HISTORY OF AN AI SPY GAME: SPION\textsuperscript{1}

Ruth H. Sanders & Alton F. Sanders

Overview

Spion is a German language adventure game for college language students, first programmed by us in 1981. We intended to use its parser as a platform for the development of more extensive and more sophisticated tutorial German parsers. While Spion and its daughter versions were indeed used by many college students of German, the program’s phrase structure based syntactic element and its lexical-semantic element ultimately proved impractical for further development. More capacious machines, more flexible programming languages, and changed goals supersede Spion’s parser. Syncheck, our subsequent comprehensive syntactic parser of German, has little in common with Spion. Yet the experience gained from several years of working with Spion, discussed in this paper, has proved illustrative of certain problems and solutions for interdisciplinary teams working on AI applications for language learning.
The Rise of the Personal Computer

In the fall of 1981, microcomputers were still a novel idea, and adventure games, strictly verbal puzzles without action graphics, were a kind of underground amusement for computer users, especially young ones. The computer world was intrigued that as much as 64K—the kind of storage capacity previously associated with computers taking up a roomful of space—could be packed into a desktop machine, and a few programming languages were becoming available for the little machines.

In the academic world these small wonders were causing a revolution: computing power was extending to those who had previously had no access to mainframe computer accounts, and almost overnight, the word computer began to mean, not a behemoth in a special building, but a piece of single-user equipment in a laboratory booth or even on a student desk.

The microcomputer market was producing scores of brands, many named after fruit, their makers hoping to emulate the Apple computer in success, if not in design or in operating system. IBM still meant a mainframe, for Big Blue was not yet a major player in microcomputing. Floppy disks were eight inches in diameter, and, except for the Apple, the commonest operating system was CP/M. Computer hobbyists liked to build their own machines (called homebrew) from mail order components. Many a university computer lab director or systems operator put together a micro lab the same way, as the new manufacturers could not yet be relied upon as long-term vendors. But one way or another, the age of microcomputing had begun, and we, along with thousands of other university researchers, began to explore the capacity of the new desktop marvels.

With access to a 64K microcomputer and a license for the programming language PL/1, we (a linguist and a computer scientist) began a collaboration to research natural language parsing techniques for German using microcomputers. We mapped out a multiyear plan to design first a small syntactic and semantic parser, then larger and larger systems, culminating in a comprehensive parser.

"Spion" (“spy” in German), a parser based adventure game in German, was to be the first program in the series.
The Initial Plan

Designing a Game for Language Learners

Our programming language for the first version of Spion was Digital Research's implementation of PL/1 running under the CP/M operating system on a homebrew Z80 machine (Sanders and Sanders 1982). The computer had 64K of main memory and two 8 inch floppy disks capable of holding 128K bytes each. Our original intent was to use the application of beginning language learning as a convenient vehicle for studying the processing of natural language. Our research paradigm, in pseudocode, may be stated as follows:

create simple model of language
implement adventure game using that model
do forever
    allow students to use game
    extend game based on feedback from users
    repeat
        try to extend model to allow for game extensions
        if successful then implement new game
        else reduce complexity of game extensions
    until new game implemented

One general approach to language processing is to have some sort of dialogue between a computer and a person. This was the approach taken by Winograd in his program, SHRDLU, which assembles blocks in response to natural language commands. The difficulty of such a model is that generally humans and machines don't have much to talk about. Hence, there is not generally a large community of users to test a working system thoroughly. One of our colleagues, observing that his daughter spent hours at the keyboard playing one of the early adventure games, suggested that an adventure game format might be a useful vehicle for language processing. At first we considered it impractical to devise a really interesting adventure game (that is, rich and complex, with lots of language) using implementable models of language comprehension. The state of the art in parsing at that time required mainframe capacity, and many now common techniques for representation of syntactic and semantic models were as yet not developed. Commercial adventure games available for small machines used a
dehydrated form of language unacceptable to us: "Go boat" and "Open door" are samples of the kind of verb-object fragments, lacking function words, then used in adventure games. Moreover, we were concerned that our research would be thought frivolous if we concentrated on games without some higher goal.

