STAC

THE ATARI ST ADVENTURE CREATOR

THE ST ADVENTURE CREATOR

USER MANUAL

by Sean T. Ellis

29th April 1988

CHAPTER ONE	*Introduction*	Page 1
CHAPTER TWO	*Writing adventures*	Page 3
CHAPTER THREE	*Use of Conditions*	Page 8
CHAPTER FOUR	*Graphics*	Page 27
CHAPTER FIVE	*Music in messages*	Page 32
CHAPTER SIX	*Our small adventure*	Page 34
CHAPTER SEVEN	*Disk and printer menus*	Page 42
CHAPTER EIGHT	*The font editor*	Page 45
CHAPTER NINE	*Special conditions and commands*	Page 48
APPENDIX A	*Keys and control characters*	Page 52
APPENDIX B	*Glossary*	Page 54
APPENDIX C	*Contents of QSTART file*	Page 58
APPENDIX D	*Character key sequences*	Page 62
APPENDIX E	*Conditional words*	Page 64
APPENDIX F	*Adventure Flowchart*	Page 70
APPENDIX G	*Handy Reference Sheet*	Page 71
INDEX		Page 72

CHAPTER ONE

INTRODUCTION

Welcome to the world of the ST Adventure Creator. I hope you will have fun in creating and playing your own adventure games, and even selling them if you wish (there is no fee for this, just put a credit in the program).

Sean

EQUIPMENT

To use this program, you will need an Atari ST computer (520, 1040 or Mega ST), at least one disk drive, and a colour monitor or a television set. In addition, a printer is useful, but not essential.

YOU CAN NOT USE THIS PROGRAM WITH A MONOCHROME MONITOR.

CONTENTS OF THE PACKAGE

Inside the box, you should find two disks (the program disk and the demonstration disk), this manual, and a registration card. Please complete and return this to become a registered user.

LOADING THE ST ADVENTURE CREATOR

To load the **ST Adventure Creator** (which will henceforth be referred to as the **STAC**), place the program disk in drive A, which is the internal disk drive if you are using a 520STF, 520STFM, 1040ST or Mega ST, and switch the machine on. The screen will show a small window with a program icon called *STAC.PRG*. Move the mouse pointer over this icon and quickly click the left hand mouse button twice.

THE PROGRAM DISK

This contains the **STAC** program itself, along with a "Quickstart" file which contains many of the common words, actions, etc. used in most adventures, a small adventure file for you to examine and change, and a set of fonts already set up for you to use.

This disk is NOT public domain. If you experience problems with this disk, please send it (just the disk, not the entire packaging) back to us at the address printed on the back of the box for a replacement. This does not affect your statutory rights.

THE DEMONSTRATION DISK

The demonstration disk contains, as its name suggests, some demonstrations of what you can do with the STAC. Full instructions can be obtained by booting up from that disk, double clicking on READ.ME and selecting "Show". The instructions will then appear on the screen.

The disk contains a demonstration runnable adventure (called *SHYMER*) produced by Sandra Sharkey using the STAC, a number of compressed pictures produced by the STAC, and a slideshow to display them all.

This disk *is* Public Domain and is not copy protected - please make copies for your friends !

ACKNOWLEDGEMENTS

I would like to use this space to thank some of the people who have helped in the development of this program.

Thanks to:

Sandra Sharkey for the Shymer adventure,
Pat Winstanley for extensive playtesting,
David Wyatt for the demo pictures,
Dicon Peake for the graphics in Shymer,
Paul for graphic conversion,
Steve 'The Bug' for lively conversation,
Ian Andrew for constructive criticism,
Mike Griffin for destructive testing,
Deborah Stannard for not minding,
Jon Clark, Jeff Maude, John Maude,
and of course you for buying it !

DEGAS is a trade mark of Batteries Included
NEOCHROME is a trade mark of Atari Inc.

ABOUT THE AUTHOR

Sean Ellis is a 21 year old graduate in Cybernetics and Computer Science from the University of Reading. His previous work includes the *Graphic Adventure Creator*, now the standard adventure writer on many 8-bit micros. He is not married, has no children, and lives in Reading with a lady Archaeologist.

The STAC was originally conceived as an extension of *GAC*, but has grown over the 12 months of design and development invested in it into a separate system in its own right. It was developed on a Mega ST 2 in 68000 Assembly Language using *Fast Asm*, and with the help of lots of coffee, music by *Rush* and *Tangerine Dream*, a shelf full of *Larry Niven* and *David Brin* books and a battered copy of *The Hitch-Hikers Guide to the Galaxy* for company.

STAC is dedicated to Deborah for her love and support.

CHAPTER TWO
WRITING ADVENTURES

Having loaded the STAC, you are now ready to write an adventure!

Well, that's not strictly true. Adventure writing needs a bit of careful thought first. However, for a bit of fun, let's write an "adventure" in under 30 seconds ! Load the STAC, and then do the following :

```
Start stopwatch
Press D
Press L
Point the mouse at "QSTART.ADV", and click the left
     hand mouse button.
Press Enter ( or Return )
Press Esc
Press R
Press Insert
Press Enter
Press Enter
Type "A cave", Enter
Type "You are in a large cave", Enter
Press Enter
Press Enter
Press Esc
Press Enter
Stop stopwatch
```

Now try a few commands. Ok, so it's not very interesting or challenging, but look on the bright side - it can only get better !

Seriously, though, writing an adventure does require a bit of thought beforehand. Speaking personally, I cannot just sit down and write an adventure - there are a few things to work out first.

You will have to decide what the purpose of the adventure is, to start with. The traditional rescue-the-beautiful-princess-and-get-as-much-gold-as-possible-then-kill-all-the-monsters type adventure is one (and a bit old hat it is, too...), another might be to tackle the robot defenses to deactivate a large automatic enemy weapons installation (better), or even to prevent the domination of Earth by a race of lust-crazed females (seems familiar from somewhere...) !

Once the main story has been decided on, the adventurer will need a world in which to play the adventure. The world (or *adventure universe*) is arranged as a series of distinct locations (*rooms*) which you can travel between, some of which may contain objects, or require the solution of a puzzle to get out of. Where you are is communicated by a description of the place you are in, along with a list of any objects which are there.

