2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN INTELLIGENT TUTOR SYSTEM FOR VISUAL AIRCRAFT RECOGNITION

by

Larry W. Campbell

June 1990

Thesis Advisor: Yuh-jeng Lee

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL (If applicable) CS	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
AN INTELLIGENT TUTOR SYSTEM FOR VISUAL AIRCRAFT RECOGNITION

12. PERSONAL AUTHOR(S)
Campbell, Larry W.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM ____ TO ____	June 1990	173

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Intelligent Tutoring System, Intelligent Computer Aided Instruction, Visual Aircraft Recognition

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Visual aircraft recognition (VACR) is a critical skill for U.S. Army Short Range Air Defense (SHORAD) soldiers. It is the most reliable means of identifying aircraft, however VACR skills are not easy to teach or learn, and once learned they are highly degradable. The numerous training aids that exist to help units train soldiers require qualified instructors who are not always available. Also, the varying degrees of proficiency among soldiers make group training less than ideal. In an attempt to alleviate the problems in most VACR training programs, an intelligent tutor system has been developed to teach VACR in accordance with the Wings, Engine, Fuselage, Tail (WEFT) cognitive model. The Aircraft Recognition Tutor is a graphics based, object oriented instructional program that teaches, reviews, and tests VACR skills at a level appropriate to the student. The tutor adaptively coaches the student from the novice level, through the intermediate level, to the expert level. The tutor was provided to two U.S. Army Air Defense Battalions for testing and evaluation.

The six month implementation, testing, and evaluation process demonstrated that, using existing technology in Computer Science and Artificial Intelligence, useful training tools could be developed quickly and inexpensively for deployment on existing computers in the field.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Yuh-jeng Lee	22b. TELEPHONE (Include Area Code) (408) 646-2361	22c. OFFICE SYMBOL CS/Le

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete
SECURITY CLASSIFICATION OF THIS PAGE
☆ U.S. Government Printing Office: 1988—608-24
Unclassified

AN INTELLIGENT TUTOR SYSTEM FOR VISUAL AIRCRAFT RECOGNITION

by
Larry W. Campbell
Captain, United States Army
B.S., Florida Southern College, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Author: [redacted]
Larry W. Campbell

Approved By: [redacted]
Yuh-jeng Lee, Thesis Advisor

[redacted]
Man-Tak Shing, Second Reader

[redacted]
For Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

Visual aircraft recognition (VACR) is a critical skill for U.S. Army Short Range Air Defense (SHORAD) soldiers. It is the most reliable means of identifying aircraft, however VACR skills are not easy to teach or learn, and once learned they are highly degradable. The numerous training aids that exist to help units train soldiers require qualified instructors who are not always available. Also, the varying degrees of proficiency among soldiers make group training less than ideal. In an attempt to alleviate the problems in most VACR training programs, an intelligent tutor system has been developed to teach VACR in accordance with the Wings, Engine, Fuselage, Tail (WEFT) cognitive model. The Aircraft Recognition Tutor is a graphics based, object oriented instructional program that teaches, reviews, and tests VACR skills at a level appropriate to the student. The tutor adaptively coaches the student from the novice level, through the intermediate level, to the expert level. The tutor was provided to two U.S. Army Air Defense Battalions for testing and evaluation.

The six month implementation, testing, and evaluation process demonstrated that, using existing technology in Computer Science and Artificial Intelligence, useful training tools could be developed quickly and inexpensively for deployment on existing computers in the field.

TABLE OF CONTENTS

I. INTRODUCTION

A. HISTORICAL PERSPECTIVE

Fratricide. Webster defines it as the act of murdering or killing one's own brother or sister or an individual having a relationship with him like that of a brother or sister. Ask a pilot about fratricide, though, and you are likely to get a definition such as "being shot down by your own forces", and will almost certainly be told that it represents an unacceptable risk (Weidman, 1985, p. 17). Fratricide was a fact of life during World War II, but occurred less often during the Korean War, and was almost non-existent during the Viet Nam conflict. Does this mean that we have learned from our mistakes, that fratricide is a tragedy of the past that no longer poses a threat to our air forces? Recent training exercises indicate that the threat remains and may in fact be greater today than any other time in history. Today's battle environment is characterised by a plethora of high technology weapons that are capable of destroying the most sophisticated helicopters and jet fighters in the world. Combine this with decentralized authority for engaging aircraft and the threat seems overwhelming and unmanageable.

B. THE THREAT

Consider the following minimalized scenario: A U.S. aircraft is returning from a strike mission. To get back to the airbase he must pass through an Army Division. Within this division area, 20 Stinger missile teams are deployed, in such a manner as to have overlapping coverage zones. So, to return to the airbase, this aircraft will come within the engagement zone of at least two teams. Each of these teams consists of two men, an E-4 or E-5 team chief and an E-1 through E-3 missileman. Final authorization for engaging the aircraft is made by the team chief and is based on visual identification of the aircraft. Assuming that an average team chief will correctly identify 90% of the aircraft that he sees, the pilot of the aircraft has an 80% chance of passing through the division area without being fired upon. The lethality of a Stinger missile is well documented. But now consider a more realistic scenario: the same aircraft is passing through the same division area, but this time must pass through a more likely number of Stinger teams, as well as several Vulcan gun crews, perhaps a Chaparral missile crew, and countless soldiers with small arms, all of which use *visual identification* as a final means of determining whether or not to engage an aircraft. The odds of the aircraft returning from the mission after successfully evading the enemies air defense forces are greatly reduced. The life of the pilot hinges upon his aircraft being correctly identified by each of the short range air defense fire units that he comes within range of.

C. THE PROBLEM

Why is there such a dependence on visual identification of aircraft? Other methods of identifiying friendly aircraft are available. Interrogation, Friend or Foe (IFF) systems exist, but are notoriously unreliable. Vulcan gun crews don't even have IFF capability, and what about allied integrated air defense systems? Air Battle Management techniques such as safe corridors work well for high altitude air defense weapons that rely on integrated radar systems for identifying and tracking aircraft, but safe air corridors are difficult to identify from the ground, and aircraft move through an area faster than current chain of control methods can keep up with. The most reliable means of identifying aircraft is still visual aircraft recognition (VACR), but VACR skills are not easy to teach or to learn, and once learned these skills are highly degradable (Pliler, 1984, p. 14).

D. GOALS AND OBJECTIVES

The ***Aircraft Recognition Tutor*** was developed in an attempt to fill a gap that exists in most VACR training programs. Because the tutor is designed to identify the soldiers' current ability and teach at a level appropriate to that ability, it is useful for introducing VACR to new soldiers as well as providing refresher training to more advanced soldiers. After implementation, the ***Aircraft Recognition Tutor*** was provided to two U.S. Army Air Defense Battalions for testing and evaluation. In the course of implementing, testing, and evaluating the tutor (six months), it was demonstrated that, using existing technology in Computer Science and Artificial Intelligence, useful training tools could be developed quickly and inexpensively for deployment on existing computers in the field.

In the Chapter II, we examine the training aids and methods currently being used for teaching visual aircraft recognition. Then, in Chapter III, we describe and evaluate the WEFT cognitive model in terms of its effectiveness as a basis for training VACR. Chapter IV consists of a survey of the attempts of others at developing intelligent tutoring systems and is presented along with a description of a basic model for an intelligent tutor and techniques that can be used to evaluate the effectiveness of a tutor. Chapter V is dedicated to describing the design and implementation of the ***Aircraft Recognition Tutor***, and Chapter VI offers an objective evaluation of the tutor as both a VACR training tool and an intelligent tutoring system. We conclude with a summary of our accomplishments, recommendations for future improvements and modifications to the tutor, and examples of other knowledge domains that the tutor could easily be adapted to teach.

II. AIRCRAFT RECOGNITION TRAINING

The requirement for visual aircraft recognition (VACR) by Short Range Air Defense (SHORAD) soldiers is emphasized in many publications, including Q-STAG 699, JCS publications, and Air Defense doctrinal manuals (Pliler, 1984, p. 14). As of October 1989, standards require SHORAD soldiers to be able to identify visually by NATO name and nomenclature 50 threat and friendly aircraft with an accuracy of at least 90 percent after an exposure period of five seconds. The goal is 100 percent accuracy (Pliler, 1988, p. 38). This is a daunting task, and numerous training aids have been developed to assist in achieving this level of proficiency. In addition, guidelines for training programs have been suggested in order to ensure effective training.

A. TRAINING TECHNIQUES AND GUIDELINES

1. Wings, Engine, Fuselage, Tail (WEFT)

The WEFT theory is currently believed to be the best method for teaching VACR. In this theory, all aircraft are composed of the same basic elements: wings to provide lift, an engine to provide motive power, a fuselage to carry the payload and controls, and a tail assembly which usually serves the purpose of controlling the direction of flight. These elements differ in their shape, size, number, and position. It is these basic elements that distinguish one aircraft type from another. Detailed parts cannot be used as the only aid to aircraft recognition, mainly because of the distances at which recognition should occur. The individual components can be isolated for descriptions and studied as separate recognition features. It is the composite of these features that must be learned in order to recognize and identify an aircraft (FM 44-30, 1986, p. 3.2).

2. **Simplicity**

Descriptive terms must be kept simple so that the soldier's energy is spent on learning VACR, not obscure specifications (Pliler, 1984, p. 15) (FM 44-30, 1986, p. 4.3). Learning aircraft recognition is difficult enough, without confronting the student with hundreds of odd names, curious designations, incomprehensible specifications, and strange descriptive terms (Wood, 1985, p. 9).

3. **Comparison**

Many aircraft have a similar appearance, and it may be difficult for a soldier to recognize the differences between them unless he is given an opportunity to view them side by side at the same aspect, heading, and climb angle. Given this, the soldier can then see directly the differences that distinguish the aircraft, and learn how not to confuse them (Pliler, 1984, p. 15) (Pliler, 1988, p. 39) (FM 44-30, 1986, p. 4.3).

4. **Controlled Image Training**

Individual components of aircraft should be isolated as separate recognition features for description and study. The composite of these features are then learned in order to identify a particular aircraft (Pliler, 1984, p. 15).

5. **Segregation**

Soldiers should be trained according to their ability. New soldiers and soldiers needing detailed instruction should be taught separately from more advanced soldiers who may only need review and testing (Pliler, 1988, p. 39).

6. **Repetition**

VACR skills are highly degradable. They are lost in a short time if reinforcement training is not vigorously pursued (Pliler, 1984, p. 15). Therefore, training should be scheduled regularly and in periods of no more than two or three hours (Pliler, 1988, p. 39).

B. TRAINING AIDS AND DEVICES

Several training aids and devices have been developed in order to help units train their soldiers in visual aircraft recognition. These aids include posters, charts, card sets, slides, movies, models, and correspondence courses. The most successful of these aids will be discussed.

1. Ground Observer Aircraft Recognition (GOAR) Kit

The most widely used and reliable training aid for VACR has been the GOAR kit. The current version of this kit consists of 528 35-mm photographic slides of 48 military aircraft. The aircraft are separated into the categories of helicopters, cargo, reconnaissance, fighters, bombers, and CAS (Close Air Support) (FM 44-30, 1986, p. 4.7). Used correctly, this kit can be very effective in training soldiers in aircraft identification skills. However, this aid requires an experienced instructor and different training methods for soldiers with different skill levels. In addition, the slides tend to age and become damaged to the point that recognition is frequently based on the characteristics of the particular slide (eg. scratches or blotches on the slide) rather than the characteristics of the aircraft. This results in a less effective tool for training soldiers.

2. Visual Aircraft Recognition Playing Cards

This training aid, used similar to flash cards, is an excellent tool for training soldiers on maneuvers or for "hip-pocket" training where access to more sophisticated aids or a power source is not available (Pliler, 1986, p. 15). Again, though, without a qualified instructor, it is not particularly effective.

3. VACR TEC Lessons

In an effort to overcome the need for an instructor, TEC lessons attempt to allow soldiers to learn visual aircraft recognition skills at their own pace and on their own schedules. The lessons consist of a continuous super 8-mm reel film with audio cassette sound. The film is projected on either a six by eight inch screen for individual training, or a larger movie screen for classroom presentation. The cassette tape contains the sound track for the training material, and both teaches and tests the student (FM 44-30, 1986, p. 4.6). The problem with this training device is that the tapes are a continuous reel, therefore random selection cannot be made while training or testing. The training is always presented in the same order.

4. FM 44-30

This manual is not so much a training aid or device as it is a doctrinal guide for the conduct of visual aircraft recognition training. It is the source for a description and explanation of WEFT theory and training guidelines and tips. This manual contains line drawings, photographs, descriptions, and specifications of the aircraft considered to be most relevant to SHORAD soldiers. It is the most comprehensive source of information concerning visual aircraft recognition.

III. THE WEFT COGNITIVE MODEL

A. DESCRIPTION OF THE WEFT MODEL

As was discussed in Chapter II, the Wings, Engine, Fuselage, Tail (WEFT) theory is based on isolating the specific features that are common among all aircraft, identifying these features, and using the composite of these features to identify the specific aircraft being viewed. This section will decribe the features that are integral to the WEFT model. The information in this section is taken directly from U.S. Army Field Manual FM 44-30 (FM 44-30, 1986, p. 3.2 - 3.10).

1. Wings

Features of wings useful in aircraft identification include location, shape, slant, taper, and wing-tip shape.

a. Location

The location on the fuselage at which the wing is attached is a valuable aid in distinquishing one aircraft type from another, particularly when viewing the aircraft from a nose-on or tail-on aspect. The usual wing positions for fixed-wing aircraft are High-Wing, Mid-Wing, and Low-Wing.

b. Shape

Wing shapes differ and are also valuable recognition features. They are classified according to their dihedral (slant), taper (diminishing width) and wing-tip shape. A wing may contain any combination of slant, taper, or wing-tip shape.

c. Slant

The vertical angle of the wing with respect to a horizontal line drawn through the fuselage is called wing dihedral. For recognition purposes dihedral will be referred to as slant.

d. Taper

Aircraft may have the leading, trailing, or both edges of the wing tapered, or the wing may be untapered.

e. Tip Shape

Wing-tip shapes are determined by the manner in which the leading and trailing edges of a wing meet. Wing-tip shapes are classified as Square, Pointed, Rounded, Blunt, or Curved.

2. Engine

Engine types, numbers and locations, and air intake shape and location play a large role in the identification of a particular aircraft.

a. Propellor Aircraft

Generally, engines which drive propellors are located on or within the fuselage, usually the nose for single-engine aircraft, and within the wing or on the leading edge of the wing for multiengine aircraft.

b. Jet Aircraft

Generally, single-engine jet aircraft have the engine mounted inside the rear section of the fuselage. Multiengine jet aircraft have their engines located along the sides of the fuselage or under the wings.

c. ***Intake Location***

Examples of jet air intake locations for single and multiengine aircraft include in the wing root, outboard on the wings, beneath the wings, in the nose, in the sides of the fuselage, and beneath the nose.

3. **Fuselage**

The body of an aircraft is made up of three separate and distinct sections. They are the nose, mid, and rear sections. The canopy/cabin is also discussed when describing a fuselage.

a. ***Nose Section***

The front or forward section of the aircraft fuselage is called the nose. Nose sections are classified by their shapes: pointed, blunt, or rounded.

b. ***Midsection***

The center section of an aircraft fuselage is known as the midsection. This section normally provides the space for crew compartments and internal stores. Midsections are classified according to their shapes: slender, bulging, tubular, or thick.

c. ***Rear Section***

The rear of the fuselage where the tail assembly is attached is known as the rear section. Rear sections are classified by their shape: upswept, blunt, or tapered.

d. ***Cockpit, Cabin***

The cockpit or cabin is the compartment in an aircraft that accommodates the pilot and/or other persons. It is usually covered by a transparant canopy or glassed-in enclosure, and may be bubble, stepped, or flush.

4. Tail

Tail recognition features on aircraft consist of the tail flat (horizontal piece) and the tail fin (vertical piece).

a. Tail Flat Position

The position of the stabilizer in relation to the fin or the fuselage is an aid to recognition. The Tail Flat may be low, mid, or high mounted on either the tail fin or the fuselage.

b. Tail Flat Slant

The vertical angle of the tail flat, with respect to a horizontal line drawn through the fuselage, is referred to as slant, either positive or negative.

c. Tail Flat Shape

Tail flats usually consist of only one element and are classified like wings; for example, tail flat shape and tail flat tip shape.

d. Fin Shape

There are many fin shapes, including unequally tapered, equally tapered, back tapered, swept-back, round, and oval. In addition, the fin tip shape can be accounted for.

e. Number of Fins

There are three combinations of fins usually seen on aircraft: single, double, or triple.

B. EVALUATION OF THE WEFT MODEL

The WEFT model was first introduced in October of 1983 (Pliler, 1984, p. 16). Since that time it has undergone rigorous evaluation by U.S. Army Air Defense units, and has stood the test of time. In personal observation of VACR training, it was noted that those soldiers who reliably recognized aircraft did so based on the WEFT characteristics of the aircraft. In fact, some of the "experts" could identify an aircraft based solely on a verbal WEFT description of it, without a visual image. Soldiers that had trouble identifying aircraft were in one of two categories: either they were new soldiers, and had not yet mastered the concepts of the WEFT theory, or they had learned to identify aircraft based on some other less reliable means of identification. Some of the other methods used by soldiers to identify aircraft include reliance on aircraft heading and speed, aircraft markings, and ordnance types. Each of these is an unreliable feature.

Criticism of the WEFT method has arisen however. Jane's World Aircraft Recognition Handbook describes mnemonic aircraft-recognition systems like WEFT as "completely useless, as recognition depends on the aircraft's total appearance and there is usually little enough time to look for individual features, let alone reciting sets of letters" (Wood, 1985, p. 5 - 6). However, it is because of the limited time available to recognize an aircraft that the discipline of the WEFT model pays off. Rather than attempting to search for a match of an overall image of the aircraft to all of those possible, the WEFT method prunes the search space by categorizing aircraft based on reliable, easily recognizable features. In addition, not all features are necessary to single out a particular aircraft from all others (indeed, it is rare that all features will be visible). In fact, even in Jane's World Aircraft Recognition Handbook, the student is advised to "work in stages, gradually acquiring knowledge and storing it. Like people's faces and figures, every aircraft and helicopter has its own characteristics. It may have one engine and straight wings; four engines and swept wings; a fat fuselage; a triangular wing; a tail plane on top of the fin; or, in the case of a helicopter, one or two rotors" (Wood, 1985, p. 9). Aircraft in this manual are separated into chapters based on "the main characteristics of each class of aircraft, so that the user can find his way immediately to the right chapter after catching a fleeting glimpse of, say, a delta shape in the sky" (Wood, 1985, p. 6). This categorization is analogous to the WEFT categorization, with the exception that WEFT does not give a predetermined preference to one category over another. This allows for a higher probabitity that an aircraft will be correctly identified given an incomplete description or an obscured image.

IV. SURVEY OF INTELLIGENT TUTORING SYSTEMS

A. INTRODUCTION

Intelligent tutors are computer programs that use Artificial Intelligence (AI) techniques to help a person learn. The most important part of developing an Intelligent tutor is choosing an appropriate way of representing the knowledge to be taught. Another major concern is how to model the student's current understanding of a topic or problem. Intelligent tutors use inferencing mechanisms capable of reasoning from one item of knowledge to another. Therefore, expert systems are an integral part of nearly all Intelligent Tutors (Harmon, 1987, p. 170).

Intelligent tutors are programs that lie at the intersection of three disciplines, Computer Science (particulary AI), Cognitive Psychology, and Education and Training (Kearsley, 1987, p. 4). Because of the different goals of researchers in each of these areas, Intelligent Tutor Systems vary greatly in their implementation methodology and overall effectiveness. Individuals with interest and expertise in all three disciplines are rare. Current theories of learning are inadequate and tend to cause controversy among cognitive psychologists (Bower & Hilgard, 1981, p. 17). So, in spite of a well developed computer science technology base, Intelligent Tutor Systems do not exist in large numbers and are not being used regularly by schools. Another major obstacle to the implementation and deployment of Intelligent Tutor Systems is the bureaucratic complexity of the institutions they are designed for (Bower & Hilgard, 1981, p. 575).

B. DESIGNS

Most of the current examples of Intelligent Tutor Systems can be classified by one of five paradigms (Kearsley, 1987, p. 6). Because the ***Aircraft Recognition Tutor*** attempts to capture an existing, proven strategy for teaching visual aircraft recognition skills, we fit the tutor to the appropriate design paradigm rather than choosing a preferred design as the starting point for the development of the system. The ***Aircraft Recognition Tutor*** uses the Coach paradigm, and because both the domain knowledge and the tutorial knowledge are well structured, this paradigm provides the best performance. In general, however, the Microworld design is the most flexible and gives the most control to the student. This design lends itself well to unstructured knowledge domains and teaching situations where the specific information to be taught to the student is not critical. With the Microworld paradigm, and given a student workbook or study guide, a student can learn about the topics that are of immediate interest to him. This increases the student's motivation to learn. The Microworld paradigm is particulary suitable for tutor systems that attempt to cover a broad range of subjects in a single system, or for a group of smaller tutor systems all under the contol of a single integration system.

1. Mixed Initiative Tutors

Mixed Initiative Tutors are the oldest style of intelligent tutor systems. In this method, the program engages the student in a two-way conversation and attempts to teach the student via the Socratic style of guided discovery. In this teaching style, the tutor first attempts to diagnose the student's misconceptions and errors, and then presents instruction that will help the student to recognize the error themselves (Park, Perez, & Seidel, 1987, p. 18). Examples of this approach include the SCHOLAR and SOPHIE programs.

The SCHOLAR program was a frame-based tutor which could teach students about the geography of South America (Goldstein, 1982, p. 53). The tutor was capable of either drilling the student in a question and answer style, or accepting questions from the student. One of the major criticisms of SCHOLAR was that it encoded only a limited depth of information about its domain, and made no attempt at providing the student with multiple levels of detail (Goldstein, 1982, p. 53).

SOPHIE was a simulation tutor that taught electronic circuit diagnosis. This was, and remains, one of the best attempts at modeling a natural language tutoring discourse (Woolf, 1988, p. 10). The natural language parser was able to skip over "noise" words and looked for semantic classes rather than syntactic entities (Sleeman & Hendley, 1982, p. 111). Unfortunately, students do not always organize and talk about knowledge in a way that reflects the way the developer of the tutor anticipated. Because of this, the tutor must look beyond the words that the student uses and determine the true meaning of the student's answers and queries. Failure at this results in the tutor being unable to guide the student, or properly model the student's understanding of the topic being taught (Woolf, 1987, p. 241). True natural language understanding eludes us even today, so most recent efforts attempt to communicate with the student through some other type of interface.

2. Diagnostic Tutors

Diagnostic Tutors try to identify the misconceptions a student may have in solving a problem by using a catalog of common problems. The BUGGY tutor was designed to teach basic mathematical problem solving skills (Park, Perez, & Seidel, 1987, p. 19). This program demonstrated that there exists a striking uniformity in the errors made by students in a discipline that is systematic in nature (Matz, 1987, p. 47). The BUGGY tutor models common mathematical errors, and when a student makes a mistake, attempts to identify the mistake by matching the procedure that the student used in solving the problem with its knowledge base of "buggy" procedures. Once the match is made, the tutor can explain not only the student's error, but teach in a way that will help the student understand the problem and its solution better.

Another example of a diagnostic tutor is PROUST. PROUST is an automatic debugger for Pascal programs. It uses a knowledge base of programming errors in order to detect and report bugs in a student's program. In order to do this however, PROUST must have some understanding of what the program is supposed to do. When PROUST fails to understand a program's goals, its ability to recognize bugs deteriorates. In fact, the developers of PROUST are not yet sufficiently satisfied with its performance to make it generally available to their students (Johnson & Soloway, 1987, p. 66).

3. Microworlds

Microworlds involve developing a computational tool that allows a student to explore a problem domain. The most famous example of this paradigm is the LOGO programming environment. LOGO was developed by Seymour Papert of M.I.T. as a tool for teaching mathematics and geometry to children. LOGO combines computational theory and artificial intelligence with Piaget's theory of learning. In this theory, people learn by naturally and spontaneously interacting with their environment (Papert, 1980, p. 156). With this, LOGO takes an opposite approach to learning than most Intelligent Tutor Systems. Where, in most cases, the computer is being used to teach or "program" the student, in LOGO, the student "programs the computer and, in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes an intimate contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building" (Papert, 1980, p. 5). Papert believes that, using the microworld paradigm, students not only learn about the tutor domain, but learn about the process of learning itself. The lack of discipline enforced by the tutor is typical of programs that use the microworld concept; the student is left on his own to explore and discover the relationships and truths about the knowledge domain. Inherent in this paradigm, then, is the need for some type of supervision and assistance other than the tutor.

4. Articulate Expert Systems

Articulate Expert Systems are expert systems that can explain their decisions. Most of these systems do not include an instructional component, and any expert system that includes the ability to explain how it reasons could be used as a limited tutor. A complete system that exemplifies this paradigm is GUIDON, an adaptation of the MYCIN expert system for medical diagnosis. The adaptation revealed that although MYCIN has an explanation facility, the explanations were "narrowly conceived" (Clancey, 1987, p. 201). Specifically, the expert system could not explain why a particular rule was correct, and it could not explain the strategy behind the design of its rule structure (Clancey, 1987, p. 198). To develop GUIDON, the expert knowledge not only had to be captured, but how the expert uses and remembers the knowledge also needed be known. With this additional information, MYCIN's rules could be made more explicit, and then be related to GUIDON's teaching of the heuristics used by the expert system.

5. Coaches

Coaches observe the student's performance in some problem solving activity and provide advice or guidance that will help the student perform better. WUMPUS is a maze exploration game that exercises basic skills in logic and probability. Several tutors (WUSOR-I, Wumpus Advisor) have been developed for the WUMPUS game (Goldstein, 1982, p. 54) (Kearsley, 1987, p .5). Each of these use the coach paradigm. WUSOR-I uses a genetic graph to model the students performance in the game. The student progresses through the graph as he proceeds through the maze. Based on the student's path and current location in the genetic graph, a specific tutoring topic for instructing the student is chosen, and with this, a means of explaining the topic is chosen based on the predecessors to the student's position in the graph (Goldstein, 1982, p. 64). In this way, the tutor can provide variations on explanations based on its perceived judgement of the student's needs. Thus, a coach tutor attempts to know (1) when to interrupt the student's problem solving activity, and (2) what to say once it has been interrupted (Burton & Brown, 1982, p. 80).

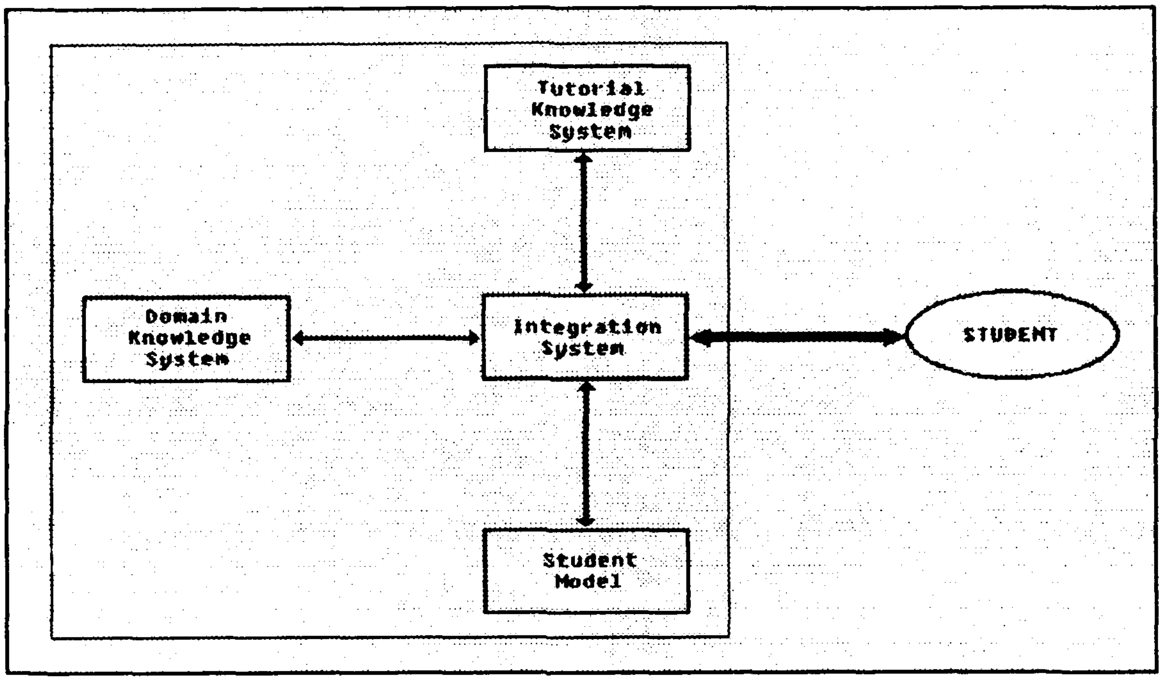


Figure 4.1 Architecture of an Intelligent Tutor System (Harmon, 1987, p. 171)

C. COMPONENTS

A typical model for an Intelligent Tutoring System, shown in Figure 4.1, includes four components, each of which can be thought of as an expert system in itself (Harmon, 1987, p. 171).

1. The Domain Knowledge System

The Domain Knowledge System is the knowledge base for all of the expertise and information about the subject matter to be taught. Techniques for representing the knowledge vary greatly, and choosing the appropriate representation method is crucial to the effectiveness of the tutor.

2. The Tutorial Knowledge System

The Tutorial Knowledge System contains the theory of the teaching method that is used. Traditionally, computer based instruction relied on the concepts of B. F. Skinner and other behavioral psychologists who developed the theory of programmed instruction (Harmon, 1987, p. 165). Skinner proposed a set of qualities that could be found in any good programmed instruction: (1) A good tutor begins where the student is, and does not insist on moving beyond what the student can comprehend, (2) A good tutor moves at a rate that is consistent with the ability of a student to learn, (3) A good tutor does not permit false answers to remain uncorrected, and (4) A good tutor does not lecture; instead, by his hints and questioning he helps the student to find and state answers for himself (Bower & Hilgard, 1981, p. 566). According to Skinner's model of a teaching machine, a tutor system first diagnoses a student to determine the skills and knowledge of the student, then presents the instruction in a carefully programmed way so that each step can be mastered by the student and leads him to the next step, providing immediate feedback on how well the student has mastered the material. A student that fails to respond adequately is provided corrective feedback or the program branches to another style of instruction. At the end of each step the program tests the student on his overall mastery of the material, and maintains a complete record of the student's performance for the entirety of the material covered. A system such as this thereby provides all of the functions of assessing, teaching, and evaluating a student in an objective, standardized manner (Iano, 1987, p. 266).

