Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik
Lehrstuhl für Künstliche Intelligenz

Konzeption und Realisierung eines benutzeradaptiven Systems musikalischer Agenten zur Komposition einfacher Melodien

Studienarbeit im Fach Informatik
vorgelegt von
Christoph Koutschan
geb. am 12. 12. 1978 in Dillingen an der Donau

Prüfer:	Prof. Dr. Herbert Stoyan
Betreuer:	Dipl.-Inf. Stefan Mandl
Beginn der Arbeit:	9. Januar 2004
Abgabe der Arbeit:	7. Oktober 2004

Kurzzusammenfassung

In der vorliegenden Arbeit wird ein Multiagentensystem zum Komponieren von Musik entworfen. Dieser Entwurf basiert auf einer Analyse des grundlegenden Aufbaus von Musikstücken. Das Agentensystem besteht aus musikalischen Agenten, die in der Lage sind, einstimmige Melodien zu kreieren. Dazu bedienen sie sich eines Expertensystems, das ebenfalls im Rahmen dieser Studienarbeit erstellt wird. Das Hauptaugenmerk wird auf die Lernfähigkeit der Agenten gerichtet, die aus den Bewertungen eines Benutzers lernen sollen, ihre Ergebnisse immer weiter dem Musikgeschmack des Benutzers anzunähern. Es werden verschiedene Lernverfahren konstruiert und gegeneinander abgewogen.

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den

Christoph Koutschan

Abstract

In this thesis a multiagent system for composing music is designed. The design is based on an analysis of basic construction principles of music. The agent system consists of musical agents which are able to create unisonous melodies. For that purpose the agents use an expert system that is also implemented in the context of this work. The main focus is on the agents' ability to learn from the user's rating to approximate his taste in music. Several methods of learning are developed and compared.

Inhaltsverzeichnis

Kapitel 1

Einführung

Musik ist die wahre allgemeine Menschensprache.
— K. J. Weber, Demokritos

Kaum waren die ersten Computer gebaut, schon gab es findige Programmierer, die den Rechner „zweckentfremdeten“, ihn also für nichtmathematische Anwendungen einsetzten. Neben automatischer Sprachübersetzung und diversen Spielen gehörte dazu auch die Komposition von Musik.

1.1 Der Computer lernt das Komponieren

Das erste namhafte Beispiel für eine computerbasierte Komposition ist die „Illiac Suite“ aus dem Jahr 1956, eine Suite für Streicherquartett, deren Partitur Lejaren Hiller und Leonard Isaacson von dem Großrechner ILLIAC an der University of Illinois berechnen ließen [7]. Dazu wurden in einem ersten Schritt zufällige Noten erzeugt und in einem zweiten Schritt dann getestet, ob diese die vorgegebenen Bedingungen (Kontrapunktregeln u. ä.) erfüllen. Die „Illiac Suite“ wurde im September 1956 an der University of Illinois uraufgeführt. Ein Jahr später näherte sich Brooks mit seinem probabilistischen Ansatz dem Thema von einer ganz anderen Richtung [2]: Markov-Ketten mit den Wahrscheinlichkeiten von Notensequenzen der Länge 2 bis 8 (extrahiert aus einer vorgegebenen Menge von Liedern) bildeten die Basis für die Komposition von neuen Musikstücken. Die Ergebnisse befriedigten jedoch nicht ganz, da bei kurzen Sequenzlängen (2 Noten) sehr zufällige und inakzeptable Melodien entstanden, bei langen Sequenzen (8 Noten) dagegen eine der Originalmelodien herauskam. Weitere Versuche folgten, so zum Beispiel ein Kompositionsprogramm für Zwölftonmusik [5] und zahlreiche Arbeiten über Generalbassaussetzung und Harmonieanalyse[1].

[1] Einen guten Überblick gewinnt der Leser in [15].

1.2 Was ist ein Agent?

Obwohl der Begriff und das Konzept des Agenten schon in den 80er Jahren aufkam und seitdem im Rahmen der verteilten künstlichen Intelligenz in zahlreichen Publikationen untersucht wurde, hat sich noch keine einheitliche Definition dafür eingebürgert. Nach Michael Wooldrigde [19] ist ein Agent ein Computerprogramm, welches in der Lage ist, innerhalb einer Umgebung autonom zu agieren, um die ihm gesetzten Ziele zu erreichen: „there is a general consensus that autonomy is central to the notion of agency“.
Zusätzlich zum Begriff der Autonomie werden oft noch die Eigenschaften Kooperation und Mobilität genannt, die einen Software-Agenten auszeichnen. Hans-Dieter Burkhard [6, Kap. 24] steuert zu diesem Thema folgende Definition bei: „Ein Software-Agent ist ein längerfristig arbeitendes Programm, dessen Arbeit als eigenständiges Erledigen von Aufträgen oder Verfolgen von Zielen in Interaktion mit einer Umwelt beschrieben werden kann“. Wie eben schon liegt auch hier das Hauptaugenmerk auf der Autonomie bzw. der Eigenständigkeit. Zusätzlich kommt noch die Idee hinzu, dass es sich bei einem Agenten um ein „längerfristig arbeitendes Programm“ handelt. Welche dieser Aspekte auf die musikalischen Agenten zutreffen, wird im folgenden Abschnitt dargelegt.

1.3 Musikalische Agenten

Ein „musikalischer Agent“ ist ein Agent, der musikalisches Wissen besitzt, welches ihn dazu befähigt, spezielle Aufgaben innerhalb eines Kompositionsprozesses wahrzunehmen. Er erfüllt die Definition eines Agenten insofern, als er autonom agiert, das heißt, sich um Kompositionsaufträge bewirbt, solche annimmt und ausführt. Er kann durch den Austausch von Nachrichten mit anderen musikalischen Agenten kommunizieren und zusammenarbeiten. Eine Komposition entsteht immer aus der Zusammenarbeit mehrerer Agenten (Kooperation). Ferner sind die musikalischen Agenten in der Lage, sich durch Kreuzung zu reproduzieren, ähnlich wie es im Bereich der genetischen Programmierung geschieht.
Das Wissen eines musikalischen Agenten besteht aus Regeln, die er zum Erstellen einer Komposition benötigt und die vom Programmierer vorab formuliert wurden. Diese Regeln beschreiben fundamentale musikalische Sachverhalte und sind Bestandteil einer Wissensbasis, welche die Grundlage für ein regelbasiertes Expertensystem ist.
Es gibt verschiedene Typen von musikalischen Agenten, die jeweils auf ein bestimmtes Teilgebiet des Kompositionsprozesses spezialisiert sind. So kann ein Agent die Grobstruktur für ein längeres Musikstück entwerfen, während ein anderer kurze Melodiefragmente erfindet oder eine gegebene Tonfolge nach einem bestimmten Muster variiert.

1.4 Ziel der Studienarbeit

In dieser Studienarbeit wird ein System lernender Agenten für die Komposition von Musik entworfen und implementiert. Im folgenden wird experimentell untersucht, inwiefern dieser Ansatz geeignet ist, den Musikgeschmack eines einzelnen Benutzers zu erlernen und nachzubilden. Denn es ist offensichtlich nicht möglich, eine „Weltformel“ zu finden, mit der man ausrechnen könnte, wie gut ein Musikstück ist, die also eine objektive qualitative Beurteilung erlauben würde. Zu verschieden sind die Geschmäcker schon innerhalb eines Kulturkreises, und erst recht, wenn man die Musikstile anderer Kulturen in die Betrachtung einbezieht. Die Beurteilung einer Komposition kann daher nur von einem Menschen vorgenommen werden und ist somit immer subjektiv. Dies kann natürlich dazu führen, dass ein und dasselbe Musikstück von zwei verschiedenen Personen völlig unterschiedlich bewertet wird – so wie es in der Realität oft genug vorkommt. Es geht hier also weniger darum, allgemeingültige Regeln für gute Melodien zu finden, als vielmehr einen Anwender zufrieden zu stellen. Dass dieses Resultat einem anderen Benutzer möglicherweise nicht gefällt, liegt in der Natur der Sache. Diverse Lern- und Bewertungsverfahren werden verglichen und in Bezug auf ihre Praxistauglichkeit evaluiert. Um den Rahmen dieser Arbeit nicht zu sprengen, werde ich mich auf die Komposition von einfachen einstimmigen Musikstücken (Melodien) beschränken.

1.5 Gliederung der Studienarbeit

In Kapitel 2 wird der Entwurf einer geeigneten Architektur vorgestellt. Dabei ist auch das Verhalten der einzelnen Agenten und deren Zusammenspiel zu spezifizieren; dies wird in Kapitel 2.4 geschehen. Der praktische Teil der Studienarbeit besteht darin, ein Multiagentensystem bestehend aus musikalischen Agenten zu implementieren, mit dem die vorangegangenen theoretischen Erörterungen in die Praxis umgesetzt und getestet werden sollen. Einen Überblick über die Implementierung gibt Kapitel 3. Über die verschiedenen getesteten Lernverfahren und deren Resultate wird in Kapitel 4 berichtet. Die Arbeit schließt mit einem Ausblick.

Kapitel 2

Konzeption des Agentensystems

Any attempts to simulate the compositional abilities of humans will probably not succeed until in fact the musical models and plans that humans use are described and modeled.

— Andy Moorer

In diesem Kapitel wird eine kurze Einleitung in die Theorie der Multiagentensysteme gegeben, bevor – ausgehend von den Grundprinzipien einer Melodie – der Entwurf für ein Agentensystem mit musikalischen Agenten hergeleitet wird.

2.1 Multiagentensysteme

Multiagentensysteme haben im Allgemeinen folgende Vorteile, die für ihren Einsatz sprechen: Abhängig von der Problemgröße können bei Bedarf zusätzliche Agenten ins System eingespeist werden, um an der Lösung mitzuwirken. Je länger zum Beispiel die gewünschte Komposition sein soll, desto mehr Agenten können gleichzeitig an mehreren Abschnitten mitwirken, und – bei verteiltem Einsatz auf mehreren Rechnern – parallel an der Aufgabe arbeiten. Durch Redundanz – es stehen mehr Agenten zur Verfügung als notwendig – erlangt das Multiagentensystem eine hohe Zuverlässigkeit, da beim Ausfall eines Agenten ein anderer dessen Rolle übernehmen kann.
Einen weiteren Vorteil stellt die Wiederverwendbarkeit der Agenten dar, die von einem Benutzer konfiguriert und trainiert werden, und von einem anderen Benutzer in einem anderen Multiagentensystem wieder eingesetzt werden können. Einem Multiagentensystem liegt das Prinzip des „divide and conquer“ zu Grunde: Einerseits wird das komplexe Problem des Komponierens aufgespaltet in Unterprobleme, die von jeweils einem Agenten bearbeitet werden. Andererseits kann die Implementierung aufgeteilt werden, indem jeder Agent von einem anderen Experten programmiert werden kann.

Die folgende Zusammenstellung nennt einige Charakteristika von Multiagentensystemen, wie sie von Jennings/Sycara/Wooldridge [19] genannt werden, und in welcher Form diese hier auftreten:

- Jeder Agent besitzt nur unvollständige Informationen über den Gesamtzustand des Systems.
- Jeder Agent ist eingeschränkt in seinen Fähigkeiten: Jeder musikalische Agent spezialisiert sich auf eine bestimmte Aufgabe.
- Verteilte Systemkontrolle: Dies geschieht durch die Weitergabe und Aufteilung eines Kompositionsauftrags.
- Dezentrale Datenhaltung: Jeder musikalische Agent hat seine eigene Wissensbasis.
- Asynchrone Berechnung

2.2 Melodie

Was genau bezeichnet der Begriff Melodie? Der Duden definiert sie als „sangbare, in sich geschlossene Folge von Tönen“, wobei interessant ist, dass sofort der Bezug zur menschlichen Stimme, dem ursprünglichsten aller Musikinstrumente, hergestellt wird. Nun stellt sich die Frage, ob diese Einschränkung gerechtfertigt ist. Eine Einschränkung ist die Sangbarkeit insofern, als die menschliche Stimme nur einen beschränkten Tonumfang besitzt – in der Regel sind das zwei bis drei Oktaven –, und damit Tonfolgen, die diesen Rahmen sprengen, nicht mehr als Melodie bezeichnet werden könnten. Eine allgemeinere und ausführlichere Definition bietet der Brockhaus: „Folge von Tönen verschiedener Höhe oder eine Folge verschieden großer und verschieden gerichteter Intervalle, die als Einheit aufgefasst wird. Eine Melodie wird daher zunächst bestimmt durch die Tonhöhenorganisation (Diastematik) und erst dann durch die Dauer (und die Betonung) der einzelnen Töne (Rhythmus) sowie die Gliederung (Periodik), das Zeitmaß (Tempo) und durch die Art der Ausführung (Tongebung, Klangfarbe, Dynamik). Die Melodie kann aus symmetrisch angeordneten Gliedern bestehen, wiederkehrende Motive aufweisen, als Periode in Vorder- und Nachsatz gegliedert sein oder sich aus zwei oder mehreren Perioden zusammensetzen, aber auch ungleichmäßig gegliedert sein, verschränkt oder unterbrochen werden“.
In dieser Arbeit wird eine Melodie vollständig charakterisiert durch die Diastematik, den Rhythmus und die Periodik. Sie sind die wesentlichsten Merkmale für die Bestimmung und die Wiedererkennung einer Melodie. Das Tempo und die Aspekte, die unter „Art der Ausführung“ genannt werden, werden dagegen ausgeklammert. Sie spielen eine eher untergeordnete Rolle für die Charakterisierung einer Melodie: Eine bekannte Melodie wird der Hörer auch wiedererkennen, wenn das Tempo (in moderatem Rahmen) variiert oder sie auf einem anderen Instrument gespielt wird.

Einen falschen Ton dagegen wird der Hörer sofort bemerken und die so entstandene Melodie nicht mehr als die ursprüngliche gelten lassen.

Wie ist eine Melodie aufgebaut und was macht eine bestimmte Tonfolge zu einer guten Melodie? So gut wie alle Melodien lassen sich – oben mit dem Stichwort Periodik angedeutet – in einzelne Teile gliedern (Takte, Verszeilen). Sehr häufig treten dabei Wiederholungen oder Variationen einzelner Teile auf. Das sollte beim Architekturentwurf berücksichtigt werden. Bei der Frage nach einer guten Tonfolge zählt vorrangig das Urteil des Benutzers, welches in seiner Subjektivität zunächst unanfechtbar ist. Einige Kriterien lassen sich jedoch angeben, die eine Tonfolge erfüllen sollte, um von der Mehrzahl der Hörer (jedenfalls sofern es sich um Angehörige des westlichen Kulturkreises handelt) als gut empfunden zu werden. Dazu gehört zum Beispiel, dass eine Melodie auf dem Grundton endet. Derartige Sachverhalte sollten auf jeden Fall in der Wissensbasis eines musikalischen Agenten enthalten sein.

2.3 Multiagentensystem mit musikalischen Agenten

Wie soll nun die Struktur eines Multiagentensystems zum Komponieren von Melodien aussehen? In seinem Artikel „A Technique for the Composition of Music in a Computer“ schreibt Stanley Gill [5], wie der Kompositionsprozess idealerweise ablaufen sollte: „The best procedure would be first to select the thematical material, then a skeleton plan of the work, and then to fill in more and more detail until ultimately the whole composition is expressed in terms of individual notes, rather in the way that a human composer might proceed.“. Das Komponieren lässt sich demnach als mehrstufiger, hierarchischer Prozess auffassen, bei dem zuerst eine Grobstruktur entworfen wird, Motive und Rhythmus kreiert, und dann erst die einzelnen Noten gesetzt werden. Als Architekturmodell bietet sich daher eine hierarchische Schichtenarchitektur an. In der vorliegenden Arbeit wurden drei Schichten gewählt, da die Anforderung, eine kurze einstimmige Melodie zu erfinden, noch relativ einfach ist. Für umfangreichere, insbesondere mehrstimmige Kompositionen wäre eine Erweiterung um zusätzliche Schichten sinnvoll, um beispielsweise die Frage nach dem Harmonieverlauf in einer eigenen Schicht abzuhandeln. Bei einstimmigen Melodien tritt diese Frage naturgemäß nicht auf.

