stattfinden. Da sich das Wachstum entlang der grünen

Pfeile ebenfalls diagonal fortbewegt, braucht man zum Überwinden eines Abstands der Länge l jeweils $2l$ Zeitschritte. Dabei kann man bereits der Zustand zum Zeitpunkt t = 0 als ein gezackter Rand betrachten. Nachdem der gezackte Rand gebildet wurde, muss die Diagonale zwei Zeitschritte lang weiterwachsen, damit erneut ein Wachstum in Richtung Mittellinie stattfinden kann. Für das Wachstum bis zur Mittellinie werden $2((t-2)\ \mathrm{div}2)$ Zeitschritte benötigt. Sei t_n der n-te Zeitpunkt, zudem ein gezackter Rand gebildet wird, dann ist d_n die Differenz zwischen dem Zeitpunkt t_n und t_{n-1}.

$$d_n = t_n - t_{n-1} = 2 + 2(t_{n-1}\ \mathrm{div}2)$$

Da 2, $2\big((t-1)\mathrm{div}2\big)$ sowie das erste Glied der Folge durch zwei teilbar sind, so müssen alle Zeitpunkte t_n auch durch zwei ohne Rest teilbar sein. Somit gilt:

$$d_n = 2 + t_{n-1}$$

Die Folge t_n gibt alle Zeitpunkte aus, zu denen ein gezackter Rand gebildet wird:

$$t_1 = 0, \qquad t_n = 2 + t_{n-1} + \sum_{i=1}^{n-1} d_i$$

Um zu zeigen, dass es sich bei der Folge a_n um die gleich wie t_n handelt, wird aufgezeigt, dass der Abstand zwischen den Folgengliedern sowie das erste Folgenglied gleich sind.

$$a_n = 2^n - 2$$

$$a_1 = 0$$

$$a_n - a_{n-1} = 2^n - 2 - (2^{n-1} - 2) = 2^n - 2 - 2^{n-1} + 2 = 2^{n-1}(2-1) = 2^{n-1}$$
$$= (2^{n-1} - 2) + 2 = a_{n-1} + 2$$

q.e.d.

4.3 Modulo-2-Regel

Die Modulo-2-Regel ist ein Kopiersystem, welches bereits aus einem eindimensionalen zellulären Automaten mit zwei Nachbarn sowie aus einem zweidimensionalen mit einer Moore-Nachbarschaft und einem quadratischen Gitter bekannt ist. Ein Kopiersystem vervielfältigt jedes Startmuster innerhalb einer gewissen Anzahl Zeitschritten. Die Modulo-2-Regel besagt, dass eine Zelle zum Zeitpunkt t genau dann lebendig ist, wenn die Anzahl lebendiger Nachbarn inklusive der Zelle selbst ungerade ist. [31] Die Modulo-2-Regel funktioniert jedoch auch bei einem zellulären Automaten mit einem dreieckigen Gitter und einer Von-Neumann-Nachbarschaft. Wenn man dieses System mit einem Smiley testet, so erscheint dieses alle 8 Zeitschritte. Dabei liegen die Smileys jedoch bei jedem zweiten Mal ziemlich nahe beieinander, sodass sie nicht gerade gut erkennbar sind. Zwischen den einzelnen Schritten ist das Muster jedoch nicht erkennbar und der Automat erscheint teilweise völlig chaotisch.

