T.A.C.L.

The Adventure Construction Language © 1988,89 Alternate Realities

Manual, Tutorial, and Language Reference

Bring Your Adventures To Life

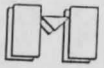
JUMPTABLE

Alternate Realities and Micro Momentum, Inc.

T.A.C.L. was conceived and written by Alternate Realities. *T.A.C.L.* is produced by Micro Momentum, Inc.

We are always looking for ways to make our products better. If you have a suggestion, please write us a letter at the address below. Remember to send in your registration card so we can notify you of upgrades to *T.A.C.L.*

Please send us your adventures or any other Amiga products you have written if you would like us to consider them for publication.



Micro Momentum, Inc.
P.O. Box 372
Washington Depot, CT 06794

Notice of Trademarks

HOW TO USE THIS MANUAL

This manual is organized into six main sections. Section 0 contains the Introduction, Copyright Notices, Special Thanks, and Getting Started.

Section I describes MADV, the adventure maker, covering in detail the basics of how to write your adventures, how to display graphics from your adventure, and how to find errors in your source code.

The Vector Graphics EDitor, VGED, is covered in Section II, with instructions on how to use VGED to draw and add graphics to your adventure. Section II also includes a short description of how the graphics are handled, and how to use standard Amiga IFF graphics in your adventure.

Section III contains a detailed description of PADV, the adventure player. This section explains all options available while playing a game, and how to use them. Also, the last part of Section III covers the debugging features of PADV, which will become invaluable while developing your own adventure.

Section IV contains a formal description of EVERY command in the *T.A.C.L.* Language, as well as reference file structures, and tips and tricks for better adventure writing.

The final section is composed of appendices that include MADV's Error Codes, PADV's Run-Time Errors that may be encountered, and a short reference for the included text editor. Finally, the manual also contains a comprehensive glossary.

SPECIAL THANKS

From Kevin Kelm: Thanks to Rhett Rodewald for finding more bugs than I care to admit ever existed. Thanks to his continual "constructive criticism" several accidental "features" have been corrected. If you find any bugs remaining, it is entirely his fault. Also, most of the sample adventures and the manual were written by Rhett. His ability to create such complex character interactions, environmental control, and a good storyline still amazes me. He has done things with *T.A.C.L.* that I didn't even know were possible.

SECTION 0

T.A.C.L. The Adventure Construction Language

Manual, Tutorial, and Language Reference.

INTRODUCTION

Welcome to "The Adventure Construction Language" development system. *T.A.C.L.* is a programming language specially designed to allow almost anyone, with or without programming experience, to construct their own text/graphic adventure games. With sufficient time and energy, you can create games as complex as the "Zork" series by Infocom. Graphic scenes may be imported or drawn for inclusion in your adventures, allowing you to produce adventures similar to "The Pawn" by Rainbird. And *T.A.C.L.* provides even more flexibility by allowing pictures for each object and specific events to be included, in addition to scenes for each room or location.

T.A.C.L. is also machine-independent which means that, after writing an adventure on the Amiga, you can play that same adventure on any other computer that *T.A.C.L.* is implemented on. (IBM and Macintosh versions are currently under development.)

COPYRIGHT NOTICES

From Kevin Kelm, Rhett Rodewald, and Alternate Realities: A special thanks to Leon Frenkel and Avant Garde Software for the amazing Benchmark Modula-2 development system, without which we would still be working on this project. Leon is also to be commended for his continually helpful and friendly customer support.

Finally, heartfelt thanks to anyone and everyone who has had to put up with either of us and our eccentricities during the development of this system, especially our families and friends.

GETTING STARTED

First we'll show you how to get the Play ADVenture (PADV) section of the *T.A.C.L.* system up and running so you can begin playing some of the included sample adventures immediately.

The easiest way to play an adventure is to click on the icon of the Game you want to play from Workbench. To do this, turn on your computer, and put the "*T.A.C.L.* Play Disk" in the internal disk drive when the computer asks for Workbench. When Workbench is ready, double-click on the "*T.A.C.L.* Disk" icon to get the game icons. After the window opens and the icons fill in, double-click on the "Game Part I" icon to start the game. The game will load PADV (the adventure player) and then load the game you have selected. Note: the same can be accomplished by typing "PADV <GameName>" from the CLI. (Remember to substitute the actual name of the game that you want to play for <GameName>.)

This page for doodles.

SECTION 1

MADV (Make ADVenture) Adventure Compiler

MADV is the adventure-building section of *T.A.C.L.*, and is to be used in conjunction with the text editor and VGED to create your adventure. MADV is similar to any other language compiler (like C or Modula-2), so those of you who have worked with a compiler before may wish to skip the first part of this section which describes the Edit/Compile/Run Cycle in detail. (If you didn't understand this last sentence, then be sure to read the first part of this section carefully.) Above all, remember: the most important thing when programming is PATIENCE!!! Relax and enjoy yourself while writing your adventure. Don't try to write within a deadline; it is not nearly as enjoyable, and you may end up skimping on details.

THE EDIT/COMPILE/RUN CYCLE

Now what? Well, we won't actually start writing an adventure until the next part, so relax. We will begin, however, by recommending a familiarity with the Command Line Interface (CLI). If you have never used the CLI, we recommend *Amiga For Beginners* (Abacus, 1988) or The *AmigaDOS Manual* (Bantam, 1986).

The three basic concepts needed to write an adventure are separate and distinct in and of themselves, but an understanding of how they work together is vital.

First is the editor. *T.A.C.L.* doesn't care if you use AmigaDOS's ED or your own favorite editor. If you wish to use ED, spend some time reading Appendix D in the back of this manual, or look to the *AmigaDOS Manual* for a complete description. There are also several excellent public domain and shareware editors available. Check with your local computer club or order some disks from the Fred Fish collection.

Use the editor to create your source code files. If you are familiar with AmigaBASIC, then you are used to an edit window and a display window. Think of your editor as your edit window, and PADV as the display window, with an extra step through MADV. After you have created the various source files (described later in this section) that compose your adventure, and have saved them to disk, exit your editor (or if you have enough memory, switch to another CLI to compile).

The compiler, known as MADV, must be run from the CLI. To do so, simply type "MADV AdvName" to compile your adventure. (Read on for other MADV options.) This will run MADV and tell it the DOS name of your adventure. MADV will read all your files and attempt to compile your adventure. If successful, it will create a .GAM and a .WRD file, as well as an .XRF file if you so choose. If it encounters errors, it will display them on the screen as it encounters them, and list the errors in a file named "AdvName.ERR". Should you encounter any errors, you must go back and fix the problems with your editor before you can do anything else. Look at each error, which file it was in, which line it occurred on, and what the error was. Fix your source files, and recompile. Continue until MADV finds no errors and your adventure compiles successfully.

When it compiles correctly, run PADV, load your adventure (the -E debug option is recommended), and try to play your adventure. If something isn't working the way you want it to, go back to your editor, look to see what you are currently telling the game to do, then make the appropriate changes to make it work as you want it to.

Keep this general model of developmental flow in mind as you work on your adventure. Compile often so that you know where your new sections are, and you will be able to locate any errors in them more easily.

RUNNING MADV

MADV must be run from the CLI, and needs a reasonably large stack size set. (We recommend a stack size of 20,000 bytes. See the PADV Section for details on setting up your stack.)

To run MADV, type "MADV", a space, the name of your adventure (e.g., "Spy.ADV", or just "Spy"), then follow that with any cross-reference options you want. Note that the -x must be used before any other option, as the rest of the options are modifiers for the -x option. The options are as follows:

```
-x   Generate a cross reference with everything, except
the "f" option.
```


Adding any other options will disable all other options not specifically selected. The options are:

- v List all variables and initial values.
- r List all rooms in the adventure.
- o List all objects in the adventure.
- c List all CODE blocks.
- g List all Graphics files included.
- f Add form-feeds after each section of the cross-reference listing.

Here's the concise format:

```
MADV <Adventure Name> [-x[v][r][o][c][g][f]]
```

where each option in square brackets is optional. Except for the "x", the options do not have to be in any particular order.

MADV always tries to generate the two main files that are needed to play the adventure. These are the .GAM and the .WRD files. The .GAM file is the main file, while the .WRD file contains encoded text. Also, if you want to see the images in an adventure, all the included graphic files are needed.

When the -x option is selected, MADV will generate one additional file with an .XRF extension. This is a helpful cross-reference listing for all the requested information about your adventure.

WHAT ARE ALL THESE FILES FOR?

T.A.C.L. uses a system of files to keep all source code nicely divided and to keep individual files from becoming too large. There are six classes of SOURCE files. The main file, known as the ADVENTURE or .ADV file, is the most important and the easiest to write. This file basically provides the system with key information like the names of the other files, the password, variable names, and which room the player will begin the adventure in. There must be exactly one ADVENTURE file per adventure. However, all other file classes may have any number of files per adventure. The other five file classes are: IMAGE, ROOM, OBJECT, SUBROUTINE, and VOCABULARY. There are file templates on the *T.A.C.L.* disk in directory "blank".

IMAGE files are simply the file names of any pictures used in the adventure. For VGED files, the password for that file must also be included on the same line, following the filename. This allows you to distribute your adventure and, at the same time, keep others from using

your graphics in their adventures, or even viewing them without permission. Of coarse, if you do want other *T.A.C.L. users to be able to view your graphics, you may use the standard password: "PUBLIC".*

The ROOM files contain all the code describing the rooms in the adventure. Each file may contain multiple rooms. There may also be any number of files, and you may name them whatever you wish (although it is often helpful to use a "Room" or ".ROOM" postfix to keep your files straight).

The OBJECT files are similar to the room files, only they define each object used in the adventure. Like the room files, it is a good idea to use an "Objects" or ".OBJ" postfix for these files.

The SUBROUTINE files are special sections of code that perform specific functions, and can be called from anywhere else in the adventure. This makes it easy to perform a particular function repeatedly.

The VOCABULARY files define all nonstandard phrases, words, and special functions. This will be described in more detail later.

To play your adventure you will need only the graphic files, and the .GAM and .WRD files. You will notice that these are specially encoded and cannot be read by others examining the adventure files.

Now that we have a basic understanding of what to edit and compile, and what the different file types are, we are ready to begin building our first adventure. So be brave, and continue reading.

MAKING YOUR FIRST ADVENTURE

Let's see... trick cigarettes, concealed automatic handgun, wiretaps, bugs, secret plans, coding/decoding telephone, invisible ink for writing adventures...

We are now ready to begin building our first adventure. Take this at your own pace, and back up if you get lost. The following are "step-by-step" and "file-by-file" instructions for building a simple, but playable sample adventure. This adventure will take a while, so plan to spend several hours creating it.

THE ADVENTURE FILE

This is the best place to start your adventure. Using your favorite editor, create a file called "Spy.ADV". Remember that your own adventures must end with the same ".ADV".

The first line of this file should read, "ADVENTURE", followed by the name of the adventure, like "Short Spy Adventure" or anything else you would like to call your game (except for the word, ADVENTURE, which is used by MADV). Feel free to use the whole initial line for the

name of your adventure. However, pick something that makes sense, as this name will be displayed at the beginning of the game.

On the next line (or a few lines down if you wish), must appear "PASSWORD", followed by a SINGLE WORD that you have chosen as this adventure's password. Do not pick easily identified, or topical passwords. Something completely off-topic is usually best, such as the name of your favorite hockey team.

Now is our chance to define any and all variables that we want to use in the adventure. We can always come back later and add more, but here are a few to give us a good start. The variable block is optional, so if you do not need any variables in your adventure, just skip this block entirely. Start the variable block with a line containing the keyword "VAR", then on the following lines we will list all our variables and their starting values. If you leave off the starting value, MADV will assume it to be 0.

Let's define seven variables right now, and call them FLOOR, PLANSFLOOR, MEMOFLOOR, DISKFLOOR, AGENTFLOOR, TIME, and BULLETS. For now, we will not bother with any of the -FLOOR variables. However, we do want the player to start with 6 bullets in his gun, the time to start at 0, and we want the first floor to be the ground floor. To do this, we make the next five lines "Floor 1", "PlansFloor", "MemoFloor", "DiskFloor", "AgentFloor", "Time 0", and "Bullets 6", respectively. Note the "1" after the Floor variable, the "0" after the Time variable, and the "6" after the Bullets variable. End the variable block by putting "ENDVAR" on the next line, and maybe add an extra blank line before the next block.

There are also several variables that are defined by the system. These are SCORE, ITEMS, and MOVES. In addition, MADV creates a variable for the position of every object in the adventure, the name of which is derived by adding "POS" to the end of the object's name. For example, an object called Agent would have a corresponding variable AgentPOS, which will always contain the number of the room the Agent is in. The SCORE variable is automatically displayed by the system when the player types the "SCORE" command in PADV, unless the adventure redefines the "SCORE" command. Add or subtract from this variable to keep track of the player's score. ITEMS and MOVES are automatically updated by the system to reflect the number of items the player currently has in his inventory, and the number of moves he has made since the start of the game.

All variables in PADV are integers, and must be between -32768 and 32767, inclusive. The reasons for these numbers are hardware specific, so don't worry about them, just make sure not to exceed them.

The IMAGE block is next but, like the variable block, it is optional and should be skipped if you do not plan to add any graphics to your adventure. Start the IMAGE block with a line stating "IMAGE", followed by one or more lines with the file names of your images, one or more spaces and, if it is a VGED file, the password for that image file. (For IFF and ILBM files, skip the password.) End the block with an "ENDIMAGE" command. Skip this block for now; you can return later to put it in should you want to draw any images as discussed in Section II.

