Building an Intelligent Tutoring System
for the T-37B-1 Flight Manual

by

Sheryl A. Risacher

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

June, 1996

© MCMXCVI Sheryl A. Risacher. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

Author

Department of Electrical Engineering and Computer Science
May 24, 1996

Certified by

Professor John V. Guttag
Thesis Supervisor

Accepted by

F. R. Morgenthaler
Chairman, Department Committee on Graduate Thesis

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUN 11 1996
LIBRARIES
Building an Intelligent Tutoring System for the T-37B-1 Flight Manual
by
Sheryl A. Risacher

Submitted to the
Department of Electrical Engineering and Computer Science

May 24, 1996

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

It is widely accepted that intelligent tutoring systems include three basic modules: domain knowledge, student model, and tutor. There is, however, no agreement as to how these modules should be implemented or interact with one another. This thesis presents a systematic approach to building intelligent tutoring systems. We construct the domain module as a rule-based expert system, deriving the knowledge directly from texts or manuals. We build the student model as an overlay of the domain knowledge base. Finally, we implement the pedagogical module, which guides interaction with the student, through differential modeling. The utility of the approach is demonstrated through the implementation of STAND-UP, an intelligent tutoring system covering the emergency procedures of the T-37.

Thesis Supervisor: John V. Guttag
Title: Professor of Computer Science and Engineering
ACKNOWLEDGMENTS

I would first like to thank Prof. Guttag, my thesis advisor, who helped me envision an exciting, interesting thesis combining my career as an MIT student and my future career as a U.S. Air Force pilot. His encouragement and support made this final experience of the MIT adventure rewarding, rather then simply a necessary evil.

Next, of course, I wish to acknowledge the support and understanding of my wonderful husband, Daniel, who was working alongside me all of those very long nights. I especially appreciate his willingness to act as sounding board, sympathizer and slave driver considering that he faced his own “cliff.”

I will be forever indebted to the network of family and friends who saw me through one of my life’s most challenging periods, with support too varied to enumerate.

This research was supported by donations from Intel, Microsoft, and assistance from the United States Air Force.
1. INTRODUCTION

Researchers have recently produced much work relating to the development of intelligent tutoring systems (ITS). The advent of multimedia interfaces that are easy to use and a perceived crisis in American education, have heightened interest in computer assisted education. This interest, however, has not been of a uniform nature. The proceedings of the annual Intelligent Tutoring Systems Conference illustrate the large number of different, and sometimes opposing, views concerning the best implementation methods, even goals, of computer assisted education. It is true that computers can be a platform for innovative education reform, presenting students with a rich environment for self motivated discovery unavailable in traditional classroom education [McArthur, D., Lewis, M.W., and Bishay, M., 1993]. However, this does not at all lessen the relevance of computer assisted education that supplements human educators in a traditional one-on-one interaction. This thesis presents a paradigm for the development of such systems.

1.1 GOALS FOR AN INTELLIGENT TUTORING SYSTEM

Conventional human tutoring is a tried and tested educational method that can result in substantial improvements in student performance [McArthur, D., Lewis, M.W., and Bishay, M., 1993]. Producing a system that can emulate that interaction is a worthy (but not easily attainable) goal. One of the most important aspects of an effective tutor is that it possesses the knowledge necessary to solve the problems presented to the student.
Using that knowledge, it can identify student errors, and explain how to arrive at a correct solution. Also, the tutor should update and modify this domain knowledge, in order to present only current information to the student. Another crucial point is that the tutor automatically adapt to the student’s needs. The pedagogical controller accomplishes this by:

- generating problems and exercises at the correct level of difficulty,
- responding to the student’s prior performance, and
- presenting explanations in a way that will be intelligible to the particular student

1.2 THE TRADITIONAL MODEL

Almost as long as there have been computers, people have been interested in using them for education. The hope has been that computers could offer individual instruction, freeing up the time of the human educator and spreading educational resources farther. Computer assisted education has long been an interdisciplinary field, involving educators, computer scientists, psychologists, and artificial intelligence researchers. Research in this field has bumped up against many of the difficult problems of artificial intelligence, including natural language processing, heuristic search, and knowledge representation. Much of the early work in the field, then called computer assisted instruction (CAI), tried to skirt these difficult issues.

Traditional computer assisted instruction programs are often described as simple branching or frame-based programs. The path of instruction was completely laid out in advance by the programmer. At runtime the computer made only the specified pre-programmed responses to previously anticipated student inputs. [Wenger, 1987] compares
computer assisted instruction programs to sophisticated books, which can be intricate, well-planned and entertaining. However, these systems only handle situations to which they have a programmed response. This makes it impossible to generate new problems for the student and analyze the student's response. Most computer assisted instruction programs did not make any attempt to separate domain knowledge and pedagogical knowledge.

A later shift in nomenclature to the title of intelligent tutoring systems signaled an attempt to focus on incorporating the ideas and methods of Artificial Intelligence. The range of programs that qualify as intelligent tutoring systems is vast. The general idea is that rather than hard-coding responses to a student's possible actions, the program is instead given enough knowledge, both about the domain and about the act of teaching itself, to act autonomously.

A general model of many intelligent tutoring systems includes three basic modules: domain knowledge, student model, and tutor (figure 1). A rough test for intelligent tutoring systems is that using their domain knowledge, they are able to solve the problems the pedagogical module puts to the students. The tutor module controls the interaction with the student, based on its own knowledge of teaching and comparisons between the student model and the domain knowledge. Intelligent tutoring systems can range from very primitive to extremely sophisticated in these modules and their interactions. For instance, the interface to the student can range from a command line interface to a graphical user interface to an interface capable of natural language processing. The tutoring aspect can be implemented in various ways ranging from a Socratic dialog to a
more passive coaching style to a discovery companion, where the computer learns the material alongside the student. Many intelligent tutoring systems focus mostly on one aspect of the larger teaching problem. Many systems focus on improving one of the three main modules, or on meeting one of the goals outlined in the introduction, for example, 

![Figure 1 - Traditional ITS Model](image)

Figure 1 - Traditional ITS Model

perfecting student models so that tutoring modules can better tailor their performance to the individual.
1.3 PROPOSED APPROACH

I propose a standardized method for developing an effective ITS (figure 2). The heart of this method is extracting the domain knowledge from texts or manuals, encoding it into a knowledge-based system to serve as the ITS' resident expert, using differential modeling to determine the correctness of the student's response, and overlaying a student model on the knowledge base to guide the tutor's pedagogical decisions.

<table>
<thead>
<tr>
<th>Module</th>
<th>My Proposed Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Knowledge</td>
<td>Taken directly from texts, Coded into rule based system</td>
</tr>
<tr>
<td>Student Model</td>
<td>Overlaid on domain knowledge base</td>
</tr>
<tr>
<td>Tutoring Module</td>
<td>Guide interaction based on differential modeling</td>
</tr>
</tbody>
</table>

Figure 2 - My Proposed Approach

1.3.1 Domain Knowledge

The first step is to build a rule based expert system from domain knowledge already codified in texts and manuals. Using texts or manuals eliminates the need for the lengthy, difficult and expensive process of extracting knowledge from experts in the field. Educators in the field already accept the knowledge contained therein, so there is no need to wonder if the ITS has its facts straight.

Using a knowledge based system to represent the domain within the ITS is beneficial in several ways. The first is that the system itself has some intelligence. Using such a system in the context of an intelligent tutoring system means that it can generate problems to present to the student, and solve them itself to gauge the correctness of the
student’s response. Knowing which steps are involved in a proper solution, it can respond to a student’s request for assistance or intervene if the student heads off in the wrong direction. Most rule-based expert systems exhibit a transparent nature—the individual rules and path to a solution (as embodied by a rule trace) are easily understandable by a human user. This makes it possible for the system to offer explanations behind correct solutions. This is greatly preferable to simply giving the student the correct answer with no background, or, even worse, just telling the student that he is incorrect with no further guidance. Since the domain knowledge is internally broken into small, stand-alone chunks, different bits of knowledge can be combined in different ways to present the material at different levels of difficulty. This modularity of the knowledge also allows the knowledge base to be easily updated with current information. Also, the pedagogical knowledge responsible for guiding the interaction with the student is separate from the domain knowledge. This makes it easier to modify teaching strategies when domain knowledge is already complete, or to add knowledge to the system without worrying about affecting its overall behavior.

1.3.2 Student Model

The knowledge base also provides a convenient way to model the student’s understanding of the material. Overlaying the student model on the domain knowledge does exactly that. Since knowledge is in the form of rules, rules can be marked as understood by the student if the student’s performance demonstrates an understanding of that particular conclusion following from the given premise. The student model is then comprised of these marked rules. Marking can be extended to any level of detail. For a
simple system, the markings ‘known,’ ‘unknown’, and ‘untested’ might be sufficient. A more complex student model could easily incorporate more information into these tags, such as the situations in which a student has trouble applying a rule he usually seems to understand.

One potential difficulty with this method is the issue of marking rules that contain variables. Rules that contain variables can be fired multiple times with different instantiations of the variable. Thus, the student could understand and utilize a particular instantiation of the rule, but have no facility with others. Implementers of overlaid student models who use a knowledge-based system tool that allows rules with variables need to be aware of this situation.

A model of the student’s knowledge is essential for tailoring the behavior of the system to match the student’s needs, yet some intelligent tutoring systems avoid compiling a student model. The lack makes the system less able to individualize its interaction with the student. Having a student model that is an overlay of the domain knowledge greatly facilitates the implementation of a student model because most of the structure is in place after assembling the domain module. Having these two modules share a representation also simplifies the tutoring module’s job of comparing the student model to the domain module.

1.3.3 Pedagogical Controller

Given a rule-based representation of the domain knowledge, and a student model overlay of the same knowledge base, the pedagogical module can use differential modeling to guide the intelligent tutoring system’s interaction with the student. Differential
modeling is simply a comparison of the student model and the domain module that allows the tutoring module to note where the two disagree. Usually differential modeling would be utilized twice during a problem solving session.

First the pedagogical module will use differential modeling to compare the responses of the student and the domain module to the problem presented. Where the student and the domain expert agree, the tutoring module will assign the student model credit for the knowledge used by the domain expert to produce the response. In the case where the student and domain module’s responses diverge, differential modeling assists the tutoring module in determining what part of the knowledge base the student misunderstands. Differential modeling can even be applied in an attempt to compare the student and domain module’s problem solving methods, for example as evidenced by the questions they each ask. Again, this is useful in pinpointing the student’s errors and correctly tagging the unknown knowledge.

Once the problem is completed, the tutoring module again applies differential modeling, this time comparing the student model to the domain knowledge base. This allows the tutoring module to select the next problem appropriately. This selection should either reexamine areas where the student has demonstrated misunderstanding or test the limits of the student’s knowledge by venturing into new material, avoiding material the student sufficiently comprehends.

1.4 CONSTRAINTS

An intelligent tutoring system built following the proposed approach is most applicable to teaching facts or procedural knowledge, i.e., the types of things that can be
well represented in a knowledge base. Research in Artificial Intelligence has shown that rule based systems capture well the knowledge of abstractions and associations involved in diagnosis type problems. Our approach, which is a refinement of the traditional intelligent tutoring system model, is much less well suited to other educational goals, for instance analytical reasoning or creative writing. I will not address the issues raised in that kind of learning environment. Situations where the goal is to emulate the interactive behavior of human tutors, communicating what to know as opposed to how to think, are good candidates for an intelligent tutoring system following this paradigm.

1.5 TESTING THE APPROACH—IMPLEMENTING STAND-UP

To test the proposed approach, I built STAND-UP, an intelligent tutoring system that helps students learn the T-37B emergency procedures. The T-37 is the Air Force’s primary trainer, used for pilot training. STAND-UP mimics the interaction between a student and safety officer in the stand-up briefings used to test emergency procedure knowledge. A student pilot’s primary reference for the material is the T-37B-1 Flight Manual, which discusses the symptoms of various emergencies and the actions to be taken should they arise during a training flight. Much of this information must be committed to memory. The emphasis in the training program is on knowing these procedures cold, as opposed to being able to figure out why certain actions are important in the sequence. The nature of the problem makes STAND-UP a perfect candidate for testing my hypothesis. The proposed approach suggests building an effective ITS by encoding the domain knowledge encapsulated in the T-37B-1 Flight Manual into a rule-based expert system, tagging the rules according to the student’s demonstrated understanding of their
content, and guiding the interaction with the student based on this model and the differences between the student’s response and that of the expert system.

In the following chapters, I will first present a short history of the field of computer assisted education. The succeeding chapters will then discuss the implementation of STAND-UP, concluding with a critique of the proposed approach and observations that result from the endeavor.

2. RELATED WORK

2.1 CASE STUDIES

The evolution and progress of intelligent tutoring systems can be traced through implemented systems. I will briefly mention and discuss some of the more important steps in the history of intelligent tutoring system.

2.1.1 SCHOLAR

Probably the first system to make the transition from computer assisted instruction to intelligent tutoring system was SCHOLAR [Carbonell, 1970]. SCHOLAR used a knowledge base about South American geography to teach students through a question/answer session. Topics were picked at random by SCHOLAR, but the student was able to change the direction of the discussion. SCHOLAR did not have any sort of student model and its teaching method was fairly crude. It used templates and keyword recognition to conduct a realistic conversation without natural language processing. It was
later able to make inferences from its knowledge to answer questions using "plausible reasoning." [Collins, 1977]

2.1.2 SOPHIE

SOPHIE [BROWN et al., 1974] was the first system to provide a simulation environment in an intelligent tutoring system. SOPHIE's domain was the debugging of electronic circuits. Rather than presenting material or quizzing the student, SOPHIE focused on an environment of discovery where the student would make various observations about a faulty circuit and eventually come up with a hypothesis. SOPHIE would then use its expert knowledge to decide if the hypothesis was correct, if the measurements the student asked for were reasonable, what the possible diagnoses were, etc. SOPHIE had no teaching knowledge, per se, and no model of the student. SOPHIE was an effective teaching tool and modified and used for a number of years.

2.1.3 WEST

WEST [Burton and Brown, 1976] filled the role of a coach, deciding when pedagogical interruption would be profitable. WEST coached students playing the game "How the West Was Won" which involved basic arithmetic skills. The coach would carefully monitor the player's moves, waiting for an opportune moment to provide a relevant, memorable example. In this case, the domain knowledge was fairly simple and straight-forward and the expert system responsible only for listing all possible moves in order of best score. The coach had the fairly complex job of determining if the student's move was optimal, what factors may have influenced the player's move selection and
whether to present the technique(s) the player had missed an opportunity to use. This concept of comparing a student’s performance with that of an expert system solving the same problem came to be known as differential modeling.

2.1.4 WUSOR

Like WEST, WUSOR [Stansfield, Carr, and Goldstein, 1976] was developed as a coaching program for a game. WUSOR was developed for WUMPUS, an exploratory adventure game. In early versions, WUSOR had no model of the student and little teaching knowledge - it would simply point out to the student any move that differed from the optimal move generated by its expert system. A major improvement came when Carr and Goldstein [1977] devised the overlay paradigm. This was a way of implementing the student model on top of the knowledge base of the expert system, by tagging rules in the knowledge base as known or unknown by the student. The advisor component of the program would observe the player closely and infer from his actions what rules of the system he were cognizant of, then use that knowledge to make coaching decisions.

2.1.5 BUGGY

Another program developed about the same time, BUGGY [Brown and Burton, 1978], focused on students’ misconceptions in a different way. Rather than endowing BUGGY with the tutorial skill needed to uncover and counter-act student misconceptions, the developers of BUGGY concentrated on developing a complete diagnostic model of the student. They attempted to enumerate the different possible procedural bugs math students might acquire while trying to solve decimal addition and
subtraction problems. In a reversal of the usual roles of ITS, teacher, and student, BUGGY used its student model to simulate a student with "buggy" thinking. The educator then diagnosed the student bug based on BUGGY's examples. Interestingly, using this catalog of possible bugs, BUGGY could generate general diagnostic tests to identify students' misconceptions.

2.1.6 GUIDON

GUIDON [Clancey, 1979] was the first intelligent tutoring system built on top of an pre-existing expert system, MYCIN, the medical knowledge-based system used for diagnosis. GUIDON would choose a case from MYCIN and then, as the student attempted a diagnosis, assist and critique the student according to its tutoring goals and the knowledge base of MYCIN. GUIDON combined many different aspects of earlier systems and set the tone for later intelligent tutoring systems by delineating the different modules of an intelligent tutoring system and their different purposes.

2.1.7 STEAMER

STEAMER [Williams, Hollan, Stevens, 1981], a simulation-based trainer for large ships' steam plants, was developed around that same period of time. STEAMER provided an easy to use graphic simulation environment where students could learn how various factors and inputs would affect the system as a whole. An expert system that understood the workings of the simulation could give feedback to the student, including explanations of the reasoning behind standard procedures, at various levels of abstraction. This was
another example from the exploratory, student-directed end of the spectrum, with some coaching as well.

2.1.8 WHY

The follow-on to SCHOLAR, that first intelligent tutoring system, was WHY [Collins and Stevens, 1982]. WHY expanded SCHOLAR’s original domain into causal meteorology and boosted the tutoring knowledge to the point where WHY’s interaction with the student was in the form of a Socratic dialog. In WHY the tutoring knowledge was distinct from the domain knowledge and in abstract form. WHY attempted to diagnose students’ misconceptions, based on their inferences and statements and to correct them with counter-examples, in the Socratic manner.

2.2 RECENT DEVELOPMENTS

Until the mid 1980’s the field of computer assisted education was fairly unified, with each new intelligent tutoring system adding to the amassed body of knowledge relevant to educational applications of Artificial Intelligence methods. The interesting issues were clearly defined, and researchers were agreed on the directions progress should take. The state of the field at that time is well described in [Wenger, 1987] and [Sleeman and Brown, 1982].

