technische universität
dortmund

Bachelorarbeit

Erzeugung perkussiver Muster zur Echtzeitbegleitung von Jazz-Soli

Fabian Ostermann
November 2015

Gutachter:
Prof. Dr. Günter Rudolph
Dr. Igor Vatolkin

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Algorithm Engineering (LS11)
http://ls11-www.cs.tu-dortmund.de

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation und Hintergrund

Einmal ein Solo wie Miles Davis[1] über *So What*[2] oder John Coltrane[3] über *Giant Steps*[4] spielen zu können – danach streben alle Jazzmusiker der Welt, seien es Amateure oder Profis. Doch das bedeutet, jeden Tag Stunden mit seinem Instrument zu verbringen.

Ein Instrument zu beherrschen, ist ein Handwerk und es dabei zur Perfektion zu bringen, benötigt viel Zeit. Will man auch die Kunst des Improvisierens erlernen, wird es noch komplizierter, denn Improvisieren passiert im Kopf. Es ist eine wahnsinnige Rechenleistung, die unser Gehirn vollbringen muss, um ein Solo zu denken. Dabei geht es nebensächlich um die Frage nach der nächsten Note. Ein improvisierender Musiker muss ein Gefühl für Melodien entwickelt haben, um eine solche unmittelbar zu erfinden. Dabei ist es wichtig, schon etwas in die Zukunft planen zu können, allerdings auch das bereits Gespielte im Hinterkopf zu behalten. Meist wird in einer Combo (3–5 Musiker) oder gar in einem großen Ensemble wie einer Big Band (mehr als 10 Musiker) gespielt. Das Solo bekommt dadurch eine untermalende Begleitung, die Rhythmus und Harmonik vorgibt, an die der Improvisierende zwar nicht gebunden ist, auf die er dennoch reagieren muss. So benötigt er ein hohes Verständnis für Takte, Tempi und Timing sowie für Tonarten, Akkorde und Skalen. Zeitgleich erfolgt dabei die Ausführung des Erdachten am Instrument.

Um eine solch komplexe Fähigkeit zu erlernen, werden Harmonie- und Rhythmuslehre studiert, die den Fokus auf ein mathematisches Verständnis von Musik legen und dem Schüler das Kategorisieren von Gehörtem und das Anwenden von Regeln ermöglichen. Zudem werden Akkorde und Skalen am Instrument geübt, bis die Abläufe automatisiert sind.

Es gibt eine Vielzahl von Ideen, welche abstrakten Übungen dem Improvisierenden helfen können. Die Kernaufgabe bleibt aber das Spielen auf dem Instrument in einer Band. Die von Jazzschülern am häufigsten genutzte Übung, ist das Spielen zu einem sogenannten *Playalong*.

Ein Playalong ist die Aufnahme eines Musikstücks, auf der die zu übende Stimme nicht enthalten ist. Der Musiker kann diese zu der Aufnahme spielen, um das Ensemblespiel zu simulieren.

[1] Miles Davis (1926–1991), ein bedeutender amerikanischer Jazz-Trompeter
[2] Erstveröffentlichung auf dem Album *Kind of Blue* (1959)
[3] John Coltrane (1926–1967), ein bedeutender amerikanischer Jazz-Saxophonist
[4] Erstveröffentlichung auf dem Album *Giant Steps* (1960)

Die unter Jazzschülern verbreitetste Sammlung ist die *Jamey Aebersold Play-A-Long Serie* [Aebersold, 1967] mit mittlerweile über 100 Bändern. Ein Band besteht aus etwa einem Dutzend Stücken, meist Jazzstandards[5], deren Noten in einem Heft beiliegen und deren Begleitung auf einen dazugehörigen Tonträger ohne Melodiestimme aufgenommen wurde. Die Aufnahmen bestehen immer aus Schlagzeug, Bass und Klavier, wobei letztere auf die beiden Stereokanäle verteilt sind, sodass mittels Panoramaregler auch Pianist oder Bassist ausgeblendet werden können. Der Schüler kann zu dieser Rhythmusgruppe die Melodie spielen und eben auch über die im Notenheft angegebenen Akkorde improvisieren.

Im Laufe der Zeit kamen viele weitere Playalong-Sammlungen heraus. Zu nennen sind Hal Leonards *Real Book Playalongs* [Hal Leonard, 2000] oder *learnjazzstandards.com* [Hughes, 2010], die mit einer Vielzahl von frei zugänglichen Playalongs auf Youtube[6] vertreten sind. Mithilfe von Playalongs haben mehrere Jahrzehnte lang Schüler das Spielen in einer Jazzcombo allein im eigenen Keller üben können.

Im Jahr 1990 wurde die Software *Band-in-a-Box* von PG Music [Gannon, 1990] veröffentlicht und schnell als Alternative zu Playalongs genutzt. Das Programm generiert zu einer gegebenen Akkordfolge eine genretypische Begleitung für mehrere Instrumente. Dabei können die Taktart eingestellt und verschiedene Genres wie Jazz, Pop, Rock oder Latin ausgewählt werden. Das Ergebnis ist ein Arrangement, das als Begleitung zu einem Solo fungieren kann. Einen Vorteil gegenüber Audio-Playalongs bringt die dynamische Erzeugung. Akkordfolgen und Tempi können beliebig verändert werden. Auch die Begleitung ist nicht mehr vorhersehbar. Eine Audioaufnahme ist dagegen unveränderlich, was dazu führt, dass der Übende nach einiger Zeit weiß, wie die Band an gewissen Stellen reagiert. Die Algorithmen von Band-in-a-Box nutzen eine Zufallskomponente.

Auch das automatische Erzeugen von Soli ist mit Band-in-a-Box möglich, bringt allerdings weit weniger interessante Ergebnisse hervor als beispielsweise die Arbeit von Al Biles (siehe GenJam in Kapitel 5).

Mittlerweile gibt es auch mobile Anwendungen, die automatisch Playalongs aus Akkordfolgen erzeugen. Das beliebteste Produkt hier ist *iReal Pro* [Biolcati, 2008].

Bewertet man die Übesituation des Jazzschülers bis hierhin, erkennt man eine positive Entwicklung in der realitätsnahen Simulation des Combospiels. Es wird allerdings auch deutlich, worin bis hierhin der große Unterschied zwischen dem Spiel mit einer Maschine und dem Spiel mit echten Menschen besteht. Die Mitmusiker können auf das Spiel des Solisten reagieren. Sie können ihr Spiel anpassen oder auch gewollt kontrastreich gestalten. Sie können den Solisten beeinflussen und ihn so in seiner Kreativität unterstützen. Sie hören zu.

[5] Jazzstandards sind besonders bekannte und verbreitete Jazz-Kompositionen, die von Jazzmusikern regelmäßig gespielt und neu interpretiert werden.

[6] `https://www.youtube.com/Learnjazzstandards` – abgerufen am 13.11.2015

Aus dieser Beobachtung entstand die Idee für diese Arbeit. Es soll ein intelligentes System entwickelt werden, das einen Jazzsolisten rhythmisch begleiten, sein Solo in Echtzeit analysieren und darauf reagieren kann.

1.2. Aufbau der Arbeit

Der erste Teil der Arbeit enthält die formalen Grundlagen, die notwendig sind, um den Entwurf und die Arbeitsweise des implementierten Systems zu verstehen. In Kapitel 2 wird das Konzept der evolutionären Algorithmen erklärt. Es stellt eine Abstraktion der natürlichen Vorgänge der biologischen Evolution zu einem algorithmischen Schema dar. Das System nutzt einen evolutionären Algorithmus zur Variation von Schlagzeug-Patterns, die aus rhythmischen perkussiven Mustern bestehen. Die musikalischen Grundlagen finden sich in Kapitel 3. Dort werden das Schlagzeug und seine Instrumente sowie die Funktionsweise eines Metronoms erklärt. Außerdem wird die klassische *Drum Machine* vorgestellt, die rhythmische Begleitungen statisch generiert und deren Funktionsprinzip für das implementierte System adaptiert wurde. Außerdem erfolgt ein theoretischer Einblick in die Welt der Jazz-Musik. Dabei wird die moderne Notenschrift als bekannt vorausgesetzt. Kapitel 4 führt die genutzte Schnittstelle zwischen den Musikinstrumenten und dem System, das *Musical Instrument Digital Interface*, ein. Nach einem kurzen Überblick über seine Geschichte werden die genaue Spezifikation sowie eine API zum Verarbeiten von MIDI-Daten in der Programmiersprache *Java* erläutert. Der formale Teil der Arbeit endet mit einem Abschnitt über die Umwandlung von Audiosignalen in MIDI-Daten.

Zu Beginn des zweiten Teils, der die praktische Umsetzung des Systems beschreibt, werden verwandte Arbeiten (Kapitel 5) vorgestellt. In Kapitel 6 definiert die geplanten Ziele der Arbeit. Daraus geht in Kapitel 7 ein Systementwurf hervor. Der Entwurf beinhaltet eine Erläuterung der Kommunikation des Systems mit seiner Umgebung und eine Einteilung in drei autonome Module. Eine Erklärung der einzelnen Module erfolgt in den folgenden drei Kapiteln. Das Input-Modul (Kapitel 8) steuert die Kommunikation des Systems mit dem solierenden Musiker. Der evolutionäre Algorithmus ist Hauptbestandteil des Genetic-Moduls (Kapitel 9). Das Playback-Modul (Kapitel 10) wandelt die generierten Patterns in MIDI-Daten um und sendet sie an einen Synthesizer. Im letzten Kapitel des praktischen Teils wird der Erfolg der Implemetierung evaluiert. Die Evaluation (Kapitel 11) erfolgt anhand von Tests mit aktiven und passiven Testpersonen, die ihre persönlichen Erfahrungen und Eindrücke schildern und das System nach musikalischen Gesichtspunkten bewerten. Auch eine Liste aller zur Laufzeit veränderbaren Parameter des Systems findet sich dort.

Der letzte Teil enthält in Kapitel 12 eine Zusammenfassung der Arbeit und in Kapitel 13 Ideen zur Erweiterung des vorgestellten Systems und zur Ergänzung der Konzepte.

Im Anhang A findet sich eine kurze Einführung der Benutzeroberfläche des Systems, in der eine Zuordnung der im praktischen Teil beschriebenen Aktionen des Benutzers zu grafischen Elementen erfolgt.

Teil I.

Formale Grundlagen

2. Evolutionäre Algorithmen

Das Gebiet der evolutionären Algorithmen (EA) umfasst Optimierungsverfahren, die von der Evolution und ihrem Prinzip der natürlichen Selektion inspiriert wurden. Die EA nutzen die Stochastik, um die zufälligen Ereignisse in der Natur zu simulieren. Zudem sind sie metaheuristisch, das heißt, sie finden eine annäherungsweise Lösung zu einem Optimierungsproblem.

Es existieren eine ganze Reihe von Varianten dieser Algorithmen. Die vier klassischen sollen kurz genannt werden. Der historisch bekannteste Entwurf geht auf die *genetischen Algorithmen* (GA) von Holland [1975] zurück. Die Funktionsweise dieses Entwurfs kommt dem biologischen Vorbild am nächsten. Die *Evolutionsstrategien* (ES) von Schwefel [1965], Rechenberg [1965] und Bienert [1967] dienen der experimentellen Optimierung, das heißt der Optimierung auf Grundlage von Erfahrungswerten. Das *evolutionäre Programmieren* (EP) von Fogel u. a. [1966] erzeugt deterministische endliche Automaten, die sich einem Problem anpassen. Diese drei Varianten wurden zeitnah und unabhängig voneinander entwickelt. Die vierte klassische Variante ist das *genetische Programmieren* (GP) von Koza [1992]. Er nutzte GP beispielsweise, um symbolische Regression zu lösen.

Die verschiedenen Eigenschaften dieser vier Varianten werden im Abschnitt 2.2 zum besseren Verständnis der Elemente in Ansätzen erklärt. Im folgenden Abschnitt soll zunächst die Entstehung der biologischen Evolutionstheorie erläutert werden.

2.1. Biologie

Die Theorie der Evolution geht auf den britischen Naturforscher Darwin (1809–1882) zurück. Auf einer Schiffsreise von 1831 bis 1836 in Südamerika erlangte er tiefgreifende Erkenntnisse über die Artenvielfalt und ihre Entstehung. Nachdem er einige Jahre hart an der naturwissenschaftlichen Formulierung seiner Theorie gearbeitet hatte, erschien 1859 *Über die Entstehung der Arten* [Darwin, 1859], das den Grundstein der modernen Evolutionsbiologie legte.

Laut Darwin liegt der Grund für die Artenvielfalt in der Unterschiedlichkeit der Individuen einer Population. Dabei sind einige Unterschiede für die Individuen in ihrem Lebensraum von Vorteil, andere von Nachteil. Die besser angepassten Individuen überleben und pflanzen sich mit attraktiven Partnern (Selektion der Eltern) fort. Dabei geben sie ihre Gene weiter und vermischen sie (Rekombination). Die resultierenden Nachkommen bilden die Basis einer nächsten stärkeren Generation (Selektion der Nachkommen).

Was Darwin noch nicht erklären konnte, war der Mechanismus der Genmanipulation. Die moderne Biologie definiert den *Phänotyp* als das äußere Erscheinungsbild und das Verhalten eines Lebewesens. Dieser ist maßgebend für die Wahrscheinlichkeit des Bestehens eines Individuums im Überlebenskampf. Der Phänotyp wird wiederum bestimmt durch den sogenannten *Genotyp*, den inneren Bauplan eines Lebewesens, der im Zellkern kodiert ist. Eine chemische Rekombination ist verantwortlich für die Veränderung des Genotyps bei der sexuellen Reproduktion. Außerdem kann es zu Mutationen im Erbgut kommen, die (meist durch fehlerhafte Reproduktion) den Genotyp wahllos verändern.

2.2. Struktur der evolutionären Algorithmen

Evolutionäre Algorithmen dienen der Optimierung. Ein Suchraum voller möglicher Lösungskandidaten (*Individuen*) wird nach einer optimalen oder hinreichend guten Lösung abgesucht. Eine Menge von Kandidaten (*Population*) wird dabei so lange verändert, bis eine gewünschte Lösung enthalten ist. Die Veränderung wird erreicht durch Selektion, Rekombination, Mutation und Verdrängung im Zyklus. Abbildung 2.1 zeigt den schematischen Ablauf eines EA, dessen Bestandteile in den folgenden Abschnitten erklärt werden.

Dieser Abschnitt folgt Miranda u. Biles [2007, S. 1–10, S. 39–48].

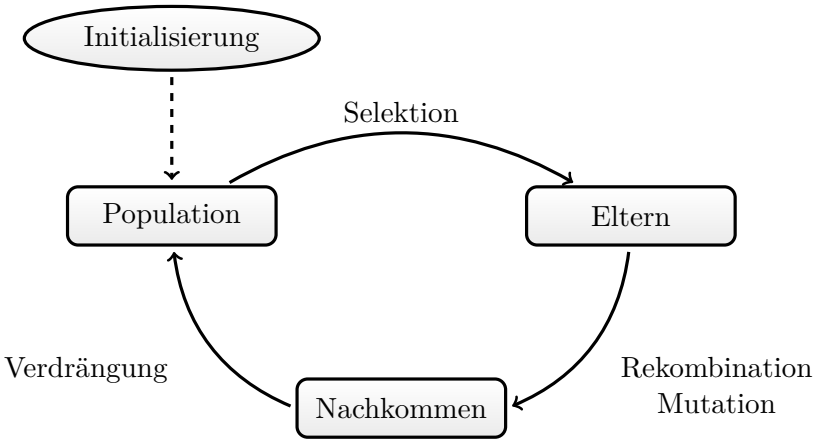


Abbildung 2.1.: Schematische Darstellung eines evolutionären Algorithmus

2.2.1. Repräsentation der Individuen

Die Wahl der genetischen Repräsentation ist essentiell für das Gelingen einer Optimierung. Sie macht einen entscheidenden Unterschied bei den Varianten der EA.

Der Phänotyp eines Individuums ist eine mögliche Lösung des definierten Problems. Der Genotyp ist seine Kodierung im Programm. Die sinnvolle Wahl der Kodierung hat maßgeblichen Einfluss auf die Konstruktion der Operatoren sowie auf die Komplexität und den Erfolg der Berechnung.

Bei den GA wird ein binärer, diskreter Genotyp $\{0,1\}^n$ genutzt. Diese Variante kommt ihrem Vorbild der Evolution damit sehr nah. Der natürliche Genotyp, die DNA, besteht aus wenigen chemischen Verbindungen, die bausteinhaft zusammengefügt sind. Die Rekombination und Mutation der Genotypen ist hier besonders einfach. Jedoch muss für eine Bewertung der Individuen meist der Phänotyp erzeugt werden. Üblicherweise ist n festgelegt, kann jedoch auch veränderlich sein, wodurch sich vielfältige Möglichkeiten bei der Reproduktion ergeben.

Bei den ES sind der Problem- und der Suchraum identisch. Das bedeutet, der Genotyp stellt eine konkrete Lösung des Problems dar. Die Individuen können beispielsweise durch $\mathbb{N}^n$, $\mathbb{Z}^n$ oder $\mathbb{R}^n$ kodiert sein. Die Operatoren sind dabei insgesamt einfacher zu entwerfen, jedoch kann es durch lokale Extrema bei der Suche nach einem globalen Extremum zu Problemen in der Abdeckung des Suchraums kommen.

Es sind auch komplexe Repräsentationen möglich. So nutzt die GP Bäume und die EP endliche Automaten.

2.2.2. Initialisierung

Bevor der Zyklus eines evolutionären Algorithmus beginnen kann, muss eine erste Population erzeugt werden. Bei den ES genügt ein einzelnes Individuum mit zufälligem Genotyp, sodass die Population später an Größe zulegen muss. Klassischerweise ist sie allerdings von fixer Größe und wird mit zufälligen Lösungskandidaten, die typischerweise gleichverteilt über den Suchraum erzeugt werden, gefüllt.

Zielführender und damit mittlerweile die gängige Vorgehensweise ist die intelligente Erzeugung von Individuen mithilfe von problemspezifischem Wissen beispielsweise durch Markovketten oder künstliche neuronale Netze. So sind von Beginn der Berechnung an gute Lösungen vorhanden, die nicht mehr von Grund auf generiert, sondern bereits fein optimiert werden können. Besonders für (zeitkritische) Approximierungsprobleme zeigt dieser Ansatz einen klaren Vorteil.

2.2.3. Selektion der Eltern

Ist eine Population vorhanden, so ist der erste Schritt des EA die Selektion der Eltern zur Reproduktion. Bei den ES wird schlicht eine gleichverteilt zufällige Selektion aus der gesamten Population vorgenommen.

Üblicherweise sollen allerdings, wie beim Vorbild der Evolution, höherwertige Individuen den minderwertigeren vorgezogen werden. Die Güte eines Individuums wird dazu, wie in Abschnitt 2.2.7 näher erläutert wird, evaluiert. Dabei wird jedem Individuum ein Fitnesswert zugeordnet. Je höher der Wert, desto besser löst der Kandidat das zugrundeliegende Problem.

Die simpelste Methode ist die *Bestenselektion*. Dazu werden die n_B hochwertigsten Individuen einer Population ausgewählt.

Von Holland stammt der Vorschlag der *Rouletteradselektion*, einer fitnessproportionalen Selektion, die ihren Namen der Idee verdankt, jedem Individuum einen bestimmten Bereich eines Roulettekessels zuzuweisen, der proportional seiner Fitness in der Population entspricht. So werden hochwertige Individuen mit höherer Wahrscheinlichkeit gewählt, jedoch besteht für jedes Individuum der Population weiterhin die Chance auf Reproduktion.

Bei der *Rangselektion* wird die Wahrscheinlichkeit der Selektion durch ihren Rang innerhalb der Population bestimmt. Große Fitnessunterschiede werden so entschärft. Beispielsweise könnte das stärkste Individuum mit der Wahrscheinlichkeit $\frac{1}{2}$ gewählt werden, das Individuum an Rang 2 mit $\frac{1}{4}$, das dritte mit $\frac{1}{8}$, usw. Auch kann bei Bedarf die Fitnessberechnung entfallen. Es genügt die Sortierung der Individuen zueinander.

Gleiches gilt auch für die *Turnierselektion*. Es werden k_T Individuen zu einem Turnier gegeneinander gezogen. Das stärkste Individuum gewinnt und wird selektiert. Es werden n_T Turniere ausgetragen. Dabei ist n_T die Anzahl der zur Reproduktion bestimmten Individuen. Je größer k_T ist, desto eher werden starke Individuen der Gesamtpopulation zur Reproduktion ausgewählt. Bei kleinem k_T haben auch schwache Individuen eine gute Chance auf eine Reproduktion.

2.2.4. Rekombination

Die Rekombination ist die nach dem Vorbild der Evolution geprägte Zusammenführung zweier Elterngenome zu einem neuen Genom. Als Genom wird eine konkrete Ausprägung des Genotyps bezeichnet. Mathematisch betrachtet wird eine Menge von Genomen auf eine Menge von neuen Genomen abgebildet. Eine solche Abbildung nennt sich Rekombinations-Operator.

Der klassischste aller Rekombinations-Operatoren ist die *Ein-Punkt-Rekombination* (siehe Abbildung 2.2). Dazu werden zwei Genome an einer zufälligen Stelle p getrennt und mit dem jeweiligen Gegenstück des Partners zusammengefügt. Es entstehen zwei neue Genome, die jeweils bis zur Stelle p_x dem einen Elterngenom gleichen und nach p_x dem jeweils anderen.

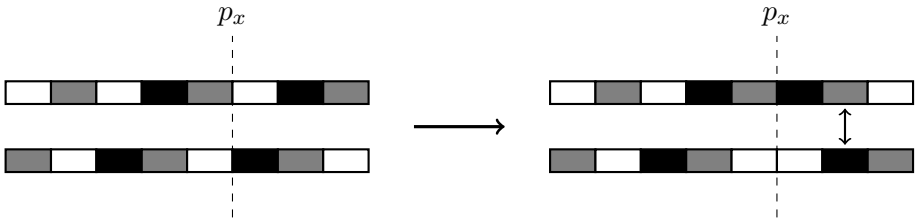


Abbildung 2.2.: Ein-Punkt-Rekombination

Eine Verallgemeinerung stellt die *k-Punkt-Rekombination* dar, die die Genome an k verschiedenen Stellen auftrennt und vermischt. Sie wurde bereits von Holland für die GAs genutzt.

Sind variable Längen des Genotyps erlaubt, darf sich die Position der Schnittstellen bei den Elterngenomen unterscheiden.

Ist das Genom in $\mathbb{R}^n$, ist auch eine numerische Vereinigung zum Beispiel über das arithmetische Mittel der einzelnen Komponenten möglich.

Das Ziel der Rekombination ist die Erzeugung neuer und vor allem höherwertiger Individuen. So kann auch hier problemspezifisches Wissen die Ergebnisse der Rekombination verbessern.

Es ist allerdings auch möglich auf Rekombination zu verzichten und die Veränderung der Population der Mutation zu überlassen, wie es beispielsweise bei den ES üblich ist.

2.2.5. Mutation

Die Mutation dient der Evolution dazu, in einen Raum vorzudringen, der durch die Rekombination der Individuen einer Population nicht erreicht werden kann.

Die einfachste Art einer Mutations-Funktion ist die wahllose Veränderung eines Genoms auf Bit-Ebene. An einer zufälligen Stelle wird ein Bit umgekehrt (siehe Abbildung 2.3). Eine weitere typische Variante ist das Drehen eines jeden Bits mit einer gewissen Wahrscheinlichkeit p_m. Dabei wird p_m meist sehr klein gewählt, um die Mutation zu entschärfen. Auch kann p_m mit der Laufzeit des Algorithmus abnehmen, da die Approximation eines Optimums erreicht werden soll und so die Veränderung mit zunehmender Zeit feiner werden muss.

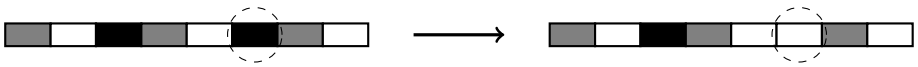


Abbildung 2.3.: Punkt-Mutation

Ist der Genotyp nicht aus $\{0,1\}^n$, ist möglicherweise eine geringe Veränderung durch Bit-Mutation nicht zu erreichen. Dann muss der Genotyp in seiner Repräsentation manipuliert werden, beispielsweise durch Addition einer Zufallszahl auf die einzelnen Komponenten, wenn der Genotyp aus $\mathbb{R}^n$ ist.

Auch hier ist problemspezifisches Wissen von Vorteil, um intelligente Mutations-Funktionen zu definieren.

2.2.6. Selektion der Nachkommen

Die Selektion der Nachkommen ist der letzte Schritt im Zyklus eines evolutionären Algorithmus. Dabei werden die durch Rekombination und Mutation erzeugten Individuen in die Population zurückgeführt. Üblich ist hierbei eine fixe Anzahl n an Individuen, die als minderwertige Individuen die Population verlassen müssen und für die neuen Platz machen. Die Selektion erfolgt dabei nach den selben Verfahren wie im Abschnitt 2.2.3 bereits erläutert. Ist n gleich der Größe der Population, so nennt man sie einfach Generation.