The rooms do not need to be indoors - it is just a convenient way of splitting up the adventure universe.

The easiest way to set out an adventure universe is, I find, to draw a map showing the positions of each room, the connections between each, and the objects which are in each one.

Objects have several characteristics. It is quite useful to know what the object is, so the *object description* tells the player that information. The objects will start the adventure in particular rooms, although they may move around later. Thus the *start room* for each object should be recorded. Also, the *weight* of each object is important. You may be able to carry 100 coins, but only one gold bar, so a gold bar will be 100 times as heavy as a coin.

With the **STAC**, it is a good idea to give all the rooms and objects their own specific numbers at this stage. It will be more difficult to add them later on, when we actually need them.

Then, another thing you will need to specify is the *vocabulary* needed. The player will be communicating with the adventure via his typed commands, which must be interpreted by the adventure and used as a basis for action. It would be impractical to include a complete English dictionary, so only the words you actually need are given. In general, the larger the vocabulary, the more "friendly" the adventure needs to be. For example, you might have a lamp which needs to be lit. If you have to type "light lamp", it would be nice if you could also type:

```
"light torch"        "torch on"        "lamp on"
"turn lamp on"       "switch on lamp"  etc.
```

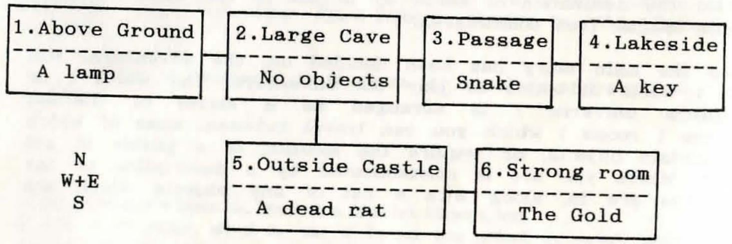
as well.

These loosely define the adventure universe, as set up initially.

Let us go through the construction of a very short adventure. We will start by setting up the adventure universe here, then conclude with the workings in Chapter Five, after they have been introduced.

Story: Get the gold from the castle strong room and bring it to the entrance. All right, so it's not very original, but then this is a demonstration !

Here is a map of the adventure :



OUR SMALL ADVENTURE

It may seem, at first sight, that the five objects are the lamp, the rat, the gold, the key, and the snake. However, there are a couple of nice tricks to be had here.

Since the player will not be allowed to pick up the snake (it will bite him and he will die), it cannot be moved from that room, and can thus be included as a permanent fixture in the room descriptions. Only things that do not move about or otherwise change should be entered with the room descriptions. We can still include a way of examining the snake to make the game more friendly.

However, we will want to light the lamp. There are many ways of doing this, but the one I find easiest is to have two distinct objects - a lamp and a lit lamp. When the lamp is lit, swap the unlit lamp for the lit one, and when it is turned off, do the opposite. The lit lamp can be started in room 0, which is not on the map. In fact, it is a sort of limbo room, where uncreated objects are put, and where destroyed objects go. Very useful, especially for magic ! (None of that in my adventure please...)

So the five objects are: a lamp, a rat, a key, a gold bar, and a lit lamp. Here are the complete descriptions of each:

No.	Description	Weight	Start room
1.	A lamp	10	1
2.	A dead rat	10	5
3.	A key	1	4
4.	A bar of gold	100	6
5.	A lit lamp	10	0

In addition, it might be a good idea to add a longer description of each one, for use when the player examines the object more closely.

Since rooms 2 and 3 are underground, we can make these dark, so that you will need the lamp to navigate them. Let's be generous and allow the player three moves in the dark before being eaten by giant horrible nasty monsters of some form. Being in the dark is signalled by using a "marker", something which is either on or off. Setting a particular marker on could indicate that it is dark, and setting a different one could mean that you have a lamp with you.

Using **STAC**, the rooms are connected together in two different ways. The first way is by connections, which are entered with the room descriptions. These are taken no matter what happens, and merely move the player to another room without affecting anything else. They can only be used sparingly in this adventure.

They cannot be used between rooms 1 and 2, 3 and 4, or 2 and 5 since these moves must be accompanied by a change from dark to light or vice versa. They cannot be used between 5 and 6 since the strong room must be unlocked first, but on the return journey from 6 to 5 they can be used since the door must already be open for you to be in room 6 anyway. Therefore the only place they can be used is between room 2 and room 3, which is always possible, and changes nothing, and between 6 and 5.

This over with, we are now able to use the **STAC** program at last. With it on the disk is a previously saved adventure file containing all the common vocabulary, etc. which is used in almost every adventure.

Having loaded the **STAC**, you will be confronted with a screen full of options. This is the *Main Menu*. A menu is a list of choices from which you can select one thing. We want to load the quick start file, which is a disk operation. Option "D" on the main menu is labelled "Disk menu". Press key *D*, and a new menu will appear. Among the options on this is *L - load adventure data*, which is the one we want. Press *L*.

In the middle of the screen you will see a thing known as a *file selector*. Using this you can select any file on a disk. In the box in the middle there should be a list of file names, including "QSTART.ADV". If you move the mouse pointer over this name, the press the left hand mouse button, (this is known as *clicking on* something) you will see the name at the bottom of the selector change to "QSTART.ADV". Then click on the "OK" box. The disk drive should whirr briefly, loading the file as requested. Press the *Esc* key to get back to the main menu. There is now a set of data in memory including all the most frequently used vocabulary.

To enter the room descriptions, press *R*. A screen will be displayed with a title at the top "Edit rooms", and a mini menu at the bottom. At the moment, there are no rooms stored in memory, so the centre part of the screen is blank. To insert a new room press *Insert* (on the right, near the arrow keys). You will be asked which room number to add. If the displayed number is not the one you want, delete it using the *Delete* key (several times if necessary), and type in the number you want. Then press *Return* or *Enter*. Enter room 1 to start with.

The screen should now look like a form, with titles and blank spaces to be filled in. At first, the "Connections" title will be displayed in white-on-black (*inverse*), showing which bit of data you are currently expected to provide. Since room 1 has no connections, there is no need to type anything here. Just press *Return* to go on to the next line, which is the short room description. For room 1 this is "Above Ground". Type that in, without the enclosing "quote marks", and press *Return* to go on to the next line, the long description.