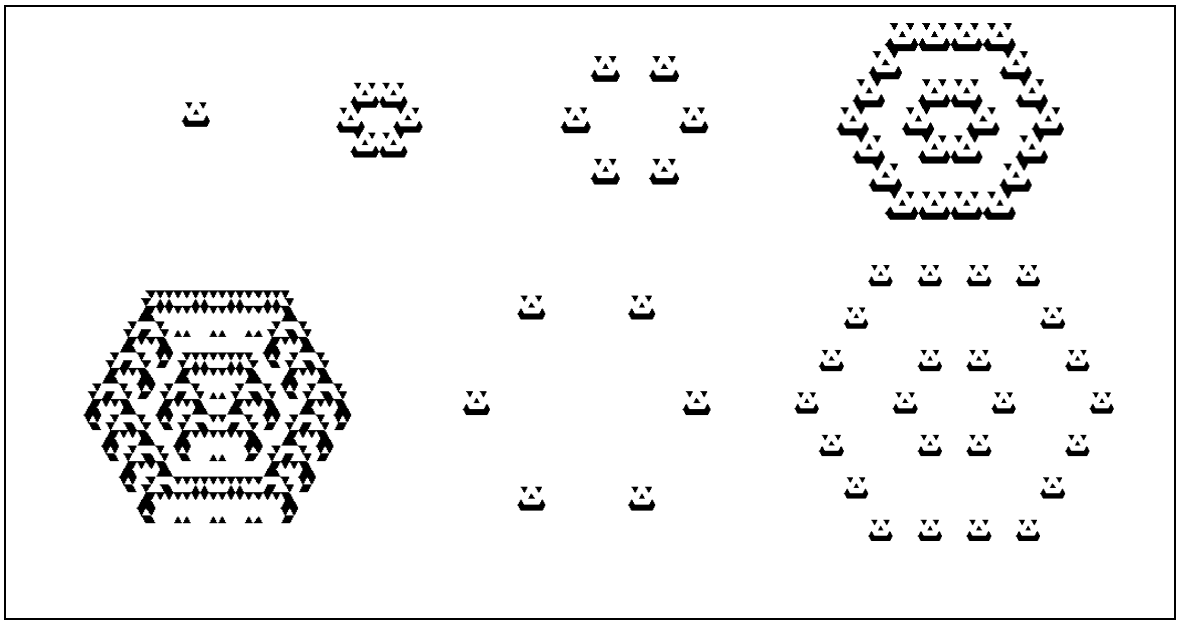


Abb. 30: Kopiersystem mit der Regel 0,2/1,3 zu den Zeitpunkten t = 0, t = 8, t = 16, t = 24, t = 26, t = 32 und t = 40

Wählt man kleinere Muster, so können diese auch in einem Abstand von 4 Zeitschritten kopiert werden. Beginnt man mit nur einer Zelle, so kann man eine Kopie davon alle 2 Schritte sehen. Die Abstände zwischen den Kopien scheinen also von der Grösse des Musters abhängig zu sein.

Die Modulo-2-Regel funktioniert zudem auch bei Automaten mit sechseckigem Gitter und sechs Nachbarn, wobei auch hier die Kopien alle 2^i Schritte erscheinen ($i \in \mathbb{N}$) . Dabei scheint i von der Grösse des kopierten Musters abhängig zu sein. Zudem stellt auch eine abgewandelte Regel ein Kopiersystem dar, die besagt, dass die Zelle genau dann lebendig ist, wenn die Anzahl lebendiger Nachbarn inklusive der Zelle selbst gerade ist.

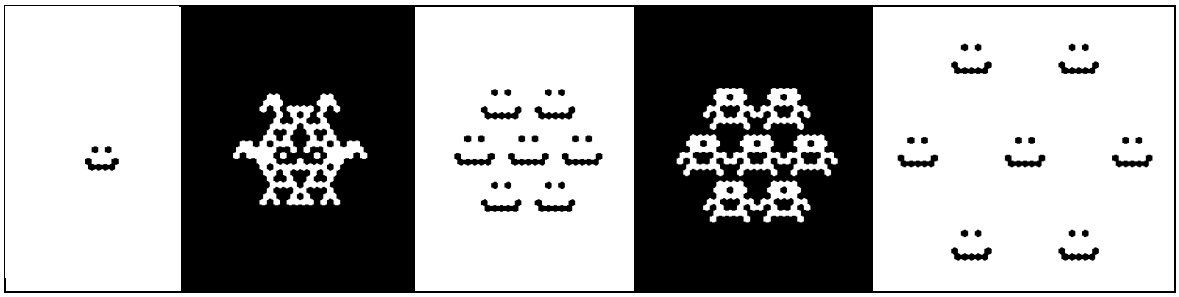


Abb. 31: Kopiersystem mit der Regel 1,3,5/0,2,4,6 zu den Zeitpunkten t = 0, t = 7, t = 8, t = 9 und t = 16

