UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Algorithmische Komposition

unter Verwendung des Lindenmayer Systems und des Zellulären Automaten

Bachelorarbeit

Zur Erlangung des Grades Bachelor of Science (B.Sc.)

Im Studiengang Computervisualistik

vorgelegt von

Timothy Steven Schürstedt

Erstgutachter: Prof. Dr.-Ing Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)

Zweitgutachter: Bastian Krayer, M.Sc.
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, 24.09.18

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

. . . . . .

(Ort, Datum) (Unterschrift)

Zusammenfassung
Algorithmische Komposition ist ein interdisziplinärer Forschungsbereich, der die beiden Bereiche Musik und Wissenschaft miteinander verknüpft. Der Computer wird in den Mittelpunkt des Kompositionsprozesses gestellt und komponiert mithilfe eines Algorithmus Musik. In dieser Arbeit wird die Algorithmische Komposition unter Verwendung der biologisch inspirierten Algorithmen Lindenmayer-System und Zellulärer Automat untersucht. Dabei werden ausgewählte Verfahren vorgestellt, implementiert und evaluiert, die die erzeugten Daten der Algorithmen in ein sinnvolles musikalisches Ergebnis transformieren.

Abstract
Algorithmic composition is an interdisciplinary topic that unites music and science. The computer is able to generate algorithmic music with the aid of a specific algorithm. In this bachelor thesis, algorithmic composition is realized with the biology-inspired algorithms called Lindenmayer-system and cellular automaton. In order to realize the compositions, several techniques are presented as well as implemented and evaluated. Those techniques map the generated data from the algorithms on a meaningful musical result.

Inhaltsverzeichnis

Um die Hörbeispiele abspielen zu können, muss das Dokument mit dem Adobe Acrobat Reader DC geöffnet werden und es muss ein Flash-Player installiert sein.

1 Einleitung

Algorithmische Komposition ist das Ergebnis der Kombination von künstlerischer Kreativität und wissenschaftlicher Methodik. Das Zusammenspiel von der einerseits kreativen Freiheit, die sich durch das Abweichen von Regeln kennzeichnet, um neue Klangwelten zu entdecken, und der andererseits geordneten Struktur, dem strengen Befolgen von harmonischen Regeln und Form, erzeugt die große Faszination Musik. Welcher Part in einer Komposition überwiegt, ist von den einzelnen Komponisten und der jeweiligen Schaffensepoche abhängig. Die Algorithmische Komposition versucht bei dem Kompositionsprozess unterstützend zu wirken.
Die Algorithmen liefern dabei die notwendigen Muster und Sequenzen, um räumlich und zeitlich in einem Medium Musik zu kreieren. In dieser Arbeit geht es darum Verfahren zur Computergestützten Komposition zu finden, zu implementieren und zu evaluieren. Dabei werden die generativen Algorithmen Lindenmayer-System (L-System) und Zelluläre Automaten verwendet. Diese Algorithmen zeichnen sich durch die wiederholte Anwendung derselben Regeln aus. Simple Eingabe Parameter werden so durch das wiederholte Anwenden von Regeln schrittweise transformiert und nehmen dabei an Komplexität zu. Jeder Entwicklungsschritt vergrößert hierbei die Menge der Daten. Komplexe Objekte können so mit einfachen Regeln dargestellt werden. Anhand der Eingangsdaten lässt sich kaum vorhersehen, wie sich die Daten nach einigen Transformationsschritten verhalten.
Eine Technik zu finden, um die Evolutionen von Zellulären Automaten, oder die Zeichenfolge eines L-Systems in wohlklingende und interessante Musik abzubilden, ist ein nichttriviales Problem. Es existieren viele Herangehensweisen, um dieses Problem zu lösen. Die vorliegende Arbeit beschränkt sich auf wenige dieser Techniken, die dem gewünschten Ziel am nächsten kommen. Der Fokus liegt dabei darauf die Algorithmen so wie sie sind zu auralisieren. Der Begriff schöne oder wohlklingende Musik wird von Person zu Person und von Kulturkreis zu Kulturkreis unterschiedlich interpretiert. Es ist nicht das Ziel, die Algorithmen in einer Art und Weise von einer subjektiven Meinung einzuschränken, damit sie irgendeinen populären Musikstil ähneln. Vielmehr geht es darum einen hohen Grad an Freiheit zu gewährleisten, um unvorhersehbare und interessante Ergebnisse zu erhalten. Dies hat jedoch auch einen gewissen Preis. So ist die von den Algorithmen produzierte Musik oftmals gewöhnungsbedürftig und fordert den Hörer auf, sich näher mit ihr zu beschäftigen, um sie voll verstehen und genießen zu können.
Es stellt sich die Frage, welche Relevanz manchen Ergebnissen zugesprochen werden kann. Die Verfahren stehen erst am Anfang eines kompletten musikalischen Ergebnisses und liefern lediglich eine erste Idee bzw. eine Melodie, die dann mit weiteren Algorithmen oder vom Menschen selbst weiter verarbeitet werden muss. Der griechische Komponist Iannis Xenakis verwendete beispielsweise für sein Orchesterwerk Horos einen Zellulären Automaten als Grundlage für den Harmonischen Bauplan des Stücks [1].
Im folgenden Kapitel werden Grundlagen der Algorithmen und ihre musikalische Interpretation erklärt, sowie eine Implementation und Visualisierung der Verfahren in SuperCollider aufgezeigt. Am Ende werden die unterschiedlichen Verfahren miteinander verglichen. [2][3]

2 Grundlagen

Das folgende Kapitel beschäftigt sich mit der grundsätzlichen Herangehensweise an das Generieren von Musik mithilfe des L-Systems und des Zellulären Automaten. Außerdem wird ein Überblick über die Techniken geliefert, die für das Umsetzen von Algorithmendaten existieren. In der Entwicklungsumgebung SuperCollider werden die verschiedenen Techniken implementiert, dafür werden kurz die Grundzüge der Sprache erläutert und besonders wichtige Klassen vorgestellt.

2.1 Auralisierungspipeline

Ähnlich wie in der Visualisierungspipeline der Computergrafik lässt sich auch für die Musikgenerierung eine generelle Abfolge von Arbeitsschritten festlegen. Auf Abbildung 1 ist eine Pipeline zur Auralisierung von L-System und Zellulären Automaten gegeben. Auralisierung bedeutet, dass die von den Algorithmen erzeugten Daten klanglich erlebbar dargestellt werden, ähnlich zur Visualisierung, in der sie optisch dargestellt werden. Nach der Wahl von Eingangsparametern liefern die Algorithmen eine bestimmte Datenmenge, die als Basis für die Weiterverarbeitung dient. Aufgrund der Natur der Algorithmen sind hier bereits viele Muster und Sequenzen vorhanden, die bei einer grafischen Darstellung schnell zu erkennen sind. Die Herausforderung besteht darin diese nun in den weiteren Schritten hörbar zu machen. Die Selektion der erzeugten Daten ist bei den in dieser Arbeit beschriebenen Algorithmen und Techniken nicht von Bedeutung, da die Daten bereits so generiert werden, wie sie schlussendlich benötigt werden.
Das Mapping stellt den wichtigsten und umfangreichsten Schritt der Pipeline dar, da hier die Algorithmendaten auf Musik abgebildet werden. Man könnte diesen Schritt auch als Projektion verstehen. So werden die von den Algorithmen erzeugten Daten in einen Raum abgebildet, der durch Tonhöhe, Tondauer und teilweise auch Tonlautstärke definiert wird. Das Mapping stellt dabei die Funktion dar, die diesen Schritt durchführt. Dies so zu bewerkstelligen, dass schlussendlich hörenswerte Musik entsteht, ist eine schwierige Aufgabe, für die es keine feste Vorgehensweise gibt. In dieser Arbeit werden drei Mapping-Verfahren vorgestellt, die diese Problematik wenigstens teilweise lösen. Die Verfahren sind so ausgewählt, dass sie die Struktur und Muster der Algorithmendaten möglichst simpel und intuitiv darstellen. Im letzten Schritt werden die fertig umgewandelten Daten für den Hörer wiedergegeben. Dafür wird in dieser Arbeit die Programmierumgebung SuperCollider verwendet.

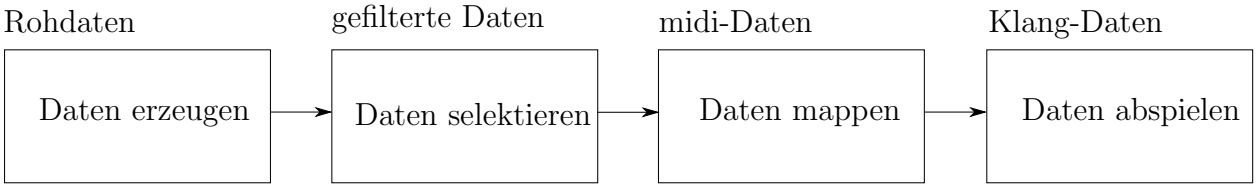

Abbildung 1: Auralisierungspipeline

2.2 Überblick über mögliche Mapping-Verfahren

Sowohl Lindenmayer-System als auch Zellulärer Automat, finden in der Literatur immer wieder musikalische Verwendung. Das erste Verfahren zur Musikgenerierung mit

L-Systemen stammt aus dem Jahr 1986 von Przemyslaw Prusinkiewicz [4]. Dieser berechnet aus einer von dem L-System generierten Grafik, Tonhöhe und Tondauer. Das Verfahren von Prusinkiewicz wird in der Literatur immer wieder durch verschiedene Techniken erweitert, so implementieren Stephanie Mason und Michael Saffle eine mehrstimmige Version des Verfahrens [3]. Stelios Manousakis erweitert das Verfahren von Prusinkiewicz um die dritte Dimension und fügt mehrere zusätzliche Parameter für die Klangfarbe ein [5]. Bei Peter Worth und Susan Stepney wird das Verfahren weiter vereinfacht und Gemeinsamkeiten mit der generierten Grafik untersucht [6]. Einen mehr grammatikorientierten Weg geht Jon McCormack, der die Zeichenfolge des L-Systems direkt auf einzelne Noten abbildet [2].
Für die musikalische Interpretation eines Zellulären Automaten sind in der Literatur viele, teils sehr komplexe Verfahren zu finden. Paul D. Reiners berechnet aus jeder Zeile eines eindimensionalen Zellulären Automaten die Tonhöhe [7], Simon Cutajar zusätzlich Tondauer und Tonlautstärke [8]. Auf der Website wolfram.tones.com ist es möglich komplette Musikstücke zu generieren, hierfür wird ein um 90° gedrehter Ausschnitt eines Zellulären Automaten als Notenbild interpretiert [9]. Das umfangreichste Verfahren stammt von Eduardo Reck Miranda. Dieser entwickelte CAMUS (Cellular Automata MUsic), welches zwei zweidimensionale Zelluläre Automaten zur Musikgenerierung verwendet [10].