This is the part where you can become poetic ! A typical long description might run:

You are standing outside a large cave entrance, which runs into the cliff face to your east.

which gives you the relevant information without submerging it in reams of irrelevancy. It could be viewed as a bit terse, however, so as an alternative :

You are standing in the warm sunshine which peeks over a majestic cliff face to your east, in which there is an archway of stygian gloom, which appears to be a cave entrance. You shiver as you look at it, reminded of the day you fought the venomous cave-creatures armed only with your elven sword, stinger.

It is really a matter of personal preference whether you prefer the first or second description. I prefer the first - the second contains much that is irrelevant. Meanwhile, back on the screen, you have typed in the first description, and pressed *Return*.

If you make any mistakes whilst typing, the *Backspace* key will delete the character to the left of the cursor, and the *Delete* key will delete the character under the cursor. You can move the cursor about using the arrow keys, and anything you type will be inserted, shuffling everything after it along.

You are now asked to supply an associated picture number. This refers to the picture that will be drawn when you look at this room. Since we haven't done any pictures yet, press *Delete*, and change this to 0. (From now on, I will assume you press *Return* at the end of each line).

You will now be asked whether this is all right. If it is, press key *Y* (for yes), if not, press *N*, which will take you back to the top of the screen, and give you another chance to edit the information you typed in. The third option, *Q*, abandons everything you have just typed. Don't press this one !

If all is well, you should now be back at the screen with the title at the top and mini menu at the bottom. In the middle, however, will be a white-on-black line saying "1 Above Ground". This is the one room you have typed in so far.

Use the *Insert* key to insert the descriptions of the other five rooms, in the same way as you did with room 1.

Room two has a connection associated with it - you go east to room three. Therefore, when you are asked for connections, type:

```
east 3
```

then Return. This means that if you type "east", you will move to room three. Make sure you have a space between the word and the number. Whole lists can be entered, with spaces between each item. Here is an example of a room with connections to the north, south and west :

```
north 4 south 9 west 2
```

Type in the rest of the descriptions, etc. After doing this, you will have 6 numbers and short room descriptions in the middle of the screen. You can scroll up and down this list using the up and down arrow keys by one item at a time. Pressing *Shift* at the same time will scroll by 16 items at a time. To delete the highlighted item, press the key marked *F1*, which is far enough away from the main keyboard that it should not be pressed accidentally. To have another go at typing in the required information (*editing* an item), press the space bar. The *Esc* key returns you to the main menu from here, as it does from most places in the **STAC**.

Entering the objects is very much the same, except that the form will ask you to supply a short description, a long description, the weight of the object, and the room it is in at the start. The **STAC** is designed so that the procedure for entering data is as similar as possible no matter what it is.

CHAPTER THREE

CONDITIONS

USE OF THE CONDITIONS

The conditions are the most powerful and flexible part of the **STAC** system. They provide the guiding intelligence for the game, by matching the player's commands and taking appropriate actions.

There are four types of condition in the **STAC**: low priority, high priority, local, and special. Confused ? Well, it's not as bad as it sounds...

Low priority conditions are executed after the user types in his command. They are usually used for interpreting that command and taking action based on it, and are executed no matter what room you are in. Typical low-priority conditions are those for manipulating objects, and controlling the "feel" of the adventure with save and load options, quit, and examining things.

High priority conditions are executed before the player enters his command, and are thus beyond his control. They are usually used for signalling dangerous events, and finally killing the player off... or letting him win ! An example would be killing the player off after 3 moves in the dark, whatever the moves were.

Local conditions cover situations which only occur in one room. Typing "jump" may do little on a level plain, but next to a cliff this could quite easily be fatal. They are also used for moving between places when the connections cannot cope.

Special conditions are called at any time. They take the form of short lists of instructions to be carried out in extraordinary situations which might occur at any point. For example, there is a special condition devoted to alerting the player that he has died, and taking appropriate action. For the most part, these can be left safely alone and work well, but they can be used to tailor the system response. Again, an example might be helpful. There is a special condition that is called when you try to pick something up that is too heavy. The standard response is to tell the player that it is too heavy and leave it at that. However, if you are feeling particularly cruel towards your player, it could quite easily be altered to make him try to pick the object up, stumble, fall over, and drop everything else he was carrying.

The conditions are entered in a language which is quite close to English, but which has a limited vocabulary. Most of the commands are quite obvious and easy to learn, so here goes...

I shall go through the conditions in logical sections, since they divide up quite neatly. The first section deals with tests.

TESTING FOR A CONDITION

Most conditions will be of the form " If such-and-such happens, then do so-and-so ", so there are two words ***if*** and ***then*** which do exactly what you would expect. (Note that words in *italics* are words that the **STAC** understands.)

So, most of the time, condition lines will look like:

```
if { test-is-true } then { do something }
```

Usually, after executing the { something } you will want to stop and wait for a new command from the player. This is done with either ***wait***, which just waits for a new command, ***ok***, which prints "Okay", and then waits for a new command, or ***newcom***, which will ignore any other commands on the line that the player typed, and ask for a whole new line of commands. This is most useful when the player does something silly and would not want to go blundering on obeying all the other commands on the line. So, our condition now looks like :

```
if { test-is-true } then { do something } wait
if { test-is-true } then { do something } ok
or
if { test-is-true } then { do something } newcom
```

The most common things you will want to test for are words that you entered as a command.

COMMANDS

Commands can be split up into three sections: verbs are words which do something, nouns are the things to do it to, and adverbs say how to do it. For example, in

Take the lamp quietly.

the verb is "take", the noun is "lamp", and "quietly" is an adverb.

You can test for these using the words ***verb***, ***noun***, and ***adverb***. The construction

```
if verb "take" then { do something } ok
```

will do whatever is after the ***then*** if and only if the word "take" was typed in the command line. Note that you must already have entered the word "take" in the verb table. You can combine several words with ***and*** and ***or*** if necessary:

```
if verb "take" and noun "lamp" then
                                  { do something } ok
```

There is no real limit to the number of ***and***s or ***or***s you can combine in a condition.