5. Diskussion

Die Programme laufen relativ langsam, sodass sie bei der Simulation von Automaten mit einer grossen Anzahl Zellen schnell ins Stocken geraten. Dies liegt meines Erachtens primär daran, dass der entwickelte Algorithmus eher ineffizient ist. Hier könnte man, wenn man mehr Zeit investieren würde, wahrscheinlich noch einiges verbessern und die Programme mit ein paar Tricks schneller machen. Für die anschliessende mathematische Betrachtung im vorgesehenen Ausmass reichten die Programme jedoch völlig aus. Ein Vorteil bestand darin, dass ich die Programme so gestalten konnte, dass sie sich für die Untersuchungen und Fragen, welche ich beantworten wollte, gut eignen. Hier wäre zu dem zufälligen Verteilen der lebenden Zellen sowie dem Speichern von Zuständen und dem Anpassen der Regeln noch eine Zoomfunktion nützlich gewesen, was jedoch zusammen mit der mathematischen Betrachtung zu weit geführt hätte.

Das Lesen von englischer Fachliteratur hat es mir ermöglicht, meine Arbeit auf soliden Grundlagen aufzubauen. Das Ziel, eine geeignete Strategie zum Suchen von Automaten einer bestimmten Klasse zu finden, ist mit Hilfe des Langton-Parameter sowie der Untersuchung von Wachstum und Zerfall erreicht worden. Jedoch wurden keine Raumschiffe gefunden, was für die Zuordnung eines Automaten zur 4. Klasse nötig wäre. Dies liegt wahrscheinlich vor allem daran, dass sie aufgrund der Seltenheit schwieriger zu finden sind als Oszillatoren, von denen doch einige entdeckt wurden. Dazu wäre vermutlich ein Algorithmus wie die Monte Carlo Methode notwendig, welcher selbständig nach Gleitern sucht. Mit wie hier verwendeten trivialen Programmen würde eine erfolgreiche Geleitersuche extrem viel Zeit beanspruchen und wäre alles andere als effizient. Das Ziel, zelluläre Automaten mit dreieckigen sowie auch sechseckigen Gittern zu untersuchen wurde zusammen mit dem Suchen der Objekte sowie auch dem Betrachten von wachsenden Strukturen erreicht. Dabei wurde zudem bei einigen Automaten eine Formel für die Anzahl lebender Zellen zu einem Zeitpunkt t gefunden sowie einige Erklärungen für gewisses Verhalten der wachsenden Strukturen im Allgemeinen und Speziellen mit der Entdeckung des Wachstums entlang der Diagonalen. Durch die Wahl der unkonventionellen Zellularfelder kann es zudem gut sein, dass die eine oder andere Entdeckung oder Überlegung zuvor noch nie gemacht wurde und sich somit mit ein bisschen Glück auch etwas Neues unter den Resultaten finden lässt. Die dritte Frage nach einem Zusammenhang zwischen den Regeln und der Klasse eines Automaten wurde ein Stück weit durch die Literaturrecherche erreicht. Der Langton-Parameter sowie die Untersuchung von Wachstum und Zerfall zeigen einen ungefähren aber nicht vollständigen Zusammenhang auf. Bei dieser Frage hätte eine vollständige Beantwortung zudem für einiges mehr als eine Maturaarbeit gereicht, wodurch auch von Beginn an nicht zu erwarten war, hierauf die endgültige Antwort zu finden. Jedoch waren Überlegungen dazu nützlich und hilfreich beim Bearbeiten der beiden anderen Problemstellungen.