3. The Student Model

The Student Model is a database that is created during the operation of the program that reflects what the student knows and does not know about the subject matter. Two main approaches are common in modeling the student's knowledge: (1) the tutor can keep track of what the student has not yet shown a knowledge of, or (2) the tutor can track what the student has demonstrated a lack of knowledge of. In an attempt to build a more sophisticated Student Model that could more accurately reflect the state of the student's understanding, some tutor systems combine both of these approaches within the Student Model.

4. The Integration System

The heart of the system, though, is the Integration System, which uses heuristic rules to combine the three other systems into an interface that is presented to the user, and controls the flow of the program, which is usually non-deterministic. A wide array of different techniques have been used to present the information to the student, including attempts at natural language interfaces and graphical interfaces. The control of the program has also varied from strict programmed control by the tutor to an exploratory approach where the student has complete control over the flow of the instruction.

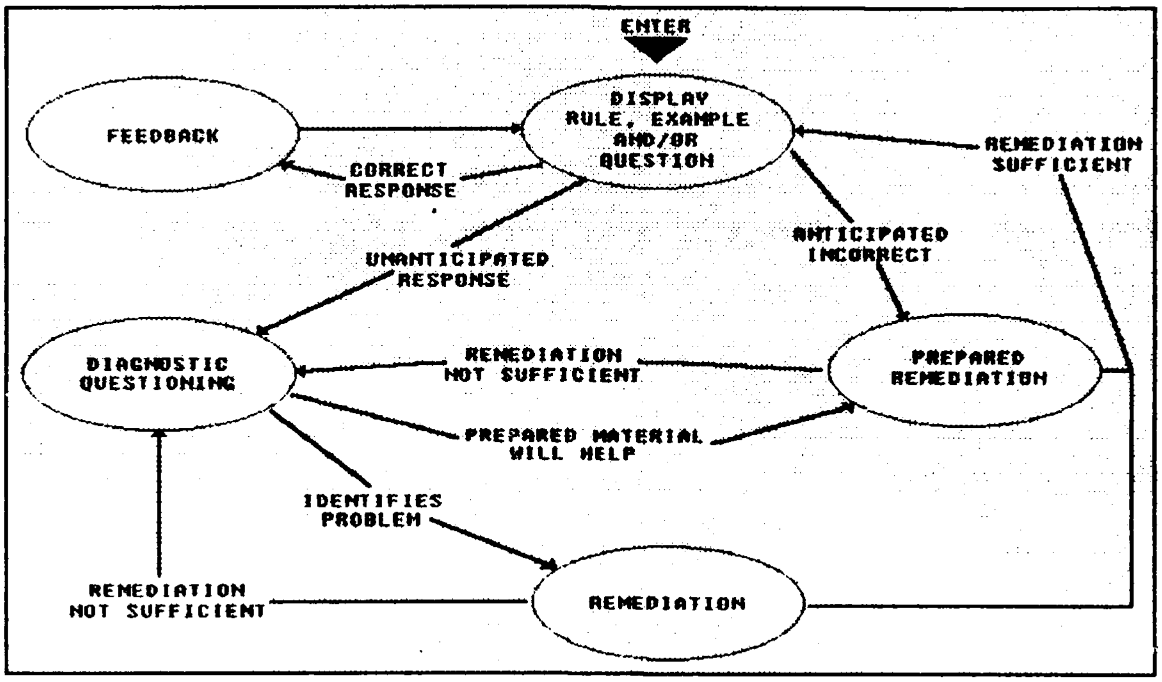


Figure 4.2 Basic Tutorial Pattern (Godfrey & Sterling, 1982, p. 49)

D. BEHAVIOR

Typical behavior for an Intelligent Tutoring System is shown in Figure 4.2 (Godfrey & Sterling, 1982, p. 49).

To begin, the program displays a rule, example, or question and prompts for a response from the user. If the user responds correctly, it is easy enough to handle, the program simply offers the student positive feedback and continues on with another question. However, two possibilities exist in the event that the student responds with an incorrect answer. The easiest to handle is an expected incorrect response. This is an incorrect response that is commonly made by students in answering the particular question, and in this case, the programmer has coded in a canned remediation sequence. After the student recieves the remediation, the program continues with another question. The other case, when the student returns an unanticipated response to the question, is the hard part, and this is where AI comes in. The program responds by questioning the student in an attempt to identify where the trouble is. Based on the students responses, the tutor either determines that some canned remediation sequence would help the student, or that some combination of remediation is called for. After the remediation is given, the student is again questioned and when the tutor is satisfied, it continues on with the main flow of the program. In each case, the student model is updated to reflect the student's understanding. A history of related wrong answers may suggest that an adaptation of the tutoring style or content in needed. Obviously, this is the hard part in creating an Intelligent Tutor, and is why some of the best examples have come from individuals or teams that have backgrounds in all three of the disciplines that make up this field. This model is a simplistic one but it is good as a general model.

E. EVALUATION CRITERIA

Intelligent Tutor Systems can be evaluated based on how well they accomplish four main activities: modeling of knowledge and reasoning, communication, cognitive processing, and tutoring (Woolf, 1988, p. 34).

1. Modeling of Knowledge and Reasoning

A good tutor will have represented the domain knowledge of the subject to be taught in such a way that it can reason about that knowledge. It may at first seem that any expert system should be able to do this, but as Clancey found while implementing the Guidon tutor for the Mycin expert system, reasoning about knowledge with the goal to teach is much different from reasoning about knowledge with the goal to diagnose or provide solutions (Clancey, 1987, p. 196). A good tutor will not only be able to reason about and explain the knowledge, but also understand the strategy behind the knowledge representation well enough to provide analogies, multiple views, and levels of explanation (Clancey, 1987, p. 201).

2. Communication

Tutor systems should take full advantage of the hardware capabilities for which they were implemented. With the advanced graphics capabilities of today's computer systems, good tutors should include some combination of simulations, animations, icons, pop-up windows, and pull-down menus in an attempt to provide an intuitive interface to the user (Woolf, 1988, p. 6). The student's time should be spent learning the domain being presented instead of learning how to interact with the tutor. With the current capabilities of independent speech recognition and synthesis systems, consideration should be given to including this type of interface for a tutor (Gallant, 1989, p. 2). To be most effective though, speech understanding should be included and this technology is not yet mature. Perhaps once we can reliably understand speech as well as recognize it, the Mixed-Initiative tutor paradigm will once again prevail.

3. Cognitive Processing

Cognitive Processing is concerned with modeling a methodology for teaching the domain knowledge, and modeling how a student learns within that domain. This is probably the most difficult part of creating a good tutor (Woolf, 1988, p. 7). In order to effectively teach a student, we must know or be able to diagnose whether the student understands the material being presented, and if not, what instructional style and content would help the student understand. A good tutor will have a robust and flexible teaching style, so that it can adapt to individual student motivations and capabilities. It will also have a dynamic student model that can reason about the student's knowledge or understanding about the material. Naturally, the more complex and fragmented the knowledge domain, the more difficult effective cognitive processing becomes.

4. Tutoring

Cognitive processing leads into tutoring. Once we identify the student's understanding of the material, tutoring takes over. Tutoring is the specific instructional style used by the system at any given time. This includes praising, remediating, interrupting, and presenting examples to the student (Woolf, 1988, p. 7). The best tutors attempt to respond to the ideosyncrasies of a student in an effort to motivate the student to continue to use the tutor. It is apparant that each of the four activities of a tutor system interact with each other and may either enhance or detract from the effectiveness of the other activities. For example, the tutoring activity's goal of motivating the student is greatly enhanced by a user-friendly communication medium, but degraded if the communication medium turns the user away from the system.

V. DESCRIPTION OF THE AIRCRAFT RECOGNITION TUTOR

The ***Aircraft Recognition Tutor*** was developed using Turbo Pascal v5.5 which also provides Object Oriented Programming support. Each of the major components of the tutor exists as an object in the program, and may contain other smaller objects as variables. Because the tutor was developed in the Object Oriented Programming paradigm, modularity is well defined. This allows changes or improvements to be made to the individual components of the tutor without the need to modify the remainder of the program.

A. DOMAIN KNOWLEDGE BASE

The domain knowledge is captured in a multi-media database consisting of aircraft images stored in binary format and WEFT descriptions stored in text (ASCII) format. Each aircraft exists in the program as a composite object, thereby encapsulating both the binary and textual information about the aircraft, as well as the procedures and functions that are used to operate on that information, into a single data structure. For example, an F-16 Fighting Falcon is an instance of the aircraft object class. All aircraft objects consist of a record containing all of the WEFT information about the aircraft, and procedures for initializing and displaying the object. In addition, the aircraft class inherits from the screen object class, and as a type of screen object, an aircraft definition also includes a pointer to the graphic image of the aircraft (this pointer is created in the initialization procedure), and a procedure to hide the object (removes the graphic image from the screen).

Although it would be possible to load each of the aircraft objects into main memory during program initialization, the objects are stored on disk and only swapped into main memory when needed. Having each of the aircrafts objects defined in the system requires a tremendous allocation of heap space, and since the program has the capability of learning new aircraft definitions, the possibility of causing a heap overflow exists. Therefore, by keeping the aircraft definitions on disk, with only one or two in main memory at any given time, we minimize the memory requirement of the program at the cost of access speed to the definitions. Even with this memory swapping, however, response time to the student is quite acceptable.

The aircraft images that are included in the system were scanned in from FM 44-30 using the PCX file format. These images were then brought into a paint program, changing the resolution of the image to match that used by the tutor (CGA 640x200, two color), resized, and cleaned up. The images remain in the PCX format, and therefor could be easily modified using any PCX compatible paint program.

In addition, the tutor includes a utility routine that allows new aircraft to be defined or existing aircraft definitions to be modified. This utility consists of a simple drawing program for creating the aircraft image combined with a menu selection for identifying the WEFT features for the aircraft. The new aircraft image is saved in the PCX format to maintain compatibility with the knowledge base, and thus could also be edited in a paint program. The utility saves the new or updated definition as an aircraft object on the disk. Another utility included within the program allows the aircraft that will be taught by the tutor to be selected from among all those currently defined and stored on the disk. With this utility, the tutor can be configured to teach all the aircraft defined, or any subset of those aircraft.

B. STUDENT MODEL

The student model for the system consists of two components, a persistent part and a transient part. The persistent part of the Student Model consists of default student models for each level, which are configurable by the *Aircraft Recognition Tutor* Administrator. The default model for the Novice Level initially consists of all of the WEFT features that are contained in the system. It is not expected that this default model will be changed. The default model for the Intermediate and Expert Levels consist of all of the specific aircraft to be taught by the system. The default initially consists of 44 different aircraft (with three views of each), but can be selected from the available number of aircraft that have been defined for the system . This model represents what the student needs to be taught, and when the student demonstrates the knowledge of an item in this model, it is removed.

In addition, the persistent part of the student model tracks student performance information, as well as the student's current Mode or Level. This information includes the number of items presented to the student, the number of correct responses, the number of anticipated incorrect responses, and the number of unanticipated incorrect responses. The transient part of the student model keeps track of student misconceptions. Both parts are updated dynamically during the tutoring session, the difference being that when a session is ended the persistent part of the student model is written back to disk and will be used by the next session, while the transient part is deleted at the end of a session.

Each student that uses the system will have an individual student model. When a new student is encountered (ie. a student model does not exist for the student), the student is diagnosed by the system to determine the appropriate level and mode at which the student should enter the system. The student is then assigned a default model corresponding to the level, and begins a tutor session at the determined level and mode. The tutor system includes utilities that allow the student model database to be accessed. These utilities include the ability to delete students from the database or retrieve student administrative and performance information.

B. TUTORIAL KNOWLEDGE BASE

The Tutor system in the ***Aircraft Recognition Tutor*** consists of three tutoring modes: Teach, Review, and Test, and three levels: Novice, Intermediate, and Expert. Tutoring sessions exist for each combination of mode and level, with the exception that no sessions exist for Novice-Test and Expert-Teach. The assumptions here are that when a student has successfully completed the Novice-Review session, he is ready for more advanced instruction and should not be delayed (the Novice level is elementary enough that it is easy to learn and we do not want to risk boredom by keeping a student at this level for an inordinate amount of time), and that a student that has successfully demonstrated his expertise in identifying aircraft at the Intermediate Level need not be retaught these same aircraft (albeit from a different point of view) in the Expert Level.

1. Novice Level

The Novice Level keys in on WEFT (Wings, Engine, Fuselage, Tail) theory, and provides the student with the background necessary to consistently identify aircraft using a validated cognitive model. Each WEFT category is broken down into separate subcategories which are taught in a logical sequence by displaying a generic aircraft and modifying a particular feature of the aircraft and identifying this feature to the student. Review consists of randomly, but completely, displaying all of the features introduced in the Teach mode. The student is then expected to identify these features, and the tutor takes an appropriate course of action based on the student's response.

2. Intermediate Level

The Intermediate Level teaches the WEFT characteristics of each individual aircraft used in the current configuration of the system by presenting a visual image of the aircraft and identifying the WEFT features that distinguish that particular aircraft from another. Similar aircraft are presented along with the aircraft currently being taught to allow the student to firmly grasp the differences between the two aircraft. The focus of this level is to associate in the student's mind the visual image of the aircraft with its corresponding WEFT description. Review is accomplished by randomly but completely presenting each aircraft and allowing the student to identify the aircraft by name and nomenclature, and taking action based on the student's response. Testing consists of presenting the student with each aircraft in the default model, allowing the student to identify the aircraft, and maintaining a record of the students performance.

3. Expert Level

The Expert Level reviews and tests the student based on WEFT characteristics of the aircraft alone; no visual image of the aircraft is presented.

4. Tutorial Strategy

During a review, three possible conditions can exist when a student attempts to identify a WEFT feature of aircraft: (1) the student may respond correctly, (2) the student may respond with an anticipated but incorrect answer, and (3) the student may respond with an unanticipated, incorrect answer. Each of these conditions is handled differently by the tutor system.

In case (1), the tutor system will recognize the student for the correct answer, remove the WEFT feature or aircraft from the persistent student model, and present another item from the model to the student.

In case (2), the student is presented with both the feature or aircraft being reviewed and the feature or aircraft that the student mistook it for. Both are identified to the student, and the system performs a comparison of the two for the student so that the differences are reinforced in the student's mind. The feature or aircraft is not removed from the student model, thereby requiring the student to demonstrate identification of it again some time later in the session. The tutor then continues with another item from the student model.

In case (3), the feature or aircraft is identified to the student, and the student is asked to identify specific features of the feature or aircraft. When all of the features have been either correctly identified, or the tutor system has corrected the student, that feature or aircraft is added to the persistent student model and a link is created in the transient student model, storing the students misconception, so that if the student makes the same mistake again, it will be handled as in (2) above. The tutor session then continues as in (1) and (2).

As one can see, the student model may contain duplicate items, thus the student may have to identify a feature or aircraft several times before the system is satisfied with the student's knowledge of that item. A student has the ability to interrupt the tutor session at any time. In this way the student may end the tutor session, get context sensitive help, or request that the tutor present a specific WEFT feature or aircraft. By providing the latter facility, we allow the student to have some control over the flow of the instruction in the program. After the student has been presented the information that he requested, control is returned to the tutor which picks up where it was interrupted.

5. Student Evaluation

Student performance is evaluated when one of two events occur. First, if the student model is empty, then the student has completed a session. The student's performance will indicate whether the student will move to the next Mode or Level, remain at the current Mode or Level, or digress to the previous Mode or Level, based on heuristics included in the system. Second, the student model is limited to twice the number of items contained in the default model. If the student completely fills the student model to capacity, then the current session will be ended, and the student will begin a new session at the previous Mode or Level. When the student moves to a new Mode or Level, both the persistent and transient portions of the student model are reinitialized. If the student interrupts a session prior to completion, the persistent model will be saved to disk, but the transient model will be lost. A student reentering the system will be taught using the persistent model that existed when the last session was terminated, thereby providing continuity of instruction.

D. INTEGRATION SYSTEM

The integration system consists of the user interface and a control program that acts to direct the actions of the other three modules, including interaction between these modules. The interface for the tutor is graphically oriented and was intentionally kept basic and intuitive. It consists of several distinct objects: help screens to provide context sensitive help information to the student, dialog boxes to communicate with the student, and menus to accept information and selections from the student. "Hot" keys allow the student to easily quit what they are doing, request help, or interrupt the tutor. In addition, the tutor system includes a one or two player game mode in order to encourage usage of the program. The one player game allows the student to compete against the computer, while the two player game allows two students to compete against each other in a "Jeopardy" style game of aircraft recognition. An aircraft is presented to the players, and the first to recognize it presses their "button". The player is then given a chance to identify the aircraft in a menu that will appear. A limited amount of time is allowed for the player to recognize the aircraft. The game consists of 25 aircraft presentations. Points are awarded for a correct response, and deducted for an incorrect response. After all 25 aircraft have been shown, the player with the highest score is the winner. Performance in the game mode of the program is not tracked in the student model.

E. IMPLEMENTATION DETAILS

As mentioned previously, the ***Aircraft Recognition Tutor*** was developed using the Object Oriented Programming (OOP) paradigm. The benefits of OOP are the addition of methods (procedures and functions) to abstract data types, inheritance, encapsulation, modularity, and code reusability.

The tutor system consists of three separate programs: an install program, an unstall program, and the tutor program itself. Figure 5.1 shows the hierarchical structure of the ***Aircraft Recognition Tutor*** programs and units.

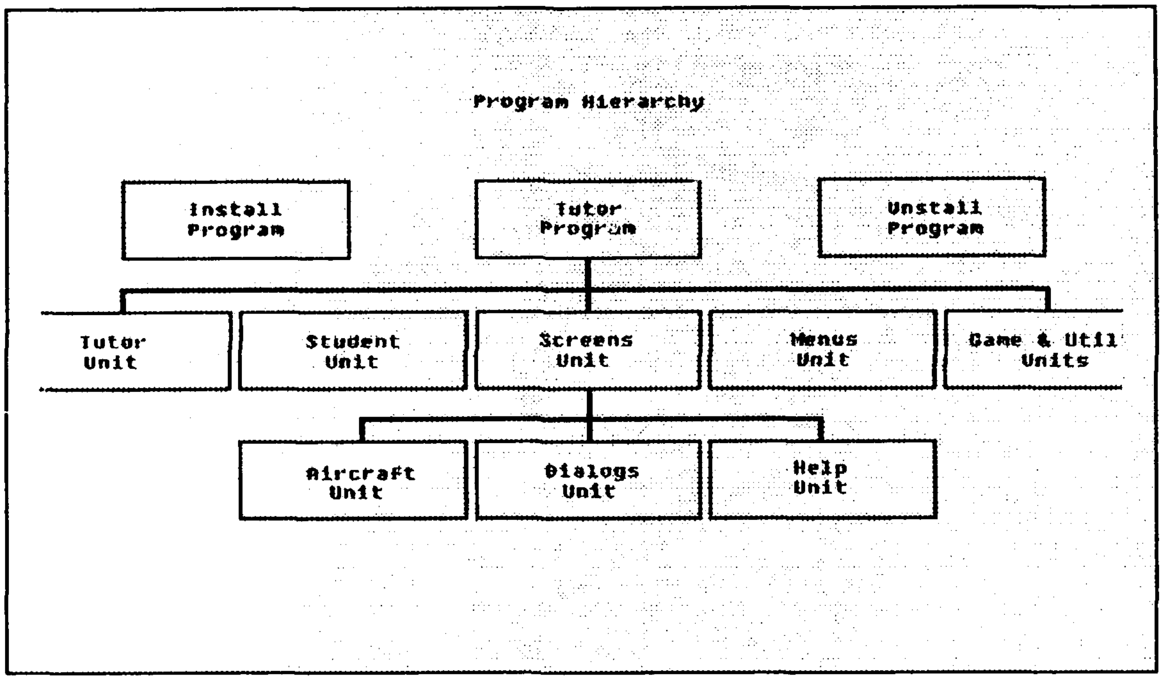


Figure 5.1 Hierarchical Structure of the Aircraft Recognition Tutor

A. The Install Program

This program allows the user to install the ***Aircraft Recognition Tutor*** on his hard disk drive simply by typing a single command and switching diskettes when prompted. In addition, the program checks that the user's system meets all of the requirements for running the tutor; for example, the install program verifies that a CGA compatible graphics adapter is present, checks for and creates a subdirectory for the tutor, insures adequate disk space is available, etc. This program will only allow one installation of the tutor system.

B. The Unstall Program

The tutor system is designed to be installed and used on a single computer. The system is copy protected to insure that incomplete or corrupted versions of the program are not distributed. In order to allow movement of the tutor system from one computer to another, the Unstall program was created. This program removes the program from the computer that it was installed on and replaces the hidden copy protection file back on the original floppy diskette. After running the Unstall program, the tutor system may be moved to another computer using the Install program.

C. The Tutor Program

The Tutor itself is comprised of a main program and several units used by the main program. These units consist of either object definitions or logically grouped functions and procedures.

1. The Main Program

The main program initializes the user's computer to be compatible with the tutor system. In addition, the main program presents the main menu to the user and passes control to the other units based on the user's input.

2. The Screens Unit

This unit defines the Screen object class. There are no instances of this class; it exists simply to allow other classes to inherit methods and variables that they have in common. A Screen object consists of variables to track the location and visibility status of the object, a file containing the graphic image of the object, a pointer to the object, and procedures to initialize, show, hide, and kill the object.

3. *The Aircraft Unit*

This unit defines the Aircraft object class. This class is a subclass of the Screen class and inherits all of the variables and methods defined for Screen objects. In addition, Aircraft objects consist of a record that contains all of the WEFT information about the aircraft, and redefine the procedures and methods inherited from the Screen class.

4. *The Dialogs Unit*

This unit defines the Dialog object class. A Dialog object is an interaction window that appears on the screen, either giving textual information to the user, or providing a location for the user to enter information. Dialog objects also inherit from the Screen class. This class redefines the Show and Hide procedures, and includes an additional pointer which is used to save the part of the screen image that is overwritten by the Dialog object so that it may be restored when the Dialog object is removed.

5. *The Menus Unit*

Menu objects are a new class of objects; they do not inherit from any other class. A Menu object consists of variables for the menu title, top item, selected item, highlighted item, number of selections, an array of menu selections, a file containing the menu information, a procedure to initialize and display the menu, and a function to get the menu selection from the user.

6. *The Student Unit*

This unit defines the Student Model object, which contains all of the information known about each individual student. An individual Student Model consists of a student name, current mode and level, the latest test score, the persistent and transient parts of the model, and the number of aircraft shown and missed during the current mode and level. In addition, procedures to get, update, save, and kill the student model, and functions to get and add entries to the model are defined.

7. *The Tutor Unit*

This unit consists of a set of procedures and functions that capture the WEFT teaching strategy. These include procedures to diagnose the student, teach and review the student at the Novice Level, teach, review, and test the student at the Intermediate Level, review and test the student at the Expert Level, and evaluate the student. In addition, functions that display individual WEFT features of an aircraft and compare WEFT features of two aircraft are defined.

8. *The Game Unit*

This unit consists of procedures that control the one and two player game modes.

9. *The Help Unit*

This unit defines the Help object class. Help objects inherit from the Screen class. This class redefines the Show and Hide procedures from the Screen class and also adds an additional pointer to save the screen that the Help object will overwrite. The procedure to get help is not defined in the Help class definition, but exists separately within this unit. This is because only one Help object can exist at one time, and defining the get help procedure outside the class definition allows it to be generic for each of the Help objects.

10. *The Utility Unit*

This unit consists of a set of procedures that allow the administrator of the tutor system to customize it and perform some database management activities. These include procedures to allow the administrator to select the aircraft that will be taught by the system from among those defined, to add to or modify the aircraft that are defined in the system, to retrieve performance and administrative information about an individual student, and to delete a student from the system.

VI. EVALUATION OF THE AIRCRAFT RECOGNITION TUTOR

The ***Aircraft Recognition Tutor*** can be evaluated in several ways. First, we will look at the tutor in terms of its effectiveness as a training aid for visual aircraft recognition. For this we will use the training guidelines described in Chapter II as a basis of evaluation[1]. Then we will evaluate the ***Aircraft Recognition Tutor*** in terms of the evaluation criteria for intelligent tutoring systems that were described in Chapter IV[2].

A. AS A VACR TRAINING AID

1. Wings, Engine, Fuselage, Tail (WEFT)

The ***Aircraft Recognition Tutor*** uses the WEFT theory as a basis for the instruction presented to the student. The student is first taught to recognize the basic elements of WEFT theory (wings, engine, fuselage, and tail) based on the differences in their shape, size, number, and position. The student is then taught to distinguish one aircraft type from another based on a composite of these WEFT elements. This instruction systematically presents the information to the student, but at the same time uses both the domain knowledge and the student model to tailor the tutoring session to the student.

2. Simplicity

The terms used to describe the WEFT features are not technical terms, but the common "layman" terms that most people associate with the different features. In addition, uniformity of naming conventions and descriptions is maintained throughout the system.

[1]See pages 4-5.

[2]See pages 24-26.

3. Comparison

During a review session, if the student mistakes an aircraft with one that is similar in appearance based on its WEFT features, the tutor displays both aircraft side by side so that the student may see directly the differences that distinguish the two aircraft, and learn how not to confuse them. The tutor then points out to the student the specific WEFT features of the two aircraft that differ. In addition, the student is given the ability to interrupt the tutor at any time and request that a specific aircraft be shown and described. In this way, the student can perform a comparison of any aircraft defined in the system. This ability to actively interact with the tutor is important to the teaching strategy. The student has a high degree of control over the instruction, if this is desired. On the other hand, the tutor is capable of independently presenting the entire lesson to the student.

4. Controlled Image Training

As each aircraft is presented during a teaching session, the individual components of aircraft are presented as separate recognition features for study. The composite of these features are then learned in order to identify a particular aircraft. During a review session, if the student mistakes the aircraft presented for one that does not resemble it, the tutor will ask the student to identify the separate recognition features that distinguish that view of the aircraft, thus reinforcing controlled image training.

5. Segregation

The tutor allows students to be trained according to their ability. Since the tutor first diagnoses a student's ability, and places him at a level in the tutor appropriate to that ability, new students and students needing detailed instruction are taught at a different level than more advanced students who may only need review and testing. In addition, since the tutor is designed to provide individual instruction, and maintains a student model for each student, the system tailors the training to each individual.

6. Repetition

Since the tutor operates on a personal computer, students may study at their own convenience, as often as necessary or desired, and in periods that suit the individual. Also, the game mode of the tutor was designed to encourage frequent use of the program.

B. AS AN INTELLIGENT TUTOR

1. Modeling of Knowledge and Reasoning

Because the ***Aircraft Recognition Tutor*** teaches a very specific knowledge domain and relies on a well developed model for reasoning about that knowledge, and because the knowledge is represented in object oriented style, both the knowledge and ability to reason about it are encapsulated in a concise, coherent manner. Not only does this allow the tutor to reason about the knowledge (eg. the tutor can compare two aircraft and decide whether they "appear" similar enough to expect confusion by the student), but the modularity of the knowledge base makes additions or changes to it simple and straight forward.

2. Communication

Although the ***Aircraft Recognition Tutor*** takes full advantage of the hardware capability that it was implemented for, this remains one of the major weaknesses of the system. Since the tutor was constrained to operate on existing U.S. Army computer hardware, and this hardware has limited capabilities in terms of graphics resolution and interface support, the possibilities for improvement of the interface are tremendous. The addition of mouse support would be a major improvement, allowing a much more natural method of selecting from the menus. Using higher resolution images of the aircraft would give the student a more realistic impression of the aircraft. It should be noted, however, that the limitations in this area were known in advance of the development of the system and are based solely on the hardware constraints of the computers it was designed for. Additional interface support in the software would not be difficult to provide.[3]

3. Cognitive Processing

Students are modeled by not only what they do not seem to understand, but also by that which they have not yet demonstrated any knowledge. This is possible because of the well defined knowledge domain taught by the tutor. Since the number of aircraft and associated features taught by the system is finite, we can start a student out with a default model based on their performance during the diagnostic phase, and then add and delete from this model as we learn more specifically what the student does and does not know.

4. Tutoring

The instructional style used by the system varies according to the current level and mode of the student.[4] A student progresses through these levels and modes, and based on the student's performance is allowed to advance to higher levels and modes, remain at a current level and mode, or revert to a previous level or mode.

[3]See pages 44-45 for a description of the types of improvement that could be made to the interface.

[4]See pages 30-32.

VII. CONCLUSION

A. ACCOMPLISHMENTS

Over a period of six months, we have designed, developed, tested, and evaluated the ***Aircraft Recognition Tutor***, an intelligent tutor system for visual aircraft recognition. The system was developed using Object Oriented programming techniques, and therefore is modular and easy to maintain. We have demonstrated that, using existing technology in Computer Science and Artificial Intelligence, a useful training system could be developed quickly and inexpensively for use on existing computer hardware in the field. We have faithfully captured the WEFT cognitive model for visual aircraft recognition instruction within a friendly medium. By distinguishing between what the student has not demonstrated a knowledge of and what he has demonstrated a lack of knowledge about, we have developed a student model that, unlike many others, accurately represents the current knowledge of the student The system is effective both as a formal training system and as an informal "game" device. Acceptance of the system among soldiers has been outstanding. Since the U.S. Army has deployed Zenith desktop and laptop personal computers at the battalion level, and in some cases at the company or battery level, failure to take full advantage of this equipment for training purposes would be unforgivable. The possibilities for the development of training applications is limited only by the imagination.

B. FUTURE IMPROVEMENT AND MODIFICATIONS

Based on the recommendations from the field evaluation, several modifications could be implemented to improve the ***Aircraft Recognition Tutor***.

1. Improvements to the Interface

Of most obvious need of improvement is the graphics resolution of the system. Currently the tutor uses the CGA 640x200 two color mode. EGA graphics offer the capability for a resolution of 640x350 pixels in 16 colors. This mode would allow a much more attractive interface, but the aircraft images, although better, would still need to be line drawings in order to show details clearly. VGA and Super VGA graphics modes (640x480 and 800x600 pixels, respectively, each with at least 16 colors) would allow the images used by the tutor to be scanned from actual photographs of the aircraft. This would be the ideal resolution for images that are scanned into and stored by the program as bitmaps. However, other, perhaps better options exist, given a higher screen resolution. Next in desirability would be to use an interactive digital video system to present actual filmed images of the aircraft. The video system would be controlled by the tutor system, and would replace the scanned images currently used. This would have the added benefit of greatly reducing the amount of disk space required by the tutor. Finally, the aircraft could be represented as three dimensional models, and manipulated through homogeneous transformations. This would allow the aircraft to be scaled, translated, and rotated, and then displayed in any of an infinite number of aspects, and would also require much less disk space than the images currently used. Combined with this could be a terrain and cloud database that allow the aircraft to be displayed superimposed or overlayed with a more realistic environment. This would provide the ability to partially obscure the aircraft, something that is likely to occur in reality.