With the arrival of the 1990’s this unity was completely lost. Researchers and developers shot off in all directions, responding to three main concerns. [Wenger, 1987] expressed a general concern that intelligent tutoring systems were developed and tested in
the lab, but rarely ever made it to the classroom. Many researchers devoted themselves to this issue, producing small systems which incorporated few, if any, new ideas in an effort to produce programs that would actually be used. A few of these systems enjoyed some success, for example CHEMPROF [Eggert, Middlecamp, and Jacob, 1992] and Anderson’s geometry and LISP tutors [Anderson, Boyle, and Yost, 1985; Anderson and Skwarecki, 1986]. Most, however, never achieved significant success, and this thrust of research contributed little to the field as a whole.

Currently, other researchers are focusing on the diverse issues raised by the implementation of the first generation of intelligent tutoring systems [McArthur, D., Lewis, M.W., and Bishay, M., 1993]. Some current work is expanding the range of domains for which an intelligent tutoring systems is appropriate. Other researchers are working on developing comprehensive student models, and providing appropriate explanations at the right level for the student. Developing shells that educators can use to produce their own intelligent tutoring systems is a hot area. Others feel that the old goals of intelligent tutoring systems should be completely revised, and that the best direction for computer assisted education is in simulations or micro worlds where students can learn by unconstrained discovery.

The explosion of graphical user interfaces and multimedia applications has had a drastic impact on the field of computer aided education. Much effort has shifted from making educational applications more intelligent to making them easier and more fun to use. It is easier and faster to produce a multimedia educational tool that does nothing...
more intelligent than present information than to devise an intelligent tutoring system to
guide the student’s learning in the same domain.

The following systems are examples of the types of intelligent tutoring systems
currently being produced:

2.2.1 AEGIS CIC ITS

SHAI inc. has developed several commercial products, including the AEGIS CIC
Intelligent Training System used by the US Navy [Stottler Henke Associates Inc, 1995].
The AEGIS ITS is built on the notion that the best way to learn is by examples, and the
best examples are animated simulations. Adhering to the theory of case-based reasoning,
each example describes a problem, the steps necessary to arrive at a solution, and the
solution. The other major tenant of ITS development at SHAI is the importance of mental
models. In the AEGIS ITS, the decision making of the student is complicated by a
number of issues, each of which is represented by a mental model in the system. The
system is capable of judging the student’s level of ability on a few test cases, presenting
relevant cases as examples and then testing the student’s learning with other appropriate
cases.

2.2.2 CALAT

NTT Information and Communication Laboratories is using an in-house ITS
development shell, CAIRNEY [Fukuhara, Kiyama, and Nakata, 1991], to develop a
distributed hypermedia intelligent tutoring system on the World Wide Web, CALAT
[Nakabayashi, et al., 1994]. Using the World Wide Web allows student access to all types
of information as part of their learning session, including such resources as on-line libraries and museums. Much of the systems can be put together from extant pieces. The user can use any web browser to interface to the program. Courseware and technology already publicly available on the Internet are not yet integrated into the system, but will later be available with little hassle. So far the project has been working on dealing with the obstacles posed by using the Web, such as authenticating users to personalize the interaction, and controlling what information the user can access. (Viewer control will eliminate the possibility that a student opens the answer page while still on the question page.) The next phase of the project will provide navigational assistance and guidance to the user, leading him to sites that will match his interests, education goals, and competence in the material.

2.2.3 EXPLAIN

EXPLAIN is under development at the University of Nottingham [Reichgelt, et al., 1994]. EXPLAIN’s developers are exploring the issue of how people learn (and from there how to best teach), and therefore are concentrating on the teaching module of the intelligent tutoring system. Their goal is to use EXPLAIN to provide control and close monitoring of an interactive learning session involving a human tutor. The developers of EXPLAIN will then use the empirical evidence gathered in these sessions to develop models of how people learn. EXPLAIN should eventually be able to adjust the level of help given to students based on their previous performance. Currently, EXPLAIN allows the educator to guide the session with EXPLAIN and to affect the strategy taken to achieve set goals.
2.2.4 EPITOME

The School of Civil and Environmental Engineering at the Georgia Institute of Technology is sponsoring EPITOME, Engineering Platform for Intelligent Tutors and Multimedia Experiences [Baker, 1994]. They make the distinction between the intelligent tutoring system shell they are developing for engineering applications and the more general purpose multimedia computer assisted instruction tools they are also developing. Their intelligent tutoring system shell is to be used for producing intelligent tutoring systems in different engineering domains. They are relying on the standard modularity of intelligent tutoring systems to ease the transitions between domains. This production method should allow them to enrich the tutoring module beyond the norm for current systems, allowing a student a choice of teaching/learning styles. Incorporation of real-world examples is also an important goal of their intelligent tutoring systems development.

2.2.5 Microworlds

A team of researchers at the RAND corporation are exploring the prospects of current intelligent tutoring systems, and are trying to formulate new educational goals and methods that can be integrated into new and unconventional intelligent tutoring systems. Some of these are different enough from the main body of intelligent tutoring systems that they have decided to call them Interactive Learning Environments, or Microworlds education [McArthur, D., Lewis, M.W., and Bishay, M., 1993]. So far their prototypes of Microworlds have used math as the domain of instruction. Microworlds stress student control of their exploration and discovery with little or no assistance or guidance from the
system. This environment is one of “construction, not instruction.” This system attains an intelligent tutoring systems’ main goal of individualized, pertinent feedback, in the sense that the inputs of the student lead to changes in the state of the system. The state of the system is implicit feedback. In this manner, the Microworlds will teach not just the domain knowledge, but also what the developers term inquiry skills.

3. DOMAIN MODULE IMPLEMENTATION

3.1 DEVELOPMENT ENVIRONMENT

The goal of this project was to implement an intelligent tutoring system in order to evaluate the proposed approach. Therefore, unlike much current educational software, an easy-to-use, multimedia graphical user interface was not the focus of the project. However, in today’s computing environment, most programs that rely on a text-based interface hold little appeal for new users. Consideration of these factors led to the decision to use Microsoft Visual Basic as the development environment for the system. Using Visual Basic allowed rapid prototyping of the system, without detracting from the interesting issues of the project.

For the implementation of the knowledge base, I decided to use a commercially available product. This provides both an inference engine and a knowledge representation system. After attempting to use a separate program for this, I settled on implementing the domain module in M4.VB. M4.VB works as a Visual Basic custom control, greatly reducing the potential communication difficulties between two stand-alone programs.
Once the decision was made to use M4.VB, implementing the domain module was reduced to building the knowledge base of emergency diagnosis and procedures from the T-37B-1 Flight Manual, and setting up the interaction between the tutoring module and M4. Unlike most knowledge-based systems, the domain module does not interact with a human user. Instead it asks its question of, and receives answers from the tutoring module. Achieving this was a bit of a hack since M4.VB was designed to interact with a human as the end user.

In STAND-UP the interaction sequence goes as follows: first the tutoring module passes key features of the scenario to the domain module. Then, M4 will take the given information and proceed to a solution, asking any necessary questions of the tutoring module and finally presenting its conclusions, including a diagnosis of the situation as well as a set of recommended actions. The domain module’s interaction sets up the ideal to which the student’s performance will be compared.

3.2 CAPABILITIES

The task of the domain module is to diagnose emergency situations that could arise in the T-37B Tweet and recommend appropriate actions for dealing with the situation. The emergency situation is determined based on such things as the airplane’s airspeed, altitude, RPM, etc. For example, for the situation in figure 3 (where ‘stopping distance’ refers to length of runway remaining for the plane to stop):
The system's output would be:

Abort
Throttles - Cutoff
Wheel Brakes - As Required

The system uses a backward-chaining inference engine to work backwards from the three stated goals, diagnosis, actions and considerations. In order to make conclusions about these three attributes and satisfy the goals, the system must find the values of any attributes that appear in the premises of rules that include the goal attributes in their conclusion (see figure 5).

For instance, the two necessary aspects of a diagnosis usually are the emergency and general-where. Most of the interaction focuses on getting the value of 'emergency'. There are several different types of emergencies, and each kind of emergency must be looked into, lest a less important emergency be diagnosed than actually exists. General-Where can be one of on-ground, taking-off, landing, low-alt or at-alt. This is determined
directly from the airspeed, pitch and altitude. Sometimes additional information is needed to pin down the diagnosis. Once a diagnosis is arrived at, the actions are quickly generated, possibly after asking the user a few more questions relevant to the particular diagnosis. Then, any extra considerations not explicitly dealt with are supplied to the user.

In the case of STAND-UP, the user is the tutoring module of the intelligent tutoring system, which is at the same time answering the questions of a human student who is presented with the same problem as the domain module. Therefore, this system's results are not an end in and of themselves, but rather are used to determine the correctness of the student's responses.

### 3.2.1 Sample Interaction

In order to more clearly illustrate exactly what STAND-UP's domain module does, an annotated version of this interaction, based on the input in figure 3 follows:

```
Domain> Is the right engine wind-milling?
   yes
   no

Student> "no"
```

In this, and the next series of questions, the program is looking for the value of the attribute emergency. Having been appraised that the left engine is on fire, and therefore has failed, the program wonders if maybe both engines have failed. This is an important distinction, because single engine operation is very different from a
complete loss of power. Engine failure can manifest itself by the engine wind-milling, freezing or flaming-out in addition to fire.

Domain> Is the right engine frozen?
   yes
   no
Student> "no"

Domain> Did the right engine flameout?
   yes
   no
Student> "no"

Domain> Please describe the fuel flow.
   fluctuating
   normal
   100
Student> "normal"

Now the system has given up on two-engine failure and is seeing if it can identify any other relevant emergencies. Here it is working on Erratic Fuel Flow. Since the fuel flow is described as normal, that line of questioning is cut off.

Domain> Is there any smoke in the cockpit?
   yes
   no
Student> "no"

Smoke inside the cockpit is a good indication of a couple different emergencies, including (unsurprisingly) 'Smoke and Fumes' and Overheat. Again, no evidence is provided that would encourage this line of questioning.
Domain> Is the engine running rough?
  yes
  no
Student >"no"

Smoke visible outside and a rough running engine can both be symptoms of an
overheat. All this questioning about other possible emergencies seems somewhat out
of place, since we already know that there’s a pretty severe emergency demanding our
attention, namely the engine fire. However, you want to make sure you don’t under-
diagnosis the emergency, say, dealing with a blown fuse while your flight suit becomes
a flaming inferno.

Domain> diagnosis = abort (100%) because rule-34.

STAND-UP’s domain module has arrived at a diagnosis at this point – abort with
100% certainty. (The term ‘abort’ may be slightly misleading since ‘abort’ is itself an
action. In STAND-UP the possible diagnosis are named according the boldface
procedures in the Flight Manual.) The emergency was engine fire, but the plane was
engaged in taking-off, making the emergency situation abort. (If any emergency
occurs on take-off, the situation will be abort.) Rule 34, which the system mentions,
states the following:

```
Rule-34:
if general-action = abort
then diagnosis = abort.
```

Figure 6 - Rule 34

Note that rule 34 in turn relies on rule 42 to determine the value of the attribute
‘general-action,’ and so on:
Rule-42:
\[
\text{if general-where = taking-off and emergency-status = have-one then general-action = abort.}
\]

Domain> Please describe the stopping distance.

sufficient

minimal

Student> "minimal"

Now that the system has reached a diagnosis, it looks for specifics about the situation that would affect the list of recommended actions (given the diagnosis). When aborting, it’s possible that you could find it impossible to brake hard enough to stop before the end of the runway, since until the moment you abort you’re running two jet engines at full power. In this case, having only minimal stopping distance will affect our chosen actions.

Action = Throttles - cutoff (99%) because rule-49. Action = Wheel Brakes - As Required (99%) because rule-51. Considerations was sought, but no value was concluded.

The program has finally produced the desired list of actions relevant to the situation, which was diagnosed as abort. Because of the minimal stopping distance, both throttles are cutoff completely (in a regular abort, they would just go to idle, while in an abort resulting from an engine fire, the malfunctioning engine would get cutoff, the other going to idle), in order to stop in the shortest possible distance and kill the engine fire. The pilot brakes as necessary to avoid obstacles and stop the plane. There
are no other considerations in this case. The considerations would ordinarily be items like the complications of minimal stopping distance, but in this case, the consideration has already been taken care of.

3.2.2 Limitations

Because this system is a component of the larger STAND-UP program, it can only handle the same subset of emergency procedures the rest of the program can also handle. Specifically, STAND-UP concentrates on the procedures classified by the Air Force as "boldface" procedures - Actions that must be memorized exactly, word for word, by heart, by the pilots in training. These critical actions, as they are also called, include Abort, Two Engine Failure (Low Altitude), One Engine Failure, Fire, Overheat during Takeoff (After Airborne), Engine Fire/Overheat During Flight (Affected Engine), Emergency Airstart, Ejection, and Single Engine Go Around. It can also identify (for no particularly good reason, except that it seemed wrong to remove knowledge from the system) Erratic Fuel Flow. For each of these boldface procedures, the system can identify the situation, provide a diagnosis, and then list the appropriate actions to take (as listed in the T-37B-1 Flight Manual).

A problem could be out of the program's league, however, if it involved a more subtle diagnosis of the problem. For example, the only way that the system will reach a diagnosis of engine fire is if the engine fire light is on. Any other way that a pilot might diagnosis an engine fire, such as a wingman radioing that he sees a great ball of fire, would be missed by this system. It has no way of collecting unplanned for information and no way of using it if it did.
3.3 Knowledge Base

STAND-UP's knowledge is encoded directly from the T-37B-1 Flight Manual. The Flight Manual covers the systems of the aircraft and the procedures in great detail. The system contains knowledge about the aircraft systems (figure 8), and different flight situations (figure 9) that allows it to diagnose the emergency situations. For example,

Rule-13:

if right-engine = failed and
left-engine = failed
then emergency = two-engine-failure

Figure 8 - Sample Aircraft Systems Rule

Rule-15:

if gairspeed = slow and
gVVI = decreasing and
galtitude = low
then general-where = landing

Figure 9 - Sample Flight Situation Rule

It also has a set of rules that spell out the actions to take, given a certain diagnosis. For example,

Rule-52:

if diagnosis = two-engine-failure-low-alt
then action = 'Glide - 100 KIAS Minimum,' cf 99.

Rule-53:

if diagnosis = two-engine-failure-low-alt
then action = 'Gear - Down.' cf 99.
The Flight Manual also contains information organized into Notes, Warnings and Cautions. This information is also incorporated into the knowledge base, either directly affecting the systems choice of action, or presented to the user as an applicable consideration.

Rule-54:

if diagnosis = two-engine-failure-low-alt
then action = 'Throttles - Cut-Off.' cf 99.

Figure 10 - Sample Action Rules

Rule-55:

if diagnosis = two-engine-failure-low-alt
then considerations = 'Canopy should be retained, offers protection from explosion.' cf 99.

Figure 11 - Sample Consideration Rule

The diagram on the next page (figure 12) illustrates the system's knowledge base. The tree structure portrays the relationships of the attributes in the knowledge base. Lines connect attributes which are inferred from the attributes below. An arc connecting lines depicts attributes that work together to influence the higher (more abstract) attribute. The three goals of the system are at the top.
Figure 12 - Knowledge Base Inference Diagram
3.4 CONCLUSIONS

Were I to do this whole project over, I would write my own inference engine. Because my application is different then the traditional ‘human user consults computer expert’ model there were times when the usual functionality wasn’t appropriate. Given that I actually implemented the project in Visual Basic using M4 as a knowledge inference engine, some things went well and others didn’t.

M4 has its quirks. Multiple conclusions can not be expressed in a single rule. This was mostly inconvenient when the diagnosis of an emergency situation had several consequences. Along similar lines, certainty factors had to be fudged to achieve realistic performance from the system.

Other issues arose which were difficult, as opposed to just inconvenient. Reacting to an emergency often involves a cycle of diagnosis and actions, dealing with the current situation then gauging the effects of your actions. Thinking of the domain module as a traditional knowledge-based system, I was unable to make it see a changing state. In many expert systems, this is a real problem because you can only input a snapshot of the situation in progress. This has small scale effects, for instance, coding such actions as ‘If the landing gear fails to extend’ and ‘If the RPM drops below...’ The larger problem is dealing with the new scenario that results from the emergency situation the user successfully dealt with. For example, after aborting a take-off because of an engine fire, the user is on the runway in a machine that’s on fire.
However, there is a solution to this problem when one uses the proposed approach to build an intelligent tutoring system. Since the pedagogical module is controlling the interaction with the domain module, feeding it the scenario, it is not confined to presenting a single situation to the domain module. Instead, the tutor controller can present the scenario, and then, re-present the scenario, having effected the actions recommended by the system. This cycle can continue as long as is necessary to completely deal with the entire problem. Unfortunately, I did not have time to actually implement this solution.

A plane’s Flight Manual is its pilot’s bible, a seemingly endless font of systems and procedures knowledge, with notes and warnings covering almost every conceivable case. So I expected that if I encoded the knowledge in the manual, my system would know everything it needed to. I was wrong. For example, there is knowledge so basic that it is not explicitly stated. An example of this the rule-1 which states that if the altitude and airspeed are both zero, the aircraft is on the ground. And even worse, much of this implicit knowledge, that I had to supply from my own experience, was difficult to state in the form of rules. In addition, at times the Flight Manual realizes the situation is a judgment call for the pilot - ‘If ejecting isn’t an option, then...’ These situations are notoriously difficult for knowledge-based systems to handle. To make the proper decision you really want to be in the plane, with your hand on the stick. (Thus violating the “telephone test”.)

