ED 215 662	IR 010 027
AUTHOR	Frye, Charles H.
TITLE	Extension of Computer-Assisted Team Training Through Coordinated Lesson Scenario.
INSTITUTION	Northwest Regional Educational Lab., Portland, Oreg.
SPONS AGENCY	Army Research Inst. for the Behavioral and Social Sciences, Arlington, Va.
REPORT NO	ARI-RR-1285
PUB DATE	Sep 77
CONTRACT	DAHC19-76-C-0042
NOTE	84p.
EDRS PRICE	MF01/PC04 Plus Postage.
DESCRIPTORS	*Computer Assisted Instruction; Computer Programs; *Instructional Development; Military Training; Models; *Programing; *Team Training; Units of Study
IDENTIFIERS	*Authoring Aids; *PLANIT Programing Language

ABSTRACT

This research report discusses several efforts which were undertaken to assist PLANIT (Programming Language for Interactive Teaching) authors in the writing of team lesson scenarios. It also describes and illustrates the assistance given to two such authors who were engaged in the development of computer assisted instruction for training teams of tactical data system users. The text of the document first sets forth a definition of team training which is meant to be a frame of reference for the authoring features later discussed. After this, several specific authoring strategies are presented which enabled the kind of lesson presentation desired by the authors. A demonstration lesson is described which was developed to quickly show PLANIT team training concepts. The rationale for the lesson is described in the text and the lesson itself is appended as well as two other products of the effort: a set of recommended modifications to the PLANIT language designed to simplify the team authoring process, and a detailed set of authoring guidelines to help conventional PLANIT authors become "team" authors. (Author/LLS)

ED215662

Research Report 1285

EXTENSION OF COMPUTER-ASSISTED TEAM TRAINING THROUGH COORDINATED LESSON SCENARIO

Charles H. Frye
The Northwest Regional Educational Laboratory

Submitted by:
James D. Baker, Chief
MANPOWER AND EDUCATIONAL SYSTEMS TECHNICAL AREA

Approved by:
Edgar M. Johnson, Director
ORGANIZATIONS AND SYSTEMS RESEARCH LABORATORY

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES
5001 Eisenhower Avenue, Alexandria, Virginia 22333

Office, Deputy Chief of Staff for Personnel
Department of the Army

September 1977

Army Project Number
2Q762722A764

Embedded Training

IR010027

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Research Report 1285	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) EXTENSION OF COMPUTER-ASSISTED TEAM TRAINING THROUGH COORDINATED LESSON SCENARIO		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Charles H. Frye		8. CONTRACT OR GRANT NUMBER(s) DAHC19-76-C-0042
9. PERFORMING ORGANIZATION NAME AND ADDRESS Northwest Regional Educational Laboratory, 710 S.W. Second Avenue, Portland, Oregon 97204		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2Q762722A764
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Research Institute for the Behavioral and Social Sciences, 5001 Eisenhower Avenue, Alexandria, VA 22333		12. REPORT DATE September 1977
		13. NUMBER OF PAGES 85
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

PLANIT
Team training
Computer-assisted instruction

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

Several efforts were made in this research project to assist PLANIT authors in writing team lesson scenarios. Team training is defined early in this report and then several specific authoring strategies are presented. A demonstration lesson is described in the text. The two most important products to come out of this research are contained in the appendices. One is a set of recommended modifications to the PLANIT language which is designed to simplify the team authoring process. The other is a detailed set of authoring guidelines which should help conventional PLANIT authors to become "team" authors.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ED010027

FOREWORD

The United States Army Research Institute for the Behavioral and Social Sciences (ARI) engaged the Northwest Regional Educational Laboratory under contract DAHC19-78-C-0042 to assist PLANIT authors in writing team lesson scenarios. Needed was a set of recommended modifications to the PLANIT language and a detailed set of authoring guidelines.

PLANIT, or Programming Language for Interactive Teaching, is a portable, time-sharing computer software system and authoring language used for the preparation and delivery of computer-assisted instruction (CAI).

Charles Frye spearheaded this effort of assisting conventional PLANIT authors to become "team" authors.

This Research Report is a comprehensive study of modifications of the conventional PLANIT language and is a complement to Research Report 1286, "Converting PLANIT Lessons to Enhanced Format," by Charles Frye.

Joseph Zeidner

JOSEPH ZEIDNER
Technical Director

EXTENSION OF COMPUTER-ASSISTED TEAM TRAINING THROUGH COORDINATED LESSON SCENARIO

BRIEF

This Research Report discusses several efforts which were undertaken to assist PLANIT authors in the writing of team lesson scenarios. It also describes and illustrates the assistance given to two such authors who were engaged in team lesson development pursuant to a contract between the United States Army Research Institute and Sensors, Data, Decisions, Inc. SDD is the prime contractor for whom this work has been done under a subcontract arrangement.

The text of the document first sets forth a definition of team training which is meant to be a frame of reference for the authoring features later discussed. After this, several specific authoring strategies are presented which enabled the kind of lesson presentation desired by the SDD authors.

A demonstration lesson is described which was developed to quickly show PLANIT team training concepts. The rationale for the lesson is described in the text and the lesson itself is attached in the appendix.

Two other important products of the effort are also contained in the appendices. One is a set of recommended modifications to the PLANIT language which is designed to simplify the team authoring process. The other is a detailed set of authoring guidelines which should help conventional PLANIT authors to become "team" authors.

EXTENSION OF COMPUTER-ASSISTED TEAM TRAINING THROUGH COORDINATED LESSON SCENARIO

CONTENTS

EXTENSION OF COMPUTER-ASSISTED TEAM TRAINING THROUGH COORDINATED LESSON SCENARIO

INTRODUCTION

The United States Army Research Institute (ARI) engaged Sensors, Data, Decisions, Inc. (SDD) under contract DAHC19-76-C-0042 to develop computer-assisted instruction that trains teams of tactical data system users. The overall objective of the work was to produce the requisite instructional strategies for computer-assisted team training and to construct and implement a "brassboard" system for conducting an on-line verification of the approach.

In pursuit of these objectives, SDD prepared lesson scenarios in a computer language called PLANIT (Programming Language for Interactive Teaching). PLANIT has been used extensively by the Army in applications which require the installation of a CAI software system in an operating tactical data system environment. Termed "Embedded Training," the concept of utilizing the tactical hardware itself to automatically dispense and supervise the training has been promoted by ARI for several years. Thus, prospective tactical operators would be trained by the equipment they are expected to learn to operate, via the same communication devices that they will eventually use in their duty station assignments.

PLANIT has been an effective CAI software vehicle for this training due to its machine independence and its extensive repertoire of authoring features as well as its execution mode for trainees where performance records are kept automatically. Since PLANIT had been developed with public funds, it was readily available for this new application at no additional expense to the government. However, PLANIT was oriented exclusively toward individualized training applications. Since tactical training needs also encompassed the training of "terms," a feasibility study was launched to determine PLANIT's utility in this regard, and to add some experimental authoring capabilities to the system which would permit further exploration of this particular application. That work was conducted by the Northwest Regional Educational Laboratory (NWREL) under contract number DAHC19-76-C-0008, resulting, among other things, in a modification to the PLANIT system which would enable the authoring of experimental team lesson scenarios.[1]

SDD used the modified PLANIT in which to author their experimental team fire mission training scenarios. This work is being reported separately. However, since the work was novel, new authoring strategies had to be devised in order to make the PLANIT lessons execute in the desired fashion using only the primitive team features which had been added. NWREL provided consulting during the early stages of this effort to help work out some of these authoring problems.

[1] Frye, C. H. The Feasibility of Adding Graphics and Team Training Support to PLANIT, Final Report, ARI Technical Report TR-77-A27, December 1977.

It became obvious as the work progressed that the consulting experience would give NWREL the information to provide two important services relative to the Embedded Training effort. They are as follows:

1. To recommend a much more comprehensive authoring modification to the PLANIT system which would ameliorate authoring problems similar to those faced by SDD, and

2. To prepare a set of authoring guidelines which would explain how to apply the authoring conveniences to a wide sample of team lesson strategy needs.

With these mandates in view, ARI negotiated an amendment to the SDD contract which provided funds for a subcontract to NWREL, making the period of performance coincide with the remaining time on the SDD contract for maximum overlap and continued mutual exchange of information. Therefore, the NWREL statement of work consisted of the above two points, plus the development of a brief demonstration PLANIT lesson which would be suitable for communicating the concept of team training within a very short period of time (such as in connection with briefings at ARI).

In reporting the results of this present effort, this document includes a conceptual statement defining the nature of team training and its requirements, and a discussion of the problems encountered in the SDD authoring process and the procedures devised to solve the problems for the short term (i.e., within the limits of the present PLANIT system).

The appendices contain the remaining contract report, including a set of recommended changes to PLANIT in order to implement a more comprehensive team authoring facility, guidelines for team authoring, and a copy of the lesson scenario and sample run for the demonstration lesson mentioned previously.

TEAM TRAINING DEFINED

The concept of teamwork and the team behaviors which are involved is so nebulous as to elicit a wide variety of mental images. Team members might be working side-by-side or separated by great distances. They might be engaged in physical work or mental work or both. They each might be assigned individual work which will later be integrated or they might be huddled and working together.

There are many reasons for performing an activity through teamwork rather than on an individual basis, such as:

- To distribute excessive energy requirements, bringing them within reasonable human limits.
- To achieve a higher quantity output than would be possible individually.
- To accomodate a geographical spread occasioned by the kind of work being performed.
- To enable the processing of stimuli which are being received too rapidly for an individual to handle.
- To distribute decision making with the intent to improve the quality of the decisions through a process of consensus.
- To build morale

Probably no team effort will have all of the characteristics. Rather, there are different kinds of team efforts. Even so, there are many ways to train teams. However, the objective here is to define what is meant by the kind of team training that characterizes the intent of the embedded team training system described in this document. It turns out that for many kinds of teamwork situations, individualized training will suffice.

Therefore, we will examine several typical team models and describe a seemingly satisfactory training model for each. Attention will then be directed to one of the models which best characterizes the kind of team training envisioned by this effort.

Before entering the discussion about particular team models, there are certain observations which will pertain to all of them, collectively. It is assumed that the observations pertain to team activity in general, and therefore pertain to team training as well.

<u>Observation 1.</u> Team activity begins with a common source of supply or information such that each of the team members draw from that source. We will call that source a <u>Common Data Base</u> (CDB).

<u>Observation 2.</u> Team activity results in a product or a related class of products, either of which might be turned out in quantities.

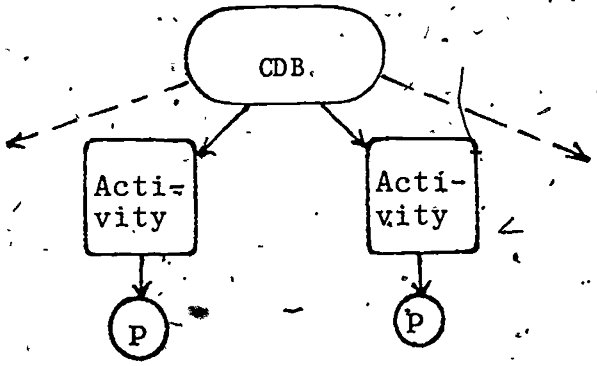
<u>Observation 3.</u> The product might consist of something (or several things) which are manufactured during the team activity or it might consist of an update of the CDB or both.

With these three observations as a frame of reference, let's proceed with the discussion of some team models.

Electronic Apparatus Assembly Model

The inspiration for this model comes from the familiar reports of production methods in certain Japanese electronics factories. The workers are given instruction, in groups and individually, regarding the assembly of some particular apparatus (e.g. a television set). This common beginning even includes group singing and calisthenics. The results are claimed to include teamwork benefits in the production of television sets, etc.

The pictorial model of this kind of teamwork is fairly simple:



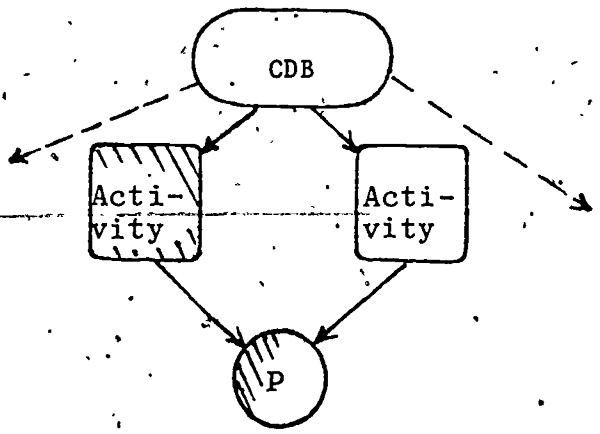
10

Note that each vertical path represents the history of a single team member culminating in a product (P). In this model, each team member produces a distinct product. Note also that the model can be generalized to a team of any size greater than or equal to two members.

The training implications for this model do not normally require team features of the kind that entail interaction between training packages. This team is essentially a collection of members who are performing individually so their training (particularly the cognitive- or manipulative-type) can just as well be administered in individualized packages.

Space Craft Assembly Model

This model differs from the "electronic apparatus assembly" model primarily in the unified nature of the product. The pictorial representation is as follows:



Note that a distinguishing characteristic of this model is in the retained identity of the contribution of each team member to the final product. It is possible to disassemble the product and recover the piece that each member produced.

Many would be quick to point out that this is a gross over-simplification of the building of a space craft, that much interaction goes on and the data base is continually being updated. However, that observation would pertain only to corrections in faulty designs (which probably often occur). This model assumes a perfect design and a delegation of tasks to produce components which are later to be integrated into a single operational unit.

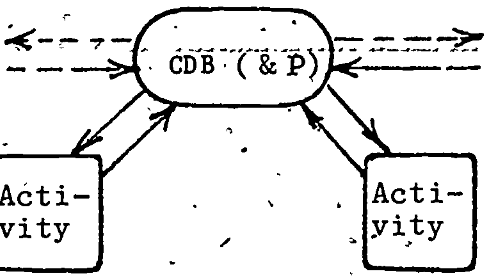
Another common example of this model is found in the programming of a large computer software system where each of many subprograms is assigned to some individual member of a programming team. The finished product will be an integration of the individual efforts of each of the members.

As in the previous model, since the task is being performed independently, the training to perform that task can just as well be administered individually. Therefore, again acknowledging that this is actually a team exercise, the nature of the team model does not require the acquisition of team behaviors as a training objective.

Air Line Reservation Model

This model differs from the previous ones in at least two respects: the product consists of updates made to the CDB, and the product cannot be disassembled into components which can be identified with particular team members.

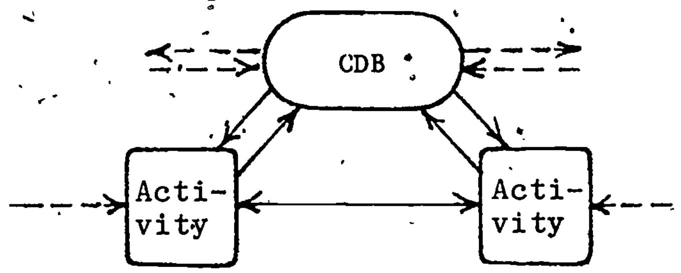
The pictorial of this model follows:



Here, as before, the members are working independently. Of course we know from experience that the agents quickly confer with one another when the computer goes down. However, when the system is working as it should, the model requires no interaction between agents. Thus, because they do their work independently, they can receive adequate job training through individualized training packages. Team behaviors are not part of the learning goals for the execution of this part of their task.

