

Antonia J. Jones

Artificial Intelligence in Computer Aided Instruction Shells

by

Antonia J. Jones

Reprinted from: Chapter 16 of **Tutoring and Monitoring Facilities for European Open Learning**. Eds. J Whiting and D A Bell. North-Holland, ISBN 0-444-702717, 1987.

Communicated by: **Jean A Millar**.

1 June 1989

Summary.

In this document we examine possible applications of Artificial Intelligence (AI) to Computer Aided Instruction (CAI) Shells. We focus on applications which have been, or could be, implemented at the present time, applications which are feasible but require further research and considerable programming effort, and applications which are conceivable but require substantive advances in the present state of AI.

Contents.

History lesson.

Setting the stage.

Away from pre-structured interactions.

Lessons from Knowledge Engineering.

Limited semantic parsing.

Research lines for Student Modelling.

Appendix - A brief description of CBESS.

Antonia J. Jones

Artificial Intelligence in Computer Aided Instruction Shells

by

Antonia J. Jones

History Lesson.

The brief history of CAI has parallels with the history of AI itself.

In the early days the apparent seductive power of computers lulled research workers in Artificial Intelligence into a false sense of optimism - the feeling thirty years ago appears to have been that the construction of machines which displayed many of the characteristics normally associated with human intelligence was just around the corner. We now know better. Human intelligence emerges from a complex interplay between immensely rich input/output streams and a massively parallel structure of loosely coupled computational processes; the whole of which, to this day, is still poorly understood. Moreover, even granted the power of present day computers, there are many quite simple human tasks which elude the capability of machines. Still, in some areas considerable progress has been made - the most striking being where human and machine co-operate, each contributing their own special capabilities.

We illustrate this last point in some detail by giving an account, due to Holte, of one striking success in human-machine co-operation, the program EURISKO written by Lenat. In EURISKO activity is governed by a set of heuristics, initially provided by Lenat (and/or the user), which suggest ways of accomplishing certain goals, and tasks/operations which might lead to interesting results. The system spends most of its time gathering information (there are heuristics which guide this process) about concepts which promise to be interesting. The heuristically guided creation of new concepts is a kind of evolution by mutation. Much of the research effort in designing EURISKO went into the design of the representation language.

EURISKO has been applied to many unrelated domains, and in each it has been moderately successful. One notable success was in a wargame called "Traveller - Trillion Credit Squadron" (TCS for short). The object is to design a fleet of hypothetical ships which is superior to anyone else's fleet. Fleets are compared by engaging in battle, but the outcome of these battles is determined entirely by the makeup of the fleet and not by the tactics used during the battle. The fleet design is constrained in several ways: each weapon, unit of armour, etc. has an associated cost and the total cost of the fleet must not exceed one trillion credits; there are also limits on the number of certain types of weapons/armour which can be carried on a single ship; and on and on - the rule book which outlines these constraints is 100 pages long.

Lenat had never played the game, but thought it would be a suitable domain for EURISKO ("because the battles were tactically trivial, and because of the thickness of the rule book"); he entered the facts and constraints from the rule book, heuristics for modifying (mutating) fleets and ships, for analyzing the outcomes of simulated battles, and so on, and started EURISKO off with a fleet of Lenat's own mediocre design. EURISKO proceeded to evolve progressively better fleets ("better" as measured in simulated battles against the previous fleets, not against a standard), and also postulated useful properties of fleets and individual ships. These properties were used as heuristics to guide subsequent search. One surprising but extremely valuable and general heuristic which EURISKO found for designing fleets was that "whenever possible design fleets and individual ships which are almost, but not quite, extreme in their features", meaning, for instance, that fleets should contain almost the maximum number of ships allowed, and ships which are not heavily armoured should have almost no armour.

Lenat ran EURISKO every night for a month: he would come in each morning and sift through the new heuristics

it had proposed, modifying the "interestingness"/worth values which EURISKO had assigned to them according to his own judgment. He estimates that the final fleet is 60% his and 40% EURISKO's. The final fleet was extremely unusual, and won the annual National (U.S.) tournament easily. The TCS governing body subsequently made a large number of rule changes, all aimed at eliminating the kind of unusual fleet EURISKO had designed. Their strategy backfired: the rule changes undermined the expertise which the humans had slowly acquired over several years of fixed rules, whereas EURISKO relatively quickly (only two weeks of evenings) found a new legal, powerful fleet design. Many of the more general heuristics it had discovered the previous year were still valid. EURISKO won the next annual tournament, and was subsequently banned from all further tournaments.