2.3 Programmierumgebung SuperCollider

SuperCollider ist eine Entwicklungsumgebung, Programmiersprache und ein Audio-Server für Algorithmische Komposition und Klangsynthese. SuperCollider wurde von James McCartney entwickelt und erschien 1996. Es besteht aus zwei Komponenten, dem Audio-Server *scsynth* und der Programmiersprache *sclang*. Alle Klassen und Verfahren in dieser Arbeit wurden in SuperCollider implementiert, unter Verwendung vieler verfügbarer Klassen und Methoden für die Algorithmische Komposition. $SynthDef$ und *Pbind* sind die wichtigsten Klassen, die in jedem implementierten Verfahren verwendet werden. *Pbind* spezifiziert was für Töne ausgegeben werden sollen und $SynthDef$ wie diese Töne klingen sollen.

$SynthDef$ steht für Synthesizer Definition und ermöglicht dem Nutzer, wie der Name verrät, ein eigenen Synthesizer zu kreieren, der dann per Klangsynthese Töne erzeugt. Mit der Klasse können unter Verwendung von Techniken wie z.B. additive oder subtraktive Klangsynthese, Klänge von bekannten Instrumenten nachgebaut oder komplett neue Klänge erschaffen werden. Für das Abspielen der, in dieser Arbeit vorgestellten Verfahren, erzeugten Musik werden zwei verschiedene $SynthDefs$ (Klavier- und Orgelklang) verwendet. Der Klavierklang zeichnet sich durch einen harten, kurzen Klang aus, indem wenige zusätzliche Frequenzen mitschwingen. Mit diesem Klang ist es möglich, in polyphoner und sehr schneller Musik, jeden einzelnen Ton deutlich zu hören. Der Orgelklang dagegen ist weicher und länger und hört sich durch viele mitschwingende Frequenzen voller an. Polyphone Klänge vermischen sich so zu einer homogenen Masse.

```
(
SynthDef(\organ, { |out, freq=440, amp=0.1, gate=1|
    var snd;
    snd = Splay.ar(SinOsc.ar(freq*Array.geom(4,1,2), mul:1/4));
    snd = snd + SinOsc.ar(freq/2, mul:0.4)!2;
    snd = snd * EnvGen.ar(Env.asr(0.001,1,0.01), gate, doneAction
       :2);
    OffsetOut.ar(out, snd * amp);
}).add;
)
```

Listing 1: SuperCollider SynthDef

Auf Listing 1 ist der *SynthDef* des Orgelklanges abgebildet. Über den Parameter *freq* kann die auszugebende Frequenz gewählt werden, die dann mittels mehrerer Klangsynthesetechniken manipuliert wird.
Pbind steht für Pattern binding und dient zur Zusammenfassung von verschiedenen Datenströmen. Mit *Pbind* ist es beispielsweise möglich ein Array mit Tonhöhen, mit einem Array von Tondauern zu verbinden und parallel zu durchlaufen bzw. abzuspielen. Es erklingt der entsprechende Ton mit der entsprechenden Dauer. Andere wichtige Parameter sind hier mit ihrem Befehl im Code aufgeführt:

- Tonhöhe (*degree* oder *midinote*)
- Skala (*scale*)
- Tondauer (*dur*)
- Tonlautstärke (*amp*)
- Instrument, bzw. Synthesizer (*instrument*)

Die Tonhöhe kann entweder als midinote oder als scale degree angegeben werden. Midinoten ordnen jeden Ton eine Zahl zu und bewegen sich im Bereich 0-127. Um degree verwenden zu können, muss auch die verwendete Skala angegeben werden. Der scale degree gibt dann mit den Zahlen 1-7 an, welcher Ton der Skala gespielt werden soll. Bei höheren oder niedrigeren Zahlen wird dann entsprechend die nächste Oktave verwendet. Um in andere Tonarten zu transponieren, ist es notwendig eine Konstante auf zu addieren. Als Instrument muss ein *SynthDef* angegeben werden, diesem *SynthDef* werden dann die in Frequenzen transformierten Töne als Parameter übergeben.

```
(
Pbind(
 \instrument, \organ,
 \degree, Pseq([1,3,5,3,1],1),
 \dur,    Pseq([1/4,1/2],inf),
 \amp,    0.5,
 \scale,  Scale.major
).play;
)
```

Listing 2: SuperCollider Pbind

Auf Listing 2 ist ein beispielhaftes *Pbind* Konstrukt abgebildet, welches den vorher erwähnten Orgelklang als Instrument verwendet. [11]

3 Implementation

In diesem Kapitel werden die Grundlagen der verwendeten Algorithmen, in Bezug auf ihre Relevanz für die musikalische Interpretation, erklärt. Danach werden Techniken vorgestellt mit denen die Algorithmen musikalisch abgebildet werden können und die Implementation und Visualisierung in SuperCollider erläutert.

3.1 Lindenmayer-System

Das Lindenmayer-System ist ein um 1968 von Astrid Lindenmayer entwickeltes Ersetzungssystem, das einzelne Bestandteile durch Anwenden von Produktionsregeln sukzessive ersetzt. Vorbild für L-Systeme ist die gleichzeitige stattfindende Zellteilung in mehrzelligen Organismen. So wurde das L-System vor allem zur realitätsnahen Abbildung von Pflanzen und Fraktalen genutzt. Durch die einfache Grundidee und den großen Interpretationsspielraum der entstandenen Zeichenfolge eignen sich L-Systeme ebenfalls gut als Basis für Algorithmische Kompositionen. Beim sonifizieren von L-Systemen wird auf das Verfahren von Prusinkiewicz [4] zurückgegriffen. Voraussetzung für dieses Verfahren ist eine vorherige Interpretation der Zeichenfolge als Turtle-Grafik, damit aus dieser dann Tonhöhe und Dauer berechnet werden kann. Somit ist die grafische und musikalische Interpretationen eines L-Systems eng miteinander verbunden.

3.1.1 Grundlegendes Verfahren

Für diese Arbeit sind die kontextfreien und deterministischen D0L-Systeme wichtig. Das sind L-Systeme, die für den gleichen Input immer denselben Output haben und deren Produktionsregeln den Kontext des eingelesenen Zeichens ignorieren.
Formell sind sie wie folgt definiert: Ein 0L System ist ein geordnetes Triplet $G = (V, \omega, P)$, wobei V das Alphabet des Systems ist, $\omega \in V^+$ ist ein nichtleeres Wort, das als Axiom dient, und $P \subset V \times V^*$ ist eine endliche Anzahl an Produktionsregeln. Falls ein Tupel (a, x) eine Produktionsregel $a \to x$ beschreibt, wird a als Vorgänger und x als Nachfolger bezeichnet. Ein 0L-System wird deterministisch genannt, falls für jeden Vorgänger $a \in V$ genau ein Nachfolger $x \in V^*$ existiert, sodass $a \to x$.
Darüber hinaus existieren auch kontextsensitive, stochastische oder parametrische L-Systeme, die sich für eine musikalische Interpretation ebenso gut eignen, hier aber nicht keine Beachtung finden.
Ein L-System liest ein Axiom ω ein und wendet darauf n-mal die Produktionsregeln an, wobei sich die Ausgabe exponentiell vergrößert. $n \in \mathbb{N}$ steht hierbei für die Rekursionstiefe des L-Systems. Der entscheidende Unterschied zu den Grammatiksystemen von Chomsky besteht darin, dass beim L-System die Regeln gleichzeitig auf alle Buchstaben angewandt werden.
Formel 1 zeigt ein L-System mit zwei Regeln, einer Rekursionstiefe von fünf und dem Axiom b. Auf Abbildung 2 ist zu sehen, wie die Regeln des L-Systems sukzessive angewandt werden: So wird aus dem Axiom b nach fünf Schritten die Zeichenfolge *abaababa*. In den nächsten Abschnitten wird es nun darum gehen, diese Zeichenfolge zu interpretieren und ihren fraktalen Charakter sichtbar zu machen. [12]

L-System (siehe Abbildung 2):

$$n = 5, \omega = b$$

$$a \to ab$$

$$b \to a \tag{1}$$

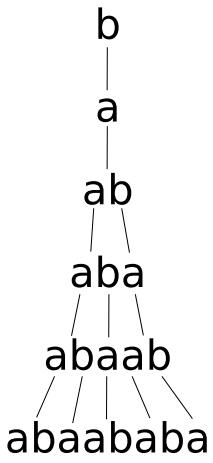

Abbildung 2: Ersetzungsschritte eines L-Systems

3.1.2 Interpretation von L-Systemen

Die klassische Interpretation von L-Systemen erfolgt mittels der Turtle-Grafik in eine zweidimensionale Grafik. Da diese die Basis für die musikalische Interpretation ist, wird sie hier zuerst erläutert.

3.1.2.1 Grafische Interpretation Zur grafischen Interpretation der Zeichenkette wird eine 'LOGO-Turtle' auf einer zweidimensionalen Fläche angenommen, die sich mit dem Alphabet $V = \{F, f, +, -\}$ über die Fläche bewegen kann. Dabei zeichnet sie eine Linie hinter sich, die ihren Laufweg kennzeichnet. Sie ist wie folgt definiert: Die Turtle hat zu jedem Zeitpunkt einen Zustand (x, y, a), wobei x und y für die entsprechenden kartesischen Koordinaten stehen und a die Blickrichtung in Grad beschreibt. Dieser Zustand kann dann durch Einlesen der einzelnen Zeichen verändert werden. Dabei wird von einer vorher festgelegten Schrittgröße d und einer Winkeländerungsgröße δ ausgegangen.

- F Bewege dich um d in Blickrichtung und zeichne eine Linie.
  neuer Status der Turtle: (x', y', α), $x' = x + d\cos\alpha$
  $y' = y + d\sin\alpha$, es wird eine Linie zwischen (x,y) und (x',y') gezeichnet
- f Bewege dich um d in Blickrichtung und zeichne keine Linie.
  neuer Status der Turtle: (x', y', α), $x' = x + d\cos\alpha$
  $y' = y + d\sin\alpha$
- $+$ Drehe dich um $\delta°$ nach rechts
  neuer Status der Turtle: (x, y, α'), $\alpha' = \alpha + \delta$
- $-$ Drehe dich um $\delta°$ nach links
  neuer Status der Turtle: (x, y, α'), $\alpha' = \alpha - \delta$

Alle anderen Zeichen werden von der Turtle ignoriert und ihr Zustand bleibt unverändert. Mit diesen einfachen Regeln können nun bekannte Fraktale, wie das Sierpinksi-Dreieck oder die Kochsche Schneeflocke, durch L-Systeme abgebildet werden. Abbildung 4 und 5 zeigen von L-Systemen erstellte FASS-Kurven (space-filling, self-avoiding,

simple and self-similar). Fass-Kurven sind raumfüllende Kurven, die eine zweidimensionale Fläche komplett durchlaufen und so ihren Flächeninhalt approximieren.
Bis jetzt kann die Turtle nur eine einzige Linie zeichnen, aber für viele Abbildungen ist eine Verzweigung der Linie notwendig. Um dies zu ermöglichen, gibt es zwei zusätzliche Branching-Zeichen, eckige Klammer links "["was eine neue Verzweigung beginnt und eckige Klammer rechts "]"welches eine bestehende Verzweigung verlässt. Vor allem zur realitätsnahen Modellierung von Pflanzen ist diese Technik elementar, um beispielsweise die vielen Verzweigungen eines Astes darstellen zu können.