My particular use of the system also complicated the implementation. One use to which I wished to put the domain knowledge, was a comparison of the student’s questions to determine if they were timely and relevant. However any pilot who matched the
domain module’s questioning pattern is destined to end up in a small box. Once you see the fire light, you should be in action, not asking questions about the fuel flow. In addition, since I was building an intelligent tutoring system, the tutoring module should be able to determine the difficulty of the questions that the student is being asked to solve. This representation and method of solving by the domain module, however, made it impossible to distinguish between hard and easy questions. This particular issue is a good example of where a special purpose inference engine and knowledge representation might be considerably more effective.

4. STUDENT MODEL

A key aspect of the proposed approach is the basic method for implementing the student model. The idea is to maintain a model of the student’s knowledge, as perceived by the pedagogical module through differential modeling, as an overlay of the domain knowledge base. This simply involves tagging different rules to reflect the student’s apparent comprehension of them. The tutoring module then uses the student model to guide the interaction with the student.

The use of M4.VB as an inference engine meant that there was a hard abstraction barrier between the knowledge representation, the rules, and the other parts of the system. Therefore, the student model originally consisted just of a boolean array. Each element of the array represented a student’s knowledge of the rule sharing its index. This eventually proved to lack the desired functionality, since no manipulation of the rules was possible without using the M4.VB interface.
To eliminate this problem and lend significantly more functionality to both the student model and the pedagogical controller, I eventually decided to maintain a separate copy of the knowledge base for the student model.

4.1 Data Structure

As STAND-UP parses the knowledge base, a rule in the internal representation is built from each of the rules in the text file. A rule contains five fields, in addition to a premise string and a conclusion string. Other fields contain the attributes used in the premise, the attribute in the conclusion, whether or not M4 used the rule in reaching a diagnosis or recommending actions, and the perceived level of student understanding of the rule. Figure 13 illustrates Rule 25 as an example of the rule data structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premise</td>
<td>'if fuel-fumes = present'</td>
</tr>
<tr>
<td>Premise Attributes</td>
<td>fuel-fumes</td>
</tr>
<tr>
<td>Conclusion</td>
<td>'then fire-danger = present'</td>
</tr>
<tr>
<td>Conclusion Attributes</td>
<td>fire-danger</td>
</tr>
<tr>
<td>M4 Used for Diagnosis</td>
<td>{true, false}</td>
</tr>
<tr>
<td>M4 Used for Actions</td>
<td>{true, false}</td>
</tr>
<tr>
<td>Student Knows</td>
<td>{0, 1, 2} (see scale below)</td>
</tr>
</tbody>
</table>

Figure 13 - Example Rule

Two fields of a rule in the student model refer to the rule’s usage by M4.VB. Rather than just having one boolean, set to true if M4.VB used the rule at all, one field is true if M4.VB used the rule to reach a diagnosis for the current scenario, the other if it was used to conclude recommended actions. A rule qualifies as used if it successfully fired during the domain module’s solution. Rules that are never tested or that fail when tested, are not marked at all. The mechanism for deciding if the rule is an action rule or a diagnosis rule is straightforward. If the emergency situation has not yet been diagnosed by
M4.VB, succeeding rules are marked as used for diagnosis. Afterwards, succeeding rules are marked as used for actions. The rationale for this breakdown is to assist the tutoring module in identifying more precisely where the student's errors occur. This breakdown is the most convenient since the student and M4.VB's diagnoses and lists of actions are the most natural parts of the interaction to compare.

Under the proposed approach, there is a lot of flexibility for the implementer when deciding how exactly to represent the student's knowledge of a given rule. Options range from coarse granularity, the student has either demonstrated knowledge of the rule or not, to a level of detail only limited by the implementer's imagination. An example of the latter might be a system that separately tracked the student's demonstrated knowledge on each scenario, perhaps attributing successes and failures to different circumstances.

The implementation that made the most sense for the available resources of this project was a compromise between the extremes. A small scale rates the student's knowledge.

<table>
<thead>
<tr>
<th>0</th>
<th>The student's knowledge of this rule is unknown.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The student has demonstrated knowledge of the rule.</td>
</tr>
<tr>
<td>2</td>
<td>The student has demonstrated a lack of knowledge of the rule.</td>
</tr>
</tbody>
</table>

Figure 14 - Student Knowledge Scale

While it's true that the scale this implementation utilizes isn't as fine grained as one might like, it is significantly better than a single boolean flag. A student's demonstrated misunderstanding of some piece of domain knowledge is critical pedagogical information, without with the tutoring module must be less effective. In addition, using a small scale
puts the framework in place for later modifications. It would be a relatively simple matter to expand the scale. Using a five point scale instead of the current three would allow for such distinctions as the student usually understands the rule, or remembers it once in a while.

4.2 Example

To make the discussion of the student model concrete, the following example of building a student model is presented. The example traces M4's rule use, noting changes to the student model and notes how the student receives credit for these rules. (Rules that appear multiple times in the following example are uses of the rule with different instantiations of a variable.)

First M4.VB solves the problem. Figure 15 illustrates this process. Rules are invoked when their conclusion results in a value for an attribute whose value is being sought. When the values of the attributes in the premise of a rule are unknown, other rules are invoked that would help to infer those values. A rule either fails or succeeds, if its premise is false or true, respectively.

<table>
<thead>
<tr>
<th>Rule Number</th>
<th>Result</th>
<th>Student Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>Invoking</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Invoking</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Invoking/Failed</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Invoking</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Invoking/Succeeded</td>
<td>M4 Used for Diagnosis</td>
</tr>
<tr>
<td>2</td>
<td>Succeeded</td>
<td>M4 for Used Diagnosis</td>
</tr>
<tr>
<td>22</td>
<td>Failed</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Invoking/Succeeded</td>
<td>M4 Used for Diagnosis</td>
</tr>
<tr>
<td>33</td>
<td>Invoking</td>
<td></td>
</tr>
<tr>
<td>29-32</td>
<td>Invoking/Failed</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Failed</td>
<td></td>
</tr>
</tbody>
</table>
Then the student has a chance at the problem. When he has arrived at a diagnosis, it is compared with M4.VB's.
Since the student’s diagnosis is correct, the student model is updated to credit the student with knowing all of the rules that M4 used to make the diagnosis.

<table>
<thead>
<tr>
<th>Unknown</th>
<th>Student Knows</th>
<th>Student Doesn’t Know</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,3,4,6-22,24-28,30-33</td>
<td>2,5,23,29,34,42,45</td>
<td></td>
</tr>
<tr>
<td>35-41,43,44,46+</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 18 - Student Model (After Diagnosis)**

Once the student has finalized the list of recommended actions, his list of actions is compared to that produced by M4 (Note the difference in the first action).

<table>
<thead>
<tr>
<th>M4’s Actions</th>
<th>Student’s Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throttles - Cutoff</td>
<td>Throttle - Cutoff (Malfunctioning Engine)</td>
</tr>
<tr>
<td>Wheel Brakes - As Required</td>
<td>Wheel Brakes - As Required</td>
</tr>
</tbody>
</table>

**Figure 19 - Actions Comparison**

In this case, the student clearly should be given credit for knowing some of the relevant rules, but not for all of them, as it seems the student supplied the actions for an engine-fire abort without considering the minimal stopping distance. The pedagogical module, by a method discussed in the next chapter, determines which of the rules contribute to the incorrect action provided by the student. The student model is again updated to assign credit where due.
The relevant rule of which the student has demonstrated a lack of understanding, Rule 65, is shown in figure 21.

<table>
<thead>
<tr>
<th>Premise</th>
<th>&quot;if diagnosis = abort and stopping-distance = minimal&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premise Attributes</td>
<td>diagnosis, stopping-distance</td>
</tr>
<tr>
<td>Conclusion</td>
<td>&quot;then action = &quot;Throttles - Cutoff&quot; cf 99.&quot;</td>
</tr>
<tr>
<td>Conclusion Attributes</td>
<td>action</td>
</tr>
<tr>
<td>M4 Used for Diagnosis</td>
<td>false</td>
</tr>
<tr>
<td>M4 Used for Actions</td>
<td>true</td>
</tr>
<tr>
<td>Student Knows</td>
<td>0 (unknown)</td>
</tr>
</tbody>
</table>

The former problem is solved by STAND-UP in a similar manner to the basic student model. As the knowledge base (which contains the possible questions M4 might ask, although these are not in the form of rules) is parsed, questions are collected in an
internal representation much like rules, with an associated attribute. These questions, like the rules, can be marked as used by M4.VB and/or asked by the student.

The more abstract details about the student, however, resist the overlay paradigm for the simple reason that there is nothing to overlay abstract concepts on. This issue is an interesting one that could be explored in another implementation following the proposed approach.

5. PEDAGOGICAL CONTROLLER

Air Force training, as stated in the T-37B-1 Flight Manual asserts that there are three main steps to take in any emergency:

- Maintain aircraft control
- Analyze the situation and take the proper action
- Land as soon as conditions permit

STAND-UP guides the student through the first two of these. The last step, land as soon as conditions permit, is not always appropriate (for example, after the student has correctly performed an abort, there is no need to land the plane – it never got off the ground). Even if an emergency landing is appropriate, such a complex series of decisions is involved in an emergency or precautionary landing, that modeling this aspect of the situation was impractical for this project. STAND-UP’s behavior is different during each of the two general steps that are handled in the interaction with the student. Even before the student declares him or herself ready to maintain aircraft control, a scenario is generated by STAND-UP and the student may query the system about the situation.
5.1 SCENARIO GENERATION

5.1.1 Choosing the Scenario

Ideally, an intelligent tutoring system should be able to generate appropriate problems for the student to work on, in this case appropriate emergency procedure scenarios. The scenarios should be appropriate in the sense that given a student model, the generated scenario is at the right level of difficulty for the student. Also, the scenario should cover new material, or material the student needs to review, as opposed to material of which student has already demonstrated sufficient knowledge.

Sometimes the educator has specific goals in mind for the tutoring session, usually based on previous experience with the student, and therefore would like to be able to dictate the parameters of the scenario, or at least the initial problem. Systems that provide this capability for the educator may wish to include some type of user authentication to confirm that the user who accesses this aspect of the system really is the educator.

Some developers of intelligent tutoring systems believe that rather than having the interaction dominated and controlled by the ITS, the students should be able to select the types of questions that he would like to work on next. These researchers insist that this type of student involvement lessens possible student resentment of a domineering, authoritative tutor, hopefully encouraging the student to become an active, enthusiastic participant in the educational process.

STAND-UP currently leaves the choice of scenario up to the student. Since it was built only as a prototype and its domain knowledge includes only the boldface procedures, STAND-UP is not likely to take the student through enough scenarios to develop the
student model to the point where it is useful for choosing appropriate scenarios. Another consideration, in addition to this matter of practicality, was the desire to provide a mechanism for some amount of control by the educator. The security issues that should be addressed for a functioning system with many users could be ignored in this case.

The result of these factors is a system that currently allows the student (or anyone) to initially choose the basics of the scenario. When the program starts up, the first screen the user sees is a window to choose the scenario. The user can then use this form to choose a general attitude (i.e. pitched up), altitude (i.e. low altitude) and emergency (i.e. engine fire). This form won't allow the user to choose an inconsistent scenario, such as an attitude with left bank while the plane is on the ground. This knowledge does not come from the knowledge base, but is hard coded into the form. STAND-UP’s current inference engine, M4, includes no mechanism that would allow STAND-UP to automatically extract the data from the knowledge base. The user is allowed to choose multiple emergencies, but he is responsible for not choosing multiple conflicting emergencies, as that would not meet the goals for the tutoring interaction. For example it is reasonable to expect that smoke and an engine fire could arise as concurrent emergencies. The goal of the system is to test the student’s knowledge of specific emergency procedures, not to overwhelm him or his with an impossible situation in which everything goes wrong.

5.1.2 Generating scenario-specific details

Once the user has selected the scenario, the system decides on specific values for all attributes of the situation, and symptoms for the requested emergencies. The specific
values relating to attitude and altitude are randomly generated according to a normal
distribution so that the scenarios always have realistic numbers. These numbers are within
the general range specified by the user. For example, the user’s choice of ‘left bank’ might
become ‘15 degrees left bank’.

The range of possible value for the user’s general selection is stored separately
from the knowledge base and can be modified by the user. Again, in a system which
differentiated between user and educator capabilities, modification of these ranges should
be restricted to the educator. The fact that these ranges can be modified makes it trivial to
reflect in the tutoring system changes to the planes capabilities. This process is used to
arrive at values for such attributes of the scenario as angle of bank and pitch, altitude,
vertical velocity, airspeed, and heading.

The process is slightly different for choosing appropriate symptoms of the chosen
emergency. Usually symptoms of an emergency are either present or not, so the possible
values of the symptom attributes are usually one of a small number of options rather than a
range. For instance, the exhaust temperature can be high, normal or fluctuating. The
options for these values are derived from the knowledge base. They are not, at this point,
automatically taken from the knowledge base, but are hard-coded into the tutoring module
according to the knowledge base. This means that they are not as easily modified as the
other settings. For an emergency which may have multiple symptoms, some number of
the symptoms are chosen at random. The values decided on here will be used to later
answer the student and the domain expert’s questions about the situation.
5.1.3 Scenario presentation

Once the user has selected the general parameters of the scenario and the specific values of the situations attributes have been calculated by the pedagogical module, the user is presented with the scenario in the form of the screen shown in figure 22. The student can click on the different dials to learn more about the scenario, until he feels ready to take the first step in any emergency - maintain aircraft control, i.e., bring the plane into straight and level flight. In order to do this, the student must know the orientation of the aircraft. To find this information out, the student can query the various dials as shown in figure 22. The dials do not automatically display the correct values, because what information the student chooses to collect is valuable pedagogical information. This way, as the student requests information, that request is logged.

Any initial information about the emergency symptoms that the tutoring module chooses to present to the student and the domain expert, is presented directly – the student does not need to make any queries about it. For example, if the emergency is an engine fire, one (or both) of the engine fire lights will be turned on. Other symptoms are presented textually in a message box. For instance, 'the left engine is wind-milling’ or ‘the plane is on short final.’
Figure 22 - STAND-UP Dial Display
5.2 MAINTAIN AIRCRAFT CONTROL

5.2.1 Student's Actions

When the student feels that he has enough information about the attitude and path of flight, he will select ‘Take Action’ from the File menu. This brings up a window that offers the student a choice of the three general emergency actions, with the focus on Maintain Aircraft Control. Selecting that button will cause the ‘Maintain Aircraft Control’ form to appear. Using this form, the student can adjust the pitch, bank, and airspeed of the aircraft by means of sliders labeled with the amount that a specific attribute is to be adjusted. This is a very crude mechanism, but the main point of the exercise is to remind the student of the importance of maintaining aircraft control before moving on to diagnose and rectify the emergency situation. If the student adjusts the pitch to zero for level flight, the vertical velocity is automatically adjusted to zero also. This is a simplification of the aerodynamics involved, but a reasonable abstraction to make. When the student clicks ‘Ok’ the form disappears and the global values for the scenario are updated to reflect the student’s changes.

If the student attempts to make a change that will put the value outside of the legal limits for that attribute (for instance, if the plane is pitched up 50 degrees and the student attempts to pitch it up 50 more degrees for a total of 100 degrees), an error message informs the student that the value is out of range and the change will not be allowed. If the student is not confident that his changes will result in straight and level flight, he is free to check the dials again, confirming that the attitude indicator and other gauges reflect that...
the student has successfully maintained aircraft control. If the changes were not sufficient to level off the plane, the student may undergo the whole process again.

5.2.2 Pedagogical Controller's Actions

Once the student has successfully maintained aircraft control, the pedagogical module passes the revised scenario, with the plane in straight and level flight, over to the domain expert, supplying the same information that was volunteered to the student. The domain module, using the knowledge base and the M4.VB inference engine, attempts to diagnosis the situation and recommend actions as previously described in Chapter 3.

As M4.VB works through the problem it asks questions of the tutoring module. The tutoring module answers these questions, using the values decided upon during the scenario generation, and marks the questions as asked by M4.VB. In addition, also as previously described, as various rules succeed, the pedagogical module marks them as used by M4.VB, either for the diagnosis or the actions depending on which is being sought. When the domain module has concluded its analysis of the situation, the pedagogical module notes that it is finished and awaits the conclusions of the student so that they may be compared.

5.3 Analyze the Situation and Take the Proper Actions

Once the student feels ready, he may enter the next phase of the interaction by again bringing up the 'Take Action' form and this time selecting the button marked 'Analyze the Situation and Take the Proper Actions.' The student will then have the opportunity to ask questions, diagnosis the situation and recommend actions.
5.3.1 Questions

There are many interfaces that would work well for the student to ask questions of the system and different intelligent tutoring systems utilize different techniques. Given the unlimited resources, perhaps one of the best ways to handle this interaction would be to have a speech recognition system use a natural language processor to divine the content of a student’s spoken questions. Perhaps an more sophisticated internal model of the aircraft’s systems would be available to answer these unanticipated questions. However, these technologies were out of the league of this particular project.

The way that STAND-UP handles this issue is straight-forward. A list of questions in the knowledge base is automatically generated when the pedagogical controller initially parses the file. Because it is taken from the knowledge base, it is exactly the same set of questions that the domain module may ask to solve the problem, ensuring that the student and the domain module have comparable resources available. The questions are presented to the student as a multiple choice list.