Another major improvement would be the integration of mouse support with the interface. In fact, this is a very simple improvement that has been made to one version of the tutor already. Unfortunately, since the hardware that the tutor was designed for does not include a mouse, this feature was left out of the final product. Thought was also given to using a speech recognition system as the primary interface between the student and the tutor, but the rarity of these systems in the field, as well as the lack of compatibility among them, prevented this from being reasonable.

2. Improvements to the Tutoring Strategy

Additional characteristics could be tracked by the student model, in order to provide a more accurate understanding of the student's knowledge of the domain. For example, once the student is in the intermediate mode, the tutor assumes that the student has mastered the individual WEFT characteristics, and no longer tracks the student's performance in this area. However, since the student is sometimes asked to identify the specific WEFT features of an aircraft that he was unable to identify correctly, the ability and perhaps the need to track this information exists. Currently, if a student reverts to a previous level or mode, he is assigned the default model for that level or mode. By tracking characteristics of all of the levels and modes at all times, the tutor could better teach the student at any level or mode.

Additional teaching styles could be implemented. This would allow the tutor to teach a student in a different manner if it appears that the student is having trouble with the current teaching style. The tutor currently teaches in a different style for each level and mode, but within a particular level and mode, only one teaching strategy is used. If the student fails to respond to this strategy, the tutor will revert them to a previous level or mode for additional preparation before returning to the current level or mode.

C. POSSIBLE ADAPTATIONS

Several training subjects are directly comparable to VACR and therefore represent an opportunity for adaptation of the ***Aircraft Recognition Tutor***. Helicopter recognition is taught in the same manner as fixed-wing aircraft recognition, and the tutor could be modified easily to teach this subject by defining the additional WEFT characteristics common among helicopters and defining the helicopter objects. This could be done by adding onto the existing knowledge domain, or by substituting directly a new knowledge domain for helicopters. Armored vehicle recognition and ship/submarine recognition represent other subjects that the tutor could teach with an adjustment of the tutorial strategy and defining the appropriate knowledge domain objects.