Es ergibt sich das in Abbildung 2.1 dargestellte Architekturschema[1]. Wie der Leser dort sehen kann, erfolgt die Kommunikation nach dem Prinzip des two-pass-control: Der Input wird von der obersten Schicht bis in die unterste durchgereicht; von dort werden die Ergebnisse sukzessive wieder zurückgegeben, bis die oberste Schicht daraus den Output generiert.

[1] Die Agentenbezeichnungen werden in Kapitel 2.4 erklärt.

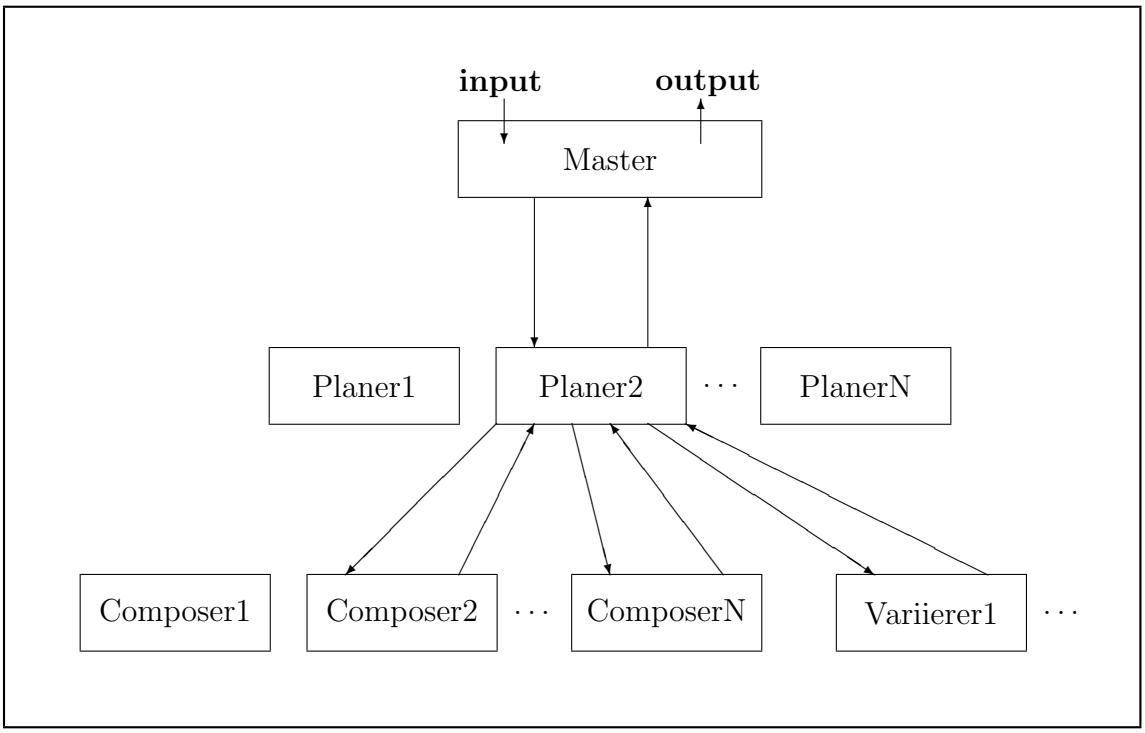


Abbildung 2.1: Schichtenarchitektur

Der Kommunikation während des Kompositionsprozesses liegt das Kontraktnetzmodell zu Grunde. Die Funktionsweise möchte ich beispielhaft anhand des Nachrichtenaustauschs zwischen einem Planer und Composern darstellen. Das Kontraktnetzmodell sieht vier Schritte vor [19]:

1. Der Planer gibt die Ausschreibung eines Auftrags bekannt, indem er ihn an alle Composer sendet.

2. Die Composer prüfen den Auftrag, evaluieren ihre Fähigkeit, diesen Auftrag zu bearbeiten und bewerben sich beim Planer um die Komposition.

3. Der Planer wählt aus allen Bewerbungen die beste aus.

4. Der Planer beauftragt den ausgewählten Composer mit der Komposition; das heißt, der Kontrakt zwischen dem Planer und dem Composer wird geschlossen. Der Composer beginnt mit dem Komponieren und gibt das Ergebnis zurück.

Die Kommunikation zwischen dem Master und den Planern erfolgt analog.

2.4 Agententypen

Wie in Abbildung 2.1 zu erkennen ist, gibt es in jeder Schicht eine oder zwei spezielle Sorten von Agenten. Es ist nun näher zu erläutern, wie sich die jeweiligen Agententypen verhalten und welche Aufgaben sie wahrnehmen.

2.4.1 Master

Auf der obersten Hierarchieebene steht der Master-Agent. In einem musikalischen Multiagentensystem gibt es genau einen Agenten dieses Typs. Der Master verfügt über eine grafische Benutzerschnittstelle (GUI = **G**raphical **U**ser **I**nterface) und ist die einzige Interaktionsmöglichkeit für den Anwender. Über ihn kann der Benutzer einen Kompositionsvorgang starten: Der Master wählt gemäß dem Kontraktnetzmodell einen Agenten der nächsten Hierarchiestufe (Planer), vergibt einen Auftrag an diesen und wartet, bis er das fertige Resultat zurückerhält. Um die Komposition bewerten zu können, muss der Anwender natürlich die Gelegenheit bekommen, sich diese anzuhören. Deshalb kann der Master das Musikstück über den systemeigenen MIDI-Synthesizer abspielen. Anschließend darf der Benutzer die Komposition bewerten und bei Bedarf speichern. Die Bewertung gibt der Master an die beteiligten Agenten weiter.

2.4.2 Planer

Auf der zweiten Hierarchieebene stehen die Planer. Wie der Name schon andeutet, plant dieser Agententyp Aufbau und Struktur der Komposition. Er legt die Gesamtlänge fest und zerlegt das Stück in Einzelteile, deren Durchführung er an die Agenten der untersten Schicht (Composer und Variierer) delegiert. Dabei muss er genau darauf achten, dass die Übergänge zwischen den einzelnen Teilen passen (z. B. durch Vorgabe der Anfangs- und Endnote). Eine wesentliche Aufgabe besteht auch darin, der Komposition eine gewisse Homogenität zu verleihen. Einige Verfahrensweisen, um dieses Ziel zu erreichen, sind die Vorgabe eines bestimmten Rhythmus' oder die Variation eines schon fertigen Melodieteils durch einen anderen Agenten.

2.4.3 Composer

Ein Composer hat die Aufgabe, ein kurzes Melodiestück vorgegebener Länge (etwa zwei bis drei Takte) zu erfinden. Um ein möglichst gutes Ergebnis zu erzielen, darf das Setzen der Noten natürlich nicht nur nach dem Zufallsprinzip erfolgen. Deshalb hat jeder Composer einen Satz von Regeln, nach denen er die Noten erzeugt. Die Reihenfolge, in der diese Regeln angewendet werden, wird dabei vom Zufall beeinflusst: Wenn zwei Regeln dieselbe Priorität[2] besitzen oder eine Regel auf mehr als eine Weise (mit unterschiedlichen Prämissen[3]) angewendet werden kann, dann wird zufällig entschieden. Schließlich soll nicht in jeder Komposition die gleiche Tonfolge erzeugt werden. Ferner wird das Komponieren von den Vorgaben gesteuert, die der Planer für das Melodiestück gemacht hat, und die der Composer einzuhalten versucht.

[2] s. Kapitel 3.3.3
[3] s. Kapitel 3.3

2.4.4 Variierer

Ein Variierer hat die Aufgabe, einen vorgegebenen Melodieteil zu variieren. Dies kann zum Beispiel eine Transposition oder Umkehrung[4] sein. Jeder Variierer ist dabei auf genau eine Variationsmöglichkeit spezialisiert.

2.4.5 Hilfsagenten

Diese Agenten komponieren nicht, sie führen lediglich Hilfsfunktionen aus. Der Player gibt die Kompositionen über den Lautsprecher aus, der Selektierer ist für die Kontrolle der Agentenpopulation zuständig: Er kann Agenten erzeugen, kreuzen und wieder entfernen.

[4] Ausführliche Erläuterung in Kapitel 3.7

Kapitel 3

Implementierung

Ohne Musik wäre das Leben ein Irrtum.
— Friedrich Nietzsche

Nachdem im vorigen Kapitel der grobe Rahmen des Agentensystems abgesteckt wurde, soll jetzt ein genauer Blick auf die Realisierung geworfen werden. Manchem Leser, der sich mit einer vagen Vorstellung von der Funktionsweise des Agentensystems begnügt, mögen diese Ausführungen zu detailliert erscheinen. Ihm wird empfohlen, dieses Kapitel lediglich zu überfliegen oder ganz auszulassen, um mit Kapitel 4 fortzufahren, in dem die Ergebnisse vorgestellt werden.

3.1 Agentenplattform JADE

Die Agentenplattform JADE (**J**ava **A**gent **DE**velopment Framework) wurde für die Implementierung der musikalischen Agenten herangezogen. Sie wurde entwickelt am Tilab (Telecom Italia Lab), dem früheren CSELT (Centro Studi E Laboratori Telecomunicazioni)[1] und der Computer Engineering Group an der Universität Parma. JADE erleichtert die Entwicklung eines Multiagentensystems insofern, als Standardprozeduren – wie das Versenden von Nachrichten oder ein „yellow-page service“ – zur Verfügung gestellt werden. All dies geschieht in Übereinstimmung mit den Spezifikationen der FIPA (Foundation for Intelligent Physical Agents). Dazu gehört auch die Kommunikation zwischen den Agenten, die mittels ACL-Nachrichten (**A**gent **C**ommunication **L**anguage) geschieht.
So wie JADE komplett in Java implementiert ist, werden auch die Agenten in dieser Sprache programmiert. Dazu werden für jeden Agenten sogenannte Verhaltensweisen (*behaviours*) definiert, die durch bestimmte Ereignisse, z. B. den Empfang einer Nachricht von einem anderen Agenten oder einer Benutzereingabe in der GUI, aktiviert werden. Ein musikalischer Agent hat jeweils nur ein *behaviour*, welches einen Nachrichtenhandler darstellt. Es wurden nicht für jeden einzelnen Agenten eigene *behaviours* programmiert, sondern nur für jeden Agententyp. Die unterschiedlichen

[1] http://jade.cselt.it/

Eigenschaften und Fähigkeiten der Agenten gleichen Typs resultieren allein aus ihrem Wissen.
Wird ein neuer Agent gestartet, so registriert sich dieser über den `DFService`, die gelben Seiten der Agentenplattform. Dabei gibt er diejenigen Dienste an, die er anbietet. Im Fall der musikalischen Agenten ist dies einfach die Tätigkeit, die dem Typ eines Agenten entspricht („plan“, „compose“, „vary“). Über die Methode `DFService.search` kann ein Agent gezielt nach anderen Agenten suchen, die einen bestimmten Dienst anbieten. Zum Beispiel benötigt ein Planer die Namen aller verfügbaren Composer, um einen Auftrag zur Bewerbung auszuschreiben.

3.2 Kommunikation

Jede ACL-Nachricht, die ein Agent einem anderen schickt, stellt gemäß der Sprechakttheorie von Austin eine performative Äußerung dar; so werden sprachliche Akte bezeichnet, mit denen eine bestimmte Handlung ausgeführt wird: Fragen, Informieren, usw.[2] In der FIPA-Spezifikation wird einer ACL-Nachricht daher ein Attribut `performative` zugeordnet, welches angibt, welcher Art die Nachricht ist. Einige nach dem FIPA-Standard gültige Werte für `performative` werden von den musikalischen Agenten verwendet, und zwar in der Weise wie in Tabelle 3.1 angegeben ist.

`INFORM`	Mitteilungen (z. B. Bewertungen)
`QUERY_IF`	Ausschreibung eines Auftrags
`AGREE`	Bewerbung um einen Auftrag
`REQUEST`	Auftrag
`PROPOSE`	Rückgabe eines bearbeiteten Auftrags
`CANCEL`	Ende eines Kompositionsvorgangs

Tabelle 3.1: Verwendung des Attributs `performative`

Abbildung 3.1 zeigt beispielhaft den Einsatz dieser Nachrichtenattribute: Ein Planer erhält mittels `REQUEST` einen Auftrag. Die Ausschreibung an die Composer erfolgt mit `QUERY_IF`. In diesem Beispiel signalisieren beide ihre Bereitschaft, diesen Auftrag auszuführen (`AGREE`), und teilen dem Planer mit, wie sie ihre Eignung für diesen Auftrag einschätzen. Aufgrunddessen wählt dieser nun den Composer1 aus und betraut ihn mit der Ausführung des Auftrags. Das Ergebnis wird dann über alle Hierarchieebenen als Vorschlag (`PROPOSE`) an den Master zurückgegeben, wobei der Planer im Allgemeinen die einzelnen Melodieteile noch zusammenfügen muss (in diesem Beispiel ist das aber nur einer). Mit `INFORM` werden die Agenten über die Bewertung des Benutzers informiert und mit `CANCEL` dazu veranlasst, wieder in ihren Anfangszustand zu gehen, um für die nächste Komposition bereit zu sein.

[2] Definition nach [8]

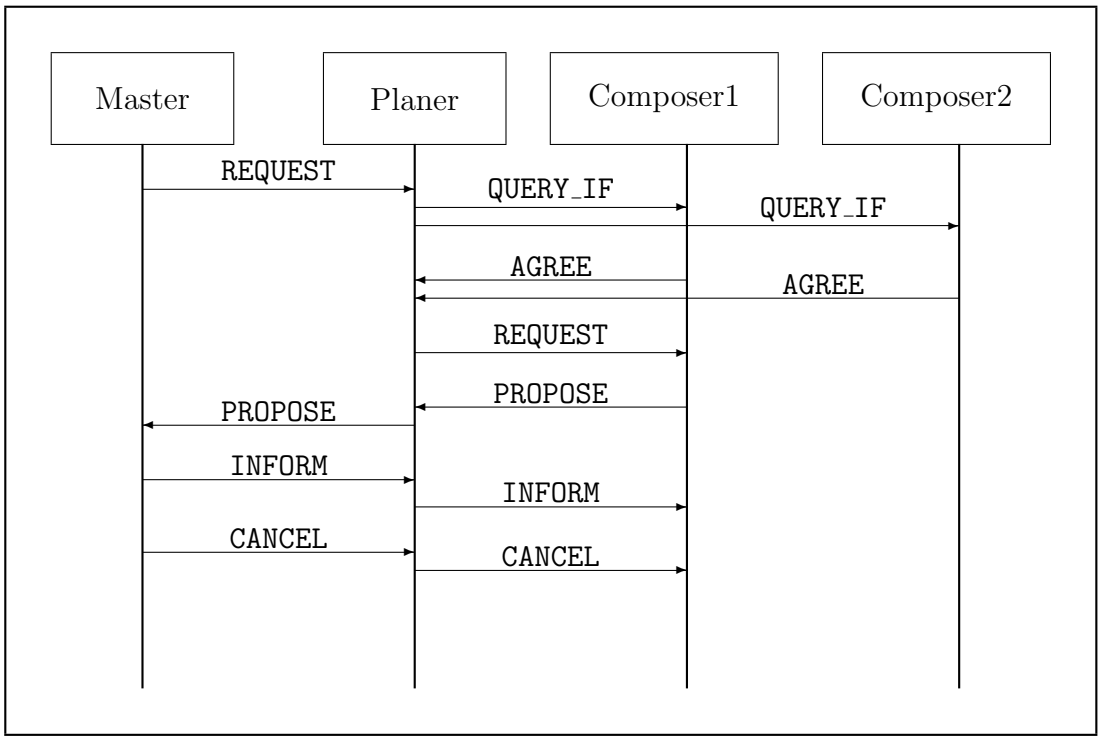


Abbildung 3.1: Kommunikationsprotokoll (Beispiel)

3.3 Expertensystem

Zum Komponieren verwenden die musikalischen Agenten ein regelbasiertes Expertensystem. In einem solchen Expertensystem ist das Wissen in Form von Regeln (der „am weitesten verbreiteten Wissensrepräsentation in Expertensystemen“ [6]) gespeichert, die die Struktur „Wenn X dann Y“ haben: Wenn die Prämisse(n) X erfüllt sind, dann wird die Implikation Y ausgeführt. Treffen die Prämissen mehrerer Regeln gleichzeitig zu, spricht man von einem Konflikt. Eine Konfliktlösungsstrategie entscheidet dann, welche der konkurrierenden Regeln zuerst ausgeführt werden soll. Regeln können nach Belieben hinzugefügt oder gelöscht werden. Eine ausführlichere Einführung in das Gebiet der Expertensysteme findet man in [13].