Es können sich auch Individuen mehrerer Generationen in einer Population befinden und miteinander reproduzieren, wobei grundsätzlich die älteren Platz für die neuen machen.

2.2.7. Evaluation

Die Evaluation ist die Berechnung der Fitness der Individuen. Die Fitness beschreibt, wie gut ein Lösungskandidat das Optimierungsproblem löst. Das Problem dabei ist das Definieren der Fitness-Funktion, die einen Phänotyp auf einen Fitnesswert abbildet. Zudem ist die Evaluation die häufigste Berechnung im Zyklus und meist ausschlaggebend für das Zeitverhalten des Algorithmus.

Automatische Fitnessberechnung

Bei der automatischen Fitnessberechnung wird versucht einen Algorithmus zu finden, der die Güte eines Individuums mithilfe des gegebenen Phänotyps berechnet. Kann der Phänotyp anhand der Problemstellung getestet werden, so ist die Berechnung trivial.

Es gibt allerdings Probleme, bei denen ein Computer die Berechnung nicht ohne Weiteres durchführen kann, und zwar dann, wenn der Phänotyp komplex ist (siehe Abschnitt 2.3). Die Maschine muss erst lernen das Material zu bewerten.

Eine Möglichkeit ist die Verwendung eines Regelwerks. Der Entwurf erfolgt aufgrund von Erfahrungen und kann verschiedene Arten der Analyse verwenden. Es wird demnach ein wissensbasiertes System entworfen.

Mithilfe von neuronalen Netzen kann die Fitnessberechnung durch Methoden der künstlichen Intelligenz gelöst werden. Neuronale Netze sind trainierbar und agieren als Stellvertreter eines Experten, um möglichst genauso zu entscheiden, wie dieser es täte.

Interaktive Fitnessberechnung

Ist es nicht möglich, die Fitnessberechnung hinreichend gut mit automatischen Verfahren zu lösen, bleibt nur noch die Bewertung der Individuen durch einen Menschen. Dies ist der Fall, wenn kein klar definiertes Problem gelöst werden soll, sondern beispielsweise ein ästhetisches (siehe Abschnitt 2.3).

Der Vorteil dieser Methode liegt klar in der hoch entwickelten Problemlösungsfähigkeit des Menschen. Jedoch liegen die Probleme, die interaktiv gelöst werden, meist außerhalb eines scharf definierten Optimierungsproblems. So wird ein Mensch bei einer ästhetischen Entscheidung niemals eine logische, objektive Lösung generieren können. Bei unterschiedlichen Personen kommt der Algorithmus deshalb zu unterschiedlichen Ergebnissen. Zudem muss ein Mensch jedes Individuum, das im EA erzeugt wird, bewerten. Die Bewertung kostet viel mehr Zeit als bei einer maschinellen Lösung. So kann nur eine kleine Zahl von Individuen evaluiert werden. Interaktive Fitnessberechnung findet daher häufig im künstlerisch kreativen Bereich Anwendung.

2.3. Anwendungsgebiete

Für unterschiedlichste Probleme der Optimierung wurden bereits evolutionäre Algorithmen entworfen. Zum Ende dieses Kapitels soll nun ein kurzer Einblick in das breite Anwendungsgebiet der evolutionären Algorithmen ohne Anspruch auf Vollständigkeit gegeben werden. Es werden einige Arbeiten genannt, ohne diese näher zu erläutern. Dazu sei auf die angegebene Literatur verwiesen.

Forschung

Der genetische Algorithmus NEAT [Stanley u. Miikkulainen, 2002] erzeugt künstliche neuronale Netze und wird für das maschinelle Lernen verwendet.

Mithilfe von evolutionärer Spieltheorie [Axelrod, 1987] wird versucht Kooperationsstrategien für das Gefangenendilemma zu evolvieren.

Das junge Teilgebiet der evolutionären Bildverarbeitung [Ebner, 2008] nutzt EA zur Rekonstruktion von Bildinformationen sowie zur Optimierung von Algorithmen.

In der Bioinformatik [Fogel u. Corne, 2002] werden EA zur Analyse großer Datenmengen verwendet und selbstverständlich simuliert die Evolutionsbiologie die Evolution [Ray, 1991], um ein besseres Verständnis der natürlichen Vorgänge zu erlangen.

Wirtschaft

Die Industrie verwendet EA zur Optimierung ihrer Produkte sowie zur automatischen Verifikation zum Beispiel von Software [Sanchez u. a., 2012]. Spieltheoretische Analysen ermöglichen den Einsatz an der Börse [Chen, 2002]. Selbst in der Landwirtschaft können EA zur Optimierung von Planungsarbeiten eingesetzt werden [Mayer, 2001].

Kunst

Evolutionäre Kunst ist ein Teilgebiet der generativen Kunst. Dabei generiert der Computer große Teile selbstständig und unterstützt den Künstler in der Erstellung komplexer Kunstwerke unter Benutzung eines EA. Die Phänotypen der erzeugten Individuen stellen dabei typischerweise ein Bild, ein Musikstück oder ein Video dar.

Ziel der evolutionären Kunst ist das Erzeugen eines kreativen Prozesses durch eine Maschine. Häufig wird eine interaktive Fitness-Evaluation genutzt, um die menschliche Kreativität und das ästhetische Empfinden in den Algorithmus einfließen zu lassen.

Eine der ersten Arbeiten zum Thema grafische Computer-Kunst stammt von Todd u. Latham [1992].

Im Jahre 2005 fand der evolutionäre Algorithmus von Collomosse u. Hall [2005] eine größere Aufmerksamkeit. Er generiert aus einer gegebenen Fotografie automatisch ein ästhetisch ansprechendes Gemälde, indem ein Bild (Phänotyp) als Kette von Pinselstrichen (Genotyp) abstrahiert wird.

Musik

Die Musik bietet ebenfalls unzählige Möglichkeiten für den Einsatz von evolutionären Algorithmen. In Miranda u. Biles [2007] findet sich eine Reihe von Anwendungen für evolutionäre Musiksysteme: Audio-Technologie, Sound Design, Komposition, Improvisation und interaktive Musiksysteme.

Ein Überblick über verwandte Arbeiten dieser Arbeit findet sich in Kapitel 5.

3. Musikalische Grundlagen

3.1. Das Schlagzeug

Das Schlagzeug ist eine Zusammenstellung verschiedener Schlaginstrumente, die durch ihre Anordnung gleichzeitig bespielbar sind. Der Schlagzeuger kann mithilfe beider Füße und zweier meist hölzerner Schlagstöcke (Sticks) in beiden Händen bis zu vier Instrumente gleichzeitig spielen. Das moderne Schlagzeug besteht aus Trommeln und Becken [Ziegenrücker, 1977, S. 277].

Die Trommeln werden in drei Klassen unterteilt. Die größte ist die liegende Bass Drum, die mit der Fußmaschine gespielt wird. Mittig befindet sich die Snare, die ursprünglich bei militärischer Marschmusik Verwendung fand. Dazu kommen 2–4 unterschiedlich große Toms, die in verschiedenen Tonhöhen gestimmt werden.

Die Becken werden in vier Klassen unterteilt. Die Hihat ist das komplexeste und jüngste Instrument. Zwei Becken werden übereinander montiert und können durch eine Ständerkonstruktion mit dem Fuß zusammengedrückt werden. Die Crash-Becken haben zwischen 14" und 18" und werden mit der Stickkante gespielt. Die Ride-Becken sind meist noch größer und werden mit der Stickspitze gespielt. Zusätzlich können noch Effekt-Becken wie Splash (besonders kleines Becken) oder China (nach außen gewölbtes Becken) hinzukommen.

Abbildung 3.1 zeigt eine typische Zusammenstellung für ein Jazz-Schlagzeug und die musikalische Notation der einzelnen Instrumente.

3.2. Metronom

Ein Metronom dient der Vorgabe eines gleichmäßigen Tempos. Klassische mechanische Metronome zeigen oder schlagen mithilfe eines Fadenpendels oder einer Feder gleichmäßig Viertelnoten in einem einstellbaren Tempo an. Deshalb lautet die Maßeinheit *Beats-per-Minute* (bpm). Bei 60bmp dauert eine Viertel demnach genau 1 Sekunde.

Es gibt auch moderne elektronische Metronome, die ohne Probleme auch andere Notenwerte und Taktarten vorgeben. Die Maßeinheit bpm mit Viertelnoten pro Minute wurde trotzdem unverändert übernommen.

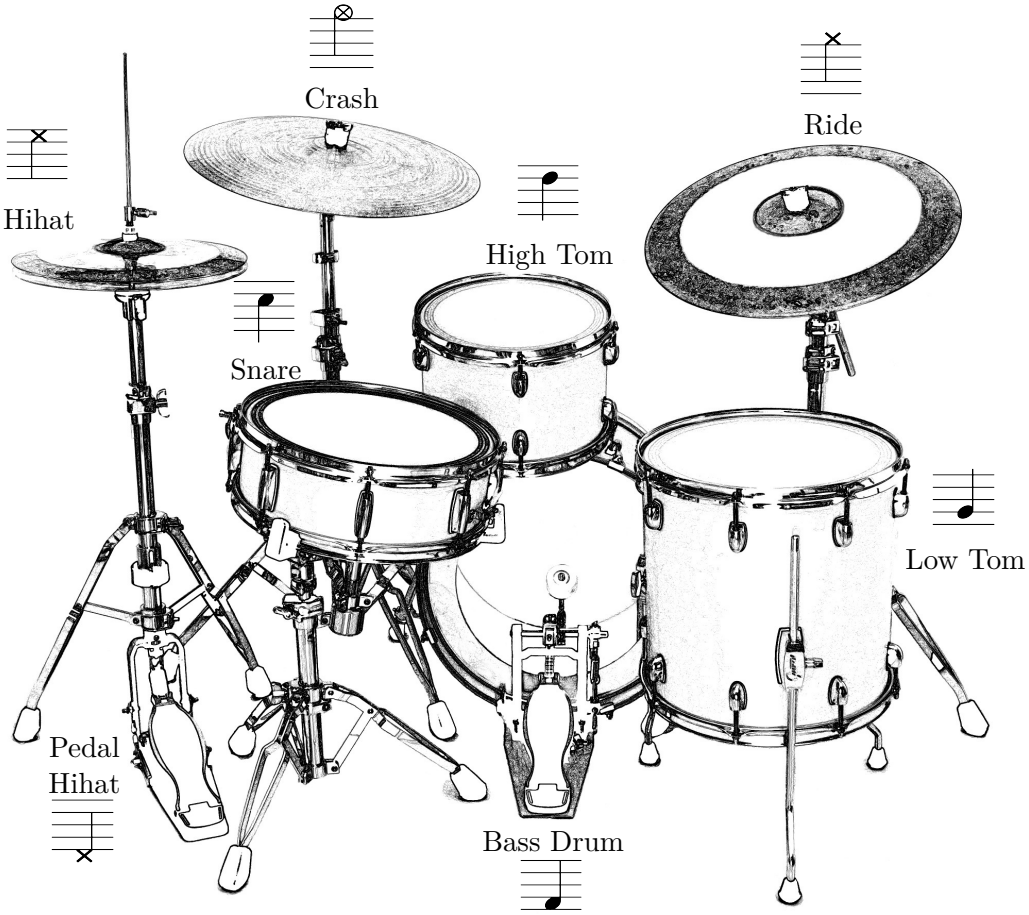


Abbildung 3.1.: Aufbau eines Jazz-Schlagzeugs und Notation der Instrumente

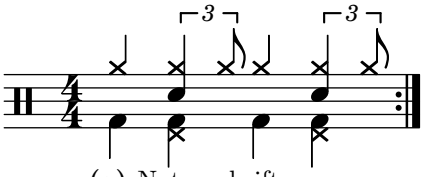
3.3. Drum Computer

Ein *Drum Computer* ist ein Gerät zur rhythmischen Wiedergabe perkussiver Klänge. Besonders populär sind frei programmierbare Maschinen (*Drum Machines*). Die Funktionsweise soll anhand der weit verbreiteten analogen Drum Machine *Roland TR-808* erklärt werden [Roland, 1981].

Die TR-808 enthält verschiedene analoge Klanggeneratoren, die Schlagzeug- und Perkussionsinstrumente imitieren. Diese werden in programmierbaren Zeitintervallen angesteuert. Dazu dient ein *Step-Sequenzer*. Dieser hält eine Sequenz von einzelnen Steps (auch Ticks oder Beats genannt). Jedem Step ist eine Menge an Generatoren zugeordnet. Ein Metronom-Modul aktiviert der Reihe nach die Steps. Das Zeitintervall zwischen den Wechseln kann mithilfe eines Temporeglers eingestellt werden. Befindet sich ein Klanggenerator in der Menge eines gerade aktivierten Steps, so wird die Klangerzeugung angestoßen. Auf der Bedienoberfläche des TR-808 findet sich für jeden Step eine eigene Taste. Um einen Klang hinzuzufügen, wird ein Generator, sprich ein Instrument, mit einem Drehrad ausgewählt und anschließend die gewünschte Step-Taste gedrückt.

Die mit einem Step-Sequenzer erstellten Rhythmen heißen Schlagzeug-Patterns. Diese können in musikalischer Notenschrift notiert werden. Eine andere Möglichkeit, die das Konzept des Step-Sequenzers aufgreift, ist eine tabellarische Notation, wie sie in modernen grafischen Oberflächen von Drum Machines und auch schon im Benutzerhandbuch des TR-808 zu finden ist. Die Spalten stehen für die einzelnen Steps, die Zeilen für die Instrumente.

Abbildung 3.2a zeigt ein Swing Pattern in Notenschrift. Abbildung 3.2b zeigt dasselbe in tabellarischer Notation. Ein weiteres Beispiel für ein typisches Pattern im Bossa Nova zeigen die Abbildungen 3.3a und 3.3b.

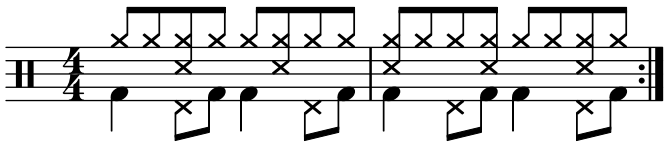


(a) Notenschrift

	1	2	3	4	5	6	7	8	9	10	11	12
Ride Cymbal	■			■		■	■			■		■
Snare				■						■		
Bass Drum	■			■			■			■		
Pedal Hihat				■						■		

(b) Tabellenform

Abbildung 3.2.: Eintaktiges Swing Pattern in Notenschrift und Tabellenform



(a) Notenschrift

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Ride Cymbal	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Side Stick			■			■			■			■			■	
Bass Drum	■			■	■			■	■			■	■			■
Pedal Hihat			■				■				■				■	

(b) Tabellenform

Abbildung 3.3.: Zweitaktiges Bossa Nova Pattern in Notenschrift und Tabellenform

In den 1990ern entstanden Sequenzer-Programme für Homecomputer. Auch dort wurde das Konzept der Drum Machine weiter verfolgt. Die Klangerzeugung geschieht digital und häufig mithilfe von Samples. Das sind kurze Audioaufnahmen der Instrumentenklänge, die nur noch zum richtigen Zeitpunkt abgespielt werden müssen.

Auch die Metronom-Module wurden vielseitiger. So ist es ohne weiteres möglich, die Anzahl der Steps beliebig zu wählen und somit verschiedene, auch ungewöhnliche Taktarten

zu erstellen (statt 4/4 z.B. auch 3/4, 5/4, 7/8 oder 15/16). Zur realistischen Wiedergabe können auch ternäre Rhythmen (Shuffle) beitragen, wie sie vor allem im Jazz und Blues Verwendung finden. Bei einem üblichen binären Rhythmus wird ein Notenwert in zwei gleiche Teile zerlegt (Verhältnis 1:1). Eine Viertelnote besteht beispielsweise aus zwei gleichlangen Achtelnoten. Im Bossa Nova-Pattern (Abbildung3.3) sind das zwei Beats. Bei einem ternären Rhythmus wird der Notenwert im Verhältnis 2:1 geteilt. Die zweite Achtel rückt in das letzte Drittel einer Viertelnote. Im Swing-Pattern (Abbildung 3.2) wird eine Viertel deshalb auf drei Beats abgebildet. Der Eindruck, dass jede zweite Note verspätet gespielt wird, erzeugt ein Gefühl der Entspannung in der Musik. Ein Step-Sequenzer mit Shuffle-Funktion bietet meist auch die Möglichkeit, das Shuffle-Verhältnis beliebig einzustellen.

3.4. Jazz

3.4.1. Geschichte

In diesem Abschnitt soll ein kurzer Überblick über die Jazzgeschichte gegeben werden, um die Kernelemente dieser Musik erklären zu können. Dieser Abschnitt orientiert sich an der ausführlichen Darstellung der Jazzgeschichte von Gioia [2011].

Die erste starke Ausprägung des Jazz entstand um 1900 in den USA. Der *New Orleans Jazz* hat seine Wurzeln in Marching Bands, Gospel und Blues und somit in der afroamerikanischen Musiktradition. Ein stilistisches Merkmal ist die Kollektivimprovisation, bei der europäische Melodieinstrumente wie Trompete, Klarinette und Posaune gleichzeitig Melodien improvisieren. Begleitet werden sie dabei von Tuba oder Kontrabass sowie einem rudimentären Schlagzeug. Eine bekannte stilistische Weiterentwicklung durch weiße Musiker ist der *Dixieland Jazz*.

Diese neuen Musikstile verbreiteten sich in den 1910er Jahren bis nach Chicago und New York. Dort entstand die bis heute populärste Form, der *Swing*, und prägte mit seinem ternären Rhythmus maßgeblich die sogenannte Big-Band-Ära von etwa 1920–1940. Ab 1940 entwickelten sich neben wichtigen Swing-Ablegern wie *Bebop*, *Hard Bop* und *Cool Jazz* Mischungen mit anderen Genres. Der *Latin Jazz* enthält Einflüsse aus Südamerika, hauptsächlich durch den kubanischen *Son* und die brasilianische *Samba* (*Bossa Nova*). Ab 1960 folgten *Fusion* (mischt Jazz mit Rock) und *Soul Jazz* (mischt Jazz mit Funk). Seither finden sich Verbindungen mit Jazz in fast allen Genres wieder. Weitere Beispiele sind *Reggae*, *HipHop*, elektronische Musik (*Nu Jazz*) und Pop (*Smooth Jazz*).

Jazz hat sich somit vom Begriff des Musik-Genres gelöst. Jazz ist nicht an Instrumentierung, Harmonik, Rhythmus oder Form gebunden. Der gemeinsame Nenner einer jeden Stilrichtung des Jazz ist die Improvisation. Jazzmusik ist demnach improvisierte Musik.

Es können drei Arten der Improvisation unterschieden werden. Es ist möglich, dass alle beteiligten Musiker gleichzeitig improvisieren, so wie es im *Free Jazz* der Fall ist. Auch

eine Gruppenimprovisation einiger Instrumente des Ensembles wie im bereits erwähnten New Orleans Jazz ist üblich. Aber die häufigste Form, die im populären Jazz der 1930er Jahre etabliert wurde, ist die abwechselnde Improvisation von Solisten.

3.4.2. Das Jazzsolo

Das Improvisieren in einer Jazzband ist nicht ohne gewisse Absprachen möglich. Vielmehr benötigt das gemeinsame Spielen verschiedener Musiker, die womöglich noch nie gemeinsam geprobt haben, eine Struktur, an der sich die Improvisation orientieren kann. Ein solches Szenario findet sich bei Jamsessions, wie sie in den 1940er-Jahren entstanden und bis heute existieren. Die theoretischen Konzepte dieses Abschnitts stützen sich auf die Jazz-Harmonielehren von Sikora [2003] und Haunschild [1997].

Der Ablauf einer typischen Session soll nun beispielhaft erläutert werden: Die Musiker einigen sich auf ein Stück, das als Improvisations-Grundlage dienen soll. Das Grundgerüst des Stücks muss allen bekannt sein. Ist das Stück jemandem unbekannt, kann ein Lead-sheet weiterhelfen. Dabei handelt es sich um ein skizzenhaftes Notenblatt, auf dem lediglich die Harmonien als Akkordsymbole in rhythmischer Form sowie die Melodie des Stückes (Thema) notiert sind (siehe Abbildung 3.4). Auf dessen Grundlage kann jeder Musiker seine Funktion im Gesamtzusammenhang ausführen. Natürlich muss eine Stilistik ausgewählt werden. Typisch für Sessions sind Swing, Ballade oder Bossa Nova. Die Basis des Ensembles bilden das Schlagzeug mit taktgebender Funktion und der (Kontra-)Bass, der vorrangig die Harmonien des Stückes ausspielt. Sie zählen zur *Rhythmusgruppe*, zu der auch die *Harmonieinstrumente* wie Klavier und Gitarre gehören.

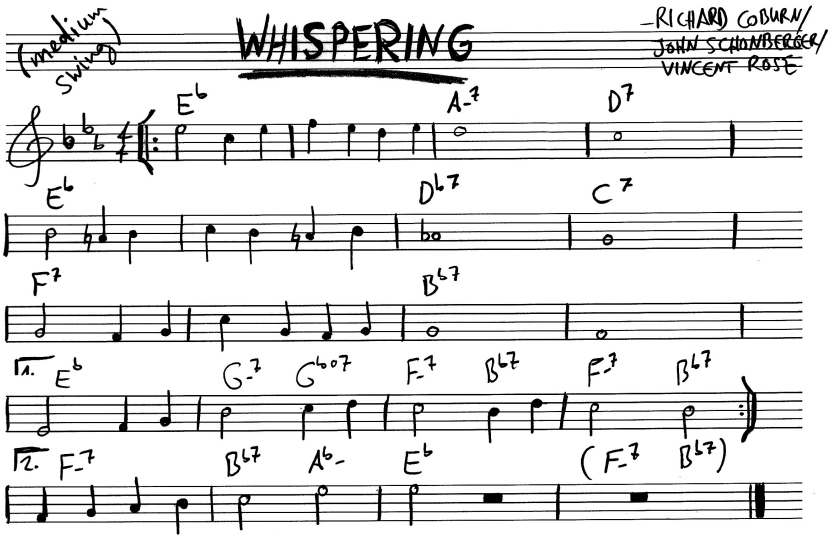


Abbildung 3.4.: Leadsheet des Jazzstandards *Whispering* (1920)

Typisch für ein Jazz-Stück ist folgender Ablauf: Zu Beginn wird das Thema von den *Melodieinstrumenten* gespielt. Dazu gehören typischerweise Trompete, Saxophon, Posau-

ne, seltener auch Instrumente wie Klarinette, Flöte oder Violine. Auch Gitarre und Klavier können Themen spielen und so zu Melodieinstrumenten werden. Eher untypisch sind Bassthemen. Dem Thema folgen die Soli. Ein Instrument, meist aus der Gruppe der Melodieinstrumente, beginnt zu improvisieren. Die anderen Melodieinstrumente hören auf zu spielen, nur die Rhythmusgruppe spielt weiterhin zur Begleitung. Dazu wird die Akkordfolge, die bereits für das Thema verwendet wurde und mit einer bestimmten Anzahl an Takten die Form des Stückes bestimmt, wiederholt. Eine Wiederholung nennt sich Chorus. Die Begleitung kann beliebig frei interpretiert werden, sollte sich allerdings an den vorgegebenen Akkorden orientieren, da der jeweilige Solist diese erwartet und seine Töne danach auswählt. So beschränkt sich ein Harmonieinstrument bei der Begleitung meist auf rhythmische Variationen und spielt im richtigen Moment die vorgegebenen Akkorde. Der Solist führt die Band an und gibt musikalische Wendungen und Höhepunkte vor, die durch die Rhythmusgruppe unterstützt werden. Das Solo endet typischerweise am Ende der Form nach einer beliebigen Anzahl von Chorussen. Dann übernimmt der nächste Solist. Wenn alle Musiker soliert haben, wird nochmals das Thema gespielt. Dieser Ablauf wird aufgrund seiner Form (Thema – Soli – Thema) auch Sandwich-Form genannt.

Zum besseren Verständnis dieser typischen Form sei an dieser Stelle auf ein Hörbeispiel verwiesen. Das Stück *Freddie Freeloader* vom Album *Kind of Blue* (1959) des amerikanischen Jazztrompeters Miles Davis ist ein 12-taktiger Bb-Blues. Das Thema wird am Anfang und zum Schluss zweimal gespielt. Dazwischen liegen fünf Soli verschiedener Solisten. Das erste Solo spielt der Pianist Wynton Kelly über vier Chorusse ($4 \times 12 = 48$ Takte). Es folgen Davis' Solo über 5 Chorusse (60 Takte) und anschließend zwei Saxophon-Soli von John Coltrane und Cannonball Adderley auch über jeweils 5 Chorusse. Das letzte Solo vor dem Schlussthema spielt Bassist Paul Chambers über zwei Chorusse (24 Takte). Das Stück besteht demnach insgesamt aus 25 Wiederholungen der Grundakkordfolge (insgesamt $25 \times 12 = 300$ Takte), die von der Rhythmusgruppe bestehend aus Klavier, Kontrabass und Schlagzeug ständig wiederholt wird. Trotzdem klingt jeder Chorus anders, da die Begleitung durchgehend variiert wird.

