# Ein Dialogsystem für einen sprechenden Schachroboter

Michael Jellinghaus

Universität des Saarlandes
Wintersemester 2002/03

# Inhaltsverzeichnis

# 1 Einleitung

Diese Arbeit ist die Dokumentation eines Dialogsystems für einen sprechenden Schachroboter, der im Rahmen des Softwareprojekts *"Sprechende Roboter mit Lego Mindstorms"*[1] entstanden ist. Wir stellen kurz die Idee des Projekts vor und geben einen Überblick über den Aufbau, die Konstruktion und die Programmierung des Roboters, bevor wir uns im Hauptteil ausführlich mit dem Dialogsystem beschäftigen. Dazu schildern wir im Detail die Funktionsweise unseres ersten Entwurfs sowie die einer verbesserten überarbeiteten Version, wobei wir auf Aspekte des Dialogverlaufs, der Spracherkennung und der Kommunikation des Dialogsystems mit anderen Softwarekomponenten eingehen.

# 2 Die Aufgabe

## 2.1 "Sprechende Roboter mit Lego Mindstorms"

In diesem Projekt ging es darum, aus den Teilen eines *"Lego Mindstorms"*-Baukastens einen interessanten Roboter zu konstruieren und so zu programmieren, dass er mittels gesprochener Sprache mit dem Benutzer kommuniziert und auf dessen Kommandos hin handelt.
Zu diesem Zweck ist der Roboter über eine Infrarotschnittstelle mit einem PC verbunden, auf dem ein Dialogsystem per Spracherkennungssoftware und Sprachsynthese die Kommunikation zwischen Mensch und Maschine kontrolliert. Der Roboter reagiert entsprechend auf diese Kommandos und gibt seinerseits Informationen an das Dialogsystem zurück.

## 2.2 Die Idee

Nachdem die erste Hälfte des Semesters einer Einführung in Dialogsysteme, die Konstruktion von Robotern und deren Programmierung gewidmet war, galt es im zweiten Teil, in Teamarbeit das Konzept für einen eigenen sprechenden Roboter zu entwickeln und umzusetzen. Unser Team, bestehend aus Tatjana Scheffler, Maik Freis und Michael Jellinghaus, hat sich für einen Schachroboter entschieden, der Schachpartien gegen einen menschlichen Spieler bestreitet. Zu diesem Zweck nimmt er gesprochene Zugkommandos entgegen, führt diese auf einem Schachbrett aus und reagiert jeweils mit einem eigenen Zug.

[1] Universität des Saarlandes, Wintersemester 2002/2003, unter der Leitung von Alexander Koller und Geert-Jan Kruijff

### 2.3 Teamarbeit

Für die Realisierung unseres Projektes war gemeinsame Entwicklungsarbeit auf drei Gebieten notwendig: die Konstruktion des eigentlichen Roboters, seine Programmierung sowie der Entwurf des zugrundeliegenden Dialogsystems. Mit der Dokumentation des letzteren beschäftigt sich diese Arbeit, die Beschreibung der Hardware hat Maik Freis übernommen, die Java-Software wird ausführlich von Tatjana Scheffler behandelt.

### 2.4 Die Hardware

Unser Roboter bewegt sich auf Rädern in Längsrichtung unterhalb eines von drei Holzwänden getragenen Schachbretts. Auf ihm sitzt eine Schiene, auf der ein mit einem hochfahrbaren Magneten versehenes Modul quer zur Bewegungsrichtung des Roboters hin- und herfahren kann. Derart kann die Unterseite jedes beliebigen Feldes von dem montierten Magneten erreicht werden, der in seiner ausgefahrenen Position die ebenfalls mit Magneten versehenen Schachfiguren oben auf dem Schachbrett bewegen und sie durch Herunterfahren auch wieder "loslassen" kann.

### 2.5 Die Software

Auf dem RCX, dem programmierbaren Hauptbaustein des Roboters, läuft ein Programm, das bei entsprechenden Kommandos die für eine bestimmte Bewegung notwendigen Motoren ansteuert. Diese Befehle kommen von einem Client-Programm auf dem PC, welches die vom Benutzer gewünschten Züge vom Dialogsystem entgegennimmt, sie nach einigen Umformungen an das kommandozeilengesteuerte Schachprogramm GnuChess weiterleitet und dessen Ausgaben interpretiert. Akzeptiert GnuChess den Zug und liefert seinen Gegenzug, so schickt der Client diese beiden Züge an den RCX, der sich zur Ausführung in Bewegung setzt. Handelt es sich um einen ungültigen Zug, geht diese Information für den Benutzer vom Client direkt an das Dialogsystem zurück, ohne dass der Roboter in Aktion getreten ist.

## 3 Das Dialogsystem

### 3.1 Die Arbeitsmittel

Für die Implementierung der Dialogkomponente unseres Projektes standen uns drei Systeme zur Verfügung: Spracherkennungs- und Sprachsynthesesoftware von Lernout & Hauspie sowie das Tool DiaWiz der clt Sprachtechnologie GmbH. Letzteres bietet eine graphische Oberfläche für die Gestaltung und Umsetzung von Dialogsystemen auf der Basis endlicher Automaten. Aufgabe in diesem Bereich war es, ein Dialogsystem zu spezifizieren sowie eine Grammatik für den Spracherkenner im JSGF[2] zu schreiben.

[2]Java Speech Grammar Format

### 3.2 Der "Dialogue Wizard"

Für den Aufbau des Dialogsystems stellt DiaWiz dem Benutzer neben der Möglichkeit zur Verbindung mit anderen Geräten, bzw. deren Software (in unserem Fall Spracherkenner, Sprachsynthese und "Schach-Client"), auch die Einführung seiner eigenen Variablen sowie eine Reihe von Knotentypen zur Verfügung, deren Ein- und Ausgänge durch Kanten miteinander verbunden werden können und so einen endlichen Automaten bilden. Das Dialogsystem beginnt bei seinem Aufruf im Startknoten und macht solange Transitionen von einem Knoten zum nächsten, bis irgendwann ein Endknoten erreicht ist. Manche Knotentypen besitzen mehrere Ausgänge und ermöglichen so alternative Übergänge zu einem aus einer Menge von Knoten, wobei es von Variablenwerten oder von Eingaben des Benutzers, bzw. des Clients abhängt, an welcher Stelle es weitergeht.

### 3.3 Die Knotentypen im einzelnen

Neben den einfachen Start- und Endknoten mit nur einem Aus-, bzw. nur einem Eingang, gibt es Ausgabeknoten, Eingabeknoten, Variablenknoten, Bedingungsknoten sowie die (in unserem System nicht zum Einsatz gekommenen) Subgraph- und Prozedurknoten.

Ausgabeknoten leiten Daten entweder an die Sprachsynthese oder als Nachricht an den Client weiter. Eingabeknoten haben diese Möglichkeit auch, danach oder gleichzeitig können sie aber zudem eine Eingabe (vom Spracherkenner oder vom Client) lesen, wobei jeder erkannten Eingabe genau ein Ausgang entspricht. Optional hat dieser Typ auch noch einen weiteren Ausgang, der bei einem vom Benutzer in der Länge festgelegten Timeout in Aktion tritt. In Variablenknoten geschieht nichts weiter als eine Zuweisung eines Wertes an eine Variable, was in der bei der Überarbeitung unseres Systems verwendeten Version von DiaWiz dann auch direkt innerhalb von Eingabeknoten möglich ist. Bedingungsknoten schließlich wählen einen ihrer beiden Ausgänge danach aus, ob ein Test erfolgreich ist oder fehlschlägt.