There is an interesting moral for CAI shell construction which can be drawn from the account of EURISKO. It is that the really intransigent problems of AI can often be sidestepped by using human-machine co-operation rather than pure dependence upon the machine to do all the work.

In CAI the prevailing climate of opinion at the onset of the 'IT revolution' seemed to be that if we simply gave every school a microcomputer the problem of creating an 'IT literate' population and half the problems of the over extended teacher would be resolved. What actually happened was rather different.

We all know now that writing decent software is a very professional and labour intensive activity, that teachers (as a group) have neither the time nor the skill to produce complex packages, that packages produced using the special facilities of one microcomputer are rarely easily ported to next year's most popular model, and that packages produced to meet one group's needs do not necessarily meet another's.

Not to mince words, the results in the UK were appalling. There was a proliferation of poorly written 'buggy' software produced by hard pressed teachers. Not least because there were few software houses that saw educational software as a market worth supporting in its present state, and they were largely right. Teachers who had been presented with microcomputer systems were rarely given budgets for software acquisition which reflected a sane ratio of software to hardware cost. They did the only thing possible, they wrote their own.

On the basis of this experience we can safely conclude that writing one-off programs in BASIC for a particular microcomputer does not constitute a sensible approach to CAI. Nevertheless there is a largish, reasonably solid, pool of such material and, subject to the qualification 'let the buyer beware', it should not be excluded from consideration in preparing a particular course.

The alternative is expensive, very expensive judging by the lessons of history, but enjoys the advantage of huge economies of scale if applied across the European base proposed by DELTA. Moreover, what works for Europe will be a very marketable product on a global basis.

Even prior to the advent of the microcomputer another, in our view far more substantial, approach to CAI was being taken. In the United States William Norris, Chairman of Control Data Corporation, was developing PLATO, a billion dollar CAI project based on networked systems. Nowadays it is fashionable to refer to PLATO as a 'dinosaur' because of its immense size. Moving sharply in the other direction, at Stanford, considerable effort is presently directed to writing one-off CAI packages for the Apple Macintosh. It may be that the scale of the PLATO initiative is open to criticism, however we believe that the concept of an integrated networked system is, in the light of technological developments, much closer to meeting the long term requirements of CAI. In particular it is noteworthy that many of the second generation CAI package-generating tools we discuss below, such as Authoring systems and languages, are direct lineal descendants of PLATO. Norris was a socially caring visionary and PLATO the forerunner of DELTA.

The significance of AI to CAI has long been recognised, see for example the survey article by Self, 1977. By 1977 several AI techniques had been borrowed and put to work in subject-specific CAI systems. Yet some ten years later the situation is not so very much different. CAI shells using AI techniques are exceedingly rare.

Setting the stage.

What is a typical university CAI environment today and what is it likely to become in the future?

Typically we may suppose a UNIX mini-computer host networked to an intelligent terminal, for example an IBM PC, an Apple Macintosh or (ideally) a Sun Workstation. The network may be a LAN or ETHERNET; the former being most likely, the latter being most desirable. The idea is to have local intelligence linked over a network to a large fileserver. The more local processing power the better. Obviously a large bitmapped screen with fast graphics, preferably colour, is optimal. In the short run hardware akin to the Sun should be available at much lower cost.

For DELTA the outline specification of the student workstation includes 100 - 500 Mbytes of onboard RAM and 3-5 Gbytes of predominantly optical (read/write/erase) peripheral storage - i.e. at least two orders of magnitude greater than commercially available personal computers on sale in 1986. The network requires high bandwidth if graphics or video data are to be downloaded to the terminal. Even granted known developments in network bandwidth and local digital storage, for the foreseeable future the video data peripheral is likely to be a necessary adjunct to the workstation.

The software environment comprises the three essential ingredients of a CAI shell:

1. An **Authoring system**. This is a sub-system designed to enable the teacher to produce courseware with the minimum of effort and expertise outside of special subject knowledge.
2. A **Delivery system**. This is that part of the system software concerned with the presentation of the courseware to the student.
3. A **Course Management system**. This monitors the student interaction and produces suitably structured information to the (human) course manager relating to system and student performance.