3.4.3. Begleitungskonzepte des Schlagzeugers

Die Begleitung der Rhythmusgruppe kennt für jedes Instrument und seine Funktion im Ensemblekontext verschiedene Konzepte. In diesem Abschnitt soll das begleitende Schlagzeugspiel erklärt werden.

Das Schlagzeugspiel im Jazz orientiert sich meistens an einer zugrundeliegenden Stilistik. Definiert wird die Stilistik im Jazz vorrangig aus rhythmischen Vorgaben. Für den Schlagzeuger bedeutet das, dass ein ungefährer Grundschlag (Schlagzeug-Pattern) vorgegeben ist. So wird beispielsweise die Stilistik Swing mit einem Swing-Pattern (siehe Abbildung 3.2a) begleitet. Tempoangaben finden sich in Stilistiken wie Bebop (Fast Swing), Medium Swing oder Swing Ballade (Slow Swing). Ein anderes klassiches Pattern ist der Bossa Nova (siehe Abbildung 3.3a).

Der Schlagzeuger erhält dadurch eine Vorstellung, wie er ein Stück interpretieren soll. Jedoch wäre das ständige Wiederholen eines Patterns äußerst eintönig. Deshalb werden die Patterns stark abgewandelt gespielt und während des Stücks variiert. Der Schlagzeuger kann entscheiden, wie sehr er sich an ein Pattern bindet und so einen Freiheitsgrad wählen. Dieser nimmt in der Praxis seit der Entstehung des Jazz tendenziell zu.

Der Schlagzeuger soll eine begleitende Rolle übernehmen, wenn ein anderer Musiker soliert. Er gibt ihm erst einmal ein Tempo und ein Pattern vor. Daran orientiert sich der Solist. Ein Solo hat einen Spannungsbogen. Ein Beispiel für einen Spannungsbogen zeigt Abbildung 3.5, die sich an den Beispielen in Sikora [2003, S. 568] orientiert. Der Spannungsbogen beschreibt die Intensität des Solos über seine zeitliche Länge. Die Intensität steigt mit hoher Dynamik (Lautstärke), hoher Schnelligkeit, ausgefallener Tonalität und aufreibenden Klängen (unsauber gespielte Töne). Ein Spannungsbogen sollte vom Solisten initiiert und von der Begleitung getragen werden. Der Schlagzeuger passt die Intensität des eigenen Spiels an das des Solisten an. Er kann sich mit zunehmender Intensität des Solos immer stärker vom vorgegebenen Pattern lösen, was wiederum eine Intensitätssteigerung hervorruft, und so für eine Entwicklung der Musik ins Unerwartete sorgen.

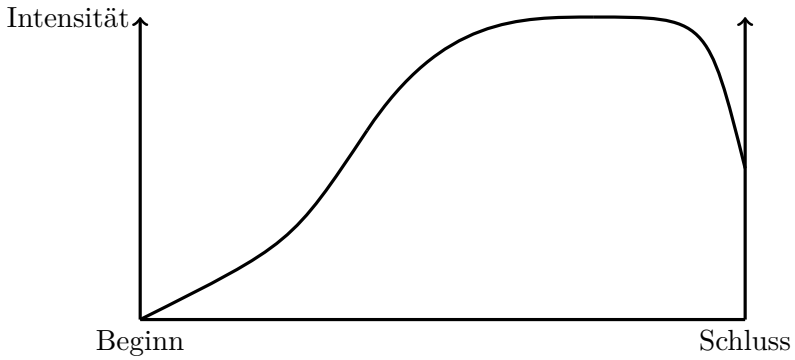


Abbildung 3.5.: Möglicher Spannungsbogen eines Solos

4. Musical Instrument Digital Interface

4.1. Geschichte

Das *Musical Instrument Digital Interface* (MIDI) ist ein Protokoll-Standard, der ursprünglich zur Kommunikation von Synthesizern verschiedener Hersteller entwickelt wurde. Sein Vorgänger, das *Universal Synthesizer Interface*, entstand 1981 und ermöglichte das Ansteuern mehrerer Klangerzeuger (Synthesizer) über eine einzige Klaviatur (Keyboard). Die Hersteller erhofften sich damit eine erhöhte Flexibilität beim Bau ihrer Instrumente, um beispielsweise Synthesizer ohne Klaviatur zu verkaufen.

Die Firmen Roland und Sequential Circuits erarbeiteten 1982 *MIDI 1.0* [MMA, 1995a] und brachten 1983 den ersten Synthesizer mit MIDI-Schnittstelle auf den Markt, der mit dem typischen 5-poligen DIN-Stecker ausgestattet war. Aus dieser Zusammenarbeit entstand die 1985 gegründete *MIDI Manufacturers Association* (MMA). Diese erweiterte den Standard durch *General MIDI* (GM) [MMA, 1991]. Eigenständige Erweiterungen wie *General Standard* (GS) von Roland und *Extended General Standard* (XG) von Yamaha folgten in 1991 und 1994.

Der Durchbruch von MIDI gelang mit der Einführung von Tonstudiotechnik auf Homecomputern. Eines der ersten Modelle mit MIDI-Schnittstelle war der 1985 erschienene Atari ST.

Heute findet sich MIDI in fast allen Formen elektronischer Musikinstrumente. Selbst Instrumente mit akustischer Klangerzeugung wie Gitarre oder Saxophon können mithilfe von Wandlern zu MIDI-Instrumenten werden und so Synthesizer ansteuern. Näheres dazu wird in Abschnitt 4.4 erläutert.

Auch bei der Komposition, dem Arrangieren oder dem Notensatz am Computer findet MIDI als Eingabehilfe, Audioausgabe oder Speicherprinzip Anwendung.

Die Erklärungen in diesem und im folgenden Abschnitt orientieren sich an Collins [2010, S. 46–50] und Mazzola [2006, S. 57–63].

4.2. Spezifikation

Wie bereits in der Einführung erwähnt, ermöglicht MIDI die Steuerung eines elektronischen Klangerzeugers mithilfe einer mechanischen Klaviatur. Dazu wird ein serielle, unidirektionale Verbindung vom MIDI-Out-Port der Klaviatur durch ein 5-poliges DIN-Kabel zum MIDI-In-Port des Synthesizers hergestellt. Die MIDI-Kommunikation stellt dabei

eine Abstraktion der Aktionen eines Pianisten an einer Klaviertastatur dar. Sie schickt einzelne Nachrichten, die *MIDI-Messages*, die genau eine Information zum Verhalten des Pianisten enthält. Eine Nachricht könnte dem Synthesizer beispielsweise mitteilen: Der Pianist drückt genau jetzt auf die Taste für die Note Db3 so kräftig er kann (laut). Der Synthesizer würde reagieren und ein Db3 erklingen lassen, und zwar solange, bis er die Nachricht erhält, dass der Pianist die Taste schließlich losgelassen hat. Die Struktur von Nachrichten kann durch dieses Modell besonders einfach gehalten werden.

Eine MIDI-Message besteht aus Sequenzen von 10-Bit-Wörtern. Das erste (Start-Bit) und das letzte Bit (Stop-Bit) sind immer 0 und dienen der Übertragung. Im Folgenden soll nur noch das innere Byte (= 8 Bit) betrachtet werden.

Das erste Wort stellt das Statusbyte dar, welches stets mit dem Bit-Wert 1 beginnt. Die folgenden drei Bits definieren den *Befehl* der Nachricht. Die übrigen vier Bits bestimmen den *Channel* (0–15), für den die Nachricht bestimmt ist. Ein Synthesizer, der auf einen bestimmten Channel eingestellt ist, wird nur Befehle für diesen ausführen. So kann beim Spiel zwischen verschiedenen Instrumenten und Sounds gewechselt werden. Der zehnte Kanal eines Synthesizers ist dabei häufig für die Schlagzeug-Ausgabe reserviert.

Die nachfolgenden Wörter sind Datenbytes. Diese enthalten Informationen, die abhängig vom Befehl interpretiert werden müssen. Der Befehl definiert außerdem, wie viele Datenbytes folgen.

MIDI-Dateien sind von der MMA ebenfalls spezifiziert worden. Für das Speichern von Musikstücken wird das Format *Standard-MIDI-File* (SMF) verwendet. Die Dateiendung ist typischerweise '`.mid`'. Die MIDI-Messages werden zeitlich sortiert gelistet und mit einem *Tick* versehen. Die Ticks geben die Zeit nach dem Start in Millisekunden an. Ein Tick wird dabei meist als Delta-Zeit zum Vorgänger angegeben. Ein Sequenzer kann die Befehle so rhythmisch synchronisiert ausführen.

4.2.1. MIDI-Hierarchie

Durch die verschiedenen Statusbyte-Typen sind die MIDI-Messages hierarchisch gegliedert (siehe Abbildung 4.1).

Die gröbste Unterteilung der MIDI-Messages erfolgt in *Channel Messages* und *System Messages*. Erstere sind konkrete musikalische Anweisungen. Unter die *Channel Voice Messages* fallen *NoteOn* und *NoteOff* sowie Befehle wie *Program Change* (Instrument-Wechsel), *Control Change* (klangbeeinflussende Aktionen abseits der Klaviatur, beispielsweise Betätigung des Haltepedals) oder *Pitch Bend Change* für musikalische Stilmittel wie Vibrati und Glissandi. Während die Channel Voice Messages das Verhalten der einzelnen Channels steuern, regeln die *Channel Mode Messages* das Zusammenspiel der verschiedenen Channels anhand von Befehlen wie *All Sound/Notes Off*, *Reset All Controllers* und *Local Control Off* (deaktiviert die lokale Klaviatur).

System Messages geben Befehle an System-Komponenten außerhalb der Klangerzeugung. Sie beginnen immer mit einem Statusbyte der Form 1111... und sind grundsätzlich

in drei Typen unterteilt. Die *System Common Messages* setzen allgemeine Systemeinstellungen wie die Position in einem Song oder die aktuelle Stimmung (absolute Tonhöhe). Die *System Real-Time Messages* enthalten Befehle zur Steuerung der MIDI *Timing Clock*, einem Takt-gebenden Modul. Die *System Exclusive Messages* senden im ersten Datenbyte eine Hersteller-ID. So haben Unternehmen Raum, ihre eigenen Ideen in den MIDI-Standard zu integrieren. Erkennt ein Gerät die ID seines Herstellers, so kann es entsprechend reagieren.

Da für diese Arbeit lediglich die Befehle *NoteOn* und *NoteOff* von Bedeutung sind, wird auf eine genauere Erläuterung der übrigen verzichtet. Für eine vollständige Auflistung aller Befehle und ihrer Kodierung siehe MMA [1995b].

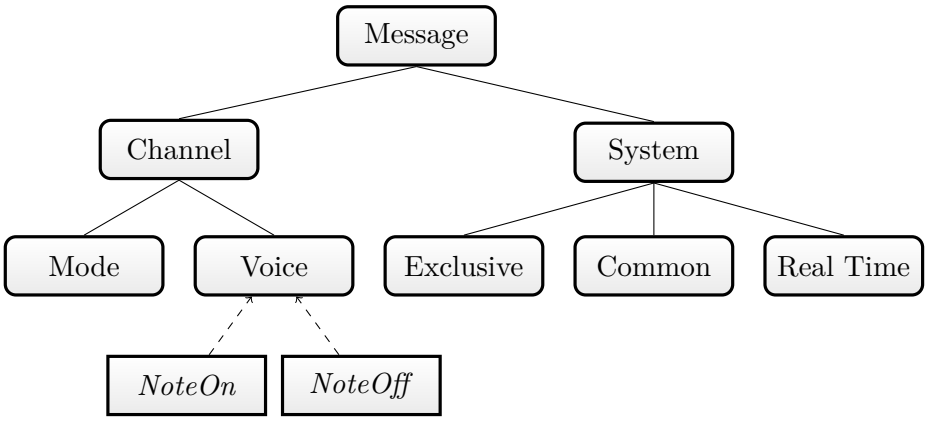


Abbildung 4.1.: Hierarchie der MIDI-Messages

4.2.2. Die Befehle *NoteOn* und *NoteOff*

NoteOn informiert den Empfänger über die Betätigung einer Taste der Klaviatur. Das erste Datenbyte enthält die Notennummer (Key) und das zweite die Anschlagsgeschwindigkeit (Velocity). Beide sind mit sieben Bit, die den Integer-Werten 0–127 entsprechen, kodiert. Ist der angesteuerte Synthesizer zur Erzeugung von Schlagzeug gedacht, so werden den Key-Werten durch *General MIDI* [MMA, 1991] Instrumente zugeordnet. Ansonsten stehen sie für Tonhöhen. Einen Auszug der Zuordnung enthält Abbildung 4.4. Der diskrete und lineare Velocity-Wert reicht von stumm (0) bis so laut wie möglich (127). *NoteOff* ist identisch aufgebaut und informiert über das Loslassen einer Taste. Die Abbildung 4.2 zeigt die Struktur einer *NoteOn*-Message.

4.3. Java Sound API

Das Java Sound Application-Programming Interface (API) ist eine auf niederer Ebene angelegte Schnittstelle für Audiosignale und MIDI-Daten. Dabei übernimmt die API die Organisation der Ein- und Ausgabe und enthält Werkzeuge zur Erzeugung und Manipulation.

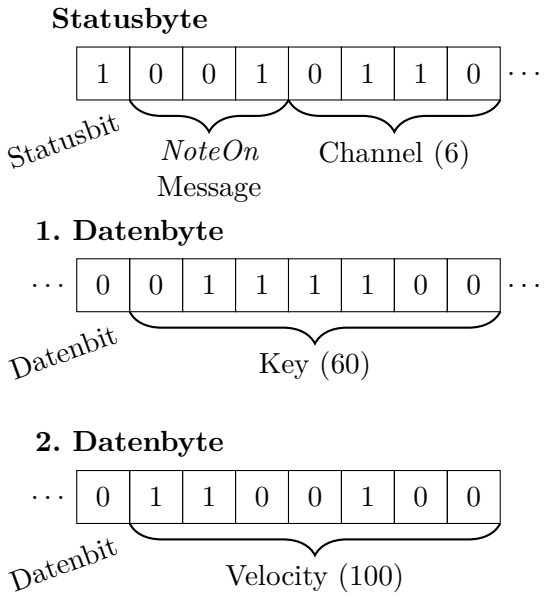


Abbildung 4.2.: Struktur einer *NoteOn*-Message

Audio-Daten lassen sich mithilfe der Pakete *javax.sound.sampled* und *javax.sound.sampled.spi* aufzeichnen, bearbeiten und abspielen. Die grundlegende Repräsentation ist dabei die eines Arrays, das mit Amplituden-Werten gefüllt ist und so ein akustisches Signal abbildet. Das Gebiet der Elektroakustik ist jedoch nicht Bestandteil dieser Arbeit. Nähere Erläuterungen zu diesen Paketen finden sich in den Java Tutorials von Oracle [1995].

Die MIDI-fähigen Pakete der Java Sound API heißen *javax.sound.midi* und *javax.sound.midi.spi*. Das erste besteht aus 15 Klassen, elf Interfaces und zwei Exceptions, während das zweite nur 4 Klassen zum Lesen und Schreiben von Dateien bereitstellt [vgl. Oracle, 2015]. Die Java Sound API wurde in Java Version 1.3 zu den Standardpaketen hinzugefügt, jedoch seitdem kaum weiterentwickelt.

Trotz einiger Schwächen wie kleinerer Bugs oder schlecht dokumentierter Methoden, die wohl der kurzen Entwicklungszeit geschuldet sind, ist die Java Sound API sehr hilfreich und zuverlässig im Umgang mit MIDI-Geräten und MIDI-Ströme. Ihr Herzstück ist die statische Klasse **MidiSystem**[1]. Diese erkennt automatisch MIDI-Geräte, die beim Betriebssystem angemeldet sind, und gibt diese als **MidiDevice**-Objekte zurück. Dabei kann eine Liste aller Geräte angefordert werden, aus der ein passendes gesucht werden muss. Ist kein bestimmtes Gerät gefordert, kann **MidiSystem** ein geeignetes als Standard zurückgeben.

Die Klasse **MidiDevice** kann ein Keyboard, einen Synthesizer oder ein anderes MIDI-fähiges Gerät repräsentieren, das sich über einen **Receiver** (MIDI-In) und einen **Transmitter** (MIDI-Out) definiert. Diese können wiederum von **MidiDevice** erfragt werden.

[1] Klassennamen sind im Folgenden **fett** markiert

Receiver ist ein Interface, das eine Methode `send()` definiert. Diese bekommt ein Objekt der Klasse **MidiMessage**, die eine MIDI-Message im Sinne des MIDI-Standards (siehe Abschnitt 4.2) darstellt, sowie einen Zeitstempel übergeben. Die Message wird an das MIDI-Gerät geschickt und verarbeitet. Ein Synthesizer wird typischerweise über seinen Receiver angesprochen. **Synthesizer** ist als von **MidiDevice** erbende Klasse in der API enthalten.

Transmitter stellt das Gegenstück zur Klasse **Receiver** dar. Sie hält ein **Receiver**-Objekt und ist somit in der Lage MIDI-Messages zu schicken. **Sequencer**, der ebenfalls von **MidiDevice** erbt und beispielsweise das Abspielen einer MIDI-Datei ermöglicht, sendet Daten über seinen **Transmitter**.

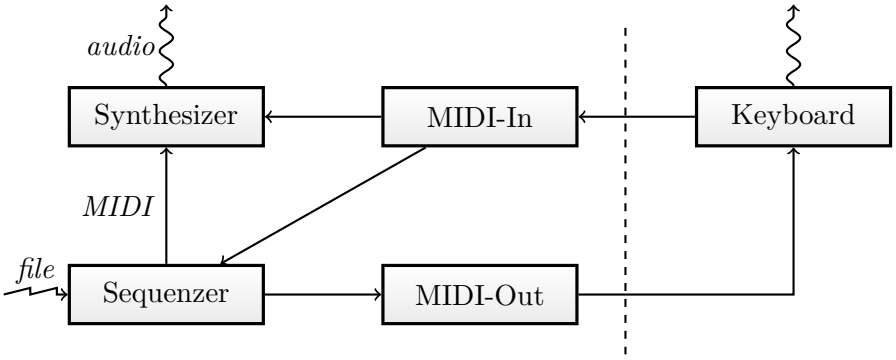


Abbildung 4.3.: Typische Anordnung von MIDI-Geräten

Die Behandlung von MIDI-Messages regelt die Klasse **MidiMessage**. Sie kennt die drei Unterklassen **ShortMessage**, **SysexMessage** und **MetaMessage**. Die erste kann alle Messages mit höchstens zwei Datenbytes darstellen. Darunter fallen alle Channel Voice Messages und Channel Mode Messages, demnach also auch die wichtigen Befehle *NoteOn* und *NoteOff*, sowie viele System Common und System Real Time Messages. **SysexMessage** behandelt die System Exclusive Messages, die eine beliebige Anzahl an Datenbytes haben können, indem die Bytes in einem Array gespeichert werden. **MetaMessages** enthalten Informationen, die über die Klangerzeugung hinausgehen. Sie sind in MIDI-Dateien enthalten und können beispielsweise Liedtexte, Copyrights oder Tempoangaben sein.

Die Java Sound API wird in dieser Arbeit für die Übermittlung der von einem MIDI-fähigen Musikinstrument gespielten Noten zu einem Analysemodul verwendet. Die Beschreibung von MIDI in Java-Code endet deshalb hier.

Es sei trotzdem auf höherentwickelte Systeme wie *jMusic* [Sorensen u. Brown, 1998] und *jFugue* [Koelle, 2002] hingewiesen. Diese legen mehr Wert auf eine Abstraktion weg von der MIDI-Technologie und ermöglichen so einen musikalischeren Umgang mit Noten und MIDI-Messages. jMusic wurde entwickelt, um ein Modell nahe der Arbeit eines Komponisten zu erstellen und grafische Ausgaben in Notensystemen einfach erzeugen zu können.

jFugue unterstützt Musik-Programmierung, indem es Soundausgaben aus Notennamen und einfachen Zeichenmustern generiert.

4.4. Audio-MIDI-Wandler

Audio in MIDI zu konvertieren ist eine hochkomplexe Aufgabe für eine Maschine. Es gilt, die Notenhöhen und die Notenlängen zu bestimmen und bei Bedarf in Echtzeit in MIDI-Messages zu übersetzen. Um das zu erreichen, muss das Eingangssignal besonders störungsfrei aufgezeichnet werden. Das kann durch ein Mikrofon (zum Beispiel vor einem Saxophon-Trichter) oder ein spezielles Tonabnehmer-System bei einer Gitarre (oder einem anderen Saiteninstrument) geschehen.

Frühe Wandler wie der Roland GI-10 von 1995 sind große, schwere Geräte, deren Fehlerquote bei Gebrauch eines Mikrofons schon bei leichten Störgeräuschen enorm ansteigt und somit nicht zur Ansteuerung eines Synthesizers geeignet sind. Allerdings sind sie bei der Analyse von Musik hilfreich, da sie zumindest ein ungefähres Bild der gespielten Noten erfassen können.

Nach der Entstehung der Wandler gewann die Entwicklung von Algorithmen zur Tonhöhenerkennung zunehmend an Qualität. Heute sind sie aus Software für Musikstudiotechnik nicht wegzudenken, wobei die meisten Algorithmen ihre Stärke nicht in der Echtzeitanalyse haben. Einen Überblick über wichtige historische Arbeiten gibt Gerhard [2003].

In dieser Arbeit wird das Sonuus G2M, ein moderner monophoner und besonders handlicher Vertreter der Audio-MIDI-Wandler, benutzt.

Key-Wert	Perkussions-Instrument	technische Tonbezeichnung	internationale Tonbezeichnung
⋮	⋮	⋮	⋮
35	Acoustic Bass Drum 1	B1	$B_{,,}$
36	Bass Drum 1	C2	$C_{,}$
37	Side Stick	Db2	$C_{,}^{\sharp}/D_{,}^{\flat}$
38	Acoustic Snare	D2	$D_{,}$
39	Hand Clap	Eb2	$D_{,}^{\sharp}/E_{,}^{\flat}$
40	Electric Snare	E2	$E_{,}$
41	Low Floor Tom	F2	$F_{,}$
42	Closed Hi Hat	Gb2	$F_{,}^{\sharp}/G_{,}^{\flat}$
43	High Floor Tom	G2	$G_{,}$
44	Pedal Hi Hat	Ab2	$G_{,}^{\sharp}/A_{,}^{\flat}$
45	Low Tom	A2	$A_{,}$
46	Open Hi Hat	Bb2	$A_{,}^{\sharp}/B_{,}^{\flat}$
47	Low Mid Tom	B2	$B_{,}$
48	High Mid Tom	C3	C
49	Crash Cymbal 1	Db3	$C^{\sharp}/D^{\flat}$
50	High Tom	D3	D
51	Ride Cymbal 1	Eb3	$D^{\sharp}/E^{\flat}$
52	Chinese Cymbal	E3	E
53	Ride Bell	F3	F
54	Tambourine	Gb3	$F^{\sharp}/G^{\flat}$
55	Splash Cymbal	G3	G
56	Cowbell	Ab3	$G^{\sharp}/A^{\flat}$
57	Crash Cymbal 2	A3	A
58	Vibraslap	Bb3	$A^{\sharp}/B^{\flat}$
59	Ride Cymbal 2	B3	B
60	High Bongo	**C4**	$\boldsymbol{c}$
61	Low Bongo	Db4	$c^{\sharp}/d^{\flat}$
62	Mute High Conga	D4	d
63	Open High Conga	Eb4	$d^{\sharp}/e^{\flat}$
64	Low Conga	E4	e
65	High Timbale	F4	f
66	Low Timbale	Gb4	$f^{\sharp}/g^{\flat}$
67	High Agogo	G4	g
68	Low Agogo	Ab4	$g^{\sharp}/a^{\flat}$
69	Cabasa	**A4**	$\boldsymbol{a}$
70	Maracas	Bb4	$a^{\sharp}/b^{\flat}$
71	Short Whistle	B4	b
72	Long Whistle	C5	c'
73	Short Guiro	Db5	$c^{\sharp\prime}/d^{\flat\prime}$
74	Long Guiro	D5	d'
75	Claves	Eb5	$d^{\sharp\prime}/e^{\flat\prime}$
76	High Wood Block	E5	e'
77	Low Wood Block	F5	f'
78	Mute Cuica	Gb5	$f^{\sharp\prime}/g^{\flat\prime}$
79	Open Cuica	G5	g'
80	Mute Triangle	Ab5	$g^{\sharp\prime}/a^{\flat\prime}$
81	Open Triangle	A5	a'
⋮	⋮	⋮	⋮

Abbildung 4.4.: Die Zuordnung der Tonhöhen zu Key-Werten
C4 ist das mittlere c auf der normalen Klaviatur mit 88 Tasten.
A4 ist der Kammerton a mit 440Hz.