### 3.4 Die Zielsetzung

Beim Design des Dialogsystems haben eine Reihe von Überlegungen eine Rolle gespielt. Zunächst sollte das System in der Lage sein, den Benutzer die von ihm gespielte Farbe auswählen und seine Züge ansagen zu lassen. Er sollte seine Züge in möglichst vielen verschiedenen Formulierungsvarianten angeben und sich dabei eines natürlichen Ausdrucks bedienen können, ohne sich sprachlich dem Computer anpassen zu müssen. Voraussetzung dafür ist die Auflösung von gebräuchlichen Referenzen und Ellipsen durch ihre Integration in die Erkennergrammatik. Zudem sollte für ein gutes Grounding gesorgt werden, indem der User am Ende seiner Spracheingabe nie im Unklaren darüber gelassen wird, ob er richtig verstanden wurde. Falls es doch einmal ein Missverständnis gegeben hat, sollte eine umgehende Richtigstellung möglich sein. Gemäß der eher unspontanen Natur des Schachspiels kann auf eine gemischte Initiative verzichtet werden; es reicht vollkommen, dem Benutzer Fragen zu stellen, die dieser dann beantwortet, wenn er an der Reihe ist ("striktes Turn-Taking") — und der User sollte wirklich nur auf die gestellten Fragen antworten und sich so an unmittelbare Adjazenzpaare halten — die Intelligenzleistung, für eine Äußerung die passende Frage zu finden, mögen *"Jeopardy"*-Kandidaten bewerkstelligen können; unser kleines Dialogsystem jedoch kennt keinerlei Sprünge im Dialogkontext und wäre damit überfordert. Allerdings sollte bei unserem System der Benutzer bei der Aufforderung, seinen nächsten Zug einzugeben, auch den Wunsch äußern dürfen, das Spiel schon vor dem eigentlichen Ende der Partie abzubrechen.

## 3.5 Die Vorgehensweise

Als nächsten Schritt galt es, alles in eine sinnvolle Reihenfolge zu bringen, die Ausgaben des Systems zu formulieren, Daten zu möglichen Benutzereingaben zu sammeln und auf diese passende Reaktionen zu entwerfen.

Da bei konventionellen Schachspielen die Züge entweder unmittelbar mechanisch, durch *"Drag & Drop"* und ähnliche Aktionen mit der Maus oder auf dem Umweg über Kurznotationen (z.B. *"Nh6"* für *"Springer nach h6"* — gebräuchlich in Zeitungs-"Schachecken", Briefpartien oder auch bei GnuChess) durchgeführt werden, gesprochene Kommandos hingegen bislang noch nie Verwendung fanden, existieren keine gängigen Formulierungen für derartige Anweisungen, auf die wir uns bei unserer Arbeit hätten stützen können. So kam es, dass wir die benötigten Daten empirisch sammeln mussten, wobei wir uns angesichts des bescheidenen Zeit- und Mittelrahmens auf Selbstbefragung beschränkt haben — bei einer eventuellen Weiterentwicklung des Systems in der Zukunft wäre es natürlich angebracht, die Erhebung auf unabhängige Versuchspersonen, z.B. durch Wizard-of-Oz-Experimente, auszudehnen, um eine größere Abdeckung zu erzielen.

## 3.6 Der Dialogverlauf

In diesem Abschnitt stellen wir die grundlegende Design-Idee der ersten Version unseres Dialogsystems vor. Beim Start stellt sich das System kurz vor und fragt, mit welcher Farbe der Benutzer spielen möchte. Nach seiner Antwort wird er um die Angabe seines nächsten Zuges gebeten. Das System wiederholt, was es verstanden hat, und bittet um eine Bestätigung durch den Benutzer. Bei einem Missverständnis wird der Benutzer zu einer Wiederholung seiner Eingabe aufgefordert. Andernfalls prüft das System den validierten Zug auf Regelkonformität, führt ihn, falls er erlaubt ist, direkt aus und lässt einen eigenen Zug folgen, woraufhin es wieder mit der Frage nach dem nächsten Zug des Users fortfährt und so weiter. Sollte es zwischendrin zu einer Bedrohung des Königs kommen, wird dem menschlichen Spieler selbstverständlich Schach angesagt. Kommt es gar zu einem Schachmatt, erfolgt eine Meldung über den Verlust der Partie, und das System verabschiedet sich mit *"Auf Wiedersehen"*. Außerdem hat der Benutzer immer die Möglichkeit, anstelle seines nächsten Zuges den Wunsch zu äußern, das Spiel zu beenden (was ihm auch stets gewährt wird).

# 4 Die Umsetzung, 1. Version

Der nun folgende Abschnitt behandelt im einzelnen, wie wir die in Abbildung 1 dargestellte erste Version unseres Dialogsystems aufgebaut haben, um den zuvor präsentierten Dialogverlauf zu erzielen. Die involvierten Knoten werden einer nach dem anderen beschrieben, wobei wir sowohl auf unsere Überlegungen bei der Umsetzung eingehen als auch auf die resultierenden Ein- und Ausgaben der Knoten, ihre Verbindung miteinander und ihr Zusammenspiel mit Erkennergrammatik und/oder Client.

## 4.1 Das System stellt sich vor

Unmittelbar auf den Startknoten folgt die Selbstvorstellung des Systems, die vor allem der Höflichkeit dienen soll und deswegen nicht zu lang geraten darf. Erfahrungsgemäß wirkt es gerade bei mehrmaligem Benutzen einer Software schnell störend, wenn die Einleitungssequenz zu viel Zeit beansprucht. Monologe hemmen die Lust an der Interaktion und vermitteln den Eindruck, es handele sich um etwas Kompliziertes, was viele Interessenten, die das System zunächst nur ausprobieren möchten (z.B. in Messesituationen), abschrecken könnte. Aus denselben Gründen haben wir auf eine "Gebrauchsanleitung" des Systems ebenso verzichtet wie auf eine Erläuterung der Schachregeln und haben eine Anredeform gewählt, die bei einem möglichst breiten Publikum Akzeptanz findet. Das System sagt also einfach:

*Hallo, ich bin der Schachroboter.*

## 4.2 Die Farbauswahl

### 4.2.1 "Welche Farbe möchten Sie spielen?"

...das wird der User als nächstes gefragt, denn vor jeder Schachpartie steht das Problem, sich zu einigen, wer anfangen darf (oder muss). Da wir ein höfliches System entwickeln wollten, bleibt bei uns diese Entscheidung komplett dem Benutzer überlassen. Zudem haben wir die in ersten Beta-Versionen noch verwendete Anredeform *"Du"* bald durch das Siezen ersetzt, um nicht den falschen Eindruck zu erwecken, unser Roboter sei lediglich ein Spielzeug für Kinder.

### 4.2.2 Farbe bekennen — Benutzereingaben

Wie allseits bekannt, hat man beim Schach die Wahl zwischen den Farben Schwarz und Weiß. Die Schwierigkeit bei der Spracherkennung bestand also weniger darin, zahlreiche verschiedene Farbnamen voneinander zu unterscheiden, als vielmehr eine Verwechslung mit anderen Schlüsselwörtern zu verhindern. Dies kann unter anderem durch eine Verlängerung des zu erkennenden Ausdrucks erreicht werden. So ist es zwar durchaus nicht unwahrscheinlich, dass das Wort *"weiß"* einmal versehentlich als das Wort *"nein"* erkannt wird, bei der Wortkette *"Ich möchte gerne mit Weiß spielen"* besteht diese Gefahr jedoch nicht mehr — es kommt zu weniger Fehlerkennungen. Hier ein paar