Die Flexibilität, die ein solches System bietet, besonders was die Erweiterung und Veränderung des Wissens anbelangt, ist für unsere Problemstellung nahezu unverzichtbar. So lässt sich das angestrebte Lernen der Agenten einerseits durch eine Veränderung der Regelmenge realisieren, andererseits mit Hilfe einer geeigneten Konfliktlösungsstrategie. Näheres dazu erfährt der Leser in Kapitel 4. Von großem Vorteil ist auch die Möglichkeit des Einfügens von Zusatzwissen im Nachhinein, da sich musikalisches Wissen, wie eine gute Melodie zu komponieren ist, kaum in einem engeren Rahmen eingrenzen lässt. So kann bei der Implementierung die Menge der Regeln sukzessive vergrößert werden, bis die Resultate den Anforderungen entsprechen.

3.3.1 JESS

Es stellt sich nun die Frage nach einem geeigneten Werkzeug, mit dem das Expertensystem realisiert werden kann. Dabei ist eine möglichst einfache Integration in die Java-Umgebung der Agentenplattform wünschenswert. Gewählt wurde daher JESS (**J**ava **E**xpert **S**ystem **S**hell), welches von Ernest Friedman-Hill an den Sandia National Laboratories in Livermore, California entwickelt wurde[3]. JESS ist ein in Java programmierter Regelinterpretierer, was den Vorteil bietet, dass es einfach in andere Java-Applikationen eingebunden werden kann und dass in JESS-Programmen sämtliche Java-Klassen und -Methoden verwendet werden können. So ist es zum Beispiel möglich, die selbstdefinierten Klassen `Melody` und `Rational` (Notenlängen und -positionen erfordern die Darstellung rationaler Zahlen) in JESS-Regeln und JESS-Funktionen zu verwenden. Die Syntax von JESS orientiert sich an LISP (z. B. Klammerausdrücke, Präfixnotation, usw.). Zur Auswertung der Regeln verwendet JESS den Rete-Algorithmus[4]. JESS bietet für die Abarbeitung der Regeln sowohl die Vorwärts- (Forward-) als auch die Rückwärtsverkettung (Backward-Reasoning) an. Im Fall des Komponierens soll die erste Alternative zum Einsatz kommen, bei der so lange Regeln ausgeführt werden, bis keine mehr anwendbar ist. Das ist dann der Fall, wenn die Melodie vollendet ist. Bei der Rückwärtsverkettung dagegen müsste beim Ziel (der fertigen Melodie) begonnen werden, was in der vorliegenden Situation unpraktikabel ist.
Als Nachteile von JESS sei hier erstens die teilweise recht komplizierte Syntax genannt, die zu unübersichtlichen und umständlichen Regeldefinitionen führt. Zweitens erschweren einige Einschränkungen, die JESS dem Programmierer beim Erstellen von Regeln auferlegt, die Arbeit. So ist es zum Beispiel nicht möglich, im Prämissenteil einer Regel einen arithmetischen Ausdruck als Belegung für einen Faktenslot zu verwenden, sondern es muss eine neue Variable eingeführt werden, die mit dem Ergebnis dieses arithmetischen Ausdrucks auf Gleichheit überprüft wird. Nicht zuletzt erfordert das Erstellen eines „Netzes“ für den Rete-Algorithmus erhebliche Rechenzeit, was zur Folge hat, dass das Erzeugen eines neuen Agenten mit einer gewissen Wartezeit für den Benutzer verbunden ist. Im Testfall betrug diese Wartezeit bis zu mehreren Sekunden. Ist dieses Netz allerdings erst einmal fertiggestellt, so geht die Berechnung (die Komposition einer neuen Melodie) relativ schnell. Hier kann der Rete-Algorithmus seine volle Stärke entfalten.

3.3.2 Wissensbasis

Jeder Planer und jeder Composer hat seine eigene Menge von Regeln, die er zum Komponieren benötigt. Zusammen mit den Fakten, die die Vorgaben und Parameter eines Auftrags angeben, bilden sie die Wissensbasis des Agenten. Aus dieser kann der Regelinterpretierer ein Ergebnis (Melodiegerüst oder Melodieteil) herleiten. Alle Vorgaben, die ein Agent bei einem Auftrag erhält, sind daher in der JESS-Syntax notiert.

[3] http://herzberg.ca.sandia.gov/jess/
[4] Die Funktionsweise des Rete-Algorithmus wird in [18] beschrieben.

3.3.3 Konfliktlösungsstrategie

JESS bietet die Möglichkeit, jeder Regel eine Priorität (so genannte `salience`) zuzuordnen. Im Konfliktfall – wenn die Vorbedingungen mehrerer Regeln erfüllt sind – wird dann diejenige mit der höheren Priorität zuerst ausgeführt. Dies wurde dazu genutzt, um das Lernen zu realisieren: Regeln, die sich bewähren, erhalten eine höhere Priorität und werden in Zukunft bevorzugt ausgeführt. Da es aber einerseits nicht möglich ist, die Prioritäten von Regeln dynamisch zu ändern und es andererseits ein zu großer Performanceverlust wäre, würde man nach jedem Lernschritt den Programmtext des JESS-Programms anpassen und eine neue JESS-Engine erstellen, hat sich ein anderes Konzept bewährt: Ein großer Vorteil von JESS ist nämlich, dass der Programmierer eigene Konfliktlösungsstrategien definieren kann. Die Bevorzugung einer als gut bewerteten Regel geschieht somit nicht durch Änderung ihrer Priorität, sondern dadurch, dass sie mit Hilfe der selbstdefinierten Konfliktlösungsstrategie an die erste Stelle der Ausführungsreihenfolge gesetzt wird.

3.4 Master

Beim Starten der Agentenplattform wartet der Master, bis sich alle anderen Agenten bei ihm angemeldet haben. Dazu schicken sie ihm eine `INFORM`-Nachricht mit dem Inhalt „Ready“. Dann erst öffnet der Master sein Fenster, so dass der Benutzer keine Komposition in Auftrag geben kann, bevor nicht alle Agenten bereit sind.

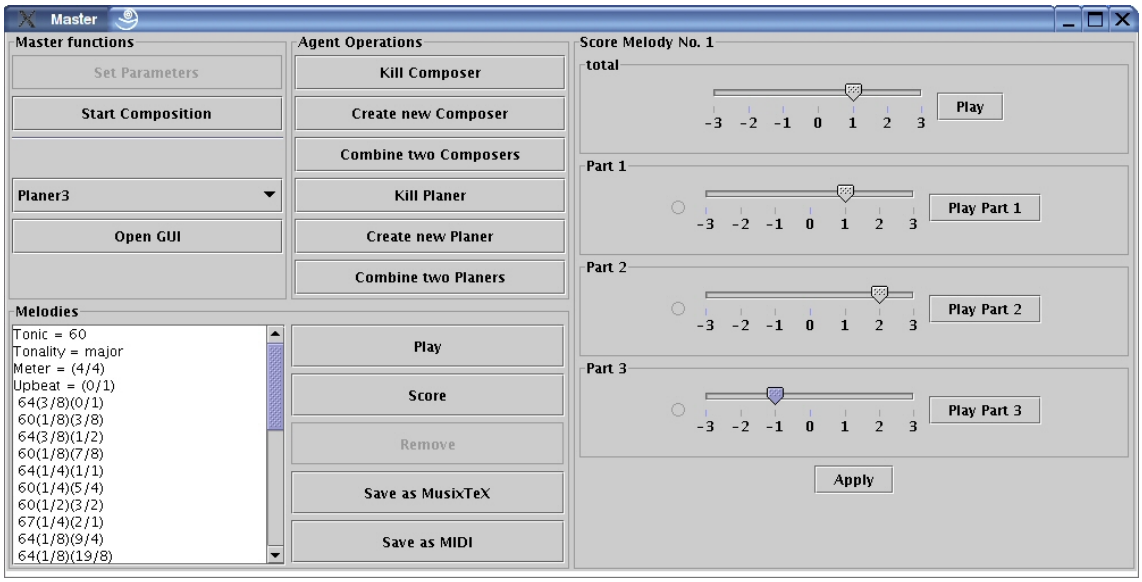


Abbildung 3.2: GUI des Master-Agenten

3.4.1 Benutzeroberfläche

Der Master ist der einzige, der beim Starten automatisch ein Fenster anzeigt (Abbildung 3.2). Dieses ist in mehrere Bereiche gegliedert. Oben links befindet sich das

Hauptmenü mit dem Befehl „Start Composition“ (Komposition in Auftrag geben) und einem Auswahlmenü, in dem der Benutzer eine Liste aller Agenten einsehen und bei Bedarf deren GUIs öffnen kann. Rechts daneben sind Schaltflächen angeordnet, die zum Steuern der Agentenpopulation dienen: So können Agenten neu erzeugt, miteinander gekreuzt und entfernt werden. Näheres dazu findet sich in Kapitel 3.8.1.
Im unteren linken Teil werden die komponierten Melodien angezeigt. In dem Textfenster erscheint die Melodie in einer textuellen Übersetzung, bei der die Tonhöhe als Integerwert und die Notendauer als Bruch angegeben werden. Mit den Buttons rechts daneben kann die Melodie abgespielt und in verschiedenen Dateiformaten (MIDI und MusiXTEX, einer Erweiterung von TEX für professionellen Notensatz) gespeichert werden. Beim Betätigen der Schaltfläche „Score“ öffnet sich in der rechten Hälfte der GUI ein Menü, mit dem der Benutzer die Bewertung der komponierten Melodie vornehmen kann.
Die Bewertung einer Melodie geschieht nach den folgenden Kriterien: Einerseits kann die Melodie als Ganzes bewertet werden; dabei sollten Aspekte wie die Gesamtkonzeption (Anzahl der Teile, Wiederholungen, etc.), die Takt- und Tonart und der Rhythmus eingehen. Andererseits kann jeder Teil der Melodie für sich bewertet werden. Für jede dieser Optionen steht ein Schieber bereit, der im Wertebereich von -3 (sehr schlecht) bis 3 (sehr gut) eingestellt werden kann. Mit dem Button „Apply“ wird die gewählte Bewertung bestätigt und wirksam.

3.4.2 Kompositionsauftrag

Erteilt der Benutzer mit „Start Composition“ einen Kompositionsauftrag, so muss der Master einen Planer aussuchen, dem er diesen Auftrag weitergibt. Zu diesem Zweck verschickt er eine Anfrage an alle Planer, die darauf mit einer Bewerbung antworten. Nun könnte der Master einfach denjenigen Planer P_i, der die beste Bewerbung eingereicht hat (d.h. derjenige mit dem höchsten Punktestand $P(P_i)$), auswählen; dies würde jedoch schnell dazu führen, dass – bei guter Bewertung – immer nur noch derselbe Planer zum Zuge käme. Andere Agenten, die möglicherweise interessante Alternativen anbieten, blieben außen vor. Darum fließt in den Vergleichswert $G(P_i)$ noch eine Zufallskomponente ein, die dafür sorgt, dass auch die Planer mit der zweit- und drittbesten Bewertung eingesetzt werden – allerdings mit geringerer Wahrscheinlichkeit. Die Gesamtbewertung eines Planers berechnet der Master nach folgender Formel:

$$G(P_i) = \frac{P(P_i)}{\max\limits_{1 \le j \le n} \{|P(P_j)|\}} + r,\ 0 \le r < 1,$$

wobei r eine mit `Math.random()` erzeugte Zufallszahl ist.

3.4.3 Bewertung

Bevor der Benutzer eine Melodie bewerten kann, muss er sie sich anhören können. Das Abspielen über den MIDI-Synthesizer nimmt der Master aber nicht selbst vor,

sondern beauftragt einen speziellen Agenten, den Player, damit (s. Kapitel 3.8.2). Hat der Benutzer eine Melodie gehört und bewertet, so übermittelt der Master eine `INFORM`-Nachricht mit den erteilten Punkten an den Planer, der die bewertete Melodie ausgeführt hat. Dieser kümmert sich darum, dass die beteiligten Composer ebenfalls benachrichtigt werden.

3.5 Planer

Der Planer bekommt vom Master den Auftrag, eine Melodie zu komponieren. Daraufhin muss er den Aufbau der Melodie planen, insbesondere allgemeine Parameter wie Ton- und Taktart festlegen, sowie die Anzahl der einzelnen Teile. Die Vorgaben für die Teile werden dann zur Ausführung an die Composer und Variierer weitergeschickt. Im Wesentlichen besteht der Planer aus einem Nachrichtenhandler und funktioniert im Detail wie folgt.

3.5.1 Bewerbung

Erhält der Planer eine `QUERY_IF`-Message vom Master, so muss er ein Angebot erstellen, mit dem er sich um den Auftrag bewirbt. In dem Angebot teilt er dem Master seinen persönlichen Punktestand mit, der sich aus allen früheren Bewertungen dieses Planers berechnet. Der Punktestand ist also ein Maß dafür, wie gut dieser Planer die vorigen Aufträge, die er erhalten hat, ausgeführt hat. Sein Angebot schickt er in einer `PROPOSE`-Nachricht an den Master zurück.

3.5.2 Auftragsbearbeitung

Wenn sich der Master für einen Planer entschieden hat, sendet er diesem eine `REQUEST`-Nachricht. Wenn JESS seine Berechnung beendet hat, holt sich der Planer alle wichtigen Fakten aus dem Expertensystem. Dies sind die Anzahl der Teile, aus denen die Melodie bestehen soll, und für jeden Teil ein Status, ein Melodieobjekt und die Menge der Fakten, in denen die Vorgaben für diesen Teil abgelegt sind. Der Status legt fest, nach welchem Verfahren die Komposition des betreffenden Teils erfolgen soll. Mögliche Werte sind der Tabelle 3.2 zu entnehmen[5].

Status	Aktion
new	Neukomposition
copy x	Teil **x** wird unverändert übernommen
Transpose x y	Teil **x** wird um **y** Halbtöne transponiert
Shift x int	Teil **x** wird um das Intervall **int** verschoben
Inversion x	Teil **x** wird umgekehrt
EndClause x	Teil **x** wird mit einer Schlussklausel versehen

Tabelle 3.2: Status

[5]Die Variationen (Transpose, Shift, Inversion, EndClause) werden in Kapitel 3.7 vorgestellt.

Für jeden Teil, der den Status „new“ trägt, muss nun ein Composer gefunden werden. Das geschieht nach demselben Auswahlprinzip, welches der Master zum Finden eines Planers einsetzt. Der Planer sendet an alle n Composer $C_i, 1 \leq i \leq n$ eine `QUERY_IF`-Nachricht, in der er diejenigen Fakten, die den entsprechenden Teil betreffen, mitschickt. Die Composer können anhand dieser Fakten entscheiden, wie gut sie für diesen Melodieteil geeignet sind und sich um die Komposition bewerben. Der Planer wartet, bis er von allen Composern eine Bewerbung erhalten hat oder bis eine bestimmte Zeit verstrichen ist. Dann vergleicht er die Angebote und sucht das beste heraus. Dabei fließen für jeden Composer C_i drei Aspekte ein:

- Der Punktestand $P(C_i)$, der sich als Summe früherer Bewertungen von C_i berechnet
- Die Eignung $E(C_i)$, die sich C_i selbst attestiert
- Die Positionsinformation $L_x(C_i), 0 \leq x \leq 2$; dabei bezeichnet $L_0(C_i)$, wie gut C_i beim Komponieren von Anfangsstücken bewertet wurde, $L_1(C_i)$ entsprechend die Mittelteile und $L_2(C_i)$ die Schlussteile von Melodien.

