University of Applied Sciences Wedel, Faculty of Media Information Science

Steuerung einer Open-GL-Szene durch die Auswertung von PROLOG Antworten

Henry G. Kleta
Dipl.-Ing.(FH), PgC

Dezember 2001

The Wumpus is dead...

Zusammenfassung

Grafische Umgebungen - z.B. VR-Anwendungen - sind oft nur durch mit den entsprechenden Eingabegeräten vertrauten Personen zu nutzen.

Um es den geübten Personen einfacher zu machen, und um es auch ungeübten Personen zu erlauben solche Anwendungen nutzen zu können, müssen andere Eingabemöglichkeiten geschaffen werden. Die Verwendung der Sprache als "Eingabegerät" bietet sich hierzu an.

Dieser Bericht zeigt anhand eines einfachen Beispiels die Grundlagen für die Nutzung von einer in PROLOG implementierten Spracherkennung zur Steuerung von grafischen Umgebungen.

1 Einleitung

Grafische Umgebungen haben den grossen Vorteil, dass auch ein mit der Nutzung von Computern unerfahrener Nutzer durchaus in der Lage ist diese zu verstehen. Durch seine Unerfahrenheit mit Computern ist es ihm jedoch oft nicht möglich die notwendigen Eingabegeräte hinreichend gut zu beherrschen, um die Umgebung zufriedenstellend nutzen zu können.

Daher bietet es sich an ein "Eingabegerät" zu nutzen, was nahezu allen Menschen zu Verfügung steht: die Sprache.

Leider ist die menschliche Sprache sehr komplex, und daher teilweise nur mit sehr grossen Aufwand von einem Computer auszuwerten.

Im Rahmen meines Master Studiums an der Fachhochschule Wedel [FHW] habe ich für einen Seminarvortrag das Thema Spracherkennung bearbeitet [HGK] und auf diesem Seminar aufbauend ein auf dem Textadventure *wumpus* (beschrieben unter anderem in [AIMA, Kapitel 6.2] basierendes Open-GL Programm entwickelt, bei dem die Bewegung des Agenten[1] mittels von PROLOG ausgewerteter Texteingabe gesteuert wird.

[1] Alle geschlechterspezifischen Formen in dieser Dokumentation wurden so gewählt, dass sie den allgemein üblichen Formen entsprechen. Sie gelten selbstverständlich gleichberechtigt für alle Lebensformen, die in der Lage sind diesen Text zu lesen.

In Abschnitt 2 werden die Grundlagen der Spracherkennung behandelt. Die Implementierung wird in Abschnitt 3 beschrieben. Im Abschnitt 4 wird auf mögliche Erweiterungsmöglichkeiten eingegangen. Abschnitt 5 erklärt die konkrete Nutzung des Programmes Abschliessend sind die verwendeten Quellen im Literaturverzeichniss angegeben.

2 Sprache [AIMA, Kapitel 22.1]

Bei dem vorliegenden Programm ging es darum angepasst an die Beispiel-Problematik eine Spracherkennung zu implementieren. In diesem Abschnitt werden die der umgesetzten Spracherkennung zugrundeliegenden theoretischen Hintergründe dargestellt.

Es gibt mehrere Typen von Sprachen, die - je nach Anwendung - sowohl Vor- als auch Nachteile haben. Generell wird zwischen formalen Sprachen (z.B. Programmiersprachen) und natürlichen Sprachen (z.B. Englisch) unterschieden.

2.1 Formale Sprache

Eine formale Sprache ist definiert als ein Folge von strings, wobei jeder string aus einer Sequenz von Terminal-Symbolen besteht. Die Menge der Terminal-Symbole ist endlich. Im Falle von Englisch umfasst die Menge ca. 400.000 Wörter. Ein String kann in Phrasen unterteilt werden (z.B. Nomen-Phrase - `np`, Verb-Phrase - `vp`, etc). Diese Phrasen erlauben die Erstellung von syntaktischen Regeln (z.B. `s --> np, vp.`) sowie eine Zuordnung von Semantik. Diese Zuordnung von Semantik ist ein essentieller Bestandteil einer Spracherkennung und wird im Abschnitt 3.2.2 beschrieben.