APPENDIX A - CODE

```
{
Aircraft Recognition Tutor
program by Larry W. Campbell
1990

No warranties whatsoever are provided with this program.  Use at own risk.

This program may be used or modified under the following conditions:
   1. Any modified program will include the author's name in the program.
   2. A copy of the modified program will be provided to the author.

Send comments, suggestions, bug reports, or modifications to:
   CPT Larry W. Campbell
   SMC 2269
   Naval Postgraduate School
   Monterey, CA  93940

   E-Mail at campbell@cs.nps.navy.mil
}

program ART;

uses CRT, Graph, Screens, Dialogs, Menus, Tutor, Game, Utility, Student, Help;

type
  name = string[20];

var
  TempScreen, BinoScreen : Screen;
  HelpItem : HelpScreen;
  MainMenu : Menu;
  HelpMenu : Menu;
  GraphDriver, GraphMode, ErrorCode, ChoiceNum : Integer;
  Choice : name;
  Ch : char;
  F : text;
  FileName : string;

procedure HallofFame;
  var FameName : name;
  X,Y : integer;
begin
```

```
   TempScreen.Init('hallfame.scr');
   TempScreen.Show(0,0);
   Assign(F,'HallFame.rec');
   Reset(F);
   SetColor(1);
   X := 50;
   Y := 30;
   while (not eof(F)) and (X < 600) do
      begin
         Readln(F,FameName);
         OutTextXY(X,Y,FameName);
         Y := Y + 10;
         if Y = 200 then
            begin
               X := X + 200;
               Y := 30;
            end;
      end;
   Ch := ReadKey;
   SetColor(0);
   ClearDevice;
   TempScreen.Kill;
   Close(F);
end;

procedure RunTutor;
begin {RunTutor}
   BinoScreen.Show(0,0);
   TutorSession;
   BinoScreen.Hide;
end; {RunTutor}

procedure RunGame;
begin {RunGame}
   BinoScreen.Show(0,0);
   PlayGame;
   BinoScreen.Hide;
end; {RunGame}

procedure GetHelp;
begin {GetHelp}
   Choice := HelpMenu.GetChoice;
   while (Choice <> 'EXIT HELP!') and (Choice <> 'null') do
      begin
         if Choice = 'ABOUT HELP!' then
            HelpItem.Init('help.hlp')
         else if Choice = 'TUTOR HELP!' then
            HelpItem.Init('tutor.hlp')
            else if Choice = 'GAME HELP!' then
               HelpItem.Init('game.hlp')
            else if Choice = 'SETUP/UTILITY HELP!' then
               HelpItem.Init('setup.hlp');
            HelpItem.Show(0,0);
            Ch := ReadKey;
            HelpItem.Hide;
            HelpItem.Kill;
            GotoXY(1,1);
            Choice := HelpMenu.GetChoice;
         end;
   end; {GetHelp}

   begin {Main Program}
      {Initialize Graphics Adapter to CGA 640x200 2-color mode}
      GraphDriver := CGA;
      GraphMode := CGAHi;
      InitGraph(GraphDriver, GraphMode, '');
      SetBkColor(Green);
      {Load the graphics and data into memory}
      TempScreen.Init('initial.scr');
      BinoScreen.Init('main.scr');
      MainMenu.Init('Main.mnu');
      HelpMenu.Init('Help.mnu');
      {Display the Initial Title Screen}
      TempScreen.Show(0,0);
      Ch := ReadKey;
      TempScreen.Hide;
      TempScreen.Kill;
      HallofFame;
      {Display the Initial Menu Screen and Get a Response}
      StudentModel.Mode := '';
      Choice := MainMenu.GetChoice;
      while (Choice <> 'EXIT') and (Choice <> 'null') do
         begin
            if Choice = 'TUTOR SESSION' then RunTutor
            else if Choice = 'GAME' then RunGame
            else if Choice = 'HELP!' then GetHelp
            else if Choice = 'SETUP' then SetUp;
            GotoXY(1,1);
            StudentModel.Mode := '';
            Choice := MainMenu.GetChoice;
         end;
      {That's all folks}
      BinoScreen.Kill;
      CloseGraph;
   end. {Main Program}
```

```
unit Screens;

interface

uses Graph;

type
    Screen = object
       X,Y : integer;
       IsVisible : boolean;
       F : file;
       MemSize : word;
       P : pointer;
       constructor Init(FileName : string);
       procedure Show(XLoc, YLoc : integer);
       procedure Hide;
       destructor Kill;
   end;

implementation

constructor Screen.Init(FileName : string);
begin
   IsVisible := false;
   Assign(F, FileName);{Prepare the file}
   Reset(F, 1);{for a read operation.}
   MemSize := FileSize(F);{Determine memory needed}
   GetMem(P, MemSize);{and allocate the memory on the heap.}
   BlockRead(F, P^, MemSize);{Read in the graphic pic file}
   Close(F);{and close the file.}
end;

procedure Screen.Show(XLoc, YLoc : integer);
begin
  if not IsVisible then
     begin
        X := XLoc;
        Y := YLoc;
        PutImage(X,Y,P^,CopyPut);{Draw the graphics on the screen.}
        IsVisible := true;
     end;
end;

procedure Screen.Hide;
begin
  if IsVisible then
     begin
         PutImage(X,Y,P^,XorPut);{Turn all pixels off.}
```

```
            IsVisible := false;
        end;
    end;

    destructor Screen.Kill;
    begin
        FreeMem(P, MemSize);{Free the heap memory.}
    end;

    end.
```

```
unit Aircfts;

interface

uses Graph,PCX_TP,Screens;

type
  name = string[20];

  ACData = record
    AircraftName : name;
    ExampleOf : name;
    ExampleInfo : name;
    Wings : array [1..4] of name;
    WingsInfo : array [1..4] of name;
    Engine : array [1..2] of name;
    EngineInfo : array [1..2] of name;
    Fuslag : array [1..4] of name;
    FuslagInfo : array [1..4] of name;
    Tail : array [1..5] of name;
    TailInfo : array [1..5] of name;
  end;

   Aircraft = object(Screen)
    vptr : longint;
    F1 : text;
    ACInfo : ACData;
    constructor Init(ACName : name);
    procedure Show(XLoc,YLoc : integer);
    procedure Hide;
    procedure Kill;
  end;
var
  retcode : integer;

implementation

constructor Aircraft.Init(ACName : name);
var
  Counter : integer;
begin
  IsVisible := false;
  retcode := pcxCreateVirtual(pcxCMM,@vptr,pcxCGA_6,250,110);
  if (retcode = pcxSuccess) then
    retcode := pcxFileVirtual(Concat(ACName,'.pcx'),vptr);
  Assign(F1,Concat(ACName,'.dat'));
  Reset(F1);
  Readln(F1,ACInfo.AircraftName);
  Readln(F1,ACInfo.ExampleOf);
  Readln(F1,ACInfo.ExampleInfo);
  for Counter := 1 to 4 do
    begin
      Readln(F1,ACInfo.Wings[Counter]);
      Readln(F1,ACInfo.WingsInfo[Counter]);
    end;
  for Counter := 1 to 2 do
    begin
      Readln(F1,ACInfo.Engine[Counter]);
      Readln(F1,ACInfo.EngineInfo[Counter]);
    end;
  for Counter := 1 to 4 do
    begin
      Readln(F1,ACInfo.Fuslag[Counter]);
      Readln(F1,ACInfo.FuslagInfo[Counter]);
    end;
  for Counter := 1 to 5 do
    begin
      Readln(F1,ACInfo.Tail[Counter]);
      Readln(F1,ACInfo.TailInfo[Counter]);
    end;
  Close(F1);
end;

procedure Aircraft.Show(XLoc, YLoc : integer);
begin
  if not IsVisible then
    begin
      retcode := pcxSetDisplay(pcxCGA_6);
      X := XLoc;
      Y := YLoc;
      retcode := PCXPutImage(vptr,pcxXOR,X,Y,0);
      IsVisible := true;
    end;
end;

procedure Aircraft.Hide;
begin
  if IsVisible then
    begin
      retcode := pcxSetDisplay(pcxCGA_6);
      retcode := PCXPutImage(vptr,pcxXOR,X,Y,0);
      IsVisible := false;
    end;
end;

procedure Aircraft.Kill;
```

```
begin
  IsVisible := false;
  retcode := pcxDestroyVirtual(vptr);
end;
end.
```

```
unit Dialogs;

interface

uses Graph, Screens;

type
   Dialog = object(Screen)
      OldP : pointer;
      procedure Show(XLoc, YLoc : integer);
      procedure Hide;
   end;

implementation

procedure Dialog.Show(XLoc,YLoc : integer);
begin
   if not IsVisible then
      begin
          X := XLoc;
          Y := YLoc;
          GetMem(OldP, MemSize);{Save the old bitmap}
          GetImage(X,Y,X+200,Y+50,OldP^);
          PutImage(X,Y,P^,CopyPut);{and draw the new bitmap.}
          IsVisible := true;
      end;
end;

procedure Dialog.Hide;
begin
   if IsVisible then
      begin
          PutImage(X,Y,OldP^,CopyPut);{Put the old bitmap back}
          FreeMem(OldP, MemSize);{and free the heap memory.}
          IsVisible := false;
      end;
end;

end.
```

```
unit Menus;

interface

uses CRT, Graph, Help;

type
  name = string[20];

  Menu = object
     MenuTitle : string;
     X,Y,Color,TopItem,SelectedItem,HighLightedItem : integer;
     NumSelections : integer;
     MenuSelection : array[1..150] of name;
     F : text;
     constructor Init(FileName : name);
     function GetChoice : name;
  end;

var
  LastSelection : integer;

implementation

const
   MaxSelections = 10;

var
  Ch : char;
  Counter : integer;

constructor Menu.Init(FileName : name);
begin
   HighlightedItem := 1;
   SelectedItem := 1;
   TopItem := 1;
   Counter := 0;
   Assign(F,FileName);
   Reset(F);
   Readln(F,MenuTitle);
   Readln(F,X);
   Readln(F,Y);
   Readln(F,Color);
   while (not eof(F)) and (Counter < 150)do
     begin
       Counter := Counter + 1;
       Readln(F,MenuSelection[Counter]);
     end;
   NumSelections := Counter;
   Close(F);
end;

function Menu.GetChoice : name;

      procedure ShowMenu(Selection : integer);
      begin
         SetColor(Color);
         SetLineStyle(SolidLn,0,ThickWidth);
         SetWriteMode(CopyPut);
         SetTextJustify(CenterText,CenterText);
         OutTextXY(X,Y,MenuTitle);
         Counter := 0;
         while (Counter + Selection <= NumSelections) and
         (Counter + Selection < Selection + MaxSelections) do
             begin
                OutTextXY(X,Counter*8+Y+15,MenuSelection[Counter + Selection]);
                Counter := Counter + 1;
             end;
         if (Selection > 1) and (Selection <= NumSelections - 10) then
            OutTextXY(X,Y+100,'PgUp/PgDn')
         else if Selection > 1 then
            OutTextXY(X,Y+100,'PgUp')
         else if Selection <= NumSelections - 10 then
            OutTextXY(X,Y+100,'PgDn');
      end;

      procedure KillMenu;
      begin
         if Color <> 0 then
            SetColor(0);
         SetWriteMode(CopyPut);
         Line(X-50,Y-3,X+50,Y-3);
         Line(X-50,Y,X+50,Y);
         Line(X-50,Y+3,X+50,Y+3);
         for Counter := 1 to 10 do
            begin
               Line(X-85,Counter*8+Y+3,X+85,Counter*8+Y+3);
               Line(X-85,Counter*8+Y+6,X+85,Counter*8+Y+6);
               Line(X-85,Counter*8+Y+9,X+85,Counter*8+Y+9);
            end;
         Line(X-35,Y+97,X+35,Y+97);
         Line(X-35,Y+100,X+35,Y+100);
         Line(X-35,Y+103,X+35,Y+103);
      end;

      procedure Highlight(NewItem : integer);
```

```
begin
  SetWriteMode(XorPut);
  SetColor(1);
  Line(X-85,HighlightedItem*8+Y+3,X+85,HighlightedItem*8+Y+3);
  Line(X-85,HighlightedItem*8+Y+6,X+85,HighlightedItem*8+Y+6);
  Line(X-85,HighlightedItem*8+Y+9,X+85,HighlightedItem*8+Y+9);
  if HighLightedItem <> NewItem then
    begin
      HighlightedItem := NewItem;
      Line(X-85,NewItem*8+Y+3,X+85,NewItem*8+Y+3);
      Line(X-85,NewItem*8+Y+6,X+85,NewItem*8+Y+6);
      Line(X-85,NewItem*8+Y+9,X+85,NewItem*8+Y+9);
    end;
end;

procedure PageUp;
begin
  if TopItem > MaxSelections then
    begin
      KillMenu;
      SelectedItem := (SelectedItem - 10) -
                (HighlightedItem - 1);
      TopItem := TopItem - 10;
      HighlightedItem := 1;
      ShowMenu(TopItem);
      Highlight(HighlightedItem);
    end;
end;

procedure PageDown;
begin
  if TopItem + MaxSelections <= NumSelections then
    begin
      KillMenu;
      SelectedItem := (SelectedItem + 10) -
                (HighlightedItem - 1);
      TopItem := TopItem + 10;
      HighlightedItem := 1;
      ShowMenu(TopItem);
      Highlight(HighlightedItem);
    end;
end;

procedure MoveUp;
begin
  if HighlightedItem > 1 then
    begin
      Highlight(HighlightedItem - 1);
```

```
                SelectedItem := SelectedItem - 1;
            end;
        end;

        procedure MoveDown;
        begin
            if (HighlightedItem < MaxSelections)
                and (SelectedItem < NumSelections) then
                    begin
                        Highlight(HighlightedItem + 1);
                        SelectedItem := SelectedItem + 1;
                    end;
        end;

        procedure GetInput;
        begin
            Ch := ReadKey;
            Case Ch of
                'h' : GetHelp;
                chr(80),chr(50) : MoveDown;
                chr(72),chr(56) : MoveUp;
                chr(81),chr(51) : PageDown;
                chr(73),chr(57) : PageUp;
            end;
        end;

begin
    ShowMenu(1);
    Highlight(1);
    repeat
        GetInput
    until (Ch = #13) or (Ch = #27);
    if Ch = #27 then
        GetChoice := 'null'
    else
        GetChoice := MenuSelection[SelectedItem];
    LastSelection := SelectedItem;
    KillMenu;
    TopItem := 1;
    SelectedItem := 1;
    HighlightedItem := 1;
end;

end.
```

```
unit Student;

interface
type
  name = string [20];

  Model = object
    StudentName, Mode, Level : name;
    TestScore : integer;
    NumShown, NumMissed : integer;
    ACArray : array [1..150] of name;
    MissedArray : array [1..150] of name;
    function Get : boolean;
   procedure Update(StuName : name;NewMode : name; NewLevel : name; NewScore : integer);
    procedure Save;
    function GetEntry(MaxNum : integer) : integer;
    function AddEntry(ACName : name; MaxNum :integer) : boolean;
    procedure Kill;
  end;

var
  StudentModel : Model;


implementation

uses DOS, CRT, Graph, Dialogs;

var
  FileName : name;
  Deleted : boolean;
  F : text;
  S : pathstr;
  Ch : char;
  Counter : integer;
  DialogScreen : Dialog;

function Model.GetEntry(MaxNum : integer) : integer;
begin
  Randomize;
  Counter := 1;
  while (StudentModel.ACArray[Counter] = '') and (Counter <= MaxNum) do
    Counter := Counter + 1;
  if Counter < MaxNum then
    begin
      Counter := Random(MaxNum - 1);
      while StudentModel.ACArray[Counter + 1] = '' do
        Counter := Random(MaxNum);
        GetEntry := Counter + 1;
      end
    else GetEntry := 0;
end;

function Model.AddEntry(ACName : name; MaxNum : integer) : boolean;
begin
   Counter := 1;
   while (StudentModel.MissedArray[Counter] <> '') and (Counter <= MaxNum+1) do
      Counter := Counter + 1;
   if Counter <= MaxNum then
      begin
         StudentModel.MissedArray[Counter] := ACName;
         AddEntry := true;
      end
   else AddEntry := false;
end;

function Model.Get : boolean;
const
   ALPHA  = ['A'..'Z','a'..'z'];
   NUM  = ['0'..'9'];
begin {GetStudentModel}
   Deleted := false;
   FileName := '';
   Counter := 1;
   DialogScreen.Init('Model.dlg');
   DialogScreen.Show(220,25);
   while Counter <= 6 do
      begin
         Ch := ReadKey;
         if (Counter = 1) and (Ch in ALPHA) then
            begin
               OutTextXY(290+10*Counter,68,Ch);
               FileName := Concat(FileName,Ch);
               Counter := Counter + 1;
            end
         else if (Counter > 1) and (Counter < 6) and (Ch in NUM) then
            begin
               OutTextXY(290+10*Counter,68,Ch);
               FileName := Concat(FileName,Ch);
               Counter := Counter + 1;
            end
         else if (Ch = #8) and (Counter > 1) then
            begin
               SetColor(1);
               Line(275+10*Counter,64,285+10*Counter,64);
               Line(275+10*Counter,67,285+10*Counter,67);
```

```
              Line(275+10*Counter,70,285+10*Counter,70);
              Counter := Counter - 1;
              FileName := Copy(FileName,1,Counter-1);
              SetColor(0);
            end
          else if (Counter = 6) and (Ch = #13) then
            Counter := Counter + 1
          else
            begin
              Sound(440);
              Delay(100);
              NoSound;
            end;
      end;
    DialogScreen.Hide;
    DialogScreen.Kill;
    S := FSearch('.',Concat(FileName,'.mdl'));
    if S = '' then
      begin
        Get := false;
        Exit;
      end
    else
      begin
        Assign(F,Concat(FileName,'.mdl'));
        Reset(F);
        Readln(F,StudentModel.StudentName);
        Readln(F,StudentModel.Mode);
        Readln(F,StudentModel.Level);
        Readln(F,StudentModel.TestScore);
        Readln(F,StudentModel.NumShown);
        Readln(F,StudentModel.NumMissed);
        for Counter := 1 to 150 do
          Readln(F,StudentModel.ACArray[Counter]);
        for Counter := 1 to 150 do
          Readln(F,StudentModel.MissedArray[Counter]);
        Close(F);
      end;
    Get := true;
  end; {GetStudentModel}

  procedure Model.Update(StuName : name; NewMode : name; NewLevel : name; NewScore :
  integer);
  begin {Update}
    if NewLevel = 'Novice' then
      Assign(F,'Novice.def')
    else
      Assign(F,'Intermed.def');
  Reset(F);
  Readln(F,StudentModel.StudentName);
  Readln(F,StudentModel.Mode);
  Readln(F,StudentModel.Level);
  Readln(F,StudentModel.TestScore);
  Readln(F,StudentModel.NumShown);
  Readln(F,StudentModel.NumMissed);
  for Counter := 1 to 150 do
    Readln(F,StudentModel.ACArray[Counter]);
  for Counter := 1 to 150 do
    Readln(F,StudentModel.MissedArray[Counter]);
  Close(F);
  StudentModel.StudentName := StuName;
  StudentModel.Mode := NewMode;
  StudentModel.Level := NewLevel;
  StudentModel.TestScore := NewScore;
end; {Update}

procedure Model.Save;
begin
  if not Deleted then
    begin
      Assign(F,Concat(FileName,'.mdl'));
      Rewrite(F);
      Writeln(F,StudentModel.StudentName);
      Writeln(F,StudentModel.Mode);
      Writeln(F,StudentModel.Level);
      Writeln(F,StudentModel.TestScore);
      Writeln(F,StudentModel.NumShown);
      Writeln(F,StudentModel.NumMissed);
      for Counter := 1 to 150 do
        Writeln(F,StudentModel.ACArray[Counter]);
      for Counter := 1 to 150 do
        Writeln(F,StudentModel.MissedArray[Counter]);
      Close(F);
    end;
end;

procedure Model.Kill;
begin
  Exec('\COMMAND.COM',Concat('/C del ',FileName,'.mdl'));
  Deleted := true;
end;

end.
```

```
unit Tutor;

interface
  procedure TutorSession;

implementation

uses CRT, Graph, Student, Aircfts, Dialogs, Menus, Help;

type
  name = string[20];

var
  Counter1, Counter2, Counter3, MaxNum, ChoiceNum, Score : integer;
  Comparison : real;
  Ch : char;
  LeftAC, RightAC : Aircraft;
  DialogScreen : array [1..5] of Dialog;
  WEFTMenu : Menu;
  StuName, Choice, FourthCh : name;
  CorrectAnswer, CloseAnswer, Done : boolean;

procedure ShowFeature(Feature : name);
begin {ShowFeature}
   if Feature <> '' then
      begin
         DialogScreen[4].Init(Concat(Feature,'.dlg'));
         DialogScreen[4].Show(10,30);
         Ch := ReadKey;
         while Ch = 'h' do
            begin
               GetHelp;
               Ch := ReadKey;
            end;
         DialogScreen[4].Hide;
         DialogScreen[4].Kill;
      end;
end; {ShowFeature}

function CompareAircraft : real;
begin {CompareAircraft}
   Comparison := 0;
   Counter3 := 0;
   for Counter2 := 1 to 4 do
      if (LeftAC.ACInfo.Wings[Ccunter2] <> '') and
        (RightAC.ACInfo.Wings[Counter2] <> '') then
          if LeftAC.ACInfo.Wings[Counter2] = RightAC.ACInfo.Wings[Counter2] then
             begin
```

```
                Comparison := Comparison + 1;
                Inc(Counter3);
              end
            else Inc(Counter3);
      for Counter2 := 1 to 2 do
        if (LeftAC.ACInfo.Engine[Counter2] <> '') and
          (RightAC.ACInfo.Engine[Counter2] <> '') then
            if LeftAC.ACInfo.Engine[Counter2] = RightAC.ACInfo.Engine[Counter2] then
              begin
                Comparison := Comparison + 1;
                Inc(Counter3);
              end
            else Inc(Counter3);
      for Counter2 := 1 to 4 do
        if (LeftAC.ACInfo.Fuslag[Counter2] <> '') and
          (RightAC.ACInfo.Fuslag[Counter2] <> '') then
            if LeftAC.ACInfo.Fuslag[Counter2] = RightAC.ACInfo.Fuslag[Counter2] then
              begin
                Comparison := Comparison + 1;
                Inc(Counter3);
              end
            else Inc(Counter3);
      for Counter2 := 1 to 5 do
        if (LeftAC.ACInfo.Tail[Counter2] <> '') and
          (RightAC.ACInfo.Tail[Counter2] <> '') then
            if LeftAC.ACInfo.Tail[Counter2] = RightAC.ACInfo.Tail[Counter2] then
              begin
                Comparison := Comparison + 1;
                Inc(Counter3);
              end
            else Inc(Counter3);
      CompareAircraft := Comparison/Counter3;
    end; {CompareAircraft}

    procedure Diagnose;
    const
      ALPHA = ['A'..'Z','a'..'z'];
    begin
      StudentModel.Mode := 'Diagnose';
      StuName := '';
      Counter1 := 1;
      SetColor(1);
      DialogScreen[1].Init('GetName.dlg');
      DialogScreen[1].Show(220,25);
      Line(220,64,420,64);
      Line(220,67,420,67);
      Line(220,70,420,70);
      while (Ch <> #13) and (Counter1 < 21) do
```

```
  begin
    Ch := ReadKey;
    if Ch = chr(32) then
       begin
         StuName := Concat(StuName,chr(32));
         Counter1 := Counter1 + 1;
       end;
    if Ch in ALPHA then
       begin
         SetColor(0);
         OutTextXY(225+10*Counter1,68,Ch);
         StuName := Concat(StuName,Ch);
         Counter1 := Counter1 + 1;
         SetColor(1);
       end;
    if (Ch = #8) and (Counter1 > 1) then
      begin
         SetColor(1);
         Line(210+10*Counter1,64,220+10*Counter1,64);
         Line(210+10*Counter1,67,220+10*Counter1,67);
         Line(210+10*Counter1,70,220+10*Counter1,70);
         Counter1 := Counter1 - 1;
         StuName := Copy(StuName,1,Counter1-1);
         SetColor(0);
      end;
  end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
SetColor(0);
StudentModel.Update(StuName,'Teach','Novice',0);
StudentModel.Mode := 'Diagnose';
MaxNum := 75;
Counter3 := 0;
DialogScreen[1].Init('Welcome.dlg');
DialogScreen[1].Show(220,25);
Ch := ReadKey;
while Ch = 'h' do
   begin
     GetHelp;
     Ch := ReadKey;
   end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
DialogScreen[1].Init('Diagnose.dlg');
DialogScreen[1].Show(220,25);
Ch := ReadKey;
while Ch = 'h' do
```

```
          begin
             GetHelp;
             Ch := ReadKey;
          end;
       DialogScreen[1].Hide;
       DialogScreen[1].Kill;
       if Ch = #27 then Exit;
       for Counter2 := 1 to 10 do
          begin
             Counter1 := StudentModel.GetEntry(MaxNum);
             if StudentModel.ACArray[Counter1] <> '' then
                begin
                   LeftAC.Init(StudentModel.ACArray[Counter1]);
                   if LeftAC.ACInfo.ExampleInfo <> '' then
                      begin
                         LeftAC.Show(25,72);
                         WEFTMenu.Init(Concat(Copy(LeftAC.ACInfo.ExampleInfo,1,5),'.mnu'));
                         Choice := WEFTMenu.GetChoice;
                         if Choice = 'null' then
                            begin
                               LeftAC.Hide;
                               LeftAC.Kill;
                               Counter3 := Counter3 + 1;
                               Exit;
                            end;
                         if Choice <> LeftAC.ACInfo.ExampleOf then
                            begin
                               Counter3 := Counter3 + 1;
                               Sound(100);
                               Delay(200);
                               NoSound;
                            end;
                         LeftAC.Hide;
                      end;
                   LeftAC.Kill;
                end;
             GoToXY(1,1);
          end;
       if Counter3 <= 1 then
          StudentModel.Update(StuName,'Teach','Intermediate',0)
       else StudentModel.Update(StuName,'Teach','Novice',0);
    end;

    procedure Teach;
    begin {Teach}
       Ch := #13;
       if StudentModel.Level = 'Novice' then
          MaxNum := 75
```

```
else MaxNum := 150;
if StudentModel.Level = 'Novice' then
  DialogScreen[1].Init('TeaNov.dlg')
else DialogScreen[1].Init('TeaInt.dlg');
DialogScreen[1].Show(220,25);
Ch := ReadKey;
while Ch = 'h' do
   begin
      GetHelp;
      Ch := ReadKey;
   end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
for Counter1 := StudentModel.NumShown to MaxNum do
   if StudentModel.Level = 'Novice' then
      begin
         if StudentModel.ACArray[Counter1] <> '' then
            begin
               LeftAC.Init(StudentModel.ACArray[Counter1]);
               if LeftAC.ACInfo.ExampleInfo <> '' then
                  begin
                     LeftAC.Show(25,72);
                     ShowFeature(LeftAC.ACInfo.ExampleInfo);
                     LeftAC.Hide;
                  end;
               LeftAC.Kill;
               if Ch = #27 then Exit;
               Inc(StudentModel.NumShown);
            end;
      end
   else
      begin
         if StudentModel.ACArray[Counter1] <> '' then
            begin
               LeftAC.Init(StudentModel.ACArray[Counter1]);
               LeftAC.Show(25,72);
               DialogScreen[2].Init(Concat(Copy(StudentModel.ACArray[Counter1],1,4).'.nam'));
               DialogScreen[2].Show(50,179);
               for Counter2 := 1 to 4 do
                  if Ch <> #27 then
                     ShowFeature(LeftAC.ACInfo.WingsInfo[Counter2]);
               for Counter2 := 1 to 2 do
                  if Ch <> #27 then
                     ShowFeature(LeftAC.ACInfo.EngineInfo[Counter2]);
               for Counter2 := 1 to 4 do
                  if Ch <> #27 then
                     ShowFeature(LeftAC.ACInfo.FuslagInfo[Counter2]);
```

```
                for Counter2 := 1 to 5 do
                  if Ch <> #27 then
                    ShowFeature(LeftAC.ACInfo.TailInfo[Counter2]);
                DialogScreen[2].Hide;
                LeftAC.Hide;
                DialogScreen[2].Kill;
                LeftAC.Kill;
                if Ch = #27 then Exit;
                Inc(StudentModel.NumShown);
              end;
          end;
        StudentModel.NumShown := MaxNum;
end; {Teach}

procedure ReviewNovice;
begin {ReviewNovice}
  MaxNum := 75;
  Done := false;
  DialogScreen[1].Init('Return.dlg');
  DialogScreen[1].Show(220,25);
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  if Ch = #27 then Exit;
  DialogScreen[1].Init('RevNov.dlg');
  DialogScreen[1].Show(220,25);
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  if Ch = #27 then Exit;
  Counter1 := StudentModel.GetEntry(MaxNum);
  while (Counter1 <> 0) and (Done = false) do
    begin
      LeftAC.Init(StudentModel.ACArray[Counter1]);
      if LeftAC.ACInfo.ExampleInfo <> '' then
        begin
          LeftAC.Show(25,72);
          WEFTMenu.Init(Concat(Copy(LeftAC.ACInfo.ExampleInfo,1,5),'.mnu'));
```

```
                Choice := WEFTMenu.GetChoice;
                  if Choice = LeftAC.ACInfo.ExampleOf then
                    begin
                      StudentModel.ACArray[Counter1] := '';
                      DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.dlg'));
                      DialogScreen[1].Show(220,25);
                      Ch := ReadKey;
                      while Ch = 'h' do
                        begin
                          GetHelp;
                          Ch := ReadKey;
                        end;
                      DialogScreen[1].Hide;
                      DialogScreen[1].Kill;
                    end
                  else if Choice <> 'null' then
                    begin
                     if StudentModel.AddEntry(StudentModel.ACArray[Counter1], MaxNum) = false
then
                        Done := true;
                      DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.dlg'));
                      DialogScreen[1].Show(220,25);
                      ShowFeature(LeftAC.ACInfo.ExampleInfo);
                      if Ch = 'h' then
                        GetHelp;
                      DialogScreen[1].Hide;
                      DialogScreen[1].Kill;
                    end;
                LeftAC.Hide;
              end
            else StudentModel.ACArray[Counter1] := '';
            LeftAC.Kill;
            if (Ch = #27) or (Choice = 'null') then Exit;
            Counter1 := StudentModel.GetEntry(MaxNum);
          end;
        Done := true;
end; {ReviewNovice}

procedure ReviewIntermediate;

procedure HandleCorrectIntermediate;
begin {HandleCorrectIntermediate}
  DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.dlg'));
  DialogScreen[1].Show(220,25);
  DialogScreen[2].Init(Concat(Copy(StudentModel.ACArray[Counter1],1,4),'.nam'));
  DialogScreen[2].Show(50,179);
  StudentModel.ACArray[Counter1] := '';
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[2].Hide;
  DialogScreen[2].Kill;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
end; {HandleCorrectIntermediate}

procedure HandleCloseIntermediate;
begin {HandleCloseIntermediate}
  RightAC.Show(365,72);
  DialogScreen[1].Init('Close.dlg');
  DialogScreen[1].Show(220,25);
  Ch := Readkey;
  if Ch = 'h' then
    GetHelp;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  if Ch = #27 then
    Exit;
  DialogScreen[1].Init('Compare.dlg');
  DialogScreen[1].Show(220,25);
  DialogScreen[2].Init(Concat(Copy(StudentModel.ACArray[Counter1],1,4),'.nam'));
  if Copy(Choice,4,1) = ' ' then
    DialogScreen[3].Init(Concat(Copy(Choice,1,3),'_','.nam'))
  else DialogScreen[3].Init(Concat(Copy(Choice,1,4),'.nam'));
  DialogScreen[2].Show(50,179);
  DialogScreen[3].Show(390,179);
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      GetHelp;
      Ch := ReadKey;
    end;
  for Counter2 := 1 to 4 do
    if (LeftAC.ACInfo.Wings[Counter2] <> '') and (Ch <> #27) and
      (RightAC.ACInfo.Wings[Counter2] <> '') then
        if LeftAC.ACInfo.Wings[Counter2] <> RightAC.ACInfo.Wings[Counter2] then
          begin
            DialogScreen[4].Init(Concat(LeftAC.ACInfo.WingsInfo[Counter2],'.dlg'));
            DialogScreen[5].Init(Concat(RightAC.ACInfo.WingsInfo[Counter2],'.dlg'));
            DialogScreen[4].Show(10,30);
            DialogScreen[5].Show(430,30);
            Ch := ReadKey;
            while Ch = 'h' do
```

```
                begin
                    GetHelp;
                    Ch := ReadKey;
                end;
            DialogScreen[5].Hide;
            DialogScreen[4].Hide;
            DialogScreen[5].Kill;
            DialogScreen[4].Kill;
        end;
for Counter2 := 1 to 2 do
    if (LeftAC.ACInfo.Engine[Counter2] <> '') and (Ch <> #27) and
      (RightAC.ACInfo.Engine[Counter2] <> '') then
        if LeftAC.ACInfo.Engine[Counter2] <> RightAC.ACInfo.Engine[Counter2] then
          begin
            DialogScreen[4].Init(Concat(LeftAC.ACInfo.EngineInfo[Counter2],'.dlg'));
            DialogScreen[5].Init(Concat(RightAC.ACInfo.EngineInfo[Counter2],'.dlg'));
            DialogScreen[4].Show(10,30);
            DialogScreen[5].Show(430,30);
            Ch := ReadKey;
            while Ch = 'h' do
                begin
                    GetHelp;
                    Ch := ReadKey;
                end;
            DialogScreen[5].Hide;
            DialogScreen[4].Hide;
            DialogScreen[5].Kill;
            DialogScreen[4].Kill;
        end;
for Counter2 := 1 to 4 do
    if (LeftAC.ACInfo.Fuslag[Counter2] <> '') and (Ch <> #27) and
      (RightAC.ACInfo.Fuslag[Counter2] <> '') then
        if LeftAC.ACInfo.Fuslag[Counter2] <> RightAC.ACInfo.Fuslag[Counter2] then
          begin
            DialogScreen[4].Init(Concat(LeftAC.ACInfo.FuslagInfo[Counter2],'.dlg'));
            DialogScreen[5].Init(Concat(RightAC.ACInfo.FuslagInfo[Counter2],'.dlg'));
            DialogScreen[4].Show(10,30);
            DialogScreen[5].Show(430,30);
            Ch := ReadKey;
            while Ch = 'h' do
                begin
                    GetHelp;
                    Ch := ReadKey;
                end;
            DialogScreen[5].Hide;
            DialogScreen[4].Hide;
            DialogScreen[5].Kill;
            DialogScreen[4].Kill;
            end;
    for Counter2 := 1 to 5 do
      if (LeftAC.ACInfo.Tail[Counter2] <> '') and (Ch <> #27) and
        (RightAC.ACInfo.Tail[Counter2] <> '') then
          if LeftAC.ACInfo.Tail[Counter2] <> RightAC.ACInfo.Tail[Counter2] then
            begin
              DialogScreen[4].Init(Concat(LeftAC.ACInfo.TailInfo[Counter2],'.dlg'));
              DialogScreen[5].Init(Concat(RightAC.ACInfo.TailInfo[Counter2],'.dlg'));
              DialogScreen[4].Show(10,30);
              DialogScreen[5].Show(430,30);
              Ch := ReadKey;
              while Ch = 'h' do
                begin
                  GetHelp;
                  Ch := ReadKey;
                end;
              DialogScreen[5].Hide;
              DialogScreen[4].Hide;
              DialogScreen[5].Kill;
              DialogScreen[4].Kill;
            end;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
    DialogScreen[2].Hide;
    DialogScreen[2].Kill;
    DialogScreen[3].Hide;
    DialogScreen[3].Kill;
    RightAC.Hide;
  end; {HandleCloseIntermediate}

  procedure HandleWrongIntermediate;
  begin {HandleWrongIntermediate}
    DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.dlg'));
    DialogScreen[1].Show(220,25);
    Ch := Readkey;
    if Ch = 'h' then
      GetHelp;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
    if Ch = #27 then Exit;
    DialogScreen[1].Init('IdWEFT.dlg');
    DialogScreen[1].Show(220,25);
    DialogScreen[2].Init(Concat(Copy(StudentModel.ACArray[Counter1],1,4),'.nam'));
    DialogScreen[2].Show(50,179);
    Ch := ReadKey;
    while Ch = 'h' do
      begin
        GetHelp;
      Ch := ReadKey;
    end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
for Counter2 := 1 to 4 do
  if (LeftAC.ACInfo.Wings[Counter2] <> '') and (Ch <> #27) then
    begin
      WEFTMenu.Init(Concat(Copy(LeftAC.ACInfo.WingsInfo[Counter2],1,5),'.mnu'));
      Choice := WEFTMenu.GetChoice;
      if Choice = LeftAC.ACInfo.Wings[Counter2] then
        begin
          DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.dlg'));
          DialogScreen[1].Show(220,25);
          Ch := ReadKey;
          while Ch = 'h' do
            begin
              GetHelp;
              Ch := ReadKey;
            end;
          DialogScreen[1].Hide;
          DialogScreen[1].Kill;
        end
      else if Choice <> 'null' then
        begin
          DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.dlg'));
          DialogScreen[1].Show(220,25);
          ShowFeature(LeftAC.ACInfo.WingsInfo[Counter2]);
          DialogScreen[1].Hide;
          DialogScreen[1].Kill;
        end
      else Ch := #27;
    end;
for Counter2 := 1 to 2 do
  if (LeftAC.ACInfo.Engine[Counter2] <> '') and (Ch <> #27) then
    begin
      WEFTMenu.Init(Concat(Copy(LeftAC.ACInfo.EngineInfo[Counter2],1,5),'.mnu'));
      Choice := WEFTMenu.GetChoice;
      if Choice = LeftAC.ACInfo.Engine[Counter2] then
        begin
          DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.dlg'));
          DialogScreen[1].Show(220,25);
          Ch := ReadKey;
          while Ch = 'h' do
            begin
              GetHelp;
              Ch := ReadKey;
            end;
          DialogScreen[1].Hide;
```

```
                DialogScreen[1].Kill;
             end
          else if Choice <> 'null' then
             begin
                DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.dlg'));
                DialogScreen[1].Show(220,25);
                ShowFeature(LeftAC.ACInfo.EngineInfo[Counter2]);
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
             end
          else Ch := #27;
       end;
 for Counter2 := 1 to 4 do
    if (LeftAC.ACInfo.Fuslag[Counter2] <> '') and (Ch <> #27) then
       begin
          WEFTMenu.Init(Concat(Copy(LeftAC.ACInfo.FuslagInfo[Counter2],1,5),'.mnu'));
          Choice := WEFTMenu.GetChoice;
          if Choice = LeftAC.ACInfo.Fuslag[Counter2] then
             begin
                DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.dlg'));
                DialogScreen[1].Show(220,25);
                Ch := ReadKey;
                while Ch = 'h' do
                   begin
                      GetHelp;
                      Ch := ReadKey;
                   end;
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
             end
          else if Choice <> 'null' then
             begin
                DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.dlg'));
                DialogScreen[1].Show(220,25);
                ShowFeature(LeftAC.ACInfo.FuslagInfo[Counter2]);
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
             end
          else Ch := #27;
       end;
 for Counter2 := 1 to 5 do
    if (LeftAC.ACInfo.Tail[Counter2] <> '') and (Ch <> #27) then
       begin
          WEFTMenu.Init(Concat(Copy(LeftAC.ACInfo.TailInfo[Counter2],1,5),'.mnu'));
          Choice := WEFTMenu.GetChoice;
          if Choice = LeftAC.ACInfo.Tail[Counter2] then
             begin
                DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.dlg'));
                DialogScreen[1].Show(220,25);
                Ch := ReadKey;
                while Ch = 'h' do
                   begin
                      GetHelp;
                      Ch := ReadKey;
                   end;
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
             end
          else if Choice <> 'null' then
             begin
                DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.dlg'));
                DialogScreen[1].Show(220,25);
                ShowFeature(LeftAC.ACInfo.TailInfo[Counter2]);
                DialogScreen[1].Hide;
                DialogScreen[1].Kill;
             end
          else Ch := #27;
          end;
   DialogScreen[2].Hide;
   DialogScreen[2].Kill;
end; {HandleWrongIntermediate}

begin {ReviewIntermediate}
   MaxNum := 150;
   Ch := #13;
   Done := false;
   DialogScreen[1].Init('Return.dlg');
   DialogScreen[1].Show(220,25);
   Ch := ReadKey;
   while Ch = 'h' do
      begin
         GetHelp;
         Ch := ReadKey;
      end;
   DialogScreen[1].Hide;
   DialogScreen[1].Kill;
   if Ch = #27 then Exit;
   DialogScreen[1].Init('RevInt.dlg');
   DialogScreen[1].Show(220,25);
   Ch := ReadKey;
   while Ch = 'h' do
      begin
         GetHelp;
         Ch := ReadKey;
      end;
   DialogScreen[1].Hide;
```

```
  DialogScreen[1].Kill;
  if Ch = #27 then Exit;
  Counter1 := StudentModel.GetEntry(MaxNum);
  while (Counter1 <> 0) and (Done = false) do
    begin
      LeftAC.Init(StudentModel.ACArray[Counter1]);
      WEFTMenu.Init('WEFT.mnu');
      CorrectAnswer := false;
      CloseAnswer := false;
      LeftAC.Show(25,72);
      Choice := WEFTMenu.GetChoice;
      if Choice = 'null' then
        begin
          LeftAC.Hide;
          LeftAC.Kill;
          Exit;
        end;
      if Choice = LeftAC.ACInfo.AircraftName then
        HandleCorrectIntermediate
      else
        begin
          if Copy(Choice,4,1) = ' ' then
        RightAC.Init(Concat(Copy(Choice,1,3),'_',Copy(StudentModel.ACArray[Counter1],5,3)))
      else RightAC.Init(Concat(Copy(Choice,1,4),Copy(StudentModel.ACArray[Counter1],5,3)));
          if CompareAircraft >= 0.7 then
            HandleCloseIntermediate
          else
            begin
              HandleWrongIntermediate;
              if StudentModel.AddEntry(StudentModel.ACArray[Counter1], MaxNum) = false
then
                Done := true;
            end;
          RightAC.Hide;
          RightAC.Kill;
        end;
      GoToXY(1,1);
      LeftAC.Hide;
      LeftAC.Kill;
      if Ch = #27 then Exit;
      Counter1 := StudentModel.GetEntry(MaxNum);
    end;
  Done := true;
end; {ReviewIntermediate}

procedure ReviewExpert;
var
  Feature : array [1..16] of name;
```

```
begin {ReviewExpert}
  MaxNum := 150;
  Ch := #13;
  Done := false;
  DialogScreen[1].Init('Return.dlg');
  DialogScreen[1].Show(220,25);
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  if Ch = #27 then Exit;
  DialogScreen[1].Init('RevExp.dlg');
  DialogScreen[1].Show(220,25);
  Ch := ReadKey;
  while Ch = 'h' do
    begin
      GetHelp;
      Ch := ReadKey;
    end;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  if Ch = #27 then Exit;
  Counter1 := StudentModel.GetEntry(MaxNum);
  while (Counter1 <> 0) and (Done = false) do
    begin
      Counter3 := 1;
      LeftAC.Init(StudentModel.ACArray[Counter1]);
      WEFTMenu.Init('WEFT.mnu');
      for Counter2 := 1 to 4 do
        if LeftAC.ACInfo.WingsInfo[Counter2] <> '' then
          begin
            Feature[Counter3] := LeftAC.ACInfo.WingsInfo[Counter2];
            Counter3 := Counter3 + 1;
          end;
      for Counter2 := 1 to 2 do
        if LeftAC.ACInfo.EngineInfo[Counter2] <> '' then
          begin
            Feature[Counter3] := LeftAC.ACInfo.EngineInfo[Counter2];
            Counter3 := Counter3 + 1;
          end;
      for Counter2 := 1 to 4 do
        if LeftAC.ACInfo.FuslagInfo[Counter2] <> '' then
          begin
            Feature[Counter3] := LeftAC.ACInfo.FuslagInfo[Counter2];
            Counter3 := Counter3 + 1;
         end;
   for Counter2 := 1 to 5 do
      if LeftAC.ACInfo.TailInfo[Counter2] <> '' then
         begin
            Feature[Counter3] := LeftAC.ACInfo.TailInfo[Counter2];
            Counter3 := Counter3 + 1;
         end;
   Counter2 := 1;
   Ch := #8;
   while Ch <> #13 do
      begin
         ShowFeature(Feature[Counter2]);
         if (Ch = '-') and (Counter2 > 1) then
            Counter2 := Counter2 - 1
         else if (Ch = '-') and (Counter2 = 1) then
            Counter2 := Counter3 - 1
         else if (Ch = '+') and (Counter2 < Counter3 - 1) then
            Counter2 := Counter2 + 1
         else if (Ch = '+') and (Counter2 = Counter3 - 1) then
            Counter2 := 1
         else if Ch = #27 then Exit
         else
            begin
               Sound(440);
               Delay(200);
               NoSound;
            end;
      end;
   Choice := WEFTMenu.GetChoice;
   if Choice = 'null' then
      Exit;
   DialogScreen[2].Init(Concat(Copy(StudentModel.ACArray[Counter1],1,4),'.nam'));
   LeftAC.Show(25,72);
   DialogScreen[2].Show(50,179);
   if Choice = LeftAC.ACInfo.AircraftName then
      begin
         DialogScreen[1].Init(Concat('Correct',Chr(Random(10)+48),'.dlg'));
         DialogScreen[1].Show(220,25);
         StudentModel.ACArray[Counter1] := '';