3.1.2.2 Edge- und node rewriting Die auf Abbildung 4 und 5 abgebildeten Kurven wurden mit den L-Systemen von Formel 2 und 3 erzeugt. Um diese L-Systeme zu entwickeln, wurden die Techniken edge- und node rewriting verwendet. Diese Techniken stellen einen Lösungsansatz für das Inferenz-Problem dar. Allgemein ist es bei L-Systemen sehr schwierig, aus der formalen Beschreibung also durch das Axiom, die Produktionsregeln und die Rekursionstiefe vorherzusehen, wie das Ergebnis der Turtle-Grafik davon aussieht. Schon kleine Änderungen an den Ausgangsparametern können großen Einfluss auf die erstellte Grafik haben. Oftmals möchte man aber eine bestimmte Struktur von einem L-System abbilden lassen. Dieses Problem wird als Inferenz-Problem bezeichnet. Edge- und node rewriting stellen dabei Techniken dar, mit deren Hilfe dieses Problem teilweise gelöst werden kann. In [12] wird ein Algorithmus beschrieben durch dem unter Anwendung der genannten Verfahren, FASS-Kurven konstruiert werden können. In dieser Arbeit wird lediglich die Grundidee der beiden Techniken skizziert.

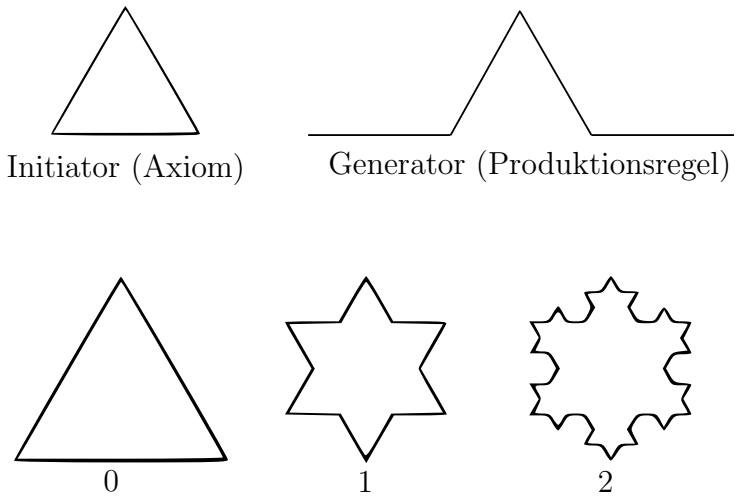

Abbildung 3: Konstruktion der Kochschneeflocke

Edge rewriting
Edge rewriting kann als Erweiterung der Koch-Konstruktionen angesehen werden, in der einzelne Linien sukzessive durch Polygonkanten (edges) ersetzt werden. In den Koch-Konstruktionen wird ein Linienstück immer wieder durch ein anderes ersetzt und das ganze rekursiv wiederholt. Auf Abbildung 3 ist dies am Beispiel der Kochschneeflocke zu sehen. Es wird einzig ein Initiator und ein Generator benötigt. Die einzelnen Linien des Initiators werden durch das Polygon des Generators ersetzt und dieser Vorgang wird rekursiv wiederholt (unten auf Abbildung 3 abgebildet). Dies lässt sich auch leicht mit einem L-System darstellen, in dem das Axiom der Initiator ist,

und die Produktionsregel der Generator. Die Erweiterung von edge rewriting ist es, zwischen den beiden Kanten links und rechts zu unterscheiden. Es gibt also eine Produktionsregel für ein rechtes Teilstück und eine für ein linkes. In Formel 2 sind diese Produktionsregeln nun als F_r für rechts und F_l für links zu sehen. F_r und F_l werden in der Produktion jeweils durch Gebilde substituiert, die Links- oder Rechtsdrehungen beschreiben. F_r und F_l werden von der Turtle-Grafik als F interpretiert.
L-System für quadratische Gosperkurve (Siehe Abbildung 4):

$$n = 2, \delta = 90, \omega = -F_r$$

$$\begin{aligned} &F_l \rightarrow F_l F_l - F_r - F_r + F_l + F_l - F_r - F_r F_l + F_r + F_l F_l F_r - F_l + F_r + F_l F_l + F_r - F_l F_r - F_r - F_l + F_l + F_r F_r - \\ &F_r \rightarrow + F_l F_l - F_r - F_r + F_l + F_l F_r + F_l - F_r F_r - F_l - F_r + F_l F_r F_r - F_l - F_r F_l + F_l + F_r - F_r - F_l + F_l + F_r F_r \end{aligned} \tag{2}$$

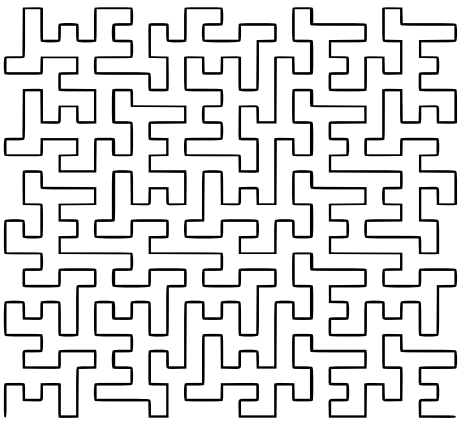

Abbildung 4: Quadratische Gosperkurve, edge-rewriting [12]

Node rewriting
Node rewriting fügt neue Polygone an bestehende an. Dafür werden neue Buchstaben zum Alphabet hinzugefügt, die willkürliche Polygone beschreiben. Ein solches Polygon besteht dabei aus zwei Kontaktpunkten, einem Eintritts- und einem Ausgangspunkt, sowie zwei Richtungsvektoren, einen Eintritts- und Ausgangsvektor. Der entsprechende Buchstabe fügt dann das Polygon in die bereits erstellte Grafik ein. Dafür wird das eingefügte Polygon so gedreht, dass sein Eintrittspunkt und Vektor mit der aktuellen Turtle-Position und Ausrichtung übereinstimmt. Nachdem das Polygon platziert wurde, wird der Turtle-Position der Ausgangspunkt und Vektor zugewiesen. Die Hilbertkurve von Formel 3 wurde bspw. mit dieser Methode konstruiert. Die Buchstaben X und Y stehen für Polygone, die jeweils eine Links- oder Rechtskurve beschreiben.
Edge und node rewriting sind eng miteinander verwandt. Mit beiden Verfahren können dieselben Kurven beschrieben werden und im Endeffekt ist es eine reine Bequemlichkeitsfrage, welches Verfahren verwendet wird. [12]
L-System für Hilbertkurve (Siehe Abbildung 5):

$$n = 4, \delta = 90, \omega = +X$$

$$\begin{aligned} X &\rightarrow -YF + XFX + FY - \\ Y &\rightarrow +XF - YFY - FX + \end{aligned} \tag{3}$$

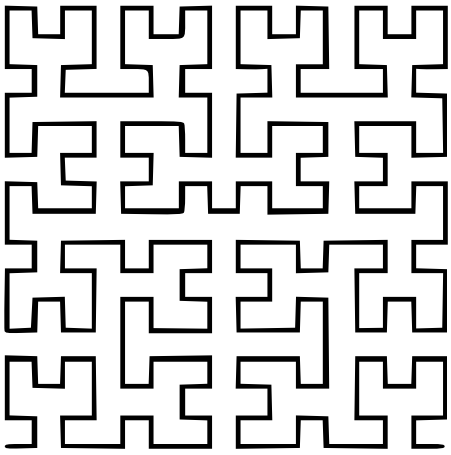

Abbildung 5: Hilbertkurve [12]

3.1.2.3 Musikalische Interpretation mit dem Verfahren von Prusinkiewicz

In Prusinkiewicz 1986 ist die entstandene Abbildung der Turtle-Grafik Ausgangspunkt für die musikalische Interpretation. Mithilfe der Grafik werden Tonhöhe und Tondauer berechnet, indem ein Koordinatensystem mit Ursprung links unten auf die Grafik gelegt wird. Die x-Achse gibt dann die Tondauer an und die y-Achse die Tonhöhe. Die Einheiten der Achsen können prinzipiell frei gewählt werden, Sinn macht eine Einteilung der y-Achse in Skalentöne, also in C-Dur bspw.: c-d-e-f-g-a-h..usw. So kann jedem Punkt in der Grafik ein Skalenton zugeordnet werden. Ob der Ton gespielt wird und wie lange er erklingt, hängt von der x-Achse ab. Hier kann keine feste Einteilung vorgenommen werden, da die Dauer des Tons von den zusammenhängenden horizontalen Linienstücken abhängt. Jedes zusätzliche Linienstück verlängert dabei den Ton um einen festen Wert. Ein Ton wird also nur tatsächlich gespielt, wenn nach einer Tonhöhenänderung, also einer vertikalen Linie, eine horizontale folgt. Mit dieser Technik ist es auch möglich denselben Ton mehrmals hintereinander zu spielen. Hierfür muss eine vertikale Linie in sich selbst zurücklaufen. Um Pausen in das Stück einzufügen, ist das Symbol f erforderlich, das die Turtle-Position verändert, aber keine Linie zeichnet. In vertikaler Richtung hat es keine Auswirkung, der Ton wird trotzdem um einen Skalenschritt erhöht oder erniedrigt, obwohl keine Linie gezeichnet wird. In horizontaler Richtung dagegen fügt jede unsichtbare Linie eine Pause um einen festgelegten Wert ein.
Auf Abbildung 6 ist das Verfahren am Beispiel der Hilbertkurve zu sehen. Der Ursprung des Koordinatensystems stellt dabei den Ton C dar und jede vertikale Linie verlängert die Tondauer um $\frac{1}{4}$. Die Kurve wird Schritt für Schritt komplett durchlaufen und jeder Ton entsprechend aufgezeichnet. Dabei spielt es bei den horizontalen Linienstücken keine Rolle, in welche Richtung sie gehen. Neben der Hilbertkurve sind die resultierenden Noten zu sehen. Der Takt kann hierbei frei gewählt werden, da er lediglich zur Festlegung der Betonungen dient und vom Computer keine Töne hervorgehoben werden.