Next is the ROOM block which, like the image block, defines the files that contain the room descriptions. "ROOM" applies to all locations within the adventure, not just indoor locations. Whether a given "ROOM" is indoors or out depends on the description you provide to the player. Start the ROOM block with "ROOM", follow it with one Room file name per line, and end it with an "ENDROOM" line. For this adventure, define the room file "Spy.ROOM".

The OBJECT block follows the ROOM block, and is of the same format except it is optional, begins with "OBJECT", ends with "ENDOBJECT", and defines the filenames for object code instead of room code. Define the object file "Spy.OBJ" for this adventure.

The SUBROUTINE block is also optional, and basically the same as the OBJECT block, but begin it with "SUB" and end it with "ENDSUB". Define the subroutine file "Spy.SUB" for this adventure.

The VOCABULARY block is last, and also optional. Begin the block with "VOCAB", and end it with an "ENDVOCAB" statement. For this adventure, define the vocabulary file "Spy.VOC".

The next line is an "INITROOM" statement, followed by a space and the name of the room you will begin the adventurer in. For this adventure, the starting room will be "Embassy".

Finally, end this file with an "ENDADVENTURE" statement, and close the file. Check your file against the entire file listing below.

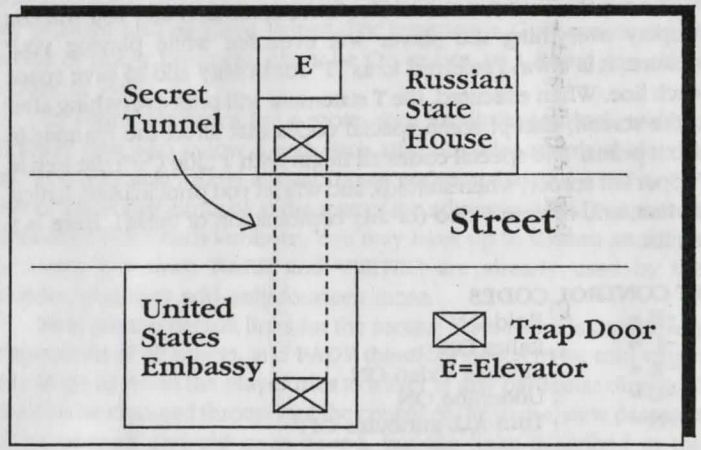
Here is the whole file we have just created, as it should look from your editor (blank lines are omitted to save space):

```
* ———————— FILE SPY.ADV ————————
ADVENTURE Short Spy Adventure
PASSWORD Capitals
VAR
Floor    1
PlansFloor
MemoFloor
DiskFloor
AgentFloor
Time    0
Bullets     6
ENDVAR
ROOM
Spy.ROOM
ENDROOM
OBJECT
Spy.OBJ
ENDOBJECT
SUB
Spy.SUB
ENDSUB
VOCAB
Spy.VOC
ENDVOCAB
INITROOM Embassy
ENDADVENTURE
* ———————— END OF FILE ————————
```


THE ROOM FILE

Although this adventure will have only one room file, remember that you may have any number of files for rooms, each containing any number of rooms, with the obvious minimum of one room per file. To keep our adventure simple, we will use only one file with seven rooms in it. First, we need to draw our map. Ours will appear to have fifteen rooms, but we will only define seven – one special room will act as ten different rooms, and one will not normally be used.

Our map will look like this:



The Embassy will be the first room, the Secret Tunnel will be the next two rooms, the street and elevator will be the fourth and fifth rooms, the State House will be our special sixth room, and the seventh room will be our spare room.

The mission will be to find and steal back the stolen plans to the Stealth Bomber. The Russian State House will be ten stories high, and we will randomly hide the document somewhere in it. This will keep the game interesting, even after playing it a few times, as the document will not always be in the same place. We will make the single room appear to be ten rooms (accessed by the elevator) through the use of room links and changing descriptions.

But before we begin with the ROOM file, we need to understand some of the *T.A.C.L.* language statements. We will introduce a few statements in each section, with the remaining statements appearing at the end of this section. For a complete reference, refer to Section IV: The *T.A.C.L.* Language.

First, we introduce the comment line. Use a comment line whenever you want to make a note to yourself about something you have written in the code. People playing your adventure will never see your comments; they exist only in your source files, for your benefit only, and are not even written into the adventure. Make a comment by placing an asterisk ("*") on the line, then typing in the comment. A comment can appear on any line except for the ADVENTURE line, a line with a Text ("T") statement, or a DIRECTIONS statement. Remember, EVERYTHING on that line after the asterisk is ignored, so do not put any other program instructions on that line; they will also be ignored by MADV.

Next is the Text statement. This is the command that you will use to display everything the player will ever see while playing your adventure. It is always referred to as "T" for brevity and to save space on each line. When executed, the T statement will print everything after it to the screen, except some special codes that affect the manner in which it prints. The special codes all begin with a tilde ("~": the key in the upper left corner, when shifted), and will let you print in bold, italics, underline, and reverse video (or any combination of these). Here is a quick list:

TEXT CONTROL CODES

~B =	Bold ON
~I =	Italics ON
~R =	Reverse Video ON
~U =	Underline ON
~N =	Turn ALL attributes OFF

These attributes will remain set until you turn them off with the "~N" option. You should always turn any attributes off before ending a block of text. Use the attributes somewhat conservatively, and alternate often. If they are used constantly, they lose their uniqueness.

The T command also allows you to print the values of your variables. To print the value of a variable, put an "AT" symbol ("@") at the location in the T statement where you want the value to print, and follow it immediately (no spaces) with the name of the variable and then a space (which will not be printed but is used to separate the variable's name from any following text). For example,

```
T ~IYou have @Bullets Bullets in your Gun.~N
```

would display:

You have 6 Bullets in your Gun.

(This whole line will print in italics, and assumes that the player has six bullets.)

Each Room in the Room file contains the following in its code: Header, Attributes (optional), DEFAULT room links (optional), a CODE block, and an "ENDROOM" statement.

The Header starts with the keyword "ROOM", and is followed by a space and the room name. (See above, or Section IV for the rules on making a legal identifier.)

Optionally, each room can have a set of attributes. In *T.A.C.L.*, an attribute is some detail that is specific to a room or object, and it can have only positive and negative values. For instance, a Gun can either be Loaded or not, a Lamp can either be On or not, or a Room can either be Dark or not.

To create attributes for a room, start an attribute block with an "ATTRIB" line, and follow it with each attribute (also standard identifiers). You may also follow any attributes with a space and a "Y" or "N" to set or unset that attribute at the start of the adventure. MADV assumes a default of N for each attribute. You may have up to sixteen attributes per room, but since DARK and VISITED are already used by the compiler, you may add only fourteen more.

Next are the default links for the rooms. Each room is completely independent of all others, and PADV therefore needs to be told which room to go to when the player tries to travel in any particular direction. Links can be changed throughout the course of the game, new passages can be opened and old ones closed, but the links described in the

DEFAULT block are the starting links that will remain in place until they are specifically changed. Start the default block with "DEFAULT", and end it with "ENDDEFAULT". A direction abbreviation, a space, and the room name that the player will reach if he moves in that direction should appear in-between each line. Any unused directions should not be included; PADV will automatically tell the player that he cannot move in any unused directions.

The final block before the "ENDROOM" statement starts with "CODE" and ends with "ENDCODE". Between these lies all the code (T statements, etc.) that describes the setting to the player. This is especially complex for the Russian State House Room. We will resume describing all the commands used here later in this section. You may also refer to Section IV if you have any questions regarding the commands.

```
* ———————— FILE SPY.ROOM ————————
ROOM Embassy     * This is where the player starts
                 * the adventure from.

DEFAULT
N Street
ENDDEFAULT
CODE
IF PLAYER HAS Plans THEN          * Player has won
 T Congratulations Mr. Super Agent!
 T You have saved the day, prevented the
 T compromising of national security, and
 T stopped the Russians cold.
 T                      Thank You!
 WIN
ELSIF Embassy NOT Visited THEN    * Startup Code
 CALL Randomize                   * Setup the game.
 T You are currently in the United States Embassy
 T in Moscow, U.S.S.R.  The Russians have stolen
 T the plans to our new F-19 Stealth Bomber.
 T ~B YOUR MISSION: ~N
 T   Recover the stolen plans from the Russian
 T    State House where they currently are hidden.
 T        Good Luck from all of us!
 T
 T You see a trapdoor in the floor.
ELSIF Trapdoor IS Open THEN
 T You are in the Embassy, a trapdoor in the
 T floor is open, and stairs lead down into a
 T dark passage.
ELSE
 T You are in the United States Embassy.
 T There is a trapdoor here.
ENDIF
```



```
ENDCODE
ENDROOM

ROOM Street
  * Since we don't feel like building the rest
  * of Moscow right now, it is deadly to enter
  * this room; we don't even bother making exits.
CODE
T You walk confidently out the door into
T the streets of Moscow. However, you are a
T spy, not a diplomat, and just notice a large,
T black van speeding by and shooting at you
T before you die.
CALL KillPlayer
ENDCODE
ENDROOM

ROOM SecretTunnelSouth
DEFAULT
 N SecretTunnelNorth
 U Embassy
ENDDEFAULT
CODE
T You are in a narrow dark passage that runs
T to the north and up.  You notice a sign on
T the wall that says,
T          "~BSECRET PASSAGE~N"
T      Authorized Personnel Only
ENDCODE
ENDROOM

ROOM SecretTunnelNorth
DEFAULT
 U StateHouse
 S SecretTunnelSouth
ENDDEFAULT
CODE
T You are in a narrow dark passage that runs
T south behind you.  There is also an opening
T above you.
ENDCODE
ENDROOM

ROOM Elevator
DEFAULT
 S StateHouse
ENDDEFAULT
CODE
T You see a panel with ten buttons labelled,
```



```
T "1" thru "10", beckoning to you to press them.
T An indicator says you are on Floor @Floor .
ENDCODE
ENDROOM

ROOM StateHouse
DEFAULT
 N Elevator
 S Street
 D SecretTunnelNorth
ENDDEFAULT
CODE
IF DiskFloor # 0 THEN    * player doesn't have disk
 PLACEOBJ Disk NoWhere
ENDIF
IF MemoFloor # 0 THEN    * player doesn't have memo
 PLACEOBJ Memo  NoWhere
ENDIF
IF PlansFloor # 0 THEN   * player doesn't have plans
 PLACEOBJ Plans NoWhere
ENDIF
IF Floor = 1 THEN
 T You are on the ground floor of the Russian
 T State House.  There is an elevator to the
 T North, doors to the south, and a hole in the
 T floor leading down.
 DIRECTIONS StateHouse N S D
ELSE
 T You are on Floor @Floor  of the Russian
 T State House.  It looks pretty bare.
 T There is an elevator in the north wall.
 DIRECTIONS StateHouse N
 IF Floor = DiskFloor THEN   * place special objects in the room
  PLACEOBJ Disk StateHouse
 ENDIF
 IF Floor = MemoFloor  THEN
  PLACEOBJ Memo  StateHouse
 ENDIF
 IF Floor = PlansFloor THEN
  PLACEOBJ Plans StateHouse
 ENDIF
ENDIF
ENDCODE
ENDROOM

ROOM NoWhere      * A spare room to store objects
CODE
T If you aren't in debug mode, you can't get here!
ENDCODE
ENDROOM
* ———————— END OF FILE ————————
```


THE OBJECT FILE

The OBJECT file defines all objects used in the adventure. For our adventure we will initially define only six objects: the player's Gun, a Trapdoor, a Memo, the secret Plans, a Computer Disk, and a KGB Agent. The KGB Agent will be a special type of Object called a Non-Player Character, or NPC. The difference between an NPC and a regular object is that a regular object's program is executed only when the object is in the same room as the player or in the player's inventory when he does an INVENTORY command, whereas an NPC is executed before every turn. This allows you to have characters who move around in their "universe" and do things when the player is not in the same room. Note that an NPC does not have to be a character, an animal, or ANYTHING. It is simply a generic term for an object, the program of which is executed at every turn.

Alternately, regular Objects can and should be characters, so long as they don't need to move or do anything else when the player is in a different room. Sometimes, it may be easier to describe a person in the room code, have him do or say something, and then leave the room. To do this, you needn't declare any objects or NPC's; you only need to "fake" the character's existence in the room's program. This presents a drawback however, as no actions can be added to this character since he does not really exist as a separate object.

The OBJECT block is similar to the ROOM block, but includes several additional sections. Start with either the "OBJECT" or the "NPC" keyword, follow it with a space, and the object's name. Note that this name is the name YOU will use in your program to refer to this object. This name must be unique. That is, if your have several keys in your adventure, you must name them differently (e.g., Key1, Key2, GoldKey, etc.).

Next is the NAME line. This line contains "NAME", followed by a space and the name(s) by which the player will refer to this object (separated by commas if there are more than one). Think of every possible synonym for the object and use as many as possible.

Following the NAME line is the ADJ line, which is the same as the NAME line only we substitute "ADJ" for "NAME" and, in place of the names for the object, list adjectives describing that object. Several objects can have the same "name" listed, but each should have different adjectives so they can be told apart by the player. When two objects that are called the same thing by the player are in the same room, PADV will require the player to specify an adjective to tell them apart, so be sure to describe them differently to the player.

Next, insert an ATTRIB block if you wish. Follow the same procedure we used for the room, only now you may have up to sixteen attributes per object.

After the ADJ line or ATTRIB block is "INITROOM", followed by a space and the name of the room where that object will be at the start of the adventure. It is usually a good idea to declare a spare room, not accessible to the rest of the adventure, in which you may store objects not currently in use. This is the purpose of the room "NoWhere" in our short adventure, and we will specify most of our objects to start there.