Teil II.

Praktische Umsetzung

5. Verwandte Arbeiten

Dieses Kapitel enthält einen Überblick über Arbeiten mit einem ähnlichen kreativen Ansatz zu Jazz- und Schlagzeug-Generierung, die diese Arbeit inspiriert haben. Die Arbeiten sind nach der Stärke des Einflusses auf diese Arbeit geordnet.

Ein evolutionärer Algorithmus zur Erzeugung von Jazz-Soli

Das System *GenJam* (Genetic Jammer) von Biles wurde 1994 erstmals vorgestellt [vgl. Biles, 1994] und seither gab es fast jährlich Publikationen über Erweiterungen. Biles hat regelmäßig Auftritte mit seinem System, bei denen er Trompete spielt.

Biles sagt selbst, dass GenJam zuerst eine Spielerei gewesen sei, eine intellektuelle Herausforderung, mit der er habe zeigen wollen, wozu evolutionäre Algorithmen im Stande sind und mit der er für ein bisschen Spaß auf Konferenzen über Computermusik habe sorgen wollte. Die Qualität der generierten Soli lagen allerdings weit über seinen Erwartungen, sodass GenJam ein ambitioniertes Projekt wurde [vgl. Miranda u. Biles, 2007, S. 162].

GenJam ist das Modell eines Jazzschülers, der anhand von Rückmeldungen der Zuhörer seine gelernten Melodien verändert. Diese Melodien werden durch zwei Populationen repräsentiert. Zunächst gibt es eine Population von *Takt*-Individuen, die Noten in einem Takt darstellen und ähnlich der MIDI-Spezifikation kodiert sind. Damit die kodierte Melodie eines Takt-Individuums über verschiedene Akkorde passt, besteht der Genotyp nicht aus festen Tönen, sondern nur aus Tonsprüngen (Intervallen) oder Pausen. Wird ein Takt gespielt, so hängt die konkrete Melodie von den darunterliegenden Akkorden ab. Dazu nutzt das Programm eine Zuordnung, welche Tonleitern über welchen Akkordtyp passen. Die zweite Population enthält sogenannte *Phrasen*-Individuen, die vier Takte zu einer musikalischen Einheit verbinden.

Die Begleitung lässt Biles vor der Ausführung von GenJam statisch von Band-In-A-Box [Gannon, 1990] generieren. Der virtuelle Improvisator spielt mithilfe eines MIDI-Tongenerators (Synthesizer) zu diesem Playalong. Ursprünglich wurden die Individuen zufällig generiert. Mittlerweile werden sie nach musikalischen Kriterien erzeugt. Anschließend beginnt die interaktive Evaluierung. Dazu werden alle Individuen dem Zuhörer (Mentor) vorgespielt. Dieser bewertet ihre Fitness durch mehrfaches Drücken der Tasten `'G'` (für „good“, +1) oder `'B'` (für „bad“, −1). So entstehen nach einigen Generationen Phrasen, die der Mentor als schön empfindet.

1998 erweiterte Biles GenJam um die Fähigkeit, *Call-And-Response* mit einem menschlichen Musiker zu spielen [vgl. Biles, 1998]. Dabei spielt der Mensch über ein paar Takte (meistens vier oder acht) eine Melodie (Call) und die Maschine antwortet mit einer passenden Gegenmelodie (Response). Dazu muss das System die erste Phrase aufzeichnen, in Echtzeit bearbeiten und rechtzeitig abspielen können. Die Aufnahme erfolgt mithilfe eines Audio-MIDI Konverters. Daraus wird ein neues Individuum erstellt, das anschließend durch evolutionäre Mutation verändert wird. Hier wird Biles' musikalisch-sinnvolle Mutation („musically meaningful mutation" [Biles, 1998, S.3]) wichtig, da ein Mensch die Evaluation in der Kürze der Zeit nicht leisten könnte. Die Operatoren müssen so gestaltet sein, dass aus einer guten Phrase ebenfalls nur gute Phrasen entstehen können.

So entstand die Idee zu einem evolutionären System ohne Fitness [vgl. Biles, 2001]. Die Start-Population darf dabei nur gute Phrasen enthalten. Biles' nutzt dafür eine Sammlung von Phrasen, die von berühmten Jazzmusikern gespielt wurden. Die Reproduktion geschieht mithilfe von intelligenter Rekombination und musikalisch-sinnvoller Mutation. Die entstandenen Individuen müssen nicht bewertet werden, da alle eine hinreichende Fitness besitzen. Biles betont, dass darüber gestritten werden müsse, ob ein System ohne Fitness noch ein evolutionärer Algorithmus sei.

Virtuelles improvisierendes Orchester

Voyager ist ein interaktives Musiksystem, das seit 1986 vom Jazz-Posaunisten Lewis entwickelt wird. Lewis nennt es ein virtuelles improvisierendes Orchester („virtual improvising orchestra" [Lewis, 2000, S. 33]). Es analysiert die Performance eines improvisierenden Musikers in Echtzeit und generiert sowohl musikalische Antworten als auch selbständige Melodien aus internen Algorithmen. Die entstehende Musik sieht er als freien Dialog zwischen Musiker und Programm, wie er im Bereich des Avantgarde- und Freejazz üblich ist. Die Ausgabe wird mithilfe von MIDI auf einem mechanischen Piano oder durch einen Piano-Synthesizer erzeugt. Voyager kann demnach als virtueller Pianist bezeichnet werden, wobei das generierte Material nicht für einen Menschen spielbar sein muss.

Lewis erklärt, dass das Modell des Programms auf den losen Strukturen basiere, auf die eine Improvisation aufbaue. Jede Form von Musik benötige Struktur als Anhaltspunkt für das Verständnis des Zuhörers. Improvisation (vor allem im Jazz) will jedoch die Grenzen der Struktur aufbrechen und eine gewisse Formlosigkeit erhalten [vgl. Lewis, 2000, S. 38]. Diese Diskrepanz wird überwunden, indem eine Reihe von musikalischen Merkmalen definiert wird, die Strukturen erkennen und extrahieren. Dazu wird das Solo mithilfe eines Audio-MIDI Konverters in digitales Notenmaterial umgewandelt und anschließend statistische Merkmale extrahiert. Beispiele für Merkmale [vgl. Collins, 2010, S. 220] sind:

- Dichte: Die Anzahl an Noten beschreibt, wie schnell und wie intensiv gespielt wird. Keine Noten impliziert Stille.

- Tonumfang: Der Abstand der tiefsten zur höchsten Note.
- Töne: Welche Töne sind erklungen? Wie viele?
- Töne einer Oktave: Welche Töne modulo 12 sind erklungen? Alle Töne werden den Tönen einer Oktave zugeordnet. Daraus lässt sich die Tonart oder der Grad an Atonalität ablesen.
- Durchschnittliche Anschlagsstärke: Bestimmt Lautstärke und Intensität des Spiels.
- Intervalle: Anhand der Bestimmung der Notenintervalle lassen sich beispielsweise aufsteigende und absteigende Linien unterscheiden.

Die Auswertung der Merkmale findet in Echtzeit statt und bedingt die musikalische Antwort. Dazu wird ein zeitliches Fenster definiert, in dem die gespielten Noten des menschlichen Musikers aufgezeichnet werden. Auf dem Inhalt des Fensters werden die Merkmale berechnet. Die Breite des Fensters hat dabei maßgeblich Einfluss auf die Qualität. Ist es zu groß, kommt es zu Verzögerungen der Antwort, ist es zu klein, werden zu wenige Noten betrachtet, um eine sinnvolle Analyse anzustellen.

Voyager ist ein rein regelbasiertes System und wird deswegen als dynamische Computer-Komposition („computer music composition“ [Lewis, 2000, S. 33]) verstanden.

The Evolving Drum Machine

Yee-King erklärt, dass kreative Musikformen wie der Jazz aus einer Vielzahl an Klangfarben (Timbre) bestehen. So sei es möglich, mit jedem Instrument durch unterschiedliche Spielweisen verschiedenste Klänge zu erzeugen. Die elektronische Musik, vor allem die Formen Techno- und House-Musik, kenne solch eine Praxis vor allem im Bereich Rhythmus nicht. Dort blieben Timbres wie von Snare oder Bass Drum über einen ganzen Song hinweg unverändert, wie es auch bei den ersten analogen Drum Machines wie dem Roland TR909 der Fall gewesen sei [vgl. Yee-King, 2007].

Yee-King versucht durch seine *Evolving Drum Machine* mit der Praxis der statischen Timbres bei Perkussionsinstrumenten zu brechen. Er entwirft einen evolutionären Algorithmus, dessen Individuen durch Frequenzanalyse an einen angestrebten Klang, der als Audio-Signal vorliegt, angepasst werden. Die Anpassung geschieht während der Wiedergabe eines rhythmischen Patterns durch einen Step-Sequenzer. Anhand dieser Audioausgabe kann der Benutzer die Optimierung verfolgen. Der Prozess stellt eine algorithmische Echtzeit-Komposition dar. Der Benutzer kann mit dem Programm interagieren, indem er das Pattern zur Laufzeit verändert und die angestrebten Klänge der verschiedenen Instrumente austauscht.

NEAT Drummer

Von Hoover und Stanley stammt ein interaktiver evolutionärer Algorithmus, der das künstliche neuronale Netz *NEAT* (NeuroEvolution of Augmenting Topologies) [Stanley u. Miikkulainen, 2002] verwendet, um eine erste, bereits musikalisch zufriedenstellende Population zu erzeugen. Der *NEAT Drummer* [Hoover u. Stanley, 2007] generiert Schlagzeug-Begleitungen zu gegebenen Musikstücken in MIDI-Format. Die Patterns bestehen nur aus Bass Drum, Hihat und Snare.

Hoover und Stanley versuchen einen Kompromiss zu finden zwischen Freiheit und Struktur der Begleitung. Generierte Kompositionen bisheriger Arbeiten seien entweder zu wenig innovativ und zu stark an Vorgaben wie ein Genre gebunden oder zu chaotisch und erzeugten damit für Menschen unverständliche Musik.

Das Programm soll beispielsweise als Unterstützung beim rhythmischen Arrangieren für weniger erfahrene Musiker Verwendung finden.

Rhythmische Kompositionen mit interaktiver Evolution

Das System *CONGA* (Composition in Genetic Approach) [Tokui u. Iba, 2001] erstellt mithilfe eines interaktiven evolutionären Algorithmus Kompositionen aus Rhythmus-Patterns. Dazu werden zwei Populationen erstellt. Die erste enthält kurze Patterns (4–16 Takte) und ist Bestandteil eines genetischen Algorithmus (GA). Die Patterns werden von zweidimensionalen Chromosomen repräsentiert, deren Zellen den Wert der Anschlagstärke für ein Instrument auf einem Beat beinhalten. Die Individuen in der zweiten Population stellen Arrangements der Patterns im Bezug zu einer möglichen musikalischen Funktion dar und werden nach dem Prinzip der genetischen Programmierung (GP) behandelt. Die Komposition wird dabei als kombinatorische Optimierung betrachtet. Der Suchraum besteht aus den Rhythmus-Patterns. Die Fitness-Funktion wird durch einen Menschen ersetzt. Die Reproduktion der Individuen bestimmen zufällige Operatoren, aber auch musikalisch sinnvolle wie Rotation, Invertierung, Verkettung und Wiederholung.

Tokui und Iba erklären, dass in Experimenten mit Testpersonen, die einen unterschiedlichen musikalischen Hintergrund hatten, zufriedenstellende Ergebnisse erzielt werden konnten. Auch ein Test mit dem Ziel der Erzeugung genretypischer Rhythmen wird als gelungen bezeichnet. Damit stelle sich die gleichzeitige Anwendung von GA und GP zur Erzeugung musikalischer Kompositionen als sinnvoll heraus.

Ein Programm zur Komposition kurzer Musikstücke

Mithilfe von *Sbeat* [Unemi u. Nakada, 2001] können kurze Musikstücke bestehend aus Gitarre, Bass und Schlagzeug kreiert werden. Verwendet wird ein interaktiver evolutionärer

Algorithmus. Die Individuen bestehen aus 16 Beats eines 4/4 Taktes. Der Benutzer wählt Individuen, die ihm gefallen, zur Reproduktion aus.

Die erstellten Musikstücke wirken sehr primitiv, werden aber selten als atonal, unschön oder störend beschrieben. Das Programm kann online heruntergeladen und kostenfrei unter Mac OS ausprobiert werden.

Menschliche, rhythmische Begleitung

Systeme zur Generierung von rhythmischer Begleitung durch ein Schlagzeug legen bisher keinen Wert auf die realistische, menschliche Wiedergabe. Diese Erkenntnis veranlasste Dostál, den *GeneticDrummer* [Dostál, 2005] zu entwickeln, einen evolutionären Algorithmus, der zu einer gegebenen Stimme eines harmonischen Instruments (wie Bass, Gitarre oder Klavier) eine Begleitung durch ein Schlagzeug mit Snare, Bass Drum, Hihat, mehreren Toms und mehreren Becken erstellt. Dafür speichern die Chromosomen nicht nur die Anschlagsstärke auf einem Beat, sondern auch, auf welche Weise ein Instrument angeschlagen werden soll (beispielsweise ist ein Becken mit der Kante (crash) oder der Spitze (ride) eines Sticks spielbar). Zusätzlich wird die Ungenauigkeit in Timing und Dynamik eines menschlichen Schlagzeugers simuliert. Die Ergebnisse werden dabei automatisch, jedoch nicht in Echtzeit erzeugt.

6. Zielsetzung

Das Ziel dieser Arbeit ist die Implementierung eines Systems zur rhythmischen Begleitung eines Jazz-Solos. Angestrebt wird eine Verarbeitung in Echtzeit, denn die Interaktion mit dem Benutzer, der gleichzeitig auch Musiker ist, steht dabei im Vordergrund. Die Ausgabe des Systems muss in akustische Form gebracht werden können, um für den Musiker unmittelbar verständlich zu sein. Der angestrebte Anwendungsbereich ist die Übesituation eines Jazzmusikers, in der die Begleitung durch einen menschlichen Schlagzeuger so gut wie möglich simuliert werden soll. Sollten gute Ergebnisse zu Stande kommen, so könnte das System auch im Kontext eines Konzerts für ein Publikum interessant sein.

Wie es in der Jazzmusik üblich ist, wird die musikalische Stilistik durch ein Schlagzeug-Pattern vorgegeben. Dieses initiale Pattern wird vom Benutzer gewählt und ist Ausgangspunkt für Variationen durch das System. Das Eingabe-Interface folgt dem Prinzip der Drum-Machine. Musiker sind üblicherweise mit deren Funktionsweise vertraut, sodass im besten Falle eine intuitive Bedienung möglich ist. Die manuelle Veränderung des Patterns zur Laufzeit, wie bei Drum-Machines üblich, soll zudem eine dynamische Anpassung an die spontanen Ideen des Benutzers ermöglichen.

Um Kreativität im musikalischen Dialog algorithmisch abzubilden, wird das Konzept der evolutionären Algorithmen gewählt. Es ermöglicht, den Grad der gewollten und zufälligen Ereignisse zu kontrollieren und die Patterns nach der Vorstellung des Benutzers zu optimieren. Dieser soll außerdem einen großen Einfluss auf die Gestaltung des EA nehmen können. Viele Parameter wie Populationsgröße und Intervallzeit sind deshalb zur Laufzeit veränderbar.

Der EA muss Schlagzeug-Pattern dahingehend optimieren, dass sie zu einem momentanen zeitlichen Solo-Ausschnitt als passende Begleitung empfunden werden. Zur Bestimmung eines Fitnesswertes werden dazu einige Regeln angewendet, die zu einem Paar aus Pattern und Solo-Ausschnitt einen Fitnesswert berechnen. Da ein solcher Wert allenfalls dem subjektiven Geschmacksempfinden des Entwicklers gerecht werden kann, soll auch hier das Konfigurieren durch den Benutzer möglich sein. Die Regeln arbeiten gewichtet. Der Versuch besteht darin, den Benutzer die Verhaltensweise des virtuellen Schlagzeugers nach seinen persönlichen Vorlieben gestalten zu lassen.

Die abschließende Bewertung des Systems kann wiederum nur eine subjektive sein. Die Arbeit kann als gelungen eingeschätzt werden, wenn Benutzer gefunden werden können, denen ein Üben und Performen mithilfe des vorgestellten Systems einen Mehrwert im Sinne ihres musikalischen Interesses oder ihres musikalischen Horizonts bringt.

7. Systementwurf

In diesem Kapitel wird ein Überblick über den Zusammenhang der einzelnen Komponenten des implementierten Systems gegeben. Im Vordergrund steht dabei die Aufteilung des Gesamtsystems in prinzipiell unabhängige Module.

Zunächst soll die Interaktion des Systems mit seiner Umgebung anhand von Abbildung 7.1 erläutert werden. Benutzerinteraktionen sind mit dem Symbol 🧍 gekennzeichnet.

Die Ein- und Ausgaben des Systems bestehen ausschließlich aus MIDI-Informationen. Die Ausgabe stellt die Wiedergabe der generierten Schlagzeug-Patterns dar. Sie werden als MIDI-Messages an einen Schlagzeug-Synthesizer gesendet, der aus diesen Anweisungen Klänge erzeugt. Diese werden von 🧍$_{Musiker}$ gehört. Er spielt dazu auf einem MIDI-fähigen Instrument (beispielsweise Keyboard). Seine Töne sind ebenfalls hörbar und werden zusätzlich als MIDI-Messages kodiert zum System geschickt. Das reagiert auf die Eingabe mit Variation der Patterns. Die Art der Variation kann 🧍$_{Benutzer}$ durch Änderung einiger Parameter über eine grafische Benutzeroberfläche (GUI) beeinflussen. Üblicherweise sind 🧍$_{Musiker}$ und 🧍$_{Benutzer}$ identisch, jedoch besteht darin keine Notwendigkeit. 🧍$_{Publikum}$ steht für das optionale Publikum, das bei einer Vorführung des Systems anwesend ist. Das Publikum lauscht dem Zusammenspiel des Solos von 🧍$_{Musiker}$ mit den Klängen des Schlagzeug-Synthesizers.

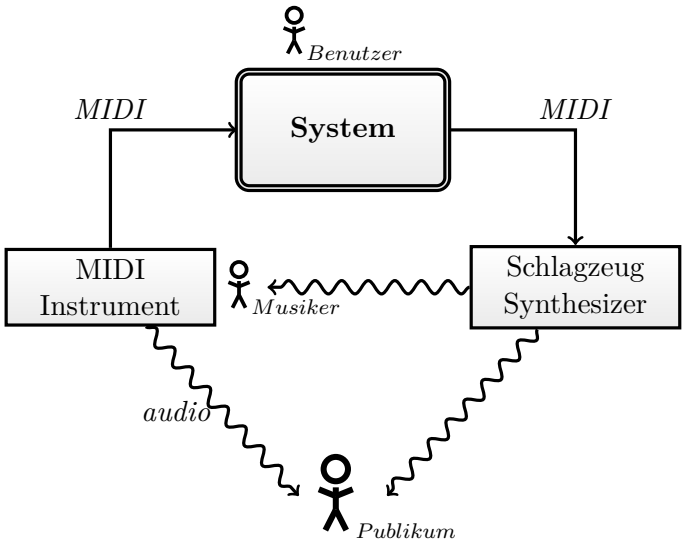


Abbildung 7.1.: Interaktion des Systems mit seiner Umgebung

Abbildung 7.2 zeigt die modularisierte Übersicht der Systemstruktur. Die Module arbeiten für sich autonom und es gibt keine direkte Kommunikation. An zwei Stellen dient jeweils eine spezielle Datenstruktur dem Austausch von Informationen zwischen zwei Modulen. Der Verarbeitungsstrom fließt im kompletten System grundsätzlich unidirektional.

Das erste Modul der Verarbeitungskette ist das *Input-Modul*. Es ist zuerst für die Aufnahme des eingehenden MIDI-Stroms zuständig. Das Modul wandelt die MIDI-Messages in eine systemeigene Datenstruktur um und schickt sie abschließend zur temporären Archivierung an das *Input-Window*. Eine genaue Erklärung der Funktionsweise folgt in Kapitel 8.

Das *Genetic-Modul* enthält den EA und ist somit Kern des Systems. Hier werden die Pattern-Variationen erzeugt und an eine Datenstruktur namens *Generation* geschickt. Die Generation ist immer eine Menge von Patterns, die vom EA als abspielbar eingestuft wurden. Da diese Bewertung durch das sich verändernde Solo ständig korrigiert werden muss, wird sie in zeitlich regelmäßigen Abständen ausgetauscht. Sie stellt somit eine Population nach dem Konzept der EA dar, die sich in jeder Generation komplett erneuert. Das Genetic-Modul nutzt eine regelbasierte Evaluation zur Bewertung der Schlagzeug-Patterns. Dazu werden sowohl der momentane Inhalt des Input-Windows als auch die Patterns im Hinblick auf musikalische Merkmale analysiert und passende Patterns positiv bewertet. Der Benutzer kann Einfluss auf das Genetic-Modul nehmen, indem er Parameter der Evolution mithilfe der GUI ändert. Auch die anfängliche Population wird vom Benutzer gewählt. Eine ausführliche Erläuterung der einzelnen Elemente des EA folgt in Kapitel 9.

Das dritte Modul ist das *Playback-Modul*, das die Wiedergabe der Schlagzeug-Patterns steuert. Es beschafft sich ständig neue Patterns aus der Generation und sorgt für die Umwandlung der systemeigenen Repräsentation in einen MIDI-Strom. Die zeitliche Organisation übernimmt ein virtuelles Metronom. Es schickt die MIDI-Messages rhythmisch synchronisiert zum Takt der Musik an einen Synthesizer außerhalb des Systems. Der Benutzer hat über die GUI Zugriff auf die Metronom-Einstellungen. In Kapitel 10 wird das Modul vollständig beschrieben.

Zur Implementierung wurde die Programmiersprache Java gewählt. Die Java Sound API ermöglicht die plattformübergreifende Verwaltung von MIDI-Geräten und die bequeme Verarbeitung von MIDI-Messages.

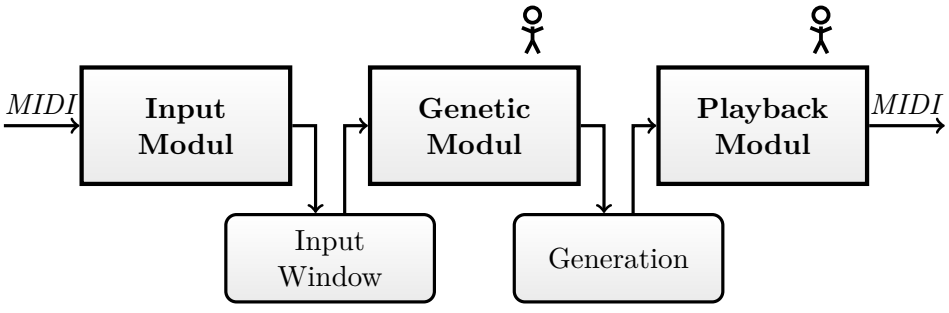


Abbildung 7.2.: Modularisierte Systemstruktur

8. Input-Modul

Das Input-Modul regelt den eingehenden MIDI-Strom, der von einem MIDI-fähigen Instrument an das System geschickt wird. Dieser Strom wird verarbeitet und im Input-Window zwischengespeichert. Die losen MIDI-Messages werden dabei in eine Repräsentation umgewandelt, die dem Modell einer Note entspricht. Das erleichtert die spätere Analyse nach musikalischen Merkmalen im Genetic-Modul.

Abbildung 8.1 zeigt den schematischen Aufbau des Moduls. Es folgt die Erläuterung der einzelnen Bestandteile.

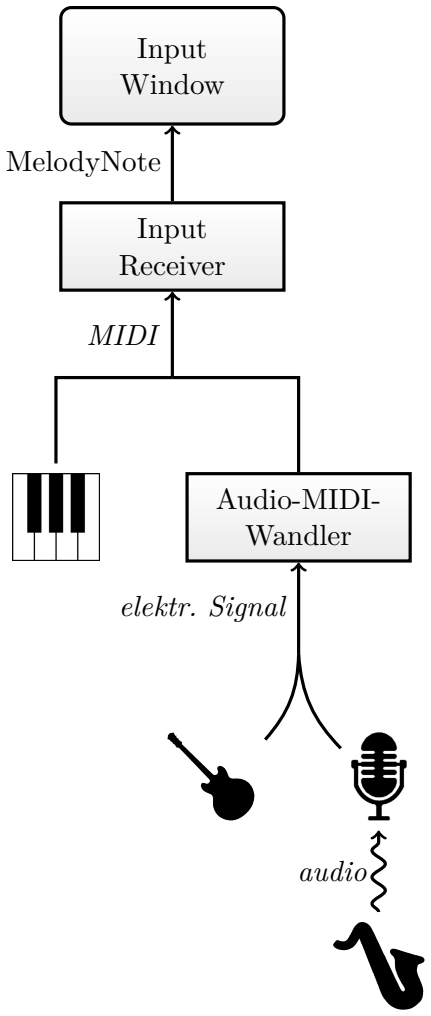


Abbildung 8.1.: Schematischer Aufbau des Input-Moduls

8.1. Verbindung MIDI-fähiger Instrumente

Da das System als Eingabe MIDI-Informationen erwartet, muss das Solo-Instrument in der Lage sein, diese direkt zu erzeugen.