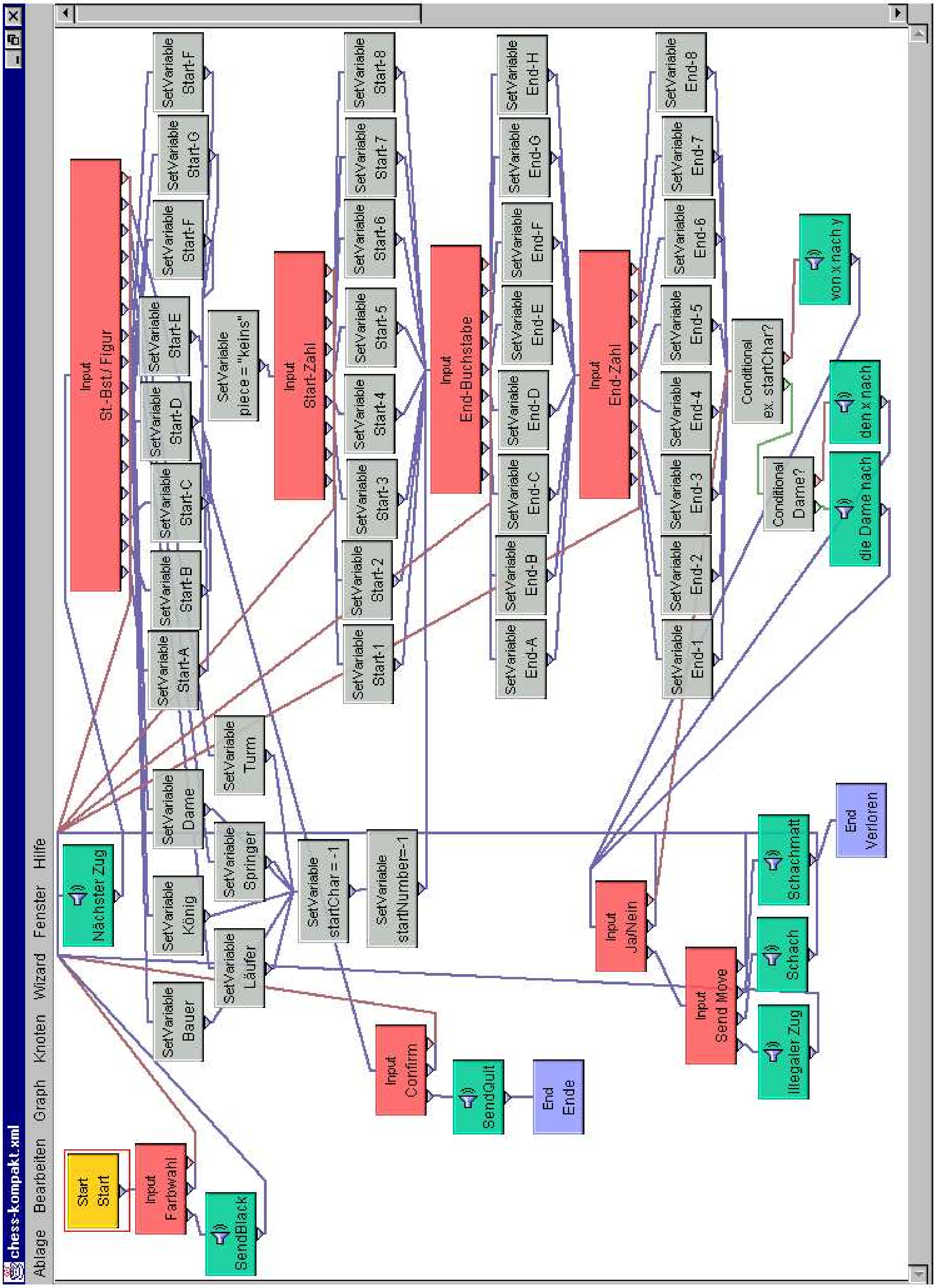


Abbildung 1: Die 1. Version des Dialogsystems

von uns gesammelte Beispiele aus der Fülle der vielen Möglichkeiten, den Wunsch, mit Schwarz zu spielen, auszudrücken:

*Schwarz.*
*Ich möchte Schwarz spielen.*
*Ich will Schwarz.*
*Ich will bitte mit Schwarz spielen.*
*Ich möchte gerne Schwarz.*

Um nicht jede dieser Varianten ausbuchstabieren zu müssen und gleichzeitig keine zu verlieren, haben wir obiges in unserer Erkennergrammatik im folgenden Eintrag zusammengefasst (eckige Klammern drücken Optionalität aus, der senkrechte Strich steht für Alternativen):

```
[ich (möchte|will)] [gerne|bitte] [mit] schwarz [spielen]
```

### 4.2.3 Die "Farbreaktion" des Systems — Auswertung der Eingabe

Wird der obige Ausdruck von der Spracherkennungssoftware erkannt, sendet die Erkennergrammatik das Schlüsselwort `schwarz` an das Dialogsystem. Ein parallel aufgebauter Eintrag tut dasselbe für das Wort `weiss`. Da die von uns verwendete Schachsoftware GnuChess dem menschlichen Spieler per Default Weiß zuweist, kann es bei der entsprechenden Wahl direkt mit dem Aus- und Eingabeknoten weitergehen, der nach dem nächsten Zug fragt.

Entscheidet sich der Benutzer allerdings für Schwarz, so ist ein Ausgabeknoten zwischengeschaltet, der dem Client eine Nachricht vom Typ `BLACK` schickt, welche bewirkt, dass das entsprechende Kommando weiter an GnuChess gesendet wird.

Nach den internationalen Schachregeln beginnt das Spiel stets mit einem Zug des weißen Spielers, weswegen nun in demselben Knoten die Nachricht

*Ich führe jetzt den ersten Zug durch. Bitte kurz Geduld.*

per Sprachsynthese an den Benutzer geschickt wird, während GnuChess seinen Eröffnungszug berechnet und der Roboter diesen ausführt. Danach geht es — wie bei *"weiß"* auch — mit der Frage nach dem nächsten Zug weiter. Auch in dem Fall, dass der Benutzer nach der Farbfrage 30 Sekunden lang weder *"schwarz"* noch *"weiß"* sagt, bzw. falls der Spracherkenner keinen der beiden Ausdrücke während dieser Zeit erkennt, springt das System in den folgenden Knoten, d.h. ein Timeout wird genauso behandelt, als hätte der User sich für Weiß entschieden.

### 4.3 Der nächste Zug

#### 4.3.1 Das System lässt bitten — Eingabeaufforderung

Nach der erfolgten Farbauswahl fordert das System den Benutzer auf, einen Zug anzugeben, den der Roboter für ihn ausführen soll. Auch an dieser Stelle haben wir uns für eine der Situation angemessen kurze und prägnante, aber dennoch freundliche Formulierung entschieden, die den Spielfluss möglichst wenig stören sollte:

*Ihren nächsten Zug, bitte!*

Die Idee, für den Eröffnungszug eine abweichende Aufforderung zu verwenden, da die Frage nach dem *nächsten* Zug ja einen vorangegangenen implizieren könnte, wurde von uns als zu kleinlich schnell wieder verworfen, zumal von einem computergesteuerten System eine gewisse Unflexibilität geradezu erwartet wird — der menschliche Benutzer würde von unterschiedlichen Formulierungen eventuell auf das Vorliegen unterschiedlicher Situationen und damit auch unterschiedlicher Eingabemöglichkeiten für ihn schließen und wäre zu sehr verunsichert.

#### 4.3.2 Wie man Bauern rumkommandiert — Benutzereingaben

Beim Eröffnungszug wie auch bei jedem folgenden Zug steht der Benutzer nun vor der Aufgabe, anzugeben, welche Figur auf welches Feld gesetzt werden soll. Hier einige von unseren Versuchspersonen (also uns) gewählte Formulierungen für ein und denselben Zug:

*Bauer nach d5.*
*Von d7 nach d5.*
*d7 — d5.*
*Dora 7 nach Dora 5.*
*Von d wie Dora 7 nach d wie Dora 5.*
*Zieh bitte den Bauer nach d5.*
*d7 auf d5, bitte.*
*Den rechten mittleren Bauern bitte zwei vor.*
*Ich spiele dieselbe Eröffnung wie immer.*

Um es gleich vorneweg zu sagen, ist es uns spätestens in der zweiten Version gelungen, alle diese Eingaben zu erkennen — mit Ausnahme der letzten beiden. Für den vorletzten wäre eine komplexere Behandlung von Referenzen vonnöten gewesen, als es uns in der begrenzten Zeit möglich gewesen wäre, für *"dieselbe Eröffnung wie immer"* bräuchte man ein System, das Benutzer identifizieren und sich deren Züge merken kann — beides Erweiterungen, die unserer Meinung nach nicht zur engeren Funktionalität eines Schachroboters gehören.

### 4.3.3 Was zu einem guten Schachzug gehört — Desambiguierung

Damit der von uns programmierte Client einen eindeutigen Zug an GnuChess schicken kann, benötigt er zwei Informationen: welche Figur bewegt und auf welches Feld sie gestellt werden soll. Letzteres erfolgt, wie die Beispielsätze zeigen, fast immer durch Angabe der Koordinaten, ersteres allerdings manchmal auch durch Nennung des Figurtyps (z.B. *"Bauer"*) , wenn nicht ebenfalls Buchstabe-Zahl-Kombinationen Verwendung finden.

### 4.3.4 Die Dialogvariablen

Zur Speicherung der Angaben haben wir fünf Variablen eingeführt: `piece` mit einem Wert vom Typ `String` für den Figurtyp, die Integervariablen `startChar` und `startNumber` für die Koordinaten des Feldes, auf dem der Zug beginnt, sowie `endChar` und `endNumber` entsprechend für das Zielfeld, ebenfalls beide vom Typ `Integer`.