         Ch := ReadKey;
         while Ch = 'h' do
            begin
               GetHelp;
               Ch := ReadKey;
            end;
         DialogScreen[1].Hide;
         DialogScreen[1].Kill;
```

```
          end
        else
          begin
            DialogScreen[1].Init(Concat('Wrong',Chr(Random(10)+48),'.dlg'));
            DialogScreen[1].Show(220,25);
            if Copy(Choice,4,1) = ' ' then
              begin
         RightAC.Init(Concat(Copy(Choice,1,3),'_',Copy(StudentModel.ACArray[Counter1],5,3)));
                DialogScreen[3].Init(Concat(Copy(Choice,1,3),'_','.nam'));
              end
            else
              begin
           RightAC.Init(Concat(Copy(Choice,1,4),Copy(StudentModel.ACArray[Counter1],5,3)));
                DialogScreen[3].Init(Concat(Copy(Choice,1,4),'.nam'));
              end;
            RightAC.Show(365,72);
            DialogScreen[3].Show(390,179);
            Ch := ReadKey;
            while Ch = 'h' do
              begin
                GetHelp;
                Ch := ReadKey;
              end;
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
            DialogScreen[3].Hide;
            DialogScreen[3].Kill;
            RightAC.Hide;
            RightAC.Kill;
            if StudentModel.AddEntry(StudentModel.ACArray[Counter1], MaxNum) = false then
              Done := true;
          end;
        GoToXY(1,1);
        DialogScreen[2].Hide;
        DialogScreen[2].Kill;
        LeftAC.Hide;
        LeftAC.Kill;
        if Ch = #27 then Exit;
        Counter1 := StudentModel.GetEntry(MaxNum);
      end;
    Done := true;
end; {ReviewExpert}

procedure TestIntermediate;
begin {TestIntermediate}
  Ch := #13;
  Done := false;
  MaxNum := 150;
```

```
DialogScreen[1].Init('Return.dlg');
DialogScreen[1].Show(220,25);
Ch := ReadKey;
while Ch = 'h' do
   begin
      GetHelp;
      Ch := ReadKey;
   end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
DialogScreen[1].Init('TestInt.dlg');
DialogScreen[1].Show(220,25);
Ch := ReadKey;
while Ch = 'h' do
   begin
      GetHelp;
      Ch := ReadKey;
   end;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Ch = #27 then Exit;
Counter1 := StudentModel.GetEntry(MaxNum);
while Counter1 <> 0 do
   begin
      if StudentModel.ACArray[Counter1] <> '' then
         begin
            LeftAC.Init(StudentModel.ACArray[Counter1]);
            LeftAC.Show(25,72);
            WEFTMenu.Init('WEFT.mnu');
            Choice := WEFTMenu.GetChoice;
            if Choice = 'null' then
               begin
                  LeftAC.Hide;
                  LeftAC.Kill;
                  Exit;
               end;
            if Choice <> LeftAC.ACInfo.AircraftName then
               begin
                  Inc(StudentModel.NumMissed);
                  Sound(100);
                  Delay(200);
                  NoSound;
               end;
            Inc(StudentModel.NumShown);
            LeftAC.Hide;
            LeftAC.Kill;
         end;
```

```
        GoToXY(1,1);
        StudentModel.ACArray[Counter1] := '';
        Counter1 := StudentModel.GetEntry(MaxNum);
      end;
   Done := true;
end; {TestIntermediate}

procedure TestExpert;
var
   Feature : array [1..16] of name;
begin {TestExpert}
   MaxNum := 150;
   Ch := #13;
   Done := false;
   DialogScreen[1].Init('Return.dlg');
   DialogScreen[1].Show(220,25);
   Ch := ReadKey;
   while Ch = 'h' do
      begin
         GetHelp;
         Ch := ReadKey;
      end;
   DialogScreen[1].Hide;
   DialogScreen[1].Kill;
   if Ch = #27 then Exit;
   DialogScreen[1].Init('TestExp.dlg');
   DialogScreen[1].Show(220,25);
   Ch := ReadKey;
   while Ch = 'h' do
      begin
         GetHelp;
         Ch := ReadKey;
      end;
   DialogScreen[1].Hide;
   DialogScreen[1].Kill;
   if Ch = #27 then Exit;
   Counter1 := StudentModel.GetEntry(MaxNum);
   while Counter1 <> 0 do
      begin
         if StudentModel.ACArray[Counter1] <> '' then
            begin
               LeftAC.Init(StudentModel.ACArray[Counter1]);
               WEFTMenu.Init('WEFT.mnu');
               Counter3 := 1;
               for Counter2 := 1 to 4 do
                  if LeftAC.ACInfo.WingsInfo[Counter2] <> '' then
                     begin
                        Feature[Counter3] := LeftAC.ACInfo.WingsInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 2 do
  if LeftAC.ACInfo.EngineInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftAC.ACInfo.EngineInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 4 do
  if LeftAC.ACInfo.FuslagInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftAC.ACInfo.FuslagInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
for Counter2 := 1 to 5 do
  if LeftAC.ACInfo.TailInfo[Counter2] <> '' then
      begin
        Feature[Counter3] := LeftAC.ACInfo.TailInfo[Counter2];
        Counter3 := Counter3 + 1;
      end;
Counter2 := 1;
Ch := #8;
while Ch <> #13 do
   begin
      ShowFeature(Feature[Counter2]);
      if (Ch = '-') and (Counter2 > 1) then
         Counter2 := Counter2 - 1
      else if (Ch = '-') and (Counter2 = 1) then
         Counter2 := Counter3 - 1
      else if (Ch = '+') and (Counter2 < Counter3 - 1) then
         Counter2 := Counter2 + 1
      else if (Ch = '+') and (Counter2 = Counter3 - 1) then
         Counter2 := 1
      else if Ch = #27 then Exit
      else
         begin
            Sound(440);
            Delay(200);
            NoSound;
         end;
   end;
Choice := WEFTMenu.GetChoice;
if Choice = 'null' then
   Exit;
LeftAC.Show(25,72);
if Choice <> LeftAC.ACInfo.AircraftName then
   begin
      Inc(StudentModel.NumMissed);
```

```
                    Sound(100);
                    Delay(200);
                    NoSound;
                 end;
              Inc(StudentModel.NumShown);
              Ch := ReadKey;
              while Ch = 'h' do
                 begin
                    GetHelp;
                    Ch := ReadKey;
                 end;
              LeftAC.Hide;
              LeftAC.Kill;
              if Ch = #27 then Exit;
              GoToXY(1,1);
              StudentModel.ACArray[Counter1] := '';
              Counter1 := StudentModel.GetEntry(MaxNum);
           end;
     end;
  Done := true;
end; {TestExpert}

procedure EvaluateStudent;
var F : text;
begin {EvaluateStudent}
  if (StudentModel.Mode = 'Teach') and (StudentModel.NumShown >= MaxNum) then
     begin
           StudentModel.Update(StudentModel.StudentName,'Review',
              StudentModel.Level,StudentModel.TestScore);
       DialogScreen[1].Init('Advance1.dlg');
       DialogScreen[1].Show(220,25);
       Ch := ReadKey;
       DialogScreen[1].Hide;
       DialogScreen[1].Kill;
     end
  else if (StudentModel.Mode = 'Review') and (Done = true) then
     begin
       if StudentModel.GetEntry(MaxNum + 150) = 0 then
          begin
             if StudentModel.Level = 'Novice' then
                begin
                   StudentModel.Update(StudentModel.StudentName,'Teach','Intermediate',
                            StudentModel.TestScore);
                   DialogScreen[1].Init('Advance2.dlg');
                   DialogScreen[1].Show(220,25);
                   Ch := ReadKey;
                   DialogScreen[1].Hide;
                   DialogScreen[1].Kill;
                end
             else
                begin
                   StudentModel.Update(StudentModel.StudentName,'Test',
                            StudentModel.Level,StudentModel.TestScore);
                   DialogScreen[1].Init('Advance1.dlg');
                   DialogScreen[1].Show(220,25);
                   Ch := ReadKey;
                   DialogScreen[1].Hide;
                   DialogScreen[1].Kill;
                end
          end
       else
          for Counter1 := 1 to MaxNum do
             begin
                StudentModel.ACArray[Counter1] := StudentModel.MissedArray[Counter1];
                StudentModel.MissedArray[Counter1] := '';
                StudentModel.NumShown := 1;
                StudentModel.NumMissed := 0;
             end
    end
 else if (StudentModel.Mode = 'Test') and (Done = true) then
    begin
       if StudentModel.NumMissed = 0 then
          begin
             if StudentModel.Level = 'Expert' then
                begin
                   Assign(F,'HallFame.rec');
                   Append(F);
                   Writeln(F,StudentModel.StudentName);
                   Close(F);
                   StudentModel.Kill
                end
             else
                begin
                   DialogScreen[1].Init('Great.dlg');
                   DialogScreen[1].Show(220,25);
                   StudentModel.Update(StudentModel.StudentName,'Review','Expert',100);
                   Ch := ReadKey;
                   DialogScreen[1].Hide;
                   DialogScreen[1].Kill;
                end
          end
       else
          begin
            Score := Round(100*(1 - (StudentModel.NumMissed/(StudentModel.NumShown-1))));
             DialogScreen[1].Init('Score.dlg');
             DialogScreen[1].Show(220,25);
```

```
SetColor(1);
OutTextXY(315,60,Chr(Score div 10 + 48));
OutTextXY(325,60,Chr(Score mod 10 + 48));
SetColor(0);
Ch := ReadKey;
DialogScreen[1].Hide;
DialogScreen[1].Kill;
if Score >= 90 then
  begin
    if StudentModel.Level = 'Expert' then
      begin
        DialogScreen[1].Init('Outst.dlg');
        DialogScreen[1].Show(220,25);
        StudentModel.Update(StudentModel.StudentName,'Test','Expert',Score);
        Ch := ReadKey;
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
      end
    else
      begin
        DialogScreen[1].Init('Good.dlg');
        DialogScreen[1].Show(220,25);
        Ch := ReadKey;
        DialogScreen[1].Hide;
        DialogScreen[1].Kill;
        if StudentModel.Level = 'Novice' then
          StudentModel.Update(StudentModel.StudentName,'Teach',
                   'Intermediate',Score)
        else
             StudentModel.Update(StudentModel.StudentName,'Review',
                   'Expert',Score);
      end
  end
else if Score >= 80 then
  begin
    DialogScreen[1].Init('Fair.dlg');
    DialogScreen[1].Show(220,25);
    Ch := ReadKey;
    DialogScreen[1].Hide;
    DialogScreen[1].Kill;
StudentModel.Update(StudentModel.StudentName,'Test',StudentModel.Level,Score);
  end
else if Score >= 70 then
  begin
    DialogScreen[1].Init('Poor.dlg');
    DialogScreen[1].Show(220,25);
    Ch := ReadKey;
    DialogScreen[1].Hide;
```

```
               DialogScreen[1].Kill;
             StudentModel.Update(StudentModel.StudentName,'Review',StudentModel.Level,Score);
             end
           else
             begin
               DialogScreen[1].Init('Fail.dlg');
               DialogScreen[1].Show(220,25);
               Ch := ReadKey;
               DialogScreen[1].Hide;
               DialogScreen[1].Kill;
               if StudentModel.Level = 'Expert' then
                 StudentModel.Update(StudentModel.StudentName,
                        'Review','Intermediate',Score)
               else
                    StudentModel.Update(StudentModel.StudentName,
                        'Review','Novice',Score);
             end;
         end;
     end;
end; {EvaluateStudent}

procedure TutorSession;
begin {TutorSession}
   if not StudentModel.Get then
     Diagnose;
   Ch := 'C';
   while UpCase(Ch) = 'C' do
     begin
       SetColor(1);
       OutTextXY(320,190,Concat(StudentModel.Mode,'/',StudentModel.Level));
       SetColor(0);
       if StudentModel.Level = 'Novice' then
         begin
           if StudentModel.Mode = 'Teach' then Teach
           else if StudentModel.Mode = 'Review' then ReviewNovice;
         end
       else if StudentModel.Level = 'Intermediate' then
         begin
           if StudentModel.Mode = 'Teach' then Teach
           else if StudentModel.Mode = 'Review' then ReviewIntermediate
           else if StudentModel.Mode = 'Test' then TestIntermediate;
         end
       else if StudentModel.Level = 'Expert' then
         begin
           if StudentModel.Mode = 'Review' then ReviewExpert
           else if StudentModel.Mode = 'Test' then TestExpert;
         end;
       SetColor(0);
```

```
            OutTextXY(320,190,Concat(StudentModel.Mode,'/',StudentModel.Level));
            EvaluateStudent;
            DialogScreen[1].Init('Contin.dlg');
            DialogScreen[1].Show(220,75);
            Ch := ReadKey;
            DialogScreen[1].Hide;
            DialogScreen[1].Kill;
        end;
    StudentModel.Save;
end; {TutorSession}

end.
```

```
unit Game;

interface
   procedure PlayGame;

implementation

uses CRT, Graph, Student, Aircfts, Dialogs, Menus, Help;

type
   name = string[20];

var
   Counter1, Counter2, Counter3, MaxTime, MaxNum, PL1Score, PL2Score : integer;
   Score : string;
   Ch, Pl : char;
   LeftAC, RightAC : Aircraft;
   DialogScreen : array [1..2] of Dialog;
   WEFTMenu : Menu;
   Choice : name;

procedure ShowScores;
begin
   SetColor(0);
   Line(260,62,280,62);
   Line(260,65,280,65);
   Line(260,68,280,68);
   SetColor(1);
   Str(PL1Score,Score);
   OutTextXY(270,65,Score);
   SetColor(0);
   Line(360,62,380,62);
   Line(360,65,380,65);
   Line(360,68,380,68);
   SetColor(1);
   Str(PL2Score,Score);
   OutTextXY(370,65,Score);
   SetColor(0);
end;

procedure PlayOne;
begin
   DialogScreen[2].Init('NoGame.dlg');
   DialogScreen[2].Show(220,100);
   SetColor(1);
   OutTextXY(319,142,LeftAC.ACInfo.AircraftName);
   SetColor(0);
   Ch := ReadKey;
```

```
    while Ch = 'h' do
      begin
        GetHelp;
        Ch := ReadKey;
      end;
    DialogScreen[2].Hide;
    DialogScreen[2].Kill;
    LeftAC.Hide;
    RightAC.Hide;
    LeftAC.Kill;
    RightAC.Kill;
    PL2Score := PL2Score + 1;
    ShowScores;
    DialogScreen[2].Init('Ready.dlg');
    DialogScreen[2].Show(220,95);
    Ch := ReadKey;
    while Ch = 'h' do
      begin
        GetHelp;
        Ch := ReadKey;
      end;
    DialogScreen[2].Hide;
    DialogScreen[2].Kill;
  end;

  procedure PlayTwo;
  begin
    DialogScreen[1].Init('GScore.dlg');
    DialogScreen[1].Show(220,25);
    ShowScores;
    DialogScreen[2].Init('Ready.dlg');
    DialogScreen[2].Show(220,95);
    Ch := ReadKey;
    while Ch = 'h' do
      begin
        GetHelp;
        Ch := ReadKey;
      end;
    DialogScreen[2].Hide;
    DialogScreen[2].Kill;
    for Counter1 := 1 to 25 do
      begin
        Counter3 := 0;
        Counter2 := StudentModel.GetEntry(MaxNum);
        LeftAC.Init(StudentModel.ACArray[Counter2]);
        RightAC.Init(StudentModel.ACArray[Counter2]);
        LeftAC.Show(25,72);
        RightAC.Show(365,72);
```

```
while (UpCase(Ch) <> 'A') and (UpCase(Ch) <> 'L') and (Ch <> #8) do
   begin
      Counter3 := Counter3 + 1;
      if Counter3 > MaxTime then
         Ch := #8;
      Delay(5);
      if KeyPressed then
         Ch := ReadKey;
   end;
PI := UpCase(Ch);
if PI = #8 then
   begin
      if MaxTime <> 1000 then
         PlayOne
      else
         begin
            LeftAC.Hide;
            RightAC.Hide;
            LeftAC.Kill;
            RightAC.Kill;
            DialogScreen[2].Init('TimeOut.dlg');
            DialogScreen[2].Show(220,95);
            Ch := ReadKey;
            while Ch = 'h' do
               begin
                  GetHelp;
                  Ch := ReadKey;
               end;
            DialogScreen[2].Hide;
            DialogScreen[2].Kill;
         end
   end
else
   begin
      RightAC.Hide;
      RightAC.Kill;
      if PI = 'A' then
         DialogScreen[2].Init('Play1.dlg')
      else DialogScreen[2].Init('Play2.dlg');
      DialogScreen[2].Show(220,25);
      WEFTMenu.Init('WEFT.mnu');
      Choice := WEFTMenu.GetChoice;
      DialogScreen[2].Hide;
      LeftAC.Hide;
      DialogScreen[2].Kill;
      LeftAC.Kill;
      if Choice <> LeftAC.ACInfo.AircraftName then
         begin
```

```
              Sound(100);
              Delay(200);
              if Pl = 'A' then Dec(PL1Score)
              else Dec(PL2Score);
              NoSound;
            end
          else
            begin
              if Pl = 'A' then Inc(PL1Score)
              else Inc(Pl2Score);
            end;
          ShowScores;
          if Counter1 < 25 then
            begin
              DialogScreen[2].Init('Ready.dlg');
              DialogScreen[2].Show(220,95);
              Ch := ReadKey;
              while Ch = 'h' do
                begin
                  GetHelp;
                Ch := ReadKey;
              end;
              DialogScreen[2].Hide;
              DialogScreen[2].Kill;
            end;
        end;
      GoToXY(1,1);
      Ch := #13;
    end;
  if PL1Score > PL2Score then
    DialogScreen[2].Init('Win1.dlg')
  else if PL2Score > PL1Score then
    DialogScreen[2].Init('Win2.dlg')
  else DialogScreen[2].Init('Tie.dlg');
  DialogScreen[2].Show(220,95);
  Ch := ReadKey;
  DialogScreen[2].Hide;
  DialogScreen[2].Kill;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
end;

procedure PlayGame;
begin
  StudentModel.Update(' ','Game','Game',0);
  MaxNum := 150;
  DialogScreen[1].Init('Game.dlg');
  DialogScreen[1].Show(220,75);
  Ch := ReadKey;
  while (Ch <> '1') and (Ch <> '2') and (Ch <> #27) do
     begin
        if Ch = 'h' then
           GetHelp;
        Ch := ReadKey;
     end;
  DialogScreen[1].Hide;
  DialogScreen[1].Kill;
  PL1Score := 0;
  PL2Score := 0;
  if Ch = #27 then Exit
  else if Ch = '1' then
     begin
        Randomize;
        MaxTime := Random(250) + 250;
        PlayTwo;
     end
  else if Ch = '2' then
     begin
        MaxTime := 1000;
        PlayTwo;
     end;
end;

end.
```

```
unit Help;

interface

uses CRT, Graph, Screens, Student;

type
  HELPScreen = object(Screen)
    OldP : pointer;
    procedure Show(XLoc, YLoc : integer);
    procedure Hide;
  end;

procedure GetHelp;

implementation

var
  HelpItem : HelpScreen;
  Ch : char;

procedure GetHelp;
begin
  if StudentModel.Mode = '' then
    HelpItem.Init('Menu.hlp')
  else if StudentModel.Mode = 'Game' then
    HelpItem.Init('Game.hlp')
  else if StudentModel.Mode = 'Setup' then
    HelpItem.Init('Setup.hlp')
  else if StudentModel.Mode = 'StuRep' then
    HelpItem.Init('StuRep.hlp')
  else if StudentModel.Mode = 'DelStu' then
    HelpItem.Init('DelStu.hlp')
  else if StudentModel.Mode = 'SelAC' then
    HelpItem.Init('SelAC.hlp')
  else if StudentModel.Mode = 'AddAC' then
    HelpItem.Init('AddAC.hlp')
  else if StudentModel.Mode = 'Diagnose' then
    HelpItem.Init('Diagnose.hlp')
  else if StudentModel.Mode = 'Teach' then
    begin
      if StudentModel.Level = 'Novice' then
        HelpItem.Init('TeaNov.hlp')
      else HelpItem.Init('TeaInt.hlp');
    end
  else if StudentModel.Mode = 'Test' then
    begin
      if StudentModel.Level = 'Intermediate' then
```

```
            HelpItem.Init('TestInt.hlp')
         else HelpItem.Init('TestExp.hlp')
      end
   else
      begin
         if StudentModel.Level = 'Novice' then
            HelpItem.Init('RevNov.hlp')
         else if StudentModel.Level = 'Intermediate' then
            HelpItem.Init('RevInt.hlp')
         else HelpItem.Init('RevExp.hlp');
      end;
   HelpItem.Show(0,0);
   Ch := ReadKey;
   HelpItem.Hide;
   HelpItem.Kill;
end;

procedure HELPScreen.Show(XLoc,YLoc : integer);
begin
   if not IsVisible then
      begin
         X := XLoc;
         Y := YLoc;
         GetMem(OldP, MemSize);{Save the old bitmap}
         GetImage(X,Y,X+639,Y+199,OldP^);
         PutImage(X,Y,P^,CopyPut);{and draw the new bitmap.}
         IsVisible := true;
      end;
end;

procedure HELPScreen.Hide;
begin
   if IsVisible then
      begin
         PutImage(X,Y,OldP^,CopyPut);{Put the old bitmap back}
         FreeMem(OldP, MemSize);{and free the heap memory.}
         IsVisible := false;
      end;
end;

end.
```

```
unit Utility;

interface
  procedure SetUp;

implementation

uses
     DOS,CRT,Graph,PCX_TP,Student,Menus,Screens,Aircfts,Dialogs,Help;

type
  name = string[20];

var
  retcode : integer;
  F : text;
  TextString : string;
  Counter : integer;
  SetUpMenu : Menu;
  DialogScreen : Dialog;
  StudentMenu : Menu;
  ACMenu : Menu;
  StudentReport : Screen;
  Choice : name;
  Ch : char;

procedure GetStudents;
var
  DirInfo : SearchRec;
begin
  Assign(F,'Student.rec');
  ReWrite(F);
  Writeln(F,'STUDENTS');
  Writeln(F,320);
  Writeln(F,50);
  Writeln(F,1);
  FindFirst('*.mdl',AnyFile, DirInfo);
  while DosError = 0 do
     begin
       Writeln(F,Copy(DirInfo.Name,1,5));
       FindNext(DirInfo);
     end;
  Close(F);
end;

procedure SelectAC;
var
  ACArray : array [1..150] of name;
```

```
    DirInfo : SearchRec;
    F1,F2 : text;
    Counter1 : integer;
    ACName : name;
  begin
    StudentModel.Mode := 'SelAC';
    Counter := 1;
    FindFirst('*.nam',AnyFile, DirInfo);
    while DosError = 0 do
      begin
        Assign(F1,Concat(Copy(DirInfo.Name,1,4),'_#1.dat'));
        Reset(F1);
        Readln(F1,ACName);
        ACArray[Counter] := ACName;
        Counter := Counter + 1;
        Close(F1);
        FindNext(DirInfo);
      end;
    Assign(F,'Aircraft.rec');
    ReWrite(F);
    Writeln(F,'AIRCRAFT');
    Writeln(F,320);
    Writeln(F,50);
    Writeln(F,1);
    for Counter1 := 1 to Counter-1 do
      if ACArray[Counter1] <> '' then
        Writeln(F,ACArray[Counter1]);
    Close(F);
    Assign(F,'Intermed.def');
    Assign(F1,'WEFT.mnu');
    Rewrite(F);
    Rewrite(F1);
    Writeln(F,'Intermed default');
    Writeln(F,'Teach');
    Writeln(F,'Intermediate');
    Writeln(F,0);
    Writeln(F,1);
    Writeln(F,0);
    Writeln(F1,'AIRCRAFT');
    Writeln(F1,490);
    Writeln(F1,76);
    Writeln(F1,0);
    DialogScreen.Init('SelAC.dlg');
    DialogScreen.Show(0,0);
    ACMenu.Init('Aircraft.rec');
    Choice := ACMenu.GetChoice;
    while Choice <> 'null' do
      begin
```

```
          if Copy(Choice,4,1) = ' ' then
            begin
              Writeln(F,Concat(Copy(Choice,1,3),'__#1'));
              Writeln(F,Concat(Copy(Choice,1,3),'__#2'));
              Writeln(F,Concat(Copy(Choice,1,3),'__#3'));
            end
          else
            begin
              Writeln(F,Concat(Copy(Choice,1,4),'_#1'));
              Writeln(F,Concat(Copy(Choice,1,4),'_#2'));
              Writeln(F,Concat(Copy(Choice,1,4),'_#3'));
            end;
          Writeln(F1,Choice);
          for Counter1 := 1 to Counter-1 do
            if ACArray[Counter1] = Choice then
              ACArray[Counter1] := ' ';
          Assign(F2,'Aircraft.rec');
          ReWrite(F2);
          Writeln(F2,'AIRCRAFT');
          Writeln(F2,320);
          Writeln(F2,50);
          Writeln(F2,1);
          for Counter1 := 1 to Counter-1 do
            if ACArray[Counter1] <> ' ' then
              Writeln(F2,ACArray[Counter1]);
          Close(F2);
          ACMenu.Init('Aircraft.rec');
          Choice := ACMenu.GetChoice;
        end;
    DialogScreen.Hide;
    DialogScreen.Kill;
    Close(F);
    Close(F1);
end;


procedure AddAC;
const
  ALPHA = ['0'..'9','A'..'Z','a'..'z','-'];
var
  ACName : name;
  DirInfo : SearchRec;
  Counter1,Counter2 : integer;
  Size : word;
  P : pointer;
  s : string;
  FileName : string;
  F : file;
```

```
    F1 : text;
    Ch : char;
    Offset,MaxX,MaxY,X,Y : integer;
    OldAC : aircraft;
  begin
    StudentModel.Mode := 'AddAC';
    ACName := '';
    Counter1 := 1;
    SetColor(1);
    DialogScreen.Init('GetAC.dlg');
    DialogScreen.Show(220,25);
    Line(220,64,420,64);
    Line(220,67,420,67);
    Line(220,70,420,70);
    Ch := #8;
    while (Ch <> #13) and (Counter1 < 21) do
      begin
        Ch := ReadKey;
        if Ch = chr(32) then
          begin
            ACName := Concat(ACName,chr(32));
            Counter1 := Counter1 + 1;
          end;
        if Ch in ALPHA then
          begin
            SetColor(0);
            OutTextXY(225+10*Counter1,68,Ch);
            ACName := Concat(ACName,Ch);
            Counter1 := Counter1 + 1;
            SetColor(1);
          end;
        if (Ch = #8) and (Counter1 > 1) then
          begin
            SetColor(1);
            Line(210+10*Counter1,64,220+10*Counter1,64);
            Line(210+10*Counter1,67,220+10*Counter1,67);
            Line(210+10*Counter1,70,220+10*Counter1,70);
            Counter1 := Counter1 - 1;
            ACName := Copy(ACName,1,Counter1-1);
            SetColor(0);
          end;
      end;
    DialogScreen.Hide;
    DialogScreen.Kill;
    DialogScreen.Init('New.dlg');
    DialogScreen.Show(0,179);
    SetTextJustify(CenterText,CenterText);
    OutTextXY(100,189,ACName);
```

```
if Copy(ACName,4,1) = ' ' then
   FindFirst(Concat(Copy(ACName,1,3),'_.nam'),AnyFile, DirInfo)
else FindFirst(Concat(Copy(ACName,1,4),'.nam'),AnyFile, DirInfo);
Size := ImageSize(0,179,200,199);
GetMem(P,Size);
GetImage(0,179,200,199,P^);
if Copy(ACName,4,1) = ' ' then
   Assign(F,Concat(Copy(ACName,1,3),'_.nam'))
else Assign(F,Concat(Copy(ACName,1,4),'.nam'));
ReWrite(F,1);
BlockWrite(F,P^,Size);
Close(F);
FreeMem(P,Size);
DialogScreen.Hide;
DialogScreen.Kill;
Offset := 140;
MaxX := 499;
MaxY := 219;
for Counter1 := 1 to 3 do
   begin
      X := 0;
      Y := 0;
      str(Counter1,s);
      SetColor(1);
      SetLineStyle(SolidLn,0,NormWidth);
      SetWriteMode(XorPut);
      if DosError = 0 then
         begin
            if Copy(ACName,4,1) = ' ' then OldAC.Init(Concat(Copy(ACName,1,3),'__#',s))
            else OldAC.Init(Concat(Copy(ACName,1,4),'_#',s));
            OldAC.Show(0,0);
            for X := MaxX downto 0 do
               for Y := MaxY downto 0 do
                  if GetPixel(X,Y) = 1 then
                     begin
                        PutPixel(X,Y,0);
                        PutPixel(X*2+140,Y*2-10,1);
                        PutPixel(X*2+141,Y*2-10,1);
                        PutPixel(X*2+140,Y*2+1-10,1);
                        PutPixel(X*2+141,Y*2+1-10,1);
                     end;
            OldAC.Kill;
         end;
      Ellipse(MaxX div 2 + Offset,MaxY div 2-10,0,360,MaxX div 2,MaxY div 2);
      repeat
         Line(X+Offset,Y,X+Offset,Y+1);
         Line(X+Offset,Y,X+Offset+1,Y);
         Ch := ReadKey;
```

```
          while Ch = 'h' do
            begin
              GetHelp;
              Ch := ReadKey;
            end;
          Line(X+Offset,Y,X+Offset,Y+1);
          Line(X+Offset,Y,X+Offset+1,Y);
          Case Ch of
            chr(80), chr(50) : if Y < MaxY then Y := Y + 2;
            chr(75), chr(52) : if X > 0 then X := X - 2;
            chr(77), chr(54) : if X < MaxX then X := X + 2;
            chr(72), chr(56) : if Y > 0 then Y := Y - 2;
            chr(32) :begin
                   PutPixel(X+Offset,Y,abs(GetPixel(X+Offset,Y)-1));
                   PutPixel(X+Offset+1,Y,abs(GetPixel(X+Offset+1,Y)-1));
                   PutPixel(X+Offset,Y+1,abs(GetPixel(X+Offset+1,Y+1)-1));
                   PutPixel(X+Offset+1,Y+1,abs(GetPixel(X+Offset+1,Y+1)-1));
                 end;
          end;
        until Ch = #13;
        SetColor(0);
        Ellipse(MaxX div 2 + Offset,MaxY div 2-10,0,360,MaxX div 2,MaxY div 2);
        for X := 0 to MaxX div 2 do
          for Y := 0 to MaxY div 2 - 10 do
            if GetPixel(X*2+Offset,Y*2) = 1 then
              begin
                PutPixel(X,Y+5,1);
                PutPixel(X*2+Offset,Y*2,0);
                PutPixel(X*2+Offset+1,Y*2,0);
                PutPixel(X*2+Offset,Y*2+1,0);
                PutPixel(X*2+Offset+1,Y*2+1,0);
              end;
        retcode := pcxSetDisplay(pcxCGA_6);
        if Copy(ACName,4,1) = ' ' then
         retcode := pcxDisplayFile(Concat(Copy(ACName,1,3),'__#',s,'.pcx'),0,0,MaxX div 2,MaxY
div 2,0)
      else retcode := pcxDisplayFile(Concat(Copy(ACName,1,4),'_#',s,'.pcx'),0,0,MaxX div 2,MaxY
div 2,0);
        {Input the data for this view}
        if Copy(ACName,4,1) = ' ' then
          Assign(F1,Concat(Copy(ACName,1,3),'__#',s,'.dat'))
        else Assign(F1,Concat(Copy(ACName,1,4),'_#',s,'.dat'));
        ReWrite(F1);
        Writeln(F1,ACName);
        Writeln(F1,'');
        Writeln(F1,'');
        SetColor(1);
        SetLineStyle(SolidLn,0,ThickWidth);
```

```
for Counter2 := 1 to 40 do
   Line(390,Counter2*3+65,590,Counter2*3+65);
SetUpMenu.Init('Wing1.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Wing10',s))
      else Writeln(F1,Concat('Wing1',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;
SetUpMenu.Init('Wing2.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Wing20',s))
      else Writeln(F1,Concat('Wing2',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;
SetUpMenu.Init('Wing3.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Wing30',s))
      else Writeln(F1,Concat('Wing3',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;
SetUpMenu.Init('Wing4.mnu');
```

```
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Wing40',s))
      else Writeln(F1,Concat('Wing4',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;
SetUpMenu.Init('Engi1.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Engi10',s))
      else Writeln(F1,Concat('Engi1',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;
SetUpMenu.Init('Engi2.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
      Writeln(F1,Choice);
      Str(LastSelection,s);
      if LastSelection < 10 then
         Writeln(F1,Concat('Engi20',s))
      else Writeln(F1,Concat('Engi2',s));
   end
else
   begin
      Writeln(F1,'');
      Writeln(F1,'');
   end;
SetUpMenu.Init('Fuse1.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
   begin
```

```
    Writeln(F1,Choice);
    Str(LastSelection,s);
    if LastSelection < 10 then
      Writeln(F1,Concat('Fuse10',s))
    else Writeln(F1,Concat('Fuse1',s));
  end
else
  begin
    Writeln(F1,'');
    Writeln(F1,'');
  end;
SetUpMenu.Init('Fuse2.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
  begin
    Writeln(F1,Choice);
    Str(LastSelection,s);
    if LastSelection < 10 then
      Writeln(F1,Concat('Fuse20'.s))
    else Writeln(F1,Concat('Fuse2',s));
  end
else
  begin
    Writeln(F1,'');
    Writeln(F1,'');
  end;
SetUpMenu.Init('Fuse3.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
  begin
    Writeln(F1,Choice);
    Str(LastSelection,s);
    if LastSelection < 10 then
      Writeln(F1,Concat('Fuse30',s))
    else Writeln(F1,Concat('Fuse3',s));
  end
else
  begin
    Writeln(F1,'');
    Writeln(F1,'');
  end;
SetUpMenu.Init('Fuse4.mnu');
Choice := SetUpMenu.GetChoice;
if Choice <> 'null' then
  begin
    Writeln(F1,Choice);
    Str(LastSelection,s);
    if LastSelection < 10 then
```

```
          Writeln(F1,Concat('Fuse40',s))
        else Writeln(F1,Concat('Fuse4',s));
      end
    else
      begin
        Writeln(F1,'');
        Writeln(F1,'');
      end;
    SetUpMenu.Init('Tail1.mnu');
    Choice := SetUpMenu.GetChoice;
    if Choice <> 'null' then
      begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
          Writeln(F1,Concat('Tail10',s))
        else Writeln(F1,Concat('Tail1',s));
      end
    else
      begin
        Writeln(F1,'');
        Writeln(F1,'');
      end;
    SetUpMenu.Init('Tail2.mnu');
    Choice := SetUpMenu.GetChoice;
    if Choice <> 'null' then
      begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
          Writeln(F1,Concat('Tail20',s))
        else Writeln(F1,Concat('Tail2',s));
      end
    else
      begin
        Writeln(F1,'');
        Writeln(F1,'');
      end;
    SetUpMenu.Init('Tail3.mnu');
    Choice := SetUpMenu.GetChoice;
    if Choice <> 'null' then
      begin
        Writeln(F1,Choice);
        Str(LastSelection,s);
        if LastSelection < 10 then
          Writeln(F1,Concat('Tail30',s))
        else Writeln(F1,Concat('Tail3',s));
      end
```

```
        else
          begin
            Writeln(F1,'');
            Writeln(F1,'');
          end;
        SetUpMenu.Init('Tail4.mnu');
        Choice := SetUpMenu.GetChoice;
        if Choice <> 'null' then
          begin
            Writeln(F1,Choice);
            Str(LastSelection,s);
            if LastSelection < 10 then
              Writeln(F1,Concat('Tail40',s))
            else Writeln(F1,Concat('Tail4',s));
          end
        else
          begin
            Writeln(F1,'');
            Writeln(F1,'');
          end;
        SetUpMenu.Init('Tail5.mnu');
        Choice := SetUpMenu.GetChoice;
        if Choice <> 'null' then
          begin
            Writeln(F1,Choice);
            Str(LastSelection,s);
            if LastSelection < 10 then
              Writeln(F1,Concat('Tail50',s))
            else Writeln(F1,Concat('Tail5',s));
          end
        else
          begin
            Writeln(F1,'');
            Writeln(F1,'');
          end;
        Close(F1);
        ClearDevice;
      end;
end;

procedure Report;
begin
  StudentModel.Mode := 'StuRep';
  GetStudents;
  DialogScreen.Init('StuRep.dlg');
  DialogScreen.Show(0,0);
  StudentMenu.Init('Student.rec');
  Choice := StudentMenu.GetChoice;
  while Choice <> 'null' do
    begin
      StudentReport.Init('StuRep.scr');
      StudentReport.Show(0,0);
      Assign(F,Concat(Choice,'.mdl'));
      Reset(F);
      SetColor(1);
      SetTextJustify(LeftText,LeftText);
      OutTextXY(325,74,Choice);
      for Counter := 1 to 4 do
        begin
          Readln(F,TextString);
          if TextString = '0' then OutTextXY(325,Counter*12+74,'Not Tested')
          else OutTextXY(325,Counter*12+74,TextString);
        end;
      SetColor(0);
      Close(F);
      Ch := ReadKey;
      while Ch = 'h' do
        begin
          GetHelp;
          Ch := ReadKey;
        end;
      ClearDevice;
      DialogScreen.Hide;
      StudentReport.Kill;
      DialogScreen.Show(0,0);
      GetStudents;
      StudentMenu.Init('Student.rec');
      Choice := StudentMenu.GetChoice;
    end;
  DialogScreen.Hide;
  DialogScreen.Kill;
end;

procedure DelStudent;
begin
  StudentModel.Mode := 'DelStu';
  GetStudents;
  DialogScreen.Init('StuDel.dlg');
  DialogScreen.Show(0,0);
  StudentMenu.Init('Student.rec');
  Choice := StudentMenu.GetChoice;
  while Choice <> 'null' do
    begin
      Exec('\COMMAND.COM',Concat('/C del ',Choice,'.mdl'));
      GetStudents;
      StudentMenu.Init('Student.rec');
```

```
          Choice := StudentMenu.GetChoice;
        end;
    DialogScreen.Hide;
    DialogScreen.Kill;
end;

procedure SetUp;
const
    ALPHA = ['0'..'9','A'..'Z','a'..'z','-'];
var
    Password : name;
    Counter1 : integer;
begin
    Password := '';
    Counter1 := 1;
    SetColor(1);
    DialogScreen.Init('Passwd.dlg');
    DialogScreen.Show(220,25);
    Line(220,64,420,64);
    Line(220,67,420,67);
    Line(220,70,420,70);
    Ch := #8;
    while (Ch <> #13) and (Counter1 < 21) do
        begin
            Ch := ReadKey;
            if Ch = chr(32) then
                begin
                    Password := Concat(Password,chr(32));
                    Counter1 := Counter1 + 1;
                end;
            if Ch in ALPHA then
                begin
                    SetColor(0);
                    OutTextXY(225+10*Counter1,68,Ch);
                    Password := Concat(Password,Ch);
                    Counter1 := Counter1 + 1;
                    SetColor(1);
                end;
            if (Ch = #8) and (Counter1 > 1) then
                begin
                    SetColor(1);
                    Line(210+10*Counter1,64,220+10*Counter1,64);
                    Line(210+10*Counter1,67,220+10*Counter1,67);
                    Line(210+10*Counter1,70,220+10*Counter1,70);
                    Counter1 := Counter1 - 1;
                    Password := Copy(Password,1,Counter1-1);
                    SetColor(0);
                end;
          end;
        DialogScreen.Hide;
        DialogScreen.Kill;
        if Password <> '101653362' then
          Exit;
        StudentModel.Mode := 'Setup';
        SetUpMenu.Init('SetUp.mnu');
        Choice := SetUpMenu.GetChoice;
        while (Choice <> 'EXIT') and (Choice <> 'null') do
          begin
            if Choice = 'SELECT AIRCRAFT' then SelectAC
            else if Choice = 'ADD/MODIFY AIRCRAFT' then AddAC
            else if Choice = 'STUDENT REPORT' then Report
            else if Choice = 'DELETE STUDENT' then DelStudent;
            GotoXY(1,1);
            StudentModel.Mode := 'Setup';
            SetUpMenu.Init('SetUp.mnu');
            Choice := SetUpMenu.GetChoice;
          end;
      end;
      end.
```

```
program Install;

uses DOS, CRT, GRAPH;

var
   Ch : char;
   grDriver : integer;
   grMode : integer;
   ErrCode : integer;
   DirInfo : SearchRec;
begin
   ChDir('A:\');
   grDriver := CGA;
   grMode := CGAHi;
   InitGraph(grDriver,grMode,'');
   ErrCode := GraphResult;
   if ErrCode <> 0 then
      begin
         writeln('This program requires CGA graphics.');
         writeln('        Install ABORTED.');
         writeln('  Press any key to return to DOS.');
         Ch := ReadKey;
         Exit;
      end;

   SetTextJustify(CenterText,CenterText);
   ClearDevice;
   FindFirst('A:COPYRITE.90',AnyFile,DirInfo);
   if DOSError <> 0 then
      begin
         Sound(440);
         Delay(100);
         NoSound;
         ClearDevice;
         OutTextXY(320,100,'The Aircraft Recognition Tutor is COPY PROTECTED.');
         OutTextXY(320,110,'Please see the User''s Manual for details.');
         OutTextXY(320,120,'Press any key to return to DOS.');
         Ch := ReadKey;
         ClearDevice;
         CloseGraph;
         Exit;
      end;

   ClearDevice;
   OutTextXY(320,100,'Welcome to the Aircraft Recognition Tutor Install Program');
   OutTextXY(320,110,'Press any key to begin');
   Ch := ReadKey;
```

```
ClearDevice;
OutTextXY(320,100,'Creating a new directory called "ARTUTOR".');
ChDir('C:\');
MkDir('ARTUTOR');
if IOResult <> 0 then
   begin
      OutTextXY(320,110,'Cannot create directory.  Install ABORTED.');
      OutTextXY(320,120,'Press any key to return to DOS.');
      Ch := ReadKey;
      ClearDevice;
      CloseGraph;
      Exit;
   end;
ChDir('ARTUTOR');

ClearDevice;
FindFirst('A:Disk.1',AnyFile,DirInfo);
while DOSError <> 0 do
   begin
      Sound(440);
      Delay(100);
      NoSound;
      ClearDevice;
      OutTextXY(320,100,'Insure that Disk 1 is in Drive A: and press any key.');
      Ch := ReadKey;
      FindFirst('A:Disk.1',AnyFile,DirInfo);
   end;

ClearDevice;
OutTextXY(320,100,'Copying files from Disk 1');
Exec('\COMMAND.COM','/C copy A:*.*');
Exec('\COMMAND.COM','/C copy A:\WEFT\*.*');
Exec('\COMMAND.COM','/C copy A:\DIALOG\*.*');
Exec('\COMMAND.COM','/C copy A:\MENU\*.*');
Exec('\COMMAND.COM','/C del A:COPYRITE.90');

ClearDevice;
OutTextXY(320,100,'Insert Disk 2 in Drive A: and press any key.');
Ch := ReadKey;
FindFirst('A:Disk.2',AnyFile,DirInfo);
while DOSError <> 0 do
   begin
      Sound(440);
      Delay(100);
      NoSound;
      ClearDevice;
      OutTextXY(320,100,'Insure that Disk 2 is in Drive A: and press any key.');
      Ch := ReadKey;
```

```
        FindFirst('A:Disk.2',AnyFile,DirInfo);
      end;
    ClearDevice;
    OutTextXY(320,100,'Copying files from Disk 2');
    Exec('\COMMAND.COM','/C copy A:*.*');

    ClearDevice;
    OutTextXY(320,100,'Insert Disk 3 in Drive A: and press any key.');
    Ch := ReadKey;
    FindFirst('A:Disk.3',AnyFile,DirInfo);
    while DOSError <> 0 do
      begin
        Sound(440);
        Delay(100);
        NoSound;
        ClearDevice;
        OutTextXY(320,100,'Insure that Disk 3 is in Drive A: and press any key.');
        Ch := ReadKey;
        FindFirst('A:Disk.3',AnyFile,DirInfo);
      end;
    ClearDevice;
    OutTextXY(320,100,'Copying files from Disk 3');
    Exec('\COMMAND.COM','/C copy A:*.*');
    Exec('\COMMAND.COM','/C copy A:\AC1\*.*');
    Exec('\COMMAND.COM','/C copy A:\AC2\*.*');

    ClearDevice;
    OutTextXY(320,100,'Insert Disk 4 in Drive A: and press any key.');
    Ch := ReadKey;
    FindFirst('A:Disk.4',AnyFile,DirInfo);
    while DOSError <> 0 do
      begin
        Sound(440);
        Delay(100);
        NoSound;
        ClearDevice;
        OutTextXY(320,100,'Insure that Disk 4 is in Drive A: and press any key.');
        Ch := ReadKey;
        FindFirst('A:Disk.4',AnyFile,DirInfo);
      end;
    ClearDevice;
    OutTextXY(320,100,'Copying files from Disk 4');
    Exec('\COMMAND.COM','/C copy A:\AC3\*.*');
    Exec('\COMMAND.COM','/C copy A:\AC4\*.*');
    Exec('\COMMAND.COM','/C del install.exe');
    Exec('\COMMAND.COM','/C del unstall.exe');
    Exec('\COMMAND.COM','/C del disk.*');
    ClearDevice;
    OutTextXY(320,100,'Install program complete.');
    OutTextXY(320,110,'Press any key to return to DOS');
    Ch := ReadKey;
    ClearDevice;
    CloseGraph;
  end.
```

```
program Unstall;

uses DOS, CRT;

var
   Ch : char;
   DirInfo : SearchRec;
begin
   ClrScr;
   Writeln('Uninstalling the Aircraft Recognition Tutor.');
   Writeln('Please Wait.');
   Exec('\COMMAND.COM','/C copy C:\ARTUTOR\COPYRITE.90 A:');
   ChDir('C:\');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\COPYRITE.90');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.bgi');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.scr');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.pic');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.dlg');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.nam');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.mnu');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.def');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.dat');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.exe');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.hlp');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.rec');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.def');
   Exec('\COMMAND.COM','/C del C:\ARTUTOR\*.mdl');
   RmDir('ARTUTOR');
   ClrScr;
   Writeln('Unstall Completed.  You may now Install the Aircraft Recognition Tutor');
   Writeln('on another computer.');
   Writeln('Press any key to return to DOS');
   Ch := ReadKey;
   ClrScr;
end.
```