One can envisage a two stage Authoring process in which the first stage involves programs running on the workstation. Here the author inputs his specialist knowledge. The second stage could run either on the host or the workstation and would consist of compiling the knowledge base into a data structure usable by the Delivery system - i.e. the production of the actual courseware. The Delivery system must run on the workstation and, where the workstation is not currently networked, must log course management data for later retrieval by the host. The Management system can run on the host.

For the system to be workable the source code should be available on the host, so that new workstations can be accommodated by the inclusion of new Authoring and Delivery drivers. Software protection in the context of an evolutionary system, comparable to UNIX itself, is not a feasible option. Far more to the point is the implementation of a comprehensive software-development management system to keep track of the inevitable rapid changes originating from authorised research and development centres.

An example of a CAI shell meeting most of these requirements is CBESS, created by Richard Brandt at Utah. Brunel University is fortunate in being the First European beta test site for CBESS and some aspects of CBESS which use simple AI techniques are described in the Appendix.

Away from pre-structured interactions.

" Since almost all of AI has some potential relevance to CAI it is necessary to delimit carefully the work to be described." (Self, 1977).

One of the principal constraints on PLATO was that using dumb terminals on a multi-user system seriously restricted the processing time per-user. As the local intelligence of workstations increases much more scope will exist for the application of AI techniques in both the authoring and delivery of Computer Aided Instruction. We may, in the first instance, identify three areas of potential gain in the possibility of using Artificial Intelligence

techniques in CAI.

The principal drawback of existing CAI software is in the lack of flexibility in routes through the instructional material and in the variety of student input with which the system may reasonably be expected to cope.

At present most CAI shells require the author to anticipate all possible student routes through the material. This places the burden of dealing with the consequent combinatoric explosion squarely upon the author. The result, not unexpectedly, is that the author (of necessity) has to sharply restrict possible student response and consequent action. If the same knowledge base is to be taught by a different technique then the entire Authoring job has to be repeated.

Thus the first lesson of conventional AI is to separate the knowledge to be taught from the precise mechanism of delivery. This brings the twofold advantage that a single knowledge base can be used to teach the same material in different ways and secondly (because of the way in which Expert Systems actually work) the routes through the material can, in principle be less structured and more flexibly dependent upon the student response.

The second area in which there are potential gains is in enhanced understanding for the system of more complex student input. In plain language teaching should be a dialogue; the student should be able to ask, probably by keyboard entry, a 'reasonable question' and get an 'intelligent response'. The full attainment of this goal is currently beyond AI - it lies in the area known as Language Understanding. However, if we replace 'reasonable question' by 'grammatically simple, syntactically correct, question which relates specifically to the material under discussion' then we might hope to achieve something along these lines. This is very much an AI research issue at the present time, since it involves, albeit at a rather restricted level, questions of semantics. We discuss one possible approach in a later section.

However, we must not forget that improvements on one side (Delivery) may also have implications or impose requirements on the other (Authoring). There is a general requirement to minimise the author load outside of his special area of expertise and undoubtedly AI will have a significant role to play in this direction. If we had a good general purpose Language Understanding Program for limited domains of discourse this in turn would require the author to input the knowledge being taught in a form from which the Language Understanding software could, given the question, construct an answer. To what extent could this need be automated, and is the goal of automating it compatible with current content-free techniques for creating and compiling expert system knowledge bases? We examine some aspects of these questions in the next section.

Finally there is the question of student modelling. The goal here is to monitor a student's progress, accommodate to differing learning strategies, be aware of error and misconception, and offer appropriate advice in relation to the subject domain. To accomplish this new meta-models involving theories of learning and cognition, and domain related epistemologies are required. An example exhibiting some of these characteristics is Digital Equipment's IVIS system which is capable of analysing the learning styles and paths of particular students. We discuss one possible general approach to student modelling in the final section.

Lessons from Knowledge Engineering.

Self, 1974, indicated three major knowledge components in a CAI system.

- (a) Knowledge of how to teach (which includes knowledge of students in general).
- (b) Knowledge of what is being taught.
- (c) Knowledge of who is being taught (i.e. knowledge of one student in particular).

What has emerged since then can be seen as a continuing debate on what constitutes the knowledge in each area and how it should be represented. There is, of course, the additional requirement to elicit each type of knowledge. Nilsson takes a realistic view of this problem.

"One must remember, however, that computer systems will be incapable of truly flexible dialogues about representations and the concepts to be used in these representations until designers are able to give these systems useful meta-knowledge about representations themselves. Unfortunately, we do not even have a very clear outline yet of a general theory of knowledge representation." (Nilsson, 1982, pp 420.)