### 4.3.5 Zuviel auf einmal — Erkennerprobleme

Die Werte, die den Variablen zugewiesen werden sollen, tauchen in den Eingabesätzen meist direkt aufeinanderfolgend auf: Bei *"d7"* entspricht *"d"* z.B. dem Wert der Variablen `startChar` und *"7"* dem Wert der Variablen `startNumber`. Die Erkennergrammatik müsste also bei Erkennen der Phrase *"d7"* zwei Signalwörter an das Dialogsystem weitergeben, was in der von uns zuerst verwendeten Version noch nicht möglich war — einer erkannten Phrase konnte nur ein Signalwort zugeordnet werden. Zudem ist es leider nicht zuverlässig möglich, an der üblicherweise nur sehr kurzen Pause zwischen *"d"* und *"7"* den Beginn einer neuen Phrase festzumachen, um beide getrennt zu analysieren. Würde man als Reaktion darauf nun eben der Phrase *"d7"* das Signalwort `d7` zuweisen, so käme man bereits bei einem einzigen Koordinatenpaar auf 64 mögliche Phrasen/Signalwörter und bei einer Analyse ganzer Ausdrücke wie *"von d7 nach d5"* dank einer kombinatorischen Explosion gar auf 4096 im Dialogsystem einzufügende Knoten. Um diesem Dilemma zu entgehen, sahen wir uns bei der hier beschriebenen ersten Version unseres Dialogsystems gezwungen, den Benutzer anzuweisen, kurze Sprechpausen zwischen den einzelnen Teilinformationen zu machen. Dank Verbesserungen in der uns zur Verfügung gestellten Software konnten wir dieses Problem später glücklicherweise in der in Abschnitt 5 beschriebenen zweiten Version beheben.

### 4.3.6 Ein kurzes Vergnügen — "Programm verlassen"

Nach seinem nächsten Zug gefragt, kann der Benutzer nun zunächst entweder die Startkoordinaten eingeben oder den Namen der gewünschten Figur äußern. Außerdem hat er an dieser Stelle die Möglichkeit, das Spiel zu beenden. Für diese letzte Alternative haben wir u.a. folgende Sprachdaten gesammelt:

*Ende.*
*Ich möchte aufhören.*
*Ich habe keine Lust mehr.*
*Schluss.*
*Programm verlassen.*
*Mir reicht's.*
*Ich will nicht mehr.*

All diesen Formulierungen — zusammengefasst als

```
schluss |
ende |
ich (will|möchte) (nicht mehr|aufhören) |
[ich habe] keine Lust [mehr] |
mir reicht's |
programm verlassen
```

weist die Erkennergrammatik das Signalwort `schluss` zu und sendet es an das Dialogsystem, welches daraufhin in einen Knoten übergeht, der beim Benutzer noch ein letztes Mal nachfragt, um so auch ganz sicherzugehen, dass es sich nicht doch um ein Missverständnis oder einen schlechten Scherz handelt:

*Möchten Sie das Spiel wirklich beenden?*

Falls sich der Spieler jetzt durch Äußern von

```
(nein|nö|nee) [stimmt nicht]
```

rettet, sendet die Erkennergrammatik das Signalwort `nein`, welches veranlasst, dass er als nächstes wieder die Aufforderung *"Ihren nächsten Zug, bitte!"* hört und das Spiel fortsetzen kann. Wenn er dagegen

```
(ja|jo|jawoll)
```

sagt, ist das Ende nicht mehr aufzuhalten. Das Signalwort `ja` kommt im Dialogsystem an, dem Client wird eine Nachricht vom Typ `quit` geschickt, und bevor das Dialogsystem schließlich in einem Endknoten angelangt, bekommt der Benutzer noch ein freundlich gemeintes

*Tschüss, bis zum nächsten Mal!*

zu hören in der Hoffnung, dass es ein nächstes Mal geben wird.

### 4.3.7 Die Dame zuerst... — Figurtyp nennen

Der ernsthaft an unserem Schachroboter interessierte User allerdings sucht sein Heil nicht in der Flucht, sondern spielt jede Partie bis zum Schachmatt. Auf jeden Fall aber wird er zumindest einmal seinen nächsten Zug ansagen, und wie wir uns erinnern, hat er dazu im ersten Eingabeknoten zwei Möglichkeiten: die Angabe der Startkoordinaten oder des Figurtyps. Sagt er z.B. *"Bauer"*, wird das Signalwort `bauer` von der Erkennergrammatik ans Dialogsystem übermittelt, dessen als nächstes besuchter Knoten die Variable `piece` auf den Wert `bauer` setzt. Entsprechende Alternativen gibt es für alle sechs Figurtypen:

```
bauer {bauer}
(könig|könich|könisch) {koenig}
dame {dame}
läufer {laeufer}
springer {springer}
turm {turm}
```

Hier steht das erkannte Wort jeweils am Anfang (mit durch in der deutschen Sprache vorkommende Aussprachevarianten bedingten orthographischen Alternativen für das Wort *"König"*) und das ans Dialogsystem gesendete Signalwort, das auch dem der Variablen `piece` zugewiesenen Wert entspricht, am Ende in geschweiften Klammern.

Nach der Bindung von `piece` an den jeweiligen Figurtyp — es gibt hierzu sechs parallelgeschaltete Knoten — werden noch `startChar` und `startNumber` auf $-1$ gesetzt, was sicherstellt, dass nicht versehentlich ein Wert vom letzten Durchlauf weitergegeben wird, und zudem bei der Formulierung der Bitte um Bestätigung des Zuges eine wichtige Rolle spielt (s. Abschnitt 4.4). Danach kann es auch schon mit der Eingabe der Koordinaten des Zielfeldes weitergehen.

### 4.3.8 Heinrich VIII. — Buchstabe-Zahl-Kombinationen

Neben der Angabe des Figurtyps als erste Information ist es auch möglich, die Koordinaten des Startfeldes zu nennen. Für diesen Fall haben unsere Daten gezeigt, dass hier im allgemeinen Sprachgebrauch durchweg mit dem Buchstaben begonnen wird. Es reicht also, im diskutierten Eingabeknoten — neben den Schlüsselwörtern fürs Spielende und die sechs Figurtypen — die acht für das Schachspiel relevanten Buchstaben erkennen zu können, womit wir auf insgesamt 15 Ausgänge kommen, zu denen sich als letztes noch ein Timeout nach einer Minute gesellt. Nach Ablauf dieser Zeit wird die Frage wiederholt und erneut auf eine Eingabe gewartet.

**Könnten Sie das bitte buchstabieren?** Schon bei nur acht Buchstaben stellt sich ein großes Problem, das nicht nur Spracherkennungssoftware, sondern auch menschliche Hörer leicht zum Verzweifeln bringen kann: *"b"*, *"c"*, *"d"*, *"e"* und *"g"* oder auch *"a"* und *"h"* klingen so ähnlich, dass es regelmäßig zu Verwechslungen kommt. Wir haben deswegen auf die u.a. im Funkverkehr bewährte Lösung des Buchstabieralphabets zurückgegriffen, bei dem jedem Zeichen ein mit ihm beginnendes Wort zur Desambiguierung zugeordnet wurde, denn mit *"a wie Anton"* und *"h wie Heinrich"* sollten Missverständnisse so gut wie unmöglich werden. Hier die von uns verwendeten Phrasen zur Erkennung der einzelnen Buchstaben (mit einigen orthographischen Varianten, die sich als hilfreich erwiesen haben):

```
[a wie] anton {anton}
[bee wie] berta {berta}
[zeh wie] (cäsar|zäsar|zesar|zesa|zeser|zäser) {caesar}
[dee wie] dora {dora}
[e wie] emil {emil}
[eff wie] friedrich {friedrich}
[geh wie] (gustav|gustaf) {gustav}
[ha wie] heinrich {heinrich}
```

Es ist also nötig, den jeweiligen Namen zu sagen, der Zusatz "`<Buchstabe>` *wie*" kann weggelassen werden. Genau wie der Variablen `piece` bei der Nennung der Figurtypen wird in den acht Knoten für die Startbuchstaben `startChar` ein der Eingabe entsprechender Integerwert zwischen 1 und 8 zugewiesen.