APPENDIX B - USER'S MANUAL

Aircraft Recognition Tutor

User's Manual

Larry W. Campbell, 1990

Contents

Contents

Aircraft Recognition Tutor
Larry W. Campbell

1 Introduction

1.1 About the Tutor

The Aircraft Recognition Tutor was developed as part of a Masters Thesis in Computer Science by CPT Larry W. Campbell. One of the major goals of the thesis was to demonstrate that, using existing technology in Computer Science and Artificial Intelligence, a useful computer training tool could be developed quickly and inexpensively. In addition, this training tool would run on existing hardware in the field (U.S. Army ADA Battalions).

Part of this goal has definately been met. The Aircraft Recognition Tutor was developed over a period of 3 months by a single individual. Whether the tutor is useful will be determined by you, the user.

The requirement that the tutor run on existing hardware imposed some serious constraints on the development of the program. U.S. Army ADA Battalions currently have Zenith Z-248 computers with 640K RAM, 20MB Hard Disk, 360K 5-1/4" Floppy Disk, and CGA graphics. Because the computers use CGA graphics, the graphics resolution of the tutor was limited to either 320 x 200 pixels with 4 colors or 640 x 200 with 2 colors. Both are inadequate in my opinion, however, until the computer systems are upgraded to EGA (640 x 350, 16 color), VGA (640 x 480, 16 color), or Super VGA (800 x 600, 16 color), they will have to do. After experimenting with both of the available graphics modes, the 640 x 200 mode was chosen as the best comprimise. If the tutor is accepted and used by the field, future versions will become available for higher resolution graphics modes which will allow the aircraft used by the system to be of almost photographic quality.

The topic for the tutor was chosen based on the past experience of the author. Aircraft Recognition remains a crucial skill needed by all SHORAD (Short Range Air Defense) soldiers. Unfortunately, this skill is difficult to acquire and is quite perishable. In spite of continuous training by units, soldiers continue to have difficulty developing and maintaining proficiency at visually identifying aircraft. Numerous training aids exist to help units train soldiers, however, they need a qualified instructor in order to be effective. These instructors are not always available, and the varying degrees of proficiency make group training less than ideal. The Aircraft Recognition Tutor attempts to fill a gap that exists in most VACR (Visual Aircraft Recognition) training programs. The tutor uses the WEFT (Wings, Engine, Fuselage, Tail) theory to teach aircraft recognition. Because the tutor is designed to identify the soldiers's current ability and teach at a level appropriate to that ability, it is useful for introducing VACR to new soldiers as well as providing refresher training to more advanced soldiers.

In order to make the program simple and interesting to use, the interface was kept very basic. In addition, a game mode was incorporated in order to encourage use of the program. The best training tools are of no value if they are not used.

Comments on the design of the tutor system, as well as any suggestions, are solicited. A suggestion form is included as Appendix C. This form can also be used to report any bugs found in the system.

1.2 WEFT Theory

The WEFT (Wings, Engine, Fuselage, Tail) theory is currently believed to be the best method for teaching VACR (Visual Aircraft Recognition). This theory is described in detail in FM 44-30, dated October 1986. The Aircraft Recognition Tutor uses WEFT theory as the basis from which to teach VACR.

According to FM 44-30, all aircraft are composed of the same basic elements: wings to provide lift, an engine to provide motive power, a fuselage to carry the payload and controls, and a tail assembly which usually serves the purpose of controlling the direction of flight. These elements differ in their shape, size, number, and position. It is these basic elements that distinquish one aircraft type from another. Detailed parts cannot be used as the only aid to aircraft recognition, mainly because of the distances at which recognition should occur. The individual components can be isolated for descriptions and studied as separate recognition features. It is the composite of these features that must be learned in order to recognize and identify an aircraft.

For a detailed discussion of WEFT theory, and the method used for describing the basic elements, see FM 44-30 Chapter 3.

2 Installing the Tutor

2.1 System Requirements

The Aircraft Recognition Tutor requires a PC compatible computer with the following features:

DOS 2.1 (or higher)
512K RAM
Hard Disk (system uses 1MB)
360K 5-1/4" Floppy Disk
CGA Graphics Adapter (or higher)

2.2 Running the Install Program

To install the Aircraft Recognition Tutor on your system:

1. Turn on your PC.
2. Type C: and press Enter.
3. Insert DISK 1 in drive A:
4. Type A:INSTALL and press Enter.
5. Follow the instructions on your screen.

The Aircraft Recognition Tutor is copy protected. It may only be installed on a single system. To move the program from one system to another repeat the install sequence on page 9, however for step 4, do the following:

4. Type A:UNSTALL and press Enter.

The Install and Unstall programs require that the disks NOT have write protect tabs on them. If you receive a "Write Protect Error" while using one of these programs, remove the write protect tab, reinsert the disk, and press "R" to retry.

To start the tutor once it has been installed, make sure that you are in the C:\ARTUTOR directory and type ART and press Enter.

3 Using the Tutor

3.1 General Information

The Aircraft Recogniton Tutor has a simple user interface. Few keys are required to use the system. Normally, when a screen is displayed to the user, the system will wait for the user to press any key before continuing. When a menu appears, the system will wait for the user to select a menu item and press Enter.

The following keys have a special meaning in the Aircraft Recognition Tutor:

Esc	Quit what you are doing. This normally brings you to a Menu one level higher than where you were.
H	Provides context sensitive help.
A	In the game mode, this is player 1's button.
L	In the game mode, this is player 2's button.
Enter	When no menu exists, this causes the program to continue after it has paused. When a menu exists, this selects the highlighted item.

↑	In a menu, this highlights the item above the one currently highlighted.
↓	In a menu, this highlights the item below the one currently highlighted.
PgUp	In a menu, this causes the previous 10 selections to be displayed in the menu.
PgDn	In a menu, this causes the next 10 selections to be displayed in the menu.
--	In the expert level, scrolls through the WEFT features in reverse order.
+	In the expert level, scrolls through the WEFT features in forward order.
C	When a Tutor session is completed, this continues with the next session.

In the Setup/Utility Mode, some of these keys have different meanings. See Chapter 5 for details.

3.2 Being Diagnosed

When a new student is encountered by the Aircraft Recognition Tutor (based on the User ID), the system first asks for the student's name, and then attempts to diagnose the student's level of proficiency at visual aircraft recognition.

The student is presented with 10 aircraft, one at a time, that are examples of specific WEFT features. The student is also given a menu that corresponds to the WEFT feature that the aircraft is exemplifying. The student is expected to identify the WEFT feature that corresponds to the menu title and is visible on the aircraft, and then select it using the menu.

If the student incorrectly responds to 2 or more of the 10, they will begin the tutor at the Novice level. If they miss 1 or less, they will begin at the Intermediate Level. Students are not allowed to begin at the Expert Level.

3.3 The Three Levels

The Aircraft Recognition Tutor allows students to be in one of three levels: Novice, Intermediate, and Expert.

The Novice Level is for students that are new to VACR and have not mastered WEFT theory.

The Intermediate Level is designed for students that have a solid background in recognizing

WEFT features. In this level, students learn to identify specific aircraft visually based on their WEFT components.

In the Expert Level, students must identify aircraft based solely on a WEFT description of the aircraft. No visual image of the aircraft is provided. This level provides a real challenge to even the best VACR students.

3.4 The Teach Mode

The Teach Mode is available in the Novice and Intermediate Levels. In the Novice Level, students are taught WEFT features. Intermediate Level students are taught WEFT features visible on specific aircraft.

The Aircraft Recognition Tutor displays an aircraft in the left window of the binoculars. Along with that aircraft is a description of a WEFT feature of the aircraft. The student should study the feature, and press Enter when done.

In the Novice Level, a new aircraft/WEFT feature will be shown, until the tutor has shown all of the WEFT features.

In the Intermediate Level, a single aircraft is shown, and the tutor displays all of the WEFT features of that aircraft before continuing with another aircraft. This is repeated until all of the aircraft that have been selected for the system to teach have been shown.

3.5 The Review Mode

The Review Mode randomly but completely presents each WEFT feature (Novice Level) or each aircraft (Intermediate and Expert Levels), asks the student to identify the WEFT feature or aircraft, and takes action based on the student's response.

During a review, three possible conditions can exist when a student attempts to identify a WEFT feature or aircraft:

(1) the student may respond correctly,
(2) the student may respond with an anticipated but incorrect answer.
(3) the student may respond with an unanticipated, incorrect answer.

Each of these conditions is handled differently by the tutor system.

In case (1), the tutor system will recognize the student for the correct answer, and present another aircraft to the student.

In case (2), the student is presented with both the aircraft being reviewed and the aircraft that the student selected from the menu. Both are identified to the student, and the system performs a comparison of the two for the student so that the differences are reinforced in the student's mind. The student will be required to demonstrate identification of the missed aircraft again some time later in the session. The tutor then continues with another aircraft.

In case (3), the aircraft is identified to the student, and the student is asked to identify specific features of the aircraft. The tutor session then continues as in (1) and (2). The student may have to identify a missed aircraft several times before the system is satisfied of the student's knowledge of that item.

3.6 The Test Mode

The Test Mode is available in the Intermediate and Expert Levels. Testing consists of presenting the student with each aircraft, asking the student to identify the aircraft, and maintaining a record of the student's performance.

A student must perform satisfactorily on the test in order to advance to the next level. In addition, poor performance may cause the student to revert to a lower level or mode.

A student at the Expert Level that receives a score of 100% will be deleted from the system and added to the Hall of Fame.

4 Playing the Game

4.1 One Player

The One-Player Game pits a student against the tutor system in a race to identify aircraft. The system opponent is based on an image recognition program that uses WEFT features to identify the aircraft. No unfair knowledge of the aircraft is available to the system opponent.

The player does not have to be enrolled in the tutor system in order to play the game, and performance in the game is not maintained in the student database.

The game is played as follows: An aircraft appears in the binoculars, and when the player recognizes the aircraft they press their "button". The "button" for the One-Player game is the 'A' key on the keyboard. The player is then given a chance to identify the aircraft in a menu that will appear. A limited amount of time is allowed for the player to recognize the aircraft. Also, the system opponent may recognize the aircraft first, and will be allowed to identify it.

A total of 25 aircraft comprise the game. Points are awarded for a correct response, and deducted for an incorrect response. After all 25 aircraft have been shown, the player with the highest score is the winner.

4.2 Two Players

The Two-Player game is played just like the One-Player game, but no system opponent exists. Instead, two players compete against each other. The "button" for Player 1 is the same as in the One-Player game, and the "button" for Player 2 is the 'L' key.

Again, the players need not be enrolled in the tutor system in order to play the game.

5 Using the Utilities

5.1 Password Protection

The Setup/Utility Mode of the Aircraft Recognition Tutor is intended to be used by the System Administrator (S-3). Because of this, access to this mode is provided only with a password. The password is included in this manual in an envelope attached to the back cover. If this password is lost, a new one can be requested by using the Suggestion Form located in Appendix C.

When the Setup/Utility Mode is selected from the Main Menu, the user is asked to enter their password. Once the password in entered correctly, the Setup/Utility Menu is presented. If the password is entered incorrectly, the system returns to the Main Menu.

5.2 Choosing the Aircraft

The Select Aircraft Utility allows the System Administrator to select the aircraft that will be taught by the system. Initially, this includes all of the aircraft listed Appendix A of this manual.

To modify this list, use the menu to select the aircraft that you want to include in the system. This aircraft will be added to the list and you will be presented with an updated menu of aircraft. Continue the selection process until all of the aircraft that you want to be taught by the system have been selected. Press 'Esc' when you are finished.

Caution: Once you start this utility, the list of aircraft used by the system is erased. You must select ALL of the aircraft that you want included in the system.

5.3 Adding or Modifying Aircraft

Adding or modifying aircraft is a complex procedure. Once you begin defining an aircraft, you must complete all of the steps listed below. An aircraft definition consists of three views: front, side, and bottom.

To add or modify an aircraft:

1. Select "Add/Modify Aircraft" from the Setup/Utility Menu.

2. When prompted, type in the nomenclature and name of the aircraft as you want it to appear (or as it already appears, if you want to modify the aircraft definition) in the Aircraft Menu. This name will appear for a brief moment in the lower left portion of the screen.
3. If the aircraft that you named already exists, it will be drawn on the screen in the upper left corner, and then translated (2 times larger) on the right side of the screen.
4. Position the cursor inside the circular area, using the arrow keys. This circle corresponds to the binocular window.
5. Draw within the circle by pressing the space bar when the cursor is positioned where you want to draw. You can erase in the same manner.
6. When you are finished drawing the aircraft, press 'Enter'.
7. The aircraft will then be reduced and drawn in the upper left corner of the screen.
8. A menu will appear. This menu will

contain selections for a particular WEFT feature. Highlight the appropriate feature visible in this view of the aircraft and press 'Enter'. If the WEFT feature is not visible in this view of the aircraft, press 'Esc'.

9. Repeat step 8 for each of the WEFT feature menus.
10. Repeat steps 3-8 for the other two views of the aircraft.

The aircraft that you defined is not automatically added to the list of aircraft taught by the system. To include new aircraft, choose the "Select Aircraft" option from the Setup/Utility Menu.

If you make an error in defining an aircraft, simply modify the definition using the same method described above.

Aircraft are identified by the tutor system based on the first four characters in the name. To define a new aircraft with a similar name as an existing one, insure that the first four characters of the name you enter in step 2 are unique.

5.4 Getting a Student Report

Reports are available for each student that uses the tutor system. To get a report on a student:

1. Select "Student Report" from the Setup/ Utility Menu.
2. Select the student that you want a report on based on their User ID from the menu shown.
3. Repeat for other students as desired. Press 'Esc' when you are finished with this utility.

The Student Report provides the following information about a student:

1. The Student's User ID.
2. The Student's Name.
3. The current Mode the student is in.
4. The current Level the student is at.
5. The last test score the student received.

5.5 Deleting a Student

Students may be deleted from the tutor system as necessary. This is useful when a student PCS's and no longer needs to be maintained in the student database. To delete a student:

1. Select "Delete Student" from the Setup/ Utility Menu.
2. Select the student to be deleted based on their User ID from the menu shown.
3. The student will be deleted and an updated menu will be shown. Repeat step 2 to delete additional students.
4. Press 'Esc' when you are finished with this utility.

When a student receives a test score of 100% in the Test/Expert Mode of the tutor, they are automatically deleted from the student database and added to the Hall of Fame.

6 Getting Help!

6.1 General Help

General Help for the Aircraft Recognition Tutor is available from the Main Menu by selecting "Get Help!". This menu selection causes the General Help Menu to appear. Help is available from this menu on the following topics:

1. About Help! - Information about the Help available in the tutor system.
2. Tutor Help! - An overview of the tutor system.
3. Game Help! - An overview of the 1 or 2 Player Game.
4. Setup/Utility Help! - A description of the various utilities available in the system.

6.2 Context Sensitive Help

Context Sensitive Help is available from almost anywhere in the system by pressing 'H' at any time. Context Sensitive Help provides detailed information about the current Mode/Level that the user was in at the time of the Help request.

If pressing the 'H' key does not bring up a Help screen, insure that the CAPS LOCK key is up and try again.

Appendix A

List of Aircraft in the System

The following aircraft are included in the Aircraft Recognition Tutor:

- A-4 Skyhawk
- A-6 Intruder
- A-7 Corsair II
- A-10A Thunderbolt II
- Alpha Jet
- AM-X
- AV-8 Harrier
- Buccaneer
- Draken
- F-4 Phantom
- F-5 Freedom Fighter
- F-14 Tomcat
- F-15 Eagle
- F-16 Fighting Falcon
- F/A-18 Hornet
- F-20 Tigershark
- F-86 Sabrejet
- F-100 Super Sabre
- F-104 Starfighter
- F-111
- Fantan A
- G.91Y
- Galeb
- Hawk
- Hunter
- Jaguar
- Lightning
- MiG-17 Fresco
- MiG-19 Farmer
- MiG-21 Fishbed
- MiG-25 Foxbat
- MiG-27 Flogger D
- MiG-29 Fulcrum
- Mirage III/5

Mirage F1	Super Etendard
Su-7B Fitter A	Tornado
Su-17,20,22 Fitter	Viggen AJ-37
Su-24 Fencer	Yak-28 Brewer
Su-25 Frogfoot	Yak-36 Forger

Appendix B

References

1. Weidman, LtCol J. D. and Harwood, LtCol W. R. (1985). Fratricide. USAF Weapons Review, vol 33, pp. 16-20.

2. Pliler, J. R. (1984). Aircraft Recognition Skills Demand Attention. Air Defense Artillery, Fall 84, pp. 14 -16.

3. Headquarters, Department of the Army (1986). FM 44-30 Visual Aircraft Recognition, October 1986.

4. Baron, R. J. (1987). The Cerebral Computer.

5. Guindon, R. (1988). Cognitive Science and its Applications for Human-Computer Interaction.

6. Papert, S. (1980). Mindstorms.

7. Bower, G. H. and Hilgard, E. R. (1981).

Theories of Learning.

8. Jonassen, D. H. (1988). Instructional Designs for Microcomputer Courseware.

9. Kearsley, G. (1987). Artificial Intelligence and Instruction.

10. Godfrey, D. and Sterling, S. (1982). The Elements of CAL.

11. Sleeman, D. and Brown, J. S. (1982). Intelligent Tutoring Systems.

Appendix C

Suggestion Form

Mail to: CPT Larry W. Campbell
SMC 2269
Naval Postgraduate School
Monterey, CA 93940

EMail at: Campbell@NPS.CS.NAVY.MIL

Comments/Suggestions:

Bug Report:

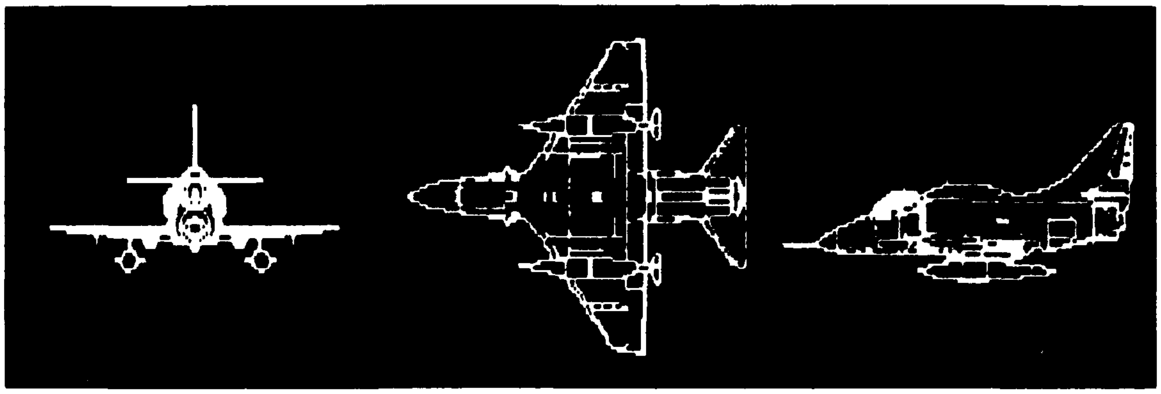
Mode in when bug was detected:

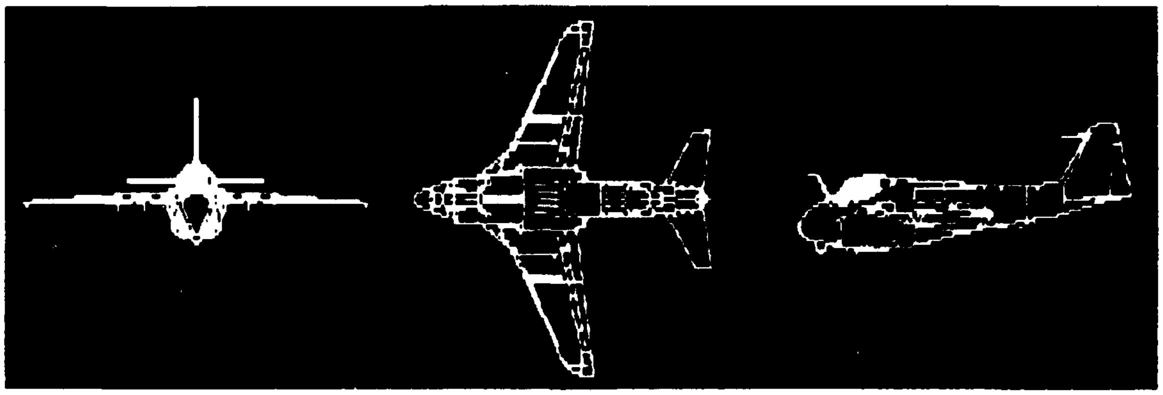
Your Name, Address, and Phone Number (Autovon if available)