Computer-Based War Games Model

This model is similar to the previous one, adding only a real time interaction requirement among the team members during the course of the activity. Its pictorial might be represented as follows:



The operation of this model consists of actual team exercises on actual equipment but with simulated data and with an aborted or non-effective product. The attack is not real (at least in the time frame of the exercise) and the guns do not actually fire (or if they do, it is not for effect).

The Computer-Based War Games Model has served as the primary vehicle in the past, together with adjunctive classroom sessions and reading assignments, to train recruits to operate the various military tactical systems. Typically, simulated runs are monitored by a trained instructor and then a debriefing takes place at a later time, perhaps in a classroom.

No argument is intended regarding the fact that training does take place within the introduction-simulation-debriefing paradigm. However, two things should be fairly obvious about the use of the tactical system in this mode: its object is not productivity since the normal product of the system (firing accuracy) is deliberately aborted, and it is not teaching, it is testing. A novice would be completely lost in the system just as one with no previous flight training would be completely lost in a LINK trainer. The training has occurred prior to the exposure to the equipment and the simulation work on the equipment tests whether the trainee can perform acceptably the skills which have been taught.

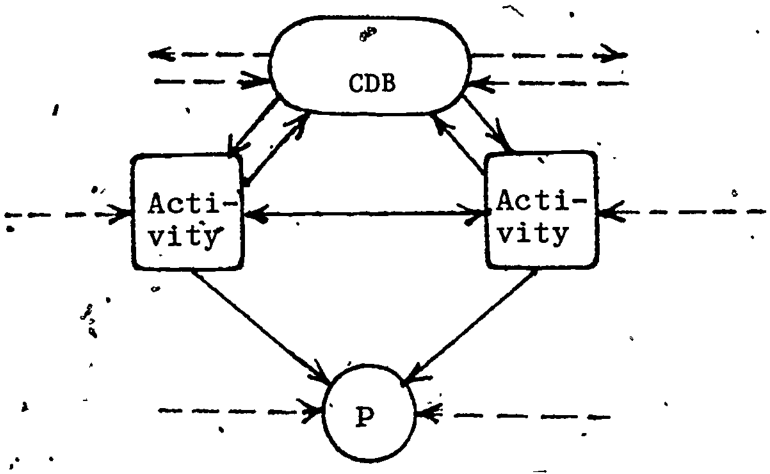
Thus, the Computer-Based War Games Model can be described as modifying a team-operated production tactical system by eliminating its normal productivity, and using it instead as a performance testing device which either declares the trainee's competence or provides data for further remediation and training.

The training for this model requires that team behaviors must be learned along with cognitive and manipulative skills. However, at the risk of too much redundancy, note that the learning takes place in the classroom or in the books but not on the simulator. The simulator is only providing feedback for the learning that should have already occurred. The new learning effectiveness of the feedback will await the debriefing (which could be a comment made at the moment or a later session in a classroom).

This last model will complete the progression.

Tactical Fire Control Model

The pictorial for this model was implied in the discussion for the previous one, namely the same but with the addition of a product (which was eliminated from the prior one for simulation purposes):



14

This Tactical Fire Control Model would seem to incorporate all the features of a generalized team exercise, at least as far as the current state-of-the-art has carried it. Note that it includes all of the following elements:

- A common data base (of information and materials)
- Differentiated team member activities
- Provision for updating the data base
- Required interaction among team members
- Common product (components not traceable to contributing team members)

Such a model as this places heavy requirements on team behavior objectives among other performance criteria for the training of recruits. To rely solely on individualized training packages (of any kind) would fail to exercise the recruit in some of the skills that will be needed to operate the system.

Implications of the Models for Team Training

In all but the last two models, individualized training packages would seem to suffice. Although there were several instances of team performance, the team members were expected to perform essentially non-interacting tasks which were later to be integrated into a single product.

However, in the last two there appears a clear mandate for addressing team behaviors in the training.

In any case, it is generally true that the more similar the training is to the intended task, the more effective it will be. The implications for this are quite clear, especially where the task in question involves the operation of a computer: that training can be threaded into the activity itself and thus produce nearly ideal similarities to the expected task. Computer-assisted training has established its effectiveness by now, and can quite feasibly be used to direct the training in either of the last three models.

In the case of the Air Line Reservation Model, the computer-assisted training can be presented in individualized fashion. Several such systems exist for this mode of instruction, depending on the kind of lesson presentation desired and the type of equipment on which the system is designed to operate.

In the case of the last two models, relating to Fire Control Systems, the PLANIT authoring system is being modified to accomodate the kinds of team training requirements which are found to be necessary. The following features are presently available or planned:

- Scenario authoring and execution facility (available)
- Provision for a Common Data Base (numerical mode available, generalized data capability planned)
- Provision to update the data base as required (available)
- Provision for acquisition of and communication among team members (available, enhancement planned)
- Provision for the independent and/or interdependent execution of team lesson scenarios (available, enhancement planned)
- Provision for manipulating the hardware in a manner that will not be distinguishable from the ultimate task (available)
- Provision for the generation of output data which can be correlated to the intended system product (available)

There are at least two major limitations which will probably be evident but will nevertheless be stated:

- The computer cannot accomodate all of the tactical software and all of PLANIT at the same time. While training is in progress, only a segment of the tactical system can be dealt with at any given moment.

- It will normally not be practical to thread training in with the activity of a production fire control system. There will be no extra time available to take the operating team member through remedial exercises while guns are firing. The exception may be found in the critical situations where trained operators have been immobilized and novices must step in.

In summary, there was an attempt to define team training in terms of a specialized kind of requirement for the deliberate teaching of team behaviors. Many of the tasks which are rightly considered to be team exercises do not require an interdependent, interpersonal kind of team member performance. Rather, the elements of the team task have been so delegated that the team members can perform their work largely independently. For this kind of team member performance, individualized training should be sufficient.

However, there are interdependent team tasks, particularly among Tactical Fire Control systems, where team members must act in concert, where no one member of the team can perform the task satisfactorily unless all members perform satisfactorily. The training for this must include the teaching of team behaviors. Also, since the facility is available, there is no better place to deliver the training than on the system hardware itself.

TEAM SCENARIO PROBLEMS AND SOLUTIONS

As already indicated, SDD did their team lesson scenario development using a PLANIT which contained only the rudimentary set of team authoring features, including:

- PUT MATRIXNAME
- FETCH MATRIXNAME
- DIAL n 'MESSAGE.' (Usable in Calc)
- TERMINAL

SDD personnel had no difficulty learning conventional PLANIT authoring from the user manuals. They requested consulting help for the team logic. This was provided in the form of one on-site visit and several exchanges over the telephone. In addition, several sections of scenario were coded for them, either to use as it was, or to provide a model which they could modify to their needs.

Initialization

Initialization is taken to include the initial gathering of the team members onto the team lesson, providing suitable points in the lesson logic where necessary data initialization could occur, and making possible a debriefing and re-initialization for the next team.

The code provided to SDD was taken from the example in the Feasibility Study report.[2] Some modification of this example was necessary to adapt it for two-person teams and to provide re-initialization logic. Also, a different COMMON Matrix size was desired. The code provided for this task follows:

---

[2] Ibid, p. 53.

```
FRAME 1.00 (D.)

G2. CRITERIA
C:SET MATRIX(X,10,4,2)
IF LINK(9) NQ 0  C:PUT X  C:LINK(9)=0
ELSE  C:LINK(9)=1  C:FETCH X
IF PROD X(1,3,I) FOR(I=1,2) NQ 0
F:SORRY, ALL POSITIONS ARE TAKEN. TRY ANOTHER TIME.  B:6

FRAME 1.50 (Q.)

G2. TEXT
ARE YOU FDO OR FDS?

FRAME 2.00 (Q)

G3. ANSWERS
0  KEYWORD ON
A  FDO
B  FDS

G4. ACTIONS
A C:SET COLOR=1
B C:SET COLOR=2
- R:ANSWER ONE OF THE TWO OR TYPE '← FINISHED'

FRAME 3.00 (D)

G2. CRITERIA
C:FETCH X
IF X(1,3,COLOR) NQ 0
F:SORRY, WE HAVE ONE OF THOSE ALREADY. CHOOSE ANOTHER.  B:2
ELSE  C:SET X(1,3,COLOR)=TERMINAL  C:PUT X  B:5

FRAME 4.00 (Q)

G2. TEXT
DON'T HAVE ALL THE PLAYERS YET. TYPE 'GO' TO CHECK AGAIN.

G3. ANSWERS
A  GO

FRAME 5.00 (D)

G2. CRITERIA
C:FETCH X
IF PROD X(1,3,I) FOR(I=1,2) EQ 0  B:4
ELSE  C:LINK(9)=0  F:OK, LET'S GO.  B:FDO,FDS;COLOR
```

```
FRAME 6.00 (Q)

G3. ANSWERS
0  WAIT 10
A  DEBRIEF

G4. ACTIONS
-  C:FINISHED

FRAME 7.00 (P)

G2. STATEMENTS
F:DEBRIEF AND RESET
C:SET MATRIX(X,10,4,2)  C:PUT X
```

Most of the logic in the above example is similar enough to previous examples that no further explanation is needed. However, a couple of things will be pointed out.

Note the second and third lesson line of Frame 1.00, where the LINK logic appears. These two lines will cause the Common Matrix to be initially defined on the first execution of the lesson.

Next, note that the last line of Frame 1.00 terminates with a branch to Frame 6.00. Under normal student conditions, if all positions are indeed occupied, the branch to Frame 6.00 will simply activate a ten second waiting period after which that student will be logged off. However, the author will know at that point that the word, "DEBRIEF" can be typed during the ten second interval which will cause Frame 7.00 to be executed. It was intended that SDD would add code to cause any values of interest to be printed in Frame 7.00, after which the Common Matrix would be re-initialized in preparation for the next team.

Comparing the above example with the initialization example in the Guidelines (in Appendix B), one will find a marked simplification due to the additional authoring directives. Not only is the code simpler, it is also more effective in that the PLANIT system makes the determination whether the team is together and when to let the students proceed. Efficiency is also improved by an order of magnitude. It is no longer necessary for a particular student's lesson to execute simply to determine whether that student should be allowed to proceed. Rather, that student is not scheduled until conditions have been met.

Synchronization

The SDD lessons were prepared for two-member teams where the members were designated the "FDO" and "FDS", respectively.

The synchronization requirements followed a regular pattern throughout the series of lessons, described by the following steps:

Step 1: Fire mission options presented to both the FDO and FDS.

Step 2: FDO chooses an option while the FDS waits.

Step 3: FDS is informed of FDO's choice while FDO moves on to the next fire mission option list.

Step 4: FDO waits to enter next option while FDS enters previous option.

Step 5: FDO enters next option as the cycle repeats.

Thus, the FDS is held exactly one move behind the FDO, moving together in this fashion through the scenario.

One variation to this pattern allowed the FDO to move to independent execution of another scenario while the FDS proceeded alone through the fire mission scenario without the interaction with the FDO.

The synchronization algorithm devised for this requirement made use of the PLANIT Common matrix, DIAL command, WAIT directive, and a synchronization code number scheme. The scenarios for the FDO and FDS were similar but not identical.

The FDO executed a normal PLANIT Question frame wherein an option list was presented and one of the options (by number) was given as the response. Then, instead of allowing execution to move from one Question frame to the next, execution always branched after each of the Question frames to a set of control frames, then back to the next Question frame in the sequence. The Question frame sequence was designated by frame numbers in an array called SYNC, where each synchronization code number value, when used as a subscript to the SYNC array, designated the appropriate next frame in the sequence. Therefore, the synchronization code number, together with the SYNC array, uniquely defined every point of progress through the lesson scenario. The control frames to implement this algorithm included an initialization frame at the beginning of the lesson segment, and a set of synchronization frames placed within the lesson segment.

The initialization frame contained the initialization logic described in the previous section as well as the SYNC array definition. The statements for the latter were as follows:

```
C:SET MATRIX(SYNC,60)
C:SYNC(1)=ARRAY(9,10,11,12,13,14,16,17,etc.)
C:SYNC(17)=ARRAY(37,etc.)
etc.
```

The complete array declaration has not been filled in above (as designated by the "etc."). However, the numbers are the frame number values corresponding to the synchronization code values. For example, a synchronization code value of "17" would show that the team is executing frame number 37. The PLANIT logic implementing the synchronization follows:

```
FRAME 5.00 (D)

G2. CRITERIA
C:FETCH X  C:X(1,3,1)=RESPONSE  C:X(3,3,1)=X(3,3,1)+1
C:PUT X
IF X(3,3,1) LS X(3,3,2)  B:SYNC(X(3,3,2))
IF X(3,3,1) EQ X(3,3,2)  B:8
ELSE  F:@$

FRAME 6.00 (Q)

G3. ANSWERS
O WAIT 5
A DUMMY

G4. ACTIONS
-A F:STANDBY.

FRAME 7.00 (D)

G2. CRITERIA
C:FETCH X
IF X(3,3,1) GR X(3,3,2)  F:@$B:6
```

```
FRAME 8.00 (P)

G2. STATEMENTS
B:M1,M2,M3,M4,M5;X(1,1,1)
M1:DIAL FDST I THINK 'PAGE' IS THE RIGHT ANSWER. B:TAIL
M2:DIAL FDST I THINK 'FIRE' IS THE RIGHT ANSWER. B:TAIL
M3:DIAL FDST I THINK 'CLEAR' IS THE RIGHT ANSWER. B:TAIL
M4:DIAL FDST I THINK 'CORRECT' IS THE RIGHT ANSWER. B:TAIL
M5:DIAL FDST I DON'T KNOW THE RIGHT ACTION, YOU CHOOSE.
TAIL:

FRAME 8.10 (D)

G2. CRITERIA
B:SYNC(X(3,3,1))
```

Each time the FDO received new fire mission instructions in a Question frame, responded to a list of optional actions, and received feedback regarding the correct or incorrect nature of the choice, execution would always branch to the above set of frames.

In Frame 5.00, the Common matrix would first be updated with the value chosen representing the option (shown as X(1,3,1)) and an increase by one of the synchronization code, X(3,3,1). (The synchronization code for the FDS was X(3,3,2)). The PUT X statement marks the completion of the update. Then a test is made to determine if the FDO happens to be out-of-step in the scenario (as would be the case if he had been executing independently). If true, the branch would put his execution back in step again.

The next test "X(3,3,1) EQ X(3,3,2)" asks whether the synchronization codes for the two players are equal, i.e. whether the FDS is ready and waiting for the FDO to respond. If true, execution branches to Frame 8.00 where the FDO's response is DIALed to the FDS and execution continues at the next frame designated by the SYNC array, the branch being implemented in Frame 8.10. If the FDS is not ready yet, the FDO will wait for five second intervals, each time testing (in Frame 7.00) if the FDS is ready yet. When the FDS is ready, the X(3,3,2) code will be advanced by one, causing execution for the FDO to drop through Frame 7.00 and on to Frame 8.00 and beyond.

In order for the FDO to execute independently in another scenario while leaving the FDS to proceed in this scenario alone, the X(3,3,1) synchronization code is set to some arbitrarily high number (e.g. 1000) and branches are no longer made to Frame 5.00 until/unless there is a need to re-establish the team relationship. Note that the team relationship will be automatically re-established and the FDO will be placed back into proper synchronization with the FDS simply by branching to Frame 5.00 where the first test will put him back into the lesson again.