Hörbeispiel Hilbertkurve | Play
Hilbertkurve aus Abbildung 6 mit Verfahren von Prusinkiewicz, Klavierklang
Hörbeispiel Gosperkurve | Play
Gosperkurve aus Abbildung 4 mit Verfahren von Prusinkiewicz, Klavierklang

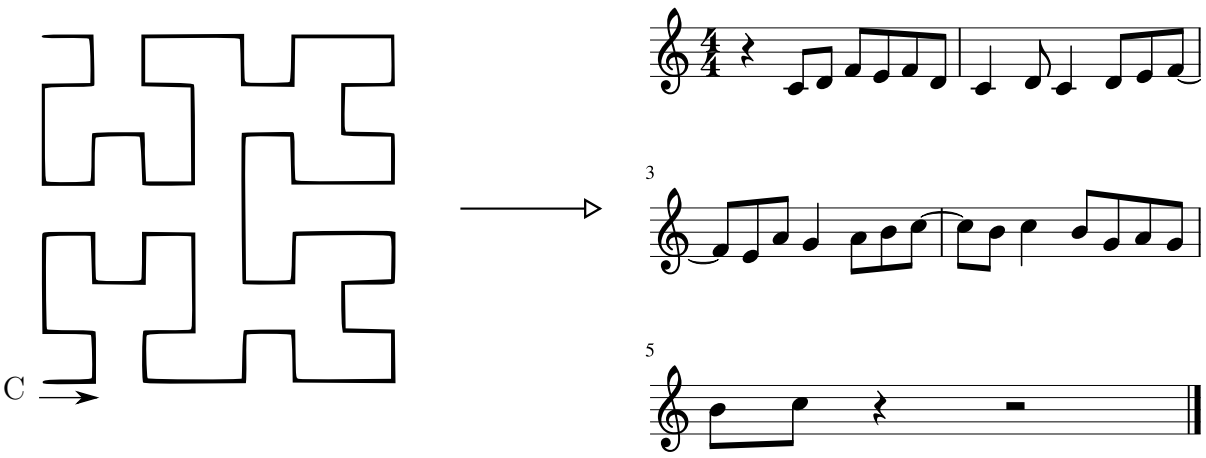

Abbildung 6: Hilbertkurve und ihre musikalische Interpretation[4]

Jede Kurve die ausschließlich aus vertikalen und horizontalen Elementen besteht, kann so musikalisch interpretiert werden und jede Melodie kann durch solch eine Kurve dargestellt werden.
Für dieses Verfahren eignen sich besonders FASS-Kurven, da sie die erforderlichen Anforderungen erfüllen und sich innerhalb einer Fläche bewegen. Andere Kurven neigen oft dazu sich außerhalb des hörbaren Bereiches zu bewegen, der ab Skalenton 65 beginnt. Um dieses Problem zu lösen, gibt es mehrere Möglichkeiten. Entweder man versucht die Kurven auf den hörbaren Bereich zu beschränken, was praktisch sehr schwer umzusetzen ist, oder man verändert die Tonhöhe, sobald die Kurve außerhalb des hörbaren Bereichs läuft. Hier bietet sich eine Modulo Operation an, um die Kurve dann wieder von Anfang der Skala beginnen zu lassen, oder eine Spiegelung am Rand. [4]
Fass-Kurven hören sich mit dem Verfahren von Prusinkiewicz sehr harmonisch an und ihre fraktalen Wiederholungen sind gut hörbar. Auf Dauer klingen sie allerdings langweilig, da sich dieselben Melodieausschnitte die ganze Zeit transponiert wiederholen. Interessanter sind chaotischere Kurven, deren grafische Erscheinung weniger geordnet ist. Ob ein Zusammenhang zwischen der Ästhetik von grafischer und musikalischer Ausgabe besteht, wird in Worth 2005 diskutiert. Hier wird ein leicht anderes Verfahren verwendet, mit dem Ergebnis, dass gut klingende L-Systeme keine schönen grafischen Interpretationen erzeugen. Das liegt vor allem an der Branching Mechanik, die für das realistische Modellieren von Pflanzen unabdingbar ist, jedoch in der musikalischen Interpretationen nicht klar ist, wie diese umgesetzt werden soll. [6]

Erweiterungen des Verfahrens von Prusinkiewicz durch Mason[3]:
In Mason 1994 wird die Technik von Prusinkiewicz erweitert. Es ist möglich, die Kurve um 90° zu drehen oder sie rückwärts abzuspielen, um ein anderes musikalisches Ergebnis zu erhalten. Solange die Kurve kein Palindrom ist, sind acht verschiedene Varianten möglich. Jede so neu entstehende Melodie ist die Transformation einer anderen. Mehrere verschiedene Versionen einer Kurve können dann parallel abgespielt werden, um eine Polyphonie bzw. einen Kontrapunkt zu erhalten.

Hörbeispiel mason	Play

Hilbertkurve normal und um 90° gedreht parallel abgespielt, Klavierklang

Außerdem wird eine erweiterte Technik für edge/node rewriting vorgestellt, die speziell auf eine musikalische Interpretation zugeschnitten ist. So ist es, anders wie bei FASS-Kurven, nicht wichtig, dass die entstehende Kurve sich selbst nicht schneidet. Um die Technik anzuwenden, muss die Kurve in ihrem Anfangsstadium gebunden sein, also jedes Feld eines beliebig großen Gitters durchlaufen. So sind die musikalischen Ausdrucksmöglichkeiten beschränkt.[3]

3.1.3 Programmierung und visuelle Darstellung mit SuperCollider

In SuperCollider existiert bereits eine Implementation von L-Systemen, die in der vorliegenden Arbeit als Basis für die Implementation des Verfahrens von Prusinkiewicz verwendet wird [13]. Auf Listing 3 ist die Klasse *Lsys* zu sehen, mit der ein L-System mit einem Axiom und den entsprechenden Regeln erstellt wird. Mit $applyRules(int\ n)$ werden die Regeln n mal angewandt und die entstehende Zeichenfolge wird mit *giveParsedString* in eine Liste konvertiert. Diese Liste ist dann Grundlage für die weitere Verarbeitung.

```
1 //Hilbert Curve
2 (
3 a = LSys("+X", [
4  "X" -> "-YF+XFX+FY-",
5  "Y" -> "+XF-YFY-FX+"
6 ]);
7 )
8 k = a.applyRules(5);
9 k = a.giveParsedString;
```

Listing 3: SuperCollider LSys

Die Funktion $prusinkiewicz(list\ l)$ wendet auf eine als Liste übergebene Zeichenfolge das oben erklärte Verfahren an und gibt ein zweidimensionales Array $Array[scale\ degree][duration]$ zurück. Dieses Array kann daraufhin mithilfe von *Pbind* abgespielt werden. Zum Abspielen wird hier als Instrument ein Klavierklang verwendet, weil so die vielen schnell aufeinanderfolgenden Töne am besten zu hören sind.
Eine grafische Interpretation des L-Systems ist ebenfalls bereits implementiert worden. Mit der Klasse *LSPlant* kann ein L-System gezeichnet werden. Die einzige Schwierigkeit zur Visualisierung der Musik besteht darin die Musik und die Grafik zu synchronisieren. *LSPlant* bietet hierfür den Befehl *speed*, mit dem die Zeichengeschwindigkeit angepasst werden kann.

$$speed = \frac{TotalLines}{TotalDuration} \tag{4}$$

Mit Formel 4 kann dann die benötigte Geschwindigkeit ausgerechnet werden, damit das Erstellen der Grafik genauso viel Zeit in Anspruch nimmt wie das Abspielen der Musik. Dafür wird die gesamte Anzahl der gezeichneten Linien (also alle F Symbole) durch die Gesamtdauer des Musikstücks (die Summe der *durations*) geteilt.

3.1.4 Genetischer Algorithmus

Wie in den vorherigen Kapiteln beschrieben ist es sehr schwierig aus der formalen Beschreibung eines L-Systems die interpretierte Grafik zu erkennen. Genauso ist es

schwierig bis unmöglich zu entscheiden, ob aus einem L-System gute oder schlechte Musik entsteht. Nur durch die Simulation und das Anwenden aller Schritte kann dies entschieden werden. Hier lassen sich Parallelen zur Genetik ziehen. Die natürliche Selektion wählt keine Gene nach bestimmten Kriterien aus, sondern rein nach den Effekten der Gene auf den Organismus und der damit verbundenen Eignung für die herrschenden Umweltbedingungen.
Genau hier liegt die Idee des Genetische Algorithmus. Ein Genetischer Algorithmus ist ein Such- bzw. Optimierungsalgorithmus, der besonders für Probleme geeignet ist, über die nur wenige Informationen vorliegen. Dabei bedient er sich der Vorgehensweise der natürlichen Selektion. Zuerst werden zufällige Chromosomen generiert, die mögliche Lösungen für ein Problem darstellen. Diese Chromosomen werden dann nach ihrer *Fitness*, also wie gut sie geeignet sind das Problem zu lösen, sortiert. Die fittesten Chromosomen werden dann selektiert, um mit diesen durch Rekombination die Berechnung fortzuführen, ähnlich wie bei der Fortpflanzung. Da dieser Schritt die Diversität der Chromosomen senkt, ist es möglich, dass ein Chromosom mutiert und sich so zufällig minimal verändert.
Übertragen auf L-Systeme bedeutet dies, aus mehreren zufällig generierten L-Systemen diejenigen auszuwählen, welche die schönste Musik abbilden, um mit diesen durch Rekombination und Mutation die Berechnung fortzuführen. Das entscheidende Problem an dieser Vorgehensweise ist die Fitnessfunktion, welche angibt, wie gut ein L-System klingt. Dies von einem Computer entscheiden zu lassen, ist sehr schwer. Um dieses Problem zu lösen gibt es nach dem aktuellen Forschungsstand mehrere Ansätze. Am effektivsten ist es, die Arbeit der Fitnessfunktion einem Menschen zu überlassen der die gewünschten L-Systeme auswählt. Dies ist jedoch eine langwierige Aufgabe, mit der nur wenige L-Systeme betrachtet werden können. Eine automatisierte Auswahl wird in [14] vorgenommen: Hier geht es darum, L-Systeme auf ihre Ästhetik in Bezug auf Pflanzen Morphologie zu bewerten. Dabei bedient man sich mehrerer physikalischer Parameter, wie bilaterale Symmetrie, die Fähigkeit Licht aufzunehmen und strukturelle Stabilität. Solche Parameter können auch für Musik gefunden werden, wobei hier deutlich subjektivere Parameter zum Tragen kommen. In [5] wird die Fitnessfunktion lediglich als Möglichkeit benutzt, um musikalische Variation zu manipulieren und weniger, um ein konkretes Ergebnis zu erhalten. [14]