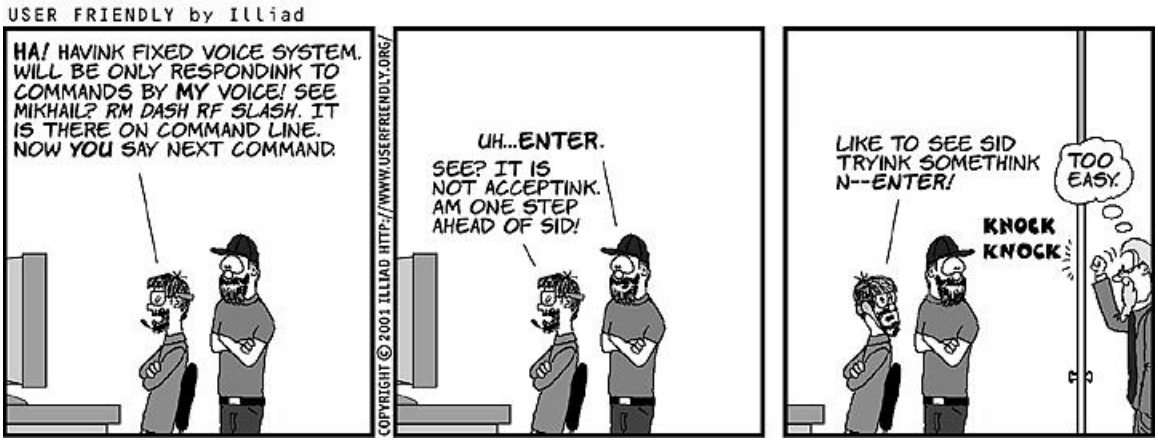
2.2 Natürliche Sprache

Eine natürliche Sprache (z.B. Englisch) ist wesentlich umfangreicher, wodurch eine höhere Flexibilität erreicht wird. Jedoch kann aufgrund folgender Punkte eine natürliche Sprache nicht als eine einfache Folge von strings charakterisiert werden:

- es gibt unterschiedliche Wortschätze (aussie, kiwi, cockney, yank, ...)
- die Sprache verändert sich mit der Zeit (z.B. Fräulein, geil, ...)
- Wortkonstrukte sind verständlich obwohl falsch (z.B. Verona Feldbusch)
- Wortkonstrukte sind weder richtig noch falsch (z.B. “next to whom did you stand”)

Desweiteren können bei der natürlichen Sprache sehr viele Probleme, vor allem in Form von Mehrdeutigkeiten [AIMA, Kapitel 22.8] auftreten, die bei der Nutzung einer eigenen, relativ streng definierten Sprache umgangen werden können.

Beispiel einer Mehrdeutigkeit [UF]:



Dieses Beispiel stellt auf humoristische Art und Weise dar, was für Folgen ein einfaches Missverständniss haben kann. In diesem Falle ordnet der Hörer (im Beispiel der Computer) dem gesprochenen Wort *ENTER* die Bedeutung der Bestätigung der Eingabe zu, und nicht - der Intention des Sprechers entsprechend - als an eine andere Person gerichtete Aufforderung den Raum zu betreten.

2.3 Sprache für Spracherkennung

Die Sprache, die bei dem erstellten Beispiel-Programm angewendet wurde, ist eine definierte Untermenge der natürlichen Sprache Englisch. Englisch wurde wegen der Einfachheit der Sprache gewählt. Deutsch hat eine wesentlich komplexere Grammatik und ist deshalb zu *schwierig*. Durch diese *Mischung* der beiden Sprachtypen, ergibt sich die Möglichkeit der Nutzung von Hilfsmitteln der formalen Sprach-Theorie bei der Bearbeitung der *natürlichen* Sprache. So können z.B. Mehrdeutigkeiten vermieden werden, indem der Wortschatz entsprechend gewählt wird.

3 Umsetzung

In diesem Abschnitt wird beschrieben, wie die in Abschnitt 2 beschriebenen Grundlagen angewendet wurden, um eine durch Sprache steuerbare Wumpus-Welt (angelehnt an [AIMA, Kapitel 6.2]) zu schaffen.

3.1 Die Beispiel Welt [AIMA, Kapitel 6.2.]

Die Beispiel Welt ist einem Schachbrett ähnlich als Raster aufgebaut und von einer undurchdringlichen Mauer umgeben.

In dieser Welt gibt es nur zwei Charaktere, die Aktionen durchführen können (also Akteure): den Agenten und den Wumpus. Der Agent kann sich in der Welt bewegen und Aussagen über seinen Lebenszustand treffen. Der Wumpus - bereits per Definition in seiner Aktionsvielfalt beschnitten - kann im Beispiel lediglich als tot bzw. lebendig beschrieben werden. Ausserdem ist der Wumpus für den Agenten nicht sichtbar.