Although this way of dealing with the student’s question does have drawbacks, including the fact that the student may not be able to ask the questions he would like to, it also has several advantages. The questions a student asks are quite telling about his understanding of the material. The ability to directly compare these questions with those M4 asks is a pedagogical advantage. The list of questions can also be manipulated to control the difficulty level of the problem as it is presented to the student. A long list of questions, some of them quite particular, might be perfect for an advanced student, while a shorter list, with only a few questions that aren’t directly relevant might be easier for a less
advanced student to handle. A standard list of questions eliminates the need for the
student to do any typing (which might be a distraction for some) and also eliminates
possible misunderstandings.

As the student selects various questions to ask by double-clicking them on the list,
the tutoring module answers the questions of the student with the same answers it would
give to the domain module. As it answers the student’s questions, the system also marks
the questions as asked by the student. The student may spend as long as he wishes asking
questions, even repeating questions if he feels the need.

5.3.2 Diagnosis

Once the student has finished asking questions, the next window offers a choice of
possible diagnoses. The choice of diagnoses is presented as a list of options from which
the student may choose one. The reasons for this design decision parallel those expressed
above in regards to the list of questions offered to the student. Since the student can only
choose one diagnosis, he must not only identify which emergencies are present, but which
of them is the most pressing and will actually guide his actions. To reuse the abort
early example once again, the situation of an engine fire on take-off which causes a lot of smoke
is correctly diagnosed as abort, as opposed to smoke or engine fire.

If the student’s diagnosis matches that of the domain module, then the student is
credited with the knowledge required to make the correct diagnosis and the student is able
to move on to listing the appropriate actions. Assigning credit is a simple matter if the
student’s diagnosis is correct. Each of the rules that are tagged as used by the domain
expert to make the diagnosis are additionally tagged as known by the student.
Assigning credit is not such a simple matter if it is possible that there are multiples paths through the knowledge base leading to a correct solution of the problem. In STAND-UP, there is only one path, but in other intelligent tutoring systems built using the proposed approach, that assumption might not hold. In that case, the system would face the same problems assigning credit or blame.

Assigning blame for the student’s failure to diagnosis the situation correctly is much more difficult. The problem is that quite a few levels of inference must navigated correctly to successfully arrive at a diagnosis, and an error on any of these levels would easily propagate.

A comparison of the questions asked by the student and M4 can be somewhat helpful, but if the student neglected to ask a key question it is not clear why. Did he know the applicable rule and just failed to apply it because the forgot to ask the question? Or did he not ask the question because his reasoning had already gone awry?

Using intermediate stages to brake down the long inference chains is also helpful, and is why the system checks that the student has the correct diagnosis before moving on to the recommended actions. In some intelligent tutoring systems this strategy is followed closely, allowing the student to move only in small steps without the system’s guidance. However, that is not practical for STAND-UP. Few students would appreciate being forced to move through the situation in an excruciatingly slow manner, explicitly stating their most basic and obvious thoughts and conclusions.

One method for pinpointing a student’s errors that is not used in this project, but could prove useful in other systems built using the same approach, would be to
systematically, temporarily, remove rules from the knowledge base and rerun M4 to see if the domain now arrives at the same faulty diagnosis as the student. This method would probably be slow, but might point out the exact rules that the student doesn’t know. If several rules were missing, it would be possible to see if they shared an attribute or some other characteristic and thus develop a more general concept of where the student’s knowledge was lacking.

For a more concrete example, consider the abort situation presented in the last chapter. In this case, assume that the student misdiagnosed the situation, simply diagnosing as an engine fire rather than an abort. All of the rules M4 used to arrive at the diagnosis, seven different rules, are added to the list of rules that the system must suspect that the student doesn’t know. Because so much information was presented in the statement of the scenario, there are really no relevant questions that the tutoring module can use to winnow down the list of suspect rules. All seven of these rules will be marked as unknown by the student, even those that deal with recognizing an engine fire.

5.3.3 Actions

If the student has correctly diagnosed the situation, the actions window pops up. The actions window consists of a list of actions that is identical to the set of actions it is possible for the domain module to select. The student selects an action by double-clicking on it. This moves the selected action into the student’s list of recommended actions. The student can remove actions from his list if he changes his mind or accidentally adds to the list.
When the student has completed this operation (signified by clicking ‘Ok’), the tutoring module begins the final analysis of the student’s response in order to credit the student model with demonstrated knowledge. Before doing a detailed analysis, STAND-UP checks quickly to determine if the comparison between the student’s list of actions and the domain expert’s actions yields either of two outcomes – a perfect match or correct except for ordering. In these two cases, the student is given credit for all the rules the domain module used to arrive at a conclusion, noting, if applicable, that the student ordered the actions incorrectly.

If the student’s action list is missing actions, the difficult problem of assigning blame again arises. In this case however, the problem is not as difficult to solve as when dealing with the diagnoses, because there are few layers of inference between the diagnosis and the final actions. In most cases the student’s action list and a comparison of the questions asked by the student and M4 are enough to pinpoint the rule (or rules) unknown by the student. To revisit the example in Chapter 4, a comparison of the student’s action list and M4’s action list shows that the student is missing the

<table>
<thead>
<tr>
<th>M4’s Actions</th>
<th>Student’s Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throttles - Cutoff</td>
<td>Throttle - Cutoff (Malfunctioning Engine)</td>
</tr>
<tr>
<td>Wheel Brakes - As Required</td>
<td>Wheel Brakes - As Required</td>
</tr>
</tbody>
</table>

‘Throttles - Cutoff’ action. A look at the rule base reveals rule-65:
Rule 65's conclusion is ‘then action = ‘Throttles - Cutoff’’ so it is clearly the rule the student should have applied. Checking the premise to see why the student failed to use the rule, STAND-UP finds diagnosis = abort. Because the interaction is broken into an intermediate step for the diagnosis, STAND-UP knows that the student is clear on the first part of the premise. The rule also depends on a second attribute, stopping distance. STAND-UP can now check for the question with the attribute stopping-distance to see if it was asked by M4 and/or the student. If it was asked, then the student doesn’t know the rule. If the student didn’t ask the question, the cause might be that he didn’t know the rule or that he simply didn’t think to ask. In either case, all the other rules marked as used by M4 for the actions can safely be credited to the student and rule 65 marked as demonstrated unknown by the student.

6. CONCLUSIONS

STAND-UP evolved continually over the course of the project, sometimes performing marvelously, other times not quite meeting expectations. In this last chapter I will discuss some of the conclusions that resulted from this work. I will distinguish
between the successes and failures of the implementation of STAND-UP and the merits of the approach I proposed back in the introduction. Before getting into that, I would like to address one issue that had a large effect on the success of the implementation of STAND-UP.

6.1 **Specialized Knowledge-Based System Tools**

One difficulty I wrestled with throughout the project was wringing the functionality I needed from a traditional knowledge based system and inference engine. In fact I tried and discarded one expert systems tool before implementing STAND-UP with M4. Early on I considered just building my own custom inference engine and knowledge representation, but deemed it too difficult for the project at hand. As it turned out, I ended up parsing the knowledge base and keeping a separate internal representation of the rules anyway.

6.1.1 **Knowledge Representation**

A knowledge representation designed specifically for intelligent tutoring systems could be expected to support features that are useful in the context of learning software but not necessarily useful in traditional knowledge based systems applications. In addition to supporting the usual certainty factor tag on a rule, for instance, a specialized knowledge representation could also have a tag for the degree of difficulty represented by the particular rule. There could also be some mechanism for supporting pedagogical rules and knowledge as opposed to domain knowledge. Some intelligent tutoring systems currently have a pedagogical module which is a separate expert system and some expert systems
provide meta rules which aren't really part of the domain. Some combination of these two methods would allow concepts like ordering to be part of the domain knowledge, and thus part of the student model overlay.

6.1.2 Inference Engine

A special purpose inference engine could also have several features built in that would be quite useful for an intelligent tutoring system, but are somewhat of a pain to hack. For instance, a special purpose inference engine could include a function for pinpointing a student's error in a long inference chain by selectively removing rules, as described in Chapter 5. Significantly more efficient algorithms might also be utilized to assign credit and/or blame, such as those used in theorem provers. These algorithms all already optimized to find a path to a particular conjecture given a set of axioms.

The ability to use the rules in the reverse direction from normal would also be useful, in order to avoid duplicating the knowledge of the knowledge base in the tutoring module. For example, the system knows which attribute/value pairs are symptomatic of particular emergencies. Therefore, shouldn't it be able to generate a plausible set of symptoms given an emergency? This sort of information could be useful in generating the specific values for the scenarios, as described in Chapter 5. It would also help make the system considerably more modular and easier to update, since all the domain knowledge would be in the knowledge base, as opposed to having necessary bits and pieces hard-coded elsewhere throughout the system.

One comparison between the expert system and the student that I thought would be useful when I began this project was the ordering of the questions. I wanted to be able
to determine if the student was asking irrelevant questions, thus demonstrating that he didn’t clearly understand what the priorities of the situation were. As it turns out, however, most expert systems ask plenty of questions that a human expert would consider irrelevant for the problem at hand. And, although the builders of knowledge based systems can sometimes kludge their way into having the system ask questions in a reasonable order, this ordering is an usually an artifact of that kludge, not any knowledge about priorities. A special purpose inference engine, however, could be built with this issue in mind.

6.1.3 Not Hard

As previously mentioned, the main reason that I did not go ahead and write a custom knowledge representation and inference engine for STAND-UP, was that I believed it would be prohibitively difficult, especially considering that commercial products were available. I didn’t realize how useful in so many different ways a special purpose inference engine would have been. Since, by the conclusion of the project, I had implemented a very basic parser for the knowledge base and was able to perform some crude manipulations of the knowledge base, I now believe it would have been well worth the not unreasonable amount of effort required.

6.2 Evaluation of STAND-UP

STAND-UP performs well in many aspects. It was meant to be a proof-of-concept to test the proposed approach. It does not, by any means test the full potential of that approach, but it does prove that the approach is viable. The mere fact that a single
MEng student working for a term was able to produce a working system is a strong advocate for the effectiveness of the approach. So, in that sense, STAND-UP is a success. However, it is a prototype, not currently ready for mass-distribution. The final product is not terribly polished, partly because the emphasis of the project was on the interesting issues of this project, not the everyday coordination issues of any software project.

Because I chose the domain for its appeal to me, as opposed to for its wide-spread appeal, the domain is one in which very few randomly selected people would be able to perform at all. The very specific nature of the boldface procedures accentuates this problem. This means that the system was tested by the same person by whom it was implemented (and who could be said to have a vested interested). With fewer time constraints, this would have been less of an issue.

STAND-UP’s most obvious flaw is the weakness of the methods of the pedagogical module. The differential modeling works well, but to conduct a truly effective tutoring session other guidelines for the interaction are also required. The proposed approach leaves plenty of flexibility to the implementer, but if the implementer knows much more about the domain than about education that flexibility can turn into a weakness. There is no unifying theory or goal behind the tutoring models methods for correcting the student’s error, encouraging his successes or guiding a student in need of a little assistance.
6.3 EVALUATION OF THE APPROACH

As discussed at several points in this paper, it can be extremely difficult using this approach to pinpoint a student’s errors, especially where long chains of inferences are required to arrive at the proper solution. This is a significant problem, since knowing where the student’s reasoning has gone astray is critical in an intelligent tutoring system. However, I believe this problem might be addressed by the use of a special purpose knowledge representation and inference engine, and as such, is not an inherent flaw in the proposed approach.

Another problem with the approach that might be ameliorated by a special purpose expert system tool is the fact that currently the tutoring module makes no attempt to take into account the fact that students are much more likely to have trouble with some rules than others. For example, the rule that states that an altitude and airspeed of zero signal that the plane is on the ground, is most likely not the rule that the student is having trouble with in a chain of inferences. A knowledge representation that could take into account the difficulty of particular rules would alleviate this problem.

A real problem with the proposed approach is that having the student model be an overlay of the domain knowledge makes no allowance for valuable information about the student that is independent of the domain. For instance, while the proposed approach is great at expressing the fact that the student understands the rule about being on the ground, there is no mechanism for expressing the fact that the student has trouble remembering the actions associated with any given diagnosis. As the system becomes more sophisticated in adaptations to the student, this information will become more
important. Quite probably the solution is to expand the student model to be a collection of information about the student, of which the overlay of the domain knowledge is just one facet. This is already true to some degree in STAND-UP which, in addition to the student model, marks the questions that the student has asked.

On the other hand, the suggested approach for dealing with the domain module - building a rule-based system directly from knowledge contained in texts or manuals - worked great in STAND-UP. Knowledge acquisition was not a problem and the knowledge base grew with the project. The domain module successfully reasons about the scenarios presented to the student. Adding a specialize inference engine as described above would increase the effectiveness of the student model and the tutoring module, but even without that addition, the domain module performs its function well.

The basic framework of the approach is extensible and flexible. As has been mentioned several times throughout this report, implementation of STAND-UP generated a whole host of potential features that could be incorporated into the system with relatively little trouble. For instance, user authentication and speech recognition would enhance the interaction with the student.

In that same vein, the basic framework leaves room for much more sophisticated pedagogical methods than were implemented in STAND-UP. Even as implemented in STAND-UP, there are many ways to individualize the interaction with the student. Some of these include:
• Presenting the student with fewer or more details of the scenario initially. More
details and the situation almost diagnoses itself. Fewer and the student must
already know what to look for in the situation in order to know what questions to
ask.

• Offering more or fewer questions at the query window and actions in the action
window. With fewer questions to choose from, hopefully it becomes more
obvious which of them are relevant. More questions eventually make it harder for
the student to use the list of questions as a crutch.

• Requiring the student to only either diagnosis the situation or recommend the
appropriate actions. Diagnosis of the situations may be beyond beginning students
who have just barely managed to memorize the boldface procedures. Other
students may be interested only in the diagnosis phase of the problem since that is
the more interesting part of the problem.

A bonus of these methods of individualizing the interaction is that they do not
depend on assigning the student to a discrete level of proficiency that restricts all
interaction between the student and the intelligent tutoring system to the specified level.
These methods can be targeted directly to a student’s strengths and weaknesses instead of
treating all students as either ‘experts’ or ‘novices.’

6.4 FINAL CONCLUSIONS

The three-part proposed approach, which builds on the traditional model for
intelligent tutoring systems, is proved viable by the implementation of STAND-UP, an
intelligent tutoring system for the T-37B-1 Flight Manual. Many of the issues left
unresolved by this project could be addressed by complementing the proposed approach with a special purpose knowledge representation and inference engine.

In order to make progress in any endeavor, it is necessary to build on the work of those who came before you. Much current work in intelligent tutoring systems, however, ignores that simple fact. Without a doubt, there is work of enduring significance to be done in this field. By harnessing the power of computers for furtherance of educational goals, the field of intelligent tutoring is in a position to make a significant difference to the quality of human life. Education is the greatest empowerment tool of human society. But for progress to be made, researchers must avail themselves of the best ideas and techniques that have been fostered by their predecessors. In addition, they must choose worthwhile projects with the potential to be augmented by others in their turn, promoting basic concepts that can be reused by others in the field. My approach takes the traditional, loosely formulated concepts of intelligent tutoring systems, and fleshes out a flexible, effective framework that can be built upon in a myriad of ways to fashion potentially quite powerful systems.
7. APPENDIX A - STAND-UP CODE

'****** Form Action
'Allow the student to take action based on the three
'essential emergency actions

Option Explicit

Sub cmdAnalyze_Click ()
'Add to the history that the user is analyzing
AddAction (cmdAnalyze.Caption)
frmAction.Hide

'let the user ask questions about the situation
frmQuery.Show 1

'frmEmergencies.Show 1

End Sub

Sub cmdLand_Click ()
AddAction (cmdLand.Caption)

End Sub

Sub cmdMaintain_Click ()

'Hide this form and Show the maintain form
frmAction.Hide

'Report that user is ready to maintain aircraft control
AddAction (cmdMaintain.Caption)

frmMaintain.Show

End Sub

'****** Form Diagnose

Option Explicit

Sub cmdCancel_Click ()
optSmoke = False
optEngineFire = False
optAbort = False
optFailure = False
optEgress = False
optOverheat = False

frmDiagnose.Hide
End Sub

Sub cmdOk_Click ()
frmDiagnose.Hide

If optEngineFire Then
    StudDiagnosis = "engine-fire"
ElseIf optAbort Then
    StudDiagnosis = "abort"
ElseIf optEgress Then
    StudDiagnosis = "ground-egress"
ElseIf optSmoke Then
    StudDiagnosis = "smoke"
ElseIf optOverheat Then
    StudDiagnosis = "overheat"
ElseIf optFailure Then
    StudDiagnosis = "two-engine-failure"
End If

'Compare what the student just did with what M.4 decided
CompareDiagnoses

End Sub

Sub Form_Load ()
opSmoke = False
optEngineFire = False
optAbort = False
optFailure = False
optEgress = False
optOverheat = False
End Sub

'******* Form DialDisplay
'Main form of the applications.
'Contains all the instruments, which can be read by double clicking
'Single Clicking, or getting focus gives the name of the object
'Losing focus will clear its value and name from the status bar

'During Initialization, opens the data file
'Beings with Senario Chooser

'Menu options bring up the senario chooser and olimit editor
Option Explicit

Sub cmdExit_Click()
'M4cmd ("exit")
Close ExFile
Close HistFile
Close FileNum
End

End Sub

Sub Command1_Click()
Dim Sting1, Sting2, Sting3, NL As String
Dim i As Integer

NL = Chr(13) + Chr(10)

Sting1 = "Rules Used by Student:"
For i = 0 To UBound(Rules)
    If Rules(i).StudKnows = 1 Then
        Sting1 = Sting1 + Str(i) + " "
    End If
Next i