. The FDS will have a slightly different synchronization sequence because he must wait to receive the FDO message before proceeding. However, the SYNC array initialization in the first frame will be the same (albeit with different frame numbers).

For the FDS sequence, the presentation of the fire mission options must be split from the response evaluation portion with a branch to the synchronization section in between. Thus, the SYNC frame numbers will reference the second half of each of these split frames. The pattern is as follows:

FRAME 2.00 (Q)

G2. TEXT
This text presents the fire mission instructions and the list of options from which the FDS chooses.

G4. ACTIONS
B:5

FRAME 3.00 (Q)

G3. ANSWERS
Answer matching instructions for the chosen option.

G4. ACTIONS
Feedback pertaining to the chosen option.

The above two Question frames constitute that which would normally occur in one Question frame were it not necessary to split the sequence to permit synchronization. In the above case, the synchronization frames begin with Frame 5.00, and the first SYNC entry would be the value "3" to correspond to Frame 3.00.

The synchronization frames for the FDS follow:

```
FRAME 5.00 (Q)

G2. TEXT
@$

G3. ANSWERS
O WAIT 1
A DUMMY

G4. ACTIONS
C:FETCH X  C:X(3,3,2)=X(3,3,2)+1  C:PUT X

FRAME 6.00 (Q)

G3. ANSWERS
A DUMMY

G4. ACTIONS
C:SET REPLY(1)

FRAME 7.00 (D)

G2. CRITERIA
C:FETCH X
IF X(3,3,1) LS X(3,3,2)
F:STANDBY FOR FDO ACTION BEFORE RESPONDING.  B:6
ELSE  C:USE REPLY(1)  B:SYNC(X(3,3,2))
```

Encountering Frame 5.00 causes a very brief pause (one second) while the remainder of the text is printed on the FDS terminal. The "@$" characters serve to suppress the extra line feed which would normally occur at that point (a special PLANIT authoring strategy but unrelated to teaming or synchronization).

In Group 4 of Frame 5.00, the FDS synchronization number in the Common matrix is increased by one. Both scenarios (FDO and FDS) rely on comparing the current values of the respective synchronization numbers to determine when to allow execution to continue. For the FDS, this determination is made in Frame 7.00. Note that the FDS must respond in Frame 6.00 before Frame 7.00 will be executed. The response in Frame 6.00 will be to the list of alternative options which had just been presented prior to the branch to Frame 5.00. However, if the FDS responds too soon (i.e. before the FDO message has been received), the test in Frame 7.00 will cause a loop back to Frame 6.00, effectively ignoring the response, but with the cryptic message to

"STANDBY FOR FDO ACTION BEFORE RESPONDING."

As soon as the FDS receives the proper message from the FDO's lesson he enters the option number chosen for the response to the previous displayed option list. That response is immediately placed in PLANIT's REPLY(1) response buffer. Then, in Frame 7.00, the ELSE alternative of the IF statement will be executed (since the sending of the FDO message is accompanied with the appropriate update of synchronization numbers). The ELSE track of the lesson scenario designates that REPLY(1) buffer be used for the next Question frame answer evaluation, and the lesson branches back to that appropriate frame number.

While the foregoing logic is not necessarily suitable for all lesson synchronization needs, it does satisfy all the requirements for this one. With this scheme, synchronization at properly paired points will be virtually guaranteed. Yet, it also allows the two members to work independently.

A variation of the above scheme would probably be useful for a wide variety of synchronization applications but specific lesson requirements would have to be known in order to modify the sequence appropriately.

Even if the above logic were to be used again just as it is, it would be useful to make some simple modifications which would eliminate the use of the DIAL directive. Instead, the chosen option number would be passed to the FDS lesson through the Common matrix and used in the FDS lesson to select the corresponding message to be printed. The results on the terminal would be identical but the logic of the control sections of both lessons would need to be changed. There would be some improvement in efficiency but, more importantly, the resulting lessons would be fully transportable to the TACFIRE hardware.

DEMONSTRATION LESSON FOR TEAM TRAINING

A demonstration lesson for team training has been reproduced in the appendices. Both the card deck for the lesson and a sample run have been included.

The problems of demonstrating a particular feature of any system are usually quite different from that of developing instructions for putting the system to practical use. This is particularly true for a computer-assisted instructional system such as PLANIT.

Instruction usually takes place over a period of time. There is typically some orientation necessary to introduce the trainees to the material, and then a sequence of imparting knowledge and assessing performance. Instructional sessions often run for an hour or longer and several sessions are often required to reach the desired instructional objectives. In terms of the lesson scenario itself, the introductory parts are most often composed of the most elementary examples of authoring strategies. Thus, for demonstration purposes, where time is often limited to 15 minutes or so, the lesson must move quickly, require minimal orientation, and show the desired operations almost from the beginning. Also, it must obviously be interesting and command attention. This helps to explain why games of a relatively unsophisticated variety are so popular for demonstration purposes.

The demonstration lesson developed to show PLANIT's team training capabilities is such a game. However, it has also been carefully devised in such a way that partic pants can improve their ability to play the game with practice. Thus, some learning takes place.

The game which was developed is modeled after a real life problem faced by truck rental firms regarding the distribution of their rental equipment resulting from one-way rentals. Many rental routes attract customers much more readily in one direction than the other, leading to the possibility of accumulating equipment in remote cities where it sits idle. Often, the rental firms must provide discount incentives in order to get the equipment back to the higher utilization areas again. Of course they discount only enough to entice the customers.

This demonstration game presents a constrained version of that one-way truck rental problem. The operating range includes only four cities. From one to eight truck rental companies compete for available customers who wish to use the truck over any one of the twelve possible routes. In addition to priority customers who are willing to pay full rate to take the truck, there are also two other classes of potential customers, one who will go at a discount involving some reduction in profit, and another who will go only at the lower rate which results in a loss to the company. Each team player governs the affairs of one company, competing to earn more accumulated profit than the other companies.

The truck rental game has been devised to provide a customer base which is proportional to the size of the city and the particular route desired. All companies draw customers from the same data base on a first-come basis. The customer pool is replenished at regular time intervals where the numbers of replacements for each possible route are chosen randomly within the limits of the expected number for that route. The total number of customers replenished in any route category is asymtotic in that remaining unserved customers tend to diminish the number of new customers which will be added to that category. Thus, the planned effect will be to maintain a relative scarce level of profit-making customers. (There is no limit to the "freebies").

Another strategy which has been built into the game is a period of time that rented equipment is out of service, i.e. enroute to its destination. This delay is differentiated according to the distance that must be traveled. This tends to limit the opportunity to capture as many of the full-paying customers as one might desire.

The game automatically calculates profits according to the distance traveled and accumulates profits for each player, making a display of all companies' profits available upon request. Also available for display is the current distribution of one's trucks (including the identification of those enroute), and the identity of the other players. Customer availability for appropriate route categories is also displayed when the player indicates the route to which trucks will be assigned. After the customer pool has been displayed, the player has the option of how many trucks to send, from zero to the maximum available.

This game is not a team exercise as the term has been used in this document since it represents competitive behavior rather than cooperative behavior. However, it would be relatively easy to modify the game to show team behavior simply by delegating the rental responsibility for each company to a group of regional office managers. Since there are four cities in the game, each team would be composed of four players. Together, they would have to develop a strategy for moving the trucks in the most optimal way.

The team demonstration did not employ the regional office manager strategy because of the added complication it would impose. At least eight players would be needed with increments of four thereafter. As it is, any number of eight or less would be appropriate (even including only one player but at the loss of the competition element). Also, most observers will be readily capable of making the mental extrapolation from the current version to that of manager teams, even without actually playing in the team version. Therefore the present version appeared to be satisfactory and the development of the regional manager team version seemed unwarranted due to the impracticality of assigning players.

The truck rental game appears to demonstrate the interaction among participants reasonably well. Time needed for the demonstration of the basic concepts can vary from a minimum of little more than five minutes to an hour or more, depending on the interest of the players. Orientation to the game consists of a few short paragraphs of instructions, and even these can be skipped if the demonstrator prefers to orient the players verbally. Nothing is told to the players about appropriate rental strategies. These are left to be worked out individually.

This truck rental game seems to fill most of the requirements for a short, interest-grabbing demonstration of interaction among players on a common lesson scenario.

CONCLUSION

This effort did indeed show that technical authoring problems in the PLANIT system could be overcome in order to implement an authoring-strategy for the training of teams. It also showed that such strategies must be simplified through ameliorating authoring conveniences in PLANIT if this kind of authoring is to become practical for the ordinary ability-levels of authors.

The attached Guidelines ought to provide much needed information to authors who might attempt to develop team lessons. There is little doubt that the authoring of team scenarios is more difficult than comparable individualized scenarios, particularly when intricate synchronization of the respective lessons is required. Required lesson communication across team members further complicates the authoring, and these complicating factors will be the norm rather than the exception when serious team authoring is contemplated (for reasons which were discussed earlier in the definition of team training). However, added complication does not necessarily imply lengthy sections of lesson code. This can be ameliorated by new authoring directives in PLANIT which perform the required operations in concise statements. Also, the added complexity need not impose too great an extra burden on authors if the Guidelines for team authoring communicate as they should and provide relevant examples. That, of course, is the intent of the attached Guidelines.

Finally, team authoring capabilities in PLANIT are still new. Attempting to find related experience in other comparable software systems has been unproductive. Therefore, it is reasonable to expect that new modifications will be recommended from time to time, with the result that any documentation attempt will become obsolete very quickly. It appears that the rapid obsolescence of manuals cannot be avoided at present but, with that problem in mind, the attached Guidelines were written to reflect not only the current status of PLANIT but also the status as it is expected to exist after the next major modification. This approach assumes that there will be opportunity to make those changes and that the changes will be made exactly according to the design. However, the benefits of having available for a longer period of time a valid document seem to outweigh the risk that the assumptions might suggest.

APPENDIX A' NEW TEAM AUTHORING DIRECTIVES RECOMMENDED FOR PLANIT

Northwest Regional Educational Laboratory 710 S.W. Second Avenue • Portland, Oregon 97204 • Telephone (503) 248-6800

June 22, 1977

Mr. Jim Baker
U. S. Army Research Institute
PERI-OR
5001 Eisenhower Avenue
Alexandria, Virginia 22333

Dear Jim:

I am sending the June, 1977 monthly report early in order to put some ideas before you. Some pertain to work that I shall be proceeding with as soon as we reach accord, and some will be of a more long range nature.

I have come up with a package of new additions to PLANIT, all of which are directly or indirectly connected with the extensions which we planned for the team training capabilities. You recall that I was to submit a plan so you could react before the additions were actually put into PLANIT.

The plan I would like to submit goes beyond that which can be accomplished in the remainder of the summer under the SDD extension, but I think it would be well to consider the plan in its entirety anyway and carve out that which could be done now. I am satisfied that these new conventions would supply everything that SDD needed but lacked in their effort, and then some. There are some ramifications for two of these changes which I will go into.

Essentially, the plan calls for adding six new features to PLANIT and changing two more that are already there. I will take them in order, with a brief explanation of what is intended. This may also call for some more discussion over the telephone.

1. The SYNC feature. Its lesson forms would be:

```
SYNC(T1,T2,···,Tn;N)
X=SYNC(T1,T2,···,Tn;N)
```

AN EQUAL OPPORTUNITY EMPLOYER 31

SYNC does the following:

- Allows lessons to have punctuated synchronization points at which all players will wait until all have caught up.

- Allows lessons to have numbered synchronization points to guarantee that all players are synchronized at the appropriate place in their respective lessons.

- Allows some players to leave the team setting and work on their own while the remainder of the team continues to work as a team without him (them).

- Allows players who have left the team to work on their own to rejoin the team, find the place where the team is currently working, and become a team member again.

In the SYNC format, the T1,T2,···,Tn names an arbitrary number of team members by their terminal numbers. These numbers can be acquired through the COMMON matrix as the current team lessons do. The "N" following the semi-colon can take on integer values from zero upward. A zero value means that this player is leaving the team for awhile. Larger values represent synchronization numbers. They can coincide with topic numbers, frame numbers, or whatever. They indicate how far the team has progressed. The ;N can be omitted from the statement and "N" will default to "1", indicating a simple synchronization point that is not defined by a sequence number. A return value can be assigned to a CALC variable ("X" in the example). This shows the highest synchronization sequence number reached by the team, and can be used by a player who wishes to automatically find the right lesson location where to rejoin the team.

If the team is not yet completely assembled when a lesson executes a SYNC statement, the computer will ask the user whether he wishes to wait. This will provide opportunity to automatically assemble the team, a task that required several frames in the SDD lessons.

This SYNC command can be implemented under the current contract.

2. The SPECIAL command. This command already exists in CALC and is in use in Litton's enhancements example. I propose that we reduce the number of arguments to eight (instead of the present ten), and provide several new capabilities in addition to those that are already there, including:

- The ability to make the lesson wait for an action that may not complete for awhile. During the wait, that user will not be consuming computer time. This could be a wait for a light-pen input for example.

- The ability to put the user into an OK/CANCEL mode where he will not proceed until he receives the OK (or CANCEL) from the operator. A good case for this would be attempting to write on a plotter where the operator must first make the plotter available.

- Some alternative actions which MIOP could cause the lesson to take, such as reporting an error, logging off the user, etc.

- Put all buffered messages out to the user before the SPECIAL call makes its request to MIOP. This would simplify some timing problems.

The current capabilities would be unchanged (except for the reduction in the number of arguments by two--there are probably too many now anyway). Default conditions would revert to the present kind of operation.

This change would have some ramifications for Litton. They are very probably the only ones who have used the SPECIAL call so far. When they mounted this version, they would have to make a minor change to their MIOP (less than an hour), and edit out the last two arguments in the SPECIAL calls in their enhancements lessons. These can be done with the PLANIT M command.

I believe this change is needed and if it is postponed it will only cause more problems as more uses of that call are made. The change can be made under the present contract.

The remaining items probably cannot be done under the current contract but they together constitute a package so I will describe them.

3. There are three command variations here:

```
SET REPLY(n)="composite expression"
FETCH REPLY
PUT REPLY
```

There is already a SET REPLY(n) which captures the most recent terminal input. This would not change. It would only add another way of putting information into the REPLY buffer. The "composite expression" format would be exactly the same as following a PRINT statement, where words and CALC values can be pieced together into a single line. In this case, instead of being printed on the terminal, it would be loaded into the REPLY buffer. From there, it can be printed, used in a Group 3 match, or whatever.

Couple the above with the FETCH and PUT and it makes a powerful team option. The FETCH and PUT would do for the entire array of REPLY buffers what it now does for the Common matrix, it would allow lessons to exchange copies of these buffers. It would enable one lesson to construct a line, send it to another lesson and that lesson use it in any fashion, such as to hold and print a message at the appropriate time, to submit a reply at one terminal and match on it in a different lesson, etc. The possibilities here stretch the imagination.

This is one case that would have to wait. It would not interfere with present lessons but be new options for authors. Therefore, it would be better to save it for the next contract.

4. This one is a suggested change for the use of the DIAL command in CALC. The form would be:

```
DIAL n "composite expression"
```

The "composite expression" would be the same as for the SET REPLY option, above. It would permit the dialing of much more flexible messages across terminals, including values of CALC items.

