Artificial Intelligence in a German Adventure Game: Spion in PROLOG

Steven R. Molla  
Logicon Inc.

Alton F. Sanders  
Wright State University

Ruth H. Sanders  
Miami University

ABSTRACT: Spion is an adventure game intended as a teaching tool for intermediate and advanced college German. It recognizes and 'understands' a small subset of German large enough to allow a student to play the game. To win the game, players are required to communicate with a fictitious agent in complete, correct German sentences. The version of the game described here was written in PROLOG as the natural outgrowth of a series of earlier programs. The program runs on an IBM-PC or compatible and is available at no cost for noncommercial purposes.

KEYWORDS: artificial intelligence, CALL, adventure games, parsing, natural language processing, PROLOG, ICAI, courseware.

Spion is a simple spy game patterned after computer adventure games. Intended for intermediate and advanced college students of German, it combines machine 'understanding' of natural language, and therefore artificial intelligence (AI) with the capacity to recognize correct German input.

A game offers a certain amount of inherent motivation for students to practice their language skills. In Spion, the student is presented with a fictitious agent, Robotky, who is in West Berlin on a mission as a spy. In order for Robotky to accomplish his mission, the student must direct him using complete sentences which must be in correct German. Since Robotky is really a program, he necessarily understands only a small subset of German. On the other hand, he understands no English at all, nor can he see gestures. Students must use only German to communicate.
The small artificial world of the game offers an acceptably natural limitation on discourse for investigative purposes. No language understanding program can deal with a very large subset of a natural language. Programs that attempt to aid in language learning have the additional burden of insisting on 'correct' language. Foreign language classes necessarily take a prescriptive view of language; programs that are too tolerant of student errors are unlikely to be effective teaching aids. Hence it is desirable to work with a programmed 'world' so that the language needed to manipulate it can grow indefinitely by either large or small increments. An adventure game provides such a world.

History

The PROLOG program described here represents the latest of several implementations. The first version of *Spion* was begun in 1981 (Sanders and Sanders 1982) and was programmed in PL/I running under CP/M on a Z80 system. That program (actually two separate independent programs) dealt with the syntactic problems of recognizing the appropriate subset of German without overgeneration and the problem of 'understanding' as two independent problems. The two were never combined into a single program. The combination of size and speed of the processor used, the programming languages available, and the experimental nature of the project made for a successful research program, but did not result in an application suitable for student use. Hence a parallel effort was undertaken to produce an unintelligent version of *Spion* for immediate student use. That effort produced *Pilot-Spion* (Sanders 1984).

*Pilot-Spion* was a fairly complex game (see Figure 1 for a diagram of scenes), but a fairly simple program, based on string matching of input against preprogrammed possible responses, rather than on parsing and language understanding. *Pilot-Spion* did, however, provide a fully functional complete program for student use. The program enforced the 'only correct German input' rule by restricting the student to one of a specific set of responses. Each move was thus a kind of multiple-choice selection. Despite its computational simplicity, *Pilot-Spion* proved very successful. Since 1983 it has been used by hundreds of college German students at Miami and Wright State Universities and elsewhere, and user responses significantly influenced the current version.

The second intelligent version was begun in LISP (TLC LISP), but ultimately completed in C-PROLOG running on a Vax 785 (Molla 1987). It
implemented the full grammar of the first Spion program as the interface to a simplified version of the Pilot-Spion game (see Figure 2). That program was the first all-in-one complete intelligent version of Spion. It performed with acceptable efficiency and interacted reasonably well with the user.
Finally, the C-PROLOG version was ported to an IBM-AT and modified according to the experience gained from *Pilot-Spion* users. Some additional features were added to the user interface as well as a facility for typing the special German characters (ä ö ü ß). The program runs in a stand-alone compiled version on IBM-compatible microcomputers.

**Related Work**

The earliest example of an intelligent program in something like an adventure game format was Terry Winograd’s now famous program, SHRDLU (Winograd 1972). At the time of Winograd’s work, there were no adventure games, and SHRDLU was not perceived as a game. There, the problem solving
was done by the program and not by the human. The human, however, did communicate in English (not a programming language) with an agent supplied by the program. The human gave the orders and the agent carried them out. The agent lived in a 'blocks world,' a small artificial world somewhat like the small world of an adventure game. Within the context of his world, Winograd's agent was much smarter than those of today's adventure games, whether intelligent games or otherwise.