Die Zahlenwerte werden normiert auf das Intervall $[-1, 1]$ und addiert. Das ergibt die Gesamtwertung $G(C_i)$:

$$G(C_i) = \frac{P(C_i)}{\max\limits_{1 \leq j \leq n} \{|P(C_j)|\}} + (2 \cdot E(C_i) - 1) + \frac{L_x(C_i)}{\max\limits_{1 \leq j \leq n} \{|L_x(C_j)|\}}.$$

Der Composer mit der maximalen Gesamtwertung wird mit der Komposition dieses Teils beauftragt und erhält eine `REQUEST`-Nachricht mit den Auftragsfakten. Ebenso wird mit den übrigen Melodieteilen verfahren, die den Status „new“ haben.
Die Melodieteile, die nicht den Status „new“ haben, können erst erstellt werden, wenn der dazugehörige Melodieteil komponiert wurde. Im Fall „copy x“ wird einfach eine Kopie des Melodieteils x erstellt. Im Variationsfall wird Teil x, sobald er fertig und beim Planer angekommen ist, zu dem entsprechenden Variierer geschickt. Wenn alle Teile fertig sind, wird die Melodie zusammengesetzt und an den Master zurückgesendet.

3.5.3 Wissensbasis

Die Regeln in der Wissensbasis eines Planers lassen sich einteilen in solche, die das gesamte Stück betreffen, und solche, die einen einzelnen Teil betreffen. Ein Beispiel für den ersten Fall ist eine Regel, die die Anzahl der zu komponierenden Teile festlegt, für den zweiten Fall eine Regel, die den Anfangston eines Teiles bestimmt.
Andererseits lassen sich die Regeln dadurch unterscheiden, ob sie beim Lernen berücksichtigt werden oder nicht. Manche Regeln werden bei jedem Auftrag ausgeführt (Initialisierungsregeln u. ä.); sie tragen also nicht dazu bei, ob das Ergebnis gut oder schlecht wird. Es ergibt mithin keinen Sinn, diese Regeln einer Bewertung zu unterziehen. Sie erhalten vorab eine Priorität größer oder kleiner als Null,

je nachdem, ob sie am Anfang oder am Ende eines Kompositionsdurchlaufes ausgeführt werden sollen. Die übrigen Regeln, im folgenden als Lernregeln bezeichnet, erhalten allesamt die Priorität 0. Die Ausführungsreihenfolge der Lernregeln wird über die selbstdefinierte Konfliktlösungsstrategie festgelegt. Exemplarisch für eine Lernregel soll hier folgender Ausschnitt aus einem JESS-Programm betrachtet werden, der eine Regel darstellt, die festlegt, dass der letzte Teil der Melodie auf dem Grundton enden soll.

```
(defrule setLastPitchTonic (declare (salience 0))
  (isActive)
  (numberOfParts ?nop)
  (mel (OBJECT ?m) (tonic ?tonic&:(> ?tonic 0)))
  (not (fact ?nr&:(= ?nr (- ?nop 1)) lastPitch ?))
=>
  (assert (fact (- ?nop 1) lastPitch ?tonic))
  (usedRule setLastPitchTonic (- ?nop 1))
)
```

Jede Regel beginnt mit dem Schlüsselwort `defrule`, gefolgt vom Namen der Regel. Da es sich um eine Lernregel handelt, wird die Priorität gleich 0 gesetzt. Diese Regel liest sich in etwa so:
Wenn

- die Anzahl der Teile gleich `?nop` ist,
- es ein Melodieobjekt mit dem Grundton `?tonic` gibt und
- es kein Faktum gibt, welches für den letzten Teil irgendeinen Wert als letzten Ton (`lastPitch`) vorgibt (der letzte Teil trägt die Nummer `(- ?nop 1)`, da die Nummerierung bei 0 beginnt),

dann

- erzeuge ein neues Faktum, welches besagt, dass Teil `(- ?nop 1)` mit dem Grundton enden soll.

Ungeklärt geblieben ist bisher, was es mit `isActive` und `usedRule` auf sich hat. Die oben betrachtete Lernregel ist insofern gutmütig, als sie nur angewendet wird, wenn nicht schon vorher durch irgendeine andere Regel der Schlusston des letzten Teils festgelegt wurde (letzte Prämisse). Es gibt aber andere Lernregeln, die dies nicht tun, sei es, weil sie zu viele Vorbedingungen prüfen müssten, oder sei es, weil andernfalls der Kontrollfluss der Regelanwendungen zu starr wäre (es könnte ja erwünscht sein, dass zwei oder drei Regeln eines Typs ausgeführt werden, und nicht genau eine). Die Aktivierung dieser Regeln (Erfüllung ihrer Prämissen) ist unabhängig davon, welche von den anderen Regeln schon ausgeführt wurden. Das heißt aber, dass diese Regeln unabhängig von ihrer Bewertung ausgeführt werden, je besser desto früher, je schlechter, desto später. Da aber diese Regeln nur Fakten hinzufügen, die später als ungeordnete Menge weitergegeben werden, spielt diese Reihenfolge überhaupt keine

Rolle, solange die frühere Anwendung einer Regel die spätere Ausführung einer anderen Regel nicht verhindert. Deshalb wird eine Anzahl von Regeln festgelegt, nach deren Ausführung das Programm beendet wird. Dazu braucht man einen Zähler, der sich immer beim Ausführen einer Regel erhöht. Dies geschieht in dem Funktionsaufruf von `usedRule`; die Funktion `usedRule` stellt streng genommen eine Prozedur dar (sie liefert keinen Rückgabewert) und ist folgendermaßen definiert:

```
(deffunction usedRule (?ruleName ?part)
    (bind ?*ruleCounter* (+ ?*ruleCounter* 1))
    (assert (usedRule ?ruleName ?part ?*ruleCounter*))
)
```

Zuerst wird der Regelzähler um 1 erhöht, dann wird ein Faktum hinzugefügt, in dem festgehalten wird, dass die Regel `?ruleName` angewendet worden ist, und auf welchen Melodieteil sich die Anwendung ausgewirkt hat (-1 wenn für alle Teile). Diese Informationen werden später bei der Bewertung benötigt. Wenn der Regelzähler einen bestimmten Wert erreicht hat, wird das Faktum `(isActive)`, welches am Beginn des Programms in die Wissensbasis eingefügt wurde, zurückgenommen, und damit die weitere Ausführung von Lernregeln (das sind genau diejenigen, bei denen `(isActive)` im Prämissenteil vorkommt) verhindert.
Jeder Planer besitzt einen unveränderlichen Satz an Regeln, welche die Mindestanforderungen ausführen, die ein Planer erfüllen muss, damit überhaupt eine Komposition entsteht (wie gut oder schlecht diese ist, spielt zunächst keine Rolle). Dazu gehören Regeln, die den Ablauf des JESS-Programms steuern, und solche, die am Ende – sofern dies vorher noch nicht geschehen ist – Standardwerte setzen. So muss zum Beispiel für jeden Teil der Ambitus[6] festgelegt werden, denn erst wenn die minimale und die maximale Tonhöhe (`minPitch` und `maxPitch`) feststeht, können die für das Komponieren verfügbaren Töne ermittelt werden. Deshalb gibt es eine Regel, die unabhängig von irgendwelchen anderen Prämissen für jeden Melodieteil einen Standardwert für `minPitch` (bzw. für `maxPitch`) setzt, sofern nicht schon irgendein anderer Wert dafür festgelegt wurde. Diese Regel hat eine negative Priorität, das heißt, sie wird erst dann ausgeführt, wenn von den übrigen Regeln – insbesondere den Lernregeln – keine mehr ausgeführt werden kann. Sie verhindert damit, dass die Angaben `minPitch` und `maxPitch` leer bleiben, was als Resultat eine leere Melodie (also eine Melodie ohne Töne) liefern würde.

3.5.4 Benutzeroberfläche

Die GUI des Planers dient nicht zur Interaktion mit dem Benutzer, sondern allein als Kontrollanzeige. Besonders beim Debuggen und Testen leistet sie wertvolle Dienste. Der Benutzer hingegen, der vor dem fertigen Agentensystem sitzt, wird sie nicht benötigen. Die Planer-GUI kann geöffnet werden, indem der Benutzer in der Master-GUI den Planer in der Liste von Agenten auswählt und den Button „Open

[6]Unter Ambitus versteht man den Tonumfang einer Melodie, also das Intervall zwischen dem tiefsten und dem höchsten vorkommenden Ton.

Faktum	Intention
tonic	Grundton
tonality	Tonart
rhythm	Rhythmus
upbeat	Auftakt
validInterval	erlaubtes Intervall
validPitch	erlaubte Tonhöhe
validDuration	erlaubte Notendauer
totalDuration	Gesamtlänge des Melodieteils
minPitch	niedrigste erlaubte Tonhöhe
maxPitch	höchste erlaubte Tonhöhe
firstPart	Es handelt sich um den Anfang einer Melodie.
lastPart	Es handelt sich um den Schluss einer Melodie.
firstPitch	Anfangston
lastPitch	Endton
firstInterval	erstes Intervall
melodyAscending	Die Melodie soll aufsteigen.
melodyDescending	Die Melodie soll absteigen.
characteristicInterval	charakteristisches Intervall; möglichst oft zu verwenden
allowSyncopes	Synkopen dürfen verwendet werden

Tabelle 3.3: Fakten, die der Planer an den Composer weitergibt

GUI“ drückt. Das Fenster, welches nun angezeigt wird, enthält drei Bereiche: Im oberen Textfeld zeigt der Planer an, aus wie vielen Teilen die komponierte Melodie besteht, sowie zu jeden Teil den Status und gegebenenfalls, welcher Composer diesen Teil ausführt. In der Mitte befindet sich eine Tabelle, in der alle Regeln aufgelistet sind, die dem Planer zum Lernen dienen. Das heißt, es sind diejenigen Regeln ausgeklammert, die alle Planer standardmäßig haben und die nicht bewertet werden. In der zweiten Spalte steht zu jeder Regel deren interne Bewertung, die im Konfliktfall die Ausführungsreihenfolge bestimmt. Im unteren Teil des Fensters werden Logginginformationen ausgegeben.

3.6 Composer

Der Composer bekommt vom Planer die Aufgabe, ein Melodiestück zu komponieren. Dabei hat der Planer bestimmte Vorgaben gemacht, die der Composer nach Möglichkeit erfüllen sollte. Es kann auch Situationen geben, wo dies nicht möglich ist. Angenommen die Vorgaben sind, dass das Melodiestück mit dem Grundton beginnen soll und mit einem anderen Ton enden, es darf aber nur das Intervall 0 (Tonwiederholung) verwendet werden. Es ist klar, dass zu dieser Problemstellung keine Lösung existiert. Es kann aber auch sein, dass ein Composer nicht die richtigen Regeln zur Verfügung hat, um die Vorgaben korrekt umzusetzen. In diesem Fall soll-

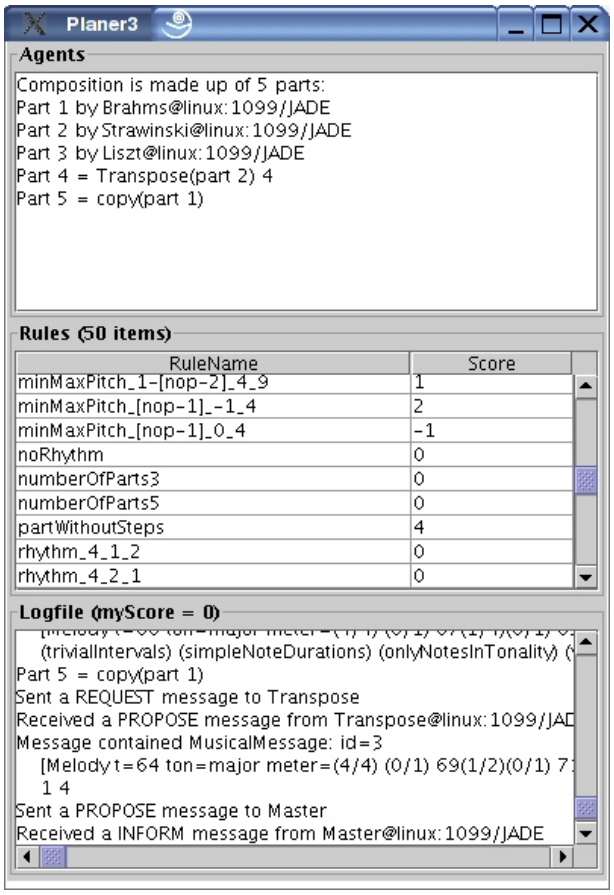


Abbildung 3.3: GUI eines Planers

te man erwarten, dass dieser Composer mit der Zeit wegen schlechter Bewertungen ausselektiert wird.

3.6.1 Bewerbung

Bei einer Bewerbung um einen Kompositionsauftrag gibt der Composer zwei Dinge an: Erstens seine allgemeine Qualität, das heißt, wie gut er bisherige Aufträge ausführen konnte (Summe früherer Bewertungen); zweitens seine persönliche Einschätzung, wie gut er für diesen speziellen Auftrag geeignet ist. Wie kann ein Composer nun seine Eignung bestimmen? Dazu prüft er für die in dem Auftrag angegebenen Fakten, ob sie in seiner Wissensbasis auftauchen, und zwar im Prämissenteil mindestens einer seiner Regeln. Ist dies für ein bestimmtes Faktum nicht der Fall, so ist das ein sicheres Indiz dafür, dass dieses Faktum bei der Komposition nicht beachtet werden wird. Mit anderen Worten, das Kompositionsergebnis wäre exakt das gleiche, wenn man besagtes Faktum weglassen würde. Ein Maß für die Eignung für einen Auftrag ist demnach der Anteil der Auftragsfakten, die der Composer kennt (d.h. die in seiner Wissensbasis vorkommen). Dieser Wert bewegt sich zwischen 0 (ungeeignet) und 1 (100-prozentige Eignung).

3.6.2 Auftragsbearbeitung

Erhält der Composer nach erfolgreicher Bewerbung einen Auftrag vom Planer, so setzt er seine JESS-Engine mit dem Befehl `reset` auf den Anfangszustand zurück. Alte Fakten, die von vorangegangenen Aufträgen stammen, werden damit gelöscht. Dann überträgt er die Fakten, die er vom Planer in der Auftragsnachricht bekommen hat und die den Auftrag spezifizieren, in die JESS-Engine und startet sie. Nach der Berechnung kann die fertige Melodie aus JESS extrahiert werden. Der Composer schickt sie an denselben Planer zurück, von dem er den Auftrag erhalten hat.

3.6.3 Wissensbasis

Wie geht nun das Komponieren vonstatten? Jede Note wird durch ein ungeordnetes Faktum repräsentiert; ein solches Faktum hat feste *slots*, die mit verschiedenen Werten belegt werden können (ungeordnet deshalb, weil die *slots* keiner Reihenfolge unterworfen sind). Ein Notenfaktum hat die *slots* Tonhöhe (`pitch`), Notendauer (`duration`), Position (`position`) und eine fortlaufende Nummer (`number`), wobei `pitch` und `number` Integerwerte sind, `duration` und `position` dagegen Brüche vom eigens dafür definierten Typ `Rational`. Somit können neben halben, Viertel- und Achtelnoten auch problemlos punktierte Noten, Triolen, Quintolen und jede beliebige andere rationale Notendauer dargestellt werden.
Die Wissensbasis setzt sich zusammen aus Initialisierungsregeln, Backtrackingregeln und Regeln, die ein Notenfaktum erzeugen, verändern oder entfernen. Die meisten Regeln befolgen die Trennung von Rhythmus und Melodieführung: Eine Regel erzeugt entweder ein Notenfaktum, wobei Position und Notendauer festgelegt werden, aber nicht die Tonhöhe, oder sie verändert ein Notenfaktum, indem sie die Tonhöhe setzt (Wert im *slot* `pitch` setzen).

Backtracking

Die ganze Berechnung ist so konzipiert, dass sie backtracking-fähig ist. Dazu gibt es einen Zähler `level`, der als Faktum repräsentiert ist und in jedem Berechnungsschritt um eins erhöht wird. Wenn sich die Komposition auf einem Irrweg befindet, wenn also keine Regel mehr angewendet werden kann, obwohl die Melodie noch nicht vollständig ist, dann muss der letzte Schritt rückgängig gemacht werden und eine andere Alternative probiert werden. Dazu muss in jedem Level gespeichert werden, welche Möglichkeiten (Regelanwendungen) schon getestet wurden. Weiterhin müssen alle Veränderungen in einem Schritt gespeichert werden, damit diese im Backtracking-Fall wieder zurückgenommen werden können.