An Expert System is a computer program that uses the experience of one or more experts in some specific problem domain and applies their problem-solving expertise to make useful inferences about a particular problem supplied by the user of the system. Knowledge engineering is the process of acquiring knowledge and building an expert system.

Plainly the process of constructing an expert system has features in common with that of constructing a particular piece of courseware. In both cases there is a subject expert from whom knowledge has to be elicited. Unfortunately the construction of courseware which can provide the basis of sophisticated CAI is an intrinsically more subtle task. The final software has to transfer expertise not simply apply it. Early attempts to transport the knowledge of an expert system into another expert system whose expertise was that of good teaching practice were not notably successful, see, for example, the discussion of GUIDON in Yazdani and Narayanan, 1984.

Eliciting problem-solving knowledge from an expert has proved to be a critical problem in building expert systems. A long series of interview, build and test cycles are necessary before a system achieves expert performance. The time required to build an expert system is typically six to twenty-four months. For this reason considerable effort is currently directed to the construction of systems to expedite the transfer of expertise to a knowledge base. We briefly focus on one such program, ETS (Expertise Transfer System) being developed by the Artificial Intelligence Center of Boeing Computer Services (Boose, 1984). ETS runs in Interlisp-D on a Xerox 1100 Dolphin Lisp Machine, using high- resolution bit-map windows, pop-up menus, and mouse interaction capability.

The goal of ETS is to provide a tool set to significantly shorten the knowledge acquisition process, and to help improve the quality of the elicited problem-solving knowledge. To do, this ETS automatically interviews the expert, and helps construct and analyse an initial set of heuristics and parameters for the problem. ETS provides the expert with knowledge analysis methods and a mechanism for exploring conflicts in the way that different problem cases are handled. With the results from ETS, the knowledge engineering team does not need to begin from scratch when interviewing the expert. They have basic problem vocabulary, important problem traits, an implication hierarchy of these traits, and conflict areas where discussions may begin. In depth analysis of these knowledge bases eliminates most of the initial interviewing process, which is typically the most painful, time consuming part of knowledge acquisition.

ETS employs clinical psychotherapeutic interviewing methods originally developed by George Kelly (Personal Construct Theory - Kelly, 1955), who was interested in aiding people to categorise experiences and classify their environment. Certain techniques for use in psychotherapy, such as the Repertory Grid Test, were developed by Kelly based on this philosophy. We shall describe these procedures more fully in the section on student modelling.

The output from the automated testing is an entailment graph from which ETS generates two types of heuristic production rules; conclusion rules, and intermediate rules. Each production rule is generated with a belief strength.

Conclusion rules are created from individual ratings in the grid. Each rating has the potential for generating a rule. The expert is asked to rate the relative importance of each construct in terms of its potential importance in solving the a problem. Then ETS employs an empirical algorithm to generate certainty factors for each rule. The algorithm takes into account grid ratings, relative construct importance, and the certainty factor algorithm in the target expert system building tool.

Intermediate rules are based on relations in the entailment graph. For each entailment, one rule is generated. The strength of the rule's certainty factor is based on the relative strength of the entailment. These rules generate intermediate pieces of evidence at a higher conceptual level than those of conclusion rules.

Once the rule have been generated, ETS has enough information to generate a knowledge base for an expert system

building tool based on production rules. These prototypes are then run to test the knowledge for necessity and sufficiency. Manual interviewing and incremental knowledge refinement are still necessary to produce a system that performs at an expert level.

Limited semantic parsing.

In the early 1970's transition network grammars were developed as a model of natural language analysis (Bobrow and Fraser, 1969; Woods, 1970; Woods, 1972).

A basic transition network looks essentially like a non-deterministic finite state machine, except that labels on the arcs may be state names as well as terminal symbols. The interpretation of an arc with a state name as its label is that the state at the end of the arc will be saved on a pushdown stack and the control will jump, without advancing the input pointer, to the state that is the arc label. When a final state is encountered, the pushdown stack is popped by transferring control to the state that is named on the top of the stack. The basic transition network is a generalised pushdown stack automata and accepts the same class of languages, viz. context-free languages - essentially the Backus Naur Form used to describe the syntax of programming languages like Pascal.