There are a large number of commercial adventure games and many have fairly sophisticated parsers. As far as we know, however, there are none that concern themselves with overgeneration, or acceptance of incorrect language. Commercial games, unlike CALL games, are solely games. There is no point in their rejecting input because of grammatical errors—in fact, users might be annoyed by this. Our goal, on the other hand, is to accept nothing that would not be acceptable on a German examination.

There is at least one other project underway for intelligent adventure games for language learners. This is an ambitious project using a single language parsing and understanding model for two adventure games, one in Latin and a second in French (Culley, Mulford and Milbury-Steen 1986). Not fully implemented as of this writing, the project will attempt to provide correction for erroneous input and will include a large complex game with many alternatives and characters who may be interrogated by the player.

The Game

The game's agent, Robotky, is initially at the airport in West Berlin. He has some money, a visa good for one trip to East Berlin, and a passport. When the game begins (after supplying instructions for players who need them), Robotky describes the scene at the airport and gives the standard prompt "Was soll ich jetzt tun?" (What shall I do now?) to the player. The student tells Robotky what to do (e.g. "Nehmen Sie den Bus!" Take the bus), or perhaps asks a question (e.g. "Was haben Sie?" What do you have?). Student inputs must be in correct German and must be complete sentences. When the input is incorrect, the desired action will not be done. The message for grammatical errors is a general one and does not spell out the error. However, other kinds of errors cause more specific messages. Use of the familiar verb form to address Robotky brings a reproof, semantically
nonsensical commands (if they are recognized by the program) bring a message that the command is impossible; and use of a word not known to the system causes a repetition of the word followed by a question mark.

The user may get help by typing the word 'Hilfe' (help) at any point in the game. This brings a list of suggestions for possible actions. The user may construct a command from these suggestions, or give a different command, and the game continues. The object of the game is to locate a secret list, find a fellow spy named Frieda, exchange with Frieda the list for some information for still another spy named Max, and finally deliver the information to Max. The object of the game is not spelled out but rather, as in most adventure games, unfolds as the game progresses. Spion needs to be simple since the player is playing in a foreign language and in any event has to be simple if the program is to be capable of understanding the language of the game.

Even in a simple game, however, the vocabulary needed to describe the scenes and actions of Robotky was inconveniently large for a full understanding system. We chose not to oversimplify the descriptive language presented to the student but to have the program’s input language much simpler than its output language. The program maintains two dictionaries—one for output words, and one for input words. Words that are in the former but not the latter are rejected as ‘forbidden’ if they appear in the input. The output dictionary provides, on request, English equivalents for all words in the game. The input dictionary contains syntactic information on each word. In addition, each input word is defined through the combined efforts of the semantic processor and the game processor.

The following excerpt will illustrate how the game is played. The student’s input is given in boldface type, our explanatory comments (which do not appear in the game) are on the right.

PROLOG SPION

Ein Abenteuer auf Deutsch
von Steve Molla, Alton F. Sanders, und Ruth H. Sanders
@ 1988

Guten Tag. Sie spielen SPION. Brauchen Sie Hinweise? Antworten Sie 'ja' oder 'nein', dann drücken Sie 'enter'!

>> no

Offers instructions.

Student answers in English, program repeats request for a yes or no in German.
****ACHTUNG!****
Antworten Sie "ja" oder "nein", bitte!

>>Nein
Student declines offer of instructions.

Ich bin im Flughafen (Berlin - West). Hier sehe ich ein
Flugzeug, ein Cafe, und den City-Bus, Leute gehen
hin und her. Ich habe einen authentisch aussehenden
Paß, ein Visum gültig für einen Besuch in der DDR,
einen Flugschein und DM100.
Moment mal! Ein Zettel fällt aus dem Paß!
Was soll ich jetzt tun?

>>Lesen Sie das Zettel!
Student corrects input.

Ich verstehe das nicht. Etwas scheint nicht richtig in
Ihrem Deutsch. Sagen Sie es anders, bitte!

>>Lesen Sie den Zettel!
Student corrects input.

Bahnhof Zoo mit der Information.—Max
Was soll ich jetzt tun?

>>Gehe in die Stadt!
Robotky reproaches student.

Wir sagen doch SIE zu einander!
Was soll ich jetzt tun?

>>Gehen sie in die Stadt!
Robotky reproaches student.