Positionsbestimmung

Bevor eine Regel ein Notenfaktum hinzufügen kann, muss klar sein, an welcher Position innerhalb der Komposition die Note stehen soll. Die Position wird ab Beginn des ersten Volltaktes (d.h. ohne den Auftakt einzubeziehen) gezählt. Der erste Schlag im ersten Volltakt hat die Position 0, in einem $\frac{4}{4}$-Takt entspricht Position $\frac{3}{2}$ dem

Schlag 3 im zweiten Takt usw. Negative Positionsangaben beziehen sich auf den Auftakt, bei drei Achteln als Auftaktnoten hat also die erste die Position $-\frac{3}{8}$. Es muss sorgfältig darauf geachtet werden, dass sich nicht zwei Noten an derselben Position befinden, oder sich die Notendauern überschneiden, also Position plus Notendauer einer Note größer ist als die Position der darauffolgenden Note. Schließlich sollen die Kompositionen unserer Agenten nur einstimmig sein. Um diese Bedingung einzuhalten, gibt es ein Faktum `freePosition`, welches eine Position angibt, an der sich noch keine Note befindet, sowie die Dauer dieser Lücke. Die Notendauer der neu einzufügenden Note darf also nicht länger sein als die Dauer der Lücke. Die Regel, die die Note einfügt, muss gleichzeitig das `freePosition`-Faktum anpassen: Die Position wird um die Notendauer erhöht, die Lückendauer um denselben Wert verringert. Manchmal wird das `freePosition`-Faktum auch ganz entfernt, wenn zum Beispiel die eingefügte Note die Lücke genau ausfüllt. In diesem Fall sind Regeln nötig, die, wenn kein `freePosition`-Faktum existiert, prüfen, ob noch Lücken in der Melodie sind, und gegebenenfalls ein neues `freePosition`-Faktum erzeugen.

Tonhöhenbestimmung

Vom Planer erhält der Composer die Vorgaben, welche Tonhöhen er verwenden darf (`validPitch`, in der Regel ist das der Tonvorrat der zugrunde liegenden Tonart) und welche Intervalle in der Melodie vorkommen dürfen. Eine allgemeine Regel, um die Tonhöhe einer Note zu bestimmen, ist, die Tonhöhe der Vorgängernote (sofern vorhanden) zu ermitteln (p') und dann eine Kombination von einer gültigen Tonhöhe p (`validPitch`) und einem zulässigen Intervall i (`validInterval`) zu suchen, so dass $p = p' + i$ gilt. Allerdings wird bei konsequenter Anwendung dieser Regel nur eine vom Zufall bestimmte Tonfolge entstehen, je nachdem, welche `validPitch`- und `validInterval`-Fakten zuerst „gefunden“ werden. Eine Klasse von Regeln, die etwas weiter gehen, beachten nicht die vorangegangene Tonhöhe, sondern das vorangegangene Intervall. Dies setzt natürlich voraus, dass sich vor der zu bearbeitenden Note schon zwei fertige Noten (mit bestimmten Tonhöhen) befinden. Ein paar dieser Regeln werden beim Erzeugen eines Composers automatisch nach dem Zufallsprinzip generiert. Sie haben z. B. die Form „Wenn das vorangegangene Intervall eine Terz ist, dann gehe eine Sekund abwärts“.
Der Fantasie, sich immer ausgefeiltere Regeln zur Melodieführung auszudenken, sind hier keine Grenzen gesetzt.

Reihenfolge

Die Regeln zum Komponieren sind so geschrieben, dass sie jede beliebige Reihenfolge, in der die Noten erzeugt werden, zulassen. So ist es unter anderem möglich, eine Melodie „von hinten“ zu komponieren, indem mit der letzten Note begonnen wird, und dann in absteigender Positionsreihenfolge die restlichen Noten davorgesetzt werden, oder am Anfang eine Note in der Mitte der Melodie zu setzen, die den Höhepunkt markiert, und anschließend Noten davor und dahinter einzufügen. Das heißt aber, dass der *slot* `number` in den Notenfakten nur die Reihenfolge angibt, in der die Noten erzeugt wurden, nicht aber deren Abspielreihenfolge. Um diese nicht

immer mühsam aus den Werten für den *slot* `position` herausfiltern zu müssen, wird eine Liste geführt, in der die Nummern der Notenfakten in der Reihenfolge stehen, in der sie später in der fertigen Melodie auftauchen.

Beispiel

Betrachten wir nun eine Regel, die ein Notenfaktum hinzufügt:

```
(defrule insertNote1
   (declare (salience -1))
   ?levelFact <- (level ?level)
   ?nlFact <- (noteList $?notes)
   ?freePosFact <- (freePosition ?nr ?pos ?freeDur)
   (validDuration ?dur&:(and (?dur leq ?freeDur)
                             (not (isSyncope ?pos ?dur))
   )                     )
   (not (testedInLevel ? insertNote -1
                        ?pN&:(= ?pN (?pos getNumerator))
                        ?pD&:(= ?pD (?pos getDenominator))
                        ?dur
   )    )
=>
   (retract ?levelFact)
   (retract ?nlFact)
   (retract ?freePosFact)
   (assert (testedInLevel ?level insertNote -1
                                 (?pos getNumerator)
                                 (?pos getDenominator)
                                 ?dur
   )       )
   (bind $?notes (insert$ $?notes ?nr ?level))
   (assert (noteList $?notes))
   (assert (freePosition (+ ?nr 1)
                        (?pos plus ?dur)
                        (?freeDur minus ?dur)
   )       )
   (setNote ?level -1 ?dur ?pos insertNote1)
   (assert (level (+ ?level 1)))
)
```

Diese Regel realisiert eine der Mindestanforderungen an einen Composer, nämlich die, dass die fertige Melodie keine „Lücken“ haben darf (Pausen werden explizit gesetzt und zählen hier nicht als Lücke). Ohne weitere Vorgaben (wie z. B. den Rhythmus) zu beachten, wird eine beliebige Note in eine Lücke gesetzt, sofern keine andere Regel mehr anwendbar ist. Deshalb erhält diese Regel die Priorität -1. Sie liest sich in Worten etwa so:

Wenn

- an der Stelle `?pos` eine Lücke der Länge `?freeDur` ist,
- `?dur` eine gültige Notendauer ist, die nicht länger als `?freeDur` und keine Synkope ist,
- und vorher noch nicht versucht wurde, an die Position `?pos` eine Note der Länge `?dur` zu setzen,

dann

- entferne zunächst die veralteten Fakten `level`, `noteList` und `freePosition`,
- merke die getestete Möglichkeit,
- füge die neue Note in die Notenliste ein und speichere diese in einem Faktum,
- erzeuge ein neues Faktum `freePosition` mit einer Lücke, die um `?dur` nach hinten verschoben und verkürzt wurde,
- füge durch den Funktionsaufruf `setNote` das neue Notenfaktum ein und
- erhöhe das Level um 1.

3.6.4 Benutzeroberfläche

Die GUI eines Composers ist ähnlich aufgebaut wie die eines Planers. Ebenso wie die Planer-GUI dient sie nicht der Interaktion mit dem Benutzer, sondern zur Anzeige von Statusinformationen, die hauptsächlich zum Debuggen gedacht sind. Auch die Composer-GUI ist in drei Teilbereiche aufgeteilt (s. Abbildung 3.4): Oben zeigt der Composer ein eben komponiertes Melodiestück an, in der Mitte eine alphabetisch geordnete Tabelle mit allen Lernregeln und deren Bewertungen und unten ein Textfeld für Logginginformationen.

3.7 Variierer

Variierer sind Agenten, die ein vorgegebenes Melodiestück in einer bestimmten Weise verändern. Jeder dieser Agenten übernimmt dabei ein spezielles Verfahren. Da die meisten dieser Verfahren, wie zum Beispiel Transposition, eindeutig definiert sind, ist der Agententyp der Variierer von der Bewertung durch den Benutzer ausgeschlossen. Die jeweilige Aufgabe ist fest in den Agenten einprogrammiert, der seine Eigenschaften während seines Lebenszyklus' nicht ändert. Für jedes Variationsverfahren gibt es genau einen Agenten auf der Plattform. Die Variierer besitzen keine Wissensbasis und keine GUI. Im folgenden werden die Variationsverfahren aufgelistet, die als Agenten programmiert wurden.

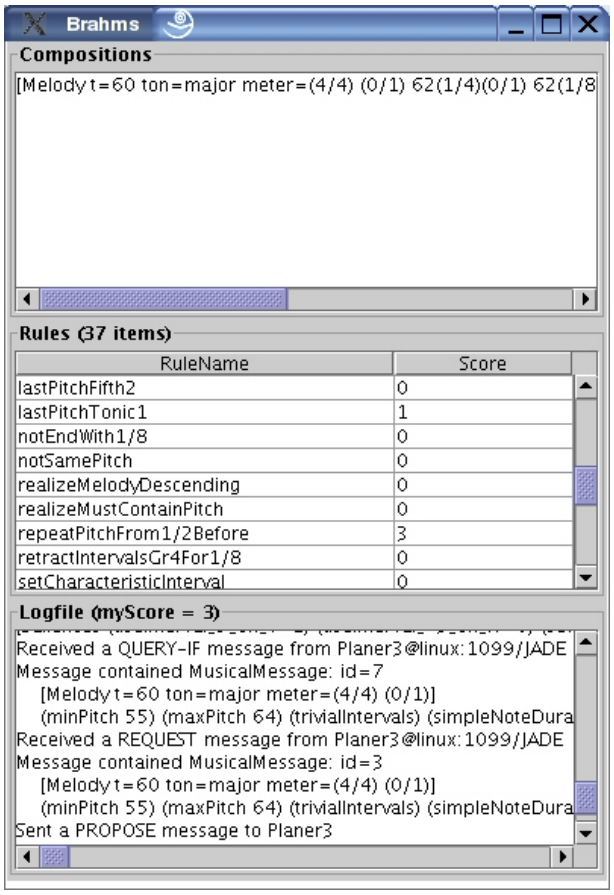
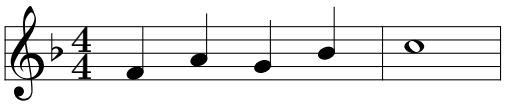


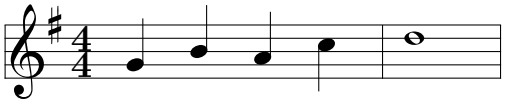
Abbildung 3.4: GUI eines Composers

3.7.1 Transposition

Der Agent mit dem Namen `Transpose` transponiert eine Melodie um eine gegebene Anzahl von Halbtönen nach oben oder unten. Dazu muss er jede einzelne Note um dieses Intervall verschieben, sowie den Grundton der Tonart anpassen. Beispiel: Der Melodieausschnitt



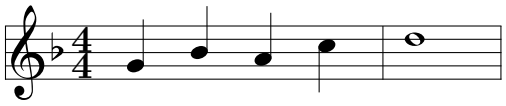
wird um zwei Halbtöne nach oben transponiert:



Die Tonart (vorher F-Dur) wird zu G-Dur, am Takt und an den Notenlängen ändert sich nichts.

3.7.2 Shiften

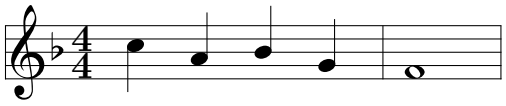
Der Shift-Agent funktioniert ähnlich wie der Transponierer, jedoch verschiebt er jeden Ton so, dass er nachher noch im Tonvorrat der vorgegebenen Tonart enthalten ist. Die Angabe, wie weit verschoben werden soll, wird hier nicht in Halbtönen angegeben, sondern in klassischen Intervallbezeichnungen. Das obige Beispiel um eine Sekund nach oben verschoben, liefert folgendes Ergebnis:



Der erste Ton (f) wird um eine große Sekund nach oben verschoben, der zweite (a) dagegen nur um eine kleine Sekund. Das Transponieren lieferte hier den Ton h, das darf aber beim Verschieben nicht sein, da h nicht im Tonvorrat von F-Dur vorhanden ist.

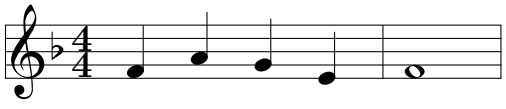
3.7.3 Umkehrung

Unter der Umkehrung einer Melodie versteht man diejenige Tonfolge, die entsteht, wenn man jedes Intervall der Ausgangsmelodie genau in entgegengesetzter Richtung notiert. Beginnt also die Melodie beispielsweise mit einer aufsteigenden Quarte, so fängt die umgekehrte Melodie mit einem Quartsprung nach unten an. Aus absteigenden Tonleitern werden aufsteigende Tonleitern, usw. Anders gesagt werden alle Töne an einer bestimmten Tonhöhe gespiegelt. Damit das Resultat im gleichen Tonhöhenbereich wie die Originalmelodie liegt, wird an der Tonhöhe gespiegelt, die sich als Durchschnitt aus der höchsten und tiefsten vorkommenden Note berechnet. Der Rhythmus, die Takt- und Tonart bleiben unverändert. Das obige Beispiel ergibt damit bei dieser Variationsmethode:



3.7.4 Schlussklausel erzeugen

Hierbei werden die letzten Noten eines Melodieteils modifiziert, und zwar in der Weise, dass die Melodie mit einer so genannten Diskantklausel endet. Dieser Begriff besagt, dass der letzte Ton – der gleichzeitig der Grundton ist – vom Leitton her erreicht wird, der letzte Schritt also eine Sekund aufwärts ist. Die Beispielmelodie aus den vorigen Abschnitten würde folgendermaßen variiert:



3.8 Hilfsagenten

Neben den schon erwähnten Agententypen, die am Kompositionsprozess beteiligt sind, gibt es noch zwei weitere, die Hilfsfunktionen wahrnehmen, die mit dem Komponieren selbst nichts zu tun haben. Es handelt sich um den so genannten Selektierer und den so genannten Player.

3.8.1 Selektierer

Dieser Agent wird tätig, wenn es darum geht, andere Agenten aus der Plattform zu entfernen oder neue Agenten zu erzeugen. Immer dann, wenn der Benutzer eine entsprechende Schaltfläche auf der GUI des Master-Agenten wählt („Kill...“, „Create...“, „Combine two...“), schickt der Master einen Auftrag an diesen Agenten, der deshalb den Namen Darwin erhält.
Darwin führt eine Liste mit den Namen aller Composer und Planer und ihren Bewertungen. Beim Herunterfahren der Agentenplattform wird diese als Datei gesichert, um beim nächsten Start wieder geladen werden zu können. Diese Liste ist notwendig, um die geforderten Befehle auszuführen: Bei „Kill Composer“ und „Kill Planer“ muss der Selektierer denjenigen Composer oder Planer heraussuchen, der die schlechteste Bewertung hat. An diesen schickt er dann eine `INFORM`-Nachricht mit dem Inhalt „KILL“, woraufhin sich dieser selbst beendet. Beim Befehl „Combine“ dagegen geht es darum, die beiden besten Agenten des geforderten Typs (also die mit der höchsten Bewertung) zu ermitteln und miteinander zu kreuzen. Das Verfahren, nach dem die Regeln ausgewählt werden, die der neue Agent erhält, ist nicht ganz trivial. Erhält das Kind alle Regeln jeweils beider Elternteile, so würde dies dazu führen, dass die Agenten mit der Zeit immer umfangreicher werden und das System an seine Grenzen stößt. Wählt man dagegen von jedem Elternteil nur die besten 50 Prozent der Regeln aus, so liegt das Problem darin, dass die Regelmengen der Agenten möglicherweise nicht disjunkt sind. Es kann im Extremfall sein, dass die Menge der besten Regeln bei Mutter und Vater identisch ist, das Kind somit nur die halbe Anzahl an Regeln besäße. Um diese beiden Probleme zu umschiffen, wurde folgendes Verfahren implementiert: Die Anzahl der Regeln, die das Kind erben soll, berechnet sich aus dem arithmetischen Mittel der Regelanzahl von Mutter und Vater, also:

$$|R_{Kind}| = \frac{|R_{Vater}| + |R_{Mutter}|}{2}\ , \quad (R_X = \text{Regelmenge des Agenten } X).$$

Die Regeln der Eltern werden nach ihren Bewertungen absteigend sortiert. Bezeichne $V_1, V_2, \ldots, V_m$ die sortierten Regeln des Vaters und $M_1, M_2, \ldots, M_n$ die der Mutter, wobei $m = |R_{Vater}|$ und $n = |R_{Mutter}|$ ist. Es wird nun immer abwechselnd eine Regel des Vaters und eine der Mutter in die Wissensbasis des neuen Agenten eingefügt: $V_1, M_1, V_2, M_2, \ldots$. Tritt der Fall auf, dass zum Beispiel $V_i = M_j$ mit $j \leq i - 1$ gilt (analog $M_i = V_j$ mit $j \leq i$), das heißt, eine Regel schon vorher eingefügt wurde, so wird die Liste des Vaters (bzw. der Mutter) so lange durchgegangen, bis eine Regel gefunden wird, die noch nicht in der Wissensbasis des Kindes enthalten ist.