Mögliche Aktionen des Agenten:

- Bewegung
  - in die vier Himmelsrichtungen
  - nach vorne, rechts, hinten, links
- Drehung
  - in die vier Himmelsrichtungen
  - nach vorne, rechts, hinten, links
- Statusaussage
  - tot / lebendig

Mögliche Aussagen über den Wumpus:

- Status
  - tot / lebendig

All diese möglichen Akteure, Aktionen und Parameter galt es grammatikalisch zu definieren und programmtechnisch umzusetzen. Die Beschreibung folgt in den nächsten Abschnitten.

3.2 Grammatik [SHOH, Kapitel 10]

Die Implementation einer Grammatik setzt normalerweise das Schreiben eines Parsers voraus, der den eingegebenen Satz untersucht, und den einzelnen Satzteilen der Grammatik entsprechende Bedeutungen zuordnet. Solche Parser sind dank des in PROLOG integrierten Formalismus *Definite Clause Grammar* kurz *DCG* relativ einfach umzusetzen.

Es werden folgende Symbole benötigt:

- Nicht-Terminal-Symbole:
  - z.B. `s` für Satz
  - z.B. `np` für Nomen-Phrase
  - z.B. `vp` für Verb-Phrase
- Terminal-Symbole:
  - z.B. `pronoun --> [i].`
  - z.B. `verb --> [go].`

Mit Hilfe dieser Symbole lassen sich die Sprache definierende Regeln aufstellen:

- Grammatik:
  - `s --> np,vp.`
  - `np --> pronoun.`
  - `vp --> verb.`

Diese Grammatik besagt, dass eine als `np` kategorisierte Liste gefolgt von einer als `vp` kategorisierten Liste einen gültigen Satz `s` ergibt.

Entsprechend der definierten Terminal-Symbole (also dem Wortschatz) bedeutet dieses im Beispiel, dass der string `i go` als ein `np` gefolgt von einem `vp`, und somit als ein gültiger Satz `s` erkannt wird.

3.2.1 Syntax

Bei der Prüfung der Syntax wird lediglich das Auftreten bestimmter Worte an bestimmter Stelle im Satz bzw. die Reihenfolge der Worte überprüft. So ist der Satz `i go north` syntaktisch ebenso korrekt wie der Satz `i go wumpus`, auch wenn dieser Satz überhaupt keinen Sinn ergibt. Die Prüfung der Semantik (der Bedeutung) der Sätze wird zu einem späteren Zeitpunkt in Abschnitt 3.2.2 beschrieben.

Um eine Syntax-Prüfung implementieren zu können, muss die Grammatik - also auch der Wortschatz - definiert werden. Dieses sollte angelehnt an die Anwendung, also angelehnt an die möglichen Sätze geschehen.

In der Beispiel-Welt soll durch Spracheingabe ein Akteur dazu bewegt werden eine bestimmte Aktion durchzuführen. Befehle, die lediglich aus Akteur und einer Aktion bestehen (z.B. `i go`) machen in dieser Welt keinen Sinn. Komplette Befehle müssen also aus Akteur, Aktion sowie Parameter bestehen.

Es geht darum einfache Befehle umzusetzen, die entweder den Agenten oder aber den Wumpus betreffen.

Je nachdem, ob der Agent als Avatar angesehen wird oder nicht, ist die erste Person (z.B. `i go north`) oder die dritte Person (z.B. `the agent goes north`) zu wählen.

Der Vollständigkeit halber müssen sowohl für die erste Person, als auch die dritte Person der Nominativ (das Subjekt des Satzes - z.B. `i` bzw. `it`) und der Akkusativ (das Objekt des Satzes - z.B. `me` bzw. `it`) betrachtet werden, da sonst eine spätere Erweiterung der Grammatik um Sätze wie z.B. `i grab it` (ein Satz der das Ziel im zugrundeliegenden Textadventure beschreibt - den Griff nach dem Gold) mit erheblichen Änderungen der Grammatik, und nicht nur mit Erweiterungen des Wortschatzes, verbunden wäre.