Sting2 = "Rules Used for diagnosis:"
For i = 0 To UBound(Rules)
    If Rules(i).M4DiagRule = True Then
        Sting2 = Sting2 + Str(i) + " "
    End If
Next i

Sting3 = "Rules Used for actions:"
For i = 0 To UBound(Rules)
    If Rules(i).M4ActRule = True Then
        Sting3 = Sting3 + Str(i) + " "
    End If
Next i

MsgBox Sting1 & NL & Sting2 & NL & Sting3
If perfect Then
    MsgBox "Perfect!"
ElseIf WrongOrder Then
    MsgBox "Wrong Order!"
End If

End Sub
Sub Form_Load()
Dim NL

'Show Form with values hidden
DialDisplay.Show
TxtAttitude.Visible = False
TxtAttitude2.Visible = False
TxtHeadInd.Visible = False
TxtVVI.Visible = False
txtAirspeed.Visible = False
txtAltimeter.Visible = False
cmdExit.SetFocus

InitializeSession

'Show Choose Senario Form
'to pick altitude, attitude and emergencies
frmRange.Show 'Calls GetValues & InitializeSenario

Controller
End Sub

Sub mnuDiagnose_Click()
'Show form to Choose Maintain, Analyze or Land
frmAction.Show
End Sub

Sub mnuExit_Click()
'M4cmd ("exit")
Close HistFile
Close ExFile
Close FileNum
End
End Sub

Sub mnuKnowledgeBase_Click()
M4IOFlag = True
M4.Show
M4IOFlag = False
End Sub

Sub mnuOlimits_Click()
FrmOLimits.Show 1
End Sub

Sub mnuSenario_Click()
frmRange.Show
End Sub

Sub picAirspeed_Click ()
'StatusBar1.SimpleText = "Gauge is: Airspeed Indicator"
End Sub

Sub picAirspeed_DblClick ()
'Show Value
'StatusBar1.SimpleText = "Gauge is: Airspeed Indicator"
txtAirspeed.Text = Str(valAirspeed) & " knots"
txtAirspeed.Visible = True

'Report the user's interest
AddDisplayQuery "Airspeed Indicator", Str(valAirspeed)
End Sub

Sub picAirspeed_GotFocus ()
'StatusBar1.SimpleText = "Gauge is: Airspeed Indicator"
End Sub

Sub picAirspeed_LostFocus ()
'StatusBar1.SimpleText = ""
txtAirspeed.Visible = False
End Sub

Sub picAltimeter_Click ()
'StatusBar1.SimpleText = "Gauge is: Altimeter"
End Sub

Sub picAltimeter_DblClick ()
'Show Value
'StatusBar1.SimpleText = "Gauge is: Altimeter"
txtAltimeter.Text = Str(valAltitude) & " feet"
txtAltimeter.Visible = True

'Report the user's interest
AddDisplayQuery "Altitude Indicator", Str(valAltitude)
End Sub

Sub picAltimeter_GotFocus ()
'StatusBar1.SimpleText = "Gauge is: Altimeter"
End Sub

Sub picAltimeter_LostFocus ()
'StatusBar1.SimpleText = ""
txtAltimeter.Visible = False
End Sub

Sub PicAttitude_Click()
'StatusBar1.SimpleText = "Gauge is: Attidtude Indicator"
'Test new thang
'PicAttitude.Cls
'PicAttitude.Print "Gauge is: Attidtude Indicator"
End Sub

Sub PicAttitude_DblClick()
'Currently using hardwired value
'Dim Hardwire As String
'Hardwire = "Straight and Level"

'Show Value
'StatusBar1.SimpleText = "Gauge is: Attitude Indicator"
'TxtAttitude.Text = Hardwire

'Decide if straight and level, else display individual values
If valAttitudeB = 0 And valAttitudeP = 0 Then
    TxtAttitude.Text = "Straight and Level"
    TxtAttitude.Visible = True
Else
    TxtAttitude.Text = "Pitch: " + Str(valAttitudeP) + " degrees"
    If valAttitudeB < 0 Then
        TxtAttitude2.Text = "Bank: " + Str(-valAttitudeB) + " degrees Left"
    Else
        TxtAttitude2.Text = "Bank: " + Str(valAttitudeB) + " degrees Right"
    End If
    TxtAttitude.Visible = True
   TxtAttitude2.Visible = True
End If

'Report User's Interest
AddDisplayQuery "Attitude Indicator - (Pitch)", Str(valAttitudeP)
AddDisplayQuery "Attitude Indicator - (Bank)", Str(valAttitudeB)
End Sub

Sub PicAttitude_GotFocus()
'StatusBar1.SimpleText = "Gauge is: Attitude Indicator"
End Sub

Sub PicAttitude_LostFocus()
'StatusBar1.SimpleText = ""
TxtAttitude.Visible = False
TxtAttitude2.Visible = False

'Clear Text
Sub PicHeadInd_Click()
'StatusBar1.SimpleText = "Gauge is: Heading Indicator"

'PicHeadInd.Cls
'PicHeadInd.Print "Gauge is: Heading Indicator"
End Sub

Sub PicHeadInd_DblClick()
'Show Value
'StatusBar1.SimpleText = "Gauge is: Heading Indicator"
TxtHeadInd.Text = Str(valHeadingInd) & " degrees"
TxtHeadInd.Visible = True

'Report the user's interest
AddDisplayQuery "Heading Indicator", Str(valHeadingInd)
End Sub

Sub PicHeadInd_GotFocus()
'StatusBar1.SimpleText = "Gauge is: Heading Indicator"
End Sub

Sub PicHeadInd_LostFocus()
TxtHeadInd.Visible = False
'StatusBar1.SimpleText = ""
PicHeadInd.Cls
End Sub

Sub PicVVI_Click()
'StatusBar1.SimpleText = "Gauge is: Vertical Velocity Indicator"

'PicVVI.Cls
'PicVVI.Print "Gauge is: Vertical Velocity Indicator"
End Sub

Sub PicVVI_DblClick()
'Show Value
'StatusBar1.SimpleText = "Gauge is: Vertical Velocity Indicator"
TxtVVI.Text = Str(valVVI) & " ft/s"
TxtVVI.Visible = True

'Report User's Interest
AddDisplayQuery "Vertical Velocity Indicator", Str(valVVI)
End Sub

Sub PicVVI_GotFocus()
'StatusBar1.SimpleText = "Gauge is: Vertical Velocity Indicator"
End Sub

Sub PicVVI_LostFocus ()
TxtVVI.Visible = False
'StatusBar1.SimpleText = ""
PicVVI.Cls
End Sub

Sub txtM4IO_Change ()
'A question or conclusion has appeared from M.4
'This sub answers the question

If txtM4IO.Text = "" Then
    Exit Sub
End If

'First add M4's query to the history of M4's actions
AddM4Question (txtM4IO.Text)

'Figure out if it's an important conclusion

If InStr(txtM4IO.Text, "QUESTION") <> 0 Then
    AnsM4Question (txtM4IO.Text)
    txtM4IO.Text = ""
ElseIf InStr(txtM4IO.Text, "CONCLUSION") <> 0 Then
    'MsgBox ("Conclusion Reached.")
    ImpM4Conclusion (txtM4IO.Text)
End If

End Sub

‘******* Form Find
'Provides a dialog box to get the olimit the user wants to find.
The user can either type it in, or select it from the combo box

Option Explicit

Sub cmdCancel_Click ()
FrmFind.Hide
End Sub

Sub cmdOk_Click ()
FrmFind.Tag = cmbFind.Text
FrmFind.Hide

75
Sub Form_Activate()
' Puts all the options in the list
Dim TmpCurrRec, TmpLastRec As Long
Dim TmpLimit As OLimit

cmbFind.Clear
' Add Items to the List
For TmpCurrRec = 1 To LastRecord
    Get #FileNum, TmpCurrRec, TmpLimit
    cmbFind.AddItem Trim(TmpLimit.Name) + " - " + Trim(TmpLimit.Range)
Next
End Sub

'******** Form Maintain
' Allow the student to adjust the pitch, bank and airspeed
' In order to maintain aircraft control

Option Explicit

Sub ChangeValues()
' Implements the changes the user indicated
Dim tester As Integer
Dim TmpLimit As OLimit

' Need to check against olimits
' AttitudeB
 tester = valAttitudeB + HSBank.Value
 If tester < 0 Then ' left
     FindLimit("Attitude Indicator (Bank) - Left Turn")
     TmpLimit = FindLimitHack
     If tester >= TmpLimit.Lower And tester <= TmpLimit.Upper Then
         valAttitudeB = tester
     Else ' Adjusted value out of limits
         MsgBox "Adjusted value is out of Limits! Value unchanged."
         HSBank.Value = 0
         AddAction "User failed to adjust bank to: " & Str(tester)
     End If ' In Limits
 Elseif tester >= 0 Then ' Right
     FindLimit("Attitude Indicator (Bank) - Right Turn")
     TmpLimit = FindLimitHack
     If tester >= TmpLimit.Lower And tester <= TmpLimit.Upper Then
         valAttitudeB = tester
     Else ' Adjusted value out of limits
         MsgBox "Adjusted value is out of Limits! Value unchanged."
     End If
 End If
End Sub
HSBank.Value = 0
    AddAction "User failed to adjust bank to: " & Str(tester)
End If 'In Limits
End If

'AttitudeP
tester = valAttitudeP + HSPitch.Value
If tester < 0 Then 'dive
    FindLimit ("Attitude Indicator (Pitch) - Dive")
    TmpLimit = FindLimitHack
    If tester >= TmpLimit.Lower And tester <= TmpLimit.Upper Then
        valAttitudeP = tester
    Else 'Adjusted value out of limits
        MsgBox "Adjusted value is out of Limits! Value unchanged."
        HSPitch.Value = 0
        AddAction "User failed to adjust pitch to: " & Str(tester)
    End If 'In Limits
Elseif tester >= 0 Then 'Climb
    FindLimit ("Attitude Indicator (Pitch) - Climb")
    TmpLimit = FindLimitHack
    If tester >= TmpLimit.Lower And tester <= TmpLimit.Upper Then
        valAttitudeP = tester
    Else 'Adjusted value out of limits
        MsgBox "Adjusted value is out of Limits! Value unchanged."
        HSPitch.Value = 0
        AddAction "User failed to adjust pitch to: " & Str(tester)
    End If 'In Limits
End If

'VVI
'This can be expanded to determine a pitch or climb
If valAttitudeP = 0 Then
    valVVI = 0
End If

'Airspeed
tester = valAirspeed + HSAirspeed.Value
FindLimit ("Airspeed Indicator - Normal")
TmpLimit = FindLimitHack
'Zero Hardcoded here to avoid some problems
If tester >= 0 And tester <= TmpLimit.Upper Then
    valAirspeed = tester
Else 'Adjusted value out of limits
    MsgBox "Adjusted value is out of Limits! Value unchanged."
    HSAirspeed.Value = 0
    AddAction "User failed to adjust airspeed to: " & Str(tester)
End If 'In Limits
Sub cmdCancel_Click()
'Zero differences, then leave
HSPitch.Value = 0
HSBank.Value = 0
HSAirspeed.Value = 0
frmMaintain.Hide
End Sub

Sub cmdOk_Click()
Dim Sting As String

'Actually change the values displayed
ChangeValues

'Report the adjustments to the history
If HSPitch <> 0 Then
    Sting = "Adjust Pitch by " & Str(HSPitch.Value) & "degrees"
    AddAction (Sting)
End If

If HSBank <> 0 Then
    Sting = "Adjust Bank by " & Str(HSBank.Value) & "degrees"
    AddAction (Sting)
End If

If HSAirspeed <> 0 Then
    Sting = "Adjust Airspeed by " & Str(HSAirspeed.Value) & "degrees"
    AddAction (Sting)
End If

frmMaintain.Hide
MaintainFlag = True
End Sub

Sub Form_Activate()
'Initializes the values of all the scrollbars
HSBank.Value = 0
HSPitch.Value = 0
HSAirspeed.Value = 0
End Sub

Sub HSAirspeed_Change()
lblAirspeed.Caption = HSAirspeed.Value
End Sub
Sub HSAirspeed_Scroll()
    lblAirspeed.Caption = HSAirspeed.Value
End Sub

Sub HSBank_Change()
    If HSBank.Value > 0 Then 'Bank is Right
        lblBank.Caption = HSBank.Value
    Else 'Bank is Left
        lblBank.Caption = -HSBank.Value
    End If
End Sub

Sub HSBank_Scroll()
    If HSBank.Value > 0 Then 'Bank is Right
        lblBank.Caption = HSBank.Value
    Else 'Bank is Left
        lblBank.Caption = -HSBank.Value
    End If
End Sub

Sub HSPitch_Change()
    lblPitch.Caption = HSPitch.Value
End Sub

Sub HSPitch_Scroll()
    lblPitch.Caption = HSPitch.Value
End Sub

'*** Form OLimits
'Edit the Olimits and Ranges

'Currently, Names are:
'  Altimeter
'  Attitude Indicator
'  Airspeed Indicator
'  Heading Indicator
'  Vertical Velocity Indicator

'Ranges are:
'  Straight and Level
'  Climb
'  Dive
'  Left Turn
'  Right Turn
'  Normal
'  On Ground
'  Take Off Roll
'  Low Altitude

79
Option Explicit

Sub Form_Load ()
' Display the Current Record
ShowCurrentRecord
End Sub

Sub mnuDelAll_Click ()
' Will erase the Entire Data File
If MsgBox("Are you sure you want to Delete All?", 49) = 1 Then
' Kill the old data file
Close FileNum
Kill "OLimit.dat"

' Open a new file
Initialize_DataFile

ShowCurrentRecord
End If

' Do nothing if they changed their mind
End Sub

Sub mnuDelete_Click ()
' Deletes the currently displayed olimit with confirmation
Dim DirResult
Dim TmpFileNum
Dim TmpLimit As OLimit
Dim RecNum As Long
Dim TmpRecNum As Long

' Get Confirmation
If MsgBox("Delete this record?", 4) <> 6 Then
' They changed their mind - do nothing
txtName.SetFocus
Exit Sub
End If

' Make sure OLimit.tmp doesn't already exist
If Dir("OLimit.tmp") = "OLimit.tmp" Then
Kill "OLimit.tmp"
End If

' Open a temporary file to copy all records except current
TmpFileNum = FreeFile
Open "OLimit.tmp" For Random As TmpFileNum Len = RecordLen
RecNum = 1
TmpRecNum = 1
'Copy other records
Do While RecNum < LastRecord + 1
   If RecNum <> CurrentRecord Then
      Get #FileNum, RecNum, TmpLimit
      Put #TmpFileNum, TmpRecNum, TmpLimit
      TmpRecNum = TmpRecNum + 1
   End If
   RecNum = RecNum + 1
Loop

'Change the temp file to the new data file
Close FileNum
Kill "OLimit.dat"
Close TmpFileNum
Name "OLimit.tmp" As "OLimit.dat"

Initialize_DataFile

'Adjust current record if they deleted the last record
If CurrentRecord > LastRecord Then
   CurrentRecord = LastRecord
End If

ShowCurrentRecord
txtName.SetFocus

End Sub

Sub mnuExit_Click ()
FrmOLimits.Hide
End Sub

Sub mnuFind_Click ()
'Finds the requested olimit (supplied by find form) and makes it current
Dim NameToFind, ing As String
Dim Found As Integer
Dim RecNum As Long
Dim TmpLimit As OLimit

SaveCurrentRecord

'Get Name to search from user
FrmFind.Show 1
NameToFind = FrmFind.Tag
'Cancel if bogus input
If NameToFind = "" Then
    txtName.SetFocus
    Exit Sub
End If

'Otherwise go for it
NameToFind = UCase(NameToFind)
Found = False

'Search
For RecNum = 1 To LastRecord
    Get #FileNum, RecNum, TmpLimit
    tmp = UCase(Trim(TmpLimit.Name)) + " - " + UCase(Trim(TmpLimit.Range))
    If NameToFind = tmp Then
        Found = True
        Exit For
    End If
Next

'If found, make it current, else squeak
If Found Then
    SaveCurrentRecord
    CurrentRecord = RecNum
    ShowCurrentRecord
Else
    MsgBox "Name " + NameToFind + " not found!"
End If

txtName.SetFocus
End Sub

Sub mnuNew_Click ()
'Saves the current record, adds a new, blank record and displays
SaveCurrentRecord

'Add new, blank record
LastRecord = LastRecord + 1
Limit.Name = ""
Limit.Range = ""
Limit.Units = ""
Limit.Upper = 0
Limit.Lower = 0
Put #FileNum, LastRecord, Limit

'Update Current Record and display
CurrentRecord = LastRecord
ShowCurrentRecord
txtName.SetFocus

End Sub

Sub mnuNext_Click ()
'Will advance to the next record in the file, unless it's the last

'Check to see if last
If CurrentRecord = LastRecord Then
  Beep
  MsgBox "End of File!", 48
Else 'Show the next one
  SaveCurrentRecord
  CurrentRecord = CurrentRecord + 1
  ShowCurrentRecord
End If

txtName.SetFocus
End Sub

Sub mnuPrev_Click ()
'Goes back to previous record, unless first

'Check to see if first
If CurrentRecord = 1 Then
  Beep
  MsgBox "Beginning of File!", 48
Else 'Show previous
  SaveCurrentRecord
  CurrentRecord = CurrentRecord - 1
  ShowCurrentRecord
End If

txtName.SetFocus
End Sub

Sub SaveCurrentRecord ()
'Fill Limit with the currently displayed data
Limit.Name = txtName.Text
Limit.Range = txtRange.Text
Limit.Units = txtUnits.Text
Limit.Upper = txtUpper.Text
Limit.Lower = txtLower.Text