In diesem Beispiel wäre das eine der Regeln $V_{i+1}, V_{i+2}, \ldots$ (bzw. $M_{i+1}, M_{i+2}, \ldots$). Immer wenn sich die Bewertung eines Agenten ändert, sendet dieser eine `INFORM`-Nachricht an den Selektierer, damit dieser seine Daten auf dem laufenden Stand halten kann.
Da die Agentenplattform beim ersten Start keine Composer und Planer enthält, muss es auch möglich sein, diese zu erzeugen. Der Agent Darwin kann sowohl Composer als auch Planer erzeugen. Dieser Vorgang besteht im Wesentlichen darin, einen Satz von Regeln für die Wissensbasis des zu erzeugenden Agenten zu erstellen. Dazu bedient Darwin sich zweier Verfahren: Zum einen wählt er aus einem fest vorgegebenen Korpus von Regeln einige aus, zum anderen erzeugt er dynamisch neue Regeln. Es wurden zwei Regelkorpora manuell erstellt, und zwar einer für die Composer-Regeln und einer für die Planer-Regeln. Der erstere enthält 55 Regeln, der andere 99. Darwin wählt beim Kreieren eines neuen Agenten nach dem Zufallsprinzip etwa die Hälfte der Regeln aus dem entsprechenden Korpus aus. Von den Regeln, die neu erzeugt werden, gibt es mehrere Typen. Ein Typ von Regeln ist dabei in seiner Anwendung auf einen engen Bereich eingeschränkt und hat ein immer gleiches Grundgerüst. Zum Beispiel werden für einen neuen Planer Regeln erzeugt, die Rhythmen erstellen.

3.8.2 Player

Der Player-Agent erhält Aufträge vom Master, und zwar immer dann, wenn der Benutzer einen der Play-Buttons auf der GUI des Master-Agenten gedrückt hat. Der Master schickt ihm dazu die abzuspielende Melodie, die der Player in eine MIDI-Sequenz umwandelt und dann über den systemeigenen MIDI-Sequencer abspielt. Erhält der Player einen neuen Auftrag, bevor der vorige zu Ende geführt wurde, so wird der Sequencer gestoppt und die neue Melodie wird gespielt.

Kapitel 4

Lernen

Lernen ist wie Rudern gegen den Strom. Sobald man aufhört, treibt man zurück.

— Benjamin Britten

Ein wesentlicher Gesichtspunkt dieser Studienarbeit ist das Lernen, das die musikalischen Agenten dazu befähigen soll, ihre Kompositionen mit der Zeit immer besser dem Geschmack des Benutzers anzupassen. Um dies zu erreichen, ist es wichtig, dass der Benutzer jedes Resultat bewertet. Nach einem in diesem Kapitel zu bestimmenden Lernprinzip müssen die Agenten daraus Konsequenzen ziehen.

4.1 Was ist Lernen?

Herbert Simon [17] definiert den Begriff „Lernen“ folgendermaßen: „Lernen ist jede Veränderung eines Systems, die es ihm erlaubt, eine Aufgabe bei der Wiederholung derselben Aufgabe oder einer Aufgabe derselben Art besser zu lösen“.
Die musikalischen Agenten können nach dieser Definition zu Recht als lernendes System bezeichnet werden, da sich ihr innerer Zustand nach jeder Bewertung ändert und diese Änderung dazu dient, dieselbe Aufgabe (eine Melodie zu komponieren) im nächsten Schritt besser zu lösen, das heißt, eine Melodie zu generieren, die dem Benutzer besser gefällt. Das geschieht nach dem Prinzip, das Wrobel/Morik/Joachims in [6, Kap. 14] nennen: „Maschinelles Lernen wird überwiegend eingesetzt, um [...] eine gegebene Regelmenge zu verbessern“. Im vorliegenden Fall wird die Regelmenge in der Weise verbessert, dass die vorhandenen Regeln unterschiedlich gewichtet werden. Wie das im Einzelnen funktioniert wird im Abschnitt 4.2.4 beschrieben.

4.2 Bewertung

Es gibt viele Möglichkeiten, auf welche Art und Weise die Bewertung durch den Benutzer geschehen soll. In diesem Abschnitt werden kurz verschiedene Ansätze

diskutiert und der hier gewählte vorgestellt.

4.2.1 Mögliche Ansätze

Am aufschlussreichsten ist sicher eine begründete Kritik, in der der Hörer genau erläutert, warum ihm eine Melodie nicht gefallen hat und was besser gemacht werden könnte. Diese Methode stellt aber einerseits sehr hohe Ansprüche an das Agentensystem, welches eine solche Kritik auswerten müsste, andererseits auch an den Benutzer, der über ein gewisses musikalisches Fachwissen verfügen müsste, um eine solche Kritik formulieren zu können.
Eine weniger anspruchsvolle Möglichkeit ist eine mehrdimensionale Bewertung, in dem Sinne, dass verschiedene Aspekte der Komposition (z. B. Rhythmus, Tonart, Schlussklausel, u. a.) getrennt mit einer Note bewertet werden. Auch diese Variante soll hier nicht weiter verfolgt werden, da sie das Lernen in zu enge Bahnen zwingt. So würde bei diesem Verfahren genau eine Tonart, eine Taktart usw. gelernt werden. Gefallen einem Benutzer aber gerade die Kombinationen „Dreiertakt in Moll“ und „Vierertakt in Dur“, andere Takt-Tonart-Verbindungen dagegen nicht, so wird die mehrdimensionale Bewertung nicht zum Ziel führen. Außerdem verlangt sie dem Benutzer ein hohes Maß an Geduld ab, da er eine Vielzahl von Parametern einstellen muss.
Diese Arbeit wird sich daher darauf beschränken, dass der Benutzer die einzelnen Kompositionen mit Punktzahlen benotet.

4.2.2 Credit-Assignment Problem

„The basic problem any learning system is confronted with is the credit-assignment problem (CAP)“ [19]. Mit dem CAP bezeichnet man die Frage, wie in einem lernenden System eine Bewertung (Belohnung oder Strafe) den einzelnen Komponenten zugeordnet werden soll, wenn mehrere an der Lösung einer Aufgabenstellung beteiligt waren. Der Benutzer bewertet in der Regel das gesamte Resultat, was die schwierige Frage aufwirft, welcher Agent positiv und welcher negativ auf das Gesamtergebnis eingewirkt hat. In [19] wird unterschieden zwischen dem inter-agent CAP und dem intra-agent CAP. Bei ersterem werden die nach außen sichtbaren Aktionen der Agenten betrachtet, insbesondere dabei die Frage, welcher Anteil einer Bewertung einem jeden Agenten zusteht. Das intra-agent CAP bezieht sich auf die interne Verarbeitung einer Bewertung in einem Agenten, also die Frage, welche Inferenzen ihn zu seinem Handeln veranlasst haben und wie diese innerhalb des Agenten zu bewerten sind. Beide Fälle des CAP lassen sich an den musikalischen Agenten festmachen, wie im folgenden gezeigt wird.

4.2.3 Bewerten der Agenten

Das inter-agent CAP wird insofern umgangen, als der Benutzer nicht nur eine einzige Note für die gesamte Melodie vergibt, sondern auch jeden Teil für sich bewertet. Auf diese Weise kann die Bewertung gut auf die beteiligten Agenten aufgeteilt werden:

Der Planer erhält die Gesamtbewertung, die eingesetzten Composer die Bewertung des von ihnen komponierten Melodieteils (s. hierzu auch Kapitel 3.4).
Jeder Agent A hat einen Punktestand $P(A)$, der ein Maß für die Qualität von A (in den Augen eines bestimmten Benutzers) ist und sich bei jeder Bewertung B nach folgender Formel ändert:

$$P_{\text{neu}}(A) = \lfloor (P_{\text{alt}}(A) + B) \cdot \frac{9}{10} \rfloor, \quad B \in \mathbb{Z}, \ -3 \leq B \leq 3.$$

Der Faktor 0.9 bewirkt, dass der Punktestand eines Agenten nicht allzu groß wird. $P(A)$ kann mit dieser Formel betragsmäßig nicht größer als 30 werden und bewegt sich somit in einem überschaubaren Bereich.

4.2.4 Bewerten der Regeln

Das intra-agent CAP dagegen lässt sich nicht so elegant umgehen: Zwar protokolliert jeder Agent (Planer oder Composer), welche Regeln er bei einem Kompositionsvorgang angewendet hat, er kann aber nicht entscheiden, welchen Anteil die einzelne Regel zur Verbesserung oder Verschlechterung des Resultats beigesteuert hat. So bleibt als einziger, nicht unproblematischer Ausweg, eine Bewertung auf alle angewendeten Regeln gleichmäßig zu verteilen. Eine Ausnahme bildet der Planer, denn er kann seine Regeln wenigstens danach differenzieren, ob sie auf die gesamte Komposition oder nur auf einen Teil gewirkt haben. So prägen Regeln, die die Anzahl der Teile oder den Rhythmus festlegen, die ganze Melodie, dagegen solche, die den Status oder den Ambitus eines Teils bestimmen, nur diesen einen Melodieteil. Je nachdem wird eine Regel somit mit der Gesamtbewertung oder mit einer Teilbewertung gewichtet. Im Gegensatz dazu kann ein Composer die Bewertung des von ihm komponierten Teils nur zu gleichen Teilen an die verwendeten Regeln weitergeben. Ebenso wie die Agenten hat auch jede Regel innerhalb eines Agenten einen Punktestand, der sich gemäß der Formel aus dem vorigen Abschnitt bei jeder Bewertung ändert. Der Punktestand einer Regel spiegelt die Priorität wider, mit der diese Regel beim Komponieren an die Reihe kommt. Es werden also Regeln bevorzugt, die durch gute Bewertungen einen hohen Punktestand erzielt haben.

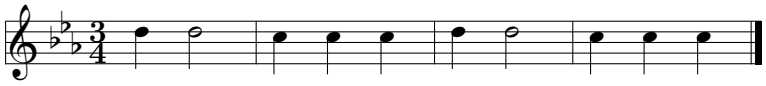
4.3 Resultate

Bevor im folgenden Unterkapitel verschiedene Lernstrategien miteinander verglichen werden, soll zunächst ein Eindruck vermittelt werden, wie sich das System im Training mit einem Benutzer verhält und welche Ergebnisse zu erzielen sind.

4.3.1 Melodien ohne Training

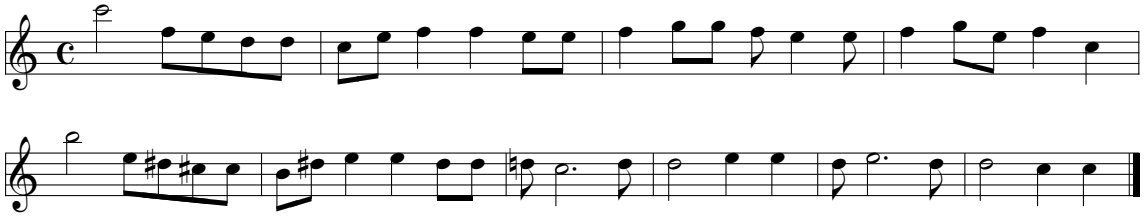
Um einen Einblick in die Variationsbreite der möglichen Kompositionen zu geben, werden einige Melodien vorgestellt, die von musikalischen Agenten komponiert wurden, die sich noch nicht einem Training unterzogen hatten. Sie mögen als Vergleich dienen für die Resultate, die mit Training erzielt wurden.

Melodie 1:



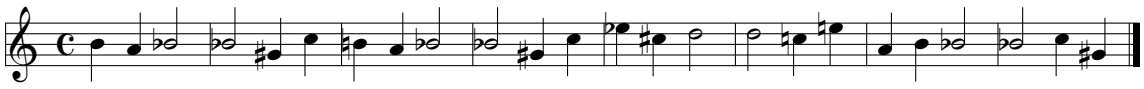
Diese Melodie ist zum einen sehr kurz, zum anderen ist sie nicht sonderlich einfallsreich: Die gesamte Komposition besteht aus lediglich zwei Tönen, die in einem einfachen Rhythmus wiederholt werden.

Melodie 2:



Melodie 2 ist dagegen schon wesentlich länger (10 Takte). Allerdings stören hier die tonartfremden Noten (cis und dis) in den Takten 5 und 6. Dazu kommt der in den Takten 7 und 9 auftretende Rhythmus mit der punktierten halben Note auf der Zählzeit „eins und“, der alles andere als gefällig ist.

Melodie 3:

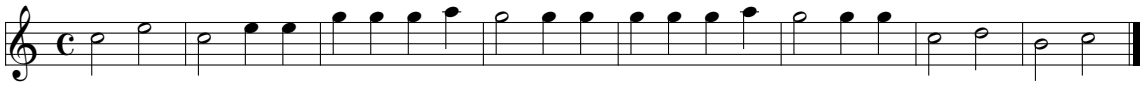


Die dritte Melodie erinnert hingegen stark an die Zwölftonmusik, kommen doch alle Töne im Bereich von gis bis e vor. Beim Hören lässt sie sich auch keiner bestimmten Tonart zuordnen. Diese Melodie wirkt wie eine beliebige Zufallsfolge von Tönen, denen allein der gleichmäßige Rhythmus eine gewisse Struktur einhaucht. Der Schluss hinterlässt beim Hörer den ungewissen Eindruck, dass das Stück noch gar nicht zu Ende sein könnte – zu abrupt endet die Melodie im letzten Takt.

4.3.2 Melodien mit Training

Einige Melodien, die mit dem vom Autor trainierten Agentensystem komponiert wurden, seien hier als Vergleich angegeben. Der Leser möge sich dazu sein eigenes Urteil bilden. Die Melodien entstanden nicht alle in derselben Trainingssitzung.

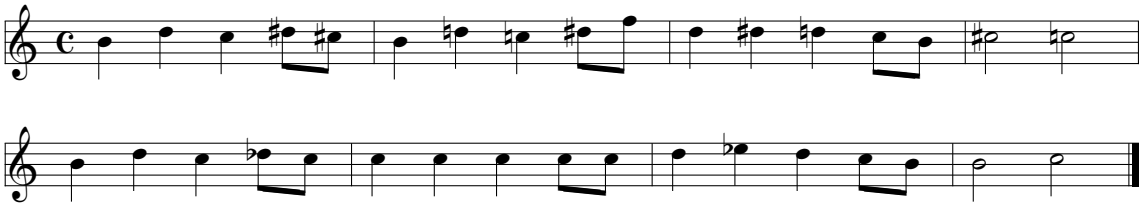
Melodie 4:



Diese Melodie erinnert von der Anlage her in ihrer Schlichtheit mit der Wiederholung im Mittelteil (Takte 3-4 und 5-6) an einfache Kinderliedmelodien wie „Hänschen klein“ oder „Alle meine Entchen“. Sehr stimmig wird der zweimalige Höhepunkt mit

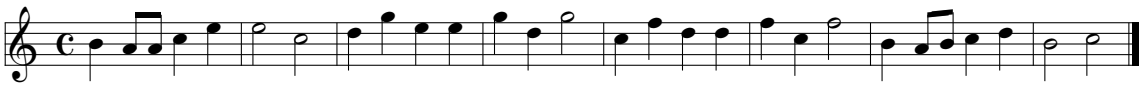
der Note a angesteuert, etwas einfallslos ist dagegen der Schluss. Ein Lernprotokoll, in dem alle Lernschritte mit den dazugehörenden Melodien und Bewertungen nachvollzogen werden können, ist in Anhang A abgedruckt. Darin ist auch zu sehen, dass zum Training von Melodie 4 insgesamt 13 Lernschritte nötig waren.