6. Schlusswort

Zuallererst möchte ich mich bei Beat Trachsler bedanken, welcher meine Maturaarbeit betreut hat und mir mit Ratschlägen zur Seite gestanden ist. Zudem geht der Dank auch an meine Familie, besonders an meine Mutter, welche sich die Zeit genommen hat, meine Arbeit durchzulesen und mich auf orthographische und grammatikalische Fehler hingewiesen hat. Ausserdem möchte ich dem Leser gratulieren, welcher meine Arbeit bis hierhin durchgelesen hat. Chapeau! Falls zudem jemand interessiert ist, eine Maturaarbeit im Bereich der Mathematik zu schreiben, ihm/ihr jedoch noch eine Idee fehlt, so kann ich den YouTube-Kanal «Numberphile» empfehlen, vielleicht lässt sich dort noch die nötige Inspiration finden.

Eine interessante Idee wäre vielleicht (jedoch wahrscheinlich nur bedingt für eine Maturaarbeit geeignet) zelluläre Automaten in Bezug auf die Frage nach dem Zusammenhang mit stochastischen Methoden zu untersuchen. Man könnte versuchen, anhand der Regeln auf die Grösse eines «Differenzenmusters» zu schliessen. «Differenzenmuster» entstehen, wenn man den Startzustand einer Zelle des Automaten ändert und anschliessend alle Zellen einfärbt, welche einen anderen Zustand als beim Durchlauf vor der Änderung haben. Mithilfe dieser «Differenzenmuster» lässt sich die Klasse eines Automaten gut bestimmen. [33] Hier könnte es möglich sein, die durchschnittliche Grösse/Population eines Differenzenmusters mithilfe der Stochastik zu bestimmen und somit auf die Klasse zu schliessen und eine Antwort auf die dritte Frage nach dem Zusammenhang zu finden oder zumindest ein Stück näher zu kommen.

7. Verweis

7.1 Literaturverzeichnis

[1] https://de.wikipedia.org/wiki/Brady_Haran (08.10.2017)

[2] https://www.youtube.com/watch?v=E8kUJL04ELA (26.07.2017)

[3] H. S. Martin Gerhardt, «Das digitale Universum: Zelluläre Automaten als Modelle der Natur,» Braunschweig/Wiesbaden, Vieweg, 1995, p. 46.

[4] H. S. Martin Gerhardt, «Das digitale Universum: Zelluläre Automaten als Modelle der Natur,»Braunschweig/Wiesbaden, Vieweg, 1995, pp. S. 72 - 84.

[5] https://en.wikipedia.org/wiki/Rule_110 (24.07.2017)

[6] http://www.wolframalpha.com/input/?i=2+%5E512 (08.08.2017)

[7] A. Ilachinski, «Parameterizing the Space of CA Rules,» in *Cellular Automata: A Discrete Universe*, Singapore, World Scientific, 2001, pp. 98 - 107.

[8] C. G. Langton, «Computation at the Edge of Chaos: Phase Transitions and emergent Computation,» *Physica D*, pp. 12 - 37, 1990.

[9] H. S. Martin Gerhardt, «Das digitale Universum: Zelluläre Automaten als Modelle der Natur,» Braunschweig/Wiesbaden, Vieweg, 1995, pp. 17 -19.

[10] H. S. Martin Gerhardt, «Das Spiel des Lebens,» in *Das digitale Universum: Zelluläre Automaten als Modelle der Natur*, Braunschweig/Wiesbaden, Vieweg, 1995, pp. 33 - 36.