**Nennen Sie eine Zahl, irgendeine Zahl...** In einem der Buchstabenerkennung nachgeschalteten Knoten wird `piece` auf den Wert `keins` gesetzt, und der nächste Eingabeknoten zur Erkennung der Zahl-Koordinate des Startfeldes wird aktiviert. Dort läuft alles ganz ähnlich ab, die in der folgenden Übersicht aufgeführten Ausdrücke bewirken eine Belegung von `startNumber` mit einer der jeweiligen Ziffer entsprechenden `Integer` von 1 bis 8, wobei die Formen *"zwo"* und *"fünneff"* die Fehlerrate des Erkenners zu reduzieren helfen.

```
eins {eins}
zwo {zwo}
drei {drei}
vier {vier}
(fünf|fünneff) {fuenf}
sechs {sechs}
sieben {sieben}
acht {acht}
```

### 4.3.9 Endlich wiedervereint — die Zielkoordinaten

Anschließend treffen sich die beiden Läufe wieder, die sich an der Entscheidung, entweder Figurtyp oder Startkoordinate zu nennen, getrennt hatten, um nun die Eingabe der Zielkoordinaten in derselben Reihenfolge wie eben (zuerst der Buchstabe, dann die Zahl) entgegenzunehmen. Die vom Spracherkenner akzeptierten Phrasen wie auch der Vorgang der Variablenbindung sind identisch mit denen bei den Startkoordinaten

— mit dem einzigen Unterschied, dass die Variablen jetzt logischerweise `endChar` und `endNumber` heißen. Hier wie auch in den vorangegangenen Knoten gilt ein Zeitlimit für die erfolgreiche Spracherkennung von jeweils einer Minute, wonach — egal, welche Angaben bereits gemacht worden sind — nach der üblichen Aufforderung (*"Ihren nächsten Zug, bitte!"*) mit der Eingabe des gesamten Zuges von vorne begonnen werden muss.

## 4.4 "Stimmt das so?" — die Rückfrage

Ist die Eingabe des Zuges abgeschlossen, erfolgt eine Wiederholung der vom Spracherkenner gesammelten Informationen, und der Benutzer wird aufgefordert, diese zu bestätigen oder gegebenenfalls zu korrigieren. Um hier die richtige Formulierung für die Sprachsynthese zu finden, wird im Dialogsystem an dieser Stelle nun zunächst geprüft, ob Figurtyp oder Startkoordinaten spezifiert worden sind. Dies erfolgt durch eine Abfrage, ob die Variable `startChar` einen sinnvollen oder den bei Nennung eines Figurtyps als Platzhalter eingetragenen Wert $-1$ trägt:

```
startChar == -1
```

### 4.4.1 Tricksen, damit die Zahlen stimmen — Stringkonstruktion

Falls sich der Benutzer sowohl beim Start- wie auch beim Zielfeld für eine Eingabe in Form von Koordinatenpaaren entschieden hatte, hört er als nächstes z.B. die Frage

*Sie möchten also von d7 nach d5 ziehen. Stimmt das so?*

Da die Variablen `startChar`, `startNumber`, `endChar` und `endNumber` vom Typ `Integer` sind, für die Sprachausgabe jedoch ein `String` an die Synthese geschickt werden muss, benötigen wir an dieser Stelle einen kleinen Trick, um die entsprechenden Zeichenketten zu erhalten. Der einfachere Fall liegt bei den beiden Zahlen vor — sie werden einfach vermittels der Typkonversionsfunktion `str` umgewandelt. So wird beispielsweise aus der `Integer` 7 der `String` "7". Schwieriger gebärdet sich die Angelegenheit für die Start- und Endbuchstaben. Die hier hilfreiche Funktion heißt `substring`. Sie liefert bei Eingabe einer Zeichenkette als erstem Parameter wieder einen Teil dieser Zeichenkette zurück, und zwar beginnend an der vom zweiten Parameter bezeichneten Position und von der Länge, die im dritten Parameter angegeben ist. Als Ausgangsstring wählen wir den für uns relevanten Anfangsabschnitt des Alphabets mit einer voranstehenden Null: "`0ABCDEFGH`", als zweiten Parameter die Nummer des vom Benutzer genannten Buchstaben und als dritten den Wert 1, da uns nur einzelne Zeichen interessieren. So wird aus

```
startChar = 4
```

und

```
substring("0ABCDEFGH", startChar, 1)
```

wieder der gewünschte `String` ‘‘D’’. Zusammengesetzt sieht die komplette Ausgabe dann folgendermaßen aus (durch + werden die einzelnen Strings miteinander verkettet):

```
"Sie möchten also von " + substring("0ABCDEFGH", startChar, 1) +
 str(startNumber) + " nach " + substring("0ABCDEFGH", endChar, 1) +
 str(endNumber) + " ziehen. Stimmt das so?"
```

### 4.4.2 Aus Rücksicht auf die Dame — Fallunterscheidung beim Artikel

Hatte der eingangs stehende Test ergeben, dass statt der Startkoordinaten ein Figurtyp gesagt wurde, sieht die Sache wieder anders aus. Die Variable `piece` ist bereits vom Typ `String`, sodass es genügen würde, folgendes zu fragen:

```
"Sie möchten also den " + piece + " nach " + substring("0ABCDEFGH",
 endChar, 1) + str(endNumber) + " ziehen. Stimmt das so?"
```

... tja, wenn es im Deutschen nicht so etwas wie ein grammatisches Geschlecht geben würde. Dies zwingt uns, einen weiteren Entscheidungsknoten mit folgender Bedingung voranzuschalten:

```
piece == "dame"
```

Schlägt dieser Test fehl, können wir bei der obenstehenden Formulierung bleiben, andernfalls sieht das Resultat, das sich lediglich in der Wahl des definiten Artikels unterscheidet, so aus:

```
"Sie möchten also die " + piece + " nach " + substring("0ABCDEFGH",
 endChar, 1) + str(endNumber) + " ziehen. Stimmt das so?"
```

## 4.5 Die Stunde der Wahrheit — Zugbestätigung

Die Antwort auf alle drei Fragevarianten lautet entweder

```
(ja|jo|jawoll)
```

oder

```
(nein|nö|nee) [stimmt nicht]
```

Bekommt das Dialogsystem das Signalwort `nein`, oder zögert der Benutzer länger als 20 Sekunden, wird der gesamte Zug verworfen, und die Aufforderung *“Ihren nächsten Zug, bitte!”* wiederholt, woraufhin alles wie oben beschrieben von neuem beginnt. Bei einer affirmativen Antwort hingegen fasst der nächste Knoten den bestätigten Zug als Datenpaket der folgenden Form zusammen:

```
[piece=piece,type="MOVE",fromY=startNumber,toY=endNumber,
 fromX=startChar,toX=endChar]
```

### 4.6 Antwort von GnuChess — Zugevaluation

Das ganze wird dann an den Client geschickt, welcher den Zug zur Evaluierung an GnuChess weiterleitet und dessen Antwort wiederum an das Dialogsystem in Form der Signalwörter `OK, Illegal, Check` oder `Checkmate` durchreicht.

#### 4.6.1 Der Normalfall — "OK"

Handelt es sich um einen regelkonformen Zug, der regulär ausgeführt werden konnte, lautet das Signal `OK`. Der Roboter setzt sich in Bewegung, um ihn und den Antwortzug von GnuChess auszuführen, und das Dialogsystem landet anschließend wieder im oben beschriebenen Knoten mit der Frage nach dem nächsten Zug.

#### 4.6.2 Schummeln unmöglich — "Illegal"

Das Signalwort `Illegal` wiederum wird zurückgegeben, wenn der vom Benutzer gewünschte Zug aus irgendeinem Grund nicht durchgeführt werden kann — sei es, weil versucht wurde, eine Figur nicht gemäß der jedem Figurtyp eigenen Bewegungsmuster zu ziehen; weil das Zielfeld beispielsweise bereits von einer eigenen Figur besetzt ist; weil aufgrund einer "Schach"-Situation der König vor der genannten Figur bewegt werden muss oder auch weil entweder als Startfeld der Bewegung ein Feld ausgewählt wurde, auf dem sich gar keine eigene Figur befindet, oder ein Figurtyp eingegeben wurde, der auf dem Schachbrett gar nicht mehr vorkommt. In jedem dieser Fälle erfolgt die Ausgabe:

*Dieser Zug ist leider nicht möglich.*

Zudem wird die Eingabe des Spielers verworfen, und er wird erneut aufgefordert, seinen nächsten Zug anzusagen.