The CODE block follows the INITROOM line, and should contain statements dictating everything that object should do when the player is in the room, when it is in the player's inventory and he performs an INVENTORY command, or for every turn if the object is an NPC. Start this block with a "CODE" line, and end it with an "ENDCODE" line.

Next will follow any number of ACTION blocks. Here we define EVERY action that should be allowed to be done to this object, and any actions that the player is likely to try. For example, almost every object should be able to be examined by the player. Start each block with "ACTION", a space, and every synonym for the verb this ACTION represents (separated by commas if there are more than one). Fill this block with normal *T.A.C.L.* commands, and end it with "ENDACT". After all actions for this object are defined, end the object with "ENDOBJECT" or "ENDNPC".

Now we will cover the IF, WIN, and DIE statements. The IF statement is used to control how things happen and is probably the most useful command in the *T.A.C.L.* language. The IF statement consists of a line starting with "IF", followed by some condition that must be met for the enclosed statements to be executed, and optionally followed by a "THEN" statement. (See below or Section IV for a list of all the forms that the condition may take.) Immediately following the IF statement are all the statements that you want to be executed if the condition is met. Optionally, if you want to check multiple conditions, you may use one or more ELSIF statements after an IF statement, but before the ELSE or ENDIF statement.

The ELSIF (note how it is spelled) statement has the same form as the IF statement, but will only be executed if the last IF or ELSIF statements are not met. If the ELSEIF condition is met, then the code following it will be executed. Otherwise it will execute the code after the ELSE statement (if there is one).

IF statements may be nested to create more complex logical sequences. Carefully look at how the IF statement is used in some of the included sample adventures. If you want something else to be done only

when the condition(s) is/are false, then put "ELSE" on a line all by itself. (It doesn't check anything; rather it uses the results of the previous IF's and ELSIF's, and only executes its block of code if all those comparisons fail.) Always make sure to tell the computer that you are finished with everything that belongs in that IF block by putting in an ENDIF statement (simply "ENDIF" on a line by itself) at the end of the IF's code.

For example, if we want to see if the player has killed the Agent and, if not, check to see if he has any bullets left, we would use:

```
IF Agent IS Dead THEN
 T The Agent is dead.
ELSIF Bullets = 0 THEN
 T Be Careful!  You don't have any bullets,
 T and there is an enemy agent after you.
ELSE * Agent is alive and Player has bullets.
 T Be on the lookout for the enemy agent.
ENDIF
```

This allows you to control many different things (directions, doors, objects, etc.) based on what the player may do at any point in the adventure. Here is a list of all the forms of the IF statement. You should try to remember all of them, although you can always refer to either this section or to Section IV should you need to look up the form of an IF statement.

FORMS OF THE "IF" STATEMENT

```
IF PLAYER IN <room_name> THEN
```

Will execute the following block of code if the player is in the listed room.

```
IF PLAYER HAS <object_name> THEN
```

Will execute the following block of code if the player has the listed object in his inventory.

```
IF <object_name> IS <attrib_name> THEN
```

Will execute the following block of code if the listed object's listed attribute is SET, or "Y".


```
IF <object_name> NOT <attrib_name> THEN
```

Will execute the following block of code if the listed object's listed attribute is UNSET, or "N".

```
IF <object_name> IN <room_name> THEN
```

Will execute the following block of code if the listed object is currently in the listed room.

```
IF <room_name> IS <attrib_name> THEN
```

Will execute the following block of code if the listed room's listed attribute is SET, or "Y".

```
IF <room_name> NOT <attrib_name> THEN
```

Will execute the following block of code if the listed room's listed attribute is UNSET, or "N".

```
IF THISROOM IS VISITED THEN
```

Will execute the following block of code if the player has already been in whatever room he is currently in.

```
IF THISROOM IS DARK THEN
```

Will execute the following block of code if the player's current room is dark. Note that when used in objects' programs, "THISROOM" IF statements give the expected result only for objects in the player's current room. (Normal objects are only executed when the player is in the room, but NPC's always execute.) THISROOM always refers to the PLAYER's current room, not the object's current room. Therefore, be extremely cautious when using this statement in an NPC's code.

```
IF <variable_name> =<>#
   <constant or variable_name>
```


Will execute the following block of code if the listed variable's value has the listed relationship to the listed constant or variable. The accepted relations are "=" (equal to), "<" (less than), ">" (greater than), "<=" or "=<" (less than or equal to), ">=" or "=>" (greater than or equal to), and "<>" or "#" (not equal to). MADV will accept other comparisons, but PADV will complain about them in the debugger and they will ALWAYS evaluate to TRUE (the condition was met). Remember that a constant is just a number, like "15" or "-999", that never changes.

```
IF PREPOSITION IS <preposition name> THEN
```

Will execute the following block of code if the preposition that the player last used (if any) is the same as the one listed. This statement should be used in the ACTION blocks to determine how the player is attempting to use an object. PADV has a set list of words that are recognized as prepositions. Some are not real prepositions, but have been included because of their usefulness in certain contexts. Here are all thirty-two "prepositions" that *T.A.C.L.* will recognize:

ABOUT	ABOVE	ACROSS	AGAINST
ALONG	AMONG	AROUND	AT
BEFORE	BESIDE	BETWEEN	BEHIND
BELOW	BY	FOR	FROM
IN	INSIDE	INTO	OF
OFF	ON	ONTO	OUT
OUTSIDE	OVER	PAST	TO
TOWARD	UNDER	UPON	WITH

```
IF USING <object_name> THEN
```

Will execute the following block of code if the OBJECT noun of the player's last command sentence is equal to one of the names of that object listed. This picks the object noun out of a sentence, or out of a prepositional phrase. For example, "Shoot the Agent with the Gun", will try to execute the action SHOOT against the object AGENT, with the PREPOSITION being "with" and USING the GUN.

The WIN and DIE statements are essentially the same. Use either statement to end the current game. Each statement consists of either "WIN" or "DIE" on a line, without any parameters. There is no real functional difference; that is, they will both end the game, but it is helpful to you as the adventure writer to know whether the player has won the adventure, or has once again failed!


```
* ———— FILE SPY.OBJ ————
NPC Agent
NAME KGB, Agent, Man, Dude
ADJ KGB, Secret, Mean
ATTRIB
 Dead   N
ENDATTRIB
INITROOM NoWhere
CODE
IF Agent NOT Dead THEN
 DROP Gun  * Player always has this object.
        * The following gives us his position.
 IF GunPOS = AgentPOS THEN
  IF Time = 0 THEN
    T You see a Russian agent behind you.
    EQU Time 1
  ELSE
   T The Russian Agent Shoots You.
   CALL KillPlayer
  ENDIF
 ELSE    * Player NOT in the same room.
  EQU Time 0
  RANDOM AgentFloor 9   * determine a random floor for agent
  ADD AgentFloor 1      * to go to!
  IF Floor = AgentFloor THEN
   PLACEOBJ Agent StateHouse
  ELSE
   PLACEOBJ Agent NoWhere
  ENDIF
 ENDIF
 GRAB Gun  * very important to put gun back in inventory!
ENDIF
ENDCODE

ACTION Shoot, Kill, Destroy
IF Bullets > 0 THEN
 T You shoot the agent with precise aim.
 T He staggers and falls out the window.
 SUB Bullets 1
 SET Agent Dead
 PLACEOBJ Agent NoWhere * hide the agent from view
ELSE
 T But you've already used all your bullets!
ENDIF
ENDACT
ENDOBJECT

OBJECT Gun
NAME Gun, Cannon, Beretta
```



```
ADJ Small, Black, Sleek
INITROOM PLAYER
CODE
 T A small 6 shot Beretta.
ENDCODE

ACTION Look, Examine, Scope, View, See
 T A very nice, sleek, black automatic Beretta
 T that contains @Bullets hollow point bullets.
ENDACT

ACTION Shoot
 T SHOOT YOUR GUN???  How?  You might try
 T shooting the guard, but can't shoot your gun
 T unless you have another gun to shoot it with!
ENDACT

ACTION Fire
IF PLAYER HAS Gun THEN
 IF USING Agent THEN
  IF Bullets > 0 THEN
   T You shoot the Agent, he staggers with
   T the blow, and falls out of the window.
   SET Agent Dead
   SUB Bullets 1
   PLACEOBJ Agent NoWhere
  ELSE
   T Sorry, you're out of bullets.
  ENDIF
 ELSE
  IF Bullets > 0 THEN
   T You enjoy shooting a hole in the wall.
   SUB Bullets 1
  ELSE
   T You've used all your bullets - You're out!
  ENDIF
 ENDIF
ELSE
 T But you don't have the gun.
ENDIF
ENDACT
ENDOBJECT

OBJECT Trapdoor
NAME Door, TrapDoor, Panel
ADJ Trap, Secret
ATTRIB
 Open N
ENDATTRIB
```



```
 INITROOM Embassy
 CODE
 ENDCODE

 ACTION Look, Examine, Scope, View, See
  T It looks like a dusty door.
  IF Trapdoor IS Open THEN
   T It is open.
  ELSE
   T It's tightly closed.
  ENDIF
 ENDACT

 ACTION Open, Lift
  IF Trapdoor IS Open THEN
   T But it's already open!
  ELSE
   T You pull the trapdoor open.
   SET Trapdoor Open
   DIRECTIONS Embassy N D
   LINK Embassy D SecretTunnelSouth
  ENDIF
 ENDACT

 ACTION Close, Shut
  IF Trapdoor IS Open THEN
   T You close the trapdoor.
   UNSET Trapdoor Open
   DIRECTIONS Embassy N
  ELSE
   T It already is.
  ENDIF
 ENDACT
 ENDOBJECT

 OBJECT Memo
 NAME Memo, Note, Paper
 ADJ Secret, Small
 INITROOM NoWhere
 CODE
 IF PLAYER HAS Memo THEN
  T A Secret Memo
 ELSE
  T There is a small memo on the floor here.
 ENDIF
 ENDCODE

 ACTION Look, Examine, Scope, View, See, Read
 T It is a secret memo.  It says that the plans
```



```
 T were given to Dr. Soifer on floor @PlansFloor .
 ENDACT

 ACTION Get, Grab, Take, Swipe, Cop, Steal
  IF PLAYER HAS Memo THEN
   T You already have it.
  ELSE
   T You grab the memo.
   GRAB Memo
   EQU MemoFloor 0
  ENDIF
 ENDACT

 ACTION Drop, Lose, Rid
  IF PLAYER HAS Memo THEN
   T You drop the offending Memo.
   DROP Memo
   EQU MemoFloor Floor
  ELSE
   T You don't have a Memo.
  ENDIF
 ENDACT
 ENDOBJECT

 OBJECT Plans
 NAME Plan, Plans, Tube, Paper
 ADJ Secret, Stealth, Bomber, Rolled,
 INITROOM NoWhere
 CODE
 IF PLAYER HAS Plans THEN
  T Top Secret Plans
 ELSE
  T There is a tightly rolled tube of paper here.
 ENDIF
 ENDCODE

 ACTION Look, Examine, Scope, View, See, Read
 T It is a set of top secret stealth bomber plans!
 ENDACT

 ACTION Get, Grab, Take, Swipe, Cop, Steal
  IF PLAYER HAS Plans THEN
   T You already have them.
  ELSE
   T You grab the set of plans.
   GRAB Plans
   EQU PlansFloor 0
  ENDIF
 ENDACT
```



```
 ACTION Drop, Lose, Rid
  IF PLAYER HAS Plans THEN
   T You drop the secret plans.
   DROP Plans
   EQU PlansFloor Floor
  ELSE
   T You don't have the plans.
  ENDIF
 ENDACT
 ENDOBJECT

 OBJECT Disk
 NAME Disk
 ADJ Floppy, Computer, Secret, Stolen
 INITROOM NoWhere
 CODE
 IF PLAYER HAS Disk THEN
  T A stolen computer disk.
 ELSE
  T An innocent-looking computer disk is here.
 ENDIF
 ENDCODE

 ACTION Look, Examine, Scope, View, See
 T It is a computer disk.  Its label says
 T      "TOP SECRET INFORMATION"
 ENDACT

 ACTION Get, Grab, Take, Swipe, Cop, Steal
  IF PLAYER HAS Disk THEN
   T You already have it.
  ELSE
   T You grab the computer disk.
   GRAB Disk
   EQU DiskFloor 0
  ENDIF
 ENDACT

 ACTION Drop, Lose, Rid
  IF PLAYER HAS Disk THEN
   T You drop the computer disk.
   DROP Disk
   EQU DiskFloor Floor
  ELSE
   T You don't have the disk.
  ENDIF
 ENDACT
 ENDOBJECT
 * ———— END OF FILE ————
```


THE SUBROUTINE FILE

The subroutine file contains special programs that can be called by any other program in the adventure. These can be short functions that display random messages when the player does something crazy like pick up something he already has, or they can compute numeric functions, end the player's life, generate musical sequences, or anything else you may need to do at more than one place in the adventure.

We will define three short subroutines for the sample adventure. One will initialize the variables, one will be a sub-subroutine used only by the first subroutine, and the third will be used when the player dies. The third will make it easy to add graphics later also. Since every time the player dies it will call this subroutine, we need only add one line to have it display an R.I.P. message or a skull and crossbones image.

To define a subroutine, start with the keyword "SUBROUTINE" or "SUB", followed by space and the name of the subroutine. Follow this with a program to perform whatever function you want it to, and end the block with a "ENDSUB" line.

To our current knowledge of *T.A.C.L.*, we now add the CALL, GRAB, DROP, GO and MOVE statements. The CALL statement is simple, powerful, and easy to use. A CALL statement consists of "CALL", followed by the name of a subroutine. When this statement is executed, it will be just as though the whole subroutine code were in the same place as the CALL statement. When everything in the subroutine that was called has been done, then execution continues on the next line after the CALL statement. PADV will allow CALLs to be nested up to 128 deep. After that, it will ignore further call statements and continue execution.