A basic transition network can be augmented, to produce a more powerful machine, by adding facilities to each arc. These include arbitrary conditions that must be satisfied in order for the arc to be followed and a set of register setting actions to be executed if the arc is followed. The power of an Augmented Transition Network (ATN) is determined by the facilities added to the arcs but in general can be that of a Turing machine. A detailed discussion of the relationships between transition network grammars and Chomsky's hierarchy can be found in Chou and Fu, 1975. However, it is known that the phrase structured (recursively enumerable) languages accepted by Turing machines cannot be parsed with certainty even for syntax.

One of the earliest and most successful question-answering system using AI ideas from around 1970 was the Lunar sciences natural language information system, commonly known as LUNAR (Woods, 1977). This system had a separate syntax analyser and semantic interpreter. The parser written as an ATN, built a Chomskian deep structure for an input sentence, using the storage facilities of the ATN registers to reorder constituents as necessary. The semantic rules were designed to operate on this tree structure, building data-base queries that reflected the meanings of the question or command that had been parsed.

Although no formal grammar can successfully deal with all the syntactic problems of natural language, existing grammars and parsers can deal with perhaps 90% of all sentences. However, a given sentence may have hundreds or even thousands of possible syntactic analyses, most of which have no plausible meaning. People are not aware of considering and rejecting such possibilities but parsing programs are swamped by meaningless alternatives (Winograd, 1984). Thus the real problem is the need to extract semantics.

Whilst syntactic analysis can be content-free both semantic and pragmatic analysis must depend on context and so these parsers must be constructed on the basis of interaction with the courseware author. Thus there is an identifiable need for research into the automatic production of content-dependent semantic and pragmatic parsers analogous to those emerging for the production of Expert Systems.

Research lines for student Modelling.

Three of the principal philosophies which have emerged for student modelling are: the information processing model, in which student responses are related to a set of 'student protocols'; the overlay model, in which the student's knowledge is regarded as a subset of the knowledge of the embedded domain expert; and genetic graphs, in which nodes represent procedural skills and arcs represent the method by which one skill evolves from another.

We have seen in the discussion of ETS that knowledge elicitation can be enhanced by software procedures based on Kelly's Personal Construct Theory. In the same way it is reasonable to propose that similar dialogical procedures might form the basis for student modelling.

Antonia J. Jones

In the Repertory Grid Test for eliciting role models, Kelly asked his clients to list, compare, and rate role models to derive and analyse character traits. Roles were listed along the top of a grid; Kelly referred to these items as elements. Clients were then asked to compare successive sets of three elements, listing distinguishing characteristics and their opposites down the right hand side of the grid: "What trait distinguishes two of the elements from the other one?". A trait and its opposite represented an internal bipolar scaled dimension or construct, and were respectively called the left and right poles of the construct. The construct was the internal concept represented by the verbal label.

As each construct was elicited, all the elements were rated against it along its bipolar scale. A collection of elements, constructs, and ratings was called a construct grid (Kelly, 1963). Following construction and analysis of the grid, the clinician entered on an interviewing phase to expand on and verify the relationships between constructs pointed out by the grid analysis.

One interviewing technique was known as laddering. This was a method which helped connect the elicited constructs in their superordinate and subordinate relationships. Hinkle, 1965, further developed the idea that constructs have locations in a hierarchy of implications.

More recently, elicitation and analysis of repertory grids has been made available through interactive computer programs. Recent programs (Gaines, 1981) produce graphs which show entailment relations among constructs and elements.

One obvious way to apply this to student modelling is an overlay model using morphisms between entailment structures rather than (say) semantic networks. However it seems likely that automated Personal Construct Theory has rather more mileage for student modelling than this simple suggestion.

On the basis of the evidence so far, it seems unlikely that any one of these approaches, when taken in isolation, will be entirely satisfactory. By any standards, good tutoring is difficult to accomplish and we may expect educational technology to go through a number of evolutionary stages before a development methodology emerges.

Acknowledgment: I am indebted to Robert Holte for permission to include his graphic description of EURISKO.