Similarly, "lamp" must have already been entered in the noun table. If you do not do this, the program does not know that the word "lamp" is a noun, and will complain at you.

Obviously we need to do something useful with the { do something } or all our efforts so far are pretty useless.

OBJECTS

Objects are there for you to manipulate. The most obvious things you do with them are to pick them up, and drop them. Each object is given a number so that we can refer to it in an unambiguous way.

Imagine that object number 1 is a lamp. To get the lamp, we use *get 1* . To drop it again, use ***drop*** *1* .

So, we can now pick up the lamp in response to a command by replacing the { something } with a ***get***.

```
    if verb "take" and noun "lamp" then get 1 ok
and
    if verb "drop" and noun "lamp" then drop 1 ok
```

We use ***ok*** here because the action of ***get***ting or ***drop***ping an object is invisible to the player.. he needs to know that his command has been dealt with correctly. Thus, the ***ok***, which just gives an indication that everything is all right.

These are very useful conditions, and they are very close to the English form. There are other things you can do with objects.

You can list the objects in a certain room with ***list***. If you want to list the objects you have with you, use ***list with***. So, to check what you have with you, you can use

```
    if verb "inventory" then list with wait
```

which will list all the objects you are carrying when you type "inventory". See how we now use ***wait*** rather than ***ok*** since the list being printed on the screen is all the player needs to see that his command has succeeded. Note that ***with*** is really a room that moves around with you, and you can use it like any other room number.

You can describe objects. There are two descriptions per object - a long one and a short one. ***objlng*** prints the long one, and ***objsht*** prints the short one.

You can summon an object (if it exists) and put it at your feet using ***bring***, and teleport yourself to an object using ***find***. (If you are already with the object, or you are carrying it, ***find*** has no effect .) So

```
    if verb "summon" and noun "lamp" then bring 1 ok
```

will bring the lamp to you when you type "summon lamp".

You can move an object to a room using ***to***. *1 to 3* will move the lamp instantly to room 3. You can then test where an object is using ***in***.

```
    if  1 in 3  then { do something } wait
```

will { do something } if object 1 (the lamp) is actually in room 3.

You can also swap two objects using ***swap***. *1* ***swap*** *2* will swap objects one and two over. This saves a lot of mucking about when using lamps, for example. You simply have two objects, an unlit lamp and a lit lamp, and swap them over when you light the lamp.

You can check whether an object is being carried using ***carried***, whether it is here in this room using ***here***, and whether it is available (ie within reach.. either being carried or in this room) using ***avail***. All the words that are understood are 7 letters or less long which is why we use ***avail***. It is the best meaningful abbreviation of "available" that fits. As usual you can combine these to produce quite complex conditions.

```
  if verb "get" and noun "lamp" and
                             here 1 then { do something } ok
```

will only execute the { do something } when you typed "get lamp" and the lamp is actually in this room. Getting and dropping everything is quite useful, hence ***getall*** and ***dropall*** which do just that. It is unrealistic to expect the player to be able to carry every single object at the same time, so the weights in the object description mount up. If this amount exceeds the strength, then you can't pick any more up. So we have a command ***setstr*** which sets the player's strength. If we do ***setstr 10*** then the player can carry 10 objects of weight 1, or 1 of weight 10, or 2 of weight 5, or one each of weights 1,2,3 and 4, or.. and so on. Any attempt to pick up more than 10 weight units will fail. You can change the strength during the adventure. Hence for a cheat, you could use:

```
    if verb "superman" then setstr 5000 ok
    if verb "superwimp" then setstr 1 ok
```

You can find out the weight of any object using ***weight***. This introduces something new. Everything we have met so far either was a test or does something. If you view the words as little servants, the tests will answer yes or no to a question, and the others are dumb brutes who just go away quietly and do something. This ***weight*** answers back with a number, but what can we do with it ?

Well, you can check it against other numbers as a test. For example:

```
      if weight 1 > 5 then { do something } wait
```

will do the { do something } if and only if the weight of object one is greater than 5. There are several of these types of tests.

Using any two numbers (which I shall call a and b since they could be anything), there are the following tests :

```
      a > b will work if number a is greater than number b
      a < b will work if number a is less than number b
      a = b will work if number a is equal to number b
```

```
    a >= b will work if a is greater than or equal to b
    a <= b will work if a is less than or equal to b
and a <> b will work if a is not equal to b
```

So the following tests will work (they are true)

```
    3 > 1    4 < 7    3 = 3    4 >= 4    5 < 9    3 <> 4
```

but these will not

```
    1 > 3    6 < 4    3 = 9    4 >= 7    3 > 9    3 <> 3
```

Many of the words used by STAC return numbers, and these numbers can be manipulated further using simple arithmetic. Although this is not used too often, it can provide some quite useful "short cuts" in some situations.

You can use + (add), - (subtract), * (multiply) and / (divide). To illustrate, here are the results of several simple calculations :

```
    4 + 3     =7
    4 * 3     =12
    4 - 3     =1
    4 / 2     =2
```

In a more complicated calculation, the *'s and /'s are always worked out first, before the +'s and -'s so:

```
    5 * 3 + 9 =24
      |
     15 + 9
        |
       24
```

To alter this, use brackets - anything in brackets will be worked out first. Note also that words that require numbers (like ***weight***) have preference even over multiplication.

*weight 3 * 5*

will return five times the weight of object 3, not the weight of object 15. Again, brackets would overrule this order of preference.

The final three words to do with objects are not used very often. ***cntobj*** counts the objects in a particular room, ***firstob*** gives you the number of the first object found in a particular room, and ***whereis*** gives the room number where the object is.

MESSAGES

So far, we have met quite a few words, but not many print things on to the screen. The few that do (***list*** for example) give a rather bare response. If you were carrying a lamp and some gold, the inventory condition above would print

a lamp, some gold

when you asked for an inventory. Messages can be used to liven it up and make it a bit more friendly. Would it not be better if we could make that into

You are carrying a lamp, some gold

Well, we can using the messages. Messages are entered like room descriptions and objects, and are called by their numbers. So, if we entered message 1 as "You are carrying ", then we could add to the condition like this:

if verb "inventory" then message 1 list with

When you type "inventory", the condition first prints message number 1 ("You are carrying "), then lists the objects you have with you.