'Save to current record
Put #FileNum, CurrentRecord, Limit
End Sub

Sub ShowCurrentRecord()
'Fill Limit with data
Get #FileNum, CurrentRecord, Limit

'Display data
txtName.Text = Trim(Limit.Name)
txtRange.Text = Trim(Limit.Range)
txtUnits.Text = Trim(Limit.Units)
txtUpper.Text = Trim(Limit.Upper)
txtLower.Text = Trim(Limit.Lower)

'Set the Form Caption to include current record number
FrmOLimits.Caption = "Operational Limits Editor - Record #" & CurrentRecord
End Sub

'******** From Query
Option Explicit

Sub cmdCancel_Click()
lstQuery.ListIndex = -1
frmQuery.Hide
End Sub

Sub cmdOk_Click()
Dim Sting As String
'Hide the form
'frmQuery.Hide 'Maybe I want to let them keep asking

frmQuery.Hide
frmDiagnose.Show 1
End Sub

Sub Form_Load()
Dim i As Integer
'Adds the stuff into into the list box

For i = 0 To UBound(Attributes)
    lstQuery.AddItem Attributes(i).TheQuestion
Next i

End Sub

Sub lstQuery_DblClick()
This procedure works because I know what order the questions are in in the kb file.
Dim i As Integer
Dim Sting As String

'Add the question to the history
Sting = lstQuery.List(lstQuery.ListIndex)
AddQuery (Sting)

'Just as a quick check
'MsgBox lstQuery.List(lstQuery.ListIndex)
' & Attributes(lstQuery.ListIndex).TheQuestion

'Answer the question
For i = 0 To UBound(Attributes)
    If Attributes(i).TheQuestion = Sting Then
        If IsNumeric(Attributes(i).AttribVal) Then
            MsgBox (Str$(Attributes(i).AttribVal))
        Else
            MsgBox (Attributes(i).AttribVal)
        End If
    Exit For
    End If
Next i

'Select Case lstQuery.ListIndex

'Case 0
'    MsgBox (Str$(valAltitude))
'    Attributes(0).StudAsked = True
'    Student is inquiring about the airspeed
'Case 1
'    MsgBox (Str$(valAirspeed))
'    Attributes(1).StudAsked = True
'Case 2
'    If valFireLightRight = "yes" Then
'        MsgBox "Right"
'    ElseIf valFireLightLeft = "yes" Then
'        MsgBox "Left"
'    Else
'        MsgBox "No"
'    End If
'    Attributes(2).StudAsked = True
'    Student is inquiring about the fuel flow
'Case 3
'    MsgBox (valFuelFlow)
'    Attributes(3).StudAsked = True
'    Student is inquiring about the smoke in the cockpit
'Case 4
  ' MsgBox (valSmokeInside)
  ' Attributes(4).StudAsked = True
"Student is inquiring about the smoke outside the aircraft
'Case 5
  ' MsgBox (valSmokeOutside)
  ' Attributes(5).StudAsked = True
"Student is inquiring if the engine is running rough
'Case 6
  ' MsgBox (valRoughRunning)
  ' Attributes(6).StudAsked = True
"Student is inquiring about the exhaust temperature
'Case 7
  ' MsgBox (valExhaustTemp)
  ' Attributes(7).StudAsked = True
"Student is inquiring about the engine's RPM
'Case 8
  ' MsgBox (valRPM)
  ' Attributes(8).StudAsked = True
"Student is inquiring about the stopping distance
'Case 9
  ' MsgBox (valStoppingDistance)
  ' Attributes(9).StudAsked = True
'Case Else
  ' MsgBox "Unknown question"
'End Select

'Strig = ""
'For i = 0 To UBound(Attributes)
  ' If Attributes(i).StudAsked Then
    ' Sting = Sting + Str(i)
  ' End If
'Next i
'MsgBox Sting

End Sub

"******** Form Range
'Provides a dialog box to choose the scenario
'Broken into sections: attitude, altitude, and emergencies

'Clicking ok generates plausible values for the Dial Display form
' based on the chosen scenario

Option Explicit

Dim can_optLeft As Integer
Dim can_optRight As Integer
Dim can_optNoBank As Integer
Dim can_optGround As Integer
Dim can_optTORoll As Integer
Dim can_optLowAlt As Integer
Dim can_optNormal As Integer
Dim can_optClimb As Integer
Dim can_optDescend As Integer
Dim can_optNoPitch As Integer

'Emergency values are hardcoded for now
Dim can chkFire As Integer

Sub cmdCancel_Click()
'Reinstate original values first

optLeft.Value = can_optLeft
optRight.Value = can_optRight
optNoBank.Value = can_optNoBank
optClimb.Value = can_optClimb
optDescend.Value = can_optDescend
optNoPitch.Value = can_optNoPitch
optGround.Value = can_optGround
optTORoll.Value = can_optTORoll
optLowAlt.Value = can_optLowAlt
optNormal.Value = can_optNormal

'Currently Hardwired Emergency
chkFire.Value = can chkFire

frmRange.Hide

End Sub

Sub cmdOk_Click()
'Use the parameters of the senario to get plausible values for the display
GetValues
frmRange.Hide
InitializeSenario

End Sub

Sub Form_Activate()
'Saves current values in case of a cancel

can_optLeft = optLeft.Value
can_optRight = optRight.Value
can_optNoBank = optNoBank.Value
can_optClimb = optClimb.Value
can_optDescend = optDescend.Value
can_optNoPitch = optNoPitch.Value
can_optGround = optGround.Value
can_optTORoll = optTORoll.Value
can_optLowAlt = optLowAlt.Value
can_optNormal = optNormal.Value

'Currently Hardwired Emergency
can_chkFire = chkFire.Value

optNoBank.Value = True
optNoPitch.Value = True
optNormal.Value = True
chkFire.Value = False
chkSmokeInside = False
chkOverheat = False
End Sub

Sub optGround_Click (Value As Integer)
optNoBank.Value = True
optNoPitch.Value = True
End Sub

'*********** Form Take Action
Option Explicit

Sub cmdCancel_Click ()
frmTakeAction.Hide
End Sub

Sub cmdOk_Click ()
Dim i As Integer

'Add Chosen Actions to StudActions
ReDim StudActions(IstChosen.ListCount - 1)
For i = 0 To IstChosen.ListCount - 1
    StudActions(i) = IstChosen.List(i)
Next i

frmTakeAction.Hide

'Tell the controller that the student is done
StudDoneFlag = True

End Sub

Sub cmdRemove_Click ()
'Removes the selected item from the Chosen List
If lstChosen.ListIndex <> -1 Then
  lstChosen.RemoveItem (lstChosen.ListIndex)
Else
  MsgBox "No item is selected in the Chosen Actions List"
End If

End Sub

Sub Form_Activate ()
lstChosen.Clear
End Sub

Sub Form_Load ()

'Load up the actions to choose from
lstActions.AddItem ("Throttle (Malfunctioning Engine) - Cutoff")
lstActions.AddItem ("Throttles - Idle")
lstActions.AddItem ("Throttles - Cutoff")
lstActions.AddItem ("Wheel Brakes - As Required")

End Sub

Sub lstActions_DblClick ()
Dim i As Integer
Dim sting As String

sting = lstActions.List(lstActions.ListIndex)

'check for doubles
For i = 0 To lstChosen.ListCount - 1
  If lstChosen.List(i) = sting Then
    Exit Sub
  End If
Next i

'Add it
lstChosen.AddItem (sting)

End Sub

"******* Attrib Module"
Option Explicit

'Declares the User-defined type Quest
Type Attrib
  Attribute As String 'The attribute which is the subject
  TheQuestion As String 'Text of the question from kb file
M4Asked As Integer 'Boolean whether M4 has asked
StudAsked As Integer 'Boolean whether student asked
AttribVal As Variant 'The value of the attribute
' Set in Control.bas GetValues

End Type

Global Attributes() As Attrib

Sub SetAttributeVal (AttribString As String, AttValue As Variant)
Dim i, found As Integer

found = False
For i = 0 To UBound(Attributes)
    If Attributes(i).Attribute = AttribString Then
        Attributes(i).AttribVal = AttValue
        found = True
        Exit For
    End If
Next i
If Not found Then
    MsgBox "Error! Cannot set attribute to value"
End If

End Sub

'******* Control Module
'Contains procedures that deal with all the interacting modules
'like GetValues

'Option Explicit
Option Compare Text
'Globals for instrument values
' (Used by GetValues, DialDisplay, AddDisplayQuery)
Global valAttitudeB As Integer
Global valAttitudeP As Integer
Global valVVI As Integer
Global valHeadingInd As Integer
Global ValAirspeed As Integer
Global ValAltitude As Integer
Global valFireLightRight As String
Global valFireLightLeft As String
Global valFuelFlow As String
Global valSmokeInside As String
Global valSmokeOutside As String
Global valRoughRunning As String
Global valExhaustTemp As String
Global valRPM As String
Global valStoppingDistance As String

Global FindLimitHack As OLimit

' Control Flags
' Set when student decides to maintain aircraft control
Global MaintainFlag As Integer
' Set to true when user wants interact directly with M4, otherwise false
Global M4IOFlag As Integer
' Set to true when M4 has finished diagnosing the scenario
Global M4DoneFlag As Integer
Global StudDoneFlag As Integer
Global Initialized As Integer

'These are for the rule trace and student model
Global RuleNumber As Integer

'This is to compare the actions of the student and M4
Global M4Diagnosis As String
Global StudDiagnosis As String
Global M4Actions() As String
Global StudActions() As String
Global CompActions() As Integer
Global Rules() As Rule
Global perfect, WrongOrder As Integer

Sub AppropriateQuestions (AttributeString As String)
Dim CurrentAttribute, NextCharacter As String
Dim i, M4AskedIt, StudAskedIt As Integer

Do Until AttributeString = "" Or AttributeString = 
    ' Get attribute words
    Do 'Get a Word
        ' Pick off the first character
        NextCharacter = Left$(AttributeString, 1)
        AttributeString = Right$(AttributeString, Len(AttributeString) - 1)
        CurrentAttribute = CurrentAttribute & NextCharacter
    Loop Until NextCharacter = " " Or AttributeString = ""
CurrentAttribute = Trim$(CurrentAttribute)

For i = 0 To UBound(Attributes)
    If CurrentAttribute = Attributes(i).Attribute Then
        M4AskedIt = Attributes(i).M4Asked
        StudAskedIt = Attributes(i).StudAsked
        Exit For
    End If
End If
Next i

If M4AskedIt And Not StudAskedIt Then
    'M4 Asked, student didn't
    MsgBox "Student didn't ask about the value of: " & CurrentAttribute
ElseIf M4AskedIt And StudAskedIt Then
    'Didn't apply the rule
    MsgBox "Didn't correctly apply rule"
End If
Loop
End Sub

Sub CompareActions()
Dim i, j, n, maxbound, minbound As Integer
Dim Sting As String
Dim NL As String
Dim MissingActions() As Integer
Dim SuspectRules() As Integer
Dim Found, NothingMissing As Integer
NL = Chr(10)
If Not UBound(M4Actions) = 0 Then
    ReDim Preserve M4Actions(UBound(M4Actions) - 1)
End If
If UBound(M4Actions) > UBound(StudActions) Then
    maxbound = UBound(M4Actions)
    minbound = UBound(StudActions)
    n = 1
Else
    maxbound = UBound(StudActions)
    minbound = UBound(M4Actions)
    n = 2
End If

'Set the Compare Array equal to the size of the bigger
ReDim CompActions(maxbound + 1)

'Set its first element to indicate which was bigger
CompActions(0) = i

'Initialize the rest of the elements
For i = 1 To maxbound + 1
    CompActions(i) = -1
Next i

'Compare for same actions, same order
If n = 1 Then
    For i = 0 To maxbound
For j = 0 To UBound(StudActions)
    If M4Actions(i) Like StudActions(j) Then
        CompActions(i + 1) = j
        Exit For
    End If
Next j
Next i
Else
    For i = 0 To maxbound
        For j = 0 To UBound(M4Actions)
            If StudActions(i) Like M4Actions(j) Then
                CompActions(i + 1) = j
                Exit For
            End If
        Next j
    Next i
End If

'Print results of comparison in a message box.
'For i = 0 To minbound
'    n = M4Actions(i) Like StudActions(i)
'    MsgBox (M4Actions(i) & NL & StudActions(i) & NL & n)
'Next i
'For i = 0 To maxbound + 1
'    MsgBox (i & ": " & Str$(CompActions(i)))
'Next i

'Check out their ordering
perfect = True
WrongOrder = True
'See if they're good
For i = 1 To UBound(CompActions)
    perfect = perfect And (CompActions(i) = i - 1)
    If CompActions(i) < 0 Then
        WrongOrder = False
    End If
Next i
Next i

'Credit Student with all rules M4 used as learned
If perfect Or WrongOrder Then
    For i = 0 To UBound(Rules)
        If Rules(i).M4ActRule = True Then
            Rules(i).StudKnows = 1
        End If
    Next i
End If

'Try to figure out what they did wrong
'First, see which actions are missing
ReDim MissingActions(0)
NothingMissing = True
'Only do this if there is something either missing or added
If Not perfect And Not WrongOrder Then
    'For each action, see if it's in the student's actions
    For i = 0 To UBound(M4Actions)
        Found = False
        For j = 0 To UBound(StudActions)
            If M4Actions(i) = StudActions(j) Then
                Found = True
                Exit For
            End If
        Next j
        'If it's not, then add its number to the missing actions list
        If Found = False Then
            NothingMissing = False
            MissingActions(UBound(MissingActions)) = i
            ReDim Preserve MissingActions(UBound(MissingActions) + 1)
        End If
    Next i
    'If anything is in Missing Actions, it's oversized by one
    If Not NothingMissing Then
        ReDim Preserve MissingActions(UBound(MissingActions) - 1)
    End If
End If

'Look for the relevant rules to the missing ones
If Not NothingMissing Then
    ReDim SuspectRules(0)
    For i = 0 To UBound(MissingActions)
        'Get the missing action text
        Sting = M4Actions(MissingActions(i))
        'Check the rules used for getting actions
        For j = 0 To UBound(Rules)
            If Rules(j).M4ActRule Then
                If InStr(Rules(j).Conclusion, Sting) <> 0 Then
                    SuspectRules(UBound(SuspectRules)) = j
                    ReDim Preserve SuspectRules(UBound(SuspectRules) + 1)
                End If 'Rule contains the action
            End If 'M4 used the rule to get an action
        Next j
    Next i
    If Not UBound(SuspectRules) = 0 Then
        ReDim Preserve SuspectRules(UBound(SuspectRules) - 1)
    End If
'See if the student asked the relevant questions

'Mark all non-suspect action rules as known, suspect as unknown
For i = 0 To UBound(Rules)
  If Rules(i).M4ActRule Then
    Found = False
    For j = 0 To UBound(SuspectRules)
      If SuspectRules(j) = i Then
        Found = True
        Exit For
      End If
    Next j
    If Not Found Then
      'Give credit
      Rules(i).StudKnows = 1
    Else
      AppropriateQuestions (Rules(i).PremAttributes)
      Rules(i).StudKnows = 2
      MsgBox "Student Doesn't Know Rule: " & Str(i)
    End If
  End If
Next i

End If 'Not Nothing Missing

"display missing
'Sting = ""
'For i = 0 To UBound(MissingActions)
  ' Sting = Sting &" " & Str(MissingActions(i))
'Next i
'MsgBox "Missing: " & Sting

'If nothing is missing and it's not perfect, or in the wrong order
'Check for extras

End Sub

Sub CompareDiagnoses ()
'Decide if the student's diagnosis agrees with M.4's

'First make sure we have M.4's
Do While M4Diagnosis ="
  DoEvents
Loop

If M4Diagnosis = StudDiagnosis Then
  MsgBox "Way to Go!"
'Give credit for rules used
For i = 0 To UBound(Rules)
    If Rules(i).M4DiagRule = True Then
        Rules(i).StudKnows = 1
    End If
Next i
End If
frmTakeAction.Show 1
End Sub

Sub Controller()
' This routine controls the interaction between the student,
'the program, and the knowledge base

' Field questions from M4 until it's solved it
' and the student has also solved the problem
Do While (Not M4DoneFlag) Or (Not StudDoneFlag)
    DoEvents
Loop

' Compare the actions M4 and the Student come up with
' for order and stuff
CompareActions
End Sub

Sub GetValues()
Dim TmpLimit As OLimit
Dim n, i As Integer

' For each thing, find it with right range in the olimit.dat file
' Then, using upper and lower limits, pick something reasonable
' If there is no range, just use the value
Randomize

'Set Attitude Bank
If frmRange.optNoBank Then
    ' Wings are level, check on pitch
    valAttitudeB = 0
Else ' Wings aren't level
    If frmRange.optLeft Then
        ' In Left Turn
        FindLimit("Attitude Indicator (Bank) - Left Turn")
        TmpLimit = FindLimitHack
        valAttitudeB = MyRand(TmpLimit.Lower, TmpLimit.Upper, 5)
    Else
        ' Right Turn
    End If
Else
    ' Right Turn
End If
FindLimit ("Attitude Indicator (Bank) - Right Turn")
   TmpLimit = FindLimitHack
   valAttitudeB = MyRand(TmpLimit.Lower, TmpLimit.Upper, 5)
End If 'Left Turn
End If 'Aren't Level