**Department of Electrical Engineering and Electronics.
Brunel University.
Uxbridge Middlesex UB8 3PH.
U.K.**

References

- Bobrow, D. and Fraser, B. An augmented state transition network analysis procedure. Proc. Int. Joint Conf. Artif. Intell., Washington, D.C., 557-67, 1969.
- Boose, J., A framework for transferring human expertise. Human- Computer Interaction, Ed. G. Salvendy, Elsevier Science Publishers B.V., Amsterdam, 1984.
- Chou, S.M., and Fu, K.S., Transition networks for pattern recognition. Tech. Rept. TR-EE 75-39, School of Electrical Engineering, Purdue Univ., West Lafayette, Ind., Dec. 1975.
- Gaines, B.R. and Shaw, M.L.G. New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in M. Shaw (Ed.), Recent Advances in Personal Construct Technology, Academic Press, New York, 1981.
- Hinkle, D.N. The Change of Personal Constructs from the Viewpoint of a Theory of Implications. Ph.D. Dissertation, Ohio State University, 1965.
- Kelly, G.A., The Psychology of Personal Constructs, Norton, New York, 1955.
- Kelly, G.A., Non-parametric Factor Analysis of Personality Theories. Journal of Individual Psychology, **19**, 1963.
- Nilsson, Principles of Artificial Intelligence, Springer-Verlag, Berlin, 1982.
- Self, J.A., Student Models in Computer Aided Instruction. Int. J. Man-Mach. Studies, **6**, 261-276, 1974.
- Self, J.A., Artificial Intelligence Techniques in Computer Aided Instruction. The Australian Computer Journal, Vol. **9**, No. 3, 118- 127, 1977.
- Winograd, T., Computer Software for Working with Language, Scientific American, September 1984.
- Woods, W.A., Transition network grammars for natural language analysis. CACM, **13** (10), 591-606, 1970.
- Woods, W.A., An experimental parsing system for transition network grammars. BBN Report No. 2382, Computer Science Division, Bolt Beranek and Newman Inc., Cambridge, Mass, May 1972.
- Woods, W.A., Lunar Rocks in Natural English: Explorations in Natural Language Question Answering. In Zampolli, A. (Ed.), Linguistic Structures Processing, Amsterdam: North Holland, 1977.
- Yazdani, M. and Narayanan, Artificial Intelligence: human effects. Ellis Horwood Ltd., 1984.

APPENDIX

A brief description of CBESS

Introduction.

CBESS was developed by Richard Brandt at the University of Utah and evolved from an earlier CAI shell VCIS, basis of the commercial Mentor system. It was written in C under Berkley UNIX 4.3 and 4.4. Brunel University is fortunate in being the first European beta test site for CBESS. It is a very large system including text and graphic editors and facilities to drive video peripherals. We are currently engaged in commissioning the system and in the meantime are developing courseware on Mentor which has reasonably compatible low-level datastructures.

Artificial Intelligence Aspects of CBESS.

The CBESS system comes with a number of AI 'games' which provide for a variety of different student activities over the same database. The database structures fall into one of three categories:

Simple pairing structure - LSCAI.

FSM simulation - EPST.

Production rules - CBMS.

All of these systems operate on a database of information, some nodes of which may be linked to video or graphical sequences. The questions and answers are not given explicitly by the author, but CBESS itself builds the questions from the rules of the database thus allowing the same material to be tested on many occasions without repetition of the questions or sequence of questions.

Simple pairing structure - Language Skills CAI.

LSCAI is a learning game designed to teach the meaning of unfamiliar words on a particular subject. It is a game that can be played with different 'lessons' to teach new words. Lessons are groups of words relating to a particular topic or subject, which are defined by Subject Matter Experts.

The activities which form the LSCAI system are:

1. Context Sequence.

In this activity sentences and graphics for the words in the lesson are displayed on the screen. Each word is given in context in a sentence to help with understanding of the meaning of the word. Graphics may also be used to show the meaning of the word.

2. Definition Building.

The Definition Building activity builds the definition of a lesson word a few words at a time. The first two or three words of the definition are displayed at the top of the screen followed by a blank line. A menu displays a list of words to fit into the blank. The definition is built by filling in the blank with one of the menu options.

3. Hidden Multiple Choice.

In the Hidden Multiple Choice activity the student is shown one complete definition at a time and is asked to say whether it is correct or not. Only one definition is shown at a time so that it cannot be compared with others. The activity shows the student a word in quotes and asks if the displayed definition is the correct one for the word.

4. Visible Multiple Choice.

In this activity the student is shown several definitions and asked to choose the correct one for the word. Four or five definitions are displayed in a menu for the student to choose from.

5. Spelling.

The spelling activity shows the student a word and allows time to study it. Once the student believes that he can spell the word it is removed from the screen and he is asked to spell the word.

6. Matching.

The object of the Matching activity is to match the lesson words with their correct definitions. All the lesson words are listed in a menu and all of the definitions are also listed. The student is then asked to match as many word/definition pairs as he can.