Note that some of the messages are used by the STAC, but you can change the wording if you wish, as long as you don't change the meaning significantly.

If you want your message to appear on a new line, then use the word ***lf*** first. This stands for LineFeed, and will move the printing position on to a new line, scrolling the screen up if necessary.

You can also print out numbers, using ***print***. If you wanted to weigh an object, for instance, you could use

if verb "weigh" and noun "lamp" then
print weight 1 wait

which will print the weight of object 1 (the lamp) when you type "weigh lamp".

ROOMS

Since you can move about in the rooms of the adventure, some words to do with this might come in handy. The first of these is ***look***, which will print a description of the room you are in, along with a list of any objects that are here, and a picture if there is one. This one is very useful if you have been in a room a long time, and wish to see the description of the room once more.

To move between rooms. use ***goto***, which takes you to a specified room. This can be used for magic teleporting words, beloved of Colossal Cave fans...

if verb "xyzzy" then goto 7 wait

will take you immediately to room 7, no matter where you were before. It also prints a description of the new room, and depending on whether you have been here before, will either print the short description if you have, or the long description if not. ***moveto*** is exactly the same, except it does not print the room description.

The ***goto*** is also very useful for movements which cannot be included in the connection table. Connections take place no matter what, but if you require something to be done before you can go somewhere, then use a condition. As an example, say you are in room 1, and there is an opening to the east. However, you need the lamp to pass through the opening. So, you must type "east", and be carrying object 1 (the lamp) to go east to room two...

if verb "east" and carried 1 then goto 2 wait

You can describe other rooms at a distance using ***descing***, which prints the long description, and ***descsht***, which prints the short one. This could be useful for, say, a magic crystal ball which allows you to look through it at another, distant, room.

For looking through doorways, you will need to know which room is connected in which direction. This is accomplished using ***connect***. It gives you the number of the room lying in a direction described by a verb... a bit confusing to say, easier to show. Again, one of my myriad examples should help. If you are in room two, and room three is in the connections to the north, then ***connect "north"*** will give you the result 3 since room 3 is to the north of your current location. This will only work with those doorways that are entered in the connections for that room, not those that have to be dealt with otherwise.

To find out which room you are in, use ***room***. So, to list all the objects in this room, use ***list room***.

To draw a picture, use ***draw***. In conjunction with this, ***pictof*** gives you the number of the picture associated with a particular room. So if room 2 has picture 17 associated with it, ***pictof 2*** will give the result 17. The construction ***draw pictof room*** will draw the picture associated with where you are. Pictures can be turned off using ***text***, and turned back on again using ***pict***.

The text beneath the pictures can be in either low resolution (40 characters across) or medium resolution (80 characters across). To switch between these use the word ***split***. The adventure will always start up in low resolution since those of you who have TV sets rather than monitors will probably find 80 character text rather small and difficult to make out. Note that this word will also clear all the text, but will not affect the picture.

To change colours on the screen, there are two words ***colour*** and ***topcol***. Both work in the same way, but ***colour*** changes the colours in the text region of the screen, below the picture, and ***topcol*** changes those in the top part of the screen, that is the picture itself. As an example,

0 colour 666

will change the background colour of the bottom part of the screen to grey. The first number (0) is the number of the colour you want to change - 0 is the background - and the second number represents the amount of red, green, and blue you want in the colour. This may range from 000 (black) to 777 (bright white). If you are unsure about the exact colour that will result, go to the graphics screen (see next chapter) and use the colour sliders to get the shade you want. Then just read off

their positions in order and voilà - your colour ! As another example, red and blue make magenta, so to get magenta (purple) text (colour 3 is what appears black normally) on a black background (colour 0), use

0 colour 000 3 colour 707

707 means 7 units of red, 0 of green, 7 of blue. You may change colours 0 to 3 using ***colour***, and 0 to 15 using ***topcol***. Experimenting with these can lead to some pleasing effects.

To test if you are in a particular room, use ***at***.

if at 4 then { do something } *wait*

will do the { do something } only if you are in room number four.

Two other, little used words are ***visit*** and ***visit?***, which set this room as having been visited already, and test whether this room has already been visited respectively. Thus

if visit? then message 2

where message 2 is "Hmmm... this looks familiar" will print that if this room has already been visited.

<u>MARKERS AND COUNTERS</u>

A lot of the time you will need to store information about the adventure "universe" for later reference - which doors have been opened, which buttons pushed, and how much gold you have. This is accomplished using markers and counters.

Markers are used for things which can be in two situations - doors can be open, or not, and buttons can be pushed, or not. It may be dark, or not, there may be air, or not... that may be enough examples, or not !

There are 512 markers, numbered 0 to 511, and they can be either set or reset. You might represent an open door by a set marker, and a closed one by a reset marker. You must be able to set and reset markers, so the words ***set*** and ***reset*** do this for you. There is also a word ***change*** which will check the state of the marker, reset it if it was already set, and vice versa. It changes the state of the marker.

You can check the state of a marker using ***set?*** and ***reset?***, which will test if the marker is set or reset respectively.

Imagine you are in room one, with a door to the east, which is initially closed. You must open the door before you can go east to room two. So, let us use marker 3 to represent the state of the door. If it is set, the door is open, otherwise it is closed. (All of the markers are initially reset when you enter the adventure.) The following conditions will take care of the door:

if verb "east" and set? 3 then goto 2 wait

will only allow you to room 2 if you typed "east" and the door was already open. To open the door *if verb "open" and noun "door" then set 3 ok*

This will set marker 3, which represents an open door, if you typed "open door".

Markers 0,1 and 2 are used by the special conditions to denote three special situations. Marker 0 is set whenever a room is described. Marker 1 is set if it is dark, and marker 2 being set indicates that you have a source of light.

There are also 512 counters, numbered 0 to 511. They can be used to store numbers, such as the amount of gold you have left in your purse. To set a counter to a value, use ***setcntr.*** To give yourself 200 pieces of gold in your purse (counter 1), use the construction ***200 setcntr 1.*** Counter number one now holds 200. You can think of the counters as little boxes, numbered 0 to 511, each of which holds a number. You can put a new number in, which is what we have done, or you can look at it using ***counter.*** If we used ***print counter 1*** now, it would print the number 200 on the screen, since counter 1 holds 200.