#### 4.6.3 Schach dem König — "Check"

Auch bei Übermittlung des Signalworts `Check` geht es für den Benutzer mit der Bewegung der nächsten Figur weiter, doch bedeutet dies, dass zuvor sein Zug ausgeführt wurde und der Antwortzug von GnuChess ihn in eine Spielsituation gezwungen hat, in der sein König gefährdet ist und somit das Ende der Partie droht. Um auf die Notlage aufmerksam zu machen, ertönt aus dem Lautsprecher umgehend die Warnung:

*Schach!*

### 4.6.4 Das tragische Schicksal des Gari K. — "Checkmate"

Ist es dem Spieler einmal nach der Ansage von Schach im folgenden Zug nicht gelungen, sich zu retten, ist das Spiel für ihn leider vorbei, denn GnuChess schlägt den König, wann immer sich ihm eine Gelegenheit dazu bietet. Der Roboter vermeldet dem hoffentlich fairen Verlierer nur noch

*Schachmatt! Sie haben leider verloren. Auf Wiedersehen.*

und dann geht das Dialogsystem in einen Endzustand über, aus dem es der Benutzer auch durch Quengeln nicht mehr hervorlocken kann.

## 5 Die Rückkehr des Schachroboters — Umsetzung, 2. Version

Ermöglicht u.a. durch einige Verbesserungen in der DiaWiz-Software und mit dem Ziel, den Roboter für den Auftritt auf der CeBIT 2003 noch robuster zu machen, wurde unser Dialogsystem komplett überarbeitet. Das Result ist in Abbildung 2 zu sehen. Die wohl grundlegendste Änderung in der zweiten Version hat im Umstellen der Erkennergrammatik auf die Übermittlung von Merkmalsstrukturen anstelle einfacher Signalwörter stattgefunden. Dies ermöglicht dem Benutzer vor allem bei der Eingabe seines Zuges eine weitaus natürlichere Ausdrucksweise und führt zudem zu einer signifikanten Verschlankung des gesamten Dialogsystems. Weitere Neuerungen umfassen die Einbeziehung des zugegebenermaßen seltenen Falles, dass der menschliche Spieler die Partie für sich entscheiden kann, sowie die des speziellen Spielzugs der kurzen oder langen Rochade.

### 5.1 Ein AVM für alle Fälle — Merkmalsstrukturen

Die neu eingeführten Merkmalsstrukturen bestehen aus Attribut-Wert-Paaren, über die die folgende Tabelle eine kurze Übersicht gibt.

| Attributname | mögliche Werte |
|---:|---|
| figur | bauer, koenig, dame, laeufer, springer, turm |
| vonbst | a, b, c, d, e, f, g, h |
| vonzahl | eins, zwo, drei, vier, fuenf, sechs, sieben, acht |
| nachbst | a, b, c, d, e, f, g, h |
| nachzahl | eins, zwo, drei, vier, fuenf, sechs, sieben, acht |
| specialmove | krochade, lrochade |
| special | schluss, ja, nein, schwarz, weiss |

### 5.2 "Ich möchte gerne Grün spielen." — Farbwahl II

Bei der Wahl der Spielerfarbe hat sich eigentlich nichts verändert, wenn man einmal davon absieht, dass die Signalwörter `schwarz` oder `weiss` jetzt gemeinsam mit ihrem Attributnamen `special` übermittelt werden.

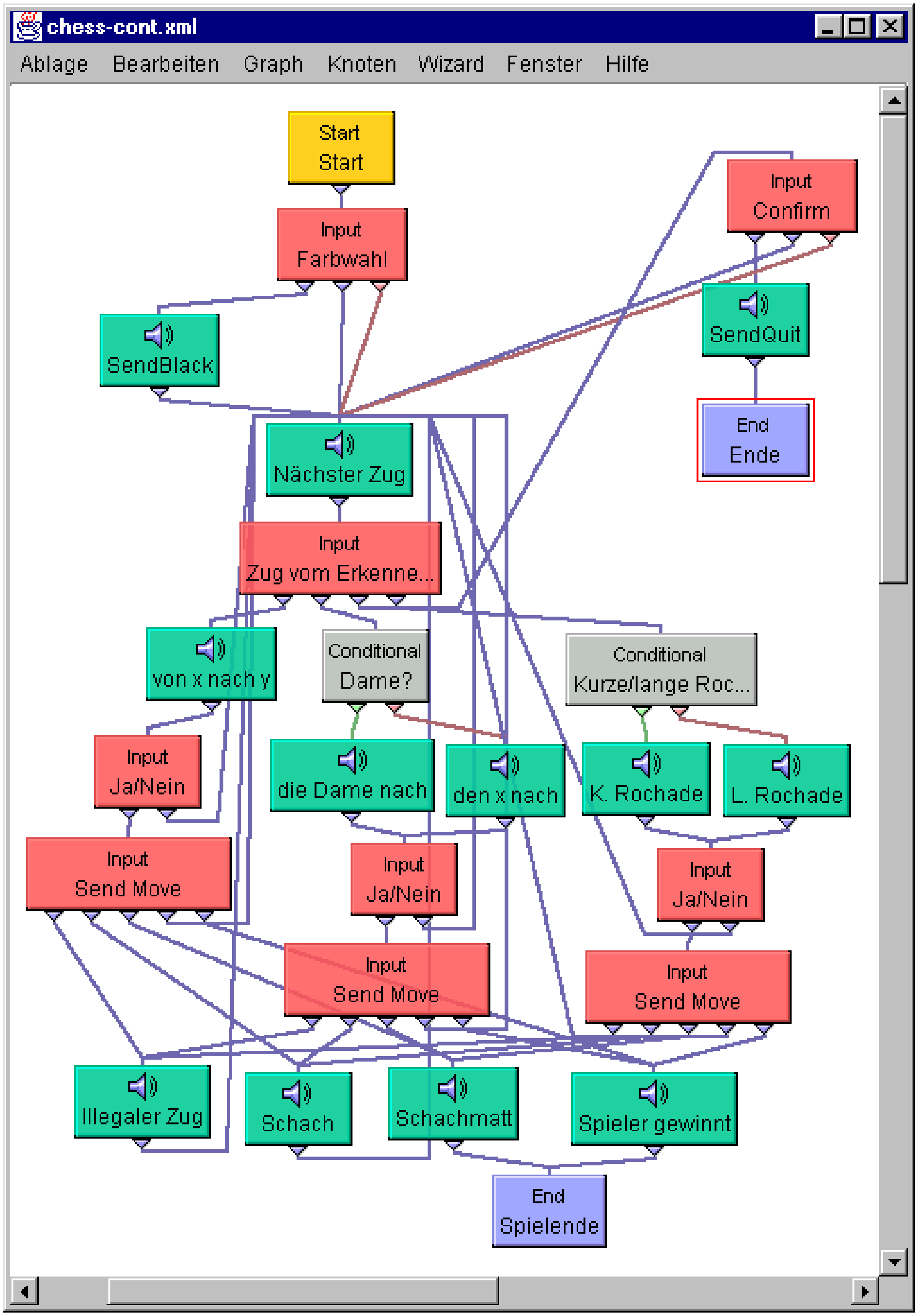


Abbildung 2: Die 2. Version des Dialogsystems

### 5.3 Der neue "nächste Zug"

Die sich in der ersten Version über vier Eingabeknoten und zahlreiche Variablenknoten erstreckende Zugeingabe fällt nach der Überarbeitung in einen einzelnen Knoten zusammen. Statt eines Ausgangs für jedes mögliche Stichwort gibt es derer jetzt lediglich vier, die einfach auf das jeweilige durch die Kombination der spezifizierten Attribute charakterisierte Muster der unterschiedlichen Eingaben reagieren. An dieser Stelle gesellt sich zu der Option, das Spiel zu beenden, und den schon vorher möglichen Formulierungen eines Zuges durch Angabe entweder des Figurtyps und der Zielkoordinaten oder sowohl der Start- als auch der Zielkoordinaten nun außerdem noch die Alternative, eine Rochade anzusagen.