The present CALC form of the DIAL command has some problems when the author tries to put characters into the DIAL message that CALC does not accept. The above format would bring the DIAL syntax back into conformity with the CALC language syntax, as it should be.

The SDD lessons are presently the only ones that have used the DIAL command in CALC. These DIAL statements would have to be modified by putting quote or prime (" or ') delimiters around the present messages. When properly delimited, any character can appear in the message. It would not be a difficult change to make.

I recommend that we plan to make this change but will not have time to do so until the next contract. The major work will be the "composite expression" which both options can share so the REPLY change and the DIAL change should be done together.

5. The last one is an ESCAPE option for authors who program themselves into trouble. Since team lessons are somewhat more difficult, it is probably even more needed for team authors. The command form is:

```
ESCAPE(n)
SET ESCAPE(n)
```

It would be used exactly like the WAIT function in PLANIT, where the value could be SET for the duration of the lesson, or only for one frame. The ESCAPE option says to log off any student who uses more than "n" seconds of CPU time between terminal inputs. Its purpose would be to automatically break loops that the author may have inadvertantly put there.

The default value for ESCAPE should probably be about 60 seconds but the author could raise that if the lesson was going to do a lot of computing in a certain section, as for example, might be done in a simulation program.

If this option had been available for SDD while they were working on their lessons, it probably would have saved them a thousand dollars or more. It is not unusual to get looping started in the lesson, and it takes an operator on the current system to stop it. The ordinary author or student would not know how. The operator must QUIT that user. The ESCAPE option would do the same thing after the prescribed amount of CPU time had elapsed. It would be automatic so that the cost of it would be negligible and a message would come to the terminal warning of the probable cause.

There may be time to include this under the current contract but I would want to wait and see. It would have no negative impact on current lessons.

These above options constitute the plan. Taken together, they provide an excellent package for enhancing team authoring, particularly for those features found lacking in the SDD experience. I am certain they would have found most of them quite useful.

One additional feature of this plan which has become very important to the PLANIT system is that there would be at most two or three additional data words required for all the new options combined. This would amount to an insignificant increase in PLANIT's Common data space. There would be quite a lot of new code involved but this can be arranged so that it can go either in core or in an overlay, at the option of the installer. Thus, there would be virtually no size increase in PLANIT (if overlaid) which would be a vital consideration for installations such as the PDP 11.

Depending on the outcome of the consideration of this plan, certain other factors enter into the decision making:

1. The obvious question is whether these recommendations are satisfactory, or whether you might want to change the options, the order of installation, etc.

2. When writing my report on effective team lesson authoring, should I include all of these options or limit the text to that which PLANIT currently executes.

3. Should the Team benchmark lesson include these new options.

4. On the assumption that the addition of new options would mean a new PLANIT release (version 3.2), is it time to change the "update" document (a copy is attached)? It is already behind (version 2.8). Finding time would be the next question. However, if you are thinking of a more involved effort at a later time of producing new user manuals, then that might influence what we do in the interim.

I have had a leased telephone line on order for about six weeks now and it should be in before too long. Having lost my unlimited WATS line, that was the least expensive option. As soon as it is in, I want to begin work on the changes. Would like to consider your feedback on this plan by mid-July or before if possible. I hope that is not pushing you.

Sincerely,

Chuck Frye

Charles H. Frye

Enc.

AUTHORING GUIDELINES FOR PLANIT TEAM TRAINING SCENARIOS

Prepared For:

The United States Army Research Institute

Alexandria, Virginia

Prepared Under Sub-Contract To:

Sensors, Data, Decisions, Inc.

San Diego, California

Contract No.

DAHC19-76-C-0042

Charles H. Frye

The Northwest Regional Educational Laboratory

Portland, Oregon

9 September 1977

ABSTRACT

This document contains guidelines to help a PLANIT author write effective team training scenarios. In order to benefit from all of the described features of the language, the author must have access to an installation of PLANIT Version 3.2 or above. However, those features which are not available in version 3.1 are marked so that they can be avoided in the interim until access to version 3.2 has been provided.

The document stresses the initialization, manipulation, and readout of data in the Common Data Space, a facility that will normally be needed in team training applications. It also discusses how to get the team started together, keep them synchronized, get them going again in the event of an interrupted session, and provide necessary cross communication.

It is expected that the reader has already acquired some proficiency in the authoring of conventional PLANIT lessons.

AUTHORING GUIDELINES FOR PLANIT TEAM TRAINING SCENARIOS

INTRODUCTION

Computer-administered team training is a relatively new form of computer-assisted instruction (CAI), and authoring skills required to produce the training scenarios seem somewhat more complex. It is still too early to determine whether this added complexity stems from the nature of the task, its novelty or awkwardness of the authoring language (or a combination of the three) but the present CAI author can be expected to need extra help when attempting to develop such scenarios.

This document attempts to provide such help for PLANIT authors. PLANIT (Programming LANguage for Interactive Teaching) is a CAI system for authoring and administering training scenarios. Recent efforts have introduced special capabilities into PLANIT which were found to be necessary adjuncts for the authoring and administering of team scenarios. Through initial experimentation, additional team capabilities have been defined for the language which, at the time of this writing, have not yet been implemented in PLANIT. However, this document will include these new features in its discussion anyway, anticipating the time when they will become available.

Thus, this document will present authoring guidelines, some of which cannot yet be used until the next PLANIT version is released, version 3.2. The features which will not be ready until version 3.2 will be marked in the text with a double asterisk (**). The absence of this mark will mean that the authoring feature is now available in version 3.1.

It will be apparent to the practitioner that the special team features will make up only a small part of the scenario, that the bulk of the scenario will be composed of conventional CAI authoring directives.

Also, in some cases, the special team directives will be a variation of conventional directives. The printout of a team lesson will look much like the printout of a conventional lesson. It will be the strategy around which the lesson is structured that will deviate from the conventional.

The strategy of team lessons will normally demand interaction among participants and with a common source of information. The concepts of team training are discussed in a separate document, a final report to the United States Army Research Institute for the project, "Extension of Computer-Assisted Team Training Through a Coordinated Lesson Scenario" dated September, 1977. That document discusses the rationale which engendered the design of the special team authoring directives in PLANIT. Therefore, the need for performing these operations will not be described or defended in this document but rather it will be assumed that the reader has already encountered these needs either through reading or practical experience, and is now looking for assistance on how to implement them in the PLANIT language.

In addition to the assumption that the reader is already aware of the special team capabilities that will be needed, it will also be assumed that the reader is an accomplished author of conventional PLANIT lessons. Authoring manuals already exist for PLANIT with respect to conventional lessons and, as was already mentioned, these conventions will form the bulk of the team lessons as well.

Finally, there will be instances where alternate methods are discussed which implement identical (or very similar) operations. In all cases, the first of the alternates will be the generally preferred method. However, the alternates will be discussed either because some necessary component of the main method may not yet be implemented, or because the alternate method may contain some variation which may make it preferred in some unusual cases, although it is a bit soon to attempt to sort out the usual from the unusual in such a new application.

Before discussing the various authoring guidelines, a summary of the present and planned team training authoring for PLANIT will be presented.

PLANIT AUTHORING DIRECTIVES FOR TEAM TRAINING

The team training authoring directives will be grouped according to those which are available for use in PLANIT, version 3.1 and those which will become available with version 3.2. Note that this distinction will be made in future sections of this document by the addition of a double asterisk (**) superscript appended to the directive in question. If the reader happens to have a PLANIT version 3.2 or higher, the distinction could easily be removed from the document with a little correction fluid.

Team Authoring Directives, Version 3.1

These authoring directives consist of a Common Matrix capability with the FETCH and PUT commands, a TERMINAL variable for identification of the user, and an extended DIAL feature so that it can be used within the lesson and during execution via the CALC mode of PLANIT.

The Common Matrix is a feature for placing a copy of any user-defined matrix into position so that it is available for inspection and use by all members of the team. Each works on his or her own copy of that matrix~~, and if~~ the matrix is to be updated, it occurs by writing the locally-updated version back over the Common copy. The directives are used via CALC and include:

```
PUT X

FETCH X
```

where "X" is any locally-defined matrix. In the case of the FETCH command, the dimensions of the local "X" matrix must exactly match the dimensions of the matrix which was most recently PUT into the Common space. Any dimensions are acceptable for the originating matrix so long as the Common space can accomodate the total matrix space requirement.

In general, the FETCH and PUT directives provide opportunity for the interaction of several team members with a common data base. This data will always be numerical although the numbers might be interpreted to index some alphanumeric sequence.

The Common Matrix will be "common" to those team members who have begun their instructional sequence in a common PLANIT lesson module wherein the Common Matrix if first defined, i.e. is PUT into the Common space. A sufficient definition of a Common Matrix can be accomplished simply by declaring any matrix of appropriate dimensions (whatever the lesson will need), and then PUT that matrix name in a CALC command. This will preset the Common Matrix space with zeros and cause the identification of that Common Space to follow each of the team members into any sequence of lessons they might encounter. Any subsequent FETCH command will reference that Common Matrix. Strategies for achieving desired results will be discussed in the guidelines to follow. Note that it would be equally appropriate to preset any desired values in the matrix prior to the initial PUT operation to preset non-zero values in the Common Matrix.

DIAL t 'Message.'

The DIAL command which has long been available in the Command mode of PLANIT is also available as a CALC directive. However, when used in CALC, there are certain additional features and restrictions which are not true in the Command mode version.

First, the "t" following the word, DIAL, can be a literal terminal number (as in the Command mode), or can alternately be a CALC variable name whose current value is equivalent to the desired terminal number.

As is true of most CALC directives, the DIAL command can be used within the lesson scenario, following the "C:" prefix, or can be used in real time by the trainee who uses the appropriate prefix control character to gain access to CALC. Thus, both the lesson and the trainee can send messages to other team members (or other PLANIT users in general) using the DIAL directive. Note that PLANIT must be installed in such a manner that it controls the time-sharing of its terminals for this DIAL command to function. Also, the installation parameter, "WRITETHRU" will require consideration in order to make the DIAL command function in the manner desired at the local site. These considerations are discussed in the installation information.

Two name particles have special meaning when used immediately after the DIAL command, in place of the "t" in the example: OP and ALL. OP will refer to the PLANIT operator and ALL will designate all current PLANIT users. The latter (ALL) will only be allowed if the user is the PLANIT operator.

Finally, the enclosing prime (') delimiters are suggested rather than required. Use of the DIAL command in CALC causes certain of the characters to receive special consideration which is not true of the Command mode equivalent. However, if the message is enclosed in primes, any character (other than a prime) can appear in the message without causing undesirable execution. Also, the 3.2 version of PLANIT is expected to require the use of the delimiters. Therefore their use is also recommended prior to 3.2.

TERMINAL

The TERMINAL particle name is a system-defined name for the current terminal. This same number will appear at login time and is the one to be used by another PLANIT user who desires to DIAL a message to this terminal. It was added as a part of the team training package since this is the first application that specifically requires the identity of one terminal to be known at another. The identities are normally exchanged between participating lessons through the Common Matrix, where designated elements in the Common Matrix are given the value of TERMINAL when the new member signs on. TERMINAL is used in CALC in exactly the same manner as such variables as FRAME, TIME and RESPONSE.

Team Authoring Directives, Version 3.2 (New and Revised)

The following features are being added to PLANIT to enhance the Common Data Base, simplify synchronization of team members, make the DIAL command correspond to standard CALC syntax, and provide failsafe escape routes for lessons which may be caught in an infinite loop.

PUT REPLY

FETCH REPLY

The operation of these two command forms is completely analagous to the PUT X and FETCH X forms for the Common Matrix. The difference is in the fact that the REPLY particle names a series of buffers each of which may contain any arbitrary single line of characters. Thus, this new feature adds the alphanumeric dimension to the Common Data capability. Note that this does not replace the Common Matrix features, but rather it enhances them by providing a Common capability for non-numerical data. It is assumed that the reader is already familiar with other uses of the REPLY buffers, or can become so by reading the PLANIT authoring manuals.

One additional new use of the REPLY buffers completes the plan for Common Alphanumeric Data:

```
SET REPLY(n)='composite expression'
```

where "n" is any literal buffer number (or expression which evaluates to a literal buffer number) that is within the range of admissible buffers, a parameter that is determined at installation time. The 'composite expression' is any mix of delimited character strings and CALC values, separated by semi-colons (;). The 'composite expression' is identical to the forms permitted in the composition of the PRINT statement in CALC where alphanumeric strings can be mixed with CALC values to compose a single message.

Initialization of the Common REPLY buffers is the same as for the Common Matrix, where the first PUT command must be encountered by all team members in the same initial PLANIT lesson, after which the identification of the Common REPLY buffers will follow the team member regardless of the subsequent routing through lesson modules.

The DIAL directive in CALC is being changed to the form:

```
DIAL t 'composite expression'
```

where "t" is the terminal designator as before. The change is in the 'composite expression' portion which becomes identical to that shown above for the SET REPLY directive. Note that DIAL messages which are enclosed within delimiters in the format cited earlier will work without change in this one, motivating the recommendation that all uses of DIAL in the lesson follow that format regardless of the version number.

A new synchronization feature is used to coordinate the activities of all of the team members. The SYNC command form is:

```
X=SYNC(T1,T2,···,Tn;N)
```

where "X" is any simple CALC variable name, T1,T2,···,Tn are terminal numbers denoting team members (or expressions which evaluate to said terminal numbers), and "N" is a lesson-assigned number that is positive and increasing in magnitude each time it is used in the lesson.

The function of this directive is to synchronize the execution of all team members of a team lesson at a given point designated by an N value. Each team member will be held at that point until all have arrived and then will progress together. The value returned to X will be the highest N value posted to that time by any team member. Knowing this value allows the lesson logic to place out-of-sync team members back into step with the other members.

The N value can be a positive integer or have up to two decimal places. PLANIT lesson frame numbers would be a logical value to use for N. An N value of zero (0) has a special meaning in that it is used to temporarily drop the synchronizing relationship with the other team members until such time as another SYNC call is made with a non-zero N value. While the sync relationship is suspended, the remaining team members will proceed in their normal synchronizing relationship without regard to the one who has dropped out. The suspended member can be re-positioned in the lesson sequence by using the value returned from a SYNC call to determine the present team location.

Variations of the SYNC call include:

```
X=SYNC(T1,T2,···,Tn)
SYNC(T1,T2,···,Tn;N)
SYNC(T1,T2,···,Tn)
```

where omission of the X value means that no return value is desired, and omission of the N value means that N will take the value of the next whole number larger than the current X value.

In addition to the above, the SYNC feature will also function to aid the initialization of the team exercise, to initially assemble the team members, and to inform other team members when one leaves the team exercise.

The final new addition to the team repertoire is not directly related to team training at all, but rather is occasioned by the kinds of problems often encountered in authoring team lessons. This feature is designed to break infinite lesson loops. The command format is:

ESCAPE(n)

where "n" represents the number of processing seconds to allow between terminal inputs. In the absense of an explicit command, a default condition of 60 seconds would be allowed. If that processing time is exceeded, the user would be so informed, warned of a possible lesson problem, and logged out.