You can increase or decrease the counters by one using ***inc*** and ***dec.*** This is useful as a countdown timer. If you have set off an alarm, say, and you have five moves to escape, you can set a counter to 5, ***dec*** it every turn, then do something nasty to the player when it reaches zero.

If, however, you are using the counter as a money indicator, when you buy something for 20 gold pieces, you do not want to have to type ***dec*** 20 times. To do this, use ***-count,*** or ***+count*** to increase your gold. ***20 -count 1*** will take 20 from the value of counter one. ***50 +count 1*** will add 50 to it. Both of these update the counter - there is no need to do a ***setcntr*** afterwards.

Finally, to check if a counter has reached a certain value, use ***=count.***

if 0 =count 1 then { do something }

will only do the { do something } if counter 1 has reached the value zero.

Again, the special conditions in the QSTART file need the use of counter 0, which you should set up to hold the score.

LIFE, DEATH AND THE DISKS

There should be some definite end to the game, either by succeeding in your quest, or by getting killed. Hence two words which do just what they say. ***death*** and ***success*** both end the game in a similar manner, but with different messages. The first gives a "You have died" type message, the second "Well done !" In addition, the player might want to give up. Thus the word ***quit,*** which is the same as ***death,*** except that the player is asked if he is sure that he wants to quit. If he answers "Y", for yes, then he is exited from the game. If he answers "N" for no, then he is returned to the game.

Adventure games are dangerous things, and you are likely to get killed several times before completing the adventure. Most adventures give the option of saving your game position to disk, and restoring it again afterwards. This is done in **STAC** using the words ***save*** and ***load,*** which do just that. The player is asked to give a name for the position before the save or load is executed.

Closely related to these are ***ramsave*** and ***ramload,*** which save and load game positions to and from a reserved portion of memory, rather than the disk. This has the advantage that it is considerably faster, and that disk space is not taken up with lots of game positions in dangerous situations. It is, however, lost on exit from the game. There are three ***ramsave*** areas reserved, and you can save and load from these using ***ramsave 1, ramsave 2, ramsave 3, ramload 1, ramload 2*** and ***ramload 3.***

MULTI-PART ADVENTURES

Adventure games can be very large, too. If an adventure cannot be squeezed on to one disk (or into memory) then extra data can be saved on to other disks using the *Encode* option on the disk menu.

When a game is saved as a runnable adventure, two files are created on the disk, both with the same name, but one with a .PRG extension and one with .LNK. The file with the .PRG extension contains the parts of **STAC** needed to run the the game whilst the .LNK file holds your game data. When data is saved using *Encode* from the disk menu, a file containing your game data is saved, also as a .LNK file. Several .LNK files can be used by the same adventure either from the same disk or spread over several disks. This allows you to write enormous adventures.

Thus, if we assume that a game is split into three parts, each one taking up a full disk (more or less), then disk 1 would hold the files PART1.PRG and PART1.LNK, disk 2 would hold the file PART2.LNK, and disk 3 would hold PART3.LNK. Parts 2 and 3 would both have been saved using the *Encode* option, whilst part 1 would have been saved as a runnable adventure.

When a new file is linked in, all the markers and counters stay intact, as do object positions, except when the object is in room 0. In this case, the object is set back to its start position. This prevents objects reappearing at their start positions if they have been picked up or moved before crossing over to the new .LNK file. However, this does mean that you cannot use room 0 to destroy objects any more ! Instead, you should use another unused room. Room 10000 is useful in this respect since there can never be a 'real' room 10000, and so the player can never get to the destroyed objects.

The command ***link m*** is used to swap from one link file to another during play. The ***m*** is the number of the message which holds the name of the .LNK file to be loaded. (Note - ***link*** will only work in a runnable adventure, not during the testing stage.)

The ***link m*** can be used anywhere in the conditions generally in the form:

If { new file needed } then moveto r link m

which will put you in room r of the new .LNK file.
For each file that can be moved into, a message must be set up containing the name of the file so that if the game was in three parts, each of which being accessible from the others, each .LNK file would need to contain two messages for the names of the other two .LNK files.

When link is operating it looks for the named file on the current disk, and if found it loads the file, describes the new room and waits for a new command. The conditions cannot be continued since a new set will have just been loaded in !

If the named file is not on the current disk then the link command is ignored and the rest of the condition line is acted on. This should ideally ask the player to insert the correct disk and then press a key. Then it should attempt to load the .LNK file again. If the name of the .LNK file is in message 5 and a prompt message asking the player to change disks is in message 6, then the following construction will work:

if { new file needed } then repeat link 5
message 6 pause 5000 until false

The ***pause 5000*** will wait for about a minute and a half, or until the player presses a key. This is a useful way of waiting for the player to respond to something important.

When writing a multi-part adventure, it is a good idea to create and save your own "multi-quickstart" file containing all the verbs, nouns, adverbs, and objects in the adventure, along with the messages and high and low priority conditions that govern things that can happen anywhere. You can then load this at the start of a new section and just put in new room descriptions, messages and local conditions, rather than having to type everything in again. This approach requires a bit more planning in the short term, but saves a LOT of time later on.

One other thing to note with multi-part adventures is that ***save*** and ***ramsave*** save the name of the .LNK file that you are currently in, for later reference, and ***load*** and ***ramload*** will attempt to reload that .LNK file if necessary. If they cannot find it, special condition 18 is called, which just prints a prompt and waits for a keypress before trying again (much like the construction above).