Ich nehme den City-Bus. Die Fahrt hat DM10
gekostet. Ich bin jetzt in der Stadt Berlin-West, an der
Ecke Kurfürstendamm-Fasanenstraße. Ich sehe eine
Bushaltestelle mit dem Zeichen: BAHNHOF ZOO.
Auf der Straße ist ein großes Plakat: "LESEN MACHT
SPAß". Ich sehe auch einen großen Buchladen.
Was soll ich jetzt tun?
Lesen Sie das Buch!
Ich habe kein Buch!
Was soll ich jetzt tun?
>>Hilfe!
Soll ich zum Bahnhof Zoo gehen, den Bus nehmen, in den Buchladen gehen, oder zur Bushaltestelle gehen? Was soll ich jetzt tun?
Buchladen?
Buchladen bedeutet "book store" auf Englisch.
Was soll ich jetzt tun?
>>Gehen Sie in den Buchladen!
Ich bin im Buchladen. Ich sehe Bücher und ein Plakat an der Wand: DER GROßE BESTSELLER: DER PRAKTISCHE SPION.
Was soll ich jetzt tun?
...

The Program
The entire program is written in Arity PROLOG, Version 4.0, and runs as a stand-alone application (without the Arity Interpreter) so it can be distributed freely (Arity does not restrict distribution of independent applications). The program currently requires a minimum of 640K bytes of memory; since Arity PROLOG provides a virtual memory system for the PROLOG database, the game could be enhanced considerably before memory becomes a problem. Although we have not conducted any timing studies, the program is currently well within acceptable bounds for speed. On a Compaq 386 there is no perceptual delay between the input and the program’s response. On an IBM-AT the delay is less than one second in most cases, although after long playing times this increases slightly.
The program consists of six major modules:

1. **The Main Program and Program Driver.** The driver controls the flow of the game. When the game ends, the player may choose to play again. No facility is provided for saving the game state or exiting prematurely.

2. **The Lexical Processor.** The lexical processor is responsible for reading the input sentence, standardizing special characters (ä, ö, ü, ß), and looking up each word in the dictionary.

3. **The Syntax Processor.** The parser uses a definite clause grammar to perform a purely syntactic parse. There is no feedback from the semantics processor. There is one simple discourse constraint that can be syntactically recognized: although the program can parse the familiar form of verbs, it refuses to do so and considers use of the familiar form a syntax error. help.

4. **The Semantics Processor.** The semantics processor checks for a meaningful sentence independent of context. For example, if Robotky is commanded to eat a fish, the semantics processor will accept the sentence regardless of whether Robotky actually has a fish (though the game processor will reject the command if he does not). If Robotky is commanded to eat an airport, the sentence will be treated as a semantic error.

5. **The Game Processor.** The game processor enforces context constraints imposed by the current state of the game, manages the appropriate game state transitions, and gives the user an appropriate response.

6. **The Dictionary.** The dictionary is actually two dictionaries. The first contains all words known by the program to exist. This dictionary contains only the English translation of each word along with an indicator as to whether the 'understands' it. Words that are 'understood' are also in a second dictionary containing syntactic information about the word. Semantic information is not kept in the dictionary but rather is embedded in the code of the semantics processor and the game processor.

*The Main Program*

Upon initiation, the program driver must offer the student instructions, provide those instructions if desired, set up the program databases, and oversee the play of the game. For the most part, this is straightforward and requires no additional comment. The only aspect of this module that needs further explanation is the use of hash tables.
Our version of Arity PROLOG does not provide first argument indexing, and so predicates which have many clauses are best stored in an explicit hash table. The most obvious examples are the two dictionaries. The dictionary of known words is stored as a single predicate (known words/3), as is the dictionary of syntactic information (lexical item/1). The former is a predicate with over 250 clauses while the latter has over 300 clauses. Conducting linear searches through these predicates would add considerably to the processing time. In addition, compiled Arity PROLOG programs do not handle predicates asserted at run time. Hence another hash table serves as a blackboard for global information. For example, if Robotky uses his visa, which is good for only one trip into East Germany, that fact is recorded in the blackboard hash table so that any future attempt to use the visa will fail.