Es gibt also folgende Akteure:

- der Agent
  - `i`
  - `me`
  - `agent`
- der Wumpus
  - `it` (sowohl Nominativ als auch Akkusativ)
  - `wumpus`

Zusätzlich ist eine Befehlsform (z.B. `go north`) wünschenswert, da viele User einen Agenten als einfachen Befehlsempfänger ansehen. Diese Befehlsform kann im englischen durch einfaches Weglassen des Subjekts (also des Akteurs) bei der ersten Person erreicht werden, da die entsprechenden Verbformen identisch sind. In solch einem Fall wird also automatisch die erste Person (im Beispiel `i`) als Akteur angenommen.

Somit ergeben sich folgende mögliche Aktionen:

- Bewegung
  - `go`
  - `goes`
- Drehung
  - `turn`
  - `turns`
- Statusaussage
  - `am`
  - `is`

Wie zuvor beschrieben muss ein Befehl aus einem Akteur, einer Aktion und einem Parameter bestehen.

Es ergeben sich folgende mögliche Parameter:

- Himmelsrichtung
  - `north`
  - `east`
  - `south`
  - `west`
- Bewegungsrichtung
  - `ahead`
  - `right`
  - `back`
  - `left`
- Status
  - `alive`
  - `dead`

Bei den Himmelsrichtungen ist zu beachten, dass diese sowohl als Nomen (wie z.B. in `i go to the north`) als auch als Adverb (wie z.B. in `i go north`) auftreten können und somit auch in beiden Formen implementiert werden müssen.

Deswegen werden auch noch zusätzlich der Artikel `the` sowie die Präposition `to` benötigt.

Mit Sätzen, die ausschliesslich aus den oben genannten Worten bestehen, lässt sich die gesamte im Abschnitt 3.1 beschriebene Welt steuern.

3.2.2 Semantik

Ein wichtiger Punkt neben der reinen Prüfung der Sätze auf Korrektheit (Syntax) ist die Zuordnung von Bedeutung (Semantik). Im Falle der im Abschnitt 3.1 beschriebenen Welt sind an Semantik Akteur, Aktion und Parameter von Bedeutung. Ein Satz, der zur Überprüfung an PROLOG übergeben wird, wird geparsed und auf syntaktische Korrektheit geprüft. Normalerweise liefert PROLOG lediglich ein `yes` (bei mindestens einer gefundenen Lösung) bzw. ein `no` (bei keiner gefundenen Lösung). Diese Antworten sind wenig hilfreich um auf die Semantik des übergebenen Satzes zu schliessen.

Werden jedoch an PROLOG neben dem zu überprüfenden Satz zusätzlich Variable übergeben, kann durch PROLOG diesen Variablen vordefinierte Werte zugeordnet werden, anhand deren das übergebende Programm Semantik zuweisen kann.

So sorgt z.B. der Eintrag `pronoun(agent) --> [i].` in der Wissensbasis des PROLOG Programmes dafür, dass einer Variable A der Wert `agent` zugewiesen wird, wenn das Wort `i` als Pronomen erkannt wird. Ein entsprechender PROLOG Aufruf wäre z.B. `pronoun(A,[i],[]).`[2]

3.3 PROLOG [SWI]

Wie in Abschnitt 3.2.1 beschrieben muss ein vollständiger Befehl aus Akteur, Aktion und Parameter bestehen. In Abschnitt 3.2.2 wurde beschrieben, dass Variable verwendet werden um diesen durch PROLOG Werte zuweisen zu lassen. Diese Werte können anschliessend durch das aufrufende Programm ausgewertet werden, um z.B. den Werten entsprechende Funktionen auszuführen.

Es werden also folgende Variable benötigt:

- für den Akteur
  - `Actor`
- für die Aktion
  - `Action`
- für den Parameter
  - `Param`

Da der Parameter immer die Aktion beschreibt, können diese beiden Variablen zusammen gefasst werden. Der Wert von `Action` wird also aus der Aktion und dem Parameter zusammengesetzt (in der Form `Action(Param)` - z.B. `go(north)`).

In den nun folgenden Abschnitten wird die konkrete Implementierung der zuvor beschriebenen Grammatik in SWI-PROLOG[3] beschrieben.