STRINGS

It will sometimes be useful to ask the player a question which requires an answer which cannot easily be handled using the standard verb-noun-adverb commands. The player's name, for example, would be impossible to determine. However, for purposes such as this we have 16 "strings", which can be used to store a sequence of letters and/or numbers up to 79 characters long. To get a string from the user, use ***get$*** (the $ on the end is usually pronounced "string", by the way, and is a fairly standard

way of showing something to do with strings). This will allow the player to type in a string. For example, if message 1 says "What is your name? ", then the following will get the player's name and store it in string number 0 :

message 1 get$ 0

Now that you have the string, you can print it out using ***print$***, or allow the player to edit it using ***edit$***. If you wanted the player to give you a number, then you can find out the value of the string using ***value***. If the player typed "100" when asked, and this was stored in string 1, then ***value 1*** would give you the number 100 as a result. Note that if the string had not been a number, then the result would be -1. This can be used as a check. Conversely, you can use ***number$*** to turn a number into a string. After ***100 number$ 1*** , string 1 would be "100".

You can also print strings from within messages. If you use the control character Control-V, then string 0 will be printed, and Control-W prints string 1. So, if string 1 is "John", and message number 1 is "Hello, [Control-W]!", then it will be printed as:

Hello John!

You can move strings around using ***copy$***. For example,

1 copy$ 2

will copy string 1 to string 2, destroying whatever was originally in string 2. ***swap$*** will swap two strings over, and is used in the same way.
You can add strings together. To add string 1 on to the end of string 2, use

1 add$ 2

If string 1 contained "Aard" and string 2 contained "vark", then after doing this, string two would be changed to "Aardvark".

You can move the text of a message into a string using ***mess$***. ***1 mess$ 0*** will copy message 1 into string 0. Note that with all of these string commands, if the string gets too long (more than 79 characters) then any characters after the end will be lost.

You can cut a number of characters off the start or end of a string using ***cutst$*** and ***cutend$***. ***5 cutend$ 7*** will cut 5 characters off the end of string 7. If you cut more characters off than are actually there, then you will be left with an empty string - a string with nothing in at all, not even a space.

To find out the length of a string, use ***length$***. This gives you the number of characters in a string.

You can add a character on to a string using ***addchr$***, and find out the value of the first character in a string using ***ascii$***. These use the fact that in a computer, all the characters are represented by a code number from 0 to 255. For example, the code for "A" is 65, so

65 addchr$ 1

would add an "A" on to the end of string 1, and if string 2 was "Aardvark", then

ascii$ 2

would give a value of 65, since the first character in string 2 was an "A".

You can find the first and last occurrence of a particular character in a string using ***first$*** and ***last$***. If string 1 was "HELLO ARTHUR AARDVARK", then ***65 first$ 1*** would give a value of 7 - the first occurrence of "A" is the seventh character in the string. ***65 lasts$ 1*** would give 19. If there were no matching characters in the string, you will get 0 back.

You can compare strings in much the same way as numbers, except that the "value" of one string is smaller than another if it comes before it alphabetically. So "aardvark" is less than "anteater", but "buzz" is greater than "bee". The alphabet is extended so that all the UPPER CASE letters are smaller than all the lower case ones, and all the numbers are smaller than that. Using this system, here is an alphabetical list.

```
" Note the space on the front"
"88 wild horses"
"Alphabet soup"
"Zoological"
"ZZ9 Plural Z Alpha"
"aardvark"
"zap gun"
```

You can compare strings using the following words :

0 =$ 1	Is true if string 0 is EXACTLY the same as string 1
0 <$ 1	Is true if string 0 comes before string 1
0 >$ 1	Is true if string 0 comes after string 1
0 <>$ 1	Is true if string 0 is NOT exactly equal to string 1
0 <=$ 1	Is true if string 0 comes before, or is equal to string 1
0 >=$ 1	Is true if string 0 comes after, or is equal to string 1

Finally for the string section (pun not intended), is the word ***obey$***. This passes control from the player to a string. As an example, if string 1 was "Go north then east", then

obey$ 1

would abandon the rest of the command line that the player typed, and then make you go north and then east, just as if you had actually typed it in. It then returns control to you, the player. This can be used to talk to other characters in the adventure and asking them to do things.

Closely related to this are the two commands ***comm$*** and ***parse$***. ***comm$ 1*** takes the rest of the command line and copies it into string 1, and ***parse$ 1*** will search for nouns, verbs and adverbs in string 1 and fill in the values as if you had typed string 1.

Note that this keeps any verbs, nouns and adverbs already found in your command line. A useful application for this is when you require some extra clarification. Say, for example, that there are two keys in your possession, a silver one and a bronze one, and the user types "drop key". The following condition will ask the user "Which one, the silver key or the bronze key ? " (in message 1), get a response from the user, and fill in extra adverbs (for example "bronze" or "silver") -

```
if verb "drop" and noun "key" and zero? adverb1
      then message 1 get$ 0 parse$ 0 lf
```

This sort of thing makes adventures really friendly and pleasant to use.

COMMENTS

You can include comments in your condition lines if you like. If you include the characters ";" (semicolon) or "\" (backslash) in the line, everything after that on the line will be ignored when the conditions are executed, but will be stored for you to refer to later. Here is an example:

if verb "quit" then quit ok \ let the player quit

or

if verb "save" then save ok ; save the position

This can be very useful in keeping track of what you are supposed to be doing next ! However, they can take up quite a bit more memory than the actual conditions !

ADVANCED CONDITIONS

There are still a few words left which are slightly more advanced than the rest, and whose use is limited at first, but can be put to very good use after the basic condition system has been mastered. They do not really fall into any neat category.

random returns a random number between one and a particular number. For example ***random 100*** could come up with any number between 1 and 100 inclusive.

caps makes sure that the first letter of the next thing printed on the screen is a capital letter. This is useful when listing objects (as in ***getall***), where the first letter is usually small.

Tests return two values - true and false. For example, 3 = 3 is true, but 4 > 7 is false. Hence the words ***true*** and ***false***, which just return true and false values. True is given value -1, false is 0.

In the verb, noun, and adverb tables you were asked to supply each word with an identifying number. Any words with the same number were taken as being the same. You can directly access the numbers of the verb, nouns, and adverbs typed in a command.

verb1 gives the number of the verb typed, ***noun1*** and ***noun2*** give the number of the first and second nouns, and ***adverb1*** and ***adverb2*** give the numbers of the first and second adverbs. This is very useful for dealing with objects. If you define the name of object 1 as noun 1, object 2 as noun 2 etc., then a whole string of conditions like :

```
        if verb "get" and noun "lamp" then get 1 ok
        if verb "get" and noun "fish" then get 2 ok
        .....
        etc.
```

can be replaced using a single condition. If the last object was, say, number 10, then the following conditions would allow you to pick up and drop all objects.

```
        if verb "get" and noun1 < 11 and noun1 > 0
                                          then get noun1 ok
        if verb "drop" and noun1 < 11 and noun1 > 0
                                          then drop noun1 ok
```

This works because the object numbers and the noun numbers are the same for each object. It will only work /if this is so.