The GRAB statement is used to move an object into the player's current inventory. Its format is "GRAB", followed by the name of the object to GRAB. Note that this must be done specifically in your code, usually when the player tries to get, take, or grab that object, but you may prevent this from happening if you do not want them to be able to grab it. (If it's too large or heavy, or if they have too many objects in their inventory, or for any other reason you may decide.)

Likewise, the DROP statement is used to move an object from the player's inventory to the player's current room. Like the GRAB statement, you must specifically program this to occur. For instance, this allows for cursed rings that cannot be removed until some condition is met. Its format is like the GRAB statement. Type "DROP", following it with the name of the object you wish to remove from the player's inventory.

The GO statement is used to force the player into another room. When this occurs, the current room's program will finish executing and the program for the player's new room will then be executed. To use this statement, create a line starting with "GO" and follow that with the name of the room you wish to send the player to. This allows you to cause the player to fall through trapdoors, be transported, or otherwise moved from one room to another.

The MOVE statement also forces the player into another room, but instead of specifying the name of the room, the DIRECTION to force the player in is given. This command is similar to the player moving in the direction you specify, except here the player doesn't have any control over what is happening. Follow "MOVE" with the abbreviated direction name to push the player in, if that is possible. If that direction is not allowed, then the player will simply remain in the current room. This might be used to simulate a player's drunkenness by forcing him to stagger from room to room for several turns.

```
* ———————— FILE SPY.SUB ————————
SUBROUTINE Randomize * set random floors for the objects
 CALL RSub
 EQU MemoFloor  Time
 CALL RSub
 EQU DiskFloor  Time
 CALL RSub
 EQU PlansFloor Time
 EQU Time 0
ENDSUB

SUB RSub  * generates a random number between 2 and 9
 RANDOM Time 8
 ADD Time 2
ENDSUB

SUBROUTINE KillPlayer
 T Too Bad, but it appears that you have DIED!!!
 T
 T      ...Please don't let it ruin your day.
 * you could display an image here!
 DIE
ENDSUB

SUBROUTINE Elevator
IF PLAYER IN Elevator THEN
 T The elevator doors closes, you wait a moment,
 T and the doors open again.  The indicator says
 T you are on floor @Floor .
```



```
ELSE
 T What???  Where?  What are you talking about?
ENDIF
ENDSUB
* ———————— END OF FILE ————————
```

THE VOCABULARY FILE

The vocabulary file allows you to define new words and phrases that do not directly correspond to objects. Any actions performed directly with or on objects should be defined in the OBJECT itself as an ACTION. Use vocab blocks for phrases and directions like "SLEEP", "HEALTH", "CREDITS", etc. that do not have a specific object they function on. Start this file with a "VOCAB" line, and define ACTION blocks just like in the OBJECTs file. The only difference is that these blocks do not correspond to any objects in particular, and they may be as many words long as you like. You need not include the articles "a", "an", "the", and "that" because they will be filtered by PADV anyway. End the file with an "ENDVOCAB" line.

Before we finish the test adventure, we will introduce the PLACEOBJ, MOVEOBJ, RANDOM, and SCORE statements.

PLACEOBJ is used to move an object to an absolute location. Specify "PLACEOBJ", the name of the object to be moved, and the name of the room you wish to put it in, and the object will instantly move to that room. "THISROOM" can be substituted for the room's name, causing the object to be sent to the player's current room.

MOVEOBJ is like PLACEOBJ, but instead of telling an object which room to go to, we tell it which direction to go in. The format is "MOVEOBJ", the object's name, and the abbreviated direction to move in. (You must use "N", "S", "NW", "U", etc. You may NOT use "North", "South", etc.) This will attempt to move the indicated object to the room that is linked to the direction indicated.

The RANDOM statement is very useful for making events differ from game to game, or even within a single game. You must decide whether you want random events to occur in your game. The format of the RANDOM statement starts with the "RANDOM" keyword, followed by the name of the variable to put the random number in, and then the maximum value to generate PLUS ONE. The maximum value can be either a constant or a variable, and if it is negative, RANDOM will generate a negative number. All numbers generated are between (and including) zero and the maximum value minus one.

The SCORE command is just a convenient way of manipulating the SCORE variable. Use "SCORE", followed by a variable or a constant, and that value will be added to the player's current score. (If the value is

negative, the player's score will be decreased.) Remember that the limit on all variables is between -32768 and 32767. Do not exceed these limits, or else the score will wrap around to the opposite end of the scale.

The rest of *T.A.C.L.'s* commands are described at the end of this section. Be sure to read over them carefully before beginning your own adventure.

```
* ———————— FILE SPY.VOC ————————
VOCAB

ACTION Sleep, Rest, Snooze
 T Zzzzzzzzzzzzzzzzzzzzzzzzzzz. <YAWN>  Happy?
ENDACT

ACTION Wait, Pause
 GO THISROOM * Do Nothing, then re-execute room
ENDACT

ACTION CREDIT, CREDITS
 T I Don't get enough!
 T       ..Respect either!
 T                       ..or Money!
ENDACT

ACTION Push 1, Push One, Press 1, Press One
 EQU Floor 1
 CALL Elevator
ENDACT

ACTION Push 2, Push Two, Press 2, Press Two
 EQU Floor 2
 CALL Elevator
ENDACT

ACTION Push 3, Push Three, Press 3, Press Three
 EQU Floor 3
 CALL Elevator
ENDACT

ACTION Push 4, Push Four, Press 4, Press Four
 EQU Floor 4
 CALL Elevator
ENDACT

ACTION Push 5, Push Five, Press 5, Press Five
 EQU Floor 5
 CALL Elevator
ENDACT
```



```
ACTION Push 6, Push Six, Press 6, Press Six
 EQU Floor 6
 CALL Elevator
ENDACT

ACTION Push 7, Push Seven, Press 7, Press Seven
 EQU Floor 7
 CALL Elevator
ENDACT

ACTION Push 8, Push Eight, Press 8, Press Eight
 EQU Floor 8
 CALL Elevator
ENDACT

ACTION Push 9, Push Nine, Press 9, Press Nine
 EQU Floor 9
 CALL Elevator
ENDACT

ACTION Push 10, Push  Ten, Press 10, Press Ten
 EQU Floor 10
 CALL Elevator
ENDACT

ENDVOCAB

* ———————— END OF FILE ————————
```

COMPILING AND DEALING WITH COMPILER ERRORS

To compile this adventure, first type all five of the files presented into the computer with your text editor, and save them to disk. It is usually a good idea to keep all your files in the same directory, but if you acquire a large number of files, you can place most files in subdirectories, with the ADVENTURE file in the parent directory of all other directories. DO NOT specify the whole path for other files in the ADVENTURE file, just the subdirectory name, a slash, and a filename. This is acceptable practice for most files. However, IMAGE files should always remain in the current directory because their pathnames are memorized so that PADV can find them later. If you don't keep them in the current directory, PADV will expect to find them in the same subdirectories, which can be very inconvenient if you decide to distribute your adventure (over a BBS or otherwise).

After all files have been entered and saved, type "MADV Spy". Sit back and watch as MADV zips through your files. Generally, each period (".") printed represents one object, one room, or whatever for that file,

so you can monitor MADV's progress through your adventure. If any problems arise (typos, misplaced statements, or other problems), just wait for the compile to finish. (If necessary, MADV can be halted using the Control-C key combination.) All errors printed are duplicated in a file called "Spy.ERR" for your convenience. Display the file by typing "TYPE Spy.ERR". Refer to Appendix A: MADV Errors for a quick description of each error and its probable cause, and refer to the remainder of this section for more specific information on finding errors.

First, find which file(s) the error(s) is/are in, and load the first file into your editor. Then note the line number of the error (at the beginning of the line). Fire up your editor and load in the file in which the problem was found. Move to the line number where MADV found the error, and try to locate the problem. If you cannot find anything wrong, check the order of the different code blocks used. If you really get stuck, try looking at the source code for some of the included sample adventures to see how we made it work. Try breaking complex problems down into small discrete steps that can be written more easily.

If you still can't find anything wrong, check the exact syntax (or "format") of the command with the full-reference listing in Section IV: The *T.A.C.L.* Language. If it matches properly, check the same for the line or two before it. Keep in mind that one error can sometimes make MADV think the following lines are wrong, too. Recompile and see if it is still rejected by MADV. If you are still having problems, try making the problem line into a comment (so MADV will ignore it), and see if the rest of your code compiles. Be creative – finding and eliminating bugs is the most difficult aspect of programming. That is what many programmers get paid for – debugging, not programming.

If you follow all the rules and think you have found a bug in *T.A.C.L.*, isolate it by making as short an adventure as possible that still causes the bug. If you can isolate and reproduce the error, we would like to hear about it. Please fill out the enclosed bug report form, and mail it in so we can fix the problem before the next version of *T.A.C.L.* is released. If possible, please include a listing of the test adventure. Thank you in advance for helping to make *T.A.C.L.* a bug-free system!

OTHER *T.A.C.L.* COMMANDS

Now we will explain the function and use of every T.A.C.L. statement that has not yet been covered.

The remaining commands are: DIRECTIONS, LINK, SET, UNSET, SHOW, IMAGE, NOTE, and the mathematical functions EQU, ADD, SUB, MULT, and DIV.

The DIRECTIONS statement determines which directions the player is allowed to travel from a room. Follow the "DIRECTIONS" keyword with the name of the room to set the directions for, and follow that with all the abbreviated direction names that are available from that room, separated from each other by spaces, and in any order you like.

The LINK command is used with the DIRECTIONS statement to tell PADV which room to go to for each available direction. Note that the previous room links (if any) remain in effect unless re-LINKed. The direction command does not remove links when you remove a direction from the available list, it just will not allow the player or objects to go in that direction. To use the LINK command, start with "LINK" and follow it with the name of the room to set an exit for, then a space and the direction to set a link for, followed by another space and the name of the room that lies in that direction. Unless you wish to "move" rooms or open and close doors, you will probably never have to re-LINK a room. Be VERY careful to always LINK all available directions; the debugger will tell you when you haven't, but without the debugger running, the results will be unpredictable.

SET and UNSET are used to change the state of a room's or object's attributes. SET will make that attribute a "Y", and UNSET will make that attribute an "N". Their format is the same. Use either "SET" or "UNSET", and follow it with either the name of an object or the name of a room, then finish with the name of the attribute that is to be changed for that object or room. "THISROOM" may also be used in place of an actual room name to indicate the player's current room, but this applies if, and only if, the attribute you want to set or unset for the current room is either VISITED or DARK.

The two graphic commands are SHOW and IMAGE. The SHOW command will immediately display an image if the player has Graphics turned on. Use "SHOW" and the name of either an image in one of the VGED files listed in the .ADV file, or the filename of an IFF picture (also listed in the .ADV file). The IMAGE statement will associate an image with a particular object or room, and list its name in the Image Select window in PADV. The player will only see these if he clicks on the appropriate name in the graphic select window. To attach an image to an object or room, use "IMAGE", then the name of the object or room, followed by either the name of an image (VGED or IFF image), or the keyword "REMOVE" which will remove any image previously attached to that room or object. This allows images to change throughout the game, even if the object doesn't. (One possible use might be a magical map that fills itself in when the player accomplishes certain goals.)

To employ sound or play music, an easy-to-use, note-generating facility is available. Use the command "NOTE", followed by two values (either constants or variables). The first value should be between 1 and 1000, and designate the pitch of the note played. (See Appendix C for a table of note values.) The second value is the delay specified in fiftieths of a second. This value should be between 1 and 500 – that is, between 1/50th of a second and 10 seconds long.

Because of their limited use, math functions in *T.A.C.L.* are functional, but not fancy. However, *T.A.C.L.'s* facilities should be more than suitable for any standard adventure applications.

The EQU statement is the *T.A.C.L.* assignment command. EQU simply sets a variable equal to either another variable or a constant. Its format is "EQU", followed by the destination variable, followed by either a number or the name of a variable.

The ADD statement adds the second value given – whether it is a variable or a constant – to the variable that follows the ADD statement. Once again, keep in mind that a variable's value is limited to the range of -32768 to 32767.

The SUB, DIV, and MULT statements are the same as the ADD statement, except that the value is either subtracted from the initial variable, divides the initial variable, or multiplies the initial variable. The same value limits apply to all these functions as well.

Note that the "SUB" keyword may be used in three different contexts. In the .ADV file, SUB denotes the start of a block that lists the names of the files containing subroutines. In those files, SUB or SUBROUTINE is used to start a new subroutine. However, within a subroutine, and everywhere else in the source code, SUB is the subtract statement. This should not cause any problems, as MADV knows how to handle each use, and how to tell them apart.

The "IMAGE" keyword is also used in more than one way. In the .ADV file, "IMAGE" starts the block that describes all the image files available for the adventure. But in the rest of the adventure files, it is used to attach an image to an object or room. Be aware of all the uses of each command so as to avoid any confusion later.

SUGGESTED ADDITIONS

If you liked the concept of the test adventure we constructed, you may want to build upon it. Here are a few suggestions for additions and enhancements. Whether you modify the sample adventure, or start your own from scratch, you may want to incorporate the following suggestions and concepts into your adventure game.

1. Add "scenery" objects (e.g., walls, windows, desks). These objects may not serve any real purpose, but they do help to keep an adventure from seeming "bare" and "empty".

2. Make the Russian State House a real ten-story building (i.e., use ten rooms, not just one). Although this may mean a lot more code, it will allow you to customize each floor more easily, and you will not have to worry about things being dropped on one floor and appearing on all other floors.

3. Add other objectives and intermediate goals, and complicate the current one. Try adding new agents (including ones on your side) and new obstacles to work through.