[2] Die Beschreibung der Aufruf-Syntax kann [SHOH, Kapitel 10] und [SWI] entnommen werden. Siehe hierzu auch Abschnitt 3.4.1.2

[3] Mein Dank geht an dieser Stelle an Dipl.-Ing.(FH) Martin "Herbert" Dietze für die Portierung von SWI-PROLOG nach IRIX

3.3.1 kompletter Satz

```
s(Actor,Action) --> np(Person,nominative,Actor),vp(Person,Action).
s(agent,Action) --> vp(1,Action).
```

Die erste Regel besagt, dass ein gültiger Satz aus einer Nomen-Phrase in der Nominativ Form gefolgt von einer Verb-Phrase in der gleichen Person wie die vorangegangene Nomen-Phrase bestehen kann.

Die zweite Regel stellt die Implementierung der im Abschnitt 3.2.1 beschriebenen Befehlsform dar: ein gültiger Satz kann alleine aus einer Verb-Phrase der ersten Person bestehen, der Variable `Actor` wird automatisch der Wert `agent` zugewiesen.

Ausserdem lässt sich an den Regeln erkennen, dass die Nomen-Phrase den Wert der Variable `Actor` und die Verb-Phrase den Wert der Variable `Action` bestimmt, was auch der im Abschnitt 3.2.1 beschriebenen Syntax entspricht.

3.3.2 Nomen-Phrase

```
np(Person,Case,Actor) --> pronoun(Person,Case,Actor).
np(3,_,Actor)         --> article,noun(Actor).
```

Die erste Regel beschreibt eine allgemeine Nomen-Phrase, die nur aus einem Pronomen besteht. Die beschriebene Nomen-Phrase kann sowohl in der ersten als auch in der dritten Person, und sowohl im Akkusativ als auch im Nominativ stehen.

Die zweite Regel beschreibt eine Nomen-Phrase in der dritten Person, die aus Artikel und Nomen besteht, sowohl im Akkusativ als auch im Nominativ.

3.3.3 Verb-Phrase

```
vp(Person,Action) --> verb(Person,Action,Param),(adverb(Param);
                                                 adjective(Param);
                                                 pp(Param);
                                                 np(_,accusative,Param)).
```

Diese Regel besagt, dass eine Verb-Phrase aus folgenden Teilen besteht:

- ein Verb gefolgt von einem Adverb (z.B. `go north`)
- ein Verb gefolgt von einem Adjektiv (z.B. `is dead`)
- ein Verb gefolgt von einer Präpostitions-Phrase (z.B. `to the north`)
- ein Verb gefolgt von einer Nomen-Phrase im Akkusativ (z.B. `grab the gold`)[4]

3.3.4 Präpositions-Phrase

```
pp(Param) --> preposition,article,noun(Param).
```

Diese Regel beschreibt eine Präpositions-Phrase als eine Präposition gefolgt von einem Artikel und einem Nomen.

[4]Diese Art der Verb-Phrasen ist bereits unterstützt, obwohl noch nicht genutzt. Siehe hierzu Abschnitt 3.2.1

3.4 Hauptprogramm

In diesem Abschnitt wird auf Besonderheiten bei der Implementierung des Hauptprogrammes eingegangen.

3.4.1 Schnittstelle C-PROLOG [SWI]

Die einzige wirkliche Besonderheit an dem Hauptprogramm ist die Einbindung der PROLOG-Umgebung. In den folgenden Abschnitten werden die Funktionen, die diese Schnittstelle nutzen, beschrieben.

3.4.1.1 PROLOG Initialisierung

Mit der Funktion `InitProlog` (siehe `prolog.h` bzw. `prolog.c`) wurde eine Funktion erstellt, die die PROLOG Umgebung für die Anwendung initialisiert und auf Erfolg der Initialisierung prüft.

3.4.1.2 PROLOG Aufruf

Wie in Abschnitt 3.3.1 erkennbar, hat ein korrekter PROLOG Aufruf für die Auswertung eines Satzes der Grammatik der Beispiel Welt folgende Syntax:

```
s(Variable1,Variable2,QUERY,[]).
```

wobei der ersten Variable der Wert von `Actor`, und der zweiten Variable der Wert von `Action` zugewiesen wird (siehe Abschnitt 3.3.1).

`QUERY` ist der komplette Satz, wobei dieser als kommaseparierte Liste (also z.B. `[i,go,north]`) angegeben werden muss. Dieses ist in der Verwendung der im Abschnitt 3.2 beschriebenen *DCG* begründet. Die gleiche Begründung gilt auch für das verpflichtende Auftreten der leeren Liste nach `QUERY`.