Melodie 5:



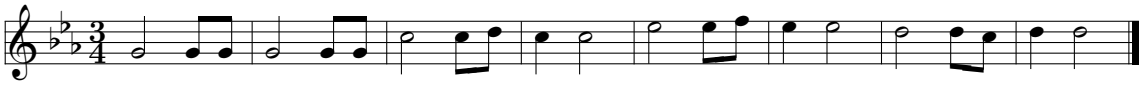
Tonal wesentlich komplexer ist Melodie 5, was schon an den zahlreichen Vorzeichen zu erkennen ist. Rhythmisch ist sie ganz klar in zwei Phrasen mit jeweils vier Takten Länge gegliedert. Die überproportional auftretenden Halbtonschritte geben dieser Melodie ihren besonderen Reiz.

Melodie 6:



Auch wenn der Anfang dieser Melodie etwas holprig wirkt, so sind doch die mittleren Takte sehr eingängig. Dazu trägt unter anderem die um einen Ton nach unten verschobene Wiederholung der Takte 3 und 4 bei. Am Schluss wird das Thema des ersten Taktes wieder aufgegriffen und in eine Schlussklausel überführt.

Melodie 7:

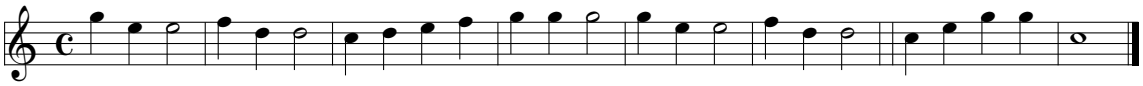


Im Gegensatz zu den Melodien 4 bis 6 ist Melodie 7 im $\frac{3}{4}$-Takt notiert, was ihr einen beschwingten Duktus verleiht. Auffällig ist die konsequente Steigerung, die von g über c bis zum e Spannung aufbaut, die zum Schluss wieder etwas abfällt. Einziger Wermutstropfen: Die Melodie wirkt unfertig und müsste noch weiter fortgesetzt werden.

4.4 Vergleich von Lernverfahren

Der Vergleich verschiedener Lernverfahren ist schwierig, da der gesamte Lernprozess sehr vom Zufall geprägt und das Lernziel nicht klar definiert ist. Es ist kaum möglich, für ein bestimmtes Verfahren eine feste Anzahl von Lernschritten zu ermitteln, nach der das Agentensystem so weit trainiert ist, dass es nur noch Melodien liefert, die dem Benutzer gefallen. Deshalb wurde der Versuch unternommen, den Lernprozess

zu automatisieren, um vergleichbare Ergebnisse zu erhalten. Dabei wird eine Melodie als Lernziel vorgegeben, was eine leichte Modifikation der in 4.3 vorliegenden Zielsetzung darstellt: Hier wird nun nicht eine Klasse von Melodien gelernt, sondern eine ganz bestimmte, was in der Konzeption des Agentensystems ursprünglich nicht vorgesehen war.
Des weiteren wird eine Funktion implementiert, die einen Abstand berechnet, wie weit eine von den Agenten erzeugte Melodie von der zu lernenden Melodie entfernt ist. Dieser Abstand dient dazu, eine automatische Bewertung zu generieren, die umso besser ausfällt, je näher eine Melodie an der Zielmelodie ist.
Als Lernziel wurde die Melodie des Kinderliedes „Hänschen klein“ vorgegeben:

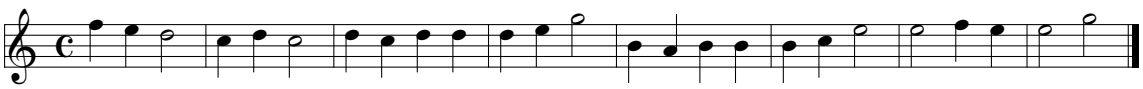


4.4.1 Kreuzen von Agenten

Kapitel 3.8.1 hat die Kreuzung zweier Agenten beschrieben; das soll nun beim automatisierten Lernen Einsatz finden. In diesem Fall wird anfangs eine bestimmte Anzahl von Agenten (jeweils acht Composer und Planer) erzeugt, indem diese zufällig jeweils rund 50 Prozent der Regeln aus dem entsprechenden Korpus erhalten. Immer nach zehn Trainingsschritten werden folgende Aktionen durchgeführt (jeweils für Composer und für Planer):

- Die beiden besten Agenten werden gekreuzt („Combine“).
- Ein neuer Agent wird erzeugt („Create“).
- Die beiden schlechtesten Agenten werden entfernt („Kill“).

Die Gesamtzahl der Agenten bleibt also konstant. Das Erzeugen neuer Agenten soll dazu dienen, dass immer wieder neue Regeln in den Lernprozess eingebracht werden, und somit verhindern, dass sich die Agenten in einem lokalen Maximum festsetzen. Andererseits führt das aber dazu, dass auch nach längerem Lernen immer wieder Melodien komponiert werden, die sich sehr von der Zielmelodie unterscheiden. Dies geschieht vornehmlich dann, wenn ein neuer untrainierter Agent hinzugekommen ist. Insgesamt konvergiert dieses Verfahren schlecht und auch nach mehreren hundert Lernschritten schwanken die Werte der automatisch generierten Bewertungen sehr stark (s. Abbildung 4.1). Trotzdem gelingt es, einige Melodien zu erhalten, die schon relativ nahe an die gewünschte Melodie heranreichen, z. B. die folgende:



Diese Melodie weist schon die korrekte Ton- und Taktart auf, die Anzahl der Takte ist korrekt und der Rhythmus kommt dem erwünschten Ergebnis schon erstaunlich nahe (geringe Abweichungen sind nur in den Takten 5, 7 und 8 zu finden). Die Melodieführung dagegen lässt noch sehr zu wünschen übrig. Allerdings ist dies ein allgemeines Problem, das bei allen Lernverfahren auftreten wird: Schließlich gibt es nur

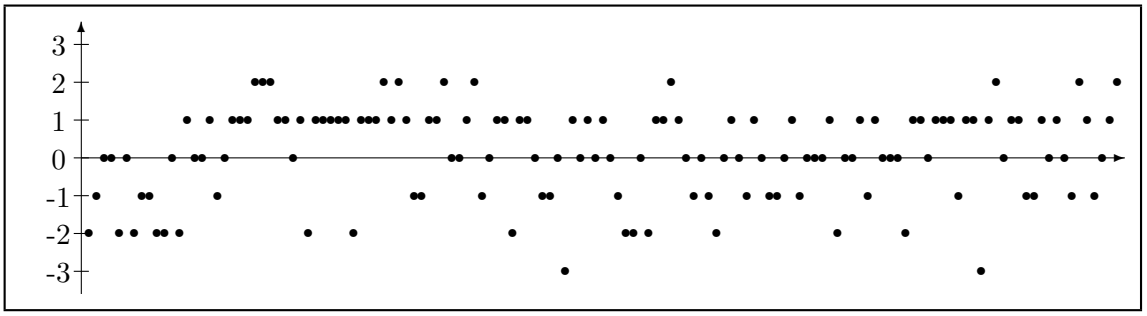


Abbildung 4.1: Entwicklung der Gesamtbewertung für das Verfahren aus 4.4.1

wenige Regeln, die eine bestimmte Tonfolge erzeugen können und es somit ermöglichen würden, eine solche Tonfolge zu lernen. Dazu ist das Setzen der Tonhöhen zu sehr vom Zufall bestimmt. Lediglich die Bedingungen, unter denen dies geschieht, werden gelernt; sie sind aber nicht so stark, dass sie die zu komponierende Melodie auf eine einzige Möglichkeit einschränken.

4.4.2 Agenten mit vollständigem Regelsatz

Das eben vorgestellte Lernverfahren kränkelt vor allem daran, dass neben den durch Kreuzung entstehenden Agenten auch immer wieder Agenten komplett neu erzeugt werden. Lässt man diese Option aber weg und kreuzt nur die schon vorhandenen Agenten, so ist nie sichergestellt, dass die „richtigen“ Regeln überhaupt in irgendeinem Agenten vorhanden sind. Um dem aus dem Weg zu gehen, wird nun folgende Idee realisiert: Alle Agenten werden von Anfang an mit dem kompletten Regelsatz ausgestattet. Dadurch entfällt jeglicher Bedarf an neuen Agenten, da diese nicht mehr oder andere Regeln haben können als die schon von Anfang an vorhandenen. Um das Verfahren zu testen, benötigt man einen Planer und fünf Composer; fünf deshalb, weil eine Komposition des implementierten Agentensystems aus maximal fünf Teilen besteht und für jeden Teil ein Composer nötig ist. Würden alle Teile von ein und demselben Composer ausgeführt, so wäre eine Spezialisierung auf Anfangs- und Schlussteile nicht möglich. Man sollte nun erwarten, dass der Planer nach wenigen Schritten diejenigen Regeln gelernt hat, die die gewünschten Parameter (Tonart usw.) einstellen.
In der Tat ist hier das Konvergenzverhalten besser, da allein schon der „Störfaktor“ der neu erzeugten Agenten entfällt (s. Abbildung 4.2). Doch auch dieses Verfahren kann in einem Nebenmaximum der Bewertungsfunktion landen. Zu dem Zeitpunkt, zu dem die Komposition schon so gut ist, dass sie eine positive Bewertung erhält, erhalten die beteiligten Regeln eine positive Bewertung und können sich gegenüber Regeln, die noch nie angewendet wurden und daher den Punktestand 0 haben, durchsetzen, auch wenn die Komposition noch nicht das Optimum erreicht hat. Genau dieses Problem trat bei den Tests auf, als sich zum Beispiel die Regel `noRhythm` gegenüber der Regel `rhythm_4_1_1_2` behauptete, obwohl die erste bewirkt, dass es

keinen festen Rhythmus in der Komposition gibt, während die zweite den korrekten Grundrhythmus (Viertel – Viertel – Halbe) bewirkt hätte.

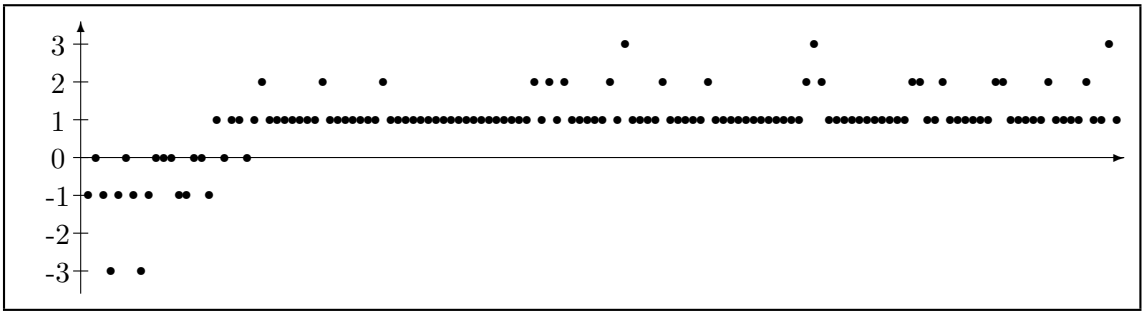


Abbildung 4.2: Entwicklung der Gesamtbewertung für das Verfahren aus 4.4.2

4.4.3 batch-Verfahren

Beim Training von neuronalen Netzen unterscheidet man das on-line- und das batch-Verfahren. Beim on-line-Verfahren werden die Gewichte des neuronalen Netzes nach jedem Trainingsschritt korrigiert, beim batch-Verfahren dagegen wird erst die gesamte Trainingsmenge durchlaufen und dabei die Gewichtsänderungen aufsummiert. Wirksam werden diese erst nach dem gesamten Durchlauf. In Analogie dazu entstand die Idee, eine Art batch-Verfahren für musikalische Agenten zu entwerfen. Dabei werden mehrere Melodien auf einmal komponiert, und zwar jeweils von denselben Agenten. Diese kann der Benutzer anhören und jede Melodie für sich bewerten. Erst dann werden die Bewertungen an die beteiligten Agenten geschickt und von diesen in der in Kapitel 4.2.4 beschriebenen Weise verarbeitet.
Dieses Verfahren hat sich allerdings nicht bewährt. Zu aufwändig und frustrierend ist das Bewerten für den Benutzer, der sich eine Menge von ähnlichen Melodien anhören muss und dann noch entscheiden soll, welche Nuancen die eine Melodie gegenüber den anderen besser oder schlechter macht. Dieses Verfahren wurde kurz nach der Testphase wieder verworfen, so dass hierfür keine Vergleichswerte für das automatisierte Lernen vorliegen. Es ist aber nicht zu erwarten, dass mit dem batch-Verfahren schnellere Konvergenz zu erreichen ist.

4.4.4 Agententurnier

Bei einem Agententurnier treten vier Agenten an, jeweils zwei gegeneinander. Die Verlierer der beiden Zweikämpfe (Vergleich der Punktestände) werden aus dem Agentensystem entfernt, wohingegen sich die beiden Gewinner fortpflanzen dürfen. Das Fortpflanzen funktioniert beim Agententurnier in abgeänderter Form (im Vergleich zu den in 4.4.1 bis 4.4.3 beschriebenen Verfahren), und zwar in Anlehung an die Ausführungen von John R. Koza [11] zur genetischen Programmierung: Wie zwei DNS-Stränge wird das „Erbgut“ (Regelmenge) der beiden Agenten nebeneinander gelegt und gekreuzt (*crossover*), quasi an einer beliebigen Stelle aufgeschnitten und

vertauscht. In der Praxis wird das folgendermaßen realisiert: Die Regeln der zu kreuzenden Agenten A und B werden fortlaufend nummeriert: $r_1, r_2, \ldots$ Dann wird eine Zufallszahl m bestimmt, so dass $0 < m < \min\{R_A, R_B\}$, wobei mit R_X wieder die Regelmenge des Agenten X bezeichnet wird. Den beiden entstehenden Kindern C und D werden nun folgende Regeln zugewiesen:

$$R_C = \{r_i \in R_A \mid \ i \leq m\} \cup \{r_j \in R_B \mid \ j > m\}$$

und

$$R_D = \{r_i \in R_B \mid \ i \leq m\} \cup \{r_j \in R_A \mid \ j > m\}.$$

In Abbildung 4.4.4 sind die Bewertungen der ersten 140 Lernschritte zu sehen, wobei immer nach 10 Schritten ein Turnier durchgeführt wurde. Aus der Abbildung geht hervor, dass das Turnierverfahren weniger zielgerichtet zu guten Bewertungen tendiert als das Verfahren in 4.4.2. Wahrscheinlich ist diese Tatsache auf die Beliebigkeit zurückzuführen, mit der der Schnittpunkt m beim *crossover* bestimmt wird. Es können somit viele Schritte erforderlich sein, bis ein Agent entsteht, der die richtige Kombination von Regeln enthält: „a computer program that solves (or approximately solves) a given problem may emerge from this combination of Darwinian natural selection and genetic operations“ [11].

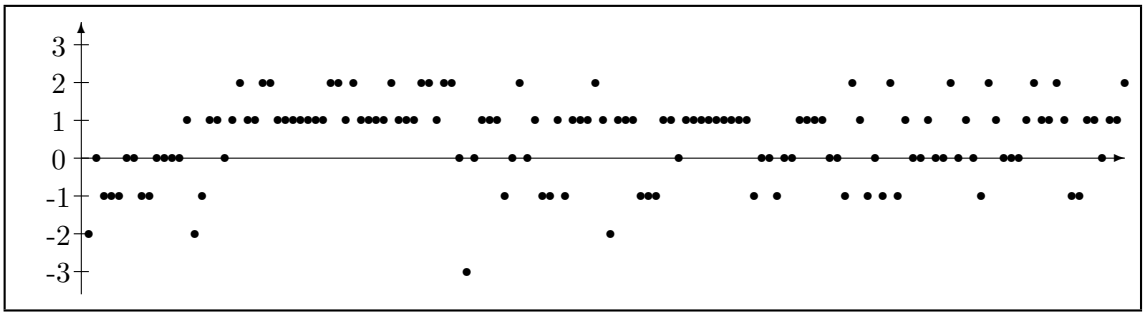


Abbildung 4.3: Entwicklung der Gesamtbewertung für das Verfahren aus 4.4.4

Kapitel 5

Zusammenfassung und Ausblick

Ich denke niemals an die Zukunft. Sie kommt früh genug.
— Albert Einstein

5.1 Zusammenfassung

In der vorliegenden Arbeit wurde ein Multiagentensystem entworfen, das aus musikalischen Agenten besteht, die in der Lage sind, einstimmige Melodien zu komponieren. Darüber hinaus besitzen die Agenten die Fähigkeit, aus Bewertung eines Benutzers zu lernen und sich mit der Zeit mehr oder weniger dessen Musikgeschmack anzupassen.
Kern eines jeden Agenten ist ein Expertensystem, welches Regeln zum Komponieren enthält und dem Agenten seine Lernfähigkeit verleiht, indem jeder Regel eine Priorität zugewiesen wird. Die Prioritäten berechnen sich nach jeder Bewertung neu, so dass „gute“ Regeln in Zukunft öfter verwendet werden als „schlechte“. Damit war es möglich, eine subjektive Verbesserung der komponierten Melodien zu erzielen.
Um objektive Ergebnisse zu erhalten, wurde das Lernen leicht modifiziert und automatisiert. Dabei zeigte es sich, dass genetische Verfahren wesentlich langsamer zum Ziel führten als ein Verfahren, das ohne Reproduktion von Agenten auskam.