The need for the ESCAPE command stems from an often-used strategy in team lessons, that of monitoring some event within a lesson loop. Using this strategy, it is very easy to find oneself in an infinite loop from which manual escape is difficult, requiring the intervention of the PLANIT operator. Meanwhile, processing charges accumulate and service degenerates for the remaining users. The ESCAPE feature provides an easier means for breaking such loops, in fact they will be broken automatically unless the lesson author should specify such a large ESCAPE value that would render the feature ineffective, which might be important if unusually long processing was to be expected at that point in time.

Summary

Special team authoring features in PLANIT, both extant and planned, have been presented only to the extent of describing the command forms and the basic purposes for each command. The next section will attempt to describe how these commands are used to implement team training strategies within the PLANIT system.

TEAM LESSON STRATEGIES

The PLANIT author who is accustomed to writing lesson scenarios for individual execution will find some reorientation of thinking necessary when dealing with several (two or more) participating team members who are interacting with the lesson and each other. Certain PLANIT features which work very well and are easy to use in the individual setting become much more complex for teams. The concept of re-entry into the lesson is one such example such that the punctuated (dotted) frame type re-entry provision may not produce the desired results, i.e. might not get the team back together and into the lesson again. It would certainly not be a sufficient provision by itself.

Thus, this discussion of team lesson strategies will be structured around several components which seem to be present, to some degree, in all team authoring projects.

Initialization

Most lessons seem to require some initialization of data, mostly CALC data. Once initialized, the data follow the trainee in the form of a "student record" such that the training session can be interrupted for any length of time and yet allow the trainee to resume wherever the author provides. There is rarely a need to change the conditions of the initialization when the trainee resumes. However, in the case of team training, re-entry into the scenario following an interruption can be a problem since some of the initialized data may no longer be valid. Specifically, assigned terminal numbers will often be a necessary part of the initialization process for team lessons, and these could easily change after an interruption.

Also, from minimal experience with team lessons, it would seem that team sessions would normally start from the beginning of the scenario whereas individual lessons normally resume at some point near the interruption.

And, after current teams complete their exercise and a new team is about to take their place, there may be some residual, unwanted data left in the Common Data Space (Common Matrix and Common REPLY Buffers) that needs to be disposed of first. In this case, although new trainees sign on with clean student records, there will often be some residual team data yet remaining.

In considering methods for use to initialize the data spaces appropriately, one must avoid initializing too much or too often. For example, suppose a three-member team is attempting to sign on to a team lesson scenario which incorrectly initializes all data unconditionally for each new trainee who starts. In this case, each new addition to the team would erase from the Common Data Space all record of other members who had signed on, leaving a team composed of one member, the last one who signed on. Thus, initialization must be controlled to occur only when it is needed and not when it will adversely affect lesson execution.

Author-Controlled Initialization. The author can assume responsibility for initializing the Common Data Space prior to each new team who signs onto the system. This can be done by signing on in the Author Mode, switching directly to CALC, declaring an appropriately dimensioned matrix, blanking all of the REPLY buffers, and then use the PUT command to update each in the Common Data Space. (Whether either or both must be initialized will be determined by their use in the scenario).

This method will accomplish the necessary initialization to the point that simple lesson logic can complete. However, this method obviously requires the presence of the author (or a trained monitor) at all sessions, negating some of the important benefits of using the computer in the first place.

Trainee-Controlled Initialization. This uses the first question in the scenario to ask if this one is the first member of the present team to sign on. If so, the lesson logic proceeds to initialize the Common Data Space via CALC lesson commands. Otherwise, the initialization step is skipped.

This method works quite well if the team members are geographically together and have voice communication. Otherwise, a well-meaning team member might answer incorrectly and erase needed data. This would require careful queueing in advance.

<u>Lesson-Controlled Initialization</u>. This would depend on the capability of making the lesson detect the first team member to sign on to perform the needed initialization. There are probably several ways to do this but two will be discussed.

The most direct method is with the SYNC** command where a PLANIT Decision frame would contain something like the following statements:

```
C:SET MATRIX(X,100)
C:FETCH X
IF SYNC(X(1),X(2),X(3);0) EQ 0
Proceed to initialize.
END
Proceed with the lesson.
```

In the above example, an assumed three team members whose identity is assumed to be in the first three entries of the Common Matrix is tested in the SYNC** command with a zero synchronization number for the purpose of retrieving the highest presently posted synchronization number. That number will be zero until the first team member posts a non-zero value, which presumably would happen very soon into the lesson. Thus, only the first team member to sign on would find a zero value, and that would cause the initialization to occur. Note that a zero (0) was given for the synchronization number (after the semi-colon) so that this occurance would not hold the user here until the others caught up. In this case, execution is intended to proceed.

Another method would use coded values in the Common Matrix to indicate the present status. Suppose X(4) was used to hold the value 0, 1 or 2 with the following meanings:

0 = Initial state, no team members yet signed on.
1 = At least one team member has signed on and waiting for the others.
2 = The team is assembled and the lesson is in progress.

With that scheme, the following Decision frame statements would be appropriate:

```
C:SET MATRIX(X,100)
C:FETCH X
IF X(4) NQ 1
C:X=0*X
C:X(4)=1
Proceed to initialize.
C:PUT X
END
Proceed with the lesson, changing X(4) in the Common Matrix
to the value, 2.
```

This will accomplish about the same as the SYNC** form (above) except that there is a potential for confusion if the team members should for some reason sign off while the value of X(4) was 1. The lesson would have to be written in such a way that no additional initialization would be needed when the next team tried to sign on.

One final consideration is the existence of the Common Matrix at the beginning. It would seem that all cases will use the Common Matrix to communicate the identifying numbers of the participating team members' terminals to the other participants. That being the case, one of the first commands will be a FETCH of that Common Matrix. On the very first attempt to execute that lesson (or after the lesson is brought in from cards), the Common Matrix will not yet exist, having not yet been PUT in place. This can be avoided by a one-time effort on the part of the author to declare an appropriate-size matrix interactively in CALC while signed onto the first of the team lesson series, and them PUT the matrix into the Common Matrix space. However, beginning with PLANIT version 3.2, this problem will no longer exist since a missing Common Matrix will be regarded the same as if all its entries were zeros.

Re-Initialization

By the term, re-initialization, is meant the process whereby all pertinent data are returned to appropriate starting values when a new team signs on.

Several of the conditions for re-initialization were already covered in the discussion on initialization, above. Each of the described methods would also encompass proper initialization for the next team. However, there are at least two additional considerations that may or may not be of concern during the process of re-initialization.

Data Readout. It will be normal during team execution for data to accumulate both in the individual team member's student record and in the Common Data Space (Common Matrix and Common REPLY** Buffers). Although the student records will be left intact following the team exercise, the Common Data Space will be used over again by the next team, changing the data which were left by the previous team. What the lesson should do with that data depends on the training objectives. If the data are no longer needed, the space can simply be re-initialized when the new team signs on (as is assumed in the examples of initialization, above). Or, the data could be allowed to "age" from one team to the next, that is, only selected entries pertaining to the identities of present team members would be re-initialized, while the remaining data space would retain the values from the previous exercise, accumulating the experience of the team sessions. Another alternative would be to display the Common Data from a team session prior to the next exercise. Each of these will require different re-initialization considerations.

Case 1 will be handled effectively by either of the initialization examples shown above. When the new team signs on, the data left over from the previous team is lost (i.e., set back to zero).

Cases 2 and 3 will require some kind of coding similar to the use of X(4) in the above example so that the lesson can assess the previous status before overwriting the data space. For example, to reset only selected entries as in case 2, the following Decision frame statements would work:

```
C:SET MATRIX(X,100)
C:FETCH X
IF X(4) EQ 0
New start, initialize everything.
IF X(4) EQ 1
Already initialized and in process of assembling the team
IF X(4) EQ 2
Re-initialization.  Reset only the appropriate entries.
END
C:X(4)=1
C:PUT X
Proceed with the lesson.  Change X(4) to 2 after the team
has been assembled and the training begins.
```

Case 3 would be a variation of this example. Instead of selectively re-initializing for X(4) equal to 2, appropriate action would be taken to preserve the data. This might take the form of a display of the data (to be given to the author), or a message which informs the user that the data from the last exercise has not yet been retrieved, and then end the execution with a C:FINISHED statement. After retrieving the data, the author would initialize the Common Matrix to ready it for the next team.

Team Interference. While the team is being assembled, two members cannot be allowed to get assigned to the same position, and after the team has assembled and is executing, no other person can be allowed to sign on to that team sequence and interrupt their execution.

A simple solution to these potential interference problems makes use of the earlier system for designating elements within the Common Matrix to represent the terminal numbers of the participants. In the initialized condition, these entries would be zero. The earlier example used three-member teams and the first three entries to represent them. Continuing that example, if the entry is still zero, that position still needs a team member.

A PLANIT Question frame can be used to ask which position in the team the person wishes to occupy. Having evaluated the choice, a Decision frame then checks the corresponding entry to determine if it is still vacant and, if so, assigns the TERMINAL value to it. If not, the user is so informed and given an opportunity to choose a different position or is signed off. However, if each of the entries is already non-zero, the user would not be allowed to start, but rather would be informed that all the positions are taken and would be signed off. This logic can be seen in the example which appears at the end of this section and incorporates all of those things considered so far.

Replay. An additional re-initialization problem is encountered if the same student identity is to be reused for a subsequent team exercise over the same lesson scenario. There will already exist records for that member which direct PLANIT to the place in the scenario where the person should resume. However, in the case of the team scenario, there will first be a need to reassemble the team.

All of the various options that might be desired for this situation are too numerous to cover here, and each will demand some logical variation in the lesson scenario to implement it. Three cases will be described.

Case 1 might be where every new session is to start from the beginning of the series. In that event, the first frame of each lesson module should be punctuated by "dotting" the frame type. For the first lesson module in the series, whatever frame is first would be dotted. In subsequent lesson modules, the first frame that is dotted would be a special Decision frame, as follows:

```
FRAME 1.00 (D.)

G2. CRITERIA
IF LINK(1) EQ n  B:START
LINK(1)=n
```

The value "n" can be any number so long as it is non-zero and different in each lesson module. The name START is to be replaced by the name chosen for the first lesson module in the series. No other frames should be punctuated.

Case 2 might be where the same person is not to be allowed back in to the series. In that case, use the same logic but replace the "B:START" with "F:YOU ARE NOT ALLOWED TO START AGAIN. C:FINISHED". This frame must also be the first to be executed after the team has been assembled.

Case 3 might be where the entire team should resume approximately where they left off in the previous session. In this case, the above special Decision frame would be interspersed frequently throughout the lesson sequence at every reasonable restart point, and with different test numbers for each and every occurrance of the frame. However, instead of "B:START", the branch would be to "B:ASSEMBLE" which would execute a sub-lesson module to reassemble the team before returning to the next frame. In this case, the same sub-lesson can also be called at the beginning.

Team Initialization Example: This example will be for a three-member team, the positions of which are designated RED, YELLOW and BLUE. Re-initialization starts over. Data are allowed to accumulate from session to session and from team to team. The first four entries of the

Common Matrix are used to control the participants. The remainder are available for other sues.

```
FRAME 1.00 (D.)

G2. CRITERIA
C:SET MATRIX(X,100)   C:FETCH X
IF SYNC(X(1),X(2),X(3);0) EQ 0
C:X(I)=0 FOR(I=1,3)
IF X(4) LQ 0 C:X=0*X
IF X(I) NQ 0 FOR(I=1,3)
F:SORRY, ALL POSITIONS ARE TAKEN. ANOTHER TIME. C:FINISHED
ELSE C:X(4)=1 C:PUT X  F:@ARE YOU RED, YELLOW OR BLUE?


FRAME 2.00 (Q)

G3. ANSWERS
0 KEYWORD ON
A RED
B YELLOW
C BLUE

G4. ACTIONS
A C:SET COLOR=1
B C:SET COLOR=2
C C:SET COLOR=3
- R:ANSWER ONE OF THE THREE, OR TYPE 'FINISHED'


FRAME 3.00 (D)

G2. CRITERIA
C:FETCH X
IF X(COLOR) NQ 0
F:SORRY, POSITION ALREADY TAKEN. CHOOSE ANOTHER. B:2
ELSE  C:X(COLOR)=TERMINAL  C:PUT X
C:SYNC(X(1),X(2),X(3);1)  B:RED,YELLOW,BLUE;COLOR
```

In the absence of the SYNC command, other logic could be substituted, which relies on coded values in the Common Matrix. However, the SYNC command provides computing efficiency which cannot be matched any other way.

Tracking Team Members

The activities of the participants in team training must interrelate in some manner or else there would be no common team objectives. This interrelationship will show up in the lesson scenarios as a need to synchronize the members' lessons at given points in such a way that they will all be at that point at a given moment.

There will be many variations to this need for lesson synchronization. In one case, similar lessons may be moving the trainee in lockstep fashion, synchronizing again after each interaction. In another, the synchronization points might be far apart, bringing the members together only at checkpoints. Another might only synchronize their initial departure. Yet another might keep some members in sync while one or more others drop out to work independently. There may be a need to bring members back into synchronization after they have been working independently.

The SYNC** command is designed to fulfill all of the above needs and a few more. Essentially, the SYNC** command needs only the terminal identity of the participants and a synchronization number to define its operation. In addition, it needs to be informed when a member should no longer be considered in the synchronization processing for some indefinite period of time. A zero (0) synchronization number serves this function.

The command format for SYNC** has already been shown. Using that format, any two or more PLANIT users whose terminal identities are named in the SYNC** command will be forced into synchronization at the point where the synchronization numbers from all of the participants agree. In the event that SYNC** commands are encountered which are different from others which have already been posted and are waiting, the highest posted number(s) will continue to wait and any which are less than the highest will continue to execute.

There is no need for any similarity to exist among the lesson sequences from which the SYNC** command is issued. Synchronization is based entirely on the synchronization. It is the responsibility of the author to make the fit between the corresponding lessons reasonable. The reason could be none other than assuring that all finish at about the same time. Or it could be for the purpose of exchanging data, responding to events which are dependent on the progress of another member or whatever.

Establishing Synchronization. The first consideration must be the initialization of those data items which will be used to control progress through the lessons. At a minimum, the members must be assembled into a team relationship and each member's lesson must be given the terminal identity of the other members so that synchronization requests can be made for the proper group of terminals and any desired communication can be addressed to the proper destination. The second consideration will be assuring that the proper team members (and only that number with no duplicates and none missing) have been assembled and the team exercise is ready to begin.

The example just shown contains all of these elements. This section will simply elaborate on the synchronization aspects of that example, and discuss additional synchronization considerations.

The first use of the SYNC** command contains a zero (0) synchronization number (which has been defined to mean that this terminal is dropping its synchronizing relationship with the remainder of the team). However, in this case, no synchronizing relationship is yet assumed to exist so the zero (0) parameter has no effect. But there will be a return value from the SYNC** call that will be important in that it will reveal whether any other member has reached a SYNC** call with a positive, non-zero synchronization number, waiting for others to catch up. If so, the return value will be that number and the lesson will have determined that the current member is not the first of the team to sign on. Otherwise, the return value will be zero (0), allowing the lesson to make any initialization necessary prior to the assembling of the team. Therefore, this call can serve an important team initialization function.

When the first SYNC** call is encountered with a non-zero synchronization number (as in Frame 3 of the example), the PLANIT system can evaluate the status of each of the identified terminals. If they are not yet in team status, PLANIT will ask the user,

```
TEAM MEMBER MISSING. DO YOU WANT TO WAIT? (Y/N)
```

If the answer is "Y", that terminal will be held until all of the other team members encounter a similar SYNC** call in their own execution, and the last to encounter it would unlock the execution for all participants. It will be clear to any who study the example in detail that only the last member to sign on will have the correct terminal identities for all of the team members.