Mit der Funktion `CallProlog` (siehe `prolog.h` bzw. `prolog.c`) wurde eine Funktion erstellt, die den fertig konvertierten Query-String an PROLOG übergibt, zwei Variable auf den Stack legt, PROLOG aufruft, und die von PROLOG genutzten Variablen wieder vom Stack liest und an die die Funktion aufrufende Instanz zurück gibt.

3.4.1.3 PROLOG beenden

Damit die PROLOG-Umgebung auch wieder beendet werden kann, wurde die Funktion `StopProlog` (siehe `prolog.h` bzw. `prolog.c`) erstellt.

Bei der Implementierung der Funktion wurde die `PL_cleanup` verwendet und nicht `PL_halt`, da die letztgenannte Funktion ein `exit` ausführt und somit das Hauptprogramm beendet, was einen ungewünschten Effekt haben kann. Siehe hierzu [SWI]

3.4.2 OpenGL [GL]

In diesem Abschnitt wird auf die Besonderheiten bei der Implementierung der GL-Szene bzw. der Tastatureingabe eingegangen.

3.4.2.1 grafische Darstellung

Die zugrundeliegende Welt wurde ihrer Definition entsprechend (siehe Abschnitt 3.1) als Raster dargestellt. Innerhalb dieses Rasters soll der Agent bewegbar (Postition und Ausrichtung) sein.

Die Position ist im Raster einfach darzustellen. Ändert sich die Position des Agenten, so wird dieser einfach mittels einer Translation um die Breite bzw. Höhe eines Rasterfeldes in die gewünschte Richtung verschoben.

Um neben der im Raster einfach darzustellenden Position des Agenten auch dessen Ausrichtung sogar für ungeübte User eindeutig darzustellen, wurde der Agent als Pfeil dargestellt, dessen Spitze in die Richtung *geradeaus* weist. Ändert sich die Ausrichtung des Agenten, so wird diese einfach mittels einer Rotation in die entsprechende Richtung geändert.

Für die konkrete Implementierung siehe `scene.h` bzw. `scene.c`.

3.4.2.2 Tastatureingabe

Bei einem GL-Programm wird durch das Drücken einer Taste ein Key-Callback ausgelöst. Für die Steuerung des vorliegendes Programmes wird jedoch - wie zuvor beschrieben - ein kompletter Query-String benötigt. Es lag also nahe, bei Auslösung eines Key-Callback zuerst auf bestimmte Funktionstasten zu prüfen (Escape: Programm Beenden, Enter: Eingabe beenden und Query-String auswerten, Backspace: letztes Zeichen löschen), und sonst den String Zeichen für Zeichen zusammenzusetzen, bis die Auswertung des String gewünscht ist (also wenn der User die Eingabe durch Drücken von ENTER beendet). Siehe hierzu `my_string.h` bzw. `my_string.c`.

Wenn der Query-String ausgewertet ist, werden die von PROLOG zurückgelieferten Variablen überprüft und deren Werten entsprechend Aktionen ausgeführt. Siehe hierzu `action.h` bzw. `action.c`.

4 Erweiterungsmöglichkeiten

In diesem Abschnitt werden Möglichkeiten dargestellt, wie das vorliegende Programm sinnvoll erweitert werden kann.

Momentan wird die Texteingabe mittels Key-Callback realisiert. Diese Implementierung verhindert gewissermassen die Anbindung einer Spracherkennung. Um diese Anbindung realisieren zu können, muss ein anderer Mechanismus gefunden werden, mittels dessen der zu bearbeitende Query-String zusammengesetzt wird. Denkbar ist hier die Schaffung eines eigenen Callbacks, der ausgelöst wird, sobald ein kompletter Query-String zur Bearbeitung vorliegt.

Die Spracherkennung sollte von einer Texterkennung zu einer Spracherkennung ausgebaut werden. Hierzu muss der gesamte Teil der Umsetzung von der Schallwelle zum Text implementiert werden.

In der momentanen Welt ist lediglich der Agent implementiert. Um dem zugrundeliegenden Textadventure auch nur halbwegs gerecht zu werden, sollten den Feldern gewisse Attribute zugefügt werden, wie z.B. die Gefahren der Wumpus Welt bzw. deren Warnungen (siehe [AIMA, Kapitel 6.2]). Diese Attribute sollten dem Agenten erst bei Betreten des Feldes offenbart werden.

Der Autor würde sich freuen, wenn er von eventuellen Änderungen / Portierungen in Kenntniss gesetzt wird, um diese dem “release” hinzufügen zu können.