#### 5.3.1 Erkennbare Muster — Benutzereingaben

Im Unterschied zur ersten Version werden die einzelnen Daten jetzt nicht mehr nacheinander, sondern in einer einzigen Merkmalsstruktur übergeben. Für die Variante mit zwei Koordinatenpaaren sieht das, was vom Erkenner geschickt wird, dann z.B. so aus:

```
[vonbst=d;vonzahl=sieben;nachbst=d;nachzahl=fuenf]
```

Beispiele für einen Zug mit Figurtyp plus Zielkoordinaten, eine (kurze) Rochade sowie ein *"Spiel beenden"*-Kommando wären:

```
[figur=bauer;nachbst=d;nachzahl=fuenf]
[specialmove=krochade]
[special=schluss]
```

Um diese Muster zu erkennen und gleichzeitig die übermittelten Werte an die dem Dialogsystem eigenen Variablen, die in dieser Version übrigens sämtlich vom Typ `String` sind, zu binden (bei *"Spiel beenden"* nicht notwendig), werden anstelle einzelner Signalwörter nun ganze Merkmalsstrukturen erkannt:

```
[vonbst=$vonbst;vonzahl=$vonzahl;nachbst=$nachbst;nachzahl=$nachzahl]
[figur=$figur;nachbst=$nachbst;nachzahl=$nachzahl]
[specialmove=$movetype]
[special=schluss]
```

#### 5.3.2 Änderungen in der Erkennergrammatik

Wir stellen zunächst das Verfahren vor, bei dem ein Zug durch zwei Koordinatenpaare beschrieben wird, und gehen danach auf die anderen Alternativen (Zug mit Figurtyp plus Zielkoordinaten, Rochade, Spiel beenden) ein.

### 5.3.3 Anton 1 schlägt Anton 2 — Startfeld + Zielfeld

Für die Erkennung einer Eingabe mit zwei Koordinatenpaaren wurde unsere Grammatik um eine Formulierung erweitert (eckige Klammern stehen wie gehabt für Optionalität):

```
[<ziehe>] [<von>] <vonfeld> [<nach>] <nachfeld> [<bitte>]
```

Hierbei dienen die Wörter in spitzen Klammern als Platzhalter für längere Ausdrücke:

```
<bitte> = bitte;
<von>  = von;
<nach> = nach | auf | schlägt;
<ziehe> = [<bitte>] zieh [<bitte>] [<accartikel>] |
  [<bitte>] ziehe [<bitte>] [<accartikel>];
<accartikel> = den | die | das;
<vonfeld> = <vonbst> <vonzahl>;
<nachfeld> = <nachbst> <nachzahl>;
```

Im Vertrauen auf eine Verbeserung der Spracherkennung durch Einbeziehung von mehr Kontext ist es jetzt auch möglich, die Buchstaben ganz ohne Buchstabieralphabet zu verwenden, was unserer eingangs erwähnten Philosophie entgegenkommt, dass der Benutzer sich, wenn möglich, eines natürlichen Ausdrucks bedienen können sollte. Hier also die Einträge für die Startkoordinaten (die für die Zielkoordinaten erhält man einfach durch Ersetzen von `vonbst` durch `nachbst`, bzw. `vonzahl` durch `nachzahl`):

```
<vonbst> = [a wie] anton {vonbst:a} |
a {vonbst:a} |
[bee wie] berta {vonbst:b} |
bee {vonbst:b} |
[zeh wie] (cäsar|zäsar|zesar|zesa|zeser|zäser) {vonbst:c} |
zeh {vonbst:c} |
[dee wie] dora {vonbst:d} |
dee {vonbst:d} |
[e wie] emil {vonbst:e} |
e {vonbst:e} |
[eff wie] friedrich {vonbst:f} |
eff {vonbst:f} |
[geh wie] (gustav|gustaf) {vonbst:g} |
geh {vonbst:g} |
[ha wie] heinrich {vonbst:h} |
ha {vonbst:h};
```

```
<vonzahl> = eins {vonzahl:eins} |
zwo {vonzahl:zwo} |
drei {vonzahl:drei} |
vier {vonzahl:vier} |
(fünf|fünneff) {vonzahl:fuenf} |
sechs {vonzahl:sechs} |
sieben {vonzahl:sieben} |
acht {vonzahl:acht};
```

Es sind nun also auch folgende Eingabeschemata möglich:

*d7 nach d5, bitte.*
*Zieh bitte von d wie Dora, 7 auf d wie Dora, 5.*
*Dora 7 schlägt Dora 5.*
*d7 — d5.*

### 5.3.4 Grammatik für Bauern — Figurtyp nennen II

Beim zweiten Zugtyp wird die Figur mitangegeben. Der Grammatikeintrag dafür sieht so aus:

```
[<ziehe>] <piece> <nach> <nachfeld> [<bitte>]
```

Für <ziehe>, <nach>, <nachfeld> und <bitte> gelten genau dieselben Definitionen wie oben, hier also nur noch der fehlende Eintrag für <piece>:

```
<piece> = bauer {figur:bauer} |
(könig|könich|könisch) {figur:koenig} |
dame {figur:dame} |
läufer {figur:laeufer} |
(springer|pferd) {figur:springer} |
turm {figur:turm};
```

Mögliche Eingaben sind jetzt unter anderem:

*Bitte ziehe den Bauer nach d5.*
*Bauer auf d wie Dora, 5, bitte.*
*Pferd schlägt Dora 5.*

### 5.3.5 Bitte kurz und klein rochieren... — ein neuer Spielzug

Für die Rochade mit ihren beiden Spielarten musste ein komplett neuer Eintrag in der Erkennergrammatik erstellt werden, der folgendermaßen aussieht: (Der Wert von `specialmove` (`krochade` oder `lrochade`) wird anschließend im Dialogsystem an die Variable `movetype` gebunden).

```
<rochade> = <willgern> (kurz|klein) <rochverb> {specialmove:krochade} |
    (kleine|kurze) <rochnom> {specialmove:krochade} |
    <willgern> (lang|groß) <rochverb> {specialmove:lrochade} |
    (große|lange) <rochnom> {specialmove:lrochade};
<willgern> = [ich (möchte|will)] [gerne|bitte];
<rochverb> = rochieren | roschieren;
<rochnom> = rochade | roschade;
```

Die Schreibweisen mit *"ch"* und *"sch"* sollen es der Software erleichtern, unterschiedliche Aussprachevarianten erkennen zu können. Es werden nun Sätze wie die folgenden akzeptiert:

*Ich will gerne klein rochieren.*
*Kurze Rochade.*
*Bitte kurz rochieren.*

### 5.3.6 Am Ende bleibt alles beim Alten — Spielende II

Der Spracherkennereintrag für die Option, das Spiel zu beenden, ist unverändert geblieben; ähnlich wie bei der Farbwahl wird allerdings zusätzlich der Attributname `special` mitübergeben. Wie gehabt, erfolgt anschließend eine Rückfrage, bevor das Dialogsystem tatsächlich verlassen wird oder die Frage nach dem nächsten Zug erneut zu hören ist. In diesem Teilabschnitt ist jedoch nach wie vor alles beim Alten, er soll hier deshalb nicht noch einmal wiederholt werden.

## 5.4 Rückfragen einfacher gemacht

Bei allen drei Zugvarianten erfolgt als nächstes eine Bitte um Bestätigung der verstandenen Eingabe, was im ersten Fall z.B. so lautet:

```
"Sie möchten also von " + vonbst + " " + vonzahl  + " nach " +
 nachbst + " " + nachzahl + " ziehen. Stimmt das so?"
```

Die in der letzten Version verwendeten Typkonversionen usw. sind nun nicht mehr nötig, da alle Variablen vom Typ `String` sind und deswegen problemlos in die Ausgabe eingebaut werden können. Bei der Bezugnahme auf einen Figurtyp jedoch ist weiterhin eine Unterscheidung bezüglich des grammatischen Geschlechts vonnöten, wozu nach wie vor ein Konditionalknoten mit der Bedingung

```
figur == "dame"
```

zwischengeschaltet ist. Die nach der Entscheidung ausgegebene Rückfrage gibt es also in zwei Varianten:

```
"Sie möchten also die Dame nach " + nachbst + " " + nachzahl +
 " ziehen. Stimmt das so?"
"Sie möchten also den " + figur + " nach " + nachbst + " " +
 nachzahl + " ziehen. Stimmt das so?"
```

Auch für die Formulierung der Bitte um Bestätigung einer Rochade ist eine Variablenabfrage notwendig:

```
movetype == "krochade"
```

Ist die Bedingung erfüllt, fragt das System, ob der Spieler eine kurze Rochade durchführen möchte. Ansonsten wird von einer langen Rochade ausgegangen:

*Sie möchten also kurz rochieren. Stimmt das so?*
*Sie möchten also lang rochieren. Stimmt das so?*