The Lexical Processor

The lexical processor is responsible for reading and standardizing the student's input sentence. Standardization is necessary because of the special characters used in German. Some keyboards may not be able to follow our conventions for umlauted characters (which, on a Compaq or IBM-AT, call for pressing the alt-key while typing the vowel to be umlauted, or, in the case of 'ß', alt-'s'), and so the most common substitutions (ae, oe, ue, ss) are allowed. The dictionary is stored with these substitutions, and input is transformed accordingly. The student is not aware of these transformations and may choose to use the substitutions from the keyboard. Once a word is found in the input dictionary, a token is formed. A token is either a standardized word or a list of standardized words. Only the first word of a sentence produces a list. This is because the first word is always capitalized. The program cannot tell if the word would have been capitalized if it had not been the first word of a sentence. Therefore, both forms must be considered. There are two occasions when the first word may not be capitalized. The first is when the student is seeking the English translation of a word (by typing the word followed by a question mark). If the word is not a noun, then it may or may not be capitalized. If the word is a noun then it must be capitalized. Students may type Hilfe (help) with or without a capital, because we believe help programs should be tolerant.

The Syntax Processor

The grammar for Spion has undergone little change since the first version (Sanders and Sanders 1982). The initial grammar was developed as an attribute grammar and converted readily into a definite clause grammar. One change is
that adverbs have been eliminated from the grammar. Since adverbs were used only to specify directions, as in *Gehen Sie nordwärts!* (Go north), this does not constitute a significant change. These commands were disallowed because, though typical of commercial adventure games, they did not seem natural to the situations of *Spion*.

**The Semantics Processor**

Each syntactically correct sentence is sent to the semantics processor to determine whether the sentence can be considered correct in any game context. In particular, it checks the following:

1. Relations between verbs and direct objects. Transitive verbs can be used only with an object defined for that verb within the game. The small number of objects in *Spion* allows this check to be accomplished through an exhaustive search of all valid combinations in the data base. As the game grows more complex, features of verbs and objects will have to be used.

2. Relations among verbs, direct, and indirect objects. When a ditransitive verb is used, the relationship between its objects must make sense. For example, *Geben Sie Max die Information!* (Give Max the information) is correct, since 'Max' can be the receiver for the verb 'give' and information can be given. On the other hand *Geben Sie dem Bus die Information!* (Give the bus the information) would not be valid since (in *Spion*’s world) a bus cannot be given anything.

3. Relations between verbs and prepositional phrases. Each verb may be used with only certain prepositions and certain objects of those prepositions. Again, a complete data base of valid combinations is maintained.

4. Relations between prepositions and their objects. Just as each transitive verb makes sense only with certain objects, only certain objects may be used with each preposition.

5. Relations between adjectives and nouns. Adjective/noun combinations must make sense. This is very simple in *Spion* since all objects have a small list of properties that could be identified by an adjective. No metaphorical language is allowed nor are there any subtleties to contend with.

The following sentences will fail the semantics test:

*Rufen Sie den Bus an!* (Call the bus)

*Lesen Sie das Cafe!* (Read the cafe)
Essen Sie den Buchladen! (Eat the bookstore)
Geben Sie Max Frieda! (Give Max Frieda)

The following sentences will pass the semantics test:
Gehen Sie auf die Straße hinaus! (Go out into the street)
Lesen Sie das Buch! (Read the book)
Essen Sie das Buch! (Eat the book)
Kaufen Sie das Buch! (Buy the book)
Geben Sie Frieda das Buch! (Give Frieda the book)

The Game Processor
A semantically valid sentence may or may not make sense in any given game context. It is the responsibility of the game processor to perform the necessary context checks of the current game state and to effect any appropriate state transitions. There are six different locations or scenes that the state of the game can take (all except the South Pole are actual locations in Berlin):
1. Flughafen (Airport), the initial scene of the game. Once Robotky leaves the airport by a means other than the airplane, he cannot return. From the airport, he can take the airplane to the South Pole, or take the bus to the City.
2. Südpol (South Pole) may be reached only from the airport. Once here, Robotky can only go back to the airport in West Berlin. If Robotky is not sent back to the airport within three turns, he freezes and the game is lost.
3. Kudamm, or Kurfürstendamm, is the main street of West Berlin. From there Robotky can go into a bookstore or to the train station at the zoo. The trip to the zoo may be on foot or by bus. Once he leaves the Kudamm, he may not return.
4. Buchladen (Bookstore). At the bookstore Robotky must buy a book in order to win the game. The book will contain a list that must be given to Frieda before she will give Robotky the information. From the bookstore, the only place to go is back into the street (Kudamm).
5. Bahnhof (train station). This station (across from West Berlin’s municipal zoo) is where Max will be in order to receive the information. Any attempt to give Max anything before getting the information will result in Robotky’s arrest and loss of the game.
6. Tor (Gate). The Brandenburg Gate, in East Berlin, is where Robotky finds Frieda. If he has the list, he may exchange it for the information for Max.
Command execution is essentially driven by the verbs of the scenario. These verbs are as follows:

- **gehen** (go) To change from one scene to another.
- **nehmen** (take) Used to take the bus or airplane. The price of train or bus fare is deducted from Robotky’s available funds.
- **fahren** (drive) Nothing can be driven in *Spion*, and the idiom ‘fahren mit’ is not recognized.
- **anrufen** (call) Included because of its use in *Pilot-Spion*. There are no phones in *Spion’s* world; if the student tells Robotky to call someone, Robotky answers that he sees no phone booth.
- **lesen** (read) There are newspapers, a slip of paper, books, etc., all of which can be read.
- **kaufen** (buy) Robotky can buy either a book or a newspaper. The price of these is deducted from Robotky’s funds.
- **geben** (give) Robotky may be commanded to give objects to either Frieda or Max. If the game is to be won, Frieda must be given the secret list and Max must be given the information.
- **haben** (have) Used only on the question *Was haben Sie?* (What do you have?). The question will cause Robotky to enumerate all items he possesses.
- **sehen** (see) Used only in the question *Was sehen Sie?* (What do you see?). Robotky will answer by telling where he is and describing the scene.
- **sind** (are) Used only in the questions *Wo sind Sie?* (Where are you?) and *Wer sind Sie?* (Who are you?). The former is equivalent to *Was sehen Sie?* The latter causes Robotky to give his name, rank, and serial number, and is included because students frequently confuse *wer* (who) with *wo* (where).
- **essen** (eat) Robotky, being a robot, is capable of eating most small items in the game. It is never helpful to do so.

The Dictionary
As mentioned above, there are two dictionaries. The first contains all words the program knows to exist. The other contains syntactic information about each of its entries, as follows.

<table>
<thead>
<tr>
<th>WORD TYPE</th>
<th>FEATURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjective</td>
<td>Case, Gender, Strength, Spelling</td>
</tr>
<tr>
<td>Article</td>
<td>Number, Gender, Case, Strength, Spelling</td>
</tr>
</tbody>
</table>
Future Work

*Spion* has been tested with a small group (8) of third-year German students at Miami University, but is not likely to become a standard teaching tool. Although the program performed satisfactorily and student comment was positive, the simplicity of the game (in comparison with the unintelligent *Pilot-Spion*) is a disadvantage in practical use. Rather than further developing *Spion* in the direction of the larger scenarios of *Pilot-Spion*, however, we intend to expand *Spion*'s grammar for a larger syntactic and semantic parser and to write a new adventure game plot for it. Thus *Spion* will remain our research prototype, *Pilot-Spion* our pedagogical prototype.

Spion Source Code and Disks

Arity PROLOG source code for the entire *Spion* program as currently implemented is available at no charge as Technical Report #WSU-CS-87-4 from the Department of Computer Science, Wright State University, Dayton, OH 45435. The program can be compiled by the Arity compiler and will run as a stand-alone program in 640K bytes of memory.

Program disks for either *Spion* (AI version) or *Pilot-Spion* (non-AI version, requires IBM-compatible with 128K bytes of memory) are also available at no charge. Send requests to Ruth Sanders, including a blank disk for each program you wish as well as a self-addressed mailing label.
References


Authors' Biodata

Steven R. Molla (M.S., Wright State) implemented the C-PROLOG version of Spion on a Vax timesharing computer as a master's thesis project. He is now a software engineer at Logicon, Inc.

Alton F. Sanders (Ph.D., SUNY at Stony Brook) and Ruth H. Sanders (Ph.D., SUNY at Stony Brook) have worked together on several projects combining artificial intelligence and CALL for German. Alton Sanders is Associate Professor of Computer Science at Wright State University; Ruth Sanders is Associate Professor of German at Miami University.

Authors' Addresses

Steven R. Molla
Logicon, Inc.
1430 Oak St., Suite 101
Dayton, OH 45430

Alton F. Sanders
Department of Computer Science
Wright State University
Dayton, OH 45435.

Ruth H. Sanders
Department of German, Russian and East Asian
Miami University
Oxford, OH 45056