5 Benutzerhandbuch

Das vorliegende Programm liegt im Source Code vor. Der Source Code steht unter folgender URL zum download zur Verfügung:

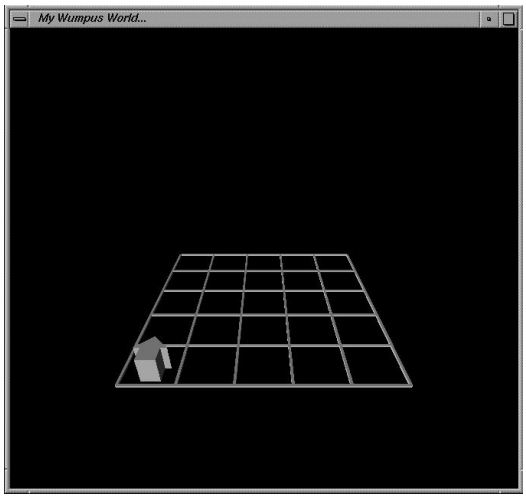
`http://www.indigo2.de`

Das Programm wurde auf SGI IRIX entwickelt, ist allerdings auch auf SuSE linux compilierbar und gestestet. Mittels einfacher Anpassung des makefiles sollte eine Portierung auf andere Unix Systeme kein Problem sein.

Die Übersetzung erfolgt durch einfachen Aufruf von `make all`. Die gewünschte Plattform ist zuvor in dem makefile bei dem target `all` einzutragen. Nach erfolgreicher Übersetzung wird die ausführbare Datei `wumpus` erzeugt.

Im Folgenden ist ein Beispiel aufgeführt, bei dem das Programm aus einer Shell gestartet wird, der Query-String `the wumpus is dead` eingegeben wird, und das Programm durch Drücken von *ESC* beendet wird. Auf der linken Seite ist die aufrufende Shell zu sehen, auf der rechten die grafische Ausgabe des Programmes:

```
kl@indigo2:~ > ./wumpus
Welcome to SWI-Prolog (Version 3.4.2)
Copyright (c) 1990-2000 University of Amsterdam.
Copy policy: GPL-2 (see www.gnu.org)

For help, use ?- help(Topic). or ?- apropos(Word).


Welcome to the 1st wumpus world...

Please set the input-focus to the wumpus world.
Press ESC to quit, anything else will be processed...

Tell Prolog: the wumpus is dead

Prolog: wumpus
Prolog: is(nomore)

Oh... poor little bugger...

Tell Prolog:

Thanks! You've made a little program very happy :-)

kl@indigo2:~ >
```



Es ist zu sehen, dass das Programm nach dem Start erst die PROLOG Umgebung lädt, und dann den User anweist, den input-focus auf das grafische Fenster zu ändern. Dieses ist notwendig, damit die Tastatureingabe von dem im Abschnitt 3.4.2.2 beschriebenen Key-Callback bearbeitet werden kann.

Nachdem der User den Query-String eingegeben hat, erscheint die Ausgabe des Programmes. Diese besteht aus den PROLOG-Antworten (zuerst der Wert von `Actor` und dann der Wert von `Action`), aus eventuellen Fehler- oder Status-Meldungen und aus einem weiteren Eingabe-Prompt. Wenn aus dem Query-String eine Aktion resultiert, wird diese im grafischen Ausgabefenster ausgeführt. In diesem Falle findet solch eine Aktion nicht statt, es wird lediglich das Mitleid den Tod des Wumpus betreffend als Status-Meldung ausgegeben.

Nach Beendigung des Programmes durch Drücken von *ESC* erscheint noch eine Abschlussmeldung.

Literatur

[AIMA] Stuart J. Russel and Peter Norvig: *Artif icial Intelligence, A Modern Approach*, Prentice-Hall International Inc.

[FHW] http://www.fh-wedel.de

[GL] Neider, Davis, Woo: *OpenGL Programming guide*

[HGK] Henry G. Kleta: *Seminar an der FH-Wedel - Sprache verstehen (vom Text zur Semantik)*, http://www.indigo2.de/master/seminar

[SHOH] Yoav Shoham: *Artif icial Intelligence Techniques in Prolog*, Morgan Kaufmann Publishers Inc.

[SWI] Jan Wielemaker: *SWI-Prolog 3.4 Reference Manual*, University of Amsterdam

[UF] http://www.userfriendly.org