Die einfachste Möglichkeit ist die Nutzung eines MIDI-Keyboards. Das sind alle klavierähnlichen elektronischen Tasteninstrumente, die einen MIDI-Out-Port besitzen. Dieser wird mittels eines 5-poligen DIN-Kabels mit dem MIDI-In-Port eines Audiointerfaces verbunden. Dort kann der MIDI-Strom über das Betriebssystem des Computers per Java Sound API vom System abgegriffen werden. Die Erzeugung von MIDI-Messages mithilfe eines MIDI-Keyboards ist besonders einfach, da sie sensorisch über die Mechanik der Tasten erfolgt. Das Spiel wird originalgetreu abgebildet und übermittelt.

Es ist ebenso möglich, andere Instrumente mit MIDI-Fähigkeit auszustatten. Dazu muss das akustische Signal mit einem Mikrofon oder einem Tonabnehmersystem (typisch für Gitarre) in ein elektrisches Signal umgewandelt und anschließend in Echtzeit in MIDI-Messages umgesetzt werden. Diese Aufgabe erfüllt ein Audio-MIDI-Wandler. Dieser arbeitet jedoch nicht exakt. Je nach Qualität des Signals und Stärke des Algorithmus kann eine mehr oder weniger starke Abweichung zwischen den gespielten Noten und den erzeugten MIDI-Messages entstehen. Das System ist allerdings nicht auf eine exakte Übertragung angewiesen. Eine ungefähre Eingabe der gespielten Noten reicht aus, um den evolutionären Algorithmus zielführend zu beeinflussen.

Das implementierte System ist somit in der Lage, mit praktisch jedem Instrument sinnvoll zu interagieren.

8.2. Notenmodell *MelodyNote*

Die MIDI-Messages sind Informationen in Form von Befehlen. Sie stellen keine Abstraktion der Noten, sondern nur der Aktionen eines Pianisten dar. Für eine musikalische Analyse wird jedoch ein notenähnliches Modell benötigt. Zu diesem Zwecke wurde die primitive Klasse *MelodyNote* entworfen.

Ein MelodyNote-Objekt besteht lediglich aus den vier Attributen Notennummer, Anschlagsstärke, Startzeit und Länge. Die Notennummer bestimmt eine Note auf der Klaviertastatur entsprechend dem MIDI-Protokoll. Die Anschlagsstärke entscheidet über die Lautstärke, mit der die Note erklingt. Die Startzeit ist der Zeitpunkt, an dem die Note zu erklingen beginnt, und wird in Millisekunden angegeben. Sie dient der zeitlichen Ordnung der Noten zueinander. Die Länge der Note ist die Zeit, die vom Startzeitpunkt verstreicht, bis das Klingen der Note beendet wird.

MelodyNote enthält somit alle wichtigen Eigenschaften einer musikalischen Note.

8.3. Eingabeverwaltung mit *Input-Receiver*

Der *Input-Receiver* ist ein Receiver im Sinne der Java Sound API. Wann immer eine MIDI-Message am verbundenen MIDI-Port ankommt, wird eine Methode aufgerufen, die als Übergabeparameter ein MidiMessage-Objekt erhält. Die Verarbeitung ist dabei nur in zwei Fällen sinnvoll und geschieht wie folgt:

- Wenn es eine *NoteOn*-Message ist, dann wird ein MelodyNote-Objekt erzeugt, das die Notennummer und die Anschlagsstärke der NoteOn-Message erhält. Die Startzeit wird auf die aktuelle Systemzeit gesetzt, die Länge bleibt unbestimmt. Abschließend wird das erzeugte Objekt in einer Liste mit unvollendeten Noten zwischengespeichert.
- Wenn es eine *NoteOff*-Message ist, dann wird in der Liste der unvollendeten Noten ein MelodyNote-Objekt gesucht, dessen Notennummer mit der Nummer der Message übereinstimmt. Da auf ein NoteOn einer Note immer ein NoteOff folgen muss, wird diese Note als vollendet erkannt. Dem Objekt wird eine Länge zugewiesen, die sich aus der Differenz der aktuellen Systemzeit und dessen Startzeit ergibt. Außerdem wird das Objekt aus der Liste gelöscht und stattdessen in das Input-Window geschrieben.
- Tritt keiner der oberen beiden Fälle ein, so wird das MidiMessage-Objekt ignoriert.

8.4. Datenaustausch durch *Input-Window*

Das *Input-Window* ist ein Puffer, in dem der zeitliche Ausschnitt eines Solos zwischengespeichert wird. Dieser Puffer wird vom Input-Receiver mit MelodyNote-Objekten gefüllt und vom Genetic-Modul ausgelesen und geleert.

Das Genetic-Modul nutzt die Noteninformationen, um musikalische Merkmale zu extrahieren, mit deren Hilfe das Gespielte analysiert werden kann. Es leert das Input-Window in äquidistanten Zeitintervallen. Die Leerung muss allerdings nicht vollständig geschehen. Es ist auch möglich, nur Noten zu löschen, deren Startzeit $t_{start} < t_{now} - T_w$ ist, wobei t_{now} die aktuelle Systemzeit und T_w der Zeitraum vor t_{now}, in dem die Noten noch nicht entfernt werden, ist. So können Noten in mehreren Analysezyklen berücksichtigt und ein größerer Solo-Ausschnitt betrachtet werden, ohne die Zeit zwischen den Zyklen zu erhöhen. Der Benutzer kann T_w zur Laufzeit beliebig verändern. Näheres zur Leerung durch das Genetic-Modul in Kapitel 9.

Die Länge des Zeitintervalls, in dem die Noten im Input-Window liegen, ist wider erwarten nicht konstant. Grund dafür ist die notwendige Bedingung, dass für die musikalische Analyse lediglich vollendete Noten verwendet werden können. Unvollendete Noten werden vom Input-Receiver bis zur Vollendung durch eine NoteOff-Message zurückgehalten. Liegt der Zeitpunkt der Leerung durch das Genetic-Modul zwischen dem *NoteOn* und *NoteOff* einer Note, so wird diese erst anschließend hinzugefügt. Bei der nächsten Leerung befindet

sich demnach eine Note im Input-Window, deren Startzeit vor dem Zeitpunkt der vorherigen Leerung liegt. An dieser Stelle wird angenommen, dass es sich um ein monophones Solo handelt. Das bedeutet, jede Note erklingt für sich allein. Auf jedes *NoteOn* folgt ausschließlich ein *NoteOff* und umgekehrt. Dadurch wird ausgeschlossen, dass eine Note n_1, deren Startzeit $t_1 < t_2$ einer anderen Note n_2 ist, in einem späteren Zyklus analysiert wird als n_2. Die Länge des Zeitintervalls darf deshalb variieren. Es wird angenommen, dass die leichten Abweichungen keinen Einfluss auf die Güte des Systems haben.

9. Genetic-Modul

Das Genetic-Modul ist das Kernstück des implementierten Systems. Es enthält den EA zur Variation der Schlagzeug-Patterns. Das Modul agiert losgelöst von den Modulen Input und Playback und ist somit weder für die MIDI-Verarbeitung noch für die Wiedergabe zuständig. Vielmehr trifft es unter Berücksichtigung der momentanen Solosituation eine Auswahl unter selbst erzeugten Patterns, die zur Begleitung des Solos am geeignetsten erscheinen.

Der grundsätzliche Aufbau des Moduls wird durch Abbildung 9.1 veranschaulicht. Sie stellt die Interaktionen der einzelnen Komponenten mit den Mengen Generation und Nachkommen durch einfache Pfeile dar. Der evolutionäre Zyklus ist durch breitere Pfeile gekennzeichnet. Die Komponenten des EA lauten Initialisierung, Mutation, Evaluation und Selektion und werden in den folgenden Abschnitten ausführlich erklärt.

Der EA wird vom Benutzer gestartet und gestoppt. Zudem ist der Benutzer in der Lage, den Zyklus zu pausieren, um sich die Patterns der derzeitigen Population genauer anzuhören. Eine Interaktion mit dem Musiker entsteht nur, wenn der EA arbeitet. Einer der Parameter, die vom Benutzer gewählt werden können, ist die Zeit, die der EA für einen Durchlauf des Zyklus zu Verfügung hat. In der Abbildung wird die Stelle des Wartens durch das Symbol ⏲ gekennzeichnet. Die sinnvolle Wahl der Zykluszeit T_z wird in Kapitel 11 diskutiert.

Der virtuelle Schlagzeuger soll immer eine Verzögerungszeit beim Produzieren musikalischer Antworten haben, wie sie auch ein menschlicher Schlagzeuger hätte. Die Länge der Verzögerung ist wiederum vom subjektiven Geschmack des Benutzers abhängig und deshalb veränderbar.

Es ist möglich, dass die Zykluszeit zu kurz ist, um Mutation und Evaluation fristgerecht auszuführen. Dann wird der Benutzer gewarnt und hat die Möglichkeit, die Zykluszeit zu erhöhen oder die Komplexität der Berechnung zu verringern.

9.1. Repräsentation der Individuen

Der Phänotyp der Individuen im EA des implementierten Systems ist ein hörbares Schlagzeug-Pattern. Um die Funktionsweise der einzelnen Komponenten des EA verständlich zu machen, muss die Repräsentation seiner Individuen als Genotyp erläutert werden.

Ein Schlagzeug-Pattern besteht aus rhythmischen Mustern verschiedener Schlaginstrumente. Die einzelnen Muster haben die selbe Länge. Diese kann nach dem Prinzip der

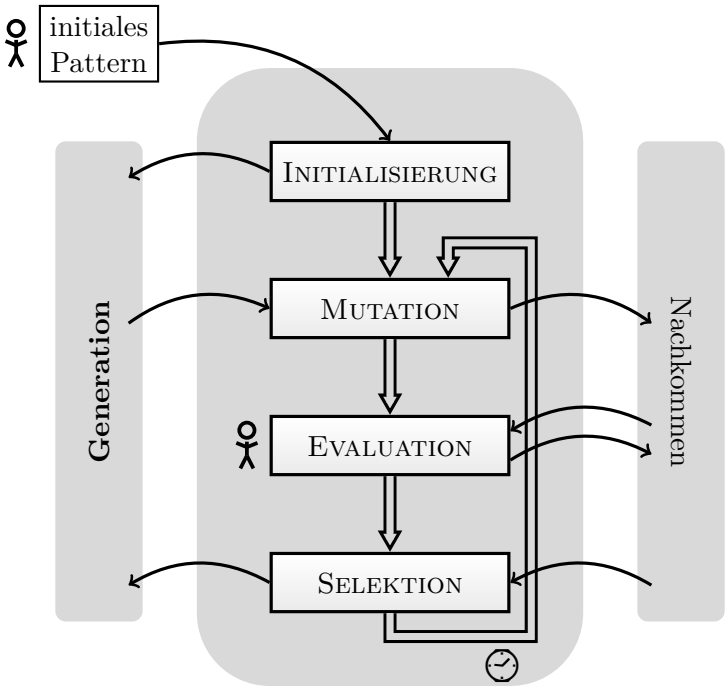


Abbildung 9.1.: Schematischer Aufbau des Genetic-Moduls

Drum Machines in einzelne Einheiten, die sogenannten Ticks, unterteilt werden. Die Anzahl der Ticks bestimmt dabei die Taktart und die Auflösung des Patterns in Notenwerten. Ein paar Beispiele:

- Wird ein Pattern in 8 Ticks unterteilt, so liegt wahrscheinlich ein 4/4-Takt vor, dessen kleinstmöglicher Notenwert eine Achtel ist.
- Wird ein Pattern in 7 Ticks unterteilt, so liegt wahrscheinlich ein 7/8-Takt vor.
- Wird ein Pattern in 4 Ticks unterteilt, so bleibt es offen, ob ein 2/4-Takt in Achteln oder ein 4/4-Takt in Vierteln gemeint ist.
- Wird ein Pattern in 16 Ticks unterteilt, so könnten ein 4/4-Takt in Sechszehnteln oder auch zwei 4/4-Takte in Achteln gemeint sein.

Meist entscheidet das Tempo über das subjektive Empfinden der Taktart. Bei hohen Tempi wird häufig das halbierte Tempo wahrgenommen. Das Tempo wird jedoch nicht von der Repräsentation spezifiziert und das Erzeugen neuer Patterns erfolgt losgelöst von dessen Abspielgeschwindigkeit. Diese wird vom Playback-Modul gesteuert und lässt sich durch den Benutzer einstellen (siehe Kapitel 10).

Notwendige Informationen sind demnach nur Muster und Instrumente eines Patterns. Die Ticks der Muster sind prinzipiell mit einer binären Information belegt, die eine Aussage darüber trifft, ob an ihrer Stelle innerhalb des Musters ein Schlag stattfindet oder nicht. Zusätzlich muss eine Information über die (Laut-)Stärke des Schlags gegeben sein.

Deshalb kann ein Tick durch einen Integerwert zwischen 0–n_{max} repräsentiert werden. Dabei bedeutet der Wert 0 keinen Schlag, während 1–n_{max} einen leichten bis starken Schlag bedeutet. Eine Zelle des Array mit einem Wert größer 0 heißt aktive Zelle. Je größer n_{max} ist, desto präziser ist die Stärke quantisiert. Für das implementierte System wurde der Einfachheit halber $n_{max} = 127$ gewählt, da dies genau dem MIDI-Standard entspricht und daher die Wiedergabe ohnehin mit eben dieser Auflösung erfolgt.

Eine Sequenz von Ticks ist ein Muster. Eine Menge von Mustern verschiedener Instrumente ist ein Pattern. Somit können die Genotypen der Patterns intern als zweidimensionales Integer-Array mit den Dimensionen Ticks und Instrumente kodiert werden.

Im Gegensatz zu der frei wählbaren Anzahl an Ticks ist die Dimension der Instrumente unveränderlich. Diese Einschränkung dient der Begrenzung der Komplexität des Systems. Um ein typisches Jazzschlagzeug zu simulieren, werden die sechs Instrumente Bass Drum, Hihat, Snare, Ride-Becken, tief gestimmte Stand-Tom-Tom (lowtom) und höher gestimmte Hänge-Tom-Tom (hightom) genutzt. Der Benutzer hat lediglich die Möglichkeit, die Anzahl der Instrumente nach oben zu beschränken. Wählt er beispielsweise eine Anzahl von zwei Instrumenten, so entfallen die vier letztgenannten und nur Bass Drum und Hihat bleiben übrig.

Abbildung 9.2 zeigt ein mögliches Pattern in seiner Repräsentation als Genotyp mit beispielhaften Markierungen der genutzten Begrifflichkeiten.

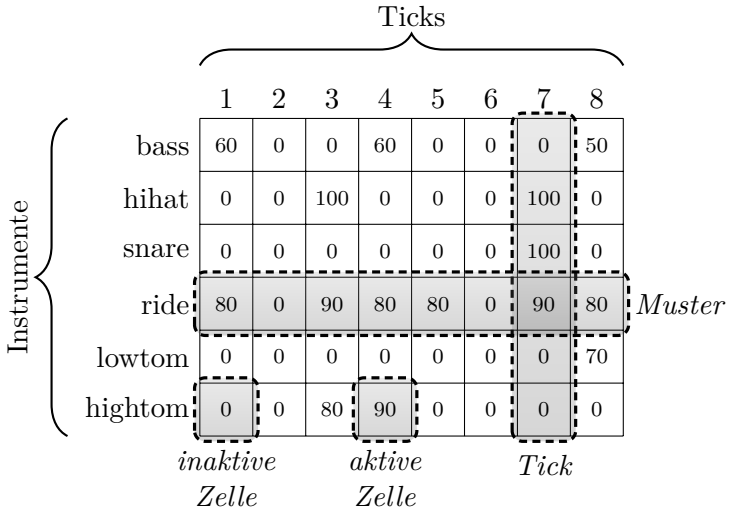


Abbildung 9.2.: Genotyp eines Schlagzeug-Patterns

9.2. Initialisierung

Beim Start des Systems legt der Benutzer die Anzahl der Ticks fest. Jedes Pattern, das innerhalb dieser Sitzung erzeugt wird, bekommt genau diese Anzahl an Ticks. Der Benutzer erstellt anschließend ein initiales Pattern, das an seinem musikalischen Vorhaben orientiert

sein sollte, indem er über die GUI die einzelnen Zellen des Arrays, das das initiale Pattern repräsentiert, mit Werten füllt. Bevor der EA gestartet wird, kann das Pattern bereits mithilfe des Playback-Moduls abgespielt und überprüft werden.

Sobald das initiale Pattern hinreichend abgestimmt ist, kann der Benutzer den EA starten. Bevor der Zyklus eintritt, wird eine vorgeschaltete Initialisierung vorgenommen. Diese führt folgende Aktionen durch:

Zuerst wird die erste Population an Individuen erzeugt. Dazu werden Kopien des initialen Patterns in die Generation geschrieben. Die Anzahl der Kopien und somit die Populationsgröße N_p ist vom Benutzer einstellbar.

Außerdem wird das Input-Window restlos geleert, da bereits vor Start des EA Noten eingegangen sein könnten, die nicht berücksichtigt werden sollen.

Nach der Initialisierung startet der Zyklus.

9.3. Reproduktion durch Mutation

Die Reproduktion neuer Individuen ist eine elementare Operation für einen EA. Hier entscheidet sich, ob Individuen entstehen, die im Vergleich zu ihrer Elterngeneration eine höhere Fitness besitzen. Nur wenn dies der Fall ist, ist es sinnvoll, die alten Individuen zu ersetzen. So gelingt die Optimierung der Population. Es gibt dabei zwei mögliche Reproduktionsvarianten: Die Mutation und die Rekombination.

Die Rekombination konnte im Hinblick auf die Variation der Schlagzeug-Patterns keine zufriedenstellenden Ergebnisse liefern. Der Grund dafür liegt in der Initialisierung der Startpopulation. Die Startpopulation besteht aus Kopien des gleichen initialen Patterns. Die Länge aller Patterns muss konstant bleiben. Bei konstanter Länge erzeugen die klassische Punkt-Rekombination und auch Mittelwertsverfahren jedoch keine Variationen. Deshalb wird keine Rekombination im implementierten EA verwendet.

Stattdessen kommt Mutation zum Einsatz. Diese soll möglichst einfach gehalten werden, da die Bewertung der Individuen bereits komplex ist und deshalb Rechenzeit benötigt. Die Mutation soll Nachkommen ohne aufwendige Berechnungen erzeugen.

9.3.1. Funktionsweise des Operators

Die Funktionsweise des Mutations-Operators ist einfach. Es wird zufällig und gleichverteilt eine Zelle des zu mutierenden Patterns ausgewählt und ihr Wert durch eine gleichverteilte Zufallszahl aus 0–127 ersetzt. Das so entstandene Pattern hat eine große Ähnlichkeit mit seinem Vorgänger. Der musikalische Zusammenhang bleibt dadurch gewahrt.

Die Mutation ist die Abbildung einer typischen Benutzerinteraktion bei der Drum Machine: Der Benutzer einer Drum Machine verändert nach und nach einzelne Zellen des Patterns, indem er entsprechende Tasten auf dem Interface drückt. Der Mutations-Operator arbeitet nach diesem Vorbild.

Nach der Anwendung des Mutations-Operators wird überprüft, ob sich einer der Zellenwerte von seinem vorherigen Wert unterscheidet. Erst dann ist die Mutation vollendet. Sollte das zufälligerweise nicht der Fall sein, so wird die Anwendung so lange wiederholt, bis eine Variation des Patterns eintritt.

Die Wahrscheinlichkeit, dass ein Pattern nach vollendeter Mutation weniger aktive Zellen besitzt als vorher, liegt bei $\frac{1}{128} < 0{,}8\%$. Viel wahrscheinlicher ist also die Schaffung neuer aktiver Zellen. Eine Verringerung kann aber musikalisch gewollt sein und muss deshalb auch vom Mutations-Operator mit hinreichender Wahrscheinlichkeit erzeugt werden. Deshalb wird die Wahrscheinlichkeit für die Entstehung einer inaktiven Zelle erhöht. Es wird angenommen, dass besonders leise Schläge nicht gehört werden oder gar störend im Kontext des Patterns sind. Alle Zellen, deren Werte eine bestimmte Grenze G_{stumm} unterschreiten, werden deshalb auf 0 gesetzt. Die Wahrscheinlichkeit für eine inaktive Zelle berechnet sich dann durch $\frac{G_{stumm}}{128}$. Im implementierten System ist $G_{stumm} = 32$. Das ergibt eine Wahrscheinlichkeit für die Entstehung inaktiver Zellen von 25%. Diese Festlegung erfolgt aus der empirischen Erfahrung, dass wohlklingende Patterns zu circa 25% aus Pausen bestehen sollten.

9.3.2. Anwendungsprinzip des Operators

Ziel der Reproduktion ist das Erzeugen von Nachkommen, die anschließend selektiert und in die Population zurückgeführt werden. Dabei ist es möglich, dass ein Pattern, das im vorherigen Durchgang hinreichend gut war, auch im folgenden selektiert wird. Pausiert das Solo beispielsweise, so soll es möglich sein, das initiale Pattern während der Pause in der Population zu behalten und abzuspielen, ohne es zu verändern. Deshalb wird die aktuelle Generation unverändert in die Menge der Nachkommen aufgenommen. Sie müssen allerdings der erneuten Bewertung durch Kriterien, die sich aufgrund des veränderten Solo-Ausschnitts verändert haben. Deshalb wäre es nicht sinnvoll zu sagen, sie werden in der Population behalten. Deshalb wäre es falsch, von Nachkommen mit identischer Struktur zu sprechen. Eine Kopie der aktuellen Generation bildet demnach den Grundstock der zu erzeugenden Menge an Nachkommen. Eine Selektion der Eltern entfällt somit an dieser Stelle.

Ausschlaggebend für das Erzeugen von Nachkommen ist die Expansionsgrenze G_{exp}. Diese kann zur Laufzeit vom Benutzer verändert werden. Er kann so Einfluss auf die Variationsvielfalt nehmen. Dabei ist G_{exp} die maximale Anzahl an Nachkommen, die erzeugt werden.

Die Anzahl der Nachkommen wird durch den Algorithmus in Abbildung 9.3 auf die Größe $|N| = G_{exp}$ gebracht. Mithilfe dieses Algorithmus wird eine Menge erzeugt, die eine besondere Eigenschaft erfüllt: Es befinden sich mindestens doppelt so viele Individuen in der Menge, die i-fach mutiert wurden, wie Individuen, die $i + 1$-fach mutiert wurden (für $i > 0$). Die Menge hält für die Selektion demnach prinzipiell mehr Individuen bereit, die sich gering von der aktuellen Generation unterscheiden, und weniger Individuen, die sich

stark von der aktuellen Generation entfernt haben. Über den Parameter G_{exp} lässt sich deshalb nicht nur die Anzahl der Nachkommen, sondern ebenso auch ihre Variationstiefe einstellen.

Seien N_i durchnummerierte leere Mengen mit $i \in \mathbb{N}_0$.
input : N_0, die aktuelle Generation
output: N, die Menge der Nachkommen
$n := |N_0|$;
while $n < G_{exp}$ **do**
  $i := 0$;
  while $\frac{|N_i|}{2} > |N_{i+1}|$ ***und*** $n < G_{exp}$ **do**
    Erstelle zufällig eine Kopie e eines beliebigen Elements aus N_i;
    Wende den Mutations-Operator auf e an;
    Füge e zu N_{i+1} hinzu;
    $i := i + 1$;
  end
end
return $N := \{e | e \in N_i \text{ mit } i \in \mathbb{N}_0\}$

Abbildung 9.3.: Algorithmus zur Anwendung des Mutations-Operators

9.4. Evaluation

Bei der Evaluation wird jedes Individuum nach seiner Güte im Verhältnis zu den anderen Individuen einer Population bewertet. Dazu muss jedem Individuum ein Fitnesswert zugewiesen werden. Der implementierte EA nutzt ganzzahlige Werte, die sich numerisch vergleichen lassen.

9.4.1. Bestimmung heuristischer Merkmale

Die Bewertung der Individuen erfolgt unter Verwendung heuristischer Merkmale eines Solo-Ausschnitts. Der Ausschnitt ist eine Menge von MelodyNote-Objekten, die Noten repräsentieren und sich im Input-Window befinden. Das Input-Window wird in jedem Zyklus des EA nach der Mutation ausgelesen. Um im nächsten Durchgang einen neuen Ausschnitt betrachten zu können, werden Noten, die ein bestimmtes Alter erreicht haben, aus dem Input-Window entfernt. Das Alter einer Note ist die Zeit, die seit ihrer Aufzeichnung durch den Input-Receiver verstrichen ist. Wie alt eine Note sein muss, um entfernt zu werden, bestimmt der Parameter T_w, der vom Benutzer verändert werden kann. So kann die Größe des betrachteten Ausschnitts, die die zeitliche Länge des Gedächtnisses des virtuellen Schlagzeugers beschreibt, an die Vorstellungen des Benutzers angepasst werden. Ein größerer Ausschnitt ermöglicht eine Reaktion in einem größeren musikalischen Kontext, allerdings auch erhöhte Rechenzeit. Ein zu kleiner Ausschnitt enthält im schlimmsten Falle keine sinnvollen musikalischen Informationen mehr.