[11] U. Renner, http://www.synatech.de/Digital_ZA/PDF/ZA_Blick.pdf (26.07.2017)

[12] https://en.wikipedia.org/wiki/Cellular_automaton.(25.07.2017)

[13] A. K. Eduardo R. Miranda, «Game of Life Music,» in *Game of Life Cellular Automata*, London, Springer, 2010, pp. 489 - 501.

[14] https://www.bernd-leitenberger.de/programmiersprachen-geschichte.shtml (26.07.2017)

[15] https://de.wikipedia.org/wiki/C-Sharp (25.07.2017)

[16] https://de.wikipedia.org/wiki/Zitronenfalter (26.07.2017)

[17] http://www.das-grosse-computer-abc.de/CSharp/index.php?chapter=02&topic=01.

(26.07. 2017)

[18] https://www.visualstudio.com/de/vs/visual-studio-express/?rr=https%3A%2F%2Fwww.google.ch%2F (25.07.2017)

[19] http://www.das-grosse-computer-abc.de/CSharp/index.php?chapter=10&topic=02

(26.07. 2017)

[20] https://www.sololearn.com/Course/CSharp/ (17.07.2017)

[21] https://msdn.microsoft.com/de-de/library/yysstebh(v=vs.110).aspx. (17.07.2017)

[22] https://www.youtube.com/watch?v=lCMaNlOvbrQ (17.07.2017)

[23] https://msdn.microsoft.com/de-de/library/system.string.indexof(v=vs.110).aspx. (18.07.2017)

[24] https://msdn.microsoft.com/de-de/library/system.timers.timer(v=vs.110).aspx. (18.07.2017)

[25] https://msdn.microsoft.com/de-de/library/system.windows.forms.timer.tick(v=vs.110).aspx. (18.07.2017)

[26] H. S. Martin Gerhadrt, in *Das digitale Universum: Zelluläre Automaten als Modelle der Natur*, Braunschweig/Wiesenbaden, Vieweg Verlag, 1995, pp. 87 - 89.

[27] D. Eppenstein, «Growth and Deca in Life-Like Cellular Automata,» in *Game of Life Cellular Automata*, London, Springer, 2010, pp. 71 - 97.

[28] N. M. Gotts, «Self-Organized Construction in Sparse Random Arrays of Conway's Game of Life,» in *New Constructions in Cellular Automata*, New York, Oxford Univeristy Press, 2003, pp. 1 - 52.

[29] L. B. Emmanuel Sapin, «Searching for Glider Guns in Cellular Automata: Exploring Evolutionary and other Techniques,» Bristol, 2007.

[30] https://de.wikipedia.org/wiki/Division_mit_Rest. (11.08.2017)

[31] H. S. Martin Gerhardt, in *Das digitale Universum: Zelluläre Automaten als Modelle der Natur*, Braunschweig/Wiesbaden, Vieweg Verlag, 1995, pp. 166 - 167.

[32] https://www.pierschel.com/de/software-blog/18-code-signing-tutorial-de. (20.10.2017)

[33] H. S. Martin Gerhardt, in *Das digitale Universum: Zelluläre Automaten als Modelle der Natur*, Braunschweig/Wiesbaden, Vieweg Verlag, 1995, pp. 84 - 86.

[34] H. V. McIntosh, «Conway's Life,» in *Game of Life Cellular Automata*, London, Springer, 1988, pp. 17 - 33.

[35] https://commons.wikimedia.org/wiki/Game_of_Life (25.07.2017)

7.2 Abbildungsverzeichnis

Titelbild: Eigene Bilder, 19.10.2017
Muschel:
https://www.conchology.be/?t=68&u=277650&g=b5750cce8aa90d5d380b33e58e2f05b9&q=0afa989dc6093d98c21e695da160b07e (19.10.2017)

8. Anhang

8.1 Technische Details

Verwendete Software:

- Microsoft Visual Studio Community 2015
- Microsoft Word 2016
- Microsoft Excel 2016