'Set Attitude Pitch and VVI
If frmRange.optNoPitch Then
   'Maintaining Altitude
   valAttitudeP = 0
   valVVI = 0
Else 'Not Maintaining Altitude
   If frmRange.optClimb Then
      'Climb
      FindLimit ("Attitude Indicator (Pitch) - Climb")
      TmpLimit = FindLimitHack
      valAttitudeP = MyRand(TmpLimit.Lower, TmpLimit.Upper, 5)
      FindLimit ("Vertical Velocity Indicator - Climb")
      TmpLimit = FindLimitHack
      valVVI = MyRand(TmpLimit.Lower, TmpLimit.Upper, 5)
   Else
      'Descend
      FindLimit ("Attitude Indicator (Pitch) - Dive")
      TmpLimit = FindLimitHack
      valAttitudeP = MyRand(TmpLimit.Lower, TmpLimit.Upper, 5)
      FindLimit ("Vertical Velocity Indicator - Dive")
      TmpLimit = FindLimitHack
      valVVI = MyRand(TmpLimit.Lower, TmpLimit.Upper, 5)
   End If 'Climb
End If 'Pitch

'Set Heading Indicator
FindLimit ("Heading Indicator - Normal")
   TmpLimit = FindLimitHack
   valHeadingInd = Int((TmpLimit.Upper - TmpLimit.Lower + 1) * Rnd + TmpLimit.Lower)

'Set Airspeed and Altitude
If frmRange.optGround Then
   'On the Ground - Speed and Height should be zero
   ValAirspeed = 0
   ValAltitude = 0
ElseIf frmRange.optLowAlt Then
   FindLimit ("Airspeed Indicator - Low Altitude")
   TmpLimit = FindLimitHack
   ValAirspeed = Int((TmpLimit.Upper - TmpLimit.Lower + 1) * Rnd + TmpLimit.Lower)
   FindLimit ("Altimeter - Low Altitude")
   TmpLimit = FindLimitHack
   ValAltitude = Int((TmpLimit.Upper - TmpLimit.Lower + 1) * Rnd + TmpLimit.Lower)
'If during Take Off Roll
ElseIf frmRange.optTORoll Then
    FindLimit ("Airspeed Indicator - T/O Roll")
    TmpLimit = FindLimitHack
    ValAirspeed = Int((TmpLimit.Upper - TmpLimit.Lower + 1) * Rnd + TmpLimit.Lower)
    FindLimit ("Altimeter - T/O Roll")
    TmpLimit = FindLimitHack
    ValAltitude = Int((TmpLimit.Upper - TmpLimit.Lower + 1) * Rnd + TmpLimit.Lower)
Else 'Must be Normal
    FindLimit ("Airspeed Indicator - Normal")
    TmpLimit = FindLimitHack
    ValAirspeed = Int((TmpLimit.Upper - TmpLimit.Lower + 1) * Rnd + TmpLimit.Lower)
    FindLimit ("Altimeter - Normal")
    TmpLimit = FindLimitHack
    ValAltitude = Int((TmpLimit.Upper - TmpLimit.Lower + 1) * Rnd + TmpLimit.Lower)
End If 'On ground, at Low Alt or T/O or Normal
setAttributeVal "altitude", ValAltitude
setAttributeVal "airspeed", ValAirspeed

'EMERGENCY SETUP

ENGINE FIRE

' First, assume there's no fire
DialDisplay.picFireLight.Picture = LoadPicture("\Okemo\saeor\images\firelt.bmp")
DialDisplay.picFireLight2.Picture = LoadPicture("\Okemo\saeor\images\firelt.bmp")
valFireLightRight = "no"
valFireLightLeft = "no"

'Then check to see if that's a bad assumption
If frmRange.chkFire.Value Then
    'We have a fire, so pick an engine
    n = Rnd
    If n < .5 Then
        valFireLightLeft = "yes"
        DialDisplay.picFireLight.Picture = LoadPicture("\Okemo\saeor\images\firelton.bmp")
    Else
        valFireLightRight = "yes"
        DialDisplay.picFireLight2.Picture = LoadPicture("\Okemo\saeor\images\firelton.bmp")
    End If 'Picking Engine
End If 'Whether or not there's a fire

OVERHEAT
'First set the default values, assuming no overheat
valFuelFlow = "normal"
valSmokeInside = "no"
valSmokeOutside = "no"
valRoughRunning = "no"
valExhaustTemp = "normal"
valRPM = "normal"

'Check to see if there's an overheat
If frmRange.chkOverheat.Value Then
  'We have an overheat, so set the symptoms

  'Go through twice to get a good batch of symptoms
  For i = 1 To 2
    n = Rnd
    If n <= .25 Then
      valFuelFlow = "fluctuating"
      valExhaustTemp = "excessive"
    ElseIf n > .25 And n <= .5 Then
      valSmokeInside = "yes"
    ElseIf n > .5 And n <= .75 Then
      valSmokeOutside = "yes"
    Else
      valRoughRunning = "yes"
    End If
  Next i
End If 'If overheat

' SMOKE
If frmRange.chkSmokeInside Then
  valSmokeInside = "yes"
End If

' STOPPING DISTANCE
' Arbitrarily decide between minimal and sufficient
n = Rnd
If n <= .5 Then
  valStoppingDistance = "minimal"
Else
  valStoppingDistance = "sufficient"
End If

'Set all the new values in the Attributes
setAttributeVal "stopping-distance", valStoppingDistance
setAttributeVal "smoke-inside", valSmokeInside
setAttributeVal "smoke-outside", valSmokeOutside
setAttributeVal "fuel-flow", valFuelFlow
setAttributeVal "exhaust-temp", valExhaustTemp
setAttributeVal "rough-running", valRoughRunning
setAttributeVal "rpm", valRPM
If valFireLightRight = "yes" Then
  setAttributeVal "fire-light", "right"
ElseIf valFireLightLeft = "yes" Then
SetAttributeVal "fire-light", "left"
Else
    SetAttributeVal "fire-light", "no"
End If

End Sub

Sub InitializeScenario()
    'Initialize everything to begin a new scenario

    ReDim M4Actions(0)
    ReDim StudActions(0)

    'First set some flags
    'Pass control to Controlling Tutoring Module
    'First setting the maintain, M4IO flag to false
    MaintainFlag = False
    M4IOflag = False
    M4DoneFlag = False
    StudDoneFlag = False
    M4Diagnosis = ""

    'Initialize Rules
    For i = 0 To UBound(Rules)
        Rules(i).M4DiagRule = False
        Rules(i).M4ActRule = False
    Next i

    'At this point, everything has been
    'loaded up and is ready to go.

    'Show the Maintaining dialog as a modal form
    'frmMaintain.Show

    'Later I should have a check that they chose to maintain
    'as the first action

    'Wait until the student is ready to maintain aircraft control
    Do Until MaintainFlag
        DoEvents
        Loop
    MaintainFlag = False

    'Later I should have a check that they adjusted correctly
    'This should be before the analyze but not immediately
    'after the maintain to give the student time to notice and
    'correct the error on their own
'Now pass the parameters of the situation over to M.4
'this is accomplished by setting the relevant expressions
'in the cache

'Since the student ensured that the aircraft is flying
'Straight and Level, and the Heading is irrelevant,
it is only necessary to pass the airspeed and altitude
'and any clues to the situation being given to the student

'Reset the cache
M4cmd ("reset")
'Pass over the altitude
M4cmd ("set altitude = " & Str$(ValAltitude))
'Pass over the airspeed
M4cmd ("set airspeed = " & Str$(ValAirspeed))

'Status of the fire-lights
'Despite whatever else I choose as a teaching method,
The student will immediately have the status of the
'firelights
If valFireLightRight = "yes" Then
  M4cmd("set fire-light = right")
ElseIf valFireLightLeft = "yes" Then
  M4cmd("set fire-light = left")
Else
  M4cmd("set fire-light = no")
End If

'Here I should make some sort of decision about how much
'information to reveal to the student and M.4
'Tell M.4 to get working on it
M4.Show
'Won't work yet since M4 wants to set focuses
'M4.Visible = False

'Set M.4 to trace all
TraceAll = True
'M4.MnuOptTrAll.Checked = TraceAll
'If M4.MnuOptTrAll.Checked Then
  ' TrString = "trace on"
'Else
  ' TrString = "trace off"
  ' TraceExp = False
'End If
M4cmd ("trace on")

'Actually start M4 on the problem
ProcessStart "restart"
DialDisplay.SetFocus

Controller
End Sub

Sub InitializeSession ()
'Initialize everything when first opened

'Open the files and set for reading/appending
'Open OLimit.dat, set record length, current record
Initialize_DataFile

'Also start the History File
StartHistory

'Read in the rules and questions
'Load in the rules
ParseKBFile

'Get M4 up and running
Load M4
'Then give M4 the command to load the knowledge base
'Define a newline
M4cmd ("load " & Chr(34) & \"Okemo\saeo\convert\test.kb\" & Chr(34))

'Initialize Rules
For i = 0 To UBound(Rules)
   Rules(i).M4DiagRule = False
   Rules(i).M4ActRule = False
   Rules(i).StudKnows = False
Next i

End Sub

********** History Module
'Contains the procedures that deal with the history kept of the
'Student's actions - a file History.Txt

Option Explicit

Global HistFile As Integer
Global ExFile As Integer

Sub AddAction (Action As String)
'Appends Report to the history File
Dim Sting As String
'Readying Report
Sting = "User decided to: " & Action

'Print Report to History
Print #HistFile, Sting

End Sub

Sub AddDisplayQuery (Source As String, Report As String)
'Appends Report to the history File
Dim Sting As String

'Readying Report
Sting = "User queried " & Source
Sting = Sting & ". Value was: "
Sting = Sting & Report

'Print Report to History
Print #HistFile, Sting

End Sub

Sub AddM4Question (question As String)
Dim Sting, Sting2 As String
Dim i As Integer
'Takes a question that M4 asks and appends it to the
'ExFile

'Mark question as asked by M4
For i = 0 To UBound(Attributes)
    If InStr(question, Attributes(i).TheQuestion) <> 0 Then
        Attributes(i).M4Asked = True
        Exit For
    End If
Next i

'Print Report to History
Print #ExFile, question
End Sub

Sub AddQuery (TheQuery As String)
Dim Sting As String
Dim i As Integer

Sting = "User asked: " + TheQuery
'Print Report to History
'Mark question as asked by M4
For i = 0 To UBound(Attributes)
    If InStr(TheQuery, Attributes(i).TheQuestion) <> 0 Then
        Attributes(i).StudAsked = True
        Exit For
    End If
Next i

Print #HistFile, String

End Sub

Sub StartHistory()
' Opens History File

On Error Resume Next
Kill "\Okemo\saero\convert\history.txt"
HistFile = FreeFile
Open "\Okemo\saero\convert\history.txt" For Append As HistFile

Kill "\Okemo\saero\convert\history2.txt"
ExFile = FreeFile
Open "\Okemo\saero\convert\history2.txt" For Append As ExFile

' File Closed in Exit from DialDisplay

End Sub

'****** Module M4 Interaction
Option Explicit

Sub AnsM4Question (question As String)
Dim i As Integer

For i = 0 To UBound(Attributes)
    If InStr(question, Attributes(i).TheQuestion) <> 0 Then
        If IsNumeric(Attributes(i).AttribVal) Then
            M4cmd (Str$(Attributes(i).AttribVal))
        Else
            M4cmd (Attributes(i).AttribVal)
        End If
    End If
Next i

' M4 is inquiring about the altitude
' If InStr(txtM4IO.Text, "altitude") <> 0 Then
M4cmd (Str$(valAltitude))

'M4 is inquiring about the airspeed
ElseIf InStr(txtM4IOText, "airspeed") <> 0 Then
    M4cmd (Str$(valAirspeed))

'M4 is inquiring about the fuel flow
ElseIf InStr(txtM4IOText, "fuel flow") <> 0 Then
    M4cmd (valFuelFlow)

'M4 is inquiring about the smoke in the cockpit
ElseIf InStr(txtM4IOText, "smoke in the cockpit") <> 0 Then
    M4cmd (valSmokeInside)

'M4 is inquiring about the smoke outside the aircraft
ElseIf InStr(txtM4IOText, "smoke visible outside") <> 0 Then
    M4cmd (valSmokeOutside)

'M4 is inquiring if the engine is running rough
ElseIf InStr(txtM4IOText, "running rough") <> 0 Then
    M4cmd (valRoughRunning)

'M4 is inquiring about the exhaust temperature
ElseIf InStr(txtM4IOText, "exhaust temperature") <> 0 Then
    M4cmd (valExhaustTemp)

'M4 is inquiring about the engine's RPM
ElseIf InStr(txtM4IOText, "RPM") <> 0 Then
    M4cmd (valRPM)

'M4 is inquiring about the stopping distance
ElseIf InStr(txtM4IOText, "stopping distance") <> 0 Then
    M4cmd (valStoppingDistance)

End If

End Sub

Sub ImpM4Conclusion (question As String)
Dim Sting As String

'Figures out if M4 is reporting a conclusion,
either an action or diagnosis
'Sets the M4Diagnosis, assumes M4 only has one
If InStr(question, "diagnosis") <> 0 Then
    M4Diagnosis = Right$(question, InStr(question, ")") + 1)
    M4Diagnosis = Left$(M4Diagnosis, InStr(M4Diagnosis, "(")) - 1
    M4Diagnosis = Trim$(M4Diagnosis)
    MsgBox M4Diagnosis
End If 'Diagnosis

'Grabs M4's actions
If InStr(question, "action") <> 0 Then
    Sting = Right$(question, Len(question) - InStr(question, ")") - 1)
    Sting = Left$(Sting, InStr(Sting, "%") - 2)
    If Right$(Sting, 1) = "" Then
        Sting = Left$(Sting, Len(Sting) - 1)
    Else

105
Sub RevEngRuleTrace (ReasonString As String)
Dim Sting As String
Dim n As Integer

'Find out if ReasonString tells of a succeeding rule
If so log it in the rule trace

n = InStr(ReasonString, "rule")
If n <> 0 And InStr(ReasonString, "succeeded.") Then
    Sting = Right$(ReasonString, (Len(ReasonString) - 5))
    RuleNumber = Val(Sting)
    'MsgBox (Str$(RuleNumber))

    'If still working on diagnosis, mark rule as diag
    If M4Diagnosis = "" Then
        Rules(RuleNumber).M4DiagRule = True
    Else
        Rules(RuleNumber).M4ActRule = True
    End If
End If
End Sub

'****** Module Olimit
'Contains the procedures that interact with the Olimit data file

Option Explicit

'User Defined Type
Type OLimit
    Name As String * .50
    Range As String * 20
    Units As String * 40
    Upper As Integer
    Lower As Integer
End Type
End Type

' Globals for I/O
Global Limit As OLimit
Global FileNum As Integer
Global RecordLen As Long
Global CurrentRecord As Long
Global LastRecord As Long

Sub FindLimit (NameToFind As String)
' Finds the requested olimit in the data file without stomping
' on current record

Dim TmpRecNum As Long
Dim TmpLimit As OLimit
Dim ing As String
Dim Found As Integer

' Otherwise go for it
NameToFind = UCase(NameToFind)
Found = False

'Search
For TmpRecNum = 1 To LastRecord
    Get #FileNum, TmpRecNum, TmpLimit
    ing = UCase(Trim(TmpLimit.Name)) + " - " + UCase(Trim(TmpLimit.Range))
    If NameToFind = ing Then
        Found = True
        Exit For
    End If
Next

If Found Then
    FindLimitHack = TmpLimit
Else
    MsgBox "Name " + NameToFind + " not found!"
End If
End Sub

Sub Initialize_DataFile ()

' Open the Olimit Data file now
' Calculate Length of Record
RecordLen = Len(Limit)

' Open data file
FileNum = FreeFile
Open "\Okemo\saeor\convert\olimit.dat" For Random As FileNum Len = RecordLen

107
'Set Record numbers
CurrentRecord = 1
LastRecord = FileLen(\Okemo\saor\convert\olimit.dat) / RecordLen
If LastRecord = 0 Then
    LastRecord = 1
End If
End Sub

Static Function MyRand (Lower As Integer, Upper As Integer, StepSize As Integer)
'Produce Random Numbers in a half a normal distribution
'Big half is the half closer to zero
Dim counter, numb, midpoint As Integer

'First accumulate (Upper - Lower) Randomly generated numbers
' in the range of 0 to 1 and add them to get a number between
' 0 and (Upper - Lower), then add Lower to get the range between
' Lower and Upper
Randomize 'Initializes Rand based on system timer
numb = 0 'Initializes Number
midpoint = Int((Upper - Lower) / 2 + Lower)

'Adds the randoms
For counter = Lower To Upper Step StepSize
    numb = numb + Rnd * StepSize
Next counter
numb = numb + Lower

'Then convert normal dist into half
If (numb < midpoint And midpoint >= 0) Or (numb > midpoint And midpoint < 0) Then
    numb = midpoint - numb
Else
    numb = numb - midpoint
End If
MyRand = Int(numb * 2)
End Function

***** Module Rule
Option Explicit

Type Rule
    Premise As String 'The Full premissis string
    PremAttributes As String 'The attributes sep by spaces
    Conclusion As String 'The Full conclusion string
    ConcAttribute As String 'The attribute set in the conclusion
M4DiagRule As Integer 'M4 Used rule for diagnosis
M4ActRule As Integer 'M4 used rule for actions
StudKnows As Integer ' 0 - default - unknown
   ' 1 - Student knows
   ' 2 - student doesn't know

End Type

Sub ParseKBFile()
  Dim KBFileNum As Integer
  Dim NextLine, Sting, Sting2 As String
  Dim InRule, CurrentRule, i As Integer
  ' Dim Rules() As String

  ' Read and parse the knowledge base
  KBFileNum = FreeFile
  Open "\Okemo\saeor\convert\test.kb" For Input As KBFileNum