5.2 Ausblick

Ein solch komplexes Unterfangen wie das Komponieren von Musik mit dem Computer kann natürlich im Rahmen einer Studienarbeit nicht vollständig abgehandelt werden. Einige Möglichkeiten, die Unvollkommenheit des implementierten Agentensystems zu mindern, werden in den folgenden Abschnitten genannt.

5.2.1 Erweiterungsmöglichkeiten

„Es gibt keine besseren Daten als mehr Daten“: Dieses Bonmot aus dem Bereich der Mustererkennung trifft in gewisser Weise auch in dieser Arbeit zu. Die Gesamtzahl

der implementierten Regeln (für Composer und Planer) ist mit etwa 150 bei weitem nicht ausreichend, um auch nur annähernd das musikalische Wissen unserer Zeit zu beschreiben. Für jeden einzelnen Musikstil wie z. B. Renaissance, Barock, Romantik oder Zwölftonmusik, wären schon tausende von Regeln notwendig, um die zu Grunde liegenden Theorien adäquat abzubilden. Aber auch wenn man nicht so tief in die Materie einsteigen möchte, so gäbe es trotzdem zahlreiche weniger tiefgründige Erweiterungsmöglichkeiten. So wurde bei der Implementierung lediglich eine Regel geschrieben, die den Grundton festlegt; dies ist immer C. Das ist natürlich keine große Einschränkung, denn die entstehenden Melodien kann der Benutzer ja beliebig transponieren. Etwas gewagter, aber nicht unbegründet[1] ist dagegen schon die Festlegung auf die Länge der einzelnen Teile: Sie beträgt immer genau zwei Takte. Diese Entscheidung wurde aufgrund der sonst explosionsartig ansteigenden Anzahl von sich neu ergebenden Melodien getroffen. Die Frage ist, wo man hier eine Grenze setzen soll: Schließlich wäre es kaum sinnvoll, jede rationale Zahl (oder zumindest jedes Vielfache von $\frac{1}{4}$ oder $\frac{1}{8}$) als mögliche Länge für einen Melodieteil zu implementieren.
Je mehr Regeln es dagegen gibt, desto länger wird der Benutzer trainieren müssen, um alle Möglichkeiten auszuschöpfen. Hier musste also ein Kompromiss gefunden werden.

5.2.2 Mehrstimmige Kompositionen

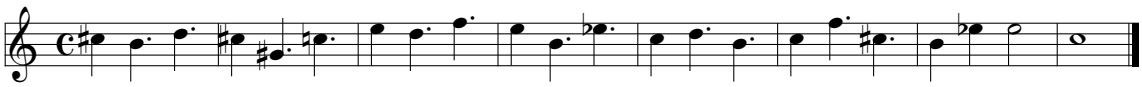
Gerade im Hinblick auf die zu Grunde liegende Agententechnologie wäre es interessant, mehrstimmige Musikstücke komponieren zu lassen. Dabei könnte jeweils ein Agent für eine Stimme beispielsweise in einem vierstimmigen Chorsatz zuständig sein. Er muss dann einerseits versuchen, seine eigenen Interessen durchzusetzen, nämlich dass die Noten im singbaren Bereich der jeweiligen Stimme bleiben und dass die Melodieführung möglichst gut singbar ist, also z. B. keine übermäßigen oder verminderten Intervalle enthält. Andererseits muss er mit den anderen Agenten verhandeln, so dass die Einzelstimmen am Ende zusammenpassen und einen harmonischen Klang ergeben. In einer zusätzlichen Schicht zwischen Planer und Composer könnte ein Harmonieplaner stehen, der den Verlauf der Harmonien und Kadenzen vorgibt. Dieses Betätigungsfeld lässt also auch für zukünftige Forschungen noch jede Menge Freiraum offen.

[1] „Die Taktgruppengliederungen $2+2=4$ und $4+4=8$ [...] beherrschten die Tanzmusik, [...] prägten auch das Volkslied und drangen dann in den Konzertsaal ein.“ [12]

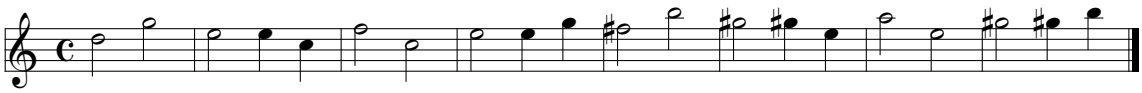
Anhang A

Protokoll einer Lernsitzung

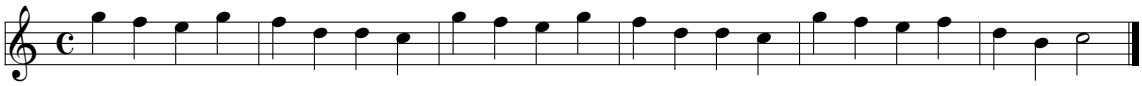
Ein Teil besteht hier immer genau aus zwei Takten.



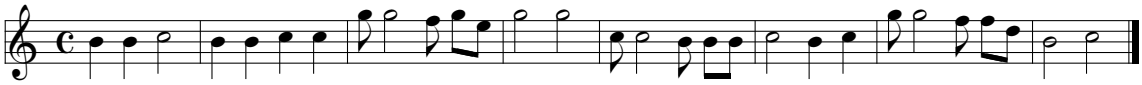
Bewertung: Gesamt: -2, Teil 1: -1, Teil 2: -3, Teil 3: 0, Teil 4: 1



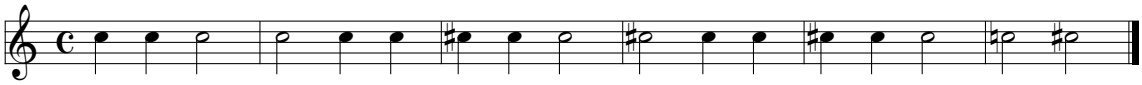
Bewertung: Gesamt: 0, Teil 1: 1, Teil 2: 1, Teil 3: -2, Teil 4: -3



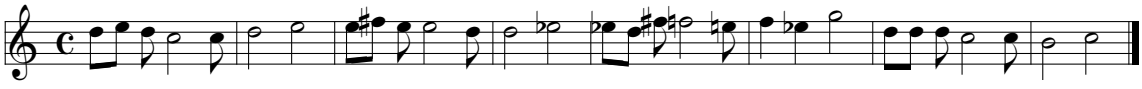
Bewertung: Gesamt: -1, Teil 1: 1, Teil 2: 0, Teil 3: 2



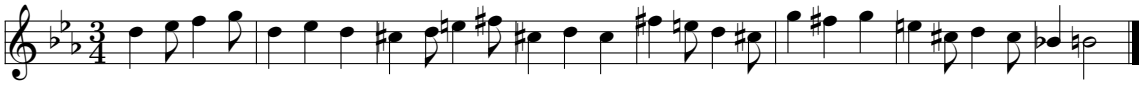
Bewertung: Gesamt: 1, Teil 1: -1, Teil 2: 1, Teil 3: -1, Teil 4: 2



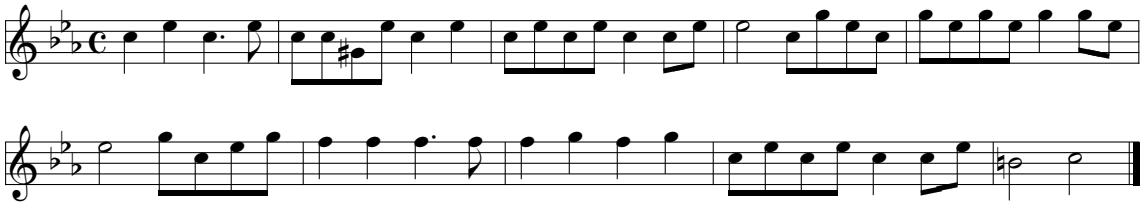
Bewertung: Gesamt: -2, Teil 1: -3, Teil 2: -2, Teil 3: -2



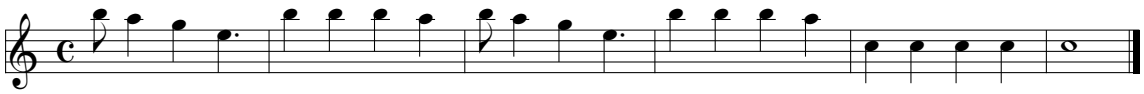
Bewertung: Gesamt: -3, Teil 1: 1, Teil 2: -2, Teil 3: -3, Teil 4: 0



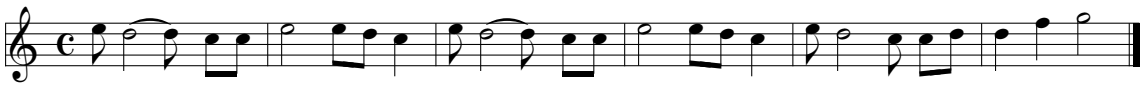
Bewertung: Gesamt: -2, Teil 1: 1, Teil 2: -3, Teil 3: 1, Teil 4: 1



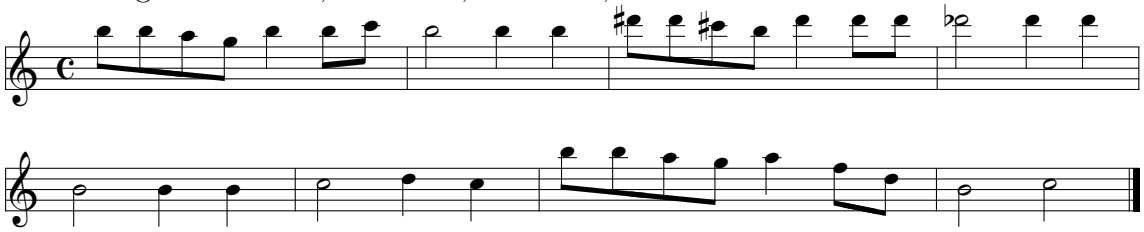
Bewertung: Gesamt: 0, Teil 1: 2, Teil 2: -2, Teil 3: -1, Teil 4: -1, Teil 5: 1



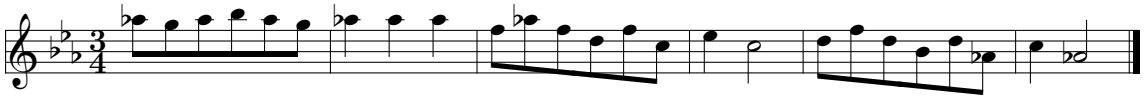
Bewertung: Gesamt: 1, Teil 1: 1, Teil 2: 1, Teil 3: -2



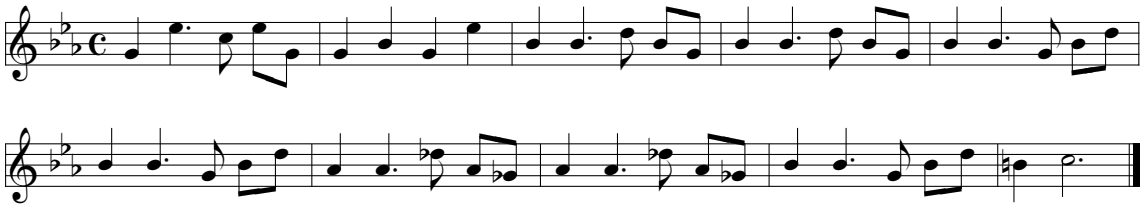
Bewertung: Gesamt: -2, Teil 1: -1, Teil 2: -1, Teil 3: -2



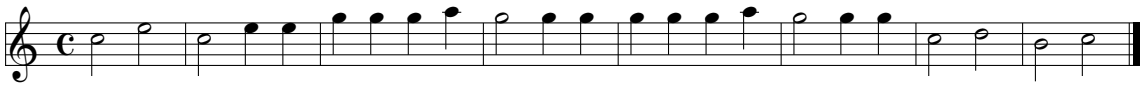
Bewertung: Gesamt: 1, Teil 1: 2, Teil 2: 1, Teil 3: 1, Teil 4: 1



Bewertung: Gesamt: -2, Teil 1: 1, Teil 2: -1, Teil 3: -3



Bewertung: Gesamt: -2, Teil 1: -1, Teil 2: -2, Teil 3: -1, Teil 4: -3, Teil 5: -1



Bewertung: Gesamt: 3, Teil 1: 1, Teil 2: 2, Teil 3: 2, Teil 4: 3

Literaturverzeichnis

[1] Bigus, Joseph und Jennifer: Intelligente Agenten mit Java programmieren. Addison-Wesley Verlag 2001

[2] Brooks, F. P. Jr./ Hopkins, A. L. Jr./ Neumann, P. G./ Wright, W. V.: An Experiment in Musical Composition. IRE Transactions on Electronic Computers 1957

[3] Ferber, Jacques: Multiagentensysteme. Addison-Wesley 2001

[4] Gárdonyi, Zsolt/ Nordhoff, Hubert: Harmonik. Möseler Verlag Wolfenbüttel, 1989

[5] Gill, Stanley: A Technique for the Composition of Music in a Computer. The Computer Journal 1963

[6] Görz, Günther (Hrsg.): Einführung in die künstliche Intelligenz. Addison-Wesley, Bonn, 1995

[7] Hiller, Lejaren/ Isaacson, Leonard: Musical Composition with a High-Speed Digital Computer. Journal of the Audio Engineering Society, 1958

[8] Hügli, Anton/ Lübcke, Poul (Hrsg.): Philosophielexikon. Personen und Begriffe der abendländischen Philosophie von der Antike bis zur Gegenwart. Reinbek Rowohlt, 1991

[9] d'Inverno, Mark/ Luck, Michael: Understanding Agent Systems. Springer-Verlag Berlin Heidelberg 2001

[10] Jeppesen, Knud: Kontrapunkt. Breitkopf & Härtel, Wiesbaden 1965

[11] Koza, John R.: Genetic programming – On the Programming of Computers by Means of Natural Selection, The MIT Press, Cambridge Massachusetts, 1992

[12] de la Motte, Diether: Melodie. Bärenreiter Verlag, Kassel 1993

[13] Puppe, Frank: Einführung in Expertensysteme. Springer-Verlag Berlin Heidelberg, 1988

[14] Rojas, Raùl: Theorie der neuronalen Netze, Springer Verlag Berlin, 1993

[15] Schwanauer, Stephan M./ Levitt, David A. (Hrsg.): Machine Models of Music. The MIT Press, Cambridge, Massachusetts, 1993

[16] Schwende, Harald Manfred: Sonifikation visueller Information zur emotionalen Wahrnehmung durch blinde Menschen. Promotionsarbeit, Erlangen 2003

[17] Simon, Herbert A.: Why Should Machines Learn? In Michalski/ Carbonell/ Mitchell (Hrsg.): Machine Learning – An Artificial Intelligence Approach, Band 1, Kapitel 2, Morgan Kaufmann, Palo Alto, CA, 1983

[18] Stoyan, Herbert: Programmiermethoden der Künstlichen Intelligenz, Band 2, Springer-Verlag Berlin Heidelberg, 1991

[19] Weiss, Gerhard (Hrsg.): Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence. The MIT Press, Cambridge, Massachusetts, 1999