3.1.4.1 Implementierung in SuperCollider

In der Implementierung in SuperCollider wird versucht L-Systeme zu finden, die eine bestimmte Melodie abbilden. Eine grundlegende Implementierung des Genetischen Algorithmus existiert bereits von Bozkurt [13] im NatureToolkit mit den Namen GAWorkbench. Es gibt drei Funktionen, die für eine konkretes Problem angepasst werden müssen: die *RandomChromosomeFunc*, die *FitnessFunc* und die *MutationFunc*. Die *RandomChromosomeFunc* erzeugt eine bestimmte Anzahl an zufälligen Chromosomen für die weitere Berechnung. Mittels der *FitnessFunc* wird ein Wert bestimmt, der angibt, wie fit jedes Chromosom ist. Die *MutationFunc* verändert ein Chromosom minimal. Im Folgenden ist beschrieben, was bei einer Implementation der Funktionen mit L-Systemen zu berücksichtigen ist.
RandomChromosomeFunc:
Bei der Generierung von zufälligen L-Systemen gibt es mehrere Variablen, die betrachtet werden müssen. Das Alphabet, aus dem das L-System besteht, benötigt für eine musikalische Interpretation die Buchstaben F, f, $+$ und $-$. Zusätzliche Buchstaben wie bspw. X und Y die nur der Ersetzung dienen und keine direkten Auswirkungen auf die Turtle-Grafik haben, sind ebenfalls möglich. Aus dem Alphabet können dann

das Axiom und die Produktionsregeln gebildet werden. Hier gibt es erneut mehrere Variablen, so kann das Axiom und der Vorgänger und Nachfolger jeder Produktionsregeln komplett zufällig oder mit einer minimalen/maximalen Länge generiert werden. Ebenfalls muss die Anzahl an Produktionsregeln festgelegt werden. Um den Algorithmus so effektiv wie möglich zu halten, ist es wichtig, die Anzahl der veränderbaren Variablen so gering wie möglich zu halten, so dass der Algorithmus die wenigen verbleibenden optimieren kann. Eine Grundstruktur eines L-Systems sollte also bereits vorliegen. Das Axiom und die Produktionsregeln werden in einer Liste gespeichert.
FitnessFunc:
Die $FitnessFunc$ berechnet für jedes L-System einen $FitnessScore$, der umso höher ausfällt, je näher das L-System an die gewünschte Melodie herankommt. Um den $FitnessScore$ zu bestimmen muss, das L-System zuerst simuliert werden. Dazu müssen die Produktionsregeln n mal auf das Axiom angewendet werden, aus der entstandenen Zeichenfolge eine Grafik generiert werden und aus der Grafik die Noten inklusive Dauer berechnet werden. Das Array mit den Noten und ihrer Dauer kann dann von der $FitnessFunc$ evaluiert werden. Hier ist es möglich für bestimmte vorkommende Noten, Notenfolgen oder Rhythmen Punkte zu vergeben.
MutationFunc:
In der $MutationFunc$ wird ein L-System auf minimale Weise verändert. Am einfachsten und intuitiven erscheint es bei dem Nachfolger der Produktionsregeln einen Buchstaben aus dem Alphabet hinzuzufügen, zu ersetzen oder zu löschen. Ebenso sind die gleichen Änderungen am Vorgänger der Produktionsregel oder am Axiom denkbar.

```
1  (
2  ~gaInstance = GAWorkbench(~poolSize, ~randChromFunc, ~fitFunc,
       ~mutFunc);
3  ~numGenerations = 20;
4  ~poolSize = 100;
5  ~gaInstance.mutationProb = 0.05;
6  ~numGenerations.do(
7   {
8    ~gaInstance.rateFitness;
9    ~gaInstance.crossover;
10  }
11 );
12 ("Ergebnis: " + ~gaInstance.genePool[0]).postln;
13 )
```

Listing 4: SuperCollider genetischer Algorithmus

Auf Listing 4 ist der grundsätzliche Ablauf des Genetischen Algorithmus zu sehen. Der Befehl *gaInstance* erstellt eine neue Instanz eines Genetischen Algorithmus mit der jeweiligen *poolSize* und den erwähnten Funktionen. Die *poolSize* gibt an, mit wie vielen Chromosomen der Algorithmus starten soll, in diesem Beispiel werden 100 zufällige L-Systeme generiert. Die *mutationProb* gibt die Wahrscheinlichkeit an, mit der ein L-System beim *crossover* mutiert. *numGenerations* steht für die Anzahl an Generationen, also wie oft der Algorithmus durchgeführt werden soll. Je höher die Anzahl an Generationen, desto besser wird die Fitness des Ergebnisses, aber umso mehr Rechenzeit wird benötigt.
Innerhalb der *do* Schleife findet der eigentliche Algorithmus statt: Jedem Chromosom bzw. L-System wird mit *rateFitness* ein $fitnessScore$ zugeteilt, der dann beim *crossover* verwendet wird. Beim *crossover* findet eine Selektion mittels tournament

selection und eine Rekombination statt, um dann aus den besten L-Systemen neue sog. $offsprings$ zu erhalten. An dieser Stelle ist es nun möglich, dass die $offsprings$ mutieren, damit eine gewisse Vielfalt an L-Systemen erhalten bleibt.
Eine naive Implementation in SuperCollider blieb ohne erwähnenswert Erfolg. Für die Implementation wurden die folgenden Parameter verwendet: eine $fitnessFunc$ die L-Systeme mit Dreiklängen bevorzugt, zufällige erzeugte L-Systeme mit zwei Produktionsregeln, die jeweils den Vorgänger X und Y besitzen und eine zufällige Zeichenfolge als Nachfolger mit einer Länge von 1-8. Das Axiom ist als X festgelegt und die Rekursionstiefe auf drei. Bei der Durchführung des Algorithmus war es oft der Fall, das sich der $fitnessScore$ nicht signifikant verbesserte oder sogar schlechter wurde. Dies könnte daran liegen, dass der Algorithmus zu wenig oder zu viel Freiraum bei der Optimierung der Variablen besitzt, die benötigten Funktionen für dieses Problem schlecht gewählt sind oder das eigentliche Dreiklangsproblem zu allgemein formuliert ist. Um mit diesem Algorithmus sinnvolle Ergebnisse zu erhalten, ist es erforderlich, alle Funktionen auf eine konkrete Problemstellung anzupassen.

3.2 Zellulärer Automat

Zelluläre Automaten sind mathematische Modelle für dynamische Systeme und wurden 1963 von John von Neumann und Stanislaw Ulam bekannt gemacht. Ein Zellulärer Automat besteht aus einer gitterförmigen Anordnung von Zellen, in der jede Zelle endlich viele Zustände oder Werte annehmen kann. Die Zustände aller Zellen entwickeln sich gleichzeitig in diskreten Zeitabständen gemäß einer Übergangsregel, die den eigenen Zustand und den Zustand der benachbarten Zellen als Parameter erhält. Der Status aller Zellen eines Zellulären Automaten nach einem bestimmten Zeitabstand wird als Generation bezeichnet. Über mehrere Generationen hinweg kann das Verhalten des Zellulären Automaten beobachtet werden. Der Raum der Zellen kann sich von der ersten bis in die dritte Dimension erstrecken. Ein bekannter zweidimensionaler Zellulärer Automat ist John Conways: „$Game\ of\ Life$“. Im Folgenden wird näher auf die Klasse der elementaren eindimensionalen Zellulären Automaten eingegangen und wie sich diese mit dem Verfahren von Reiners [7] und Cutajar [8] für eine musikalische Darstellung abbilden lassen.

3.2.1 Eindimensionaler Zellulärer Automat

Ein eindimensionaler Zellulärer Automat ist ein Zellulärer Automat mit einer unendlichen Raum- und Zeitdimension. In praktischen Implementationen muss die Raumdimension dann begrenzt werden, hierfür gibt es mehrere Möglichkeiten, eine wird in Kapitel 3.2.4 näher beschrieben. Wie erwähnt hängt der Zustand einer Zelle von den Zuständen der Nachbarzellen ab. Bei einem eindimensionalen Zellulären Automaten sind das die Zellen links und rechts von der betrachteten Zelle. Der neue Zustand lässt sich dann mit folgender Formel berechnen:

$$a_i^{(t)} = \boldsymbol{F}[a_{i-r}^{(t-1)}, a_{i-r+1}^{(t-1)}, ..., a_i^{(t-1)}, ..., a_{i+r}^{(t-1)}] \tag{5}$$

$a_i^{(t)}$ ist der Zustand der Zelle i zum Zeitpunkt (t), berechnet über die Funktion $\boldsymbol{F}$, die die Zustände aller Zellen im Radius r zum Zeitpunkt $(t-1)$ berücksichtigt. Der Zustand jeder Zelle ist eine ganze Zahl zwischen 0 und $k-1$, wobei k die Anzahl der möglichen Zustände darstellt. $\boldsymbol{F}$ ist eine willkürliche Funktion die die Regel des Zellulären Automaten spezifiziert.

Für diese Arbeit bedeutend sind die elementaren Zellulären Automaten, welche die einfachsten nichttrivialen Automaten darstellen. Elementare Zelluläre Automaten besitzen $k = 2$ Zuständen, oftmals als „Tod“ und „Lebendig“ bezeichnet, und $r = 1$ berücksichtigte Nachbarzellen. Der Zustand einer Zelle zum Zeitpunkt (t) hängt also von der Zelle selbst, $a_i^{(t-1)}$ und den beiden direkten Nachbarzellen, $a_{i-1}^{(t-1)}$ und $a_{i+1}^{(t-1)}$, zum Zeitpunkt $(t-1)$ ab.
Auf Abbildung 11 sind die ersten 16 Generationen eines Zellulären Automaten zu sehen. Die Raumdimension erstreckt sich horizontal und wird rechts und links von einem Rand begrenzt, der immer den Zustand 0 besitzt. In der obersten Zeile ist die erste Generation zu sehen, die vorgegeben wird. Danach wird die Generation der nächsten Zeiteinheit $(t+1, t+2, ...)$ unter der vorherigen dargestellt.

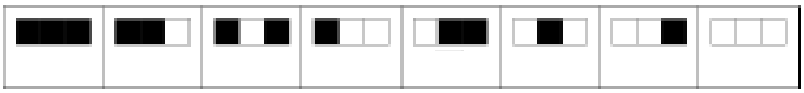

Abbildung 7: Die acht möglichen Zustandskombinationen eines eindimensionalen Zellulären Automaten[7]

Die zwei möglichen Zustände jeder Zelle eines elementaren Zellulären Automaten lassen sich in, $k^{(2r+1)} = 8$ mögliche Kombinationen der Zustände innerhalb der Zweiernachbarschaft anordnen. Auf Abbildung 7 ist der Zustand 0 weiß und der Zustand 1 schwarz gekennzeichnet. Auf der Abbildung sind die acht verschiedenen Anordnungsmöglichkeiten zu sehen, die für alle Übergangsregeln in dieser Reihenfolge vorliegen. Für diese acht möglichen Zustandskombinationen kann man nun wie auf Abbildung 8 Übergangsregeln definieren. So bedeutet die erste Regel ganz links:
$F_{150}[1, 1, 1] = 1$,
In Worten: Ist der Zustand der Zelle selbst und die Zustände der benachbarten Zellen 1, so ist auch der Zustand in der nächsten Generation 1. Die übrigen Übergangsregeln verfolgen das gleiche Prinzip. Da die Reihenfolge der acht möglichen Nachbarzustände immer gleich bleibt, ändert sich nur der Ergebniszustand, also die zweite Reihe auf Abbildung 8. Alle Ergebniszustände zusammen ergeben die Binärzahl $10010110_2 = 150_{10}$. Jede mögliche Übergangsregel für elementare Zelluläre Automaten lässt sich auf diese Weise als Dezimalzahl darstellen. Mit $k^{k^{2r+1}}$ lässt sich die Anzahl an verschiedenen Regeln berechnen [15]. Mit anderen Worten: Da es zwei verschiedene Zustände und acht verschiedene Möglichkeiten gibt diese anzuordnen, existieren $2^8 = 256$ verschiedene elementare Zelluläre Automaten.
Von diesen 256 Automaten sind viele das Spiegelbild eines anderen, wobei einige auch äquivalent zu ihrem Spiegelbild sind. Nur wenige Automaten zeigen interessantes Verhalten, bei vielen Automaten wird ein Satz von Zuständen periodisch wiederholt.
Das Verhalten der Automaten ist auch von der Startkonfiguration abhängig. Meistens wird hierfür der Zustand einer einzige Zelle auf 1 gesetzt, die dann den Mittelpunkt des Automaten markiert (siehe Abbildung 11). Aber auch andere, zufallsgenerierte Startkonfigurationen sind möglich. [16]

3.2.2 Musikalische Interpretation

Für die musikalische Interpretation von Zellulären Automaten beschränke ich mich auf die Verfahren von Reiners und Cutajar. Beide Verfahren haben eine einfache Grundidee und sind gut zu implementieren.