By aiming our adventure game at language learners, we addressed both of these problems. We could create a small microworld using closed world semantics for simplicity. The microworld could be accessed using the same type of relatively simple language used by second year college language students. Moreover, as part of the language learning process, students would be likely to use the system extensively. Because as learners they were uncertain in their grasp of the language, they would have to play the game over and over to "get it," thus serving as meticulous bug detectors for us. We hoped the intrinsic interest of the game would repay them for their efforts, and that their repetitive play would teach them German as effectively as an equally repetitive, but less absorbing, drill program.

Within these constraints, we were able to use the PL/1 language to develop a low level, phrase-structure based syntactic element to represent the German imperative and interrogative forms expected from students, and a simple lexical semantics to represent the knowledge base underlying the game world. By requiring correct input from students, moreover, we avoided having to develop error detection and processing mechanisms.

The Game

A syntactic parser and a semantic network are the basis of Spion, whose plot concerns spies in the East and West Berlin of preunification Germany. Finally completed in 1987 after several interim versions, Spion has been used by hundreds of German students at Miami University of Ohio and elsewhere either in its AI version or its nonintelligent variant, Pilot-Spion.

To play the game, students communicate by means of written commands or requests to the spy Robotky, who describes events and objects (with place names and locations of the real Berlin). The player must question Robotky for information, move him to
ich bin jetzt in der Stadt Berlin-West, an der Ecke Kurfürstendamm-Fasanenstraße. Ich sehe eine Bushaltestelle mit Zeichen: BAHNHOF ZOO. Auf der Straße steht ein großes Plakat: LESEN MACHT SPASS. Ich sehe auch einen kleinen Buchladen. Was soll ich tun?

Figure 1. A screen from Spion with a message from the spy agent Robotky. The syntactic/semantic processor accepts free input, so the player is not constrained to respond by preprogrammed choices. In English, Robotky says: “I am in West Berlin on the corner of the Kurfürstendamm and Fasanenstrasse. I see a bus stop that says ‘Zoo Railroad Station.’ In the street is a large sign reading ‘Reading is fun.’ I also see a small bookstore. What should I do?”
The Research Environment

Evolving Goals in an Interdisciplinary Team

Academic research is seldom carried out in a single plan to goal manner. Nor are the principal researchers usually the only people involved in the research activities. Certainly, in the case of Spion, there were many associated trial efforts and many other people connected indirectly with the project. In 1980, at Wright State University, a group of researchers that included the authors formed a joint research group entitled FLITE (for Foreign Language Instructional Technology Engineering). The research group intended to examine all aspects of the computational problems associated with using computers to assist in language instruction. This included particular applications such as Spion, programming languages suitable for implementing such applications, and architectures suitable for such languages. At each level, there were theoretical and philosophical issues that were also relevant. The group at its largest included five computer scientists, three philosophers, one linguist, one mathematician and several computer science graduate students. Regular weekly seminars with this group provided us with many ideas with which we experimented. Some were partially implemented, some were fully implemented and many more were shelved, perhaps to be considered further later.

The primary purpose of the Spion project originally had been to study models of language understanding by computer. As time went on, however, our focus gradually changed, and applications to CALL became our primary goal. As we became more and more involved with CALL, we viewed the Spion project more as an engineering problem than as basic research. This probably occurred first when we began to polish Spion as a product. Because end users (students) were required by our research paradigm as "testers" of our program, we had to provide a finished, usable product for them. It turned out that this element of our paradigm had a considerable effect on our work.

Producing finished products for users involves additional problems not necessarily directly related to a project's research purpose. Solving these problems can be very time consuming, but is necessary to the success of the project. For example, decisions concerning screen design and what kind of information or help should be available to users were matters of some importance. Things such as printing German characters...
(umlauts, ö) on the screen, and deciding what keys the user must press to get them, or differentiating between lowercase and upper-case letters in the input, may seem trivial to designers who think of themselves as working in higher realms than this. Yet, if these questions are not taken care of the program will look amateurish on the screen no matter how sophisticated the linguistic or computer expertise behind the design. Students who use the program must be asked for their reactions, and these reactions must be listened to; user friendliness is essential, and when users find a program difficult to use, it must be changed no matter how deeply the developers feel that the program is in fact not difficult to use.

These concerns are essentially engineering problems rather than basic research problems. We found them interesting and challenging, and ultimately they captured more of our interest than the original research problem. We redefined ourselves as a CALL or ICALL applications team.