Die Analyse, die zur Fitnesszuweisung notwendig ist, basiert auf heuristischen Merkmalen, die als musikalisch aussagekräftig angesehen werden. Die Merkmale werden für einen Solo-Ausschnitt, also für eine Menge von Noten, bestimmt. Folgende Merkmale werden extrahiert:

- $\#_{\textbf{Noten}}$: Anzahl der Noten.
- $\#_{\textbf{Noten(okt)}}$: Anzahl der Noten ohne Berücksichtigung der Oktavzugehörigkeit.
- $\#_{\textbf{diffNoten}}$: Anzahl der Noten mit unterschiedlichen Notennummern.
- $\#_{\textbf{diffNoten(okt)}}$: Anzahl der Noten mit unterschiedlichen Notennummern ohne Berücksichtigung der Oktavzugehörigkeit.

Zu den folgenden Eigenschaften werden jeweils Minimum, Maximum, Spannweite[1] R, arithmetisches Mittel[2] $\overline{x}$ und Standardabweichung[3] σ über alle Noten des Ausschnitts berechnet:

- **Länge (leng)**: Zeitliche Länge einer Note.
- **Notennummer (note)**: Zuordnung zu den Tasten des Klaviers.
- **Lautstärke (vol)**: Die Lautstärke einer Note.
- **Intervall (int)**: Das Intervall zwischen zwei aufeinanderfolgenden Noten.
- **Distanz (dist)**: Die Zeit zwischen den Startzeitpunkten zweier aufeinanderfolgender Noten.
- **Lücke (gap)**: Die Zeit zwischen dem Endzeitpunkt einer Note bis zum Startzeitpunkt der darauffolgenden Note.

9.4.2. Gewichtete Regeln

Die statistische Beschreibung der heuristischen Merkmale soll mithilfe eines Regelwerks interpretiert werden, das in der Lage ist, Schlagzeug-Patterns im Hinblick auf ihre musikalische Kompatibilität zu Solo-Ausschnitten zu bewerten. Dazu werden unabhängige Regelfunktionen definiert, die anhand der bestimmten Merkmale für ein Schlagzeug-Pattern einen Fitnesswert berechnen. Jede Funktion bestimmt sich durch eine heuristisch definierte musikalische Aufgabe, die der virtuelle Schlagzeuger erfüllen soll. Jedes Pattern aus der Menge der Nachkommen wird einmal von jeder Regelfunktion bewertet. Die Gesamtbewertung ergibt sich aus der Summe der Einzelbewertungen.

[1] Spannweite $R = x_{max} - x_{min}$
[2] Arithmetisches Mittel $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
[3] Standardabweichung $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2}$

Die Funktionsweisen der Regeln sind nach folgendem Prinzip entworfen: Es werden musikalische Eigenschaften definiert, die sich in den heuristischen Merkmalen widerspiegeln. Ist deren Ausprägung im Solo-Ausschnitt hoch, so werden Patterns mit ähnlichen Eigenschaften hoch bewertet. Die Regeln teilen einen nahezu einheitlichen Aufbau. Zu Beginn wird bei fast allen Regeln anhand der heuristischen Merkmale eine Bedingung geprüft, die über die Ausführung der Berechnung bestimmt. Ist die Bedingung nicht erfüllt, so erhält das Pattern keine Bewertung. Ansonsten folgt die Ermittlung zweier Faktoren aus $\mathbb{R}$, die die Ausprägung der gewünschten Eigenschaften im Solo (f_s) und im Pattern (f_p) beschreiben. Sie liegen zwischen 0 (geringe Ausprägung) und 1 (hohe Ausprägung). Mithilfe der heuristischen Merkmale wird f_s berechnet. Zur Berechnung von f_p wird die Patternmatrix $P_{i,j}$ analysiert. Der Fitnesswert f errechnet sich in den meisten Fällen aus dem arithmetischen Mittel der beiden Faktoren.

Da die Darstellung der Fitnesswerte in Fließkommazahlen im Programmcode Probleme beim Vergleich und bei der Summierung bereitet, erfolgt intern eine Abbildung auf Integerwerte durch Multiplikation mit einer Konstanten.

Die Gestaltung der Regeln ist die subjektivste Entscheidung, die beim Entwurf des Systems gefällt werden muss. Sie bestimmt grundlegend das Verhalten des virtuellen Schlagzeugers. Deshalb soll auch hier die Möglichkeit zur Anpassung durch den jeweiligen Benutzer möglichst vielfältig sein. Die Regeln erhalten dazu eine Gewichtung, die zur Laufzeit verändert werden kann. Ein Gewicht ist ein reellwertiger Faktor $g \in [0,1]$, das mit dem berechneten Fitnesswert der Regelfunktion multipliziert wird. Somit kann der Einfluss einer Regel abgeschwächt oder gar eine Regel deaktiviert werden, während der Einfluss anderer Regeln verstärkt wird.

Die Regeln sind in die drei Klassen *Keep*, *Reaction* und *Random* eingeteilt. In den nächsten Abschnitten folgt die Erläuterung aller implementierten Regeln.

9.4.2.1. Die Keep-Regeln

Die Keep-Regeln bewerten die Patterns im Hinblick auf ihre Distanz zum initialen Pattern. Sie sorgen dafür, dass die erzeugten Variationen eine musikalische Bindung zum initialen Pattern behalten. Diese Regeln arbeiten ständig, da sie keine aktivierende Bedingung prüfen. Ist das Solo verstummt und somit die Arbeit der Reaction-Regeln unterbrochen, dann nähern sich die Individuen der Population wieder dem initialen Pattern und damit auch einander an. Das hat eine musikalische Entspannung zur Folge, die für den Kontrast zwischen niedriger und hoher Intensität des Dialogs sorgt. Die Keep-Regeln berechnen keinen Solo-Faktor und bewerten die Patterns somit unabhängig vom Solo.

Im folgenden sei $P_{i,j}$ die Matrix des zu bewertenden Patterns mit $i \in \{0, \ldots, I-1\}$, wobei I die Anzahl der Ticks ist, und $j \in \{0, \ldots, J-1\}$, wobei J die Anzahl der Instrumente ist. Außerdem sei $A_{i,j}$ die Matrix des initialen Patterns.

KeepOriginal-Regel

- **Beschreibung**: Die KeepOriginal-Regel bewertet ein Pattern P nach der mathematischen Distanz zum initialen Pattern A. Je größer die Distanz, desto geringer die Fitness. Für jede Zelle wird die Distanz berechnet und anschließend zur Normierung durch die größtmögliche Distanz geteilt.

  Mithilfe der KeepOriginal-Regel bleibt das initiale Pattern erhalten und kehrt bei inaktiven Reaction-Regeln vollständig zurück.

- **Bedingungen**: —

- **Solo-Faktor**: —

- **Pattern-Faktor**:

$$f_p = 1 - \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \frac{|P_{i,j} - A_{i,j}|}{MAX\{A_{i,j}, 127 - A_{i,j}\}}$$

- **Fitness**: $f = f_p$

KeepInstruments-Regel

- **Beschreibung**: Die KeepInstruments-Regel bewertet Patterns, die exakt die gleichen Instrumente wie das initiale Pattern nutzen, höher als andere. Die Idee ist, dass Patterns mit ähnlichen Instrumenten einen höheren musikalischen Zusammenhang aufweisen.

- **Bedingungen**: —

- **Solo-Faktor**: —

- **Pattern-Faktor**:

$$f_p = 1 - \frac{1}{J} \sum_{j=0}^{J-1} XOR\left(\sum_{i=0}^{I-1} P_{i,j}, \sum_{i=0}^{I-1} A_{i,j}\right)$$

$$XOR(x, y) := \begin{cases} 1, & (x \neq 0 \wedge y = 0) \vee (x = 0 \wedge y \neq 0) \\ 0, & sonst \end{cases}$$

- **Fitness**: $f = f_p$

KeepTicks-Regel

- **Beschreibung**: Die KeepTicks-Regel stellt eine Umkehrung der KeepInstruments-Regel dar. Die Idee ist, dass Patterns, die Schläge auf den gleichen Ticks haben und somit das gleiche Muster besitzen, als musikalisch zusammenhängend empfunden werden.

- **Bedingungen**: —
- **Solo-Faktor**: —
- **Pattern-Faktor**:

$$f_p = 1 - \frac{1}{I}\sum_{i=0}^{I-1} XOR\left(\sum_{j=0}^{J-1} P_{i,j}, \sum_{j=0}^{J-1} A_{i,j}\right)$$

- **Fitness**: $f = f_p$

9.4.2.2. Die Reaction-Regeln

Die Reaction-Regeln sorgen für eine musikalische Anpassung der Patterns an die Gestalt des momentanen Solo-Ausschnitts. Sie haben eine musikalische Funktion im Kontext des Dialogs von Solo und Begleitung. Deshalb muss das Solo bestimmte Merkmale besitzen, damit eine Regel Anwendung findet (Bedingung).

Chromatic-Regel

- **Beschreibung**: Chromatik ist die Veränderung eines Tones um einen Halbton (kleinstmögliche Veränderung). Enthält das Solo viel Chromatik, ist eine Begleitung mit möglichst wenigen Pausen erwünscht.

  Die Bedingung der Regel lautet, dass mindestens zwei Noten und als kleinstes Intervall die kleine Sekunde (entspricht einem Halbton) vorhanden sind. Der Solo-Faktor bestimmt sich wie folgt: Je kleiner das durchschnittliche Intervall, desto größer der Faktor. Der Pattern-Faktor wächst mit der Anzahl der belegten Ticks. Ein Tick ist belegt, wenn eine Zelle des Ticks einen Wert größer als die Hälfte der durchschnittlichen Sololautstärke hat. Ist das Pattern lückenlos, ist der Pattern-Faktor 1.

- **Bedingungen**: $\#_{Noten} > 1 \wedge Min_{int} \leq 1$
- **Solo-Faktor**:

$$f_s = \frac{1}{MAX\{1, \overline{x}_{int}\}}$$

- **Pattern-Faktor**:

$$f_p = \frac{1}{I}\sum_{i=0}^{I-1} \Theta_1\left(\sum_{j=0}^{J-1} h_{0,5}(P_{i,j})\right)$$

$$\text{Sprungfunktion: } \Theta_n(x) := \begin{cases} 1, & x \geq n \\ 0, & x < n \end{cases}$$

$$h_n(x) := \begin{cases} 1, & x > n \cdot \overline{x}_{vol} \\ 0, & sonst \end{cases}$$

- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

FreeJazz-Regel

- **Beschreibung**: Eine normale Tonart und ihre Tonleiter haben sieben Töne. Sind in einem Solo-Ausschnitt mehr als sieben verschiedene Töne zu finden, bedeutet das, dass offenbar Tonart-fremde Töne gespielt wurden – wie es im Free Jazz häufig der Fall ist. Dort ist eine unkonventionelle Begleitung gewünscht.

  Die Patterns werden zufällig bewertet, wodurch die Begleitung unvorhersehbar wird. Der Solo-Faktor ist umso höher, je mehr unterschiedliche Töne benutzt werden. Da $\#_{diffNoten(okt)}$ höchstens 12 werden kann, ist der Bruch maximal $0{,}8\overline{3}$ und ist dann, von 1,08 abgezogen, nahezu 1.

- **Bedingungen**: $\#_{diffNoten(okt)} > 7$

- **Solo-Faktor**:
$$f_s = 1{,}08 - \left(\frac{1}{\#_{diffNoten(okt)}}\right)$$

- **Pattern-Faktor**: Zufallszahl $r \in [0,1]$

- **Fitness**: $f = \frac{1}{2}(f_s + r)$

Holdsworth-Regel

- **Beschreibung**: Die Regel lehnt sich an die Spielpraxis des englischen Jazzgitarristen Allan Holdsworth an. Er ist für die Verwendung großer Intervalle in seinen Soli bekannt. Weil das für Soli eher untypisch ist, soll der virtuelle Schlagzeuger in diesem Fall speziell reagieren.

  Werden im Solo Intervalle von durchschnittlich mindestens 5 Halbtönen gespielt, bewertet die Holdsworth-Regel Patterns hoch, die auf jedem Tick den Akzent auf einem anderen Instrument haben als auf dem vorherigen. Je größer die Intervalle, desto höher fällt die Fitness aus. Ein Akzent ist hier ein Tick, der eine Zelle mit einem Wert größer der momentan durchschnittlichen Sololautstärke besitzt.

- **Bedingungen**: $\#_{Noten} > 1 \wedge \overline{x} \geq 5$

- **Solo-Faktor**:
$$f_s = MIN\left\{1, \frac{\overline{x}_{int}}{12}\right\}$$

- **Pattern-Faktor**:
$$\frac{1}{I}\sum_{i=1}^{I-1} 1 - id\Big(akz_P(i-1), akz_P(i)\Big)$$

$$\text{Gleichheitsfunktion: } id(x,y) := \begin{cases} 1, & x = y \geq 0 \\ 0, & sonst \end{cases}$$

$$\text{Akzentfunktion: } akz_P(x) := \begin{cases} J-1, & P_{x,J-1} \geq MAX\{P_{x,0},..,P_{x,J-1}\} \wedge P_{x,J-1} > \overline{x}_{vol} \\ \vdots & \vdots \\ 0, & P_{x,0} \geq MAX\{P_{x,0},..,P_{x,J-1}\} \wedge P_{x,0} > \overline{x}_{vol} \\ -1, & sonst \end{cases}$$

- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

Legato-Regel

- **Beschreibung**: Legato ist eine musikalische Vortragsanweisung und bedeutet, dass die Töne gebunden gespielt werden sollen. Der Musiker versucht, die Noten ohne Pausen dazwischen zu spielen. Die Begleitung soll sich daran anpassen und ebenfalls möglichst lückenlos ausfallen.

  Legato wird von der Regel erkannt, indem die Lücke zwischen den Noten analysiert wird. Je kleiner die Lücke, desto höher der Solo-Faktor. Der Pattern-Faktor ist derselbe wie bei der Chromatic-Regel.

- **Bedingungen**: $\#_{Noten} > 1$

- **Solo-Faktor**:
$$f_s = \frac{1}{\sqrt{MAX\{1, \overline{x}_{gap}\}}}$$

- **Pattern-Faktor**: f_p, siehe Chromatic-Regel.

- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

Loudness-Regel

- **Beschreibung**: Die Loudness-Regel bewertet Patterns hoch, deren mittlere Lautstärke wenig von der mittleren Lautstärke des Solos abweicht. So wird die Lautstärke der Patterns an die des Solos angepasst.

  Der Fitness-Faktor berechnet sich hier nicht durch das arithmetische Mittel, sondern über die Distanz der mittleren Lautstärken.

- **Bedingungen**: $\#_{Noten} > 0$

- **Solo-Faktor**: $\overline{x}_{vol}$

- **Pattern-Faktor**:

$$\frac{1}{\#_{aktiv}} \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} P_{i,j}$$

$$\text{Anzahl der aktiven Zellen von P: } \#_{aktiv} = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} \Theta_1(P_{i,j})$$

- **Fitness (!)**: $1 - \frac{1}{127}|\overline{x}_{vol} - f_p|$

Ostinato-Regel

- **Beschreibung**: Ein Ostinato ist in der Musik ein sich ununterbrochen wiederholendes Motiv. In einem Solo bewirkt die schnelle Wiederholung von Noten (Repeating Pattern) eine Intensitätssteigerung. Diese soll sich auch in der Begleitung wiederfinden.

  Die Bedingung der Regel ist, das in einem Solo-Ausschnitt dreimal so viele Noten vorkommen, wie unterschiedliche Tonhöhen auftreten. Je weniger unterschiedliche Tonhöhen und je mehr Tonwiederholungen im Solo-Ausschnitt auftreten, desto höher ist der Solo-Faktor. Die Patterns werden dahingehend optimiert, dass mindestens zwei Akzente auf jedem Tick liegen.

- **Bedingungen**: $\#_{Noten} > 0 \wedge (3 \cdot \#_{Noten}) \geq \#_{diffNoten}$

- **Solo-Faktor**:

$$f_s = 1 - \frac{MAX\{1, \#_{diffNoten} - 3\}}{\#_{Noten}}$$

- **Pattern-Faktor**:

$$f_p = \frac{1}{(f'_p)^2}$$

$$f'_p = \left| I - \sum_{i=0}^{I-1} \Theta_2 \left(\sum_{j=0}^{J-1} h_1\left(P_{i,j}\right) \right) \right| + 1$$

- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

Pedal-Regel

- **Beschreibung**: Pedaltöne sind besonders lang gehaltene Töne und sollen Patterns mit Pausen erzeugen.

  Je länger eine Note, desto höher ist der Solo-Faktor. Ist eine Note länger als 5 Sekunden, so ist der Faktor 1. Der Pattern-Faktor zählt die Ticks mit Pausen (stumme Ticks) und wird 1, wenn das Verhältnis zwischen Pausen und Schlägen mindestens 2:1 beträgt.

- **Bedingungen**: $\#_{Noten} > 0$
- **Solo-Faktor**:
$$\frac{MIN\{5000, \overline{x}_{leng}\}}{5000}$$
- **Pattern-Faktor**:
$$f_p = MIN\left\{1, \frac{3 \cdot \#_{stummeTicks}}{2 \cdot I}\right\}$$
$$\#_{stummeTicks} = I - \sum_{i=0}^{I-1} \Theta_1\left(\sum_{j=0}^{J-1} P_{i,j}\right)$$
- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

Staccato-Regel

- **Beschreibung**: Die Vortragsanweisung Staccato bedeutet in der Musik, dass Noten besonders kurz gespielt und nicht übergebunden werden. Damit bildet Staccato das Gegenteil zum Legato. Staccato vorgetragene Passagen wirken oft hektisch. Diese Wirkung soll von der Begleitung unterstützt werden.

  Eine Note soll als Staccato erkannt werden, wenn ihre Distanz zur nächsten Note mindestens doppelt so hoch ist wie ihre Länge. Außerdem wird auf Grundlage empirischer Betrachtungen festgelegt, dass sie kürzer als 380ms sein muss. Bei einer Länge von unter 80ms wird der Solo-Faktor 1. Die Patterns werden dahingehend optimiert, dass sie durchschnittlich alle 3 Ticks einen harten Akzent aufweisen. Ein Tick hat einen harten Akzent, wenn eine seiner Zellen einen Wert von mindestens 110 aufweist.
- **Bedingungen**: $\#_{Noten} > 0 \wedge 2 \cdot \overline{x}_{leng} \leq \overline{x}_{dist} \wedge \overline{x}_{leng} \leq 380$
- **Solo-Faktor**:
$$f_s = 1 - \frac{MAX(80, \overline{x}_{leng}) - 80}{380}$$
- **Pattern-Faktor**:
$$f_p = \frac{1}{(f'_p)}$$
$$f'_p = \left|\frac{I}{3} - \sum_{i=0}^{I-1} \Theta_1\left(\sum_{j=0}^{J-1} \Theta_{110}\Big(P_{i,j}\Big)\right)\right| + 1$$
- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

Virtuoso-Regel

- **Beschreibung**: Virtuosität in der Musik ist die technische Beherrschung eines Instruments in Perfektion. Damit einher geht meist die Fähigkeit, in sehr hoher Geschwindigkeit zu spielen. Zu einem virtuosen Solo kann die Begleitung einen musikalisch sinnvollen Beitrag leisten, indem sie die Zahl der genutzten Instrumente erweitert. Das bewirkt einen vielfältigeren Höreindruck.

  Je mehr Töne im Solo-Ausschnitt liegen, desto größer ist der Solo-Faktor. Der Pattern-Faktor wird umso höher, je mehr Instrumente verwendet werden, die nicht im initialen Pattern vorkommen. Ein Instrument gilt hier als verwendet, wenn sein Muster mindestens eine Zelle besitzt, die einen Wert größer als die durchschnittliche Lautstärke des Solo-Ausschnitts aufweist. Belegt das initiale Pattern schon alle möglichen Instrumente, werden alle Patterns gleich bewertet. Es ist daher ratsam, nicht alle Instrumente im initialen Pattern zu belegen.

- **Bedingungen**: $\#_{Noten} > 0$

- **Solo-Faktor**:
$$f_s = \Big(1 - \frac{1}{\#_{Noten}}\Big)^4$$

- **Pattern-Faktor**:
$$f_p = \frac{\sum_{j=0}^{J-1} AND\Big(\sum_{i=0}^{I-1} 1 - \Theta_1(A_{i,j}), \sum_{i=0}^{I-1} h_1(P_{i,j})\Big)}{\sum_{j=0}^{J-1} \Theta_1\Big(\sum_{i=0}^{I-1} 1 - \Theta_1(A_{i,j})\Big)}$$

$$AND(x, y) := \begin{cases} 1, & x \neq 0 \wedge y \neq 0 \\ 0, & sonst \end{cases}$$

- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

Wide-Regel

- **Beschreibung**: Benutzt der Solist besonders hohe und tiefe Noten innerhalb eines kurzen Zeitraums, bewirkt das eine Intensitätssteigerung, die vom virtuellen Schlagzeuger erwidert werden soll. Es sollen Patterns erzeugt werden, die jedes Instrument mindestens einmal benutzen, um die hohe Streuung an Tönen auf eine hohe Klangvielfalt des Schlagzeugs zu übertragen.

  Der Solo-Faktor wird 1, wenn die Spannweite zwischen niedrigster Note und höchster Note im Solo-Ausschnitt 36 (entspricht 3 Oktaven) erreicht. Der Pattern-Faktor steigt mit der Anzahl der verwendeten Instrumente.

- **Bedingungen**: $\#_{Noten} > 0$

- **Solo-Faktor**:
$$f_s = \frac{MIN\{36, R_{note}\}}{36}$$
- **Pattern-Faktor**:
$$f_p = \frac{1}{J}\sum_{j=0}^{J-1}\Theta_1\left(\sum_{i=0}^{I-1}P_{i,j}\right)$$
- **Fitness**: $f = \frac{1}{2}(f_s + f_p)$

9.4.2.3. Die Random-Regel

- **Beschreibung**: Die Random-Regel ist die primitivste Regel des Regelwerks. Sie erzeugt eine Zufallszahl und gibt diese als Bewertung aus. Solo und Pattern werden schlicht ignoriert. Mithilfe der Random-Regel kann der Benutzer dem virtuellen Schlagzeuger eine gewisse Unvorhersehbarkeit erlauben. Sie wird möglicherweise als kreativer Impuls im musikalischen Dialog wahrgenommen und bewirkt Abwechslung, falls die Population einmal zu einheitlich ausfällt.
- **Bedingungen**: —
- **Solo-Faktor**: —
- **Pattern-Faktor**: —
- **Fitness**: Zufallszahl $r \in [0,1]$.

9.5. Selektion

Da sich die Optimierungsgrundlage aufgrund des sich verändernden Solos ständig ändert und aufgrund der Problemstellung die Festlegung des Optimums subjektiv ist, wird eine einfache Form der Selektion gewählt. Um aus den Nachkommen Individuen für die nächste Generation zu bestimmen, nutzt der implementierte EA das Prinzip der *Bestenselektion*. Diese Selektion wählt N_p-mal das Individuum mit dem höchsten Fitnesswert unter den Nachkommen aus und fügt es zur neuen Generation hinzu. Gibt es mehrere mögliche Individuen aufgrund identischer Fitness, so wird zufällig gewählt. Die Population hat nach der Selektion die Größe N_p.

Der Benutzer des Systems kann N_p frei wählen und zur Laufzeit verändern. Er passt so die untere Schranke der Kompatibilität zwischen Solo-Ausschnitt und Pattern an, denn es gilt: Je größer N_p, desto minderwertigere Individuen befinden sich in der Generation. Jene Individuen mit geringerer Fitness stellen aber gleichzeitig Freiheiten des virtuellen Schlagzeugers gegenüber der Begleitung des Solos dar und werden möglicherweise als kreativer Impuls wahrgenommen.

10. Playback-Modul

Das Playback-Modul steuert die Wiedergabe der Schlagzeug-Patterns, die sich in der Generation befinden. Ein virtuelles Metronom, das der Benutzer einstellt, synchronisiert die Erzeugung von *RhythmNote*-Objekten durch den Pattern-Player mit dem gewünschten Tempo. Die Klasse RhythmNote ist ein Modell von rhythmischen Noten. Der Schlagzeug-Generator verarbeitet die RhythmNote-Objekte und erzeugt daraus MIDI-Messages, die vom Output-Generator an den externen Schlagzeug-Synthesizer versendet werden.

Abbildung 10.1 zeigt den schematischen Aufbau des Moduls. Es folgt die Erläuterung der einzelnen Bestandteile.