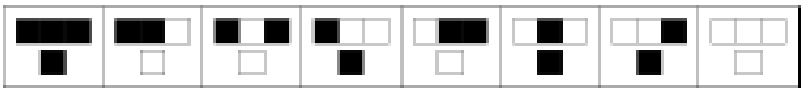

Abbildung 8: Regel 150 [7]

Verfahren von Reiners [7]:
Das Verfahren von Reiners ordnet allen Zellen einer Generation eine ganzzahlige Dezimalzahl zu. Hierfür wird jede Zelle einer Generation als Binärziffer interpretiert. Alle Zellen zusammen ergeben dann eine Binärzahl, die schließlich zu einer Dezimalzahl umgewandelt werden kann. Diese Dezimalzahl kann dann von *Pbind* als Midinote interpretiert werden. Da ein eindimensionaler Zellulärer Automat nach links und rechts unbeschränkt ist, muss jede Generation von Zellen sortiert werden, um einen Anfang und ein Ende der Binärzahl zu finden. In den hier betrachteten Zellulären Automaten, ist meistens der Zustand genau einer Zelle in der Startkonfiguration 1. Diese Zelle ist die Mitte, bzw. der Nullpunkt und damit der Anfang der Binärzahl. Falls in der Startkonfiguration mehrere Zellen den Zustand 1 besitzen, muss ein Mittelpunkt mit angegeben werden. Danach werden die Zellen links und rechts von der Mitte mit steigendem Radius an die Binärzahl angefügt. Die Zellen rechts des Nullpunktes werden mit positiven Indizes beschrieben und die Zellen links mit negativen. Ein mögliches Ordnungsschema könnte also so aussehen: $0, -1, 1, -2, 2, ...$. Damit würden die Zelle links des Nullpunktes bevorzugt werden, sie kommt immer vor der entsprechenden Zelle rechts dran. Diese Bevorzugung wird Bias genannt und ist auch entsprechend in die andere Richtung möglich. Auf Abbildung 9 ist das Verfahren Beispielhaft für eine Generation eines Zellulären Automaten abgebildet. Ausgehend von dem Mittelpunkt des Automaten, werden neue Ziffern nach und nach der Binärzahl angefügt. Dabei werden die Zellen links bevorzugt. Die berechnete Binär- bzw. Dezimalzahl entspricht dann $0101110_2 = 46_{10}$. Da es möglich ist, das die vom Verfahren berechneten Zahlen größer als 127 werden, müssen sie auf den Wertebereich der Midinoten (0-127) beschränkt werden. Dies wird mit einer modulo Operation erreicht. Durch diese modulo Operation werden aber praktisch nur die ersten sieben Zellen jeder Generation betrachtet ($1111111_2 = 127_{10}$), die Information aller anderen Zellen geht verloren. Insbesondere bei fortschreitenden Generationen gehen so viele Informationen des Zellulären Automaten verloren.
Eine Variante des Verfahrens ergibt sich bei der Berechnung der kumulativen Summe der Binärzahlen jeder Generation. Die Binärzahl jeder Generation wird auf die vorherige hinzu addiert und die so entstehende kumulative Summe wird dann als finale Midinote verwendet. Damit werden allzu häufige Tonsprünge unterbunden und die einzelnen Generationen beeinflussen sich gegenseitig.
[7]
Verfahren von Cutajar [8]:
Beim Verfahren von Cutajar werden aus jeder Generation eines eindimensionalen Zellulären Automaten neben der Tonhöhe auch Tondauer und Lautstärke berechnet. Dafür werden 13 zusammenhängende Zellen eines eindimensionalen Zelluläre Automaten betrachtet, wobei jede Zelle, bzw. Gruppe von Zellen einen musikalischen Parameter bestimmt. Auf Abbildung 10 ist eine Generation eines Zellulären Automaten zu sehen. Die farbige Umrandung kennzeichnet welche Zellen im Folgenden für die Berechnung verwendet werden.

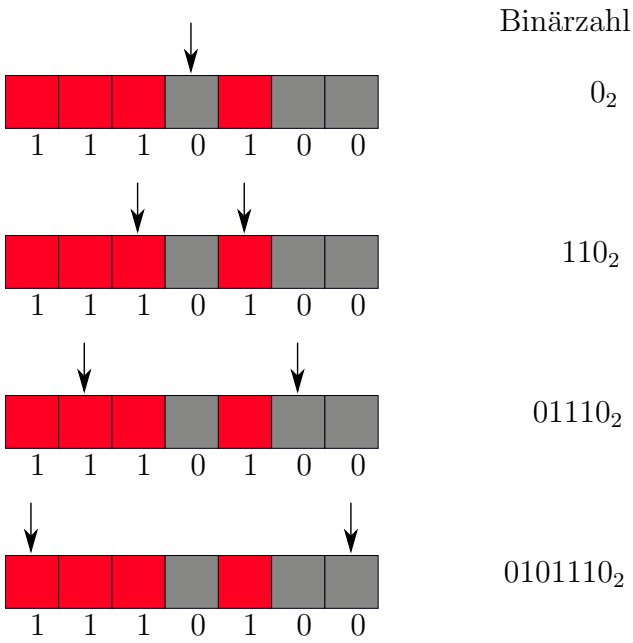

Abbildung 9: Verfahren von Reiners mit links bevorzugt

Die Zelle innerhalb des blauen Rahmens gibt an, ob die Note gespielt wird (1), oder eine Pause (0). Wenn die Note gespielt wird, geben die nächsten sieben Zellen im lila Rahmen an, welche Note gespielt wird. Die sieben Zellen stehen dabei für die sieben Töne einer Skala. Im Beispiel auf Abbildung 10 käme dann der vierte oder siebte Skalenton in Frage. Bei mehreren möglichen Tönen wird ein zufälliger ausgewählt. Ein Problem entsteht, falls keine der sieben Zellen als Skalenton in Frage kommt, also keine den Zustand 1 hat. Hier ist nicht definiert was passieren soll. Es wäre dann möglich die letzte Note noch einmal zu spielen, eine Standard Note zu spielen, oder doch eine Pause einzufügen. Die verwendete Skala kann frei gewählt werden und auch innerhalb der Interpretation einer Regel verändert werden. Die nächsten beiden Zellen im grünen Rahmen geben die Lautstärke des Tones an. Es gibt vier verschiedene Lautstärken, die ansteigend von 00 bis 11 gehen. Die letzten drei Zellen im roten Rahmen geben die Dauer des Tones, bzw. der Pause an. Hier gibt es wieder vier verschiedene Möglichkeiten von 00x bis 11x (x steht für eine 1 oder 0), die dann von halben bis Sechzehntel Noten gehen. [8]

Abbildung 10: eine Generation eines Zellulären Automaten [8]

Hörbeispiel Regel 30 Play

Regel 30 mit dem Verfahren von Cutajar, feste Startkonfiguration, Orgelklang

3.2.3 Komplexitätsklassen von Zellulären Automaten

Das Verhalten von einzelnen elementaren Zellulären Automaten lässt große Unterschiede erkennen. Um dies zu gliedern, gibt es nach Wolfram vier verschiedene Klassen für Zelluläre Automaten. Dabei betrachtet Wolfram den Status, den der Automat mit einer

zufälligen Startkonfiguration nach einer langen Zeit erreicht. Bei anderen Startkonfigurationen kann ein Automat seine Klasse ändern. Klasse 1 stellt hierbei die einfachsten und Klasse 4 die komplexesten Zelluläre Automaten dar. Interessant ist, inwiefern sich dieses unterschiedliche Verhalten in der Musikgenerierung bemerkbar macht. Alle Hörbeispiele sind mit zufälliger Startkonfiguration und dem vierstimmigen Verfahren von Reiner erstellt, soweit nicht anders angegeben. Der Rhythmus ist festgelegt auf $\frac{1}{4}$ pro Note. Für die Ausgabe wird die *Pbind* Klasse mit Orgelklang verwendet.

Klasse 1 (uniformes Verhalten):
Zelluläre Automaten in dieser Klasse entwickeln sich nach endlich vielen Schritten in immer den gleichen homogenen Fixstatus, indem keine weitere Änderung mehr passiert und alle Zellen den gleichen Zustand haben. Die Funktion **F** liefert für alle möglichen Parameter denselben Wert. Das Ergebnis des Automaten ist dabei komplett unabhängig von der Startkonfiguration, diese Information geht somit verloren.
⇒ z.B. Regel 0, 8, 40, 136

Hörbeispiel Regel 40	Play

Automaten aus der Klasse 1 entwickeln sich immer in einen Fixstatus, in den sich keine Zelle verändert, was bedeutet das auch immer dieselbe Note erklingt.

Klasse 2 (periodisches Verhalten):
Automaten in dieser Klasse entwickeln simple Strukturen, die entweder dauerhaft sind, oder sich in kleinen Perioden wiederholen. Der Zustand einer Zelle an einem beliebigen Zeitpunkt hängt von einer geringen Anzahl von Zellen der Startkonfiguration ab.
⇒ z.B. Regel 4, 37, 56, 108

Hörbeispiel Regel 108	Play

Klasse 2 Automaten bilden bereits einfache Melodien ab, die an sich interessant klingen, aber unendlich wiederholt werden. Je nach Automat kann es länger oder kürzer dauern bis sich ein die generierte Sequenz wiederholt.

Klasse 3 (chaotisches Verhalten):
Klasse 3 Zelluläre Automaten entwickeln sich zu chaotische Strukturen, die sich nie wiederholen. Der Zustand einer Zelle zu einem beliebigen Zeitpunkt hängt von einer stetig ansteigenden Anzahl an Zellen der Startkonfiguration ab. Ein einfacher Algorithmus kann den Wert einer Zelle mit der gegebenen Startkonfiguration berechnen. Zelluläre Automaten in dieser Klasse werden in der Kryptographie benutzt und als Zufallszahlengenerator.
⇒ z.B. Regel 18, 30, 60

Hörbeispiel Regel 60	Play
Hörbeispiel Regel 60 (2)	Play

Zum Vergleich Regel 60 (2) mit einer lebenden Zelle als Startkonfiguration. Dann wahrscheinlich Klasse 2.