However, this will not pose a problem for the earlier ones because PLANIT will detect the improper terminal number and use that information to decide that the member is missing. When the last member signs on, PLANIT will then have all the correct terminal numbers so that the correct synchronization check can be made.

Therefore, this provision in the SYNC** feature will enable PLANIT to automatically collect the team members. Notice that it will also detect if one should leave the team exercise by signing off without first notifying the other members. This will avoid the prospect of team members, waiting in frustration for another member to catch up who is no longer on the system.

Maintaining Synchronization. It is simply a guarantee within the PLANIT system that wherever a SYNC** call is made in the lesson, execution will not pass that point until the lessons of all listed terminal numbers have made SYNC** calls with the same synchronization number or at least one has made a SYNC** call with a higher synchronization number. There is no restriction about the nature of the lesson out of which the SYNC** call is being made. Therefore any lessons can be synchronized at any point.

Leaving The Team Temporarily. It has already been shown that one can be temporarily dropped from the synchronized team relationship by making a SYNC** call with a synchronization number of zero (0). Actually, this call has no effect on the execution of the lesson that makes the call; execution continues as though nothing had happened. The purpose of the call is to mark that terminal so that others who are in a synchronized team relationship with it will not be waiting for it to catch up (i.e. to make a SYNC** call with the highest posted synchronization number). Instead, that terminal will be dropped from synchronizing consideration even though that terminal number appears in the list of another SYNC** call. Thus, the person at the terminal can proceed independently without holding up other team members, and will do so until a SYNC** call is made from that lesson with a non-zero synchronization number.

Re-Establishing Synchronization With The Team. This can be done very simply by issuing a SYNC** call with a non-zero synchronization number. If the number is less than the highest existing one, then all will wait for this one to catch up. If the given synchronization number is the highest, then this terminal will wait for the others to catch up.

However, it is fair to assume that the team will have progressed by an unspecified amount while the one member was working independently, and it may be more important to rejoin the team at the point where execution would normally have been had he not dropped out of the team. In this case, a list of frame numbers corresponding to synchronization numbers will make it possible to immediately rejoin the team at exactly the right place. For example, suppose the array, SYNCNO, contained a frame number for each synchronization number, where the synchronization number was an integer to be used as a subscript into the array. Then, the following statement would put the member back into the team at the right point:

```
B:SYNCNO(SYNC(X(1),X(2),X(3);0))
```

The example shown above assumes three members in the team whose terminal numbers are the first three entries of array X. It would also be possible to use the frame numbers as the synchronization numbers. Then, no SYNCNO array would be needed; the statement would be simply:

```
B:SYNC(X(1),X(2),X(3);0)
```

Notice that the return value from that SYNC** call will mark the progress of the team.

Re-Synchronizing The Entire Team. There may be situations where a team exercise is not finished in one session and all team members sign off with the expectation of resuming at some future time at approximately the point where they left off. If it can be guaranteed that each of the team members will be using the same terminal number when they resume as in the prior session, then no special provision needs to be made. PLANIT's normal features for resuming the lesson will be enough to handle the task automatically. When the next SYNC** call is encountered, each member will be automatically held in place until the complete team is assembled again.

However, if the terminal numbers may vary from session to session, that adds a new dimension of difficulty to the problem. It would mean that the terminal number values in the Common Matrix will need to be reassigned so that the parameters of the SYNC** call will be correct. Otherwise, there could be a call for a synchronizing relationship with terminals which are not executing that team lesson at all.

The solution to this problem, if it becomes a problem, can be devised through some control logic in the lesson. One such solution will be illustrated; many more are also possible.

Assume that LINK(1) is used to represent the current synchronization number throughout the sequence of lesson modules. That would make the the value of LINK(1) continually increase as the lesson progresses. Also assume that LINK(2) holds the designation of the participant's role ("COLOR" in the previous example). Then the following Decision frame, if included at the beginning of every lesson module and was the only punchtuated (dotted) frame in that module, would perform the necessary reassignment of terminal numbers and would reposition the trainee appropriately in the lesson. (Note that the number '78' is supposed to be the first synchronization number of this module and the TOPIC array contains frame numbers corresponding to appropriate reentry points for each of the synchronization numbers which are within the range of this module.)

```
FRAME 1.00 (D.)

G2. CRITERIA
C:SET MATRIX(X,100)  C:SET MATRIX(TOPIC,50)
C:TOPIC(1)=ARRAY(2,5,8,11,12,15,19,20,23,25,29,31,33,36)
IF LINK(1) LS 78  B:2
ELSE  C:FETCH X  C:X(LINK(2))=TERMINAL
IF SYNC(X(1),X(2),X(3);0) GR 1
F:ANOTHER TEAM ALREADY IN PROGRESS.  C:FINISHED
ELSE  C:PUT X  C:SYNC(X(1),X(2),X(3);1)
B:TOPIC(LINK(1)-77)
```

The values in the TOPIC array would be adjusted to fit the structure of the lesson module in which it was used. The values "77" and "78" would be changed to agree with the first synchronization number in the current module. Before each SYNC** call in the remainder of the lesson module, the LINK(1) entry would be assigned the corresponding synchronization number. For example:

```
C:LINK(1)=78  C:SYNC(X(1),X(2),X(3);78)
```

This lesson logic would enable the automatic reassembly of the team where all members would keep their originally chosen roles. Their new terminal numbers will replace the previous ones in the Common Matrix, and the SYNC** calls will prevent them from interrupting another team that may be using the lesson, as well as holding them in the first frame until the entire team has assembled. The author can choose frame number values for the TOPIC array which cause the team members to resume at the most reasonable place, which may not necessarily be the exact place where they previously left off.

Placement of Synchronization Checkpoints. Where the SYNC** calls are placed in the lesson modules depends primarily on the content and where it might be important for team members to be brought together in time. However, some simple guidelines will be useful, and the examples in the next section will also pertain to this subject.

If the training lesson is structured in such a way that all participants execute ~~the same lesson~~ sequence, then the correspondence of SYNC** calls will be a trivial matter since each participant will encounter the same SYNC** call in the same position due to the fact of executing the same lesson.

If the subject matter calls for participants in different roles to execute separate lesson sequences, then corresponding SYNC** calls will be placed in the different sequences where the context demands. SYNC** calls correspond if the same terminal values are designated in the list and if the same synchronization number value is used.

In addition, there will be cases where two or more participants will be required to take turns in their interaction with the training materials. This will be enforced through proper use of corresponding SYNC** calls. In general, it will require two sets of corresponding SYNC** calls to enforce a "turn." The participant who is waiting will execute two SYNC** calls in immediate succession, one to inaugurate the other's turn, and the next to hold execution until the other's turn has completed. Meanwhile, the other participant will also encounter two similar SYNC** calls but they will be separated to bracket the lesson logic which will govern the activity during his or her turn. A sample of this kind of logic appears in the example which is included in the next section, where the "turn" shifts from one participant to the other, as indicated by the three sets of corresponding SYNC** calls.

LESSON STRUCTURING

Lesson structuring is intended to refer to the number and organization of lesson modules to be used in the complete training package.

Many of the lesson structuring decisions will be a simple matter of author preference. In general, considerations which go into any PLANIT lesson structuring process will be true of team lessons as well. Lesson modules are often chained in order to obtain the total number of frames desired, which normally exceed the total number allowed in a single lesson module. Another common structure is where each lesson topic is contained in a lesson module (or series of modules) and these are called, almost in table-of-contents fashion from the supervising lesson module. This method is particularly efficient if pretest questions are included in the supervisory lesson to first determine whether the trainee needs to improve in the areas covered by the sub-module. These are decision factors which are true of all PLANIT lesson-writing efforts.

A few additional considerations are present for authoring team training scenarios in this matter of lesson structuring. The decision factors affecting how the modules are to be structured deal mainly with the nature of the role that each of the participants plays. If the various roles are substantially different, it is usually advisable to form separate lesson sequences for each role (as was shown in the previous example). Note however that the sequences must all begin in a single lesson module and branch from there in order to retain a common identity for the Common Data Space. On the other hand, if any of the participants perform identical roles (or very nearly so), they may share the same lesson modules.. If role differences are minor, tests can be made in the scenario to differentiate the roles, such as:

IF COLOR EQ 2 Do something special for COLOR=2.

Very many such tests would soon make it more efficient to separate the lesson modules by role, especially because of the rate at which this would consume frames plus the added inefficiency of having to execute the test.

Conditions Affecting Data Transfer In The Lesson Structure

There are only two kinds of data which can be transferred between lesson modules, LINK array data and Common Data Space data (Common Matrix and Common REPLY** Buffers).

The LINK array, usually dimensioned at 10 elements (which can be changed at installation time), contains the only data which follow execution from one lesson module to another. This is true in all PLANIT applications, not just team training. However, the LINK data only follow a single trainee. Each trainee has a different copy of LINK which moves from module to module with lesson execution. If a LINK entry is changed in one module, it will retain that new value in all subsequent modules unless it is changed again. That value will not appear automatically in another trainee's LINK array though.

Another feature distinguishes the LINK array, making it useful for special applications. The LINK entries are the only originally-defined named entries which can receive different values (as compared for example to FRAME, TIME, RESPONSE and PI, which receive their values only internally in the system). Therefore, a LINK entry is very useful for an indicator variable. For example, suppose there is a need to determine whether execution has passed a given point. A sample use of LINK might be:

```
FRAME 1.00 (D)

IF LINK(1) NQ 0 Execution has passed this point before.
ELSE C:LINK(1)=1
Execution has not passed this point before.
```

Of course, this example assumes that LINK(1) has not been changed from its initialized zero value prior to this frame. Note that this test would not be suited to a user-defined name because on the first encounter of that name the fact that it had not yet been declared would make the lesson stop, showing an error. And, declaring the name in a prior lesson would not carry forward to this one. Thus, LINK serves this function quite well.

The Common Data Space (composed of the Common Matrix and Common REPLY** Buffers) serve a different function from the LINK array, primarily because that data is available to all members of a team regardless of the lesson module that happens to be executing. Recall that the definition of members who belong to a single team encompasses all who have begun the training sequence through one common lesson module wherein the Common Matrix and Common REPLY** Buffers were first PUT into the Common Data Space.

Another important difference between the LINK array and the Common Data Space is the fact that the entries in that space are never manipulated by the lesson or user. Rather, a local copy of a matrix or REPLY buffers is PUT into that space, overwriting whatever was currently there. If the values are to be examined or updated, they must first be FETCHed and then be PUT back again (in the case of an update). When updating, the FETCH and PUT should occur in the same group of a single frame with no intervening opportunity for delay (such as a terminal input request). If that guideline is followed, PLANIT contains provisions which will inhibit any other user from interrupting the FETCH-PUT sequence before the update has been completed. The data arrays in the Common Data Space are unnamed, and therefore not referenced by name. The name to be used after the FETCH and/or PUT will be a local name only, designating the local copy within which the work will be done.

Thus the Common Data facility allows data to be moved freely among participants in the team. The author must take into consideration the fact that the entries in that Common Data might be changed in different ways by different participants. For example, suppose that the first entry (e.g. X(1)) were set aside to hold the terminal number of the participant and there were three participants. Then the first entry would finally hold only the terminal number of the participant who most recently updated it. This is in contrast to LINK(1) which will remain unique for each participant. It is for this reason that the prior example uses the item "COLOR" to subscript the data from the Common Matrix in order to distinguish each of the participants.

Note that the possibilities for using the Common Data seem almost limitless. Numbers which are PUT by one team member can be FETCHed and used to affect the execution of another. Prior to the addition of the SYNC** command, Common Matrix codes were used to synchronize team members in much the same way as the synchronization number does in SYNC**. A Decision frame tests the code numbers in the Common Matrix which were PUT there by each of the participants and holds each at that point until all the numbers agree.

In the case of the Common REPLY** Data, combining that with the SET REPLY and USE REPLY commands enables the passing of character strings for almost any purpose whatever, from simple messages to having one participant respond to another participant's questions. It would even be a simple matter to have one participant execute two or more different lessons concurrently, passing the responses through the Common REPLY** Buffers. Messages sent from one to another could be examined along the way and the identity of the content could be stored along with the student records. The variety possible makes examples very difficult. Let's choose the case where one member's lesson asks a different member to respond to one of its questions, evaluates the response and provides feedback to both members. Suppose the originating member is RED and the other member is BLUE.

The implementing lesson sequence for <u>RED</u> will be:

```
FRAME 10.00 (D)

F:BLUE IS BEING ASKED WHAT YEAR COLUMBUS SAILED WEST TO
F:FIND INDIA. WAIT A MOMENT UNTIL BLUE HAS GIVEN AN
F:ANSWER.
C:SET REPLY(1)='WHAT YEAR DID COLUMBUS SAIL WEST TO FIND INDIA?'
C:PUT REPLY  C:SYNC(X(1),X(2);10)  C:SYNC(X(1),X(2);11)
C:FETCH REPLY  C:USE REPLY(1)

FRAME 11.00 (Q)

G3. ANSWERS
1+ 1492

G4. ACTIONS
F:BLUE'S REPLY, "$
C:PRINT REPLY(1) $F:" WAS $
F:
1 C:SET REPLY(1)='RIGHT.'  C:PUT REPLY  C:SYNC(X(1),X(2);12)
- C:SET REPLY(1)='WRONG.'  C:PUT REPLY  C:SYNC(X(1),X(2);12)
```

The corresponding implementing sequence for BLUE will be:

```
FRAME 20.00 (D)

G2. CRITERIA
C:SYNC(X(1),X(2);10)  F:YOU WILL NOW GET A QUESTION FROM RED.
C:FETCH REPLY  C:PRINT REPLY(1)

FRAME 21.00 (Q)

G3. ANSWERS

A  DUMMY

G4. ACTIONS
C:SET REPLY(1)  C:PUT REPLY  C:SYNC(X(1),X(2);11)
C:SYNC(X(1),X(2);12)  C:FETCH REPLY  C:PRINT REPLY(1)
```

One possible execution sequence would be (assuming that BLUE gave the correct answer):

RED's terminal:

```
BLUE IS BEING ASKED WHAT YEAR COLUMBUS SAILED WEST TO
FIND INDIA. WAIT A MOMENT UNTIL BLUE HAS GIVEN AN
ANSWER.
BLUE'S REPLY, "1492" WAS RIGHT
```

BLUE's terminal:

```
YOU WILL NOW GET A QUESTION FROM RED.
WHAT YEAR DID COLUMBUS SAIL WEST TO FIND INDIA?

1492
RIGHT.
```

Notice the use of the SYNC** commands. They define the points at which the lesson executions must be synchronized in order for the sequence to come out correctly. The three synchronization points can be described as follows:

SYNC Point 10:	Waiting to start the sequence together at which time BLUE will receive the question and be given time to answer.
SYNC Point 11:	Waiting for BLUE to answer at which time RED will evaluate the answer and set up a feedback message.

SYNC Point 12: Waiting for RED to set up the feedback message at which time both RED and BLUE will continue their respective lessons.

This same sequence could also be implemented using only number codes in the Common Matrix but with much more lesson logic and requiring several times as much computer processing time. Using the above example, the computer processing time will be little if any more than each executing independently.