FSM simulation - Equipment Problem Solving Techniques.

Theoretically EPST can be used to simulate any system that can be decomposed into parts whose behaviour is describable in terms of states, inputs, and outputs - i.e. any Finite State Machine. These parts can be combined by connecting outputs to inputs. Examples of devices that can be represented in EPST include digital integrated circuits, switches, operational amplifiers, hydraulic valves, and power supplies. Examples of devices that cannot easily be represented as parts include resistors, capacitors, transistors and inductors.

The EPST author designs his piece of equipment by using a series of statements. These statements can be descriptions of states of some device, identification of inputs and outputs to the device, and a series of assertions which describe the relationship between the various components which comprise the equipment.

Having described the equipment, the author can then describe a number of problems which might arise (e.g. a component failure) and the symptoms which would be exhibited at various test points in the equipment.

The student is then presented with the problem, and making a series of tests and measurements is asked to find the fault.

Production Rules - Computer Based Memorisation System.

CBMS and its associated games are designed to help students memorize large bodies of structured, factual material such as taxonomies or technical data. CBMS games can be used with any body of material that has been cast into the form of a database compatible with the game system. The most common form of data which can be 'compiled' into a suitable database is a text file containing a set of 'isa' production rules. Here a few lines from such a database which describes the attributes of various types of aircraft:

```
aircraft isa db_element
```

"jet aircraft" isa aircraft "turboprop aircraft" isa aircraft

"Boeing 737" isa "jet aircraft" DC-10 isa "jet aircraft"

"Beechcraft C-99" isa "turboprop aircraft" "Mohawk 298" isa "turboprop aircraft"

isa STATEMENT "%s% falls under the classification %o%." isa SUBJECT "What are the subtopics of %o%?" isa OBJECT "What categories does %s% lie in?" isa YES_NO "Is %s% a subtopic of %o%?"

propulsion_relation isa db_relation

engines isa propulsion_relation horsepower isa propulsion_relation propulsion_relation SET HIDDEN_RELATION

propulsion_relation STATEMENT "%s% has %o% %r%." propulsion_relation SUBJECT "What has %o% %r%?" propulsion_relation OBJECT "%s% has _____ %r%?" propulsion_relation YES_NO "Does %s% have %o% %r%?"

unit_relation isa db_relation

"passenger capacity" isa unit_relation "wing span" isa unit_relation "pay load" isa unit_relation "cruising speed" isa unit_relation "gross weight" isa unit_relation length isa unit_relation range isa unit_relation unit_relation SET HIDDEN_RELATION

unit_relation STATEMENT "%s% has a %r% of %o%." unit_relation SUBJECT "What has a %r% of %o%?" unit_relation OBJECT "%s% has a %r% of _____?" unit_relation YES_NO "Does %s% have a %r% of %o%?"

units isa db_other

jet isa units ft isa units lbs isa units people isa units mph isa units miles isa units hp isa units

Most of the games involve a single player in situations that exercise memory for facts in a variety of ways. For example, a computerised flash-card game requires recognition or recall of the material in the database, and a twenty-questions game requires discrimination amongst items in the database.

The games which form the CBMS system are:

1. Concentration.

The student tries to identify a hidden item, which is some item in the chosen category. A list of items to match is shown. When the student matches one item in the list with all others that are related to it, a clue to the hidden item is given.

2. Constraint.

The objective of the Constraint game is to choose from a list of items, all those items that correctly answer the presented questions. The effect is to learn to discriminate amongst items in the database.

3. Flash Card.

The Flash Card objective is to answer the questions that are displayed. If the student cannot answer the question, he can ask for hints.

4. Identification.

The Identification objective is to identify an item in the category selected from facts about the item and then to answer questions about the item.

5. Jeopardy.

The objective of Jeopardy is to identify the question associated with each of the answers on the game board. Each answer is a description of one item in the category that the student selects. Jeopardy asks the student to select up to five categories.

6. Matrix.

The Matrix game board is a matrix in which each row represents an item, and each column represents a property that the items have. The objective of the game is to fill in the missing values represented by each block.

7. Twenty Questions.

The aim of the Twenty Questions game is to use yes/no questions to eliminate all but the hidden item from a list of items.

8. Picture Quiz.

The objective of the Picture Quiz is to identify each of the pictures displayed and all known parts of each picture. The game is available only for a database that has video or graphics pictures associated with some items.