4. Construct an arsenal of unusual gadgets, like James Bond himself always has. Try including weapons, lockpicks, or special computers.

5. Add additional vocabulary statements to handle any special commands that you are likely to encounter – for instance, Sleep, Sit, Stand, Yell, Scream, or Talk. Most commands of this type deal with what the character himself is doing. You may also wish to implement vocabulary statements to rebuke the player (and/or lower his score) for being obscene, or trying to destroy or kill everything and everyone in the adventure. Carefully worded, humorous statements are generally best.

6. Add more rooms to the adventure. Where they are added is up to you. The Embassy would be a good start: the Russian State House is crying out for more rooms and passages. And, of course, the Secret Tunnel could easily link up to a whole network of passages and tunnels that lead to more buildings. Creating a deadly barrier (like the streets) is usually a better way to limit your adventure than to merely say, "You can't go that way." It leaves the impression that all of Moscow (and the rest of the universe) is still there – it is just currently inaccessible to your character.

7. Add more color and flavor to the descriptions. They were purposely kept brief here, but they generally should be longer, and more descriptive.

8. Let others play your adventure, and ask for suggestions: What might be added? Changed? What is too confusing? Too easy? All comments and suggestions should be considered.

9. Add sound effects and music. Maybe a warning sound when the agent appears, or other complementary audio effects for when the player wins, dies, or finds something that is important.

10. Add graphics. (Refer to Section II: VGED.) Create images for all important rooms and objects in the adventure, as well as death scenes, victory scenes, etc.

SECTION II

VGED (Vector Graphics EDitor)

VGED is a special kind of paint program. Rather than saving a picture after you are done, VGED instead saves a "recipe" to redraw the picture. This makes graphic files MUCH smaller, and more easily transferable between different computers. The Amiga version also supports standard IFF images, although not all computers will be able to display these. Keep this in mind when designing graphics for your adventure.

GRAPHICS CONSIDERATIONS

When deciding whether to include graphics in your adventure, be sure to consider all the pro's and con's. Even if you aren't much of an artist, it usually will not take long to draw up illustrated signs, room maps, diagrams, special screens for specific events, or special warning screens. You may even want to include a few audio notes to complement the graphics. Also, since IFF images are accepted, you can use all the facilities of your favorite paint program, digitizer, frame grabber, image scanner, or graphic tablet to create images for your adventure.

Given the wide variety of image-making capabilities, there aren't too many reasons not to include at least some graphics in your adventure. The extra style and professional quality that graphics can add to your adventure are usually well worth the effort.

How many images should you add? That depends on what type of images you are using, and how much time it takes to create each one. If you are using many large IFF pictures, you will have to limit your graphics to what will fit on a disk. If you are using VGED, space should not be a problem unless you create literally hundreds or thousands of complex images, but doing so also limits you to the VGED editor because no other program can create VGED images. We suggest a mixture of the two, and a consistent method to distribute images throughout your adventure. Here is one suggested plan:

1. IFF picture Title Screen (possibly HAM).
2. Either a VGED or IFF picture for each important room or region of similar rooms. (A Plan view is easiest to draw.)
3. Make either a VGED picture, or an IFF brush for each important or unusual object in the adventure, keeping in mind that an object's appearance may change as the game progresses.
4. Make a special graphics screen for special events (e.g., when the player dies, wins, or possibly an image for when something sudden happens).

Note that it is usually not necessary to include this many images. The included images should start with places where you consider it important to use an image to clarify or add important details.

Next, add images to any portion of the adventure where you have an idea for an interesting picture. Try not to make your adventure dependent on the included graphics unless you specifically mention at the start of the adventure that the included images are necessary to complete the adventure.

Lastly, create enough images to fill the adventure evenly. That is, try not to bunch all your images at the start or at the end. If you must distribute them unevenly, do so by order of importance to the adventure.

MAKING YOUR FIRST VGED IMAGE FILE

The first step in creating a VGED image file is to run VGED. This can be done from either the Workbench or the CLI. From the Workbench, just double-click on the VGED icon. From the CLI, type "VGED", or alternately, "Run VGED". VGED does not have any Command Line options, so there is nothing else to it.

Once VGED is up and running, select NEW from the PROJECT menu. VGED will pop up a file requester. Use this file requester to select the volume and directory you want to save your graphic files to, and enter the name to use. If you already have a file started, and just want to add to it, or make changes to it, select LOAD from the PROJECT menu instead of NEW.

Regardless of the option you choose, after you select the filename, VGED will ask for a password. If you wish to keep your images from spying eyes, enter your own secret password. Be sure to remember this password. (It might be a good idea to keep a notebook containing all the passwords for your graphic files.) Try to use different passwords for each file. That way, if someone discovers one of your passwords, they still will not be able to access all of your files. However, if you want other

T.A.C.L. users to have access to your images, use the standard password: "PUBLIC". If you wish to copyright your drawings, or at least claim them as your own, it is a good idea to use a password. We also recommend using a different password than the one you use for your adventure.

Now you are ready to draw your first picture. (See the next section for details on VGED's available options.) When you have finished drawing your image, select the SAVE option from the IMAGE menu. VGED will ask for the name of the image, and then add it to the graphic file. Note that in a VGED session it is not necessary to select NEW before you draw the first picture; if you simply begin drawing, you can select NEW PROJECT or LOAD PROJECT, then SAVE IMAGE afterwards. If you wish to determine how much memory your images actually take, remember that each graphic file has a base size of 14,856 bytes.

Make sure to read the next part of this section carefully to learn how to use VGED to its full potential.

VGED's CAPABILITIES

VGED is designed to produce images that can be quite complex while taking little memory. It is not designed to compete with DeluxePaint, or any of the other professional graphics packages available for the Amiga. However, with a base file size of less than 15 kilobytes, and an average image size of less than 1 kilobyte, you can store literally five hundred to a thousand or more images on a standard Amiga disk.

VGED displays some important information in its title bar. On the left, the name of the current graphic file and the name of the current image are separated by a period. Near the center, VGED displays the current X and Y position of the mouse pointer, followed by the current line length (or radius) when you are drawing lines and polygons. On the right is a word describing the current drawing mode.

VGED has been designed to be very straightforward and easy to use. There are three ways to talk to VGED: through the Menus, the Help Window, and the Keyboard Shortcuts. Almost all options are available from both the mouse and the keyboard. We recommend that you use the mouse to draw with, although you can even do that with the keyboard if you wish.

VGED's MENUS

VGED has three menus. The first of these, the PROJECT menu, allows you to create or select a graphic file. (Remember that each file can, and should, have multiple images.) The PROJECT Menu contains the following items:

NEW	(Right-Amiga-B)
LOAD	(Right-Amiga-I)
ABOUT	(Right-Amiga-A)
HELP	(Right-Amiga-H)
QUIT	(Right-Amiga-Q)

Use the NEW or LOAD option to select a graphic file. Use NEW if you want to create a new file, and LOAD if you want to modify or add to an existing file. Remember, these options select files with multiple images, and do not affect individual images. Both these options will display a file requester to ask for the name of the graphic file to load or create.

The ABOUT option will display a window with pertinent information about the program and its copyright status. Display it often.

The HELP menu option, the Right-Amiga-H, or the HELP key on the keyboard will toggle the Help Window on and off. The Help Window is smart; that is, it remembers where you last placed it, and returns there when you recall it. Usually you will want it on to select the different modes, colors, and patterns, but sometimes it can get in the way.

The QUIT option will, but not before it checks with you first to make sure you are really serious. Like PADV, you can type "Y" or "N" from the keyboard, rather than click on the appropriate gadget in the requester.

VGED's second menu is the IMAGE menu. It deals with functions that affect the individual images, rather than the file as a whole. This menu is as follows:

NEW	(Right-Amiga-N)
LOAD	(Right-Amiga-L)
SAVE	(Right-Amiga-S)
REDRAW ALL	(Right-Amiga-R)
SET PALETTE	(Right-Amiga-C)
RENAME AN IMAGE	
DELETE AN IMAGE	
COMPRESS IMAGE FILE	

Use the NEW option to clear the screen so you can create a new image. Be careful not to activate this option if you wish to save your current image. Use LOAD to edit any image already saved in the file, and use SAVE to put the current image in the graphic file, whether you LOADed it or it is a NEW image.

Since PADV will draw every image from the start, it is often a good idea to know how long it takes to redraw the current picture. To watch a picture being redrawn, just select the REDRAW ALL option. Generally speaking, try not to create pictures that are so complex as to take more than five to ten seconds to draw, as the player may begin to wonder what's going on.

Every image can have its own color palette using any sixteen of the Amiga's 4096 available colors. To choose your favorite sixteen, use the SET PALETTE option, or use "Right-Amiga-C" (think of "C" for "Colors"). Select the color you wish to change by clicking on the appropriately colored box, and adjust the amount of Red, Green, and Blue in that color with the proportional gadgets at the bottom of the window. When you are finished adjusting the colors, select the OK gadget to retain the changes, the CANCEL gadget to revert to the colors previously in use, or the RESET gadget to change all colors back to the original defaults.

If you decide that the name you have assigned an image is not appropriate, or if it was used in another graphic file in the same adventure, and you wish to change it, select RENAME AN IMAGE from the IMAGE menu. This will allow you to choose the image from the standard image requester. You will then be presented with a window requesting a new name for that image.

If you find that one (or more) of your images are unsuitable for use in your current adventure, select DELETE AN IMAGE from the IMAGE menu. Then use the standard image requester to select the image to delete, and confirm that you want to delete it.

Finally, the COMPRESS IMAGE FILE option lets you reduce the size of the image file if any DELETED images are present. Since this is a reasonably time-intensive operation, it is not performed after each save and delete. VGED will attempt to create a temporary file in the "T:" directory, so be sure that there is enough room left here to hold a duplicate of your graphic file. It is generally a good idea to assign "T:" to a directory in RAM to speed up compresses.

The last menu in VGED is the OPTIONS menu. This menu has only two options:

MAGNIFY WINDOW
CROSS HAIRS

Each of these options has a submenu containing two options: ON and OFF. This allows you to turn the Magnification window on or off with the MAGNIFY WINDOW option, and toggle the full-screen cross hairs on and off with the CROSS HAIRS option.

VGED's Help Window allows you to access the many drawing functions by clicking in a window full of gadgets. Each gadget is presented below along with an explanation of how to use it.

VGED's GADGETS

First are the function gadgets. There are seven of these gadgets. They begin in the upper left corner and continue across the top and then down the right side.

The first, and often one of the most useful gadgets, is UNDO. Because of the way VGED builds images and remembers every step, the UNDO feature lets you undo every single step all the way back to the first item drawn.

The next gadget is the FILL gadget, which has a drop of paint depicted on it. Use this gadget with care. Because of the complex nature of this operation, it can take a long time to complete. Try to avoid FILLing large or complex areas. Use the POLYFILL instead of the FILL command wherever possible as it is much faster. Also, avoid filling within several pixels of your borders. If your adventure is played on a different computer, with a different resolution, the graphic translations may move your fill point slightly and end up filling in your border. This applies doubly when filling in and around text, because the character fonts will change with the computer system.

The gadget sporting the large, fashionable "T" is the TEXT gadget. Select this gadget whenever you need to write text to the graphics screen. A requester will appear, asking for the text string to print, and then allow you to move and place that string of text anywhere on the screen.

The gadget with a straight line cutting across it is the LINE gadget. To draw a line, select this mode, then place the cursor at the position you wish to start the line, click the left mouse button, move the cursor to the ending position, and click the left mouse button again.

The two remaining gadgets on the top line both bear a square and a triangle. However one is filled, and one is not. The outlined polygons selects the POLY mode. This mode pops up a requester that asks for the number of vertices. Enter a number between 3 and 100 to draw anything from a triangle, to a square, to a circle with 3, 4, and 20+ vertices, respectively. A circle requires a reasonably large number of vertices. If your circle is small, 20 or 30 might work well, but if it is larger, 50 to 100 might work better. Click the mouse at the exact center where you wish to draw your polygon, then move the mouse away from the point, sizing and rotating the shape until you have it where you want it. If you are not satisfied with the current position of the polygon, type "U" from the

keyboard, and the drawing function currently in progress will be abandoned. This also works for lines and POLYFILLs. The POLYFILL function, represented by the fill polygons, is the same as the POLY function, except it draws filled polygons instead of outlined ones. Use this in conjunction with the background color you are using to "erase" objects from the screen.

On the second row at the far right-hand side is a gadget that resembles a camera. Click on this gadget once and you are in CUT mode. You now have the power to cut out sections of the screen and paste them elsewhere at incredible speeds. Click the left mouse button at any corner of an area you want to clip. Stretch the box over the area you want, and click the left mouse button again. This will automatically put you in PASTE mode. (This works basically the same as a "brush" in DeluxePaint.) This snapshot is now available anytime, and in PASTE mode will follow the mouse pointer around. You can use the snapshot again – even if you have drawn new images since – by clicking TWICE on the snapshot gadget. The first click will put you in CUT mode, and the second will put you into PASTE mode.

The next section of the Help Window is the Colors section. There are eighteen gadgets to select colors. On the left side of the Help Window are sixteen gadgets, each with a different color. Just to the right of these are two larger, very important gadgets. The top gadget is the current foreground color, and the bottom gadget is the current background color. To change the current foreground color, click on the foreground color box (the big colored box in the middle on top), then click on the appropriate color box. To change the current background color, click on the background color box (the big colored box in the middle on the bottom), then click on the appropriate color box.

To the right of the colors are the patterns, sixteen in all. The upper left corner pattern is merely the solid foreground color. Click on the pattern you want to use for all fill operations (FILL and POLYFILL). A sample of the current pattern will be shown in a large box in the lower right-hand corner of the Help Window. This box is not a gadget and will not do anything when you attempt to select it. Everything else in this window is a fully functional gadget.