Early Designs

Design Choices to Support Form-Focused Pedagogy

Language processing for language learners poses quite different problems from those posed by other natural language interfaces. The prescriptive nature of language instruction prohibits the acceptance of ill formed sentences (at least for some purposes), even if they are understandable. Taking a strong stand concerning input errors, we required the student to retype an erroneous input sentence, even if the program could determine with a high degree of certainty what the error was. For example, it is easy for the program to detect when a student forgot to end an imperative sentence with an exclamation point, but we chose for pedagogical reasons to require retyping of this and other erroneous sentences, even where the error was a small one. In that sense, Spion retains some of the rigidity that has come to be associated with drill and practice programs.

Design Choices to Accommodate Lexical Limitations

Some design decisions were possible explicitly because of the artificial context provided by the game format. For example, the language used by the fictitious agent, Robotky, was generally much richer and included many more words than we wished to include
in our input parser. Though it sometimes happens in commercial adventure games, too,
that the program claims not to know a word, even when that word has been used
earlier by the agent, we didn't want to do the same, because it is frustrating to the user.
Although we knew that a student was likely to parrot Robotky's words back to him, we
weren't able to make our lexicon large enough to include complete semantic
information on every word used by Robotky. We sidestepped that problem by labeling
certain words as forbidden (*verboten*) to the player, though permissible to Robotky,
presumably because of his higher rank as a spy. A message to that effect in German,
was presented to a player who uses one of these words:

"[Word] ist ein verbotenes Wort: das heißt, verboten für Sie."

This selective restriction may have been equally frustrating to the student user as a
claim not to know a word, but it had, in our view, at least the virtue of being more or
less candid.

**Designing in Anticipation of Hardware and Software Developments**

While working on our first design, we always anticipated that the program ultimately
would run on a machine faster and bigger than our initial Z80. Thus we were
unconcerned when our first version ran slowly, with a noticeable wait from the input to
the response. We also proceeded with an artificial design in order to work around the
lack of memory. The first version of Spion was actually two completely separate
programs, each of which demanded close to the maximum from our small machine.
Because we expected that computers with increased speed and capacity would soon be
available, we assumed that we could join the programs later, when the technology
catched up to us. We were criticized at the time for this assumption and for being in
general too ambitious. In the event, however, it turned out that we had been too
cautious, and that our caution had cost us quite a bit of time.

We greatly underestimated the rapidity of technology development. Within only a year
or two of our initial work, commonly available in microcomputers provided all the
capacity we needed for a full version of Spion; in the meantime we had spent a great
deal of time and energy devising memory allocation and speed up mechanisms that
were unnecessary almost as soon as we completed them.
Moreover, in 1981, microcomputers were (by today’s standards) small and slow, but the unsettled computer market was an even greater impediment to our work. We spent quite some time and effort selecting a computer language for the Spion project, eventually settling on PL/1 for reasons having to do not with the nature of the language but with the practicality of its compiler.

It was not yet clear which operating system or which disk format would become dominant, and there were very few reliable compilers that produced acceptable code for any of the systems. We experimented with a number of languages, compilers and interpreters. We needed to be able to provide a compiled version or a version with a public domain interpreter, since we intended to give away our program and didn't want to have to deal with distribution fees or licensing. Additionally, we wanted a mature and reliable product. Our preference would have been some variant of LISP, but we were unable to locate an implementation that performed satisfactorily, even if we had been willing to work with an interpreter. We experimented with Pascal, LISP, Forth, and PL/1 under CP/M. Digital Research's PL/1 compiler was clearly the best choice of the ones we tested. We guessed (incorrectly) that Digital Research would port that compiler to whatever system ultimately won in the marketplace. PL/1 disappointed us on this and several other scores. We abandoned all of the initial program fairly early on.