8.2 Arbeitsprotokoll

Datum	Was	Dauer in Minuten
01.04.2017	Einlesen ins Thema	60
05.04.2017	Zeitplan erstellt	15
06.04.2017	Vorbereitung Gespräch	15
18.04.2017	http://www.maturaarbeit.net	30
18.04.2017	Literaturrecherche	90
19.04.2017	Literaturrecherche	180
19.04.2017	Einlesen ins Thema, Video	40
20.04.2016	Literaturrecherche	60
21.04.2017	Einlesen	120
24.04.2017	Programmieren eindimensionaler ZA	180
25.04.2017	Brainstorming, Zielsetzung	30
25.04.2017	Programm Fehlerbehebung	90
27.04.2017	Lesen	60
29.04.2017	Lesen, "A New Kind of Science"	120
21.05.2017	Programmieren Game of Life	180
25.05.2017	Programmieren Game of Life	60
25.05.2017	Programmieren Fehlersuche	90
25.05.2017	Programmieren Dreieck	60
04.06.2016	Programmieren Dreieck	120
08.06.2017	Zentralbibliothek	60
10.06.2017	Programmieren Dreieck	120
15.06.2017	ETH-Bibliothek	60
30.06.2017	Programmieren Sechseck	60
01.07.2017	Programmieren Sechseck	120
08.07.2017	Lesen, "Das digitale Universum"	75
09.07.2017	Lesen, "Das digitale Universum"	180
17.07.2015	Schreiben Programmieren und Verbessern der Programme	360
18.07.2017	Schreiben Programmieren und Verbessern der Programme	330
19.07.2017	Recherche zu Studien	300
20.07.2017	Schreiben und Verbessern Dreieck- und SechseckZA	180
21.07.2017	ETH-Bibliothek, Zentralbibliothek	120

24.07.2017	Programmieren eindimensionalerZA2	30
24.07.2017	Schreiben Grundlagen Mathematik	240
25.07.2017	Schreiben Grundlagen Mathematik	210
27.07.2017	Schreiben Grundlagen Mathematik	300
28.07.2017	Schreiben Grundlagen Mathematik	420
29.07.2017	Schreiben Grundlagen Programmieren	345
31.07.2017	Lesen "Cellular Automata: A Discrete Universe" & Langton's Parameter	450
01.08.2017	Lesen, "New Constructions in Cellular Automata"	180
01.08.2017	Lesen, "Cellular Automata Machines" & "Game of Life Cellular Automata"	320
03.08.2017	Lesen, "Game of Life Cellular Automata" & "The Recursive Universe"	360
04.08.2017	Schreiben Mathematik	210
07.08.2017	Schreiben Mathematik und durchführen von "Experimenten"	375
08.08.2017	Schreiben Mathematik und durchführen von "Experimenten"	270
09.08.2017	Lesen, "Game of Life Cellular Automata"	270
10.08.2017	Schreiben Mathematik	300
11.08.2017	Programmieren Querschnitt	180
22.08.2017	Suchstrategie	90
29.08.2017	Suchstrategie	90
05.09.2017	Wachstum	180
09.09.2017	Programmieren Querschnitt	90
12.09.2017	Wachstum	90
19.09.2017	Programmieren Querschnitte	90
05.10.2017	Überlegungen zu Ausbreitungen von Veränderungen (Differenzenmuster)	120
08.10.2017	Überarbeiten Grundlagen	360
09.10.2017	Überarbeiten Methoden	330
10.10.2017	Verbessern der Programme	180
13.10.2017	Wachstum	360
14.10.2017	Wachstum	240
16.10.2017	Wachstum	150
17.10.2017	Modulo-2-Regel und Wachstum	300
18.10.2017	Wachstum, Überarbeiten	360
19.10.2017	Einleitung, Abstract, Titelblatt	330
20.10.2017	Diskussion, Schlusswort, CD brennen, Überarbeiten	420
21.10.2017	Überarbeiten	480
22.10.2017	Überarbeiten	420