Finally, you can drag the Help Window to any position you want, or cause it to go away by clicking on the Close gadget in the extreme upper left corner.

VGED's KEYBOARD COMMANDS

All of VGED's options are easily accessible from either the Help Window or from the Menus. Most of these options are also available

from the keyboard. Here is a list of all the implemented keystrokes, not including menu shortcuts:

<Help>	The Help key will toggle the Help Window on and off.
A	Select AREAFILL (POLYFILL) mode.
C	Toggle CROSS HAIRS on and off.
F	Select FILL mode.
L	Select LINE mode.
M	Toggle MAGNIFY mode on and off.
P	Select POLY mode.
S	Toggle between the CUT and PASTE modes.
T	Select TEXT mode.
U	UNDO the last or current command.

The following apply to the NUMERIC KEYPAD ONLY!

7	Decrement foreground color number.
9	Increment foreground color number.
4	Decrement background color number.
6	Increment background color number.
1	Decrement current pattern number.
3	Increment current pattern number.

Now that you know how VGED operates, you need only worry about what to draw. Take your time, experiment, and enjoy. After you have drawn a few images and want to tie them into an adventure, continue to the last part of this section.

ADDING GRAPHICS TO YOUR ADVENTURES

Now that you know how to draw graphics, the important question comes up: How do I include the images in my adventure? Actually, this is quite easy. First, decide if each image will be attached to an object, or a room, and if it will be automatically displayed by your adventure. Now include all the names of the image files contained in the ADVENTURE file in the IMAGE block. On each line, include the name of a graphics file. If it is a VGED file, follow that name with that file's password. Here is a sample IMAGE block to add graphics to the "Spy.ADV" adventure file, assuming the existence of a VGED file called "Spy.GFX" that contains two images called "GunPic" and "Skull". We also assume the file's password is "BOND". Also assumed is the existence of an IFF file with the name "Spy.PIC".

```
IMAGE
 Spy.GFX  BOND
 Spy.PIC
ENDIMAGE
```

Suppose we want to display the "Spy.PIC" IFF image as our title screen, the "Skull" image whenever the player dies, and that we want to attach the VGED image "GunPic" to the Gun in the Spy adventure. To do this, we add the line:

```
IMAGE Gun GunPic
```

into the code block of the gun itself. This will allow the player to display the picture using the Image Select window in PADV. If we also want to force the display of this picture, we could add the line:

```
SHOW GunPic
```

to the "ACTION Look, Examine, etc." block of code, within the object Gun. To display your title screen, add the following lines to the start of the Embassy's CODE block:

```
IF Embassy NOT VISITED THEN
  SHOW Spy.Pic
ENDIF
```

This will only display the title screen the first time the player enters the room, or if the player does a "LOOK" command. To show the skull when the player dies, we add the line:

```
SHOW Skull
```

to the SUBROUTINE KillPlayer just before the line with the DIE statement. There, wasn't that easy? Enjoy adding graphics to your adventures and impressing friends with your ability to create high-quality graphic adventures!

This page for doodles.

SECTION III

PADV (Play ADVenture) Adventure Player

Now comes the fun part of *T.A.C.L.*, where you get to play the games that you or someone else has created with MADV. PADV has been designed to be as flexible as possible, with each computer's version having been written so as to take advantage of that particular computer's special capabilities. The Amiga version, for example, uses pull-down menus, special windows for various functions, a separate screen for graphics which may be manipulated using the standard Amiga layering and drag gadgets, speech, and other Amiga-specific features.

LOADING AN ADVENTURE

From the Workbench, either double-click on the icon of the game you wish to play, or double-click on the PADV icon to load PADV, and then load the game using the menu options in PADV as described below.

From the Command Line Interpreter (CLI), type "PADV <GameName>.GAM" or "PADV <GameName>". Or just type "PADV", and then load the game using the menu options as described below. Note that PADV often uses considerable stack space, and the stack size should be set to at least 20,000 bytes. To do this, type "STACK 20000" at the CLI prompt, or put it in your startup-sequence. (Just edit the file "s:Startup-Sequence" and add the "STACK 20000" line somewhere in the file.)

Debug Mode may be entered only from the CLI, and only on your own adventures. Simply include the "-E" switch after the adventure name when you run PADV, like this: "PADV <GameName> -E". After loading the game, PADV will ask for the Adventure password (the one for the ".ADV" file) before putting you in debug mode. This is to prevent others from moving things around, or otherwise using the powers of debug mode to easily master your adventure. Debug mode also opens

a window on the screen to report any run-time errors encountered. (See Appendix B: PADV Compiler Errors.) In addition to the error window, PADV will write a file with the adventure name with a ".ERR" ending instead of a ".ADV" ending. This file will list all errors encountered during your last adventure-playing session. The debug mode commands are described in detail later in this section.

PADV's MENUS

PADV has three main menus. The ADVENTURE menu gives you control over system options and which adventure game to play. The GAME menu allows Save, Restore, and Restart options. The OPTIONS menu allows you to toggle between Graphics, Audio, and Speech.

The ADVENTURE menu is as follows:

Load	**(Right-Amiga-R)**
Color	**(Right-Amiga-C)**
About	**(Right-Amiga-A)**
Help	**(Right-Amiga-H)**
Quit	**(Right-Amiga-Q)**

Use the LOAD Option to load a new adventure, or to switch to a different adventure. (See the section below regarding LOAD and SAVE game for a complete description of how to use the file requester.) As screen colors are a matter of personal choice, feel free to adjust the colors to your preferences with the COLORS menu option. At the end of the game session, PADV will attempt to save the file "TACL-PAdv.Config" to the "s:" directory so that, next time you run PADV, it will begin with your favorite colors, as well as your selections for Graphics, Audio, and Speech.

The ABOUT option will display a window with some information regarding PADV's legal status. The HELP menu option, the Right-Amiga-H combination, or the HELP Key can be used to pop up a window that describes the commands available in any adventure. Note that both the Help and About windows, and any PADV requesters will accept keyboard input. The Help and About windows will accept any keypress as an "OK", and any requesters that ask for a yes or no response will accept "Y" or "N" from the keyboard.

The GAME menu is as follows:

New	**(Right-Amiga-N)**
Load	**(Right-Amiga-L)**
Save	**(Right-Amiga-S)**

Select the NEW option to abandon your current game, and restart the adventure from the beginning. Use the LOAD and SAVE functions to save games in progress and to resume from any previously saved point. PADV is a nice program, and does not erase your saved games when your character dies. This means that, as long as you save your game just before your character dies, you may keep trying to get past a difficult portion of the adventure.

Both LOAD and SAVE games, and LOAD adventure, will pop up a file requester. To look for files in the directory that you were last in (or started the program from), just click in the display window. If you wish to look on a specific drive, just click on the appropriate icon on the right side of the requester. You may not have all of the drives or devices listed, so ignore the ones you don't have. If you have a device that is not listed on a gadget, click in the path selector on the bottom, and type in the device name followed by a colon (":").

If you already know which file you want, just type it in the file selector by first clicking in that box, and then typing the name of the file you wish to load or save. The PARENT gadget will back up one directory, and the ROOT gadget will read the root directory of the current device. When you see the file you want, either double-click on the filename, or single-click and then click the OK gadget. If you decide that you want to stop and go back to what you were doing, click the CANCEL gadget.

The OPTIONS Menu is as follows:

Graphics
Audio
Speech

No keyboard shortcuts are available for this menu. (Options from this menu are generally not used often, so this should not present an inconvenience.) Each item in this menu has a submenu that will have a checkmark in front of the current state of each option.

The GRAPHICS submenu will turn the graphics screen on and off, thus enabling you to see or not see any graphics that may be part of the adventure you are currently playing. The AUDIO submenu turns on and off any sound effects in the adventure. And using the Amiga's built-in speech capabilities, the SPEECH option will read ALL text, including PADV requesters, and Help and About windows. The submenu allows three different speech speeds to be selected. This feature has been included in the hope that it will prove useful for a child learning to read,

for someone who has difficulty seeing the computer screen, or for someone who just wants to show off the Amiga.

CONTROL KEYS

PADV's control keys are simple and straightforward. In addition to the menu shortcuts mentioned above, all the standard characters – "0" thru "9", "a" thru "z" and "A" thru "Z", dash ("-"), underscore ("_"), the Spacebar, the Up and Down cursor control keys, F1, F2, F10, the Backspace, Return, Help, and Escape ("ESC") keys – are accepted. Here's how to use them.

ESC:	Blank the current line of input.
UpArrow:	Recall the last line of input. (Use to redo a previous command.)
DownArrow:	Recall the first line of input.
Help:	Display the Quick Help window.
F1:	Toggle between text and graphics.
F2:	Toggle Image Select window.
F10:	Toggle Debug window (in debug mode only).
Backspace:	Delete the last character typed (as usual).
Return:	Tell PADV you're done typing that line, and to try to do what you've asked.
Other Keys:	Same as usual (i.e., the "A" key will type an "a" or an "A").

Use the F1 key to swap back and forth between the graphics screen and the text screen. The graphics screen will automatically pop to the front after any graphics are drawn or displayed. To return to the game, press F1 without clicking in the graphics window. If you accidentally do click in the graphics window, click about one centimeter to the left of the upper right corner of the screen. The layering gadgets are really there, in their usual places, but are invisible.

The F2 key is used to toggle the Image Select window. When visible, this window will display all objects (and the current room) for which there are images. The first one is usually the room, with any items in the player's inventory that have images appearing next, followed by the objects in the current room (if they have images). You do not have to wait for the adventure to display these; single-clicking on any listed object or room will draw and display the picture for that object. This allows every object to have its own picture, often saving on long explanations. If you do not want to see the objects that have available

pictures, except when they are automatically displayed by the adventure program, simply press the F2 key and the Image Select window will disappear. Note that clicking on the Close gadget will turn off the entire graphics system.

When PADV is run with debug mode, it opens a window to report any errors encountered during the session. Since all errors are also written to a file, you may not wish to see the Error window. The F10 key will toggle this window on and off in debug mode. Otherwise, the key is ignored.

PADV's STANDARD COMMANDS

PADV contains several commands that are standard and should always be available, but most of these can be redefined by the adventure creator, and may function differently. All directions – NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, EAST, WEST, UP, and DOWN, including abbreviations (N, NW, NE, S, SW, SE, E, W, U, D) – are standard, and move you from room to room as described in the rooms' links. (See Section I: MADV Adventure Compiler for complete details on room links.)

The SCORE command will display your current score. The INVENTORY command (abbreviated as "I") will display all objects you are currently carrying. The LOOK command (abbreviated as "L") will display the full room description you were presented with when you first walked into the room.

LOAD or RESTORE will bring up the "LOAD GAME" file requester, to allow you to continue or restart an adventure. SAVE will give you a file requester to save your current position in an adventure so you can continue from that position at a later time. QUIT will perform the obvious, but will first provide you with a last chance to confirm your request.

PADV's DEBUG COMMANDS

Note that the debug commands can only be used when PADV is run with the -E option, and the correct password for the adventure is entered. If the wrong password is entered, or if the -E option is not used, you will be dropped to normal play mode, and none of these special commands will work. Capitalization of the password does not matter.

The JUMP command is one of the most enjoyable commands available in Debug Mode. First type JUMP, a space, and the name of the room that you want to go to, press <Return> and <*KAPOW*> you are THERE. (Sound effects courtesy of DC Comics.) Neato!!! Almost as much fun as moving yourself instantaneously from place to place, is moving

arbitrary objects instantaneously from place to place. Type PLACE, a space, the object's name, another space, and the name of the room to send it to. Press <Return> and it is there. Savor the Power! As an added bonus, simply leave off the room name to send the object to, and instead it will tell you what room that object is currently in.

Using a little finesse, we move to the more subtle commands. First is the EQU command. Use EQU, a space, the name of a variable, another space, and the value to give to that variable. Like the PLACE command, leaving off the final value will instead tell you that variable's current value.

The last Debug Command is SET. Type SET and a space, followed by either a room name or an object name, and the number of the attribute to report on or set. If you want to set the attribute, follow the attribute number with either a "Y" or "N" to set the attribute on or off. Otherwise, the current state of that attribute will be displayed. Be careful! The names of attributes are not saved, and the numbers must be used. The numbers begin at zero with the first attribute defined, and go in the same order as they are defined.

Rooms have the first two attributes already defined automatically, but objects have no predefined attributes. The "zero" attribute for all rooms is the DARK attribute which is predefined for your convenience. The "one" attribute is VISITED, which is set and reset automatically by the PADV system. Compiling the adventure with cross referencing will help you determine the numbers of attributes. See the MADV section for details on how to use these.

Keep in mind that most commands can be overridden by defining a VOCAB block that redefines a command. This is usually not a good idea, but sometimes is quite useful. Also, some commands cannot be overridden.

SENTENCE STRUCTURE

One of the most important parts of an adventure game is the command parser. *T.A.C.L.'s* parser is a solid, top-quality, but not overly complex parser. Generally, the biggest limitation is that *T.A.C.L.* only allows one verb per sentence, and one sentence per turn, unless specifically implemented for specific situations by the adventure writer. This simplifies programming to keep track of numbers of turns taken, and timing different events based on turns taken.

The following sentence structures are standard, and should serve for almost all situations. Other sentence structures can be built in through your program, using vocabulary blocks. See the MADV section on vocabulary blocks for more. Note that articles ("a", "an", "the", and "that") can be placed in front of any noun (or in front of an adjective modifying the noun).