Note in the above example that BLUE's response, "1492" will be recorded in RED's student record in the frame number 11 entry. It would have been easy to have it also recorded in BLUE's student record by changing the Group 3 entry of Frame 21 from "A . DUMMY" to "1+ 1492" as in the RED Frame 11. However, that would suppose that the BLUE lesson was aware of the particular question that would be asked, making the whole exercise much less meaningful. Suppose, on the other hand, it was important for the BLUE lesson to record that response and did not know the question in advance. That would still be quite possible by adding another Question frame to the BLUE lesson such as follows:

```
FRAME 22.00 (Q)

G3. ANSWERS
0  USE REPLY(1)
A+ RIGHT.
```

This would simply evaluate the feedback from the RED lesson, and then the student record for Frame 22 would contain the right/wrong information for the response in Frame 21.

It soon becomes obvious that there can be few "canned" sequences for desired operations. Rather, the operation itself must be examined and the sequence is developed accordingly.

These same kinds of interactions can also take place with larger teams of three, four, five, etc., members but the complexity of the synchronization can grow, too. The subject of synchronization has already been discussed in the prior section.

Inter-Terminal Communication

In conventional CAI, inter-terminal communication is seldom considered to be a factor of the lesson structure since any inter-terminal communication is usually incidental to the lesson content. However, inter-terminal communication in a team training context will often be an integral part of the lesson structure, content and strategy.

Inter-terminal communication can be expedited in two ways, directly through the DIAL command and indirectly through the Common Data Space. In the case of the Common Data Space, the communication can be free-form through the SET REPLY(n) and PUT REPLY** sequence in one lesson and FETCH REPLY** and PRINT REPLY(n) in the other, or it can be a prepared message list, passing only the message number across terminals and using that to display the desired message.

Several considerations will suggest the most appropriate method to use. First, the DIAL command is immediate. There is no provision to time the display of that message at the target terminal to correspond with the execution of that lesson. SYNC** calls strategically placed could introduce that timing however.

Second, there is no provision to allow the examination of the content of the DIALed message by either lesson.

Third, using the DIAL, the desired message must be prefixed by the appropriate command form, and enclosed in primes (or quotes). This will be true whether the message originates within the lesson or in real time by the trainee.

Fourth, the DIALed message automatically carries a "FROM n" prefix to identify the source of the message.

Fifth, the DIALed message has very little impact on execution time.

Sixth, using the REPLY** buffer facility, messages can be sent which are subject to examination prior to transmission and/or after reception and display. They can be identified as to their source, or not, at the discretion of the author. They can be transmitted and held until the receiver is ready for them, or SYNC** calls can time the sending and receiving (as in the case of the DIAL). Since the communication is through disk blocks, there will be modest impact on execution times in the lessons.

If trainees are to send messages through the REPLY** buffers, Question frames must be designed to collect and send the message and more lesson logic must be prepared on the receiving end to retrieve and display the message, whereas the trainee can send a message via DIAL outside the framework of the lesson and it will be displayed at the other end without the need for advanced preparation.

Finally, there might be a need for sending the message from the trainee to the lesson, to be retrieved and displayed by the author at the end of the session along with other performance data. This can be accomplished by holding the message(s) in the REPLY buffers of the individual student records (without executing a PUT REPLY**). Each participant would have separate REPLY space. However logical problems could develop if that strategy was combined with PUT REPLY** and FETCH REPLY** sequences since the buffer areas would be written over by other data.

Whether the DIAL or REPLY buffer method is used to convey messages between terminals will depend on the particular circumstances of the message. Typically, if the purpose of the exchange of messages is for an unmonitored conference, the DIAL facility is most suitable. However, if the message must coincide with lesson execution at both terminals, if it must reach its proper destination without expecting the sender to put the address prefix on it (DIAL n), if it is to be enqueued and held until the receiver calls for it (or it is called for by the receiver's lesson logic), or if it is to be examined by lesson logic prior to transmission and/or following reception, then the REPLY** buffer method is appropriate.

Examples of Two Message Transmission Methods. The following two examples will illustrate the sending and receiving logic for the DIAL and REPLY** buffer methods of message transmission.

Trainee-Originated DIAL message:

```
DIAL 3 'THIS IS THE MESSAGE.'
```

Lesson-Originated DIAL message:

```
C:DIAL 3 'THIS IS THE MESSAGE.'
```

Reception of the DIALed message:

```
FROM 3 THIS IS THE MESSAGE.
```

Trainee-Originated REPLY buffer message:

```
FRAME 4.00 (Q)

G2. TEXT
TYPE YOUR MESSAGE TO 'RED.'

G3. ANSWERS
A   DUMMY

G4. ACTIONS
C:SET REPLY(1)   C:PUT REPLY
```

Lesson-Originated REPLY buffer message:

```
C:SET REPLY(1)='THIS IS THE MESSAGE.'   C:PUT REPLY
```

Reception of message in the REPLY buffer:

```
C:FETCH REPLY   C:PRINT REPLY(1)
```

Notice in the last line of the examples that a delay is possible, both before the REPLY buffer is FETCHed and before it is displayed via the PRINT command. Thus, the message could be queued, examined, modified, or whatever before the receiving party sees it. In fact, there may be no intention of displaying the message but rather to subject it to some other kind of processing as in the example on pages 26 and 27.

LESSON MAINTENANCE

Team lesson applications add very little to lesson maintenance which does not already apply to conventional lessons. The main difference is in regard to the Common Data Space, and the manner in which the data there are to be used.

Setting The Lesson Up For Execution

The easiest way to initialize data spaces is to make the lesson expect them to begin with initialized values of zero (0) (or "null" in the case of the REPLY buffers). At most, this only requires the declaration of that data. However, lesson needs may dictate data entries which, in their initialized state, have a whole range of values. If the data are to be local to the lesson module, then the initialization logic must be at the beginning of each module in which it is being used. But, if the data will be kept and manipulated in the Common Data Space, then initialization of that data can be performed more economically in a special lesson module. However, if a special lesson module is to be used for that purpose, it is important that the author must sign on to the same beginning lesson module that the team members do so that the proper Common Data Space will get initialized.

A place that is often convenient for doing this Common Data initialization is within the beginning lesson module itself. The lesson organization is kept tidier if the first lesson module contains only that part of the training scenario that assigns participants to their respective sequences, as shown in the example on page 16. If that strategy is adopted, then that first lesson module will be very small.

Suppose that first lesson were extended, beginning with Frame number 4, where the frame was given some label of the author's choosing which would serve as a special password for lesson maintenance. Recall that the lesson does not automatically execute for the author as it does for a trainee. Therefore, the author can GET the lesson by name, and then give the command,

```
EX PASSWORD
```

where PASSWORD can be any label name that the author has chosen to direct execution to the extension of that lesson module. Also note that, due to the execution logic, there is no way for the trainee to enter that portion of the lesson module.

The author is now free to develop as much lesson logic as necessary to initialize the lesson data as it is designed to be. The example below illustrates such a plan.

Data Readout Following Team Execution

The same method that was suggested for setting up the data requirements in team lessons in the paragraphs above can also be used to display the pertinent data following the team exercises. The following example will illustrate this aspect as well.

Example Of Lesson Maintenance Frames

This example is meant as a continuation of the beginning lesson in the team training sequence such as was illustrated on page 16. Note that the author (and only the author) can choose to display data only, display and initialize the data, or initialize the data only. This sequence can obviously be tailored to suit local needs.

```
FRAME 4.00 (M) LABEL=PASSWORD

G2, TEXT
DO YOU WANT TO:

G3. ANSWERS
A. DISPLAY ONLY
B. DISPLAY AND INITIALIZE
C. INITIALIZE ONLY

G4. ACTIONS
C B:INIT
```

FRAME 5.00

This frame, or series of frames, will be designed to FETCH and display the appropriate data in any report form chosen by the logic. The frame (5.00) was not further identified because it might be a Decision frame which contains statements for declaring, FETCHing and printing data, or it could be a Question frame to provide choices among alternate display formats.

Remember that prior to the FETCHing of a Common Matrix, a local matrix must be declared into which the Common Matrix values are to be copied, and, even though such a matrix is already declared in Frame 1 of this lesson, the author will be bypassing that frame so another declaration must be provided here, e.g.:

```
C:SET MATRIX(X,100) C:FETCH X
```

Now to continue the example beginning at some frame number past the display section.

```
FRAME 10.00 (D) LABEL=INIT
IF 4,A  C:FINISHED
ELSE
```

Beginning at this point, the lesson logic is now written to perform any initializations desired.

Recall again that the data declarations in the display section (Frames 5 through 9) could be bypassed if the author chose to "INITIALIZE ONLY." Therefore, the data ought to be declared again. If there was any objection to declaring the data in both sections (which is completely satisfactory although perhaps not the most elegant), then all the data declarations could be made in Group 4 of Frame 4 (Label=PASSWORD), ahead of the "C B:INIT" statement. That would declare all the necessary data spaces for either or both of the following sections.

Note that the test being made in Frame 10, before initialization proceeds, is to terminate execution in the event that the author requested choice "A. DISPLAY ONLY", designated by the characters "4,A" in the test.

After the initialization has been completed, the final statements PUT the data into the Common Data Space and the training sequence is ready for the first team.

Choosing The Proper ESCAPE Parameter Value

The ESCAPE** feature is designed to log off any user who exceeds a specified number of computer processing seconds without an intervening input at the terminal. It will normally be rare that the author will ever be concerned with this command. Yet it is there to protect against the kind of infinite loops that improperly designed lessons (especially team lessons) sometimes originate.

If the ESCAPE** value has been exceeded, the user will be adequately informed at the terminal before being logged out. It will then be the responsibility of the author to determine whether the lesson contains legitimate processing demands which caused the condition to occur or whether the lesson is not functioning properly. This can be determined by using the various debugging aids in PLANIT, such as "TRACE ALL" and "BREAK."

If it has been determined that the lesson is executing properly, then a statement will need to be added to increase the value of the ESCAPE** parameter. The format for this is completely analagous to "WAIT". The following CALC statement:

```
ESCAPE(n)
```

where "n" names a value for processing seconds, increases (or decreases) the limit for the next single interval of time beginning with the next terminal input. The corresponding statement:

```
SET ESCAPE(n)
```

changes that limit for all future terminal response intervals in the current lesson, or until another ESCAPE** command is executed.

If the ESCAPE** limit has halted lesson execution, one should not immediately assume insufficient execution time unless the lesson logic is known to involve long sequences of computation. Team lessons in particular seem to involve complex monitoring and simulation iterations where simple mistakes can inject the lesson into an infinite loop.