  ReDim Rules(0)
  ReDim Attributes(0)
  InRule = False
  Do Until EOF(KBFileNum)
    Line Input #KBFileNum, NextLine
    If Left(NextLine, 4) = "rule" Then
      InRule = True
      CurrentRule = Val(Right(NextLine, Len(NextLine) - 5))
      ' ReDim Preserve Rules(CurrentRule)
      Sting2 = Trim$(NextLine)
    ElseIf InRule Then
      Sting2 = Sting2 & " " & Trim$(NextLine)
      If Right(NextLine, 1) = "." Then
        InRule = False
        ' MsgBox Sting2
        ParseRule (Sting2)
      End If
    ElseIf Left(NextLine, 8) = "question" Then
      Sting = Right$(NextLine, Len(NextLine) - 9)
      Sting = Left$(Sting, InStr(Sting, ") =") - 1)
      Attributes(UBound(Attributes)).Attribute = Sting
      Sting = Right$(NextLine, Len(NextLine) - InStr(NextLine, """"))
      Sting = Left$(Sting, Len(Sting) - 2)
      Attributes(UBound(Attributes)).TheQuestion = Sting
      ReDim Preserve Attributes(UBound(Attributes) + 1)
    End If
  Loop
  ReDim Preserve Attributes(UBound(Attributes) - 1)

  ' For i = 1 To UBound(Attributes)
  '   MsgBox "Question: " & Attributes(i).Attribute & ", " & Attributes(i).TheQuestion
'Next i

Close KBFileNum
End Sub

Sub ParseRule (RuleString As String)
Dim InPremise, GotAttribute, InConclusion As Integer
Dim CurrentWord, NextCharacter, NL As String
Dim StringLength, RuleNumber, i As Integer

NL = Chr(13) + Chr(10)

InPremise = False
GotAttribute = False
InConclusion = False

'Read until the period signals the end
Do 'Parse Rule
   CurrentWord = ""
   Do 'Get a Word
      'Pick off the first character
      NextCharacter = Left$(RuleString, 1)
      RuleString = Right$(RuleString, Len(RuleString) - 1)
      CurrentWord = CurrentWord & NextCharacter
      If NextCharacter = "." Then
         Exit Do
      End If
   Loop Until NextCharacter = ""
   CurrentWord = Trim$(CurrentWord)
   If Not InPremise And Not InConclusion Then
      If CurrentWord = "if" Then
         InPremise = True
         Rules(RuleNumber).PremAttributes = ""
         Rules(RuleNumber).Premise = CurrentWord
         Rules(RuleNumber).Conclusion = ""
      ElseIf InStr(CurrentWord, "rule") Then
         'grab the rule number
         RuleNumber = Val(Right(CurrentWord, Len(CurrentWord) - 5))
         'see if array is big enough
         If UBound(Rules) < RuleNumber Then
            ReDim Preserve Rules(RuleNumber)
         End If
      End If
   ElseIf InPremise Then
      If CurrentWord = "then" Then
         InPremise = False
         GotAttribute = False
         InConclusion = True
      End If
   End If

End Sub
ElseIf CurrentWord = "not" Then
    i = 1
ElseIf GotAttribute = True Then
    If CurrentWord = "and" Or CurrentWord = "or" Then
        GotAttribute = False
    End If
ElseIf GotAttribute = False Then
    Rules(RuleNumber).PremAttributes = Rules(RuleNumber).PremAttributes & " " & CurrentWord
    GotAttribute = True
End If
Rules(RuleNumber).Premise = Rules(RuleNumber).Premise & " " & CurrentWord

ElseIf InConclusion Then
    If GotAttribute = False Then
        Rules(RuleNumber).ConcAttribute = CurrentWord
        GotAttribute = True
    End If
    Rules(RuleNumber).Conclusion = Rules(RuleNumber).Conclusion & " " & CurrentWord
End If 'Conclusion
Loop Until NextCharacter = "."

' Sting = "Rule " & Str(RuleNumber) & NL
' Sting = Sting & "Premise: " & Rules(RuleNumber).Premise & NL
' Sting = Sting & "Premise Attributes: " & Rules(RuleNumber).PremAttributes & NL
' Sting = Sting & "Conclusion: " & Rules(RuleNumber).Conclusion & NL
' Sting = Sting & "Conclusion Attribute: " & Rules(RuleNumber).ConcAttribute
' MsgBox Sting

End Sub
8. APPENDIX B - STAND-UP KNOWLEDGE BASE

goal = diagnosis.

goal = action.

goal = considerations.

rule-1:
  if altitude = 0 and
  airspeed = 0
  then general-where = on-ground.

rule-2:
  if altitude = 0 and
  gairspeed = slow
  then general-where = taking-off.

rule-3:
  if galtitude = low and
  gairspeed = slow and
  gVVI = climbing and
  gpitch = up
  then general-where = taking-off.

rule-4:
  if altitude = M and
  M<1000
  then galtitude = low.

rule-5:
  if airspeed = N and
  N<120 and
  N>0
  then gairspeed = slow.

rule-6:
  if pitch = 0
  then gpitch = level.

rule-7:
  if pitch = P and
  P>0
then g\text{pitch} = \text{up}.

\text{rule-8:}
\begin{align*}
\text{if } \text{pitch} &= \text{P and} \\
&\quad \text{P}<0 \\
\text{then } g\text{pitch} &= \text{down}.
\end{align*}

\text{rule-9:}
\begin{align*}
\text{if } v\text{vi} &= 0 \\
\text{then } g\text{VVI} &= \text{steady}.
\end{align*}

\text{rule-10:}
\begin{align*}
\text{if } v\text{vi} &= \text{V and} \\
&\quad \text{V}>0 \\
\text{then } g\text{VVI} &= \text{climbing}.
\end{align*}

\text{rule-11:}
\begin{align*}
\text{if } v\text{vi} &= \text{V and} \\
&\quad \text{V}<0 \\
\text{then } g\text{VVI} &= \text{descending}.
\end{align*}

\text{rule-12:}
\begin{align*}
\text{if } \text{galtitude} &= \text{low and} \\
&\quad \text{not general-where} = \text{taking-off} \\
\text{then general-where} &= \text{low-alt}.
\end{align*}

\text{rule-13:}
\begin{align*}
\text{if not } \text{galtitude} &= \text{low} \\
\text{then general-where} &= \text{at-alt}.
\end{align*}

\text{rule-14:}
\begin{align*}
\text{if } \text{gairspeed} &= \text{slow and} \\
&\quad \text{g\text{pitch} = down and} \\
&\qquad \text{galtitude} = \text{low} \\
\text{then general-where} &= \text{landing}.
\end{align*}

\text{rule-15:}
\begin{align*}
\text{if } \text{gairspeed} &= \text{slow and} \\
&\quad g\text{VVI} = \text{decreasing and} \\
&\qquad \text{galtitude} = \text{low} \\
\text{then general-where} &= \text{landing}.
\end{align*}

\text{rule-16:}
\begin{align*}
\text{if general-where} &= \text{taking-off and} \\
&\quad \text{stopping-distance} = \text{sufficient}
\end{align*}
then could-abort = yes.

rule-17:
    if altitude = M and
        M>50
    then could-abort = no.

rule-18:
    if altitude = M and
        M<2000
    then eject-advisable = no.

rule-19:
    if gairspeed = slow
    then eject-advisable = no.

rule-20:
    if altitude = M and
        M<3000
    then airstart-advisable = no.

rule-21:
    if landing-looks = long or
        (landing-looks = short or
            landing-looks = high)
    then landing-looks = bad.

rule-22:
    if general-where = landing and
        landing-looks = bad
    then emergency = go-around.

multivalued(emergency).

rule-23:
    if fire-light = right or
        fire-light = left
    then emergency = engine-fire.

rule-24:
    if emergency = engine-fire
    then fire-danger = present.

rule-25:
    if fuel-fumes = present
then fire-danger = present.

rule-26:
  if fire-danger = present
  then airstart-advisable = no.

rule-27:
  if fire-danger = present
  then canopy = retain.

rule-28:
  if fire-danger = present and
    X-is-electric = true
  then dont-use-X = true.

rule-29:
  if fire-light = X
  then X-engine = failed.

rule-30:
  if X-engine-windmilling = yes
  then X-engine = failed.

rule-31:
  if X-engine-frozen = yes
  then X-engine = failed.

rule-32:
  if X-engine-flameout = yes
  then X-engine = failed cf 99.

rule-33:
  if right-engine = failed and
    left-engine = failed
  then emergency = two-engine-failure cf 99.

rule-34:
  if left-engine = failed and
    not right-engine = failed
  then emergency = one-engine-failure cf 99.

rule-35:
  if right-engine = failed and
    not left-engine = failed
  then emergency = one-engine-failure cf 99.
rule-36:
if fuel-flow = 100
then fuel = cutoff.

rule-37:
if fuel-flow = fluctuating and
fluct-rpm = fluctuating and
exhaust-temp = fluctuating
then emergency = erratic-fuel-flow cf 80.

rule-38:
if fuel-flow = fluctuating and
exhaust-temp = excessive
then emergency = overheat cf 80.

rule-39:
if smoke-inside = yes or
smoke-outside = yes
then emergency = overheat cf 60.

rule-40:
if smoke-inside = yes
then emergency = smoke.

rule-41:
if rough-running = yes
then emergency = overheat cf 60.

multivalued(general-action).

rule-42:
if general-where = taking-off and
emergency-status = have-one
then general-action = abort.

rule-43:
if general-where = on-ground and
emergency-status = dealt-with
then general-action = ground-egress.

rule-44:
if general-where = at-alt and
emergency-status = dealt-with
then general-action = forced-landing.
rule-45:
   if emergency = E
   then emergency-status = have-one.

rule-46:
   if emergency = one-engine-failure and
       general-where = at-alt
   then emergency = engine-failure-in-flight.

rule-47:
   if emergency = engine-failure-in-flight and
       not airstart-advisable = no
   then diagnosis = emergency-airstart.

rule-48:
   if general-action = abort
   then diagnosis = abort.

rule-50:
   if (emergency = overheat or
       emergency = engine-fire) and
       general-where = at-alt
   then diagnosis = engine-fire-in-flight.

rule-51:
   if emergency = one-engine-failure and
       (general-where = taking-off or
        general-where = low-alt) and
       could-abort = no
   then diagnosis = one-engine-failure-low-alt.

rule-52:
   if (general-where = low-alt or
       general-where = taking-off) and
       could-abort = no and
       eject-advisable = no and
       airstart-advisable = no and
       emergency = two-engine-failure
   then diagnosis = two-engine-failure-low-alt.

rule-53:
   if emergency = two-engine-failure and
       not eject-advisable = no
   then diagnosis = eject.
rule-54:
    if emergency = one-engine-fire and
        emergency = go-around
    then diagnosis = single-engine-go-around.

rule-55:
    if diagnosis = emergency-airstart
    then action = 'Starter - Air.' cf 99.

rule-56:
    if diagnosis = emergency-airstart
    then action = 'Fuel System - Emergency.' cf 99.

rule-57:
    if rpm = M and
        M<16
    then low-rpm = true.

rule-58:
    if diagnosis = emergency-airstart and
        low-rpm = true
    then action =
        'Starter and Ignition - GND - ON and Hold until 30% RPM.' cf 99.

rule-59:
    if diagnosis = engine-fire-in-flight
    then action = 'Throttle - Retard.' cf 99.

rule-60:
    if diagnosis = engine-fire-in-flight
    then action = 'Fuel Shutoff T-Handle - Pull-Off.' cf 99.

rule-61:
    if action = 'Fuel Shutoff T-Handle - Pull-Off.' and
        not fuel = cutoff
    then considerations = 'Fire still being fed.' cf 99.

rule-62:
    if action = 'Fuel Shutoff T-Handle - Pull-Off.' and
        not fuel = cutoff
    then considerations = 'Check fuel shutoff circuit breaker.' cf 99.

rule-63:
    if diagnosis = engine-fire-in-flight
then action = 'Throttle - Cutoff.' cf 99.

rule-64:
if diagnosis = abort and
(emergency = engine-fire or
emergency = overheat) and
not stopping-distance = minimal
then action = 'Throttle (malfunctioning engine) - cutoff' cf 99.

rule-65:
if diagnosis = abort and
stopping-distance = minimal
then action = 'Throttles - cutoff' cf 99.

rule-66:
if diagnosis = abort and
not stopping-distance = minimal and
not (emergency = engine-fire or
emergency = overheat)
then action = 'Throttles - idle' cf 99.

rule-67:
if diagnosis = abort
then action = 'Wheel Brakes - As Required' cf 99.

rule-68:
if diagnosis = two-engine-failure-low-alt
then action = 'Glide - 100 KIAS Minimum.' cf 99.

rule-69:
if diagnosis = two-engine-failure-low-alt
then action = 'Gear - Down.' cf 99.

rule-70:
if diagnosis = two-engine-failure-low-alt
then action = 'Throttles - Cut-Off.' cf 99.

rule-71:
if action = 'Gear - Down.' and
gear = indicates-up
then considerations =
'Consider Emergency Landing Gear Extension system.' cf 99.

multivalued(considerations).
rule-72:
  if diagnosis = two-engine-failure-low-alt
  then considerations =
    'Canopy should be retained to afford protection explosion.'cf 99.

rule-73:
  if diagnosis = two-engine-failure-low-alt and
  gear = indicates-up
  then considerations =
    'Do not sacrifice control while extending gear with emergency system.'cf 99.

rule-74:
  if diagnosis = one-engine-failure-low-alt
  then action = 'Flaps - 50%.'cf 99.

rule-75:
  if diagnosis = one-engine-failure-low-alt
  then action = 'Gear - Up.'cf 99.

rule-76:
  if diagnosis = one-engine-failure-low-alt
  then action = 'Flaps - Up (100 KIAS Minimum).'cf 99.

rule-77:
  if diagnosis = eject
  then action = 'Handgrips - Raise.'cf 99.

rule-78:
  if diagnosis = eject
  then action = 'Triggers - Squeeze.'cf 99.

rule-79:
  if diagnosis = eject and
  action = 'Handgrips - Raise.'
  then considerations = 'Sit erect, Head against headrest, Feet back.'cf 99.

rule-80:
  if diagnosis = eject and
  action = 'Triggers - Squeeze.'
  then considerations =
    'Squeeze simulateously. Keep area between handgrip, trigger clear.'cf 99.

rule-81:
if diagnosis = single-engine-go-around
then action = 'Throttle - Military.' cf 99.

rule-82:
if diagnosis = single-engine-go-around
then action = 'Speedbrake - In.' cf 99.

rule-83:
if diagnosis = single-engine-go-around
then action = 'Flaps - 50 percent' cf 99.

rule-84:
if diagnosis = single-engine-go-around
then action = 'Gear - Up.' cf 99.

rule-85:
if diagnosis = single-engine-go-around
then action = 'Flaps - Up (100 KIAS minimum).' cf 99.

question(altitude) = 'What is the current altitude?'.
legalvals(altitude) = integer(0,10000).

question(airspeed) = 'What is the current airspeed?'.
legalvals(airspeed) = integer(0,300).

question(fire-light) = 'Are either of the engine fire lights on?'.
legalvals(fire-light) = [left,right,no].
automaticmenu(fire-light).

question(fuel-flow) = 'Please describe the fuel flow'.
legalvals(fuel-flow) = [fluctuating,normal,100].
automaticmenu(fuel-flow).

question(smoke-inside) = 'Is there any smoke in the cockpit?'.
legalvals(smoke-inside) = [yes,no].
automaticmenu(smoke-inside).
question(smoke-outside) = 'Is there smoke visible outside the aircraft?'.

legalvals(smoke-outside) = [yes,no].

automaticmenu(smoke-outside).

question(rough-running) = 'Is the engine running rough?'.

legalvals(rough-running) = [yes,no].

automaticmenu(rough-running).

question(exhaust-temp) = 'Please describe the exhaust temperature.'.

legalvals(exhaust-temp) = [fluctuating,excessive,normal].

automaticmenu(exhaust-temp).

question(fluct-rpm) = 'Is the RPM fluctating?'.

legalvals(fluct-rpm) = [fluctuating,normal].

automaticmenu(fluct-pm).

question(rpm) = 'What is the RPM?'.

question(stopping-distance) = 'Please describe the stopping distance.'.

legalvals(stopping-distance) = [sufficient,minimal].

automaticmenu(stopping-distance).

question(fumes-present) = 'Are fuel fumes present?'.

legalvals(fumes-present) = [yes,no].

automaticmenu(fumes-present).

multivalued(action).

question(right-engine-windmilling) = 'Is the right engine windmilling?'.

legalvals(right-engine-windmilling) = [yes,no].

automaticmenu(X-engine-windmilling).
question(left-engine-windmilling) = 'Is the left engine windmilling?'.
legalvals(left-engine-windmilling) = [yes,no].

question(right-engine-frozen) = 'Is the right engine frozen?'.
legalvals(right-engine-frozen) = [yes,no].

question(left-engine-frozen) = 'Is the left engine frozen?'.
legalvals(left-engine-frozen) = [yes,no].

automaticmenu(X-engine-frozen).

question(left-engine-flameout) = 'Did the left engine flameout?'.
legalvals(left-engine-flameout) = [yes,no].

question(right-engine-flameout) = 'Did the right engine flameout?'.
legalvals(right-engine-flameout) = [yes,no].

automaticmenu(X-engine-flameout).

question(gear) = 'In what position is the landing gear?'.
legalvals(gear) = [indicates-up,indicates-down].

automaticmenu(gear).

rule-86:
    if emergency = overheat
    then emergency = one-engine-failure.
Bibliography