SENTENCE STRUCTURE LISTING

```
<verb> <noun>
ex: GRAB TIMER

<verb> <adjective> <noun>
ex: GRAB the NUCLEAR BOMB

<verb> <noun> <preposition> <noun>
ex: DETONATE the BOMB WITH the TIMER

<verb> <adj> <noun> <preposition> <noun>
ex: STEAL the SECRET DOCUMENT FROM the AGENT

<verb> <adj> <noun> <prep> <adj> <noun>
ex: DECODE SECRET DOCUMENT WITH the DECODER RING

<verb> <prep> <noun>
ex: LOOK UNDER the TRASHCAN

<verb> <prep> <adj> <noun>
ex: SEARCH THROUGH the OAK DESK
```

OFTEN USED SPECIAL CASES: (These must be specifically implemented by the adventure writer.)

```
<verb> <preposition>
ex: LAY DOWN
ex: STAND UP

<verb>
ex: YELL
ex: SLEEP
```

A well written adventure should always include a good supply of witty, or at least appropriate responses to some of the more common short requests. For example, if the player typed, "STAND UP," while their character was already standing, the adventure could reply, "You make a fool of yourself trying to stand up before you discover that you already are."

We hope you enjoy using PADV when playing adventure games written with the *T.A.C.L.* system.

This page for doodles.

SECTION IV

T.A.C.L. - The Language

***T.A.C.L.* OVERVIEW**

The Adventure Construction Language was devised specifically for the implementation of text and graphic adventure games. With *T.A.C.L.*, you do not need to know the nitty-gritty details of HOW the system works. Rather, you can concentrate solely on the creative effort of writing an adventure game.

T.A.C.L. is designed to be as English-like as possible; the more readable an adventure is, the easier it is to write and debug. To further its ease of use, *T.A.C.L.* is completely case-insensitive, meaning that it does not matter how or whether you capitalize words.

Comments can be used in *T.A.C.L.*. Everything past an asterisk ("*") on a line is ignored by MADV. Note that each comment line must have its own asterisk. Comments may be inserted on lines by themselves or following any command.

To protect your secrets, *T.A.C.L.* adventure games are written in a specially encoded format that renders them unreadable. Also, passwords are associated with both the adventure game file and VGED graphic files. Nobody can look at your images or use PADV's debugger on your game without knowledge of these passwords.

There is a limit to the complexity of your adventures, but this limit should not impede your creativity. Per adventure, you may have a maximum of:

14 attributes per room
6 attributes per object
256 images per VGED graphic file

65,534 graphic files (including IFF images)
65,534 rooms
65,535 objects (including NPC's)
65,535 object NAMEs (total)
65,535 object ADJectives (total)
65,535 subroutines
65,535 ACTION blocks (including both VOCAB and object actions)
65,535 variables
4,294,967,295 characters of text

Therefore, there are only two real restrictions: 1. The amount of memory you have to compile. (MADV is pretty efficient!) 2. The amount of disk space you have.

DESCRIPTION NOTATION

In the next subsection, we will describe *T.A.C.L.* formally. To do so, we will use a formal description standard similar to Backus-Naur Form (BNF). Statement names that are to be used directly will be printed in uppercase, while the parameters and arguments to the commands will be printed in-between angle brackets (less-than and greater-than symbols). For example,

```
PLACEOBJ <object_name> <room_name>
```

means that PLACEOBJ is the command, the phrase <object_name> should be replaced with the name of an object in your adventure, and that <room_name> should be replaced with the name of a room in your adventure. The brackets indicate that whatever lies between them should not be taken literally, but can assume the name of any valid identifier of the same class. This is called a VARIABLE. When a choice of classes is available, they are separated by an OR symbol ("|"). EITHER one or the other may be used, but not both. Like the brackets, the OR symbol is not to be typed; it is presented here only to help you understand *T.A.C.L.'s* format.

T.A.C.L. identifier classes:

<text> = any line of text
<word> = any word containing no spaces
<variable_name> = the name of an integer variable (1 to 19 characters)
<file_name> = a standard AmigaDOS filename

<room_name> = the name of a room (1 to 39 chars)
<object_name> = the name of an object or NPC (1 to 19 chars)
<subroutine_name> = the name of a subroutine (1 to 19 chars)
<image_name> = the name of an image from a VGED graphics file (1 to 39 chars)
<attribute_name> = the name of an attribute (1 to 19 chars)
<adjective> = any word describing an object (1 to 19 chars)
<noun> = any noun (1 to 19 chars)
<verb> = any single-word verb
<direction> = N|S|E|W|NE|NW|SE|SW|U|D
<condition> = any of the comparisons used in *T.A.C.L.* "IF" statements
<comparison> = any of the *T.A.C.L.* "IF" variable comparators ("=","<",">", or "#")
<literal> = any literal constant integer such as 11 or -300
<statements> = a sequence of program statements (see below)

Square brackets are used around words or blocks that do not necessarily have to be present.

FORMAL DESCRIPTION OF *T.A.C.L.*

Format for the main Adventure file <word>.ADV :

```
ADVENTURE <text> * a whole line

PASSWORD <word>  * a single word

[
VAR  * this block is optional
<variable_name>   [<literal>] * pre-init to 0, else init to
<literal>
<variable_name>  [<literal>] * any number of variables can be
here
]

[
ENDVAR
]

[
IMAGE  * this block is optional
(<file_name>  <word>) | * this one is used for VGED graphics
files
(<file_name>) * this one is used for IFF images
[(<file_name>  <word>) | * any number of files can be here
```



```
(<file_name>)

]

ROOM
<file_name>
[<file_name>    * any number of room files can be here

]
ENDROOM

[
OBJECT * this block is optional
<file_name>
[<file_name>    * any number of object files can be here

]
ENDOBJECT
]

[
SUB  * this block is optional
<file_name>
[<file_name>    * any number of subroutine files can be here

]
ENDSUB
]

[
VOCAB  * this block is optional
<file_name>
[<file_name>    * any number of vocab files can be here

]
ENDVOCAB
]

INITROOM <room_name> * the room the player starts in

ENDADVENTURE
```

Format for Room files:

```
ROOM <room_name>

[
ATTRIB  * this block is optional
<attribute_name>  [Y|N]  * defaults to N, else Y or N, whichever
here
[<attribute_name>  [Y|N]  * up to 14 total attributes per room

]
ENDATTRIB
]

[
DEFAULT * this block is optional
<direction> <room_name>  *initial room links
[<direction> <room_name>

]
ENDDEFAULT
]

CODE  * executed when player is here
 statements>
ENDCODE

ENDROOM  * put as many rooms per file as you want: follow this
 * with another ROOM statement if you want.
```

Format for Object files:

```
(OBJECT|NPC) <object_name>  * either an OBJECT or an NPC

NAME <noun> [, <noun>, ... ]  * as many aliases as you need (on
same line)

 * ADJ is optional
ADJ <adjective> [, <adjective>, ...] * as many as you need (on
same line)

[
ATTRIB  * this block is optional
 <attribute_name>  [Y|N]  * defaults to N, else Y or N, whichever
here
[<attribute_name>  [Y|N]  * up to 16 total attributes per object

]
ENDATTRIB
]
```



```
 INITROOM <room_name>|PLAYER * PLAYER means it starts out in
 inventory

 CODE
 <statements>
 ENDCODE

 [
   * ACTIONS are optional
 ACTION <verb> [, <verb>, ...] * as many verbs as you need (on
 same line)
 <statements>
 ENDACT
 * as many actions as you want for this obj
 ]

 ENDOBJECT|ENDNPC * put as many objects per file as you want:
 follow this
  * with another OBJECT or NPC statement if you want.
```

Format for Subroutine files:

```
SUBROUTINE <subroutine_name>
 <statements>
ENDSUB

[
SUBROUTINE <subroutine_name> * put as many subroutines as you
want
 <statements>  * in each file
ENDSUB

]
```

Format for Vocab files:

```
VOCAB

ACTION <text>  * any line of text the player can type as a command
<statements>
ENDACT

[
```



```
* as many actions as you want per file
ACTION <text>  * any line of text the player can type as a command
<statements>
ENDACT
]

ENDVOCAB
```

Format for *T.A.C.L.* statements (one per line, listed in alphabetical order):

```
ADD <variable_name> <variable_name|literal>

CALL <subroutine_name>

DIE

DIRECTIONS <room_name> <direction> [<direction> ...]

DIV <variable_name> <variable_name|literal>

DROP <object_name>

ELSE

ELSIF <condition> [THEN]
(see IF for conditions)

ENDIF

EQU <variable_name> <variable_name|literal>

GO <room_name>

GRAB <object_name>

IF <condition> [THEN]
where <condition> can be one of these:
PLAYER IN <room_name>
PLAYER HAS <object_name>
<object_name|room_name> IS  <attribute_name>
<object_name|room_name> NOT <attribute_name>
<object_name> IN  <room_name>
THISROOM IS VISITED
THISROOM IS DARK
<variable_name> <comparison> <variable_name|literal>
```



```
PREPOSITION IS <preposition>
USING <object_name>

IMAGE <object_name|room_name> <image_name>

LINK <room_name> <direction> <room_name>

MOVE <direction>

MOVEOBJ <object_name> <direction>

MULT <variable_name> <variable_name|literal>

NOTE <variable_name|literal> <variable_name|literal>

PLACEOBJ <object_name> <room_name>

RANDOM <variable_name> <variable_name|literal>

SCORE <literal>

SET <object_name|room_name> <attribute_name>

SHOW <image_name>

SUB <variable_name> <variable_name|literal>

T <text>

UNSET <object_name|room_name> <attribute_name>

WIN
```

PROGRAM EXECUTION ORDER

T.A.C.L. functions by selectively running the programs associated with the objects and rooms of your adventure. In each situation below, the listed programs are executed in the shown order.

When the player moves or is moved to a new room (or the game is started), or when the player does a LOOK command:

1. That room's program.

Before each turn:

1. The programs for all NPC's in the adventure.

2. The programs for all objects that are in the same room as the player.

When the player does an INVENTORY command:

1. The programs of all objects that are in the player's inventory.

When the player performs some action on an object:

1. The program for the corresponding ACTION block for that object.

When the player does one of the commands listed in a VOCAB ACTION block:

1. The program for that ACTION block.

TIPS & TRICKS

There are many little tricks and techniques you can employ to make writing adventure games both easier and more successful. The sample adventure games enclosed utilize many of these to accomplish their magic. The following is a list of several tips and tricks designed to help you write a more successful and complete adventure.

1. THE WRITTEN PLAN. Whenever designing a complex adventure, write a detailed explanation of the adventure's major points. You should also draw up some kind of map so you can be sure, for example, that rooms do not overlap each other. This lets you work out all the quirks and design flaws before erroneous or conflicting code is written.

2. PLAYABILITY. An adventure game should be a mental challenge to the player that he or she can solve in a logical, clear manner (with perhaps a little luck). It should NOT be winnable only with an act of God. Also, a game is often more enjoyable if it contains humor. A classic example of the effective use of humor is Infocom's "Hitchhiker's Guide to the Galaxy".

3. OVERRIDING SYSTEM COMMANDS. You can override most of the system commands, including the direction commands and the play commands. The only commands that cannot be completely overridden with your own VOCAB ACTIONs are the GAME SAVE/LOAD commands (because they work from the menu, too).

4. THE OBJECT "SELF". When you are using NPC's in a game, it may be necessary at some point to check if the NPC is in the same room as the player. Well, the object has an Object Position Variable, but what about the player? A trick to getting around this is to declare an object called "Self", which is in the player's inventory at all times. You can even define some cute ACTIONs on the object so that the player can look at himself, kill himself, etc.. But the object's main function is to check the player's current room. The system variable SelfPOS will be automatically defined to the object, but its value will be -1 because it is in the player's inventory. To get the player's current position, you can DROP the object (temporarily), copy the value of SelfPOS to another variable, and then – this is important – GRAB the object Self again. This can be done in a subroutine to save time. The other variable now contains the number of the room the player is currently in. Its value can be compared with the NPC's position variable.

5. CAPITALIZATION. As you know, *T.A.C.L.* is case-insensitive, but it is wise to adopt some scheme of capitalization and STICK WITH IT throughout the project. One that works very well is to write all the system words in all capitals, and then capitalize appropriate letters in your own identifiers. This method allows you to scan your program and quickly tell the difference between your words and *T.A.C.L.'s* words.

6. INDENTATION. It is a good habit to indent one or two columns each time you enter another level of IF statements and when you first enter a new block. This also makes the program more readable. Indenting too much can make your program difficult to read in that text lines will run off the edge of the screen.

7. COMMENTS. Another good practice is to comment the more complex sections of your program, or at any point that its function is not blatantly obvious. Maybe not at every line, but a comment here and there will clear things up nicely. You needn't worry about security because your comments DO NOT end up in the game file; MADV ignores them.

8. DYNAMIC ROOM LINKS. You can add a lot of spice to your adventure by using dynamic room links. For instance, you can change the legal directions and links in a room based on some condition. An example can be found in Section I in the Spy Adventure; hidden passages in the basements appear under certain situations.

9. LIMITING INTERACTION. Non-Player Characters (NPC's) can be an essential part of your adventure game. Each character should be designed to have some limited interaction with the player. That is, you cannot possibly account for everything the player tries to do to the NPC, so what you DO implement should either be what the player is most LIKELY to do, or it should be specific to the NPC's function. For an example, refer to the Agent's program in Section I's Spy Adventure.

10. ROOM SWITCHING. As in the Spy Adventure in Section I, you can simulate the existence of a large number of rooms with just a few. This is also done in the Space adventure on the asteroid's surface; over 2,500 apparent rooms are simulated by just 11 rooms. One drawback to this technique is that objects dropped in these rooms must be handled appropriately, or they will show up in all of the simulated rooms.