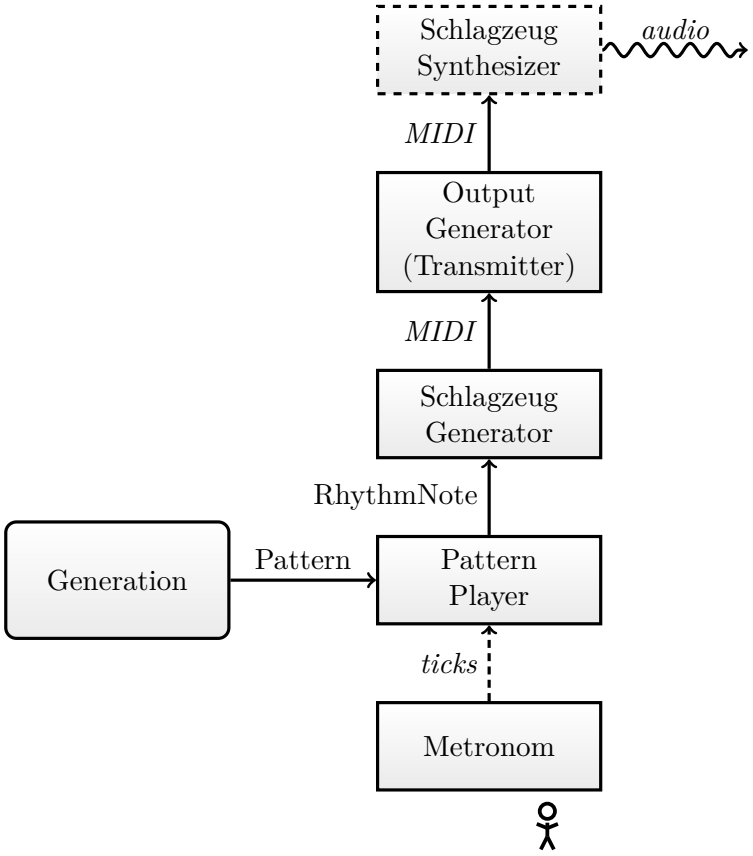


Abbildung 10.1.: Schematischer Aufbau des Playback-Moduls

10.1. Der Puls des Metronoms

Das virtuelle Metronom des implementierten Systems funktioniert ähnlich einem klassischem Metronom. Da das System allerdings keine Taktarten kennt, sondern nach dem Prinzip einer Drum Machine aufgebaut ist, werden als rhythmische Einheit statt bpm *Ticks-per-Minute* (tpm) eingestellt, wobei ein Tick die kleinste rhythmische Einheit der Patterns darstellt. Die Ticks geben den Taktschlag vor. Für einen 4/4-Takt, der durch 8 Ticks (Achtel) dargestellt wird und mit 120bpm abgespielt werden soll, müssen demnach 240tpm eingestellt werden.

Das Metronom gibt seine Ticks nach dem Beobachter-Muster aus. Es berechnet die Wartezeit zwischen zwei Ticks. Nach Ablauf der Zeit informiert es alle seine Beobachter, in diesem Fall der Pattern-Player und zur visuellen Nachvollziehbarkeit auch die GUI, über die Nummer des aktuellen Ticks.

Die Berechnung der Wartezeit in Millisekunden ergibt sich bei einem gleichmäßigen Tempo durch folgende Formel:

$$t_\Delta = \frac{60000\text{ms}}{TPM}$$

Das Metronom bietet allerdings auch die Möglichkeit, die Ticks im *Shuffle* abzuspielen. Shuffle bedeutet die zeitliche Verschiebung jeder zweiten Note um einen konstanten Faktor und ist zum Beispiel ein typisches Stilmittel im Swing und Funk. Mathematisch bedeutet Shuffle, dass jede zweite Wartezeit $t_{\Delta,2}$ kürzer ist als die vorherige $t_{\Delta,1}$. Mit dem Faktor $f_{shuffle}$ ergeben sich die Formeln:

$$t_{\Delta,1} = \frac{2f_{shuffle} \cdot 60000\text{ms}}{TPM}$$

$$\text{und } t_{\Delta,2} = \frac{2(1 - f_{shuffle}) \cdot 60000\text{ms}}{TPM}$$

$$\text{mit } f_{shuffle} \in [0{,}5, 1] \Rightarrow t_{\Delta,1} \geq t_{\Delta,2}$$

Über die GUI kann der Benutzer das Tempo TPM und den Shuffle-Faktor $f_{shuffle}$ frei wählen und so die Wiedergabe der Patterns nach seinen Vorstellungen gestalten. Er hat auch die Möglichkeit, das Metronom zu starten und zu stoppen und so die Wiedergabe zu unterbrechen.

10.2. Wiedergabe der Patterns durch den Pattern-Player

Der Pattern-Player hält immer ein aktuelles Pattern, das zur Wiedergabe genutzt wird. Er ist ein Beobachter des Metronoms und wird somit informiert, wenn ein neuer Tick gespielt werden soll. Immer wenn dieser Tick die Nummer 0 hat (erster Tick), erfragt er ein beliebiges Pattern aus der Generation, das das aktuelle Pattern ersetzt. Wie oft

das passiert, ist also durch den Zyklus des Metronoms vorgegeben. Da die Dauer eines Evolutionszyklus jedoch weit unter der des Metronomzyklus liegen kann, wird zusätzlich auf jedem Tick geprüft, ob die Generation erneuert wurde. Anstatt das aktuelle Pattern bis zum Ende zu spielen, wird dann bereits ein neues erfragt. Der Pattern-Player stellt so sicher, dass er auf dem neuesten Stand der Evolution bleibt und somit jederzeit die besten Patterns zur Verfügung hat.

Der Pattern-Player erzeugt für alle aktiven Zellen des aktuellen Patterns auf dem aktuellen Tick RhythmNote-Objekte, die Rhythmusnoten darstellen. Diese besitzen Informationen über das gespielte Instrument und die Lautstärke. Für einen Tick können entsprechend der Vorgabe der 6 Instrumente demnach maximal 6 RhyhtmNote-Objekte erstellt werden. Die Objekte werden anschließend an den Schlagzeug-Generator gegeben, der daraus MIDI-Daten erzeugt.

10.3. MIDI-Generierung aus RhythmNotes

Der Output-Generator ist ein Transmitter im Sinne der Java Sound API. Er kennt einen Receiver (in diesem Fall ein MIDI-Gerät zur Synthese von Schlagzeugsounds) und schickt auf Aufforderung des Schlagzeug-Generators MIDI-Daten in Form von MIDI-Messages an seinen Receiver.

Der Schlagzeug-Generator ist für die Umwandlung von RhythmNote-Objekten in MIDI-Daten verantwortlich. Er erhält die Objekte vom Pattern-Player und sendet sie umgewandelt in MIDI-Daten an den Output-Generator.

Da es sich um die Generierung perkussiver Klänge handelt, werden lediglich *NoteOn*-Kommandos benötigt. Der Schlagzeug-Channel ist ebenfalls gemäß General MIDI auf 10 festgelegt. Die Lautstärke kann direkt von RhythmNote erfragt werden.

Lediglich die Umwandlung der intern genutzten sechs Instrumente auf MIDI-Notennummern bedarf einer Zuordnung. Die genutzte Zuordnung ist in Abbildung 10.2 aufgeführt. Zu beachten ist, dass es auch einen intelligenten Zuordnungsmodus gibt, der ein Instrument abhängig von der gespielten Lautstärke auf unterschiedliche MIDI-Instrumente abbildet. So kann ein Becken beispielsweise bei geringer Lautstärke mit einem Ride-Sound und bei hoher mit einem Crash-Sound erklingen. Die Ausdrucksmöglichkeiten des virtuellen Schlagzeugers werden dadurch vielfältiger.

internes Instrument	Lautstärke	MIDI-Instrument	MIDI-Nummer
Bass	—	**Acoustic Bass Drum**	**35**
Hihat	< 70	Pedal Hihat	44
Hihat	70–120	**Closed Hihat**	**42**
Hihat	> 120	Open Hihat	46
Snare	< 70	Side Stick	37
Snare	≥ 70	**Electric Snare**	**40**
Ride	< 110	**Ride Cymbal 1**	**51**
Ride	110–121	Ride Bell	53
Ride	122-126	Crash Cymbal 1	49
Ride	> 126	Chinese Cymbal	52
LowTom	—	**High Floor Tom**	**43**
HighTom	—	**High Mid Tom**	**48**

Abbildung 10.2.: Zuordnung der intern genutzten Instrumente zu MIDI-Notennummern
Ohne intelligente Zuordnung werden die Instrumente
auf die **fettgedruckten** MIDI-Nummern abgebildet.

11. Evaluation

Der Erfolg des Systems soll anhand von aktiven Testpersonen (Musikern) sowie passiven Testpersonen (Zuhörern) evaluiert werden. Die Hauptmerkmale für den Erfolg sollen die Akzeptanz der musikalischen Erzeugnisse und der von den Musikern empfundenen Nutzen sein. Zudem soll erprobt werden, inwieweit das Verhalten des virtuellen Schlagzeugers durch die veränderbaren Parameter an die spontanen Wünsche der Benutzer angepasst werden kann. Abbildung 11.1 zeigt zur Übersicht die Liste aller Parameter.

Parameter	Intervall[1]	Trägermenge
initiales Pattern	$P_{i,j} \in [0, 127]$	$\mathbb{N}$
Populationsgröße	$N_p \in [1, \infty)$	$\mathbb{N}$
Zykluszeit [ms]	$T_z \in [0, \infty)$	$\mathbb{N}$
Expansionsgrenze	$G_{exp} \in [0, \infty)$	$\mathbb{N}$
Größe des Input-Windows [ms]	$T_w \in [0, \infty)$	$\mathbb{N}$
Gewicht einer Regel	$g \in [0, 1]$	$\mathbb{R}$
Ticks-Per-Minute	$TPM \in [1, \infty)$	$\mathbb{N}$
Shuffle-Faktor	$f_{shuffle} \in [0{,}5, 1]$	$\mathbb{R}$

Abbildung 11.1.: Auflistung aller zur Laufzeit veränderbaren Parameter

Die aktiven Testpersonen waren zwei Pianisten, zwei Gitarristen, ein Bassist und ein Saxophonist. Dabei konnte nur für die Pianisten eine exakte Darstellung des Gespielten in MIDI-Noten erzeugt werden. Die Soli der anderen Instrumente wurden durch den Gebrauch des Audio-MIDI-Wandler verfälscht. Festzustellen war dabei, dass diese Ungenauigkeiten nicht zu entscheidenden Fehlern in der Analyse durch die Regeln führen. Die Ergebnisse können daher für jedes akustische Instrument übernommen werden, das mit einem hinreichend hochwertigen Wandler benutzt wird.

Getestet wurde wie folgt: Die aktiven Testpersonen wurden gebeten, unter Begleitung des virtuellen Schlagzeugers zu solieren. Anschließend wurden Kritikpunkte geäußert und daraus Verbesserungsvorschläge erstellt. Es folgten weitere Testläufe mit veränderten Parametern, aus denen ermittelt wurde, ob sich bei bestimmten Einstellungen eine Verbesserung bei den genannten Kritikpunkten einstellt. Da die Kritik zum Teil sehr subjektiv

[1] Da ein Computer nur endlich viele Zahlen darstellen kann, entspricht ∞ im implementierten System dem maximalen Wert der Integers in Java von 2.147.483.647.

und speziell ausfiel, werden im folgenden nur wiederkehrende Motive in der Kritik aller Testpersonen diskutiert.

Bei jedem Test wurde der Wunsch nach einer geringen Reaktionszeit des Schlagzeugers auf Veränderung des Solos geäußert. Das wird grundsätzlich durch eine möglichst kleine Zykluszeit T_z erreicht. Bei besonders kleinen $T_z < 30$ms kam es bei zeitintensiven Analysen oftmals zur Überschreitung der Zykluszeit. Die Testpersonen beschrieben allerdings bei $T_z = 30$ms die Reaktionszeit als unmittelbar und somit als hinreichend gering. Zusätzlich können die Expansionsgrenze G_{exp} erhöht und die Größe des Input-Windows T_w verringert werden. Ein hohes G_{exp} erzeugt viele variantenreiche Nachkommen, sodass die Unterschiede der aufeinanderfolgenden Generationen erhöht werden, wodurch eine schnelle Veränderung der Patterns eintritt. Allerdings wird die Anzahl der Evaluationen erhöht und somit die Zykluszeit. Mit T_w um die 1000ms waren alle Testpersonen zufrieden und $T_z = 30$ms konnte eingehalten werden. Bei 100ms $< T_w <$ 1000ms äußerten die Testpersonen eine gute Balance zwischen direkter Ansprache auf das Solo und Erkennung von Solostruktur. Bei $T_w < 100$ms konnte das System keine sinnvollen Solo-Ausschnitte mehr erkennen. Die meisten waren leer, sodass zwischen zwei Noten Stille erkannt wurde, die eine ungewollte Entspannung der Begleitung zur Folge hatte. Zudem muss stets $T_w \geq T_z$ gelten, da sonst Noten unbeachtet aus dem Input-Window entfernt werden. Für $T_w > 2000$ms wurde eine Erkennung größerer musikalischer Strukturen beobachtet, jedoch war die Reaktionszeit des Schlagzeugers für die meisten Testpersonen nicht mehr akzeptabel.

Als besonders gut wurde die Entspannung der Begleitung und die Wiederkehr des initialen Patterns bei geringer Reaktionszeit während einer Unterbrechung des Solos bezeichnet. Allerdings sollte die Random-Regel stets ein zumindest geringes Gewicht bekommen, damit bei Stillstand des Solos kleine Variationen des initialen Patterns entstehen können. Der Schlagzeuger wirkt sonst unselbstständig.

Die Testpersonen erwarteten aus ihrer musikalischen Vorerfahrung, dass der virtuelle Schlagzeuger nicht ausschließlich reagiert, sondern den Dialog zuweilen auch selbstständig gestaltet. Stärkeres Gewichten der Random-Regel brachte die Entwicklung zufälliger Patterns, die kreativ, da nicht durch die Eingabe bestimmt, bezeichnet wurden. Je höher die Gewichtung, desto selbstständiger agiert der Schlagzeuger. Jedoch ist der Grad der Selbstständigkeit dabei konstant, was wiederum nicht dem von den Testpersonen gewünschten Geben und Nehmen im musikalischen Dialog mit echten Menschen entsprach. Als Alternative stellte sich die Vergrößerung der Population heraus. Bei kleiner T_z wird meist nicht einmal ein Pattern komplett abgespielt, bevor die Population ersetzt wird. Deshalb ist die Wahl von $N_p = 1$ grundsätzlich sinnvoll. Soll den abgespielten Patterns allerdings eine gewisse Unschärfe gegeben werden, so kann N_p hoch gewählt werden. Bei $N_p = 1000$ und $G_{exp} = 2000$ sind bereits oft Patterns mit relativ geringer Fitness unter den Individuen der Generation. Da aus ihnen zufällig gewählt wird, wird manchmal ein besonders gut

zum Solo passendes Pattern und manchmal ein schlechter passendes und somit als kreativ empfundenes Pattern gespielt.

Oft wurden fehlende Abwechslung der Begleitung und die regelmäßige Wiederkehr ähnlicher Muster kritisiert. Die Patternvariationen erschienen schnell ausgeschöpft. Abhilfe brachte eine höhere Gewichtung der Random-Regel, jedoch bringt das, wie bereits erwähnt, auch eine gewisse Unschärfe, die möglicherweise nicht gewollt ist. Eine Verbesserung brachten überraschenderweise die Verdoppelung der Anzahl der Ticks für die Patterns und eine schlichte zweifache Eingabe des initialen Patterns hintereinander. Die Regeln schienen dadurch eine größere Vielfalt zu entwickeln. Das liegt vermutlich an dem Übergang vom letzten zum ersten Tick, der im System einen Sprung darstellt, weil die Patterns für sich und nicht im Gesamtzusammenhang ihrer Hintereinanderreihung analysiert werden. Empfunden wird aber ein Schlagzeugspiel, das die Patterns fließend ineinander übergehen lässt. Bei der mehrfachen Wiederholung des gewünschten Patterns in einem initialen Individuum, das mehrere Takte repräsentiert, wird der Sprung zwischen Patterns nur alle paar Takte ausgeführt. Die Übergänge sind fließender und zugleich vielfältiger, weil sie in die Analyse mit einbezogen werden. Allerdings muss G_{exp} und somit die Kosten der Berechnung erhöht werden, da mehr Mutationen auf einem Individuum ausgeführt werden müssen, um die Variationsvielfalt zu wahren.

Ebenfalls wurde eine rhythmische Primitivität aufgrund der statischen Auflösung der Patterns empfunden. Die Schläge passieren stets auf den Ticks. Ein menschlicher Schlagzeuger würde jedoch vermutlich öfter auch Schläge zwischen den Ticks ausführen. Das System ist dazu aufgrund der Wahl des Prinzips der Drum Machine nicht in der Lage. Jedoch ist es möglich, von vornherein eine höhere Auflösung zu wählen und das initiale Pattern nur durch Ticks einer geringeren Auflösung zu definieren. Dann können bei der Variation durch den EA jene Ticks zwischen den definierten belegt werden. Die Testpersonen bezeichneten Momente des Auftretens solcher Ticks als Schlagzeug-Fills, was als besonders hochwertiges Strukturmerkmal einer Schlagzeugbegleitung gesehen werden kann. Ein großes Problem stellte dabei allerdings eine auftretende Überfüllung der Patterns dar, wobei der Schlagzeuger als zu wild bezeichnet wurde. Dies konnte durch die hohe Gewichtung der KeepTicks-Regel verbessert werden.

Die bisher vorgestellten Kritikpunkte wurden hauptsächlich mithilfe eines zur Vergleichbarkeit einheitlich gewählten Swing-Patterns bei Shuffle-Faktor $f_{shuffle} = 0{,}67$ erkannt, jedoch durch Anwendung anderer Patterns bestätigt. Die Möglichkeit, bei hohen Tempi $f_{shuffle}$ zu verringern, wurde von den Testpersonen als sinnvolle Maßnahme für die realistische Wiedergabe einer Swing-Begleitung bezeichnet. Bei enorm hohen Tempi von über 250bpm wurde stets $f_{shuffle} = 0{,}5$ gewählt.

Ein besonders beliebtes Pattern war der Bossa Nova. Aufgrund von geraden Rhythmen mit $f_{shuffle} = 0{,}5$ und durchgehenden Mustern auf allen Instrumenten wurde dieses Pattern als mitreißend beschrieben. Der akzentuierte Einsatz von Toms, der für das implementierte System typisch scheint, passt besonders gut zur Stilistik des Bossa Nova.

Natürlich wurde auch oft versucht, das System an seine Grenzen zu bringen und experimentelle Musik zu schaffen. Dazu wurden beispielsweise ungerade Taktarten, zufällige und ungewöhnliche Regelgewichte, unstrukturierte initiale Pattern und auch ungewöhnliche Schlagzeug-Synthesizer-Klänge gewählt. So konnte äußerst innovative Musik erzeugt werden. Diese Möglichkeit, das System sowohl zur leichten Variation als auch zur starken Verfremdung der Patterns aufgrund eigener Kalibrierung verwenden zu können, wurde insgesamt besonders positiv hervorgehoben.

Was allerdings nicht erzeugt werden konnte, waren größere musikalische Strukturen. Der virtuelle Schlagzeuger hat kein Gedächtnis und er plant nicht. Er versagt deshalb bei komplexeren Aufgaben wie dem Erkennen einer Komposition und ihrer spontanen Begleitung und wird dabei insbesondere von den Zuhörern als störend empfunden. Die spontane Improvisation im Moment, wie sie oft im Jazz praktiziert wird, beherrscht er dafür gut. Dort ordneten Zuhörer die Erzeugnisse als musikalisch sinnvoll ein.

Die aktiven Testpersonen stimmten der Frage, ob sie sich vorstellen könnten, das System zum Üben zu verwenden, uneingeschränkt zu. Sie sahen einen Ersatz für primitive Drum Machines und erklärten, das Üben mithilfe des Systems sei interessanter, spannender, unterhaltsamer und abwechslungsreicher. Zuhörer empfanden die erzeugte Musik sogar als hochwertig, wenn der Solist zusätzlich zum System noch von Harmonieinstrumenten wie Klavier und Gitarre und/oder einem Bass begleitet wurde. Die resultierende Performance zeigte Konzertqualität.

Das grundlegende Konzept des implementierten Systems, eine interaktive automatische Begleitung zu erzeugen, kann abschließend als sinnvoll bewertet werden. Die Wahl eines evolutionären Algorithmus zeigt gute Ergebnisse in Akzeptanz und Nutzen für aktive und passive Testpersonen.

Teil III.

Fazit

12. Zusammenfassung

Ziel der Arbeit war die Entwicklung eines reaktiven Systems, das mithilfe eines evolutionären Algorithmus eine rhythmische, musikalisch sinnvolle und dynamische Begleitung für ein Jazz-Solo in Echtzeit erzeugt. Die Idee entstand aus dem Wunsch, ein interaktives Playalong zum Üben nutzen zu können.

Durch die Auseinandersetzung mit verschiedensten Arbeiten auf dem musikalisch kreativen Gebiet der Musikgenerierung wurden die ersten Entwurfsentscheidungen getroffen, darunter die Nutzung eines EA als zentrales Instrument. Der EA ist dafür am besten geeignet, weil er eine zufallsbasierte approximative Optimierung vornimmt, die als Prozess einer künstlichen Kreativität gesehen werden kann. Er sucht nach dem Vorbild der natürlichen Evolution die am besten an einen momentanen Solo-Ausschnitt angepassten Schlagzeug-Patterns. Zudem wird die verbreitete Schnittstelle MIDI zur Kommunikation des Systems mit Musikinstrumenten genutzt. Die Festlegung auf die Implementierung nur eines einzigen virtuellen Musikers, im Fall des Systems eines Schlagzeugers, dient der Begrenzung der Komplexität der Arbeit. Aufgrund der Einfachheit der Java Sound API im Umgang mit MIDI wird Java als Programmiersprache gewählt. Zudem wird das implementierte System dadurch plattformübergreifend nutzbar.

Der Systementwurf sieht eine Unterteilung des System in drei autonome Module vor. Das Input-Modul regelt die Verarbeitung des eingehenden MIDI-Stroms. Dieser wird von einem Musiker, der ein Jazz-Solo spielt, durch ein MIDI-fähiges Instrument erzeugt. Ein akustisches Instrument kann mithilfe eines Audio-MIDI-Wandlers MIDI-Fähigkeit erlangen. Tests zeigen dabei keine deutlichen Qualitätseinbußen bei der erzeugten Begleitung durch die leicht ungenaue Umwandlung. Das System ist deshalb mit jedem beliebigen Instrument verwendbar. Das Input-Modul wandelt die empfangenen MIDI-Messages in MelodyNote-Objekte um, deren Modell sich mehr an einer musikalischen Note orientiert, als es MIDI-Daten tun.

Die MelodyNotes wandern in eine verwaltende Datenstruktur zur Zwischenspeicherung, das Input-Window, um dort vom Genetic-Modul bei Bedarf abgefragt und wieder entfernt zu werden. Das Genetic-Modul beinhaltet den EA des Systems. Die Individuen des EA sind Schlagzeug-Patterns, deren Genotyp-Repräsentation aus einem zweidimensionalen Array mit den Dimensionen Ticks und Instrumente besteht. Die einzelnen Zellen sind mit einer Anschlagsstärke belegt. Der Benutzer stellt das initiale Pattern über ein an klassische Drum Machines angelehntes grafisches Interface ein. Kopien dieses initialen Patterns bilden die Startpopulation.

Die Produktion von Nachkommen erfolgt ausschließlich mithilfe von Mutation. Als Operator wird eine einfache Punkt-Mutation verwendet. Ein besonderes Anwendungsprinzip des Operators ermöglicht dem Benutzer die einfache Einstellung der Variationstiefe, da diese proportional mit der einstellbaren Anzahl der Nachkommen zunimmt.

Die Evaluation bewertet die Patterns, indem sie ihnen im Hinblick auf den aktuellen Solo-Ausschnitt im Input-Window einen Fitnesswert zuordnet. Grundsätzlich soll dabei gelten: Je besser ein Pattern das Solo begleitet, desto höher ist die Fitness. Da die Zuordnung keine triviale Funktion ist, sondern eine komplexe Berechnung bestimmt von Expertenwissen und subjektivem Geschmack, wird ein Regelwerk entworfen. Dieses besteht aus einzelnen Regeln, die anhand heuristisch bestimmter Merkmale des Solos sowie eines Patterns Fitnesswerte berechnen. Die Fitness eines Patterns ist die Summe aller Bewertungen durch die Regeln. Eine Gewichtung der Regeln durch den Benutzer ermöglicht dabei die Anpassung des Verhaltens des virtuellen Schlagzeugers an persönliche Vorlieben. Bei Tests mit Musikern konnte gezeigt werden, dass verschiedene Wünsche nach Verhaltensänderungen durch passende Gewichtungen erfüllt werden konnten. Die Patterns mit der höchsten Fitness werden für die nächste Generation selektiert.