Later Designs

Research versus Engineering: Developing a Nonintelligent Spion

The second and most ambitious version of Spion's design was started, but never finished, in TLC Lisp. Here our model was categorial grammar, taking some ideas from John Sowa.³ Our closed world was represented by a semantic network modelled after Sowa's. As syntactic reductions occurred via the categorial grammar's cancellation mechanism, a graph transformation was constructed in the form of a pair of "pattern graphs." Well formed sentences would produce well formed transformations that would alter the semantic network to reflect the state change required by the sentence. In a categorial grammar, each word is represented as a series of symbols indicating which kinds of other words it is allowed to precede or to follow: nouns usually follow articles
or adjectives, and so forth. The various symbols are "canceled" like an algebra equation during parsing. What is left is the parsing "solution," which then is sent to the semantic element as shorthand for what should happen in the environment as a result of the sentence: if a newspaper is read, its contents are to be displayed to the player; if an apple is eaten, it is no longer in the scene description, and so on. Actually, since the output buffer was part of the state, even ill formed sentences produced a state change, but no graph transformations were involved. The network modification was carried out by mapping one of the pattern graphs onto the network and replacing the image of that pattern within the network with a copy of the other pattern graph. In effect, we used a categorial grammar for our syntactic model and a graph grammar for our semantic model.

This design was very flexible and, of all the designs we considered, offered the most promise as a research vehicle. It had the potential of limited concurrent processing (one of our interests) and was straightforward to extend. Unfortunately, seemingly it would be years before a genuinely usable product could be produced with this design. There were many problems to be solved; some were technical problems, but others appeared to be basic research questions. That was the moment of truth with regard to the research versus engineering issues. We ultimately chose on the side of engineering: we abandoned the above design for a simpler one that could be implemented and used effectively.

While the second version was underway, a separate, nonintelligent version of the game was programmed in the authoring language PILOT and named "Pilot-Spion" (Sanders 1984). In that version, students were given at each decision point a list of choices and asked to write a request based on one of the alternatives. This is illustrated by the screen from Pilot-Spion in Figure 2, which can be contrasted with the screen from Spion in Figure 1. Pilot-Spion provides a kind of multiple choice, although students were still required to write requests (rather than simply selecting an alternative). There was no parsing of these written productions and no updating of a semantic network, although a simple list of variables and switches was altered to reflect changes in the scene, so that apples disappear when eaten and the like. The natural language element had been transmuted into simple pattern matching, feasible because of the limited choices given the student. While Pilot-Spion was not an AI program and had no parser, the game
itself was a much more complex version of the simple game of Spion. It featured eighteen locations (geographical stations such as a candy shop or a border crossing, where actions take place), while Spion had used only six. Pilot-Spion seemed to challenge second year students, taking them an average of two and a half hours to complete successfully. Used in second year German at Miami University every year since 1984, this computationally simple game showed that the adventure game format could prove quite useful as a learning tool. It has been provided free of charge on request to about fifty universities in the U.S. and Europe.

The Final Design

Incorporating User Findings into Intelligent Spion

Experience with the unfinished LISP version and the nonintelligent Pilot-Spion caused a return to the simplicity of the original design with only a small portion of the second design included. In effect, Pilot-Spion became the initial version in the terms of our research paradigm, the one that would provide us with user reaction on which we could build for subsequent versions. We did in fact learn a great deal about the needs of the user interface, and incorporated what we learned into our next version. A graduate student, Steven Molla, programmed that version in C-Prolog on a Vax 785; it became the first all-in-one complete intelligent version of Spion (Molla 1987). We then ported that program to an IBM AT and modified it according to the experience gained from Pilot-Spion. That program runs in a stand-alone compiled version and (on a 16MHz 386 machine), produces a response to a user's input sentence with no perceptual delay (Molla, Sanders and Sanders 1988).

The final design had a naive language model, its syntax limited to simple imperatives and questions, its semantics limited to a small vocabulary and simple relationships. It was far simpler than it had to be, since the marketplace soon provided more than enough capacity for a much richer language element and game than the one we had designed. By this time, however, we had learned that, from a user's point of view, a

Figure 2. A screen from Pilot-Spion. The player formulates a command from the alternatives given by the spy agent Robotky. This version lacks umlauts and the ß. In English, Robotky says: “I am in West Berlin, on the corner of Kurfuerstendamm and Fasanenstrasse. There is a lot of traffic here, and many people. I see a bus stop with a sign: ‘Zoo Train Station.’ On the street is a large poster: ‘Reading is fun.’ A large hotel—the Hotel Kempinsky—is on the Kurfuerstendamm, an I also see a tiny chocolate shop and a small bookstore. There is a lot to do here! Shall I take the bus, go into the hotel, go to the chocolate shop or visit the bookstore?

more complex game was far more interesting than a complex language model. Our nonintelligent version, Pilot-Spion, supplanted the intelligent version for classroom use.