This page for doodles.

APPENDIX A

MADV Compiler Errors

The following is a list of compiler errors, in numerical order, and a brief explanation of their causes.

1. Undefined room: Reference was made to a room name that was not defined. Check spelling.

2. Undefined object: Reference was made to an object name that was not defined. Check spelling.

3. Undefined comparator: Programmer tried to use an invalid comparison in an IF statement.

4. Undefined attribute: Reference was made to an attribute that was not defined. Check spelling.

5. Undefined system attribute: An illegal reference was made to an attribute that is not compatible with THISROOM. Only DARK and VISITED can be used with THISROOM.

6. Illegal comparison: Programmer tried to use a comparator in an IF statement other than "=", ">", "<", or "#", which are the only legal variable comparators.

7. Comparator(s) "=", "#", "<", or ">" expected: No legal comparison was found in the IF statement.

8. Undefined variable: Reference was made to a variable that was not defined. Check spelling.

9. Bad IF block: MADV did not recognize the attempted comparison in the IF block, perhaps because an identifier was misspelled.

10. Undefined object or room: The identifier following a SET or UNSET instruction was not a valid object or room. Check spelling.

11. Unknown direction: Reference was made to an invalid direction abbreviation.

12. Illegal Statement: A statement other than one of the 23 legal statements was used. Check spelling or format.

13. Object name already used: An attempt was made to declare an object with the same system name as an existing object.

14. Object name already used as room name: An attempt was made to declare an object with the same name as an existing room.

15. "OBJECT" or "NPC" expected: The beginning of an object definition was not one of the legal choices.

16. Attribute declared twice: An attribute with the same name was already declared for this object or room. Rename or respell.

17. Too many attributes: In a ROOM, an attempt was made to declare more than 14 attributes. In an OBJECT or NPC, an attempt was made to declare more than 16 attributes.

18. 'INITROOM' expected: MADV found something other than an INITROOM statement where it expected one.

19. CODE block expected: MADV found something other than the expected CODE block.

20. Verb list expected: A line containing comma-separated verbs or commands was expected, but not found.

21. "ACTION", "ENDOBJ", or "ENDNPC" expected: The object definition was not terminated properly. Check spelling or format.

22. Room declared twice: This room name has already been used by another room.

23. Cannot redefine room: Something happened to the ROOM file between compile pass 1 and pass 2. File may have been deleted by someone. A very unlikely error.

24. 'ENDROOM' expected: The room definition was not properly terminated. Check spelling and format.

25. Cannot open VOCAB file: The file could not be opened. Usually a DOS error.

26. "VOCAB" expected: The VOCAB file must start with this word.

27. "ACTION" or "ENDVOCAB" expected: The VOCAB file has a bad format.

28. "ADVENTURE" expected: The ADV file must start with ADVENTURE.

29. "PASSWORD" expected: MADV expected to find this word next.

30. Password string expected: A protection password MUST be supplied.

31. Illegal password: Try another. The password provided was insufficient to protect the game. Another must be used.

32. Variable already declared: This variable name has already been used.

33. Cannot open file: The filename supplied was probably misspelled.

34. "ENDADVENTURE" expected: The ADV file was not properly terminated.

35. Cannot open master file: The main ADV file could not be accessed.

36. OUT OF MEMORY: The system ran out of memory and could not finish compiling the game.

37. Unknown preposition: Programmer referred to an unknown preposition in an IF PREP statement, or misspelled a valid one.

38. THISROOM attributes MUST be system attributes "DARK" or "VISITED": Programmer tried to SET or UNSET an illegal THIS-ROOM attribute.

39. NAME block expected: In an OBJECT or NPC definition, the NAME block may NOT be omitted.

40. Block not ended properly: MADV encountered a statement that was out of sequence. It was found in the middle of another block.

41. At least one name expected: You must provide at least one name in the NAME block of an object or NPC.

42. Text line is too long: A Text statement ran for more than seventy-nine characters. Truncate it.

43. Subroutine name already used: A subroutine name was used for more than one subroutine.

44. "SUB" or "SUBROUTINE" expected: MADV expected to see one of these words.

45. Unable to redefine subroutine: An unlikely error, MADV could not find the subroutine file again.

46. Undefined subroutine: A CALL command used an invalid subroutine name.

47. Cannot open .WRD file: MADV could not create a WORD file for the adventure. This is probably a DOS error.

48. Object or room name expected: MADV expected to see the name of a declared object or room.

49. Image not found in graphic file: An IMAGE or SHOW command used an image name that does not exist in the graphic files declared in the main .ADV file.

50. Password does not match: The password supplied for a graphic file does not match the graphic file's password.

51. Graphic file already declared: The same DOS name was used for more than one graphic file.

52. Note value out of range: A NOTE command's pitch value was out of the legal range of 1 to 1000.

53. Note duration out of range: A NOTE command's duration was out of the legal range of 1 to 500.

54. Line too long: Line cannot exceed seventy-nine characters in length.

1000, 1001, ... : These are system errors. You should not see them.

This page for doodles.

APPENDIX B

PADV Debugger Errors

1. Division by zero: A DIV statement tried to divide a variable's value by zero, an impossible task.

2. Strange comparison yields TRUE: An IF statement tried to compare two values using a strange or logically impossible comparison.

3. Object not in player's inventory: A DROP statement cannot drop an object that is not in the player's inventory.

4. Requested direction has no room linked to it: A DIRECTION instruction was called and there was likely no LINK instruction following it. Therefore, the direction led to the "edge of the world". This error occurs only in the MOVE instruction.

5. No exit in requested direction: An attempt was made to "walk through the wall", or go in a direction that has no exit.

6. SYSTEM ERROR in variable formatting: If you receive this error, shoot Kevin Kelm.

7. Object is still in player's inventory: An attempt was made to MOVEOBJ an object that the player is carrying.

8. Object cannot move in that direction in the current room: An attempt was made to MOVEOBJ an object that is in a room which does not possess that object's requested direction as a legal exit.

9. ELSE not found in IF block: Like it says. PADV found an ELSE statement that wasn't where it belonged.

10. Illegal statement: PADV encountered a statement it didn't recognize. Either the game file is corrupt, or there is an incompatibility between the version of PADV and the version of the MADV compiler that generated the game. The latter is unlikely.

11. Linked direction not enabled: An attempt was made to form a direction link in a room that does not currently have that direction as an exit.

12. Position variable set to invalid room number: An attempt was made to set an object's ObjectPOS variable to a value that would put it in a nonexistent room.

13. Note value out of range: A NOTE command's pitch value was out of the legal range of 1 to 1000.

14. Note duration out of range: A NOTE command's duration was out of the legal range of 1 to 500.

APPENDIX C

Note Value Table

Using the values listed in the table below, you can make *T.A.C.L.* generate simple music for your adventures. This might include a victory song, or a funeral dirge at the close of the game, or anything else you can imagine.

	OCTAVES		
NOTE	**LOW**	**MIDDLE**	**HIGH**
A	108	616	870
A#	164	644	884
B	218	671	898
C	268	696	910
C#	316	720	922
D	362	743	934
D#	404	764	944
E	446	785	954
F	484	804	964
F#	520	822	973
G	554	839	981
G#	586	855	989

This page for doodles.

APPENDIX D

How To Use ED

If you do not have much experience with programming or the CLI, this section may be of use to you. We will briefly explain the basics of using ED, the simple screen editor that comes with your Workbench disk.

To get started, ED is used from the CLI by typing:

```
ED <filename>
```

where <filename> is the name of the file you want to edit or create. ED will then open its own window in the Workbench screen. If the file already exists, it will be loaded. ("At" symbols ("@") will be displayed to show its progress loading.)

When ED is ready, the screen will blank and you can begin typing. You can use the arrow keys on your keyboard to move the cursor around.

ED uses the escape key ("ESC") at the top left side of your keyboard to enter what is called COMMAND MODE. In command mode, an asterisk ("*") appears in the lower left side of the window. You can then type next to it any of the commands that make ED do special things. Some of the more useful of these are listed on the following page.

ESC	Result
Q	Exits WITHOUT SAVING that which you have typed in this session.
X	Exits and SAVES whatever you have typed in this session.
T	Moves to the TOP of the file.
B	Moves to the BOTTOM of the file.
Mnnn	MOVES to the line number immediately following the M.
F/xx/	FINDS the next occurrence of the text between the forward slashes, starting from the cursor's current position.
BF/xx/	BACKWARD FINDS the last occurrence of the text from the cursor's current position.
E/xx/yy/	EXCHANGES the next occurrence of the first text (xx) and replaces it with the second text (yy).
U	UNDO typing on the current line (only if the cursor has not yet been moved from the line).

ED also uses CONTROL codes as special commands. These are accessed by pressing down the control ("CTRL") key on the left side of your keyboard while SIMULTANEOUSLY pressing one of the keys listed below.

CTRL	Result
A	Inserts a new line.
U	Scrolls the cursor UP half a screen.
D	Scrolls the cursor DOWN half a screen.
B	Deletes the line the cursor is currently on.
Y	Deletes the remainder of the line the cursor is currently on (everything to the right of the cursor).
G	Performs the last ESCAPE command again.
]	("right bracket") Alternately moves the cursor to the beginning and end of the current line.

Glossary

Useful Words Defined

AmigaDOS: The standard Disk Operating System built into your Amiga. Used by the user through the CLI and Workbench interfaces.

Compiler Errors: Syntactic problems (formed/written incorrectly) in your adventure's code that are detected at compile time.

Cross-Reference: Special listing generated by MADV that will selectively show all variables, rooms, objects, code blocks, and/or graphic files used. Can also include form-feeds between sections to ease readability when printed.

Debug Commands: Special commands available only in Debug Mode.

Debug Mode: A special mode in PADV that reports run-time errors to the player, and allows the player to directly manipulate objects in the adventure. Unauthorized players may not enter this mode without the adventure's password.

Editor: The program used to enter source code into the computer and save it on disk. If you do not have a special editor of your own, use AmigaDOS's ED command. (See Appendix D for quick instructions on how to use it.)

Gadgets: Standard Amiga "buttons" for layering windows or screens, closing and resizing windows, and dragging windows around the screen.

IFF/ILBM: Standard Amiga graphic files. (IFF is the Interchange File Format, and ILBM stands for InterLeaved Bit Map, a standard way of saving image files.) May be read by PADV, and displayed as part of your adventure. Use DeluxePaint from Electronic Arts, or any other paint program to create images for your adventure. IFF/ILBM images generally take considerably more space than VGED pictures.

Images: Either IFF (ILBM or HAM) or VGED image files that can be used in your adventures.

Jungle: A small sample adventure that demonstrates *T.A.C.L.'s* graphic capabilities. The entire adventure was created in less than a week.

Keyboard Shortcut: A special key combination that uses the Right-Amiga-Key and the key listed next to a menu option to accomplish a function without having to select it from the menu.

Keywords: Special words in your source code which have specific meanings that do not change, ever. These are words like IF, MOVE, DIRECTIONS, etc. They cannot be used as variable names, room names, etc.

MADV: Stands for "Make ADVenture". This is the adventure player, and it is responsible for reading all source code for an adventure, checking for syntactic errors, and generating the encoded adventure.GAM file for PADV to read. This program is NOT freely redistributable.

Menus: Allow easy access to many features in PADV and VGED. Use the right mouse button to access these menus.

OBJECTS: Term used to describe all objects, creatures, people, and anything else that exists in the adventure that the player can interact with. Includes NPC's.

PADV: Stands for "Play ADVenture". This is the adventure player, and it is responsible for actually playing the adventures created with MADV. It executes the adventure code, accepts input from the users, displays pictures, and plays sounds. This program is freely redistributable.

Rescue: One of the sample adventures included with the *T.A.C.L.* package.

ROOMS: Every geographical location in *T.A.C.L.*. These define WHERE the player is, and WHERE he is able to go. One of the main concepts of *T.A.C.L.*.

Run-Time Errors: Problems with the logic behind how your adventure works. These cannot be detected by the compiler, and can only be found with Debug mode. For example, allowing the direction North as a legal direction in a room, but forgetting to link that room, to the North, to another room would be a run-time error.

Source Code: The collection of text files that describe your adventure using a special format known as The Adventure Construction Language.

Spy: A sample adventure that the manual uses to demonstrate the many features of *T.A.C.L.*.

Subroutines: Special sections of code that can be used multiple times from different sections of your program.

***T.A.C.L.*:** The Adventure Construction Language, *T.A.C.L.*, is the composite system that consists of MADV, PADV, and VGED. It is also the name of the language syntax used. Note that *T.A.C.L.*: the language, MADV, and VGED are copyrighted software, while VGED is also copyrighted (and NOT Public Domain), but may be freely copied and distributed with your adventures, or posted to any BBS or Information Service.

Text Styles: The different text styles on the Amiga consist of Bold, Underline, and Italics. To turn off ALL styles, select Text Style: "Plain".

Title Bar: The top line of the screen that contains various information independent of the rest of the screen.

Variables: Special names that contain numbers representing different values (usually specified by their name). For example, Score, Moves, and Dollars could all be variables representing the player's score, how many moves he has made, and how much money he has.

VGED: The "Vector Graphics EDitor". A special paint program that creates images that only PADV can display. VGED creates an image by recounting the steps taken to draw the image whenever it needs to be displayed. PADV is the only other program capable of displaying VGED images. However, VGED's images are almost always MUCH smaller than a "normal" Amiga IFF picture file.

Vocabulary: Special statements that allow you to include your own terms in the understandable vocabulary of your adventure. Use this when trying to get your adventure to recognize a phrase that PADV does not currently acknowledge.

Produced by:
Micro Momentum, Inc.
P.O. Box 372
Washington Depot, CT 06794
(800) 448-7421