APRENDIX C

DEMONSTRATION TRUCK RENTAL LESSON

```
1FRAME 1.00 (Q.)
2ARE YOU THE FIRST OF THE PLAYERS TO SIGN ON?
3/POSNEG/
4 C:SET NUM=4  C:SET UPDATE=3 C:SET MULT=20
4  C:SET MATRIX(COM,13+NUM,2)  C:ROUND 0 C:TRACE OFF
4  C:CHANGE ABSOLUTE TO ABS  C:CHANGE TRUNCATE TO TR
4+ C:PUT COM F:OK, INITIALIZED FOR A NEW GAME.
1FRAME 1.50 (D)
2C:FETCH COM
2IF PROD COM(I,1) FOR(I=13,NUM+12) NQ 0
2F:SORRY, ALL COMPANIES ARE TAKEN. TRY LATER. C:FINISHED
1FRAME 2.00 (Q)
2DO YOU WANT TO SEE THE RULES FOR THE GAME?
3/POSNEG/
4- B:5
1FRAME 3.00 (Q)
2YOU HAVE A TRUCK RENTAL COMPANY WHICH SERVES FOUR CITIES, $
2INCLUDING LOS ANGELES (LA), SAN FRANCISCO (SF), PORTLAND (PO) $
2AND SEATTLE (SEA).  YOU START THE GAME WITH YOUR TRUCKS $
2DISTRIBUTED AS FOLLOWS:@        SEA - 20@      PO  - 15
2      SF  - 35@        LA  - 50
2@YOU CAN SEND THE TRUCKS AT ANY TIME TO ANY OF THE CITIES BY $
2TYPING ANY TWO OF THE ABBREVIATED CITY NAMES (E.G. LA SEA $
2MEANS TO SEND THE TRUCK FROM LA TO SEA).  HOWEVER, THERE WILL $
2BE A LIMITED NUMBER OF FULL-PAYING CUSTOMERS READY TO GO $
2BETWEEN ANY TWO POINTS.  THERE WILL ALSO BE SOME WHO WILL BE $
2WILLING TO RENT AT A DISCOUNT (50% CUT IN PROFIT) AND AN $
2UNLIMITED NUMBER WHO WOULD GO FREE (YOU LOSE 25% OF NORMAL $
2PROFIT).  YOUR FULL-FARE PROFITS ARE AS FOLLOWS:
2      LA SF  - $40@      SF PO  - $60@      PO SEA - $20
2DISCOUNT PROFITS ARE HALF OF THE ABOVE, FREE LOADERS COST $
2YOU 25% OF THE ABOVE, AND PROFITS ARE ACCUMULATIVE OVER THE $
2VARIOUS LEGS (E.G. LA SEA - $120).
```

```
1FRAME 4.00 (Q)
2YOU CAN SEE WHERE YOUR TRUCKS ARE CURRENTLY LOCATED BY $
2TYPING THE WORD, 'TRUCKS'.  YOUR CURRENT ACCUMULATED EARNINGS $
2CAN BE SEEN BY TYPING THE WORDS, 'NET PROFIT' (OR JUST 'NET').  $
2YOU WILL SEE THE EARNINGS OF THE OTHER COMPANIES, TOO.  $
2THUS, THERE ARE THREE ACCEPTABLE INPUTS:@      1) TRUCKS
2     2) NET@      3) CITY1 CITY2 (E.G. LA SF)
2@THE THIRD INPUT FORMAT WILL PRODUCE A REPORT OF THE NUMBER $
2OF TRUCKS AVAILABLE TO BE SENT AND THE NUMBER OF PAYING $
2CUSTOMERS TO GO (BOTH FULL-FARE AND DISCOUNT), THEN ASK HOW $
2MANY TRUCKS ARE TO BE SENT.  CUSTOMERS WILL BE ASSIGNED TO $
2TRUCKS AT THE BEST INCOME RATE.
2@THERE WILL BE SEVERAL IDENTICAL RENTAL COMPANIES, NAMED $
2RENT1, RENT2, RENT3, ETC.  YOU CAN CHOOSE TO BE ANY ONE OF $
2THEM.  ALSO, YOU CAN DIAL MESSAGES TO ANY OTHER COMPANY $
2BY NAME (USING CALC).  MAYBE YOU CAN STRIKE A BARGAIN OF $
2SOME KIND WITH ANOTHER COMPANY THROUGH AN EXCHANGE OF $
2MESSAGES.  DON'T WAIT TOO LONG THOUGH TO DECIDE HOW MANY $
2TRUCKS TO SEND AFTER YOU HAVE CHOSEN A ROUTE BECAUSE ANOTHER $
2COMPANY MIGHT GRAB THE PAYING CUSTOMERS FIRST.
2@YOUR GOAL IS TO EARN AS MUCH FROM RENTING THE TRUCKS AS $
2POSSIBLE.  CUSTOMERS WILL GENERALLY BE MORE READILY AVAILABLE $
2AT THE LARGER CITIES.  YOU MUST TRY TO HAVE TRUCKS WHERE THE $
2PAYING CUSTOMERS ARE.  HERE GOES.
1FRAME 5.00 (P)
2F:@WHICH COMPANY DO YOU WANT TO BE?@
2C:I=13 C:FETCH COM
2L1: IF COM(I,1) EQ 0 C:PRINT '      ';I-12;') - RENT';I-12
2IF I LS NUM+12 C:I=I+1 B:L1
1FRAME 6,00 (Q)
31 NUM/2+.5 WITHIN(NUM/2-.5)
4-. R:CHOOSE A NUMBER.
1FRAME 7.00 (D)
2C:FETCH COM
2IF COM(RESPONSE+12,1) NQ 0,
2F:THAT COMPANY IS TAKEN, CHOOSE ANOTHER. B:6
2ELSE C:COM(RESPONSE+12,1)=TERMINAL C:PUT COM
2C:SET PROFIT=0 C:SET MATRIX(TRUCKS,4)
2C:SET MATRIX(CHK,NUM) C:SET MATRIX(QUE,4,4)
2C:SET MATRIX(EARN,12) C:SET MATRIX(PRTY,12)
2C:TRUCKS(1)=ARRAY(20,15,35,50) C:SET CO=TR(RESPONSE)
2C:EARN(1)=ARRAY(20,80,120,20,60,100,80,60,40,120,100,40)
2C:PRTY(1)=ARRAY(.6,.2,.3,.5,.4,.3,.5,.5,.8,.6,.6,1)
2C:SET X=0 C:SET Y=0 C:SET Z=0
2C:FUNCTION ENR(I)=SUM QUE(I,J) FOR(J=1,3)
```

```
1FRAME 8.00 (Q)
2TRANSACTION?
30 TEXT ON
3A SEA PO
3B SEA SF
3C SEA LA
3D PO SEA
3E PO SF
3F PO LA
3G SF SEA
3H SF PO
3I SF LA
3J LA SEA
3K LA PO
3L LA SF
3M TRU
3N NET
3P SAN
3Q LOS
4A C:X=1 C:Y=1 C:Z=2
4B C:X=2 C:Y=1 C:Z=3
4C C:X=3 C:Y=1 C:Z=4
4D C:X=4 C:Y=2 C:Z=1
4E C:X=5 C:Y=2 C:Z=3
4F C:X=6 C:Y=2 C:Z=4
4G C:X=7 C:Y=3 C:Z=1
4H C:X=8 C:Y=3 C:Z=2
4I C:X=9 C:Y=3 C:Z=4
4J C:X=10 C:Y=4 C:Z=1
4K C:X=11 C:Y=4 C:Z=2
4L C:X=12 C:Y=4 C:Z=3
4M B:AVBL
4N B:EARNINGS
4P R:IF YOU MEANT SAN FRANCISCO, TYPE 'SF'.
4Q R:IF YOU MEANT LOS ANGELES, TYPE 'LA'.
4- R:?
1FRAME 9.00 (D)
2IF TRUCKS(Y) EQ 0 F:NO TRUCKS AVAILABLE. B:8
2END C:FETCH COM
2IF TIME^COM(NUM+13,1) OR UPDATE C:COM(NUM+13,1)=TIME
2C:COM(I,1)=COM(I,1)+TR((PRTY(I)MULT-COM(I,1))RANDOM)FOR(I=1,12)
2C:COM(I,2)=COM(I,2)+TR((2PRTY(I)MULT-COM(I,1))RANDOM)FOR(I=1,12)
2C:PUT COM
2END,C:TRUCKS(I)=TRUCKS(I)+QUE(I,1) FOR(I=1,4)
2C:QUE(I,J)=QUE(I,J+1) FOR(J=1,3 I=1,4)
2C:PRINT TRUCKS(Y);' TRUCKS AVAILABLE. ';COM(X,1)$
2C:PRINT ' FULL-FARE AND ';COM(X,2);' DISCOUNT '$
2F:CUSTOMERS ARE READY TO GO. HOW MANY TRUCKS DO $
2F:YOU WANT TO SEND?
```

```
1FRAME 10.00 (G)
31   ABS(TR(RESPONSE))
4- F:CAN'T ACCEPT THAT NUMBER. B:8
1FRAME 11.00 (D)
2IF RESPONSE GR TRUCKS(Y)
2F:DON'T HAVE THAT MANY TRUCKS AVAILABLE. B:8
2END C:TRUCKS(Y)=TRUCKS(Y)-RESPONSE
2C:QUE(Z,J)=QUE(Z,J)+RESPONSE FOR(J=ABS(Y-Z))
2C:SET N1=RESPONSE C:FETCH COM
2IF N1 GR COM(X,1) C:N1=COM(X,1)
2END C:SET N2=RESPONSE-N1
2IF N2 GR COM(X,2) C:N2=COM(X,2)
2END C:SET N3=RESPONSE-N1-N2.
2C:PROFIT=PROFIT+TR((N1+.5N2-.25N3)EARN(X)+.1)
2C:COM(X,1)=COM(X,1)-N1 C:COM(X,2)=COM(X,2)-N2
2C:COM(CO+12,2)=PROFIT C:PUT COM B:8
1FRAME 12.00 (D) LABEL=AVBL
2F:@CITY  AVBL.  ENROUTE@----  -----  --------
2C:ALIGN 'SEA',2;TRUCKS(1),10;ENR(1),18
2C:ALIGN 'PO',2;TRUCKS(2),10;ENR(2),18
2C:ALIGN 'SF',2;TRUCKS(3),10;ENR(3),18
2C:ALIGN 'LA',2;TRUCKS(4),10;ENR(4),18
2C:TRUCKS(I)=TRUCKS(I)+QUE(I,1) FOR(I=1,4)
2C:QUE(I,J)=QUE(I,J+1) FOR(J=1,3 I=1,4) B:8
1FRAME 13.00 (D) LABEL=EARNINGS
2F:@COMPANY  EARNINGS@-------  --------- C:FETCH COM
2C:ALIGN 'RENT',2;I,7;'$',10;COM(I+12,2),18 FOR(I=1,NUM)
2IF CHK(I) EQ COM(I+12,1) FOR(I=1,NUM) B:8 .
2END C:CHK(I)=COM(I+12,1) FOR(I=1,NUM)
2F:@COMPANY  TERMINAL@-------  --------
2C:ALIGN 'RENT',2;I,7;CHK(I),14 FOR(I=1,NUM)
2C:RENT1=COM(13,1)
2IF NUM GR 1 C:RENT2=COM(14,1)
2IF NUM GR 2 C:RENT3=COM(15,1)
2IF NUM GR 3 C:RENT4=COM(16,1)
2IF NUM GR 4 C:RENT5=COM(17,1)
2IF NUM GR 5 C:RENT6=COM(18,1)
2IF NUM GR 6 C:RENT7=COM(19,1)
2IF NUM GR 7 C:RENT8=COM(20,1)
2END. B:8
SSSS
```

APPENDIX D

SAMPLE RUN OF DEMONSTRATION LESSON

```
ARE YOU THE FIRST OF THE PLAYERS TO SIGN ON?

*YES
OK, INITIALIZED FOR A NEW GAME.

DO YOU WANT TO SEE THE RULES FOR THE GAME?

*YES

YOU HAVE A TRUCK RENTAL COMPANY WHICH SERVES FOUR CITIES, INCLUDING LOS
ANGELES (LA), SAN FRANCISCO (SF), PORTLAND (PO) AND SEATTLE (SEA).  ALL
START WITH THE SAME NUMBER OF TRUCKS DISTRIBUTED AS FOLLOWS:
     SEA - 20
     PO  - 15
     SF  - 35
     LA  - 50

YOU CAN SEND THE TRUCKS AT ANY TIME TO ANY OF THE CITIES BY TYPING ANY
TWO OF THE ABBREVIATED CITY NAMES (E.G. LA SEA MEANS TO SEND THE TRUCK
FROM LA TO SEA).  HOWEVER, THERE WILL BE A LIMITED NUMBER OF
FULL-PAYING CUSTOMERS READY TO GO BETWEEN ANY TWO POINTS.  THERE WILL
ALSO BE SOME WHO WILL BE WILLING TO RENT AT A DISCOUNT (50% CUT IN
PROFIT) AND AN UNLIMITED NUMBER WHO WOULD GO FREE (YOU LOSE 25% OF
NORMAL PROFIT).  YOUR FULL-FARE PROFITS ARE AS FOLLOWS:
     LA SF  - $40
     SF PO  - $60
     PO SEA - $20
DISCOUNT PROFITS ARE HALF OF THE ABOVE; FREE LOADERS COST YOU 25% OF
THE ABOVE, AND PROFITS ARE ACCUMULATIVE OVER THE VARIOUS LEGS (E.G. LA
SEA - $120).

YOU CAN SEE WHERE YOUR TRUCKS ARE CURRENTLY LOCATED BY TYPING THE WORD
'TRUCKS'.  YOUR CURRENT ACCUMULATED EARNINGS CAN BE SEEN BY TYPING THE
WORDS, 'NET PROFIT' (OR JUST 'NET').  YOU WILL SEE THE EARNINGS OF THE
OTHER COMPANIES, TOO.  THUS, THERE ARE THREE ACCEPTABLE INPUTS:
     1) TRUCKS
     2) NET
     3) CITY1 CITY2 (E.G. LA SF)

THE THIRD INPUT FORMAT WILL PRODUCE A REPORT OF THE NUMBER OF TRUCKS
AVAILABLE TO BE SENT AND THE NUMBER OF PAYING CUSTOMERS TO GO (BOTH
FULL-FARE AND DISCOUNT), THEN ASK HOW MANY TRUCKS ARE TO BE SENT.
CUSTOMERS WILL BE ASSIGNED TO TRUCKS AT THE BEST INCOME RATE.

THERE WILL BE SEVERAL IDENTICAL RENTAL COMPANIES, NAMED RENT1, RENT2,
RENT3, ETC.  YOU CAN CHOOSE TO BE ANY ONE OF THEM.  ALSO, YOU CAN DIAL
MESSAGES TO ANY OTHER COMPANY BY NAME (USING CALC).  MAYBE YOU CAN
STRIKE A BARGAIN OF SOME KIND WITH ANOTHER COMPANY THROUGH AN EXCHANGE
OF MESSAGES.  DON'T WAIT TOO LONG THOUGH TO DECIDE HOW MANY TRUCKS TO
SEND AFTER YOU HAVE CHOSEN A ROUTE BECAUSE ANOTHER COMPANY MIGHT GRAB
THE PAYING CUSTOMERS FIRST.  HERE GOES.
```

```
WHICH COMPANY DO YOU WANT TO BE?

     1) - RENT1
     2) - RENT2
     3) - RENT3
     4) - RENT4

*3

TRANSACTION?

*NET-PROFITS

COMPANY  EARNINGS
-------  --------
 RENT1   $       0
 RENT2   $       0
 RENT3   $       0
 RENT4   $       0

COMPANY  TERMINAL
-------  --------
 RENT1      0
 RENT2      0
 RENT3      1
 RENT4      0

TRANSACTION?

*TRUCKS

CITY  AVBL.  ENROUTE
----  -----  -------
 SEA   20       0
 PO    15       0
 SF    35       0
 LA    50       0

TRANSACTION?

*NET

COMPANY  EARNINGS
-------  --------
 RENT1   $       0
 RENT2   $       0
 RENT3   $       0
 RENT4   $       0
```

```
TRANSACTION?

*LOS SEA
IF YOU MEANT LOS ANGELES, TYPE 'LA'.

*LA SEA
50 TRUCKS AVAILABLE. 11 FULL-FARE AND 5 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*11

TRANSACTION?

*LA PR
?

*LA PO
39 TRUCKS AVAILABLE. 2 FULL-FARE AND 2 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*2

TRANSACTION?

*TRUCKS

CITY  AVBL.  ENROUTE
----  -----  -------
 SEA   20       11
 PO    15        2
 SF    35        0
 LA    37        0

TRANSACTION?

*SF SEA
35 TRUCKS AVAILABLE. 3 FULL-FARE AND 8 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*3

TRANSACTION?

*TRUCKS

CITY  AVBL.  ENROUTE
----  -----  -------
 SEA   31        3
 PO    17        0
 SF    32        0
 LA    37        0
```

```
TRANSACTION?

*SEA SF
34 TRUCKS AVAILABLE. 3 FULL-FARE AND 6 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*3

TRANSACTION?

*SEA LA
31 TRUCKS AVAILABLE. 1 FULL-FARE AND 12 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*1

TRANSACTION?

*SEA LA
30 TRUCKS AVAILABLE. 0 FULL-FARE AND 12 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*0

TRANSACTION?

*PO LA
17 TRUCKS AVAILABLE. 3 FULL-FARE AND 10 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*3

TRANSACTION?

*SEA LA
30 TRUCKS AVAILABLE. 0 FULL-FARE AND 12 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*12

TRANSACTION?

*PO LA
14 TRUCKS AVAILABLE. 4 FULL-FARE AND 15 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*4
```

```
TRANSACTION?

*TRUCKS

CITY  AVBL.  ENROUTE
----  -----  -------
 SEA   18       0
 PO    10       0
 SF    35       0
 LA    41      16

TRANSACTION?

*-18+10+35+41+16
120

*LA SEA
57 TRUCKS AVAILABLE. 10 FULL-FARE AND 13 DISCOUNT CUSTOMERS ARE READY
TO GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*10

TRANSACTION?

*SF SEA
35 TRUCKS AVAILABLE. 5 FULL-FARE AND 28 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*33

TRANSACTION?

*LA SF
47 TRUCKS AVAILABLE. 18 FULL-FARE AND 48 DISCOUNT CUSTOMERS ARE READY
TO GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*18

TRANSACTION?

*NET

COMPANY  EARNINGS
-------  --------
 RENT1   $      0
 RENT2   $      0
 RENT3   $   6980
 RENT4   $      0
```

```
TRANSACTION?

*TRUCKS

CITY  AVBL.  ENROUTE
----  -----  -------
 SEA   18      43
 PQ    10       0
 SF     2      18
 LA    29       0

TRANSACTION?

*SEA LA
61 TRUCKS AVAILABLE. 2 FULL-FARE AND 16 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*18

TRANSACTION?

*SEA SF
43 TRUCKS AVAILABLE. 3 FULL-FARE AND 10 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*13

TRANSACTION?

*SF LA
20 TRUCKS AVAILABLE. 15 FULL-FARE AND 54 DISCOUNT CUSTOMERS ARE READY
TO GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*20

TRANSACTION?

*TRUCKS

CITY  AVBL.  ENROUTE
----  -----  -------
 SEA   30       0
 PQ    10       0
 SF     0      13
 LA    29      38
```

```
TRANSACTION?

*LA SF
67 TRUCKS AVAILABLE. 7 FULL-FARE AND 75 DISCOUNT CUSTOMERS ARE READY TO
GO.  HOW MANY TRUCKS DO YOU WANT TO SEND?

*17

TRANSACTION?

*TRUCKS

CITY  AVBL.  ENROUTE
----  -----  -------
 SEA   80       0
 PO    10       0
 SF    13      17
 LA    50       0

TRANSACTION?

*NET

COMPANY  EARNINGS
-------  --------
 RENT1    $      0
 RENT2    $      0
 RENT3    $  10000
 RENT4    $      0

TRANSACTION?
```