Closely related to these is ***itis***. Usually, the word "it" in a command refers to the last noun that you typed in. You can force it to something else if you wish by using ***itis***. For example, having just given a message "A rock narrowly misses you", you might like "it" to refer to the rock. Therefore you can use

```
        itis "rock"
```

to achieve this (assuming that "rock" is a recognized noun).

For really advanced tricks, you can even alter the values of ***noun1, noun2, verb1, adverb1*** and ***adverb2*** as you go along. To do this, use the word ***word*** :

```
        13 word 1        puts 13 into noun1
         9 word 2        puts 9  into noun2
        11 word 3        puts 11 into verb1
        99 word 5        puts 99 into adverb1
        54 word 6        puts 54 into adverb2
```

Using 7,8,9,11 and 12 changes the command you are repeating when you type "again", in a corresponding order.

Similarly, you can change the value of ***amount, turns,*** and ***with*** using the three words ***setamnt, setturn, setwith.*** This will be of use when constructing multi-character adventures. This also requires a more comprehensive explanation of ***with.*** It is, as mentioned before, a room which moves around with you, and is, by default, given the number -1. If you wish to have multiple inventories, like one for your pockets, and one for your backpack, this can be achieved by changing the value of ***with*** to, say, -2 for your backpack inventory.

To get a yes-or-no response from the player (like in ***quit***), you can use ***yesno.*** This waits for the player to press either "Y" or "N", which are treated as true and false.

```
   if yesno then { do something }
```

will do the { do something } if the player pressed "Y", or just go on to the next line of conditions if the player pressed "N". Note also that the "Begin Where" option on the main menu will allow you to change the keys used for "Yes" and "No". This is useful for adventures which are not in English, and the words begin with different letters. For example, French is "O" (Oui) or "N" (Non) and German is "J" (Jah) or "N" (Nein).

You can get the amount of stuff you are carrying using ***amount,*** and find out your current strength using ***stren?.*** You can also find out the number of turns taken since the start of the game using ***turns.***

There are a set of tests for checking numbers which have not yet been mentioned.

```
   zero?      will be true if the number is zero
   pos?       will be true if the number is positive
   neg?       will be true if the number is negative
   notzer?    will be true if the number is not zero
   notpos?    will be true if the number is not positive
   notneg?    will be true if the number is not negative
```

Tests may be negated using ***not.*** To check if object 1 is not being carried, use

```
   if not carried 1 then { do something }
```

Combinations of tests use ***and*** and ***or,*** which we have seen already, and can also use ***xor,*** which is, in effect, either one or the other but not both.
Additionally, you may use abbreviations - ***&*** or ***&&*** for ***and,*** ***¦*** or ***¦¦*** for ***or*** (this is accessed by using shift-backslash, next to the Z key), ^ or ^^ for ***xor,*** and ~ for ***not.*** This is used to save time and space, and also to make C programmers feel at home.

pause will pause for a specified time. Fifty pauses is one second, so ***pause 50*** will wait for one second, ***pause 250*** will wait for 5 seconds. Long pauses may be cut short by the player by pressing a key.

Additional special conditions may be invoked by using the word ***special.*** They can then be used for frequently used sets of conditions. To return from a special condition early, usually as the result of a test, use ***return.*** For example, the special condition that describes a room first checks whether it is dark, and if so it prints a message to that effect, and ***returns*** before it prints the room description.

There are two extra words which are designed especially for use within special conditions. ***byebye*** exits you from the game immediately - no messages, nothing. ***newcom*** waits for a new command line to be entered, discarding any commands not yet done on the previous line. This is useful for player errors, such as trying to pick up an object that is not there. Whatever is after this on the command line is ignored, since the player will normally want to try again before proceeding.

The ***if..then*** construction can be extended to ***if..then..else*** if necessary. The part after the ***else*** is executed only if the test is false. An example might be:

if at 1 then message 1 wait else message 2 wait

will print message 1 if you are at room 1, and message 2 if you are not.

Finally, there are two words used for repeating things - ***repeat*** and ***until***. The ***repeat*** marks the beginning of a block of conditions that will be continually repeated until the condition after the ***until*** is true. So, for example, let us write an "Examine all" condition using a loop.

What we need to do is go through every object, see if it is available, and if so print its long description. We can also put it in a special condition so that we can use it easily wherever it is needed. Special condition 20 is not used, so we can put it there.

We need to keep track of the number of the object we are looking at, by using a counter. Again, we must choose one which is not used anywhere else, say counter 3.

The construction ***objlng counter 3*** will print the long description of the object we are looking at, and we can test whether it is available using ***avail counter 3***. All we need to do now is to put these together in a loop which will count from 1 to 511 (the legal objects). Here it is (don't type in the comments in *italics*).

```
1 setcntr 3
                 set the counter to the first object
repeat                                start the loop
if avail counter 3 then if objlng counter 3
          if this object is available then describe it
inc 3
        increase counter 3 ( look at the next object )
until 512 =count 3
                            until counter 3 reaches 512
```

Putting these lines into special condition 20 allows us to describe everything around just by using ***special 20***. See how the ***repeat*** and the ***until*** are on different lines, repeating the whole block of lines in between them. One thing to note is that an ***if..then*** inside a ***repeat..until*** must not be on one line. If you find you have something like :

repeat if { test } *then* { something } *until* { test }

then split it on to three lines:

```
repeat
if { test } then { do something }
until { test }
```

You can put ***repeat..until*** loops inside each other, up to a maximum of 16 at a time. If you come up with a legitimate use for a 16-level loop, please write and tell me.. I can't think of one!

This restriction does not apply for loops which are not inside each other - you can have as many of these as you like !

Similarly, ***special*** conditions can be nested as far as you like, as long as you do