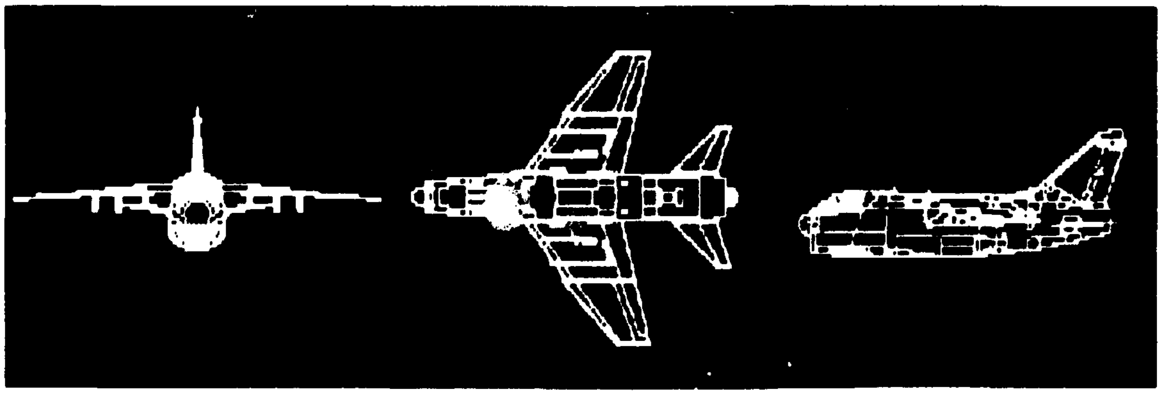
Index

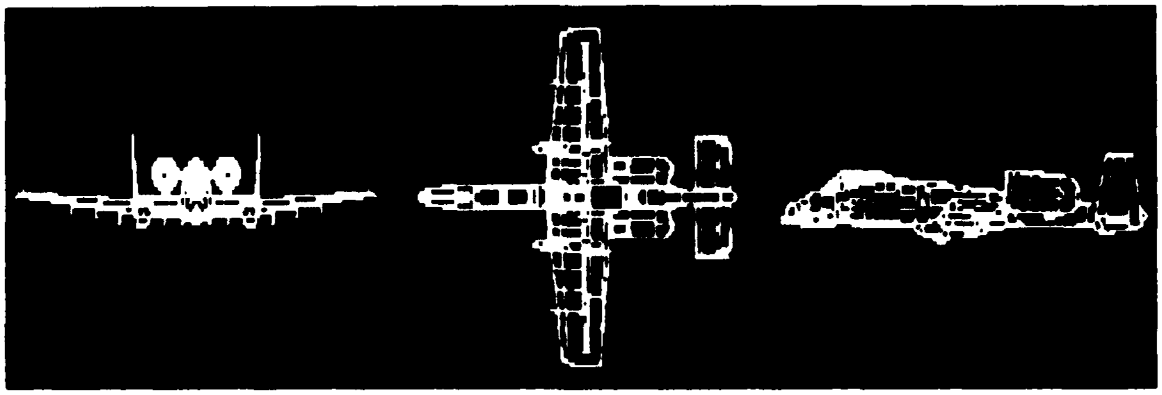
Notes

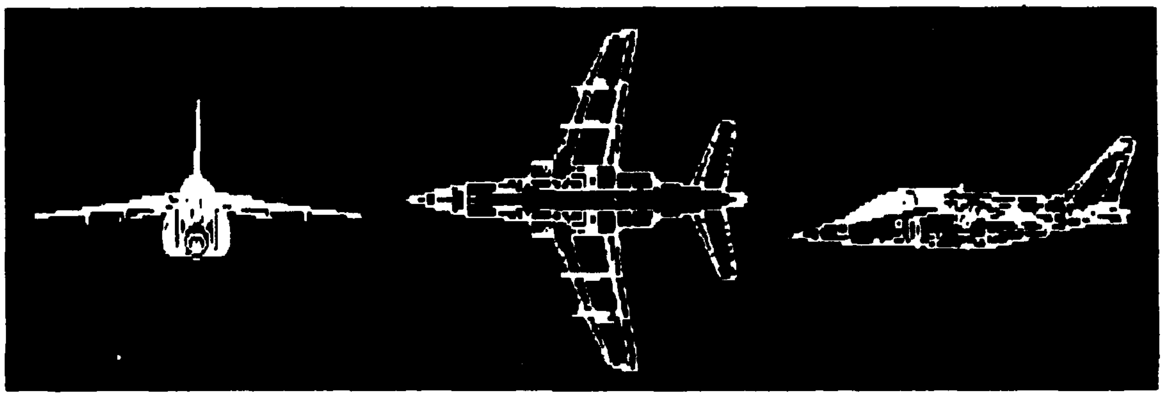
APPENDIX C - AIRCRAFT IMAGES

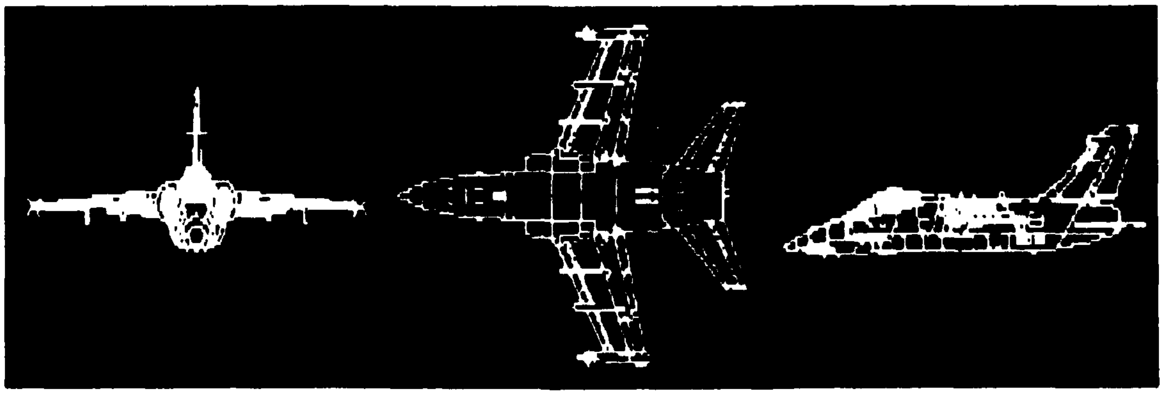

A-4 Skyhawk

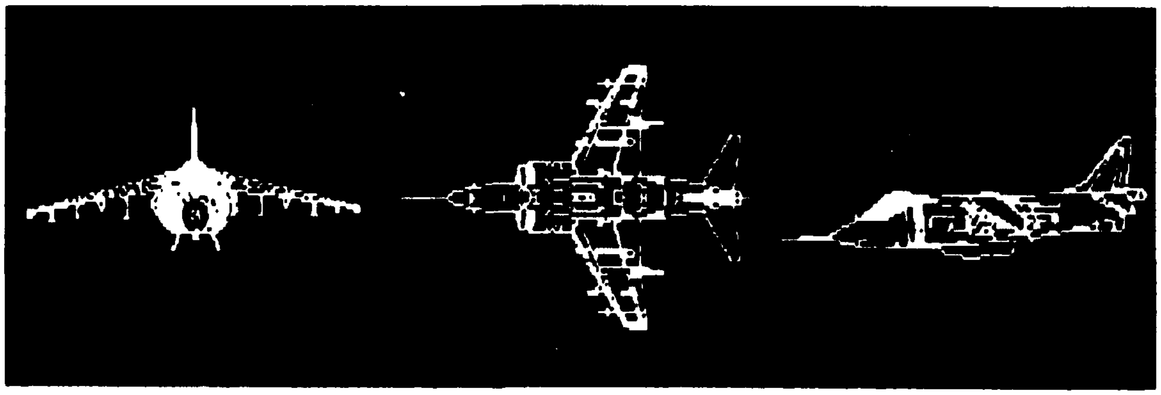

A-6 Intruder

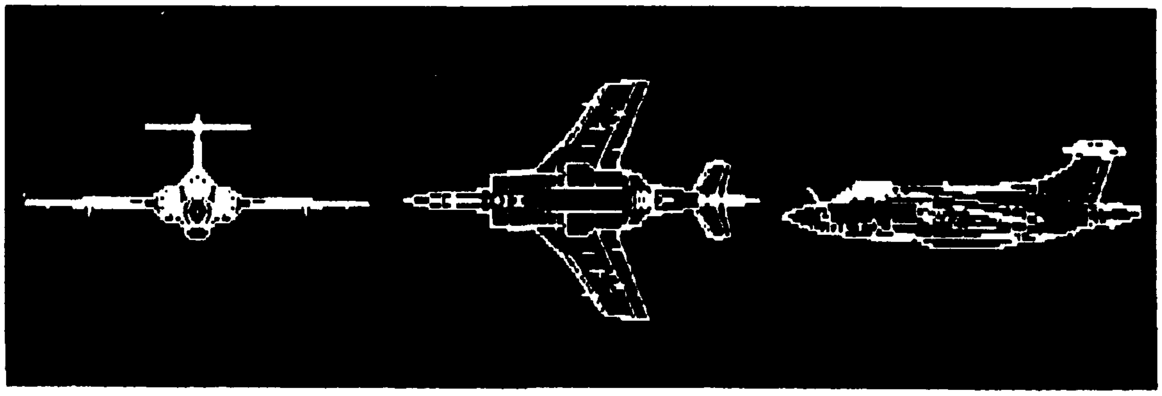

A-7 Corsair II

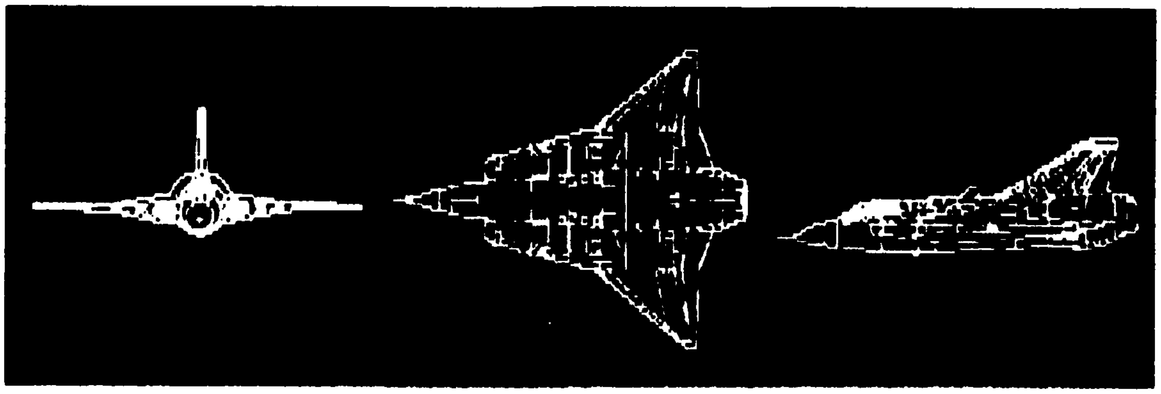

A-10A Thunderbolt II

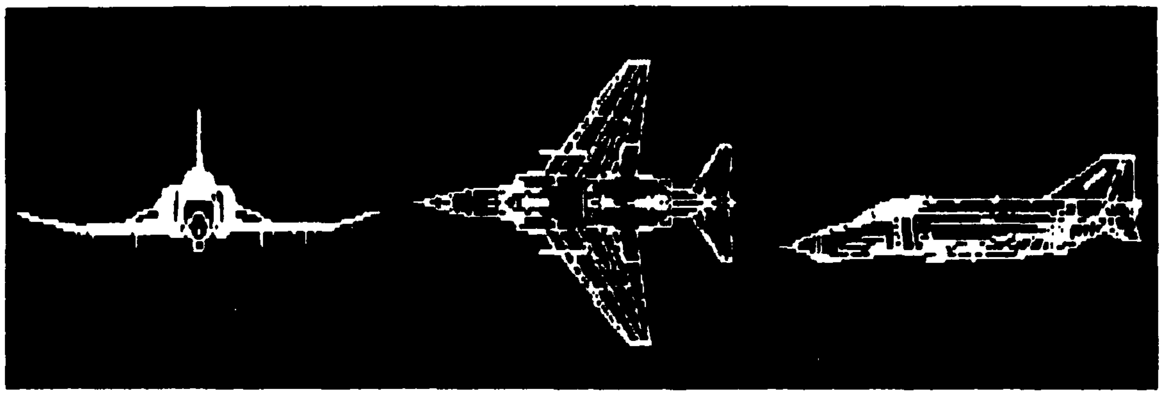

Alpha Jet

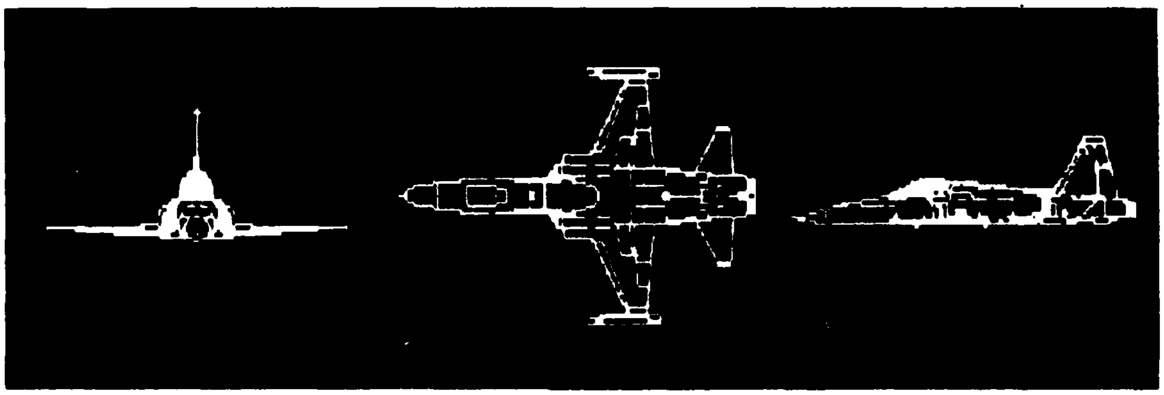

AM-X

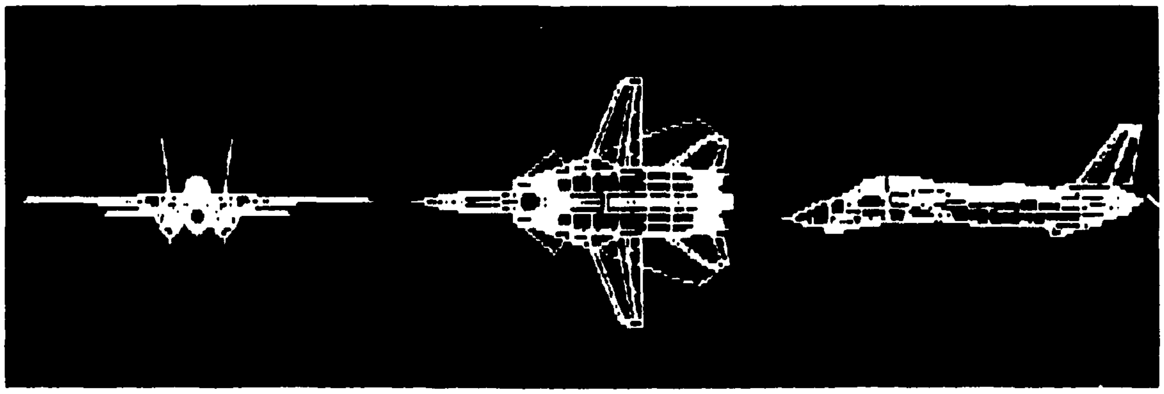

AV-8 Harrier

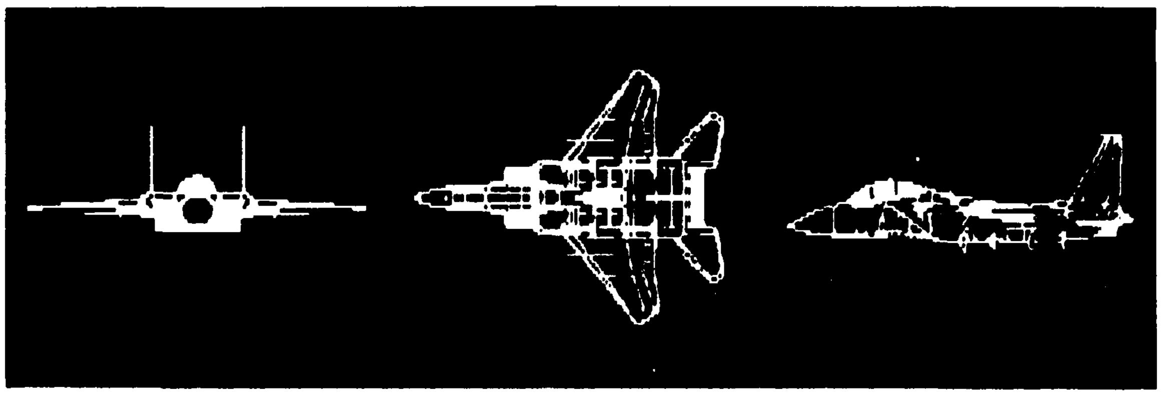

Buccaneer


Draken

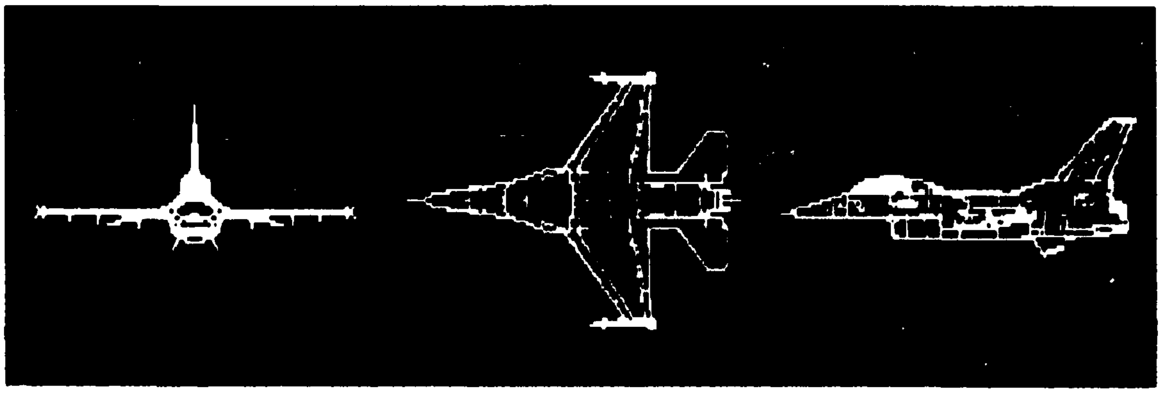

F-4 Phantom


F-5 Freedom Fighter

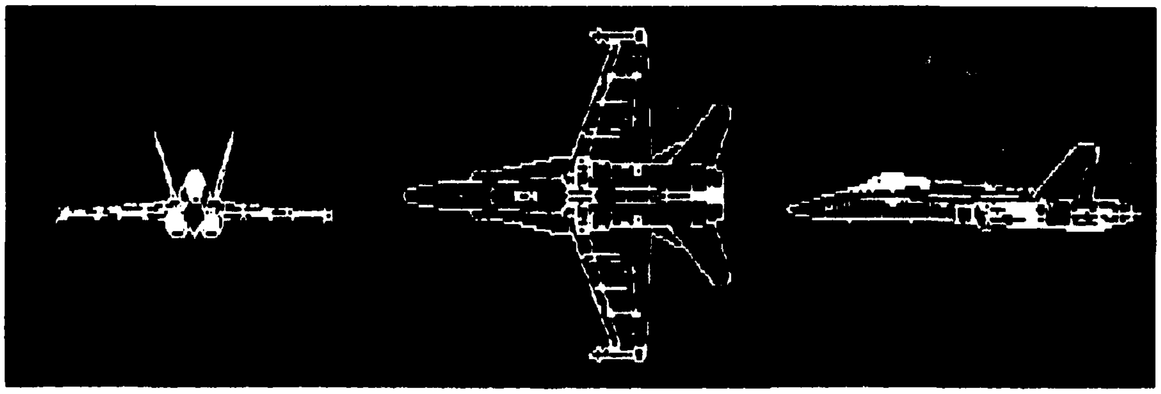

F-14 Tomcat



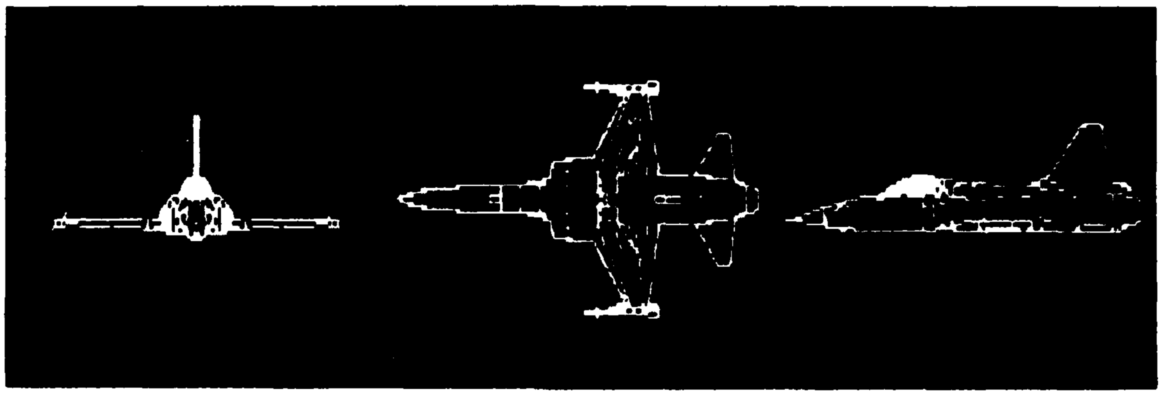
F-15 Eagle

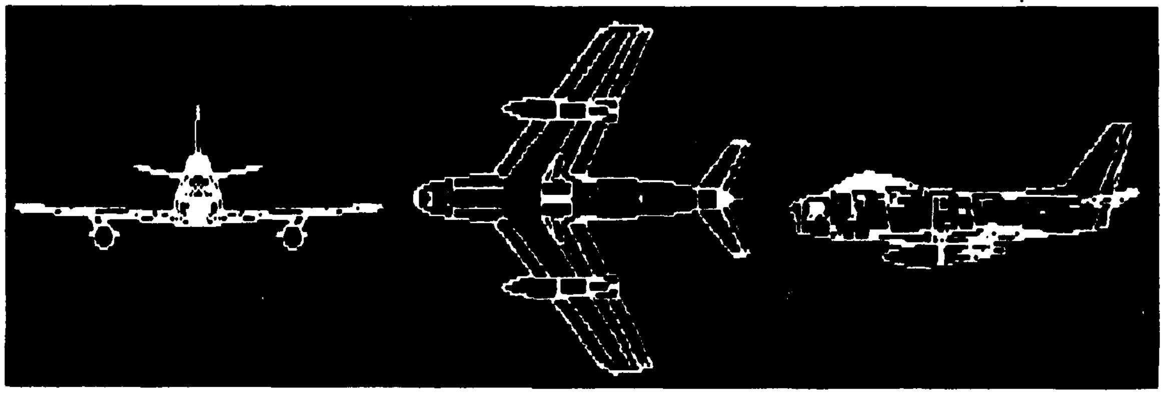


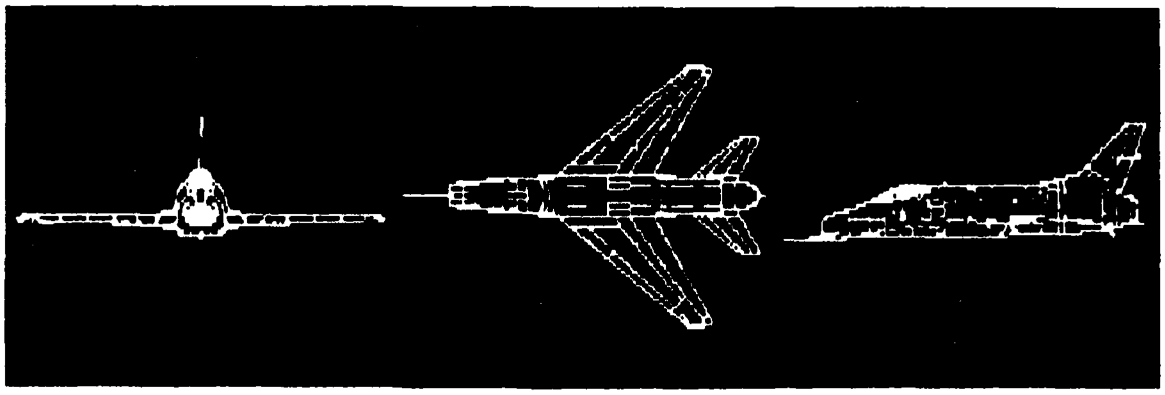
F-16 Fighting Falcon

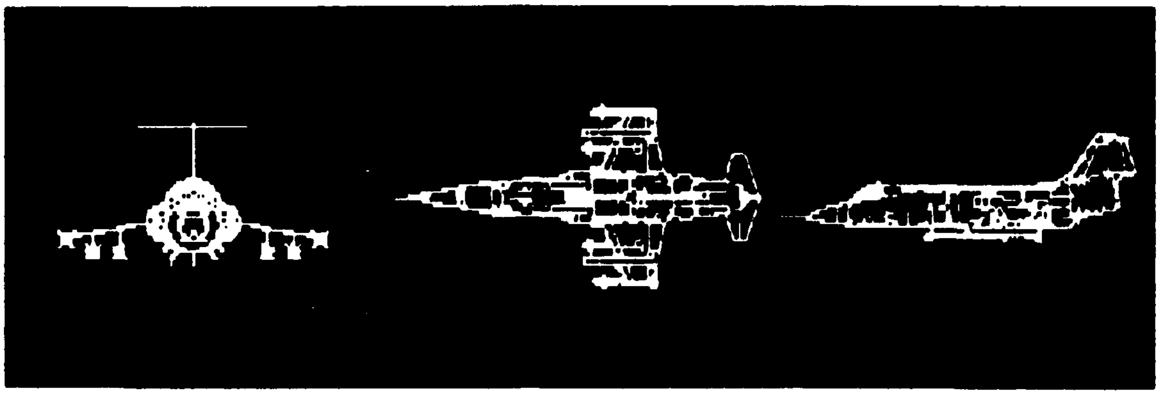


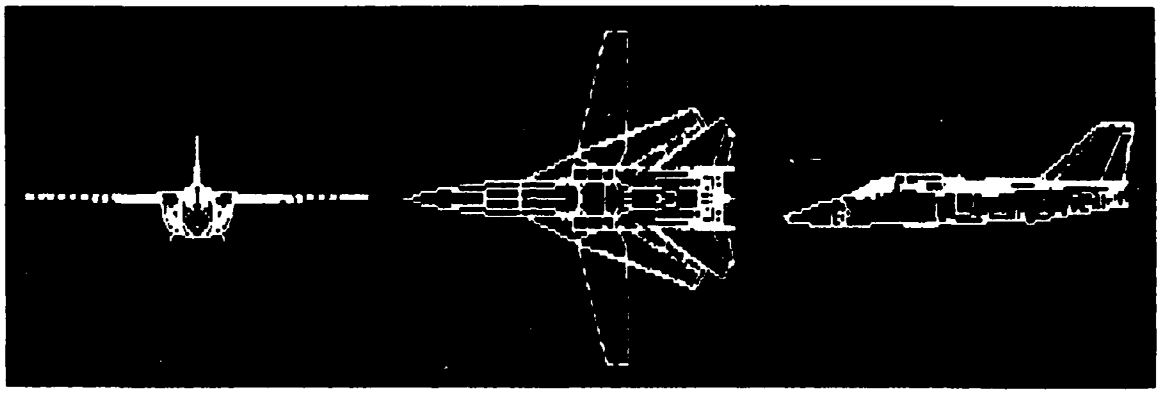
F/A-18 Hornet

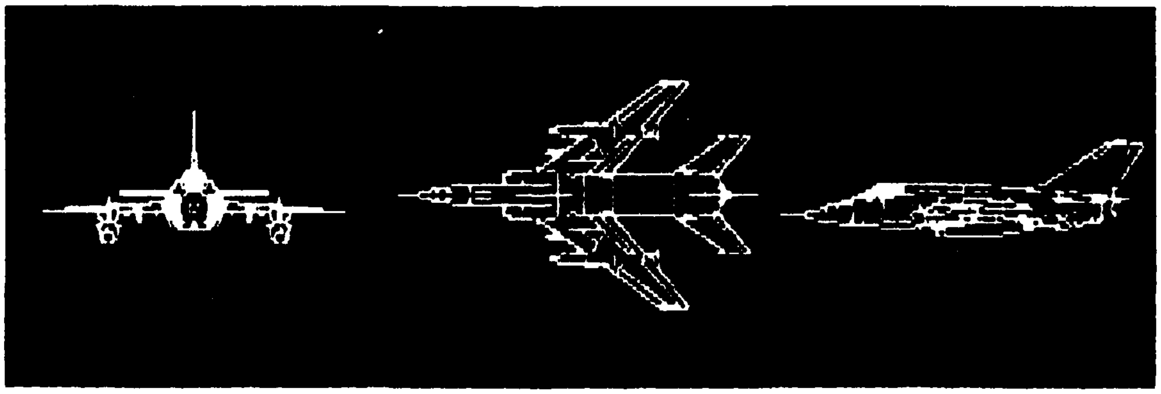

F-20 Tigershark

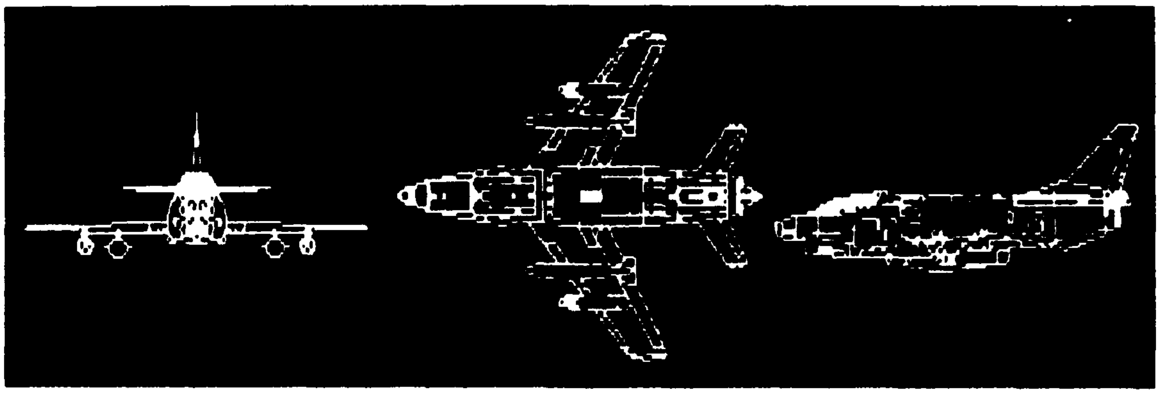

F-86 Sabrejet

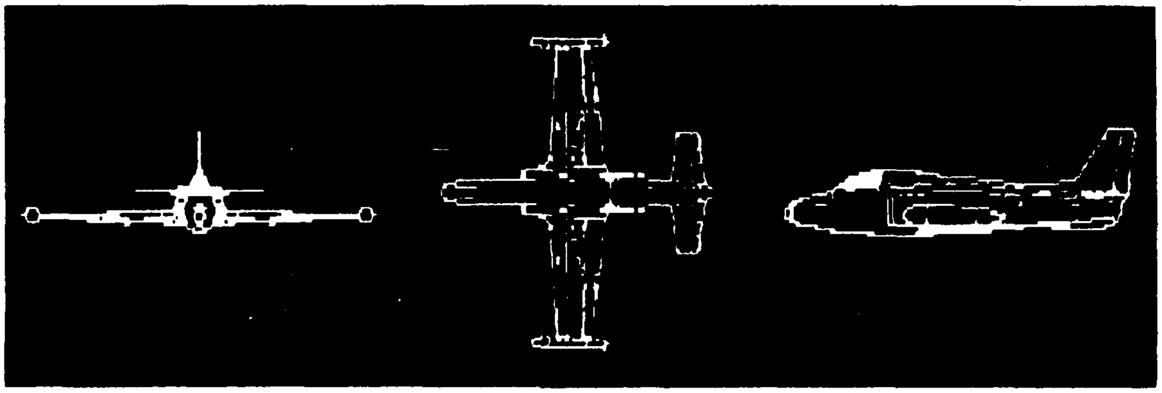

F-100 Super Sabre

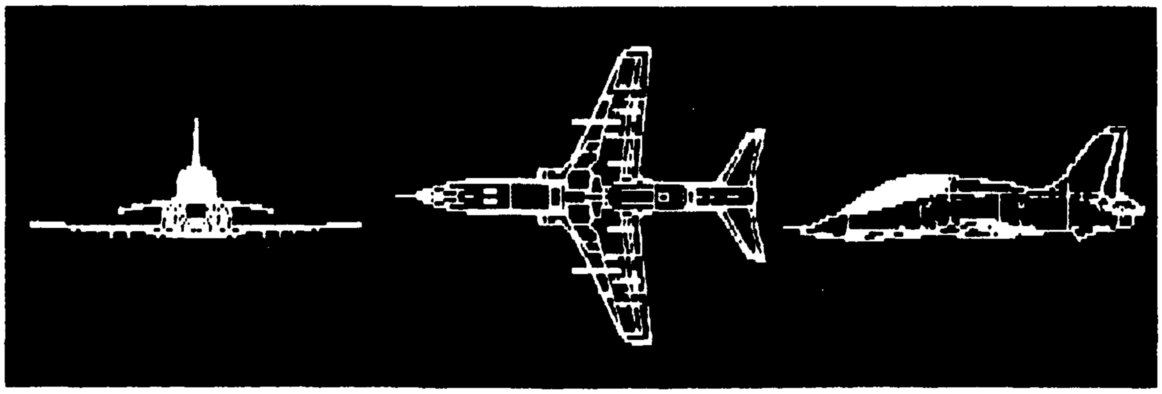

F-104 Starfighter

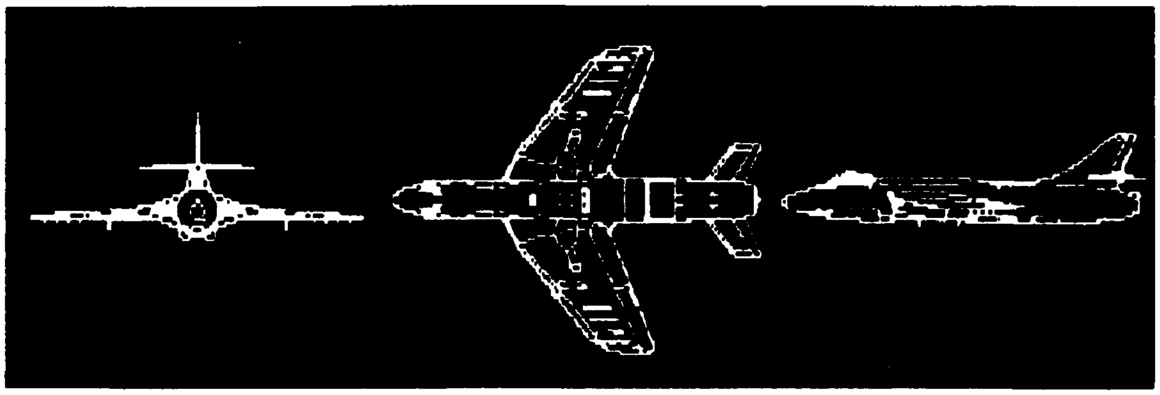

F-111


Fantan A

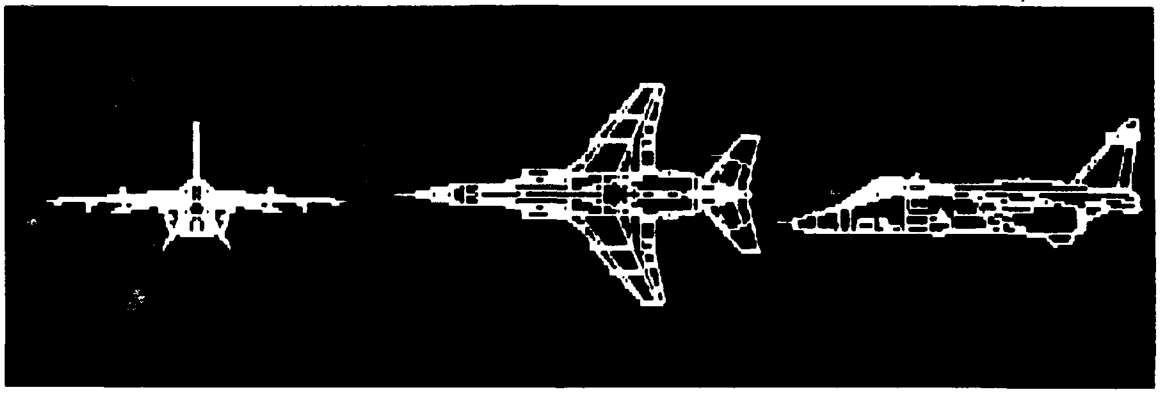

G.91Y


Galeb

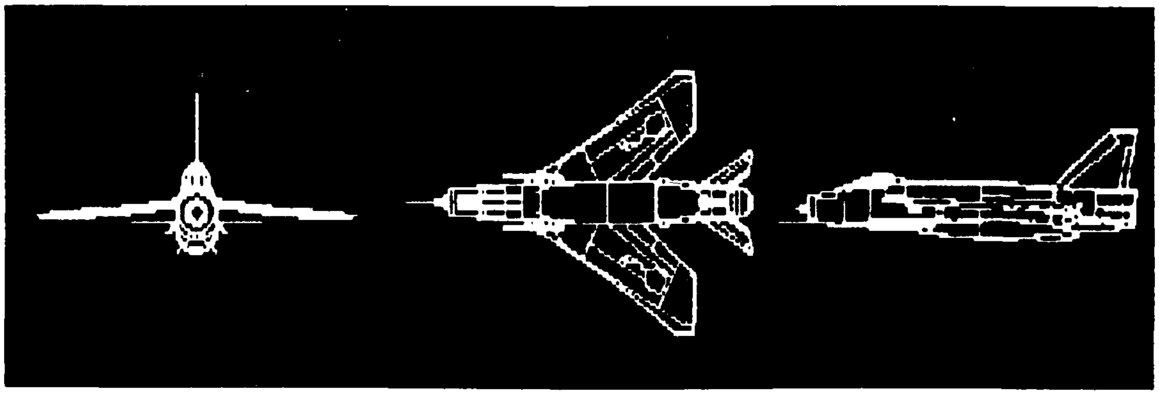

Hawk



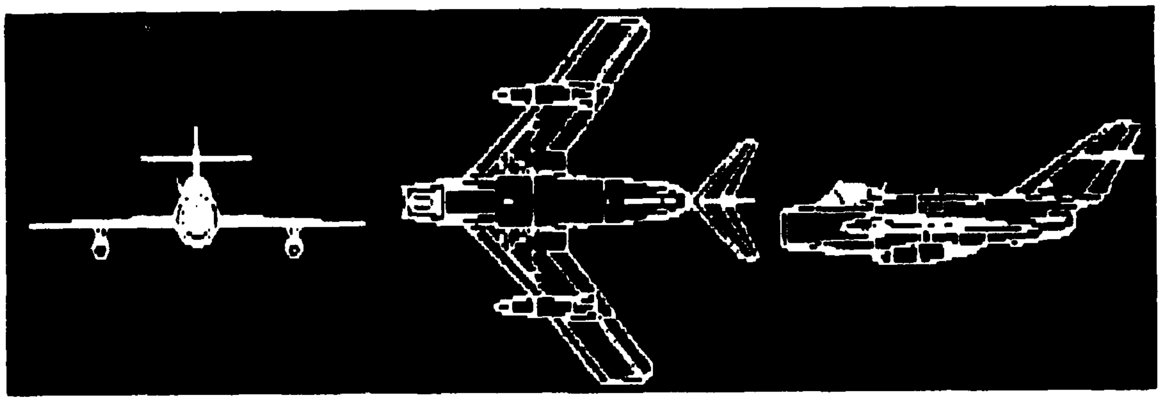
Hunter

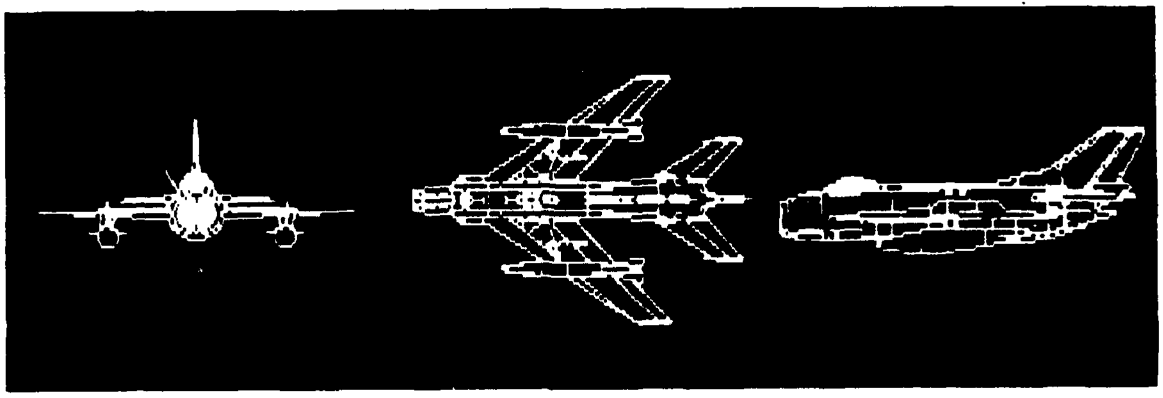


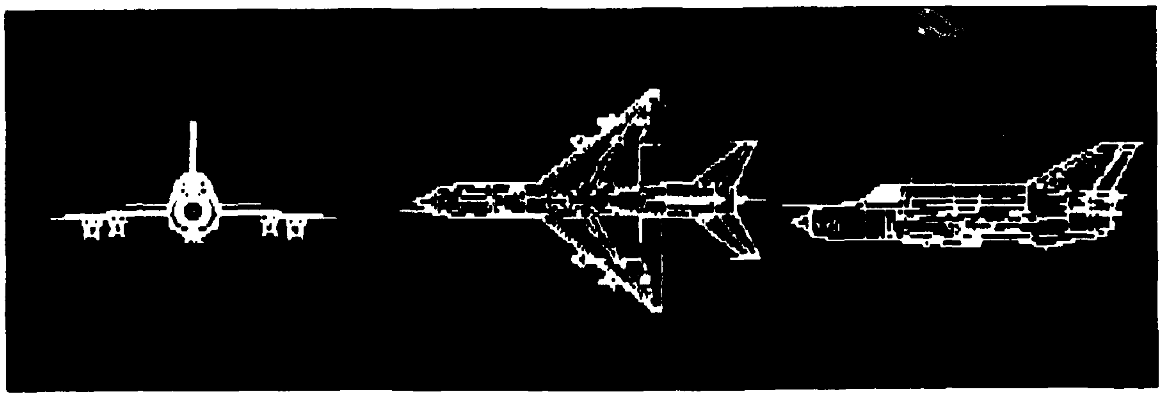
Jaguar

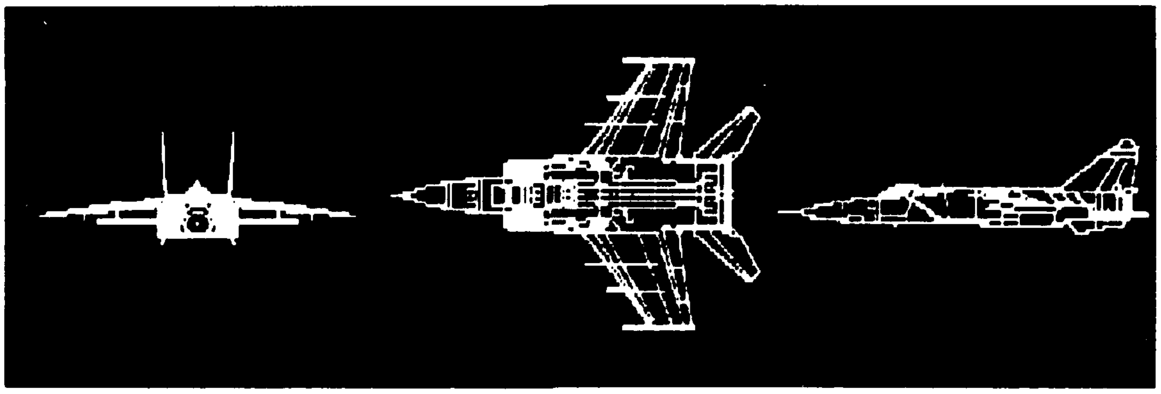


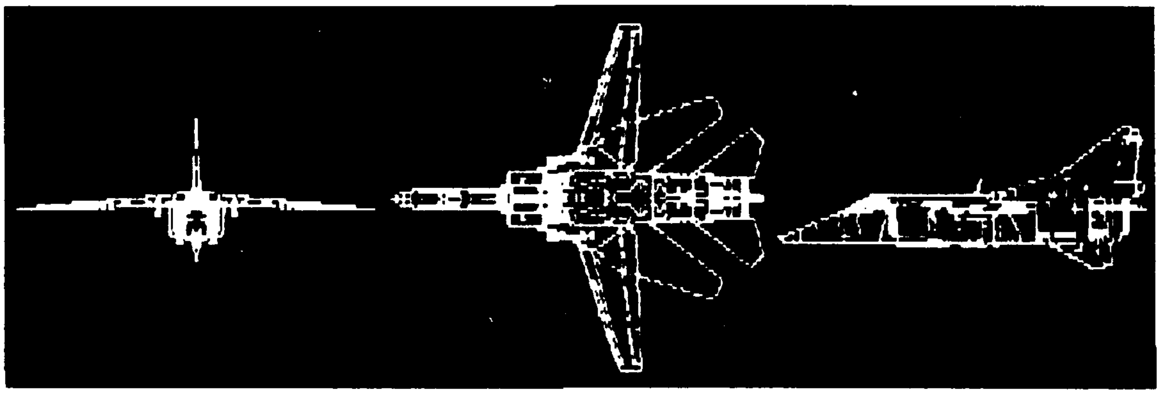
Lightning

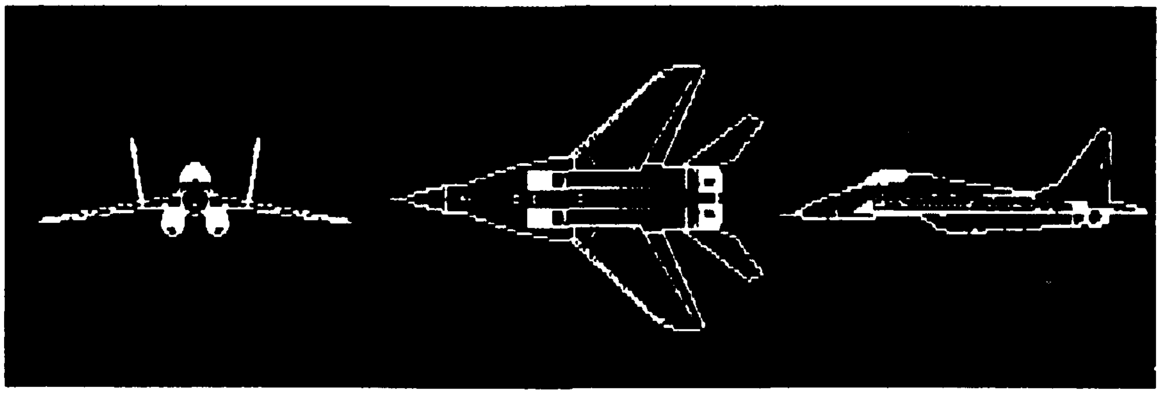

MiG-17 Fresco

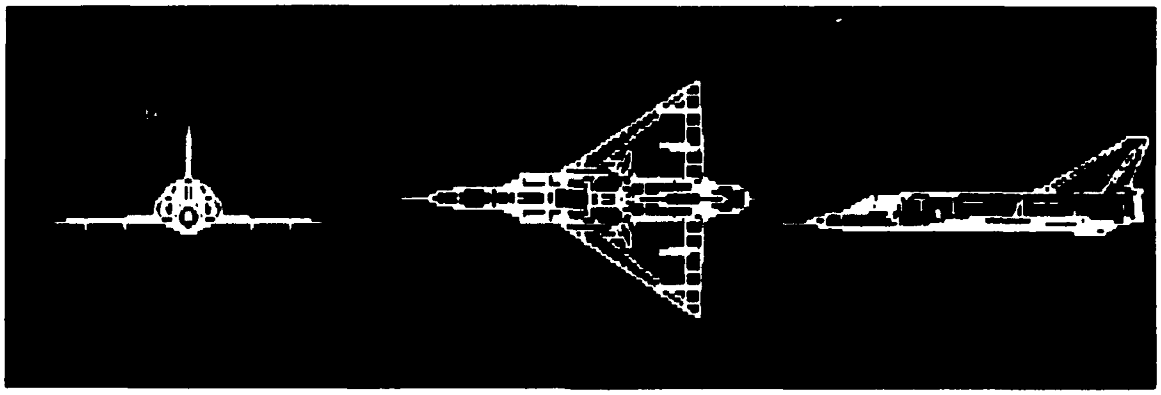

MiG-19 Farmer

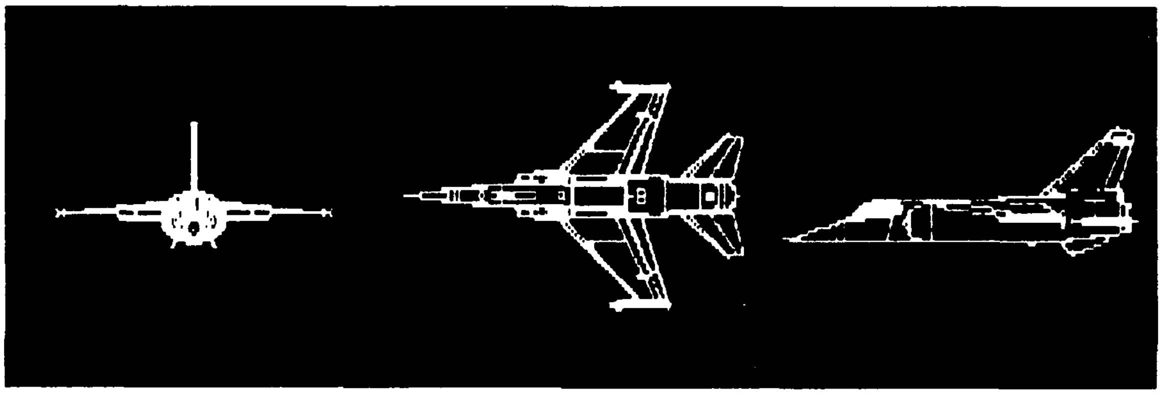

MiG-21 Fishbed

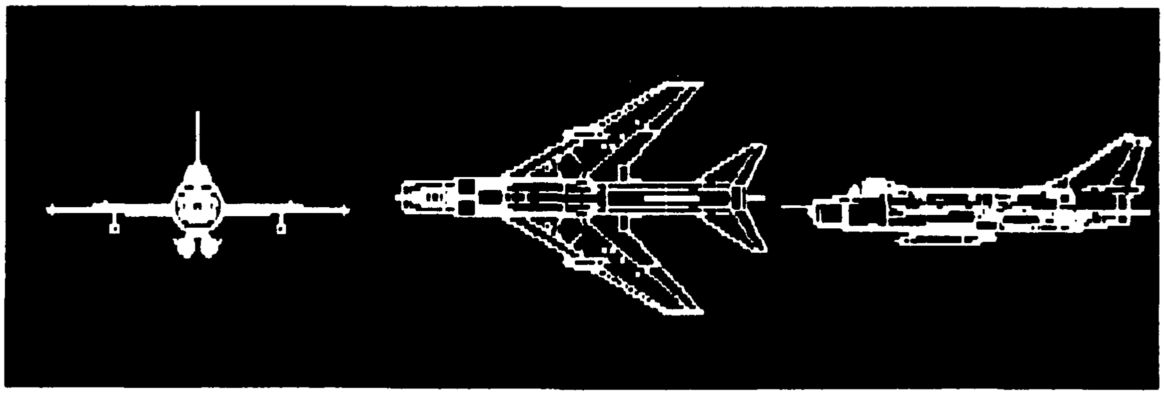

MiG-25 Foxbat

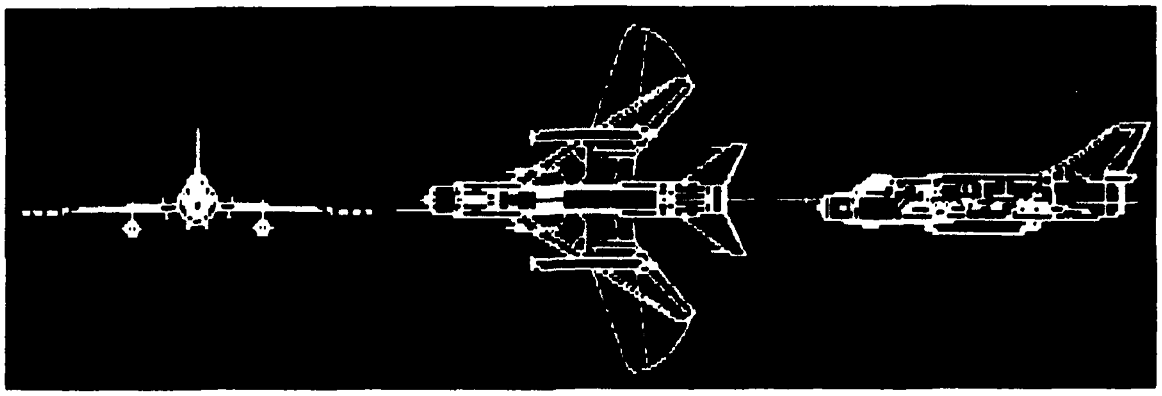

MiG-27 Flogger D

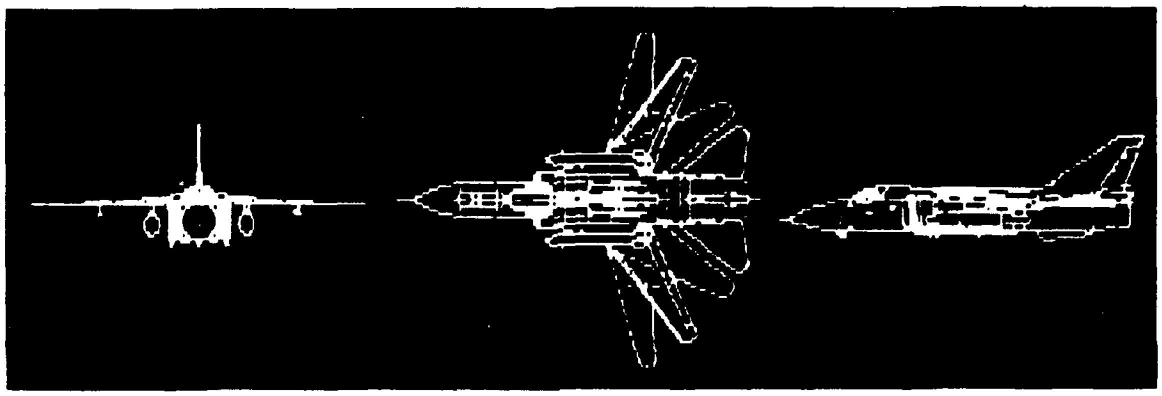

MiG-29 Fulcrum

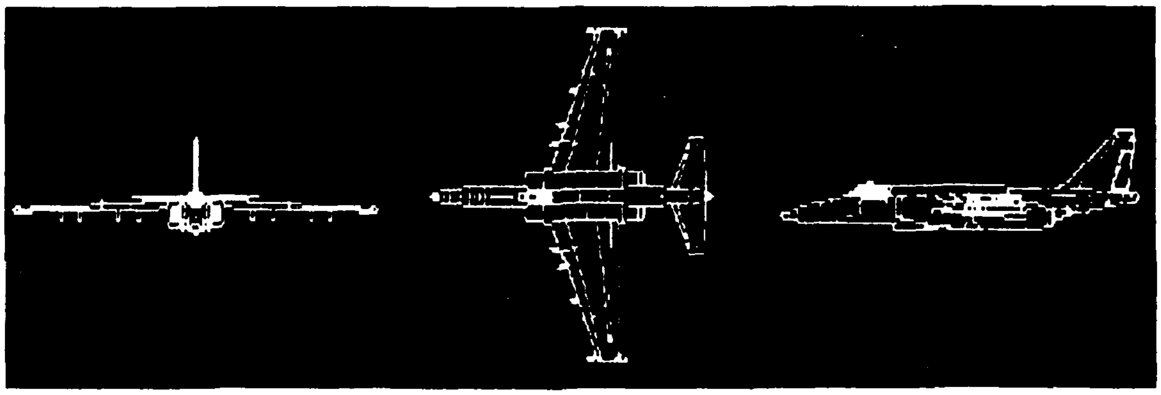

Mirage III/5

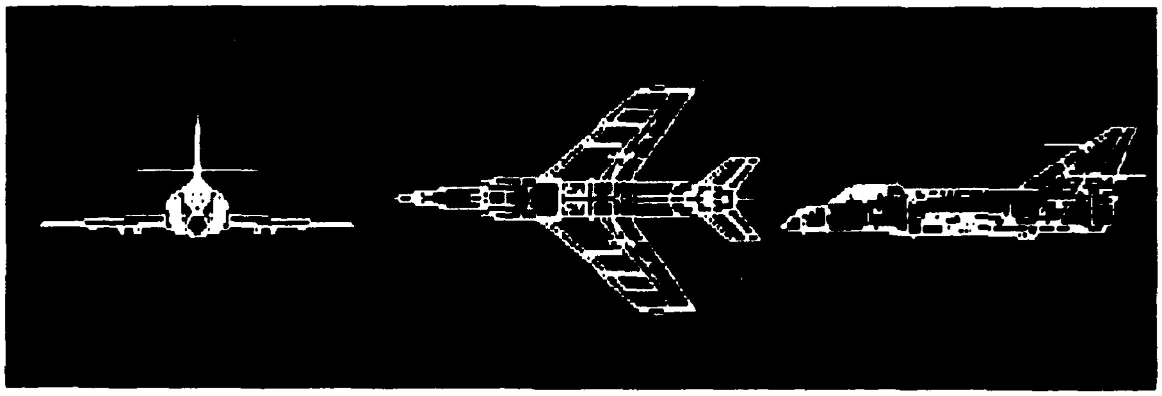

Mirage F-1


Su-7B Fitter A

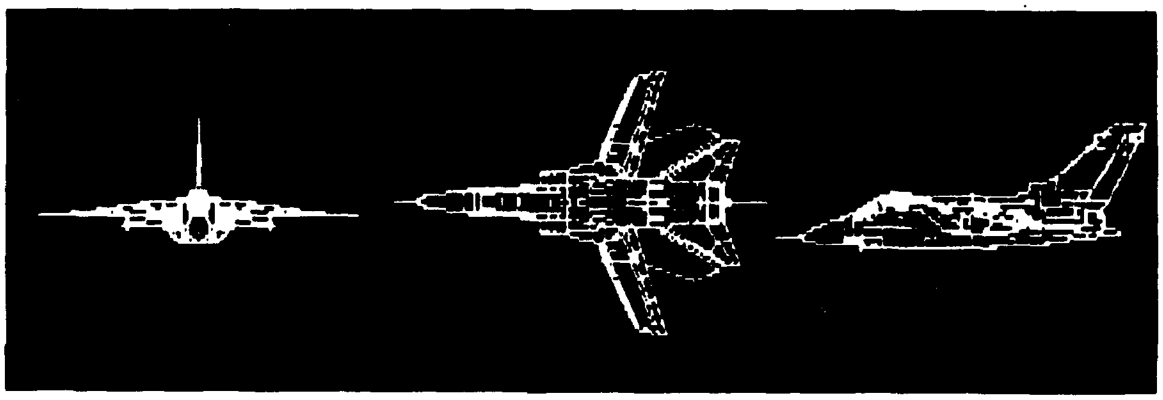

Su-17, 20, 22 Fitter


Su-24 Fencer

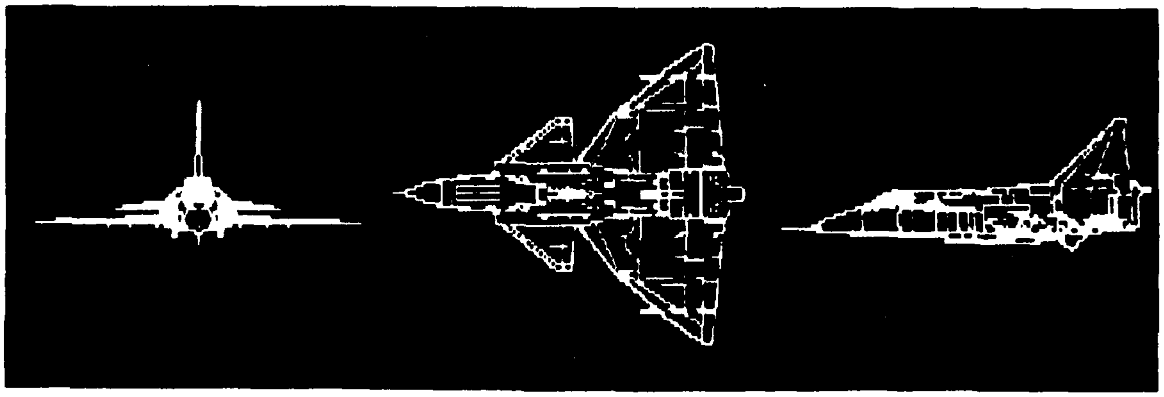

Su-25 Frogfoot



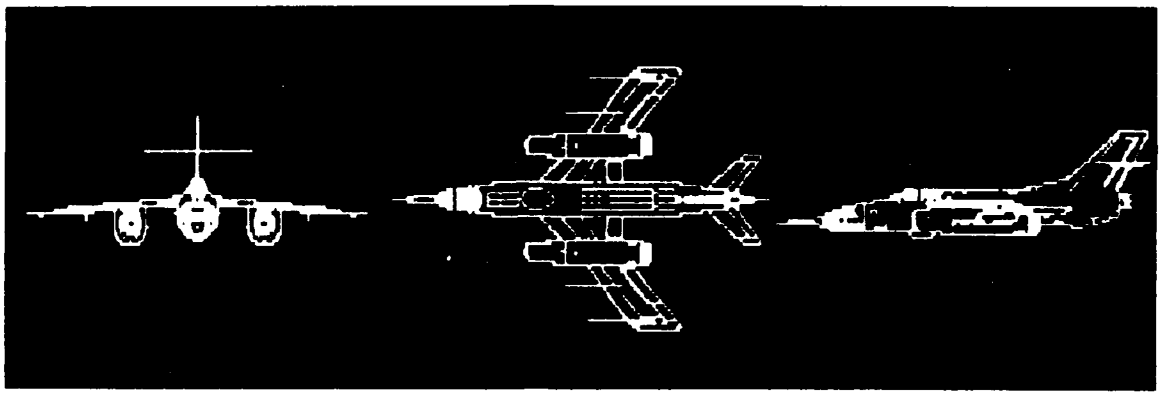
Super Etendard



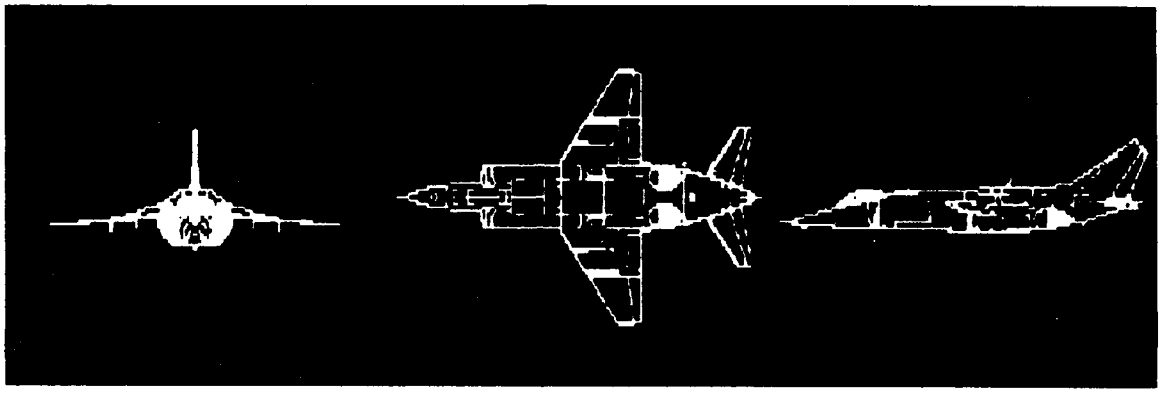
Tornado



Viggen AJ-37



Yak-28 Brewer



Yak-36 Forger

BIBLIOGRAPHY

1. Weidman, LtCol J. D. and Harwood, LtCol W. R., "Fratricide" *USAF Weapons Review*, vol 33, pp. 16-20, Summer 1985.

2. Pliler, J. R., "Aircraft Recognition Skills Demand Attention" *Air Defense Artillery*, pp. 14 -16, Fall 1984.

3. Pliler, J. R., "Recognition List Grows", *Air Defense Artillery*, pp. 38-39, January-February 1988.

4. Headquarters, Department of the Army, Field Manual 44-30, *Visual Aircraft Recognition*, pp. 1.1-5.138, Government Printing Office, Washington, DC, October 1986.