Klasse 4 (komplexes Verhalten):
In dieser Klasse sind die komplexesten Zellulären Automaten. Sie entwickeln komplexe Muster mit Strukturen, die sich räumlich und zeitlich bewegen und haben ein unvorhersehbares Verhalten. Das bedeutet unter anderem, das es keinen endlich Algorithmus gibt, der entscheiden kann, ob sich ein Klasse 4 Zellulärer Automat in einen Nullzustand entwickelt oder nicht (Halteproblem für Turingmaschinen). Der Zustand einer

Zelle zu einem beliebigen Zeitpunkt kann nur durch Simulation, also das komplette Durchlaufen der Entwicklung aller vorherigen Zellen, bestimmt werden. Für Regel 110 ist bewiesen, dass sie Turing vollständig ist, d.h. es ist damit möglich jeden beliebigen Algorithmus abzubilden.[16][17][18]
⇒ z.B. Regel 41, 54, 110

Hörbeispiel Regel 110	Play

Klasse 3 und Klasse 4 Automaten sind schwerer voneinander zu unterscheiden. Während bei Klasse 3 rein zufällige Noten erklingen, liegt bei Klasse 4 ein komplexeres Verhalten zugrunde. Dies innerhalb ein paar Sekunden herauszuhören ist sehr schwierig. Wenn man die grafische Interpretation eines Klasse 4 Automaten betrachtet, sind sehr schnell komplexe Strukturen erkennbar, die bei Klasse 3 Automaten fehlen. Das liegt daran, dass eine Vielzahl an Generationen sofort ersichtlich ist, bei der musikalischen Interpretation fehlt dieses Element, da sich der Automat erst nach und nach erschließt.

3.2.4 Programmierung und visuelle Darstellung mit SuperCollider

Als Basis für die Verfahren von Reiner und Cutajar dient die Funktion
$ca(int\ generations, int\ length, int\ rule, boolean\ randConfig)$,
die einen Zellulären Automaten mit Rand erstellt. Der Zelluläre Automat wird als mehrdimensionales Array mit *generations* Dimensionen erstellt, von dem jedes Array *length* Länge besitzt. Das erste und letzte Element jedes Arrays stellt hierbei den Rand dar, der immer den Zustand 0 besitzt. Als Startzustand wird das mittlere Feld des ersten Arrays auf den Zustand 1 gesetzt, falls $randConfig = false$ ist, ansonsten wird das erste Array zufällig befüllt. Die restlichen Arrays werden dann nach der Regel *rule* befüllt.
Das Verfahren von Reiners wurde mit der Funktion $reiners(ca\ c, int\ bias)$ und
$reinerscumulativ(ca\ \ c, int\ \ bias)$ realisiert. *bias* gibt dabei an, ob zuerst die Werte rechts oder links von dem Mittelpunkt, zur Berechnung der Binärzahl betrachtet werden sollen. *reiners* liefert dann ein Array mit den entsprechenden Tonhöhen in Midinoten. *reinerscumulativ* ist die gleiche Funktion mit der kumulativen Variante. Aus der Kombination von *bias* links, rechts und kumulativ oder nicht, ergeben sich insgesamt vier verschiedene Möglichkeiten. Jede einzelne dieser Möglichkeiten wird dann einem *Pbind* übergeben und alle *Pbinds* zusammen können dann mit *Ppar* gleichzeitig abgespielt werden. So ist eine Mehrstimmigkeit mit vier Stimmen möglich.
Das Verfahren von Cutajar wurde mit der Funktion $cutajar(ca\ c)$ implementiert. Die Funktion berechnet für ein gegebenen Zellulären Automaten ein dreidimensionales Array mit Tonhöhe in *degree*, Tondauer und Tonlautstärke.
Zur Visualisierung des Zellulären Automaten wird die *window* Klasse von SuperCollider benützt. Die Zellen sind als eine Reihe von Buttons dargestellt, wobei Grau für tote Zellen und Rot für lebende steht. Der Zelluläre Automat wächst dann im Rhythmus der Musik nach unten (siehe Abbildung 11).

4 Evaluation

Das Ziel dieser Arbeit war es Verfahren zur Musikgenerierung mit L-System und Zellulärem Automaten zu untersuchen. Die untersuchten Verfahren sollen nun in Hinblick auf ihr musikalisches Ergebnis evaluiert werden. Die Evaluation der entstandenen Musik stellt für alle Verfahren aus mehreren Gründen ein schwieriges Unterfangen dar.

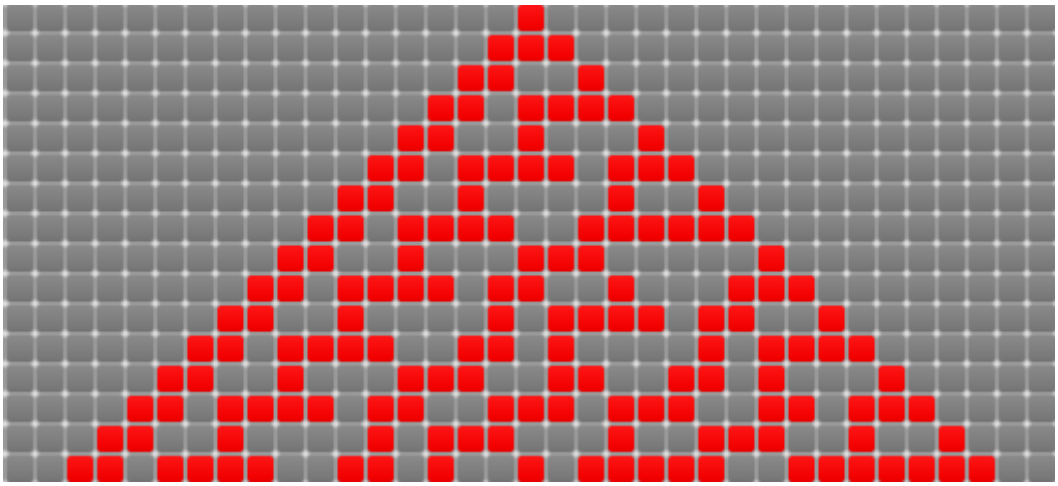

Abbildung 11: Regel 30

Jedes Verfahren besitzt einen experimentellen Charakter. Das bedeutet, dass untersucht wird, was möglich ist, und weniger, ob etwas möglich ist oder ob ein konkretes Ziel erreicht werden kann. Insofern werden die musikalischen Ergebnisse der Verfahren nicht genau bewertet, sondern die Verfahren werden auf ihre Chancen und Möglichkeiten hin evaluiert. Generell ist Kunst bzw. in diesem Fall Musik schwer zu bewerten, da keine objektiven Bewertungskriterien vorliegen und letztendlich jede Bewertung subjektiv geprägt ist. Ebenfalls ist es nicht möglich die generierte Musik mit wohlbekannter populärer Musik zu vergleichen, da hierbei ein komplett anderer Ansatz vorliegt. Ein populäres Musikstück ist oftmals über mehrere Monate von mehreren beteiligten Personen aufwändig produziert. Dies mit dem innerhalb von Sekunden generierten Werk eines bis jetzt noch relativ einfachen Algorithmus zu vergleichen, lässt es nicht zu eine qualifizierte Aussage zu treffen.
Letztendlich können die Verfahren nur unter sich verglichen werden und mit anderen möglichen Interpretationen der zugrunde liegenden Algorithmen. Die grafische Interpretation des L-Systems ist deutlich eindrucksvoller als seine musikalische. Das kann daran liegen, dass bei den Grafiken das komplette Ergebnis mit allen Entwicklungsstufen sofort sichtbar ist, während bei der klanglichen Darstellung das Ergebnis erst Schritt für Schritt abgespielt wird. Der fraktale Charakter ist damit bei der grafischen Interpretation leichter zu erkennen. Es muss aber auch beachtet werden, dass für die grafische Interpretation wesentlich mehr Forschung existiert als für die musikalische.

4.1 Vergleich von Reiners, Cutajar und Prusinkiewicz

In dieser Arbeit wurden drei verschiedene Verfahren zur Musikgenerierung vorgestellt, zwei auf Basis eines eindimensionalen Zellulären Automaten und eines auf Basis eines 0DL-System. Diese drei Verfahren stellen dabei jeweils eine individuelle Lösung des Mapping-Problems dar, also wie die Rohdaten der generativen Algorithmen in musikalische Parameter transformiert werden können. Im Folgenden werden die Verfahren getrennt voneinander untersucht und es wird insbesondere auf die Anwendbarkeit auf verschiedene Algorithmendaten, die Möglichkeiten zur Erweiterung und die etwaigen Probleme eingegangen.

4.1.1 Verfahren von Prusinkiewicz

Das Verfahren von Prusinkiewicz besitzt eine einfach verständliche und intuitive Grundidee und ist das Standardverfahren für L-Systeme. Mit diesem Verfahren werden sowohl Tonhöhe als auch Tondauer aus der Turtle-Grafik einer L-System Zeichenfolge berechnet. Jedes L-System kann mit dem Verfahren musikalisch interpretiert werden, solange die für das Verfahren benötigte Turtle-Grafik eine Winkeländerungsgröße von 90° besitzt. Dies stellt sicher, dass die zu interpretierende Grafik nur aus horizontalen und vertikalen Linien besteht. Ein häufiges Problem entsteht, sobald sich die zu interpretierende Turtle-Grafik zu weit nach oben oder unten bewegt. Dann liegen die resultierenden Tonhöhen außerhalb des für Menschen auditiv erfassbaren Bereichs.
Das Verfahren bietet einige Möglichkeiten zur Erweiterung. Für viele durch L-Systeme erzeugte Grafiken ist die Branching-Technik elementar, um bspw. realistische Abbildungen von Pflanzen zu generieren. Eine musikalische Interpretation der Branching Symbole, um polyphone Musik zu generieren wäre denkbar [4]. Weiterhin wäre es möglich, zusätzlich Turtle-Grafiken mit anderen Winkeländerungsgrößen als 90° musikalisch zu interpretieren; die entstehenden angewinkelten Linien könnten dann als Glissandi interpretiert werden [19]. Ebenfalls ist das Verfahren nicht nur auf D0L-Systeme beschränkt, es wäre genauso möglich ein stochastisches L-System zu verwenden, um einen Zufallsfaktor mit in die Komposition einfließen zu lassen.
Durch das ausschließliche Verwenden von Tonhöhen einer bestimmten Skala, klingt die erzeugte Musik oftmals harmonisch. Die fraktalen Eigenschaften des L-Systems sind durch die häufig transponierten Melodiewiederholungen und den oft gleichbleibenden Rhythmus gut wahrnehmbar. Die Vielfalt der generierten Musik ist dabei immens und theoretisch unendlich, genauso wie die Vielfalt der Turtle-Grafiken. Jede beliebige Melodie kann relativ einfach mit dem Verfahren durch ein entsprechendes L-System konstruiert werden.