Ironically, the six location Spion seems to be much more difficult for second year German students than does the eighteen location Pilot-Spion, which requires full sentence responses that are partially copied from a multiple-choice set. As was illustrated in Figure 1, the player of Spion is presented with the situation and left to his or her own devices in “talking” to Robotky. In contrast, Pilot-Spion presents several clear choices, as was shown in Figure 2, and these choices include the vocabulary and partial sentence structure for a response; the user only needs to formulate one choice correctly to advance the game. Thus, when Pilot-Spion's Robotky asks (in German), "Should I take the bus, go to the hotel [etc.],” the student chooses from these alternatives and types, "Go to the hotel" (or some other choice).
Because students appeared to have difficulty with freely created commands in Spion, we revised the program to provide a help facility that might be consulted at any time for suggestions on what to do. In practice, this resulted in student players' using the help option at every turn of play, so that the freedom of input we had worked so hard to make available was never actually used by the students. It appears that the open-ended Spion tends to intimidate many early language learners, whereas the delimited possibilities of Pilot-Spion give them confidence.

Future Designs

Although we believe our original paradigm had merit for studying language models, we are now more interested in new products associated with language learning. Hence, although we may well produce another product with a game format, we are unlikely to extend Spion's microworld for yet another version of Spion.

We intended at the outset to use Spion's parser as the basis for our next project, called Syncheck, a parser based writing aid whose input would be the flawed compositions of intermediate and advanced German-language learners (Sanders and Sanders 1989). During Spion's development, microcomputer capacity advanced yet again, and we began working in Prolog because it offered a formalism that both of us found congenial. This decision was to facilitate our work greatly. Earlier, transformation of a single grammar clause into computer code using PL/1 might take the two of us several hours of work together, because the linguist understood only vaguely the language of the computer scientist and vice versa. Prolog, on the other hand, with its user-friendly logic formalism, provided in effect an interlanguage for us. Large pieces of the linguist's formal grammar could be coded into Prolog with very little change, and the program optimized for computational efficiency afterwards. The comprehensive parser we designed for Syncheck proved so different from the limited parser of Spion that virtually nothing of Spion survived in the later project.
Conclusions

Although we began our collaboration with the idea of basic research into natural language processing techniques, we found that the requirements of producing finished products demanded an increasing amount of our time and effort. It is clear why researchers who design laboratory prototypes do not generally provide user ready versions of their programs. The challenges of final implementation are sometimes wholly unrelated to the challenges of design. However, from our point of view, these engineering aspects of user products are interesting and satisfying, and we intend to continue this line of work.

One of these engineering challenges has been "massaging" the hardware to provide the effects we have sought. In our case, the technology appeared to provide capacity faster than we could redesign to compensate for its lack originally. However, we find looking into the future no easier now than it was in 1981, and we cannot be sure of avoiding the error of underestimating technology a second time.

Additionally, we have experienced at first hand the principle that a more sophisticated program (Spion) is not necessarily a more effective program in practical use than a simple program (Pilot-Spion). Unquestionably, artificial intelligence and parsing have much to offer to foreign language teaching programs; but computational complexity in itself is no guarantee of usefulness in application.

Notes

1 This article, in slightly different form, appeared under the same title in Intelligent Language Tutors: Theory Shaping Technology, edited by V. Melissa Holland, Jonathan D. Kaplan, and Michelle R. Sams. Mahwah, NJ: Lawrence Erlbaum, 1995. It is reprinted here by kind permission of the publishers.

AUTHORS’ BIODATA

Alton F. Sanders was born in Florida, studied political science and mathematics as an undergraduate. M.S., Ph.D. in Computer Science, State University of New York at Stony Brook. He has taught at Wright State University in Dayton, Ohio, and since 1990 has been Professor of Systems Analysis at Miami University of Ohio. He has published on artificial intelligence, machine languages, and natural language parsing. He is co-developer of SPION and SYNCHECK, both parsing programs for German.

Ruth H. Sanders was born in Michigan, holds a B.A. in German, University of Michigan; M.A., Ph.D. in Germanic Languages and Literatures, State University of New York at Stony Brook. She is Professor of German at Miami University of Ohio, where she has taught German language and linguistics since 1976. Publications on CALL include artificial intelligence and parsing; co-developer of the German parsing programs SPION and SYNCHECK.