5. Wood, R., *Jane's World Aircraft Recognition Handbook*, pp. 5-55, Jane's Publishing Company, 1985.

6. Harmon, P., *Intelligent Job Aids: How AI Will Change Training in the Next Five Years*, pp. 165-190, Addison-Wesley Publishing Company, 1987.

7. Kearsley, G., *Artificial Intelligence and Instruction*, pp. 1-10, Addison-Wesley Publishing Company, 1987.

8. Bower, M. and Hilgard, J., *Theories of Learning*, pp. 1-511, Addison-Wesley Publishing Company, 1981.

9. Park, O. Perez, R. S., and Seidel, R. J., *Intelligent CAI: Old Wine in New Bottles, or a New Vintage?*, pp. 11-40, Addison-Wesley Publishing Company, 1987.

10. Goldstein, I. P., *The Genetic Graph: A Representation for the Evolution of Procedural Knowledge*, pp. 51-75, Academic Press, 1982.

11. Woolf, B., *Intelligent Tutoring Systems, A Survey*, pp. 1-44, Morgan Kaufmann Publishers, 1988.

12. Sleeman, D. and Hendley, R. J., *ACE: A System Which Analyses Complex Explanations*, pp. 99-116, Academic Press, 1982.

13. Matz, M., *Towards a Process Model for High School Algebra Errors*, pp. 25-49, Academic Press, 1982

14. Johnson W. L. and Soloway E., *PROUST: An Automatic Debugger for Pascal Programs*, pp. 49-67, Addison-Wesley Publishing Company, 1987.

15. Papert, S., *Mindstorms*, pp. 3-190, Basic Books, 1980.

16. Clancey, W. J., *Tutoring Rules for Guiding a Case Method Dialogue*, pp. 201-222, Academic Press, 1982.

17. Burton, R. R. and Brown, J. S., *An Investigation of Computer Coaching for Informal Learning Activities*, pp. 79-97, Academic Press, 1982.

18. Iano, R. P., *Is Education a Science? No Way!*, pp. 256-271, W. H. Freeman and Company, 1987.

19. Godfrey, D. and Sterling, S., *The Elements of CAL*, pp. 1-281, Reston Publishing Company, 1982.

20. Gallant, J., "Speech Recognition Products", *EDN*, pp. 1-8, January 19, 1989.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 0142 — 2
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Headquarters — 1
   Department of the Army
   Training and Doctrine Command
   Fort Monroe, Virginia 23651-5000

4. Commandant — 1
   US Army Air Defense Artillery School
   Fort Bliss, Texas 79916-7004

5. Commander — 1
   1/62 Air Defense Artillery Battalion
   25th Infantry Division (Light)
   Schofield Barracks, Hawaii 96857-6051

6. Commander — 1
   2/62 Air Defense Artillery Battalion
   7th Infantry Division (Light)
   Fort Ord, California 93941

7. Dr. Yuh-jeng Lee — 50
   Naval Postgraduate School
   Code CS, Department of Computer Science
   Monterey, California 93943-5000

8. CPT Larry W. Campbell — 2
   Suite #124
   12917-H Jefferson Avenue
   Newport News, Virginia 23602

9. Dr. Man-Tak Shing — 1
   Naval Postgraduate School
   Code CS, Department of Computer Science
   Monterey, California 93943-5000

10. Ong Seow Meng — 1
    #09-102, Block 272
    Yishun St. 22
    Singapore 2776
    Republic of Singapore