## 5.5 Mal schauen, was GnuChess dazu sagt... — Zugevaluation II

Wie auch immer die Frage lautet — der Benutzer bejaht oder verneint sie als nächstes. Daraufhin landet er entweder ohne Umwege wieder direkt bei der Aufforderung, einen Zug einzugeben, oder das System sendet den Zug an den Client. Da dies mit drei unterschiedlichen Informationspaketen geschieht, gibt es auch nach der Bestätigung drei Ein-/Ausgabeknoten mit den folgenden alternativen Nachrichtentypen:

```
[type="MOVE",fromY=vonzahl,toY=nachzahl,fromX=vonbst,toX=nachbst]
[piece=figur,type="MOVE",toY=nachzahl,toX=nachbst]
[piece=movetype,type="MOVE"]
```

Die Rückmeldungen, die von GnuChess über den Client zurückkommen, unterscheiden sich jedoch nicht, und so treffen sich alle Varianten nun in denselben fünf Ausgabeknoten wieder. Das Signalwort `OK` bedeutet dabei, dass Zug und Gegenzug problemlos und ohne besondere Vorkommnisse ausgeführt werden konnten — die nächste Station ist in diesem Fall wieder einmal die Eingabeaufforderung. Auch `Illegal`, `Check` und `I win` (das alte `Checkmate`) bringen keine Veränderungen gegenüber der ersten Version mit sich. Neu ist dagegen das selbsterklärende `You win` — der glückliche Benutzer bekommt ein knappes und ein wenig eingeschnapptes

*Sie haben gewonnen!*

zu hören, und ein Endknoten bezeichnet den Schluss der Partie.

# 6 Zusammenfassung

Wir haben ein Dialogsystem für einen Schachroboter entwickelt, das in einer relativ natürlichen Art und Weise mit dem Benutzer kommuniziert. Für die Realisierung der einzelnen Aufgaben Farbauswahl, Zugeingabe, Zugbestätigung und Spielende haben wir in DiaWiz Dialogknoten spezifiziert, diese sinnvoll miteinander verknüpft, Sprachausgaben formuliert und Einträge für eine Spracherkennergrammatik geschrieben. Anfangs notwendige Einschränkungen der Benutzereingabe haben wir in einer überarbeiteten Version durch eine Umstellung auf die Erkennung ganzer Merkmalsstrukturen statt einzelner Signalwörter beseitigen können.

Das Resultat unserer Arbeit wurde abschließend auf der CeBIT 2003 einem breiten Publikum präsentiert und hat sich dort bewährt. Als Erweiterung des Systems in der Zukunft wären z.B. eine Integration von Klärungsdiskursen bei ambigen Zügen oder auch eine tutorielle Funktion, die dem Spieler im Verlauf der Partie Tipps gibt, denkbar.

# 7 Anhang

## 7.1 Komplette Erkennergrammatik 1. Version

```
#JSGF V1.0 ISO8859-1 de;

grammar chess;

public <cmd> = [von|nach] <char> |
<number> |
<piece> |
<schluss> {schluss} |
(ja|jo|jawoll) {ja} |
(nein|nö|nee) [stimmt nicht] {nein} |
[ich (möchte|will)] [gerne|bitte] [mit] schwarz [spielen] {schwarz} |
[ich (möchte|will)] [gerne|bitte] [mit] (weiß|weiss) [spielen] {weiss};
<char> = [a wie] anton {anton} |
[bee wie] berta {berta} |
[zeh wie] (cäsar|zäsar|zesar|zesa|zeser|zäser) {caesar} |
[dee wie] dora {dora} |
[e wie] emil {emil} |
[eff wie] friedrich {friedrich} |
[geh wie] (gustav|gustaf) {gustav} |
[ha wie] heinrich {heinrich};
<number> = eins {eins} |
zwo {zwo} |
drei {drei} |
vier {vier} |
(fünf|fünneff) {fuenf} |
sechs {sechs} |
sieben {sieben} |
acht {acht};
<piece> = bauer {bauer} |
(könig|könich|könisch) {koenig} |
dame {dame} |
läufer {laeufer} |
springer {springer} |
turm {turm};
<schluss> = schluss |
ende |
ich (will|möchte) (nicht mehr|aufhören) |
[ich habe] keine Lust [mehr] |
mir reicht's |
programm verlassen;
```

### 7.2 Komplette Erkennergrammatik 2. Version

```
#JSGF V1.0 ISO8859-1 de;

grammar chess-cont;

public <cmd> =
  [<ziehe>] <piece> <nach> <nachfeld> [<bitte>] |
  [<ziehe>] [<von>] <vonfeld> [<nach>] <nachfeld> [<bitte>] |
  <rochade> |
  <schluss> {special:schluss} |
  (ja|jo|jawoll) {special:ja} |
  (nein|nö|nee) [stimmt nicht] {special:nein} |
  [ich (möchte|will)] [gerne|bitte] [mit] schwarz [spielen] {special:schwarz} |
  [ich (möchte|will)] [gerne|bitte] [mit] (weiß|weiss) [spielen] {special:weiss};

<piece> = bauer {figur:bauer} |
(könig|könich|könisch) {figur:koenig} |
dame {figur:dame} |
läufer {figur:laeufer} |
(springer|pferd) {figur:springer} |
turm {figur:turm};

<bitte> = bitte;
<von>  = von;
<nach> = nach | auf | schlägt;

<ziehe> = [<bitte>] zieh [<bitte>] [<accartikel>] |
  [<bitte>] ziehe [<bitte>] [<accartikel>];

<accartikel> = den | die | das;

<vonfeld> = <vonbst> <vonzahl>;
<nachfeld> = <nachbst> <nachzahl>;

<vonbst> = [a wie] anton {vonbst:a} |
a {vonbst:a} |
[bee wie] berta {vonbst:b} |
bee {vonbst:b} |
[zeh wie] (cäsar|zäsar|zesar|zesa|zeser|zäser) {vonbst:c} |
zeh {vonbst:c} |
[dee wie] dora {vonbst:d} |
dee {vonbst:d} |
[e wie] emil {vonbst:e} |
```

```
e {vonbst:e} |
[eff wie] friedrich {vonbst:f} |
eff {vonbst:f} |
[geh wie] (gustav|gustaf) {vonbst:g} |
geh {vonbst:g} |
[ha wie] heinrich {vonbst:h} |
ha {vonbst:h};

<vonzahl> = eins {vonzahl:eins} |
zwo {vonzahl:zwo} |
drei {vonzahl:drei} |
vier {vonzahl:vier} |
(fünf|fünneff) {vonzahl:fuenf} |
sechs {vonzahl:sechs} |
sieben {vonzahl:sieben} |
acht {vonzahl:acht};

<nachbst> = [a wie] anton {nachbst:a} |
a {nachbst:a} |
[bee wie] berta {nachbst:b} |
bee {nachbst:b} |
[zeh wie] (cäsar|zäsar|zesar|zesa|zeser|zäser) {nachbst:c} |
zeh {nachbst:c} |
[dee wie] dora {nachbst:d} |
dee {nachbst:d} |
[e wie] emil {nachbst:e} |
e {nachbst:e} |
[eff wie] friedrich {nachbst:f} |
eff {nachbst:f} |
[geh wie] (gustav|gustaf) {nachbst:g} |
geh {nachbst:g} |
[ha wie] heinrich {nachbst:h} |
ha {nachbst:h};

<nachzahl> = eins {nachzahl:eins} |
zwo {nachzahl:zwo} |
drei {nachzahl:drei} |
vier {nachzahl:vier} |
(fünf|fünneff) {nachzahl:fuenf} |
sechs {nachzahl:sechs} |
sieben {nachzahl:sieben} |
acht {nachzahl:acht};

<schluss> = schluss |
```

```
ende |
ich (will|möchte) (nicht mehr|aufhören) |
[ich habe] keine Lust [mehr] |
mir reicht's |
programm verlassen;
<willgern> = [ich (möchte|will)] [gerne|bitte];

<rochade> = <willgern> (kurz|klein) <rochverb> {specialmove:krochade} |
    (kleine|kurze) <rochnom> {specialmove:krochade} |
    <willgern> (lang|groß) <rochverb> {specialmove:lrochade} |
    (große|lange) <rochnom> {specialmove:lrochade};
<rochverb> = rochieren | roschieren;
<rochnom> = rochade | roschade;
```