4.1.2 Verfahren von Reiners

Das Verfahren von Reiners besitzt auch eine einfache und intuitive Grundidee, ist aber nicht so weit entwickelt wie das Verfahren von Prusinkiewicz. Mit diesem Verfahren wird lediglich die Tonhöhe aus einem eindimensionalen Zellulären Automaten berechnet. Die Tondauer muss extern festgelegt werden und ist nicht Teil des Algorithmus. Alle 256 elementaren Zellulären Automaten können mit beliebigen Startkonfigurationen von dem Verfahren musikalisch interpretiert werden.
Das Verfahren bietet viele Möglichkeiten zur Erweiterung und Verbesserung. Damit die berechneten Tonhöhen auf den hörbaren Bereich beschränkt sind, wird eine modulo Operation angewandt. Dadurch gehen viele Informationen verloren und nur ein kleiner Teil eines Zellulären Automaten wird tatsächlich musikalisch interpretiert. Die übrigen Zellen könnten benutzt werden, um daraus die Tondauer zu berechnen, damit auch der Rhythmus von dem Verfahren generiert wird. Ebenso ist es notwendig die Startkonfiguration der Automaten anzupassen, um die Struktur der Musik besser steuern zu können. Durch spezielle Startkonfigurationen könnte es möglich sein, bestimmte Melodien von dem Verfahren erzeugen zu lassen.
Da die erzeugten Tonhöhen von keiner Skala begrenzt werden, sondern vom Bereich der Midinoten, besitzen viele Zelluläre Automaten einen atonalen und unharmonischen Charakter. Trotzdem sind aber Strukturen und Melodiefolgen erkennbar, da sich dieselben Tonhöhen wiederholen. Außerdem werden durch die durchgeführte modulo Operation Tonsprünge begünstigt. Sobald die Tonhöhe den kritischen Bereich überschreitet,

wird wieder von vorn begonnen.
Durch die Beschränkung auf 256 elementare Zelluläre Automaten und dadurch, dass nur ein kleiner Teil der Startkonfiguration aufgrund der modulo Operation für das Verfahren relevant ist, ist die Vielfalt der erzeugten Musik beschränkt. Dies wird dadurch begünstigt, dass viele Klasse 1 und Klasse 2 Zelluläre Automaten für eine Musikgenerierung unbrauchbar sind, da sich die generierte Tonfolge nicht ändert. Mit dem Verfahren ist es nicht möglich, jede beliebige Melodie zu erzeugen. Durch die im vorherigen Absatz beschriebenen Erweiterungen könnten diese Beschränkungen aufgehoben werden.

4.1.3 Verfahren von Cutajar

Das Verfahren von Cutajar besitzt keine vergleichbar simple und intuitive Herangehensweise wie die beiden anderen Verfahren. Es ist das einzige Verfahren, welches neben Tonhöhe und Tondauer auch die Tonlautstärke aus dem Zellulären Automaten berechnet. Der Versuch, die musikalischen Dynamiken mit in den Algorithmus einzubeziehen, ist lohnend, aber noch ausbaufähig. Da für jeden Ton eine separate Lautstärke berechnet wird, entsteht schnell ein konfuser Gesamteindruck. Besser wäre es hier über einen Zeitraum eine ähnliche Tonlautstärke zu halten, um eine bestimmte Tonstimmung zu erzeugen.
Das Verfahren ist nicht für alle Zellulären Automaten sinnvoll anwendbar. Probleme entstehen bei rechtsbündigen Automaten (z.B. Regel 60), deren Zelle ganz links immer den Zustand 0 besitzt. Hier wird jede Generation des Automaten von dem Verfahren als Pause interpretiert und es erklingt kein einziger Ton. Ein häufiges Problem ist weiterhin, dass die ersten Generationen vieler Zellulärer Automaten in der Zelle ganz links den Zustand 0 besitzen (siehe Abbildung 11) und somit die generierte Melodie immer mit Pausen beginnt. Außerdem kann es passieren, dass Teile der generierten Musik unnatürlich lange Pausen haben. Eine vorherige Filterung der Daten des Zellulären Automaten wäre für dieses Verfahren sinnvoll, um die Daten so anzupassen, dass oben genannte Probleme nicht entstehen. Ähnlich wie beim Verfahren von Reiners wird nur ein kleiner Teil von 13 Zellen des Zellulären Automaten für die Berechnung verwendet. Die Informationen aller anderen Zellen gehen verloren.
Beim Verfahren von Cutajar sind oftmals keine konkreten Melodieelemente zu erkennen, da sich fast nichts wiederholt. Das liegt daran, dass ein zufälliger Ton ausgewählt wird, sobald mehrere zur Auswahl stehen. Mehrere Töne stehen zur Auswahl, wenn mehr als eine Zelle an zweiter bis achter Zelle den Zustand 1 besitzt, was bei vielen Zellulären Automaten der Fall ist. Ebenfalls ist die Melodie auf eine Oktave beschränkt.

5 Schluss

In dieser Arbeit wurde die Algorithmische Komposition mit dem L-System und dem Zellulären Automaten untersucht. Diesbezüglich wurden die Grundlagen von L-System und Zellulären Automaten in Bezug auf ihre Verwendung in einem musikalischen Kontext beschrieben. Aus den vielen verfügbaren Verfahren zur musikalischen Interpretation der Algorithmen wurden drei Verfahren genauer ausgeführt, in SuperCollider implementiert, visualisiert und evaluiert.
Die Freiheit, die diesen Verfahren bei der Musikgenerierung gegeben wurde, führte wie gewünscht zu neuer und interessanter Musik, die einen starken eigenen Charakter besitzt. Teilweise führte diese Freiheit aber auch zu unstrukturierter Musik, in der keine

konkreten Elemente zu erkennen sind. Abschließend lässt sich sagen, dass ein gewisses Maß an Regeln und Struktur von Nöten ist, um interessante und komplexe, aber auch verständliche Musik zu erzeugen, die das Unbekannte mit dem Bekannten verbindet.
Kurzum bieten die Verfahren viel Raum für weitere Verbesserungen und Forschung. Generell könnte es nötig sein, die Grenzen der traditionellen Musik aufzuheben, um die Möglichkeiten der Algorithmischen Komposition nicht einzuschränken. Die Benutzung der Midinoten als Ausgabeformat für die erzeugen Kompositionen stammt von herkömmlichen Instrumenten, die auf diesen Tonbereich beschränkt sind. Für einen Computer dagegen ist es möglich, auch Töne zwischen den konventionellen Halbtonschritten zu spielen, sogenannte Mikrotöne. Hier besteht der Bedarf nach einem neuen Notensystem, das den Möglichkeiten der Algorithmischen Komposition gerecht wird.
Für L-Systeme ist es möglich noch andere Grammatiken für die Musikgenerierung zu benutzen. Stochastische L-Systeme funktionieren nach dem Zufallsprinzip und liefern bei jeder Anwendung ein anderes Ergebnis und damit auch andere Musik. Die Produktionsregeln von kontextsensitiven L-Systemen betrachten bei der Ersetzung zusätzlich auch Zeichen oder Zeichenfolgen nach oder vor dem zu ersetzenden Zeichen. Hiermit lassen sich mehr strukturspezifische Produktionsregeln erzeugen.
Der in Kapitel 3 vorgestellte Genetische Algorithmus bedarf noch einiger Anpassung, um relevante Ergebnisse mit einem L-System zu erzielen. Ergiebiger könnte eine Anwendung mit einem Zellulären Automaten sein. Hier könnte der Genetische Algorithmus Startkonfigurationen für bestimmte Übergangsregeln ermitteln. Besonders geeignet erscheint Regel 110, da sie Turing vollständig ist. Es müsste möglich sein mit ihr jede Melodie durch eine entsprechende Startkonfiguration abzubilden. Eine Implementation des Genetischen Algorithmus wäre hierbei wesentlich einfacher als beim L-System, da bedeutend weniger Parameter berücksichtigt werden müssen. Beim L-System führte die Fülle an beeinflussbaren Variablen (Axiom, Anzahl der Produktionsregeln, Vorgänger und Nachfolger der Produktionsregeln, Iterationstiefe und Alphabet) zu ungenauen Anweisungen für den Algorithmus und schlussendlich zu wenig aussagekräftigen Ergebnissen. Beim Zellulären Automaten dagegen müssen lediglich die Länge und die Zustände der Startkonfiguration verändert werden. Wenige veränderbare Variablen können von dem Genetischen Algorithmus schneller und effizienter optimiert werden.
Die in dieser Arbeit erläuterten Musikverfahren zu L-System und Zellulären Automat geben lediglich einen kleinen Einblick in die Möglichkeiten der Algorithmischen Komposition. Es existieren viele weitere Verfahren und auch komplett andere Herangehensweisen, um Algorithmische Kompositionen zu erzeugen. Besonders vielversprechend erscheint hier die Anwendung von neuronalen Netzen, mit denen es bisher unter anderem gelungen ist, überzeugende Bach Choräle zu komponieren [20]. Es gibt demnach beinahe keine Grenzen für Algorithmische Komposition.

Literatur

[1] Makis Solomos, Agostino Di Scipio, Peter Hoffmann, and Horacio Vaggione. CELLULAR AUTOMATA IN XENAKIS' MUSIC. THEORY AND PRACTICE. *Benoît Gibson (Portugal)*, 2005.

[2] Jon McCormack. Grammar based music composition. *Complex systems*, 96:321–336, 1996.

[3] Stephanie Mason and Michael Saffle. L-Systems, Melodies and Musical Structure. *Leonardo Music Journal*, 4(1):31–38, 1994.

[4] Przemyslaw Prusinkiewicz. Score Generation with L-systems. *Proceedings of the International Computer and Music Conference*, pages 455–457, 1986.

[5] S Manousakis. Musical L-systems. *MA thesis, Institute of Sonology,< modular-brains. net/support/SteliosManousakis-Musical_L-systems. pdf*, 2006.

[6] Peter Worth and Susan Stepney. Growing Music : musical interpretations of L-Systems Plants to Music : finding a rendering. *Applications on Evolutionary Computing*, 3449/2005(3):545–550, 2005.

[7] Paul D. Reiners. Cellular automata and music, 2004.

[8] Simon Cutajar. Interpreting one dimensional cellular automata as music.

[9] Stephen Wolfram. Wolframtones, 2005.

[10] Eduardo Reck Miranda. Cellular Automata Music: An Interdisciplinary Project. *Interface*, 1993.

[11] James McCartney. SuperCollider, 1996.

[12] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants (The Virtual Laboratory)*. 1991.

[13] Batuhan Bozkurt. NatureToolkit, 2010.

[14] Gabriela Ochoa. On genetic algorithms and Lindenmayer systems. *Parallel Problem Solving from Nature, PPSN V*, 1998.

[15] Genaro J. Martinez. A note on elementary cellular automata classification. *Journal of Cellular Automata*, 2013.

[16] Stephen Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 1984.

[17] Matthew Cook. Universality in Elementary Cellular Automata. *Complex Systems*, 2004.

[18] Dave Burraston, Ernest Edmonds, Dan Livingstone, and Eduardo Reck Miranda. Cellular Automata in MIDI based Computer Music. *Proceedings of the International Computer Music Conference*, 2004.

[19] G. L. Nelson. Real time transformation of musical material with fractal algorithms. *Computers and Mathematics with Applications*, 1996.

[20] Gaetan Hadjeres, Francois Pachet, and Frank Nielsen. DeepBach: a Steerable Model for Bach Chorales Generation. 2017.