Das Playback-Modul ist für die Wiedergabe der Patterns in der aktuellen Generation zuständig. Ein virtuelles Metronom, das vom Benutzer bedient wird, sorgt für die synchronisierte Umwandlung der im Pattern enthaltenen Informationen in MIDI-Messages. Diese werden an einen Schlagzeug-Synthesizer geschickt, der anhand der Messages für den Benutzer und die potentiellen Zuhörer hörbare Schlagzeug-Klänge erzeugt.

Das System wird von allen Testpersonen grundsätzlich als praktisch und hilfreich beschrieben. Die Akzeptanz als musikalischer Mitspieler ist hinreichend hoch, um als nützliche Unterstützung und Bereicherung für das tägliche Üben gesehen zu werden.

13. Ausblick

Der Entwurf und die Implementierung eines künstlerisch kreativen Systems ist in sich ebenfalls ein Prozess, der die Kreativität des Entwicklers fordert. Wann immer ein Problem auftritt, muss eine innovative Lösung gefunden werden. Je ungewöhnlicher diese ist, desto interessantere Erzeugnisse sind zu erwarten. Es ist daher nicht verwunderlich, dass während der Entwicklung dieser Arbeit eine Vielzahl von Ideen aufgrund der Faktoren Zeit und Kosten nicht realisiert werden konnten. Einige erwähnenswerte sollen nun abschließend diskutiert werden.

Das aufwendigste Element der Entwicklung war eindeutig der Entwurf des Regelwerks. Die entstandenen 14 Regeln wirken in ihrer Unveränderlichkeit etwas starr und endgültig. Obwohl sie durch ihre Gewichte angepasst werden können, müssen sie dem Benutzer grundsätzlich zusagen. Findet er die Regelkonzepte abwegig, so ist das System für ihn unbrauchbar. Deshalb sollte die Anzahl der Regeln im Sinne einer Verbesserung zumindest erhöht werden, um Benutzern mehr Auswahlmöglichkeiten zu geben. Besser noch wäre, für die Regeln einstellbare Parameter zur Veränderung der Funktionsweisen anzubieten. Da die meisten Regeln sich über zwei unabhängige Faktoren (Solo-Faktor und Pattern-Faktor) definieren, könnte das Erstellen eigener Regeln durch Kombination verschiedener Faktoren ermöglicht werden.

Die umfassendste Methode zur Individualisierung des Regelwerks ist selbstverständlich die Programmierung eigener Regeln. Das System enthält intern bereits eine Klasse **RuleManager**, die Regel-Objekte hält, die von einer abstrakten Klasse **Rule** erben. Für einen erfahrenen Java-Programmierer ist es ein Leichtes, die abstrakte Methode `rate()` mit einer Fitnessberechnung zu füllen und den bestimmten Fitnesswert durch die Methode `rateWeighted()` gewichten und einem Pattern zuweisen zu lassen. Da das System hauptsächlich für Musiker entwickelt wurde, bei denen keine Programmiererfahrung vorausgesetzt werden kann, steht diese Möglichkeit für einen durchschnittlichen Benutzer nicht zur Verfügung.

Darüber hinaus bietet der EA weitere Elemente, deren Umgestaltung getestet werden sollte. Die einfache Punkt-Mutation als einzige Reproduktionsvariante wurde als sinnvolle Wahl erachtet. Dennoch sollte die Anwendung anderer Mutations-Operatoren genauer untersucht werden. Mögliche Operatoren sind Krebs (horizontale Spiegelung), Rotation (horizontale Verschiebung), Sortierung der Ticks nach der Lautstärke oder die zufällige Umsortierung der Ticks oder auch der Muster.

Die Wahl eines anderen Selektionsverfahrens könnte ebenso dabei helfen, die als kreativ empfundene Unvorhersehbarkeit des virtuellen Schlagzeugers zu erhöhen. Eine Turnierselektion zeigt die nötige Unschärfe und könnte beispielsweise mithilfe der Parameter n_T und k_T vom Benutzer entsprechend seiner Vorstellungen kalibriert werden.

Ein großer Kritikpunkt beim System ist die statische Anzahl der Ticks. Die Rhythmen wirken dadurch äußerst maschinell. Deshalb könnte an dieser Stelle die Erzeugung von *Zwischenticks* bei der Mutation erlaubt werden. Die zeitliche Auflösung der Patterns könnte dadurch dynamisch erhöht werden, wodurch musikalische Strukturen wie Fills besser abgebildet würden. Die Länge der Individuen wäre dann dynamisch, sodass auch Rekombinations-Operatoren Verwendung finden könnten.

Zwei Ideen für einfache Erweiterungen des Systems wurden bei den Tests durch die Testpersonen entwickelt. Das Pattern sollte unveränderliche Muster besitzen. Ein menschlicher Schlagzeuger würde bei einem Solo beispielsweise auch ein für eine Stilistik typisches Muster durchgehend auf einem seiner Instrumente spielen. Das Pattern bekommt dadurch mehr Konsistenz und ist verständlicher. Außerdem sollten andere Instrumente des Schlagzeugs auswählbar sein, um beispielsweise mehr als zwei Toms oder auch Perkussionsinstrumente wie Conga, Bongo, Cowbell oder Woodblock in ein Pattern zu integrieren.

Zur Extraktion der Merkmale wird im implementieren System ausschließlich ein kleiner Solo-Ausschnitt betrachtet. Typisch für Improvisationen vor allem im Jazz ist aber auch die Berücksichtigung größerer musikalischer Zusammenhänge. Beispielsweise müssen das Thema, der Zyklus der Chorusse sowie Anfang und Ende eines Solos erkannt und begleitet werden. Das rein reaktive Verhalten des Systems reicht dafür nicht aus. Eine komplexe Analyse unter Berücksichtigung der kompletten bereits gespielten Musik ist dafür von Nöten und kann wohl von keinem aktuellen System geleistet werden.

Da der virtuelle Schlagzeuger sozusagen Teil eines interaktiven Playalongs sein soll, müssten selbstverständlich auch weitere virtuelle Musiker wie beispielsweise ein Bassist und ein Pianist entwickelt werden. Erst dann wäre das eigentliche Ziel erreicht, beim Üben eine virtuelle Begleitband zur Verfügung zu haben. Dazu könnte die Entwicklung eines ähnlichen Systems mit evolutionärem Algorithmus auf konzeptionellen Erfolg überprüft werden.

Durch all diese Erweiterungen könnte ein mächtiges System entstehen, dass den übenden Jazzschüler dabei unterstützt, seinem Traum ein ganzes Stückchen näher zu kommen – einmal ein Solo wie Miles Davis oder John Coltrane spielen zu können.

A. Überblick über die Benutzeroberfläche

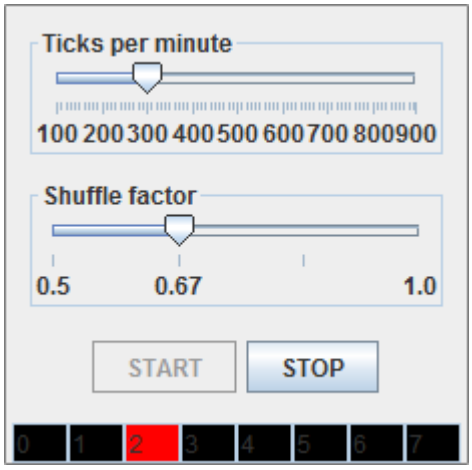


Abbildung A.1.: Steuerung des Metronoms

Abbildung A.2.: Eingabe des initialen Patterns
Rechtsklick auf eine Zelle öffnet einen Popup-Menu, in dem ein Wert für die Zelle eingestellt werden kann. Der Wert jeder Zelle ist proportional zu seiner Helligkeit, um die Lesbarkeit zu erhöhen. Der aktuelle Tick ist rot hinterlegt.

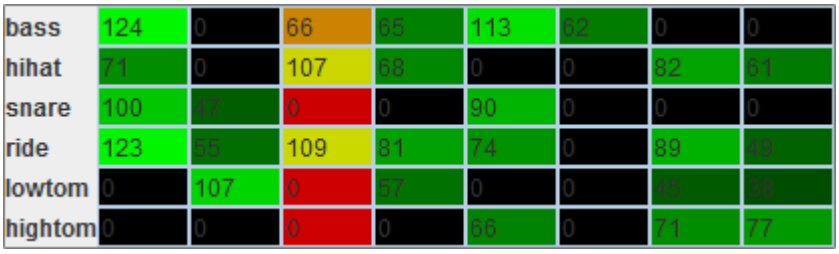


Abbildung A.3.: Anzeige des aktuellen Patterns

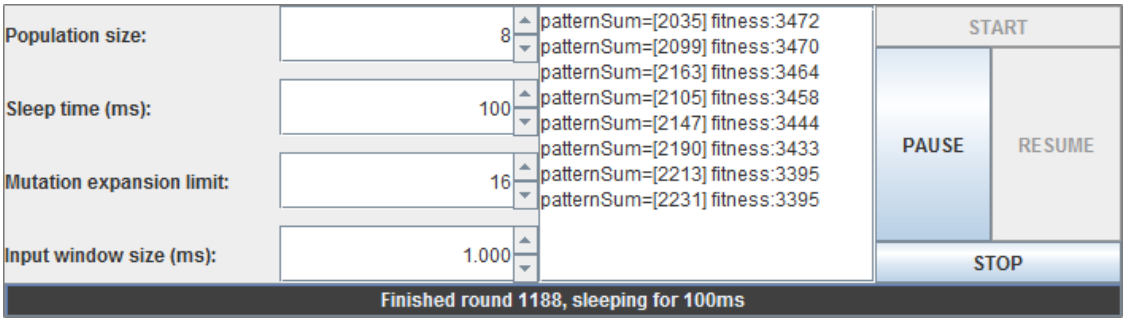


Abbildung A.4.: Steuerung des evolutionären Algorithmus
Die vier Evolutionsparameter N_p, T_z, G_{exp} und T_w können hier eingestellt werden. Die Anzeige in der Mitte zeigt für jedes Pattern der Population die Fitness und in eckigen Klammern die Summe aller Zellen als Aktivitätsmaß an. Die Steuerung rechts startet, stoppt und unterbricht den Evolutionsprozess. Die Leiste unten zeigt den aktuellen Status der Evolution an.

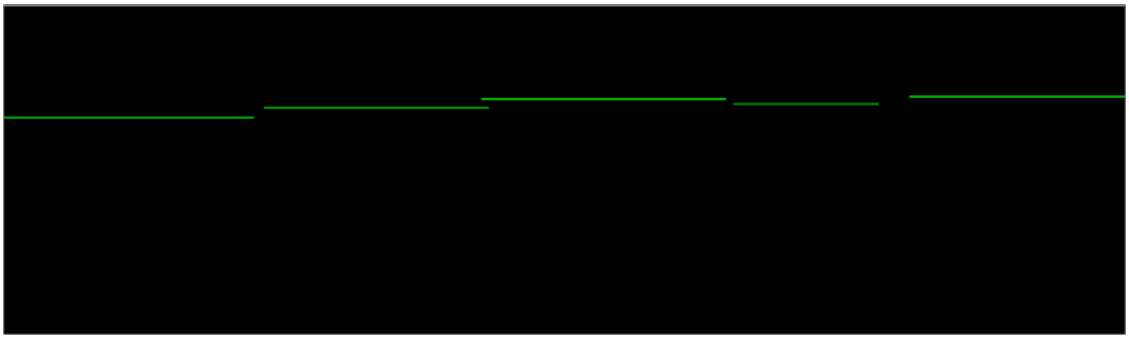


Abbildung A.5.: Anzeige des Inhalts im Input-Window
Die grünen Striche repräsentieren die MelodyNote-Objekte im Input-Window. Die genauen Attribute der Noten sind dabei nicht abzulesen. Die Anzeige dient lediglich der visuellen Verifikation des Eingabestroms. Die vertikale Achse zeigt die relative Tonhöhe der Noten an. Die horizontale Achse zeigt die Zeit immer im Bereich zwischen erster und letzter Note. Die Helligkeit ist proportional zur Lautstärke.

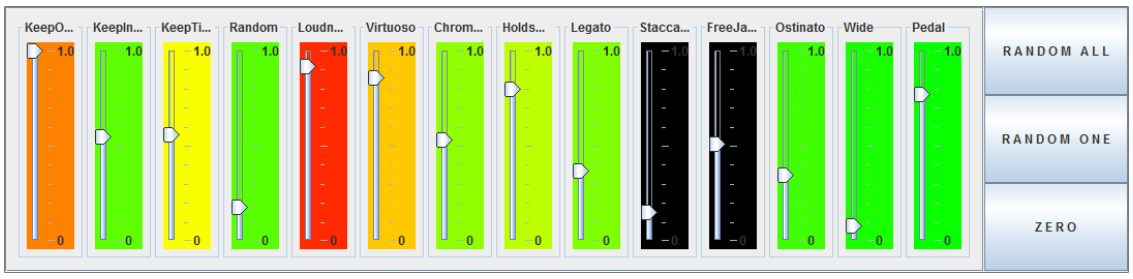


Abbildung A.6.: Gewichtung der Regeln
Mithilfe der Regler werden die Regeln gewichtet. Durch die Metapher des „Heiß-Laufens“ wird der momentane Einfluss der Regeln visualisiert: Schwarz bedeutet inaktiv, Grün eine Bewertung durch niedrige Fitnesswerte, Gelb mittlere Fitness und Rot maximale Fitness. Der „RANDOM ALL“-Button gewichtet alle Regeln zufällig. Der „RANDOM ONE“-Button gewichtet eine beliebige Regel zufällig. Der „ZERO“-Button setzt alle Regelgewichte auf 0.

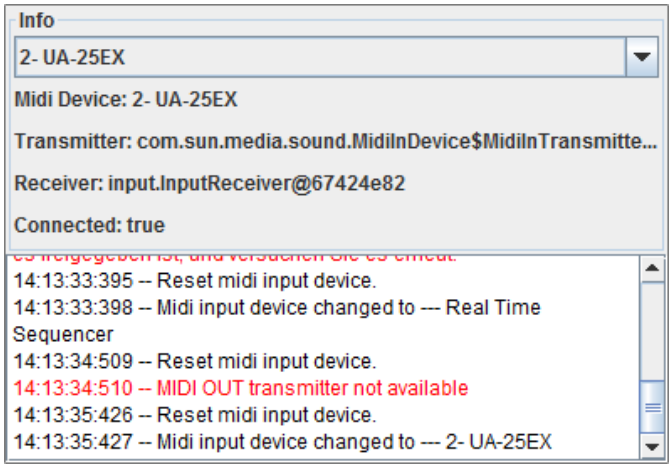


Abbildung A.7.: Auswahl des Eingabegerätes
Hier wird das MIDI-fähige Instrument ausgewählt.

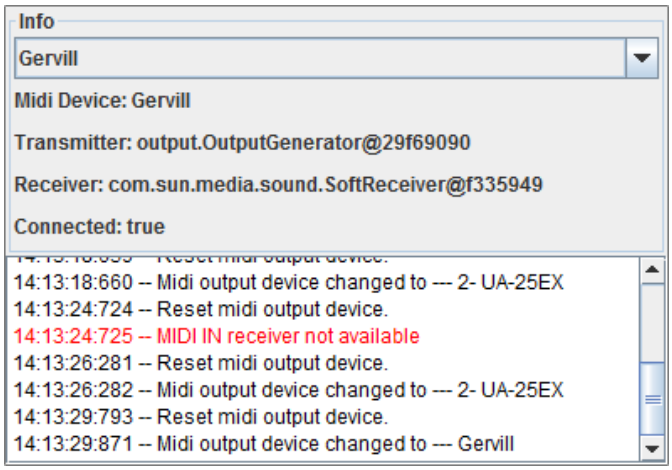


Abbildung A.8.: Auswahl des Ausgabegerätes
Hier wird der Schlagzeug-Synthesizer ausgewählt.

Abbildung A.9.: Fensterverwaltung
Mithilfe der Fensterverwaltung können ungenutzte Fenster ausgeblendet und geschlossene Fenster wiedergeholt werden.

Abbildungsverzeichnis

Literaturverzeichnis

[Aebersold 1967] Aebersold, Jamey: *VOLUME 1 - HOW TO PLAY JAZZ & IMPROVISE.* Jamey Aebersold Jazz, 1967 (Jamey Aebersold Play-A-Long Series)

[Axelrod 1987] Axelrod, Robert M.: *Die Evolution der Kooperation.* München : Oldenbourg Verlag, 1987

[Bienert 1967] Bienert, Peter: *Aufbau einer Optimierungsautomatik für drei Parameter*, Technische Universität Berlin, Institut für Mess- und Regelungstechnik, Diplomarbeit, 1967

[Biles 1994] Biles, John A.: GenJam: A Genetic Algorithm for Generating Jazz Solos. In: *Proceedings of the International Computer Music Conference (ICMC 1994).* San Francisco, USA : International Computer Association, 1994, S. 131–137

[Biles 1998] Biles, John A.: Interactive GenJam: Integrating Real-Time Performance with a Genetic Algorithm. In: *Proceedings of the International Computer Music Conference (ICMA 1998).* San Francisco, USA : International Computer Association, 1998, S. 131–137

[Biles 2001] Biles, John A.: *Autonomous GenJam: Eliminating the Fitness Bottleneck by Eliminating Fitness.* 2001 `http://igm.rit.edu/~jabics//GECCO01/index.html`. – abgerufen am 16.10.2015

[Biolcati 2008] Biolcati, Massimo: *iReal Pro.* New York, USA : Technimo LLC, 2008

[Chen 2002] Chen, Shu-Heng: *Evolutionary Computation in Economics and Finance.* AI-ECON Research Center, Department of Economice, National Chengchi University, Taipei, Taiwan : Springer Verlag, 2002

[Collins 2010] Collins, Nick: *Introduction to Computer Music.* Chichester, UK : John Wiley & Sons Ltd, 2010

[Collomosse u. Hall 2005] Collomosse, John P. ; Hall, Peter M.: Genetic Paint: A Search for Salient Paintings. In: *Applications of Evolutionary Computing* 3449 (2005)

[Darwin 1859] Darwin, Charles R.: *On the Origin of Species.* London : John Murray, 1859

[Dostál 2005] Dostál, Martin: Genetic Algorithms as a Model of Musical Creativity – on Generating of a Human-Like Rhythmic Accompaniment. In: *Computing and Informatics*, 2005 (22), S. 321–340

[Ebner 2008] Ebner, Marc: Evolutionäre Bildverarbeitung. In: *Informatik-Spektrum* 31 (2008), Nr. 2

[Fogel u. Corne 2002] Fogel, Gary B. ; Corne, David W.: *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann Publishers, 2002

[Fogel u. a. 1966] Fogel, Lawrence J. ; Owen, Alvin J. ; Walsh, Michael J.: *Artificial Intelligence Through Simulated Evolution*. New York, USA : John Wiley & Sons, Inc., 1966

[Gannon 1990] Gannon, Peter: *Band-in-a-Box*. Hamilton, Ontario : PG Music, Inc., 1990

[Gerhard 2003] Gerhard, David: Pitch Extraction and Fundamental Frequency: History and Current Techniques / Department of Computer Science, University of Regina. Regina, Saskatchewan, Kanada, November 2003. – Forschungsbericht

[Gioia 2011] Gioia, Ted: *The History of Jazz*. 2nd Edition. New York, USA : Oxford University Press, Inc., 2011

[Hal Leonard 2000] Hal Leonard: *The Real Book Play-Along*. Milwaukee, WI : Hal Leonard Corporation, 2000

[Haunschild 1997] Haunschild, Frank: *Die neue Harmonielehre. Ein musikalisches Arbeitsbuch für Klassik, Rock, Pop und Jazz*. Brühl : AMA Verlag, 1997

[Holland 1975] Holland, John H.: *Adaption in Natural and Artificial Systems*. Michigan, USA : University of Michigan Press, 1975

[Hoover u. Stanley 2007] Hoover, Amy K. ; Stanley, Kenneth O.: *NEAT Drummer: Interactive Evolutionary Computation for Drum Pattern Generation*. University of Central Florida, Orlando, 2007 `http://www.amalthea-reu.org/pubs/amalthea_tr_2007_03.pdf`. – abgerufen am 18.10.2015

[Hughes 2010] Hughes, Camden: *Learn Jazz Standards*. 2010 `http://www.learnjazzstandards.com/about/`. – abgerufen am 13.10.2015

[Koelle 2002] Koelle, David: *Music Programming for Java^{TM} and JVM Languages*. 2002 `http://www.jfugue.org/`. – abgerufen am 15.10.2015

[Koza 1992] Koza, John R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge : MIT Press, 1992

[Lewis 2000] Lewis, George E.: Too Many Notes: Computers, Complexity and Culture in *Voyager*. In: *Leonardo Music Journal* 10 (2000), S. S. 33–39

[Mayer 2001] Mayer, David G.: *Evolutionary Algorithms and Agricultural Systems*. Norwell, MA, USA : Kluwer Academic Publishers, 2001

[Mazzola 2006] Mazzola, Guerino: *Elemente der Musikinformatik*. Basel, Schweiz : Birkhäuser Verlag, 2006

[Miranda u. Biles 2007] Miranda, Eduardo R. ; Biles, John A.: *Evolutionary Computer Music*. London : Springer-Verlag, 2007

[MMA 1991] MMA, MIDI Manufacturers Association: *General MIDI 1, 2 and Lite Specifications*. 1991 `http://www.midi.org/techspecs/gm.php`. – abgerufen am 14.10.2015

[MMA 1995a] MMA, MIDI Manufacturers Association: *The Complete MIDI 1.0 Detailed Specification*. 1995 `http://www.midi.org/techspecs/midispec.php`. – abgerufen am 14.10.2015

[MMA 1995b] MMA, MIDI Manufacturers Association: *MIDI Messages*. 1995 `http://www.midi.org/techspecs/midimessages.php`. – abgerufen am 15.10.2015

[Oracle 1995] Oracle: *The Java^{TM} Tutorials: Sound*. 1995–2015 `http://docs.oracle.com/javase/tutorial/sound/`. – abgerufen am 15.10.2015

[Oracle 2015] Oracle: *Java^{TM} Platform, Standard Edition 8 API Specification*. 2015 `http://docs.oracle.com/javase/8/docs/api/`. – abgerufen am 15.10.2015

[Ray 1991] Ray, Thomas S.: An approach to the synthesis of life. In: *Artificial Life II (Santa Fe Institute Studies in the Sciences of Complexity Proceedings)* 11 (1991)

[Rechenberg 1965] Rechenberg, Ingo: *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment Library Translation 1122, 1965

[Roland 1981] Roland: *TR-808 – Operation Manual*. Hamamatsu, Shizuoka, Japan : Roland Corporation, 1981

[Sanchez u. a. 2012] Sanchez, Ernesto ; Squillero, Giovanni ; Tonda, Alberto: *Industrial Applications of Evolutionary Algorithms*. Berlin Heidelberg : Springer-Verlag, 2012

[Schwefel 1965] Schwefel, Hans-Paul: *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*, Technische Universität Berlin, Hermann Föttinger Institut für Strömungsmechanik, Diplomarbeit, 1965

[Sikora 2003] Sikora, Frank: *Neue Jazz-Harmonielehre*. 4. Auflage. Mainz : Schott Musik International, 2003

[Sorensen u. Brown 1998] Sorensen, Andrew ; Brown, Andrew: *jMusic - Music composition in Java*. 1998 `http://explodingart.com/jmusic/`. – abgerufen am 15.10.2015

[Stanley u. Miikkulainen 2002] Stanley, Kenneth O. ; Miikkulainen, Risto: Evolving Neural Networks through Augmenting Topologies. In: *Evolutionary Computation* 10 (2002), Nr. 2

[Todd u. Latham 1992] Todd, Stephen ; Latham, William: *Evolutionary Art and Computers*. Waltham, Massachusetts : Academic Press Inc., 1992

[Tokui u. Iba 2001] Tokui, Nao ; Iba, Hitoshi: *Music Composition with Interactive Evolutionary Computation*. Graduate School of Engineering, The University of Tokyo, 2001 `http://www.generativeart.com/on/cic/2000/ga2000-tokui.htm`. – abgerufen am 18.10.2015

[Unemi u. Nakada 2001] Unemi, Tatsuo ; Nakada, Eiichi: A Tool for Composing Short Music Pieces by Means of Breeding. In: *Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference*, 2001

[Yee-King 2007] Yee-King, Matthew J.: The Evolving Drum Machine. In: *MusicAL 2007 Proceedings*, `http://cmr.soc.plymouth.ac.uk/Musical2007/proceedings.htm`, 2007. – abgerufen am 19.10.2015

[Ziegenrücker 1977] Ziegenrücker, Wieland: *ABC Musik - Allgemeine Musiklehre - 446 Lehr- und Lernsätze*. 6., überarbeitete und erweiterte Auflage 2009. Leipzig : Deutscher Verlag für Musik, 1977

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 23. November 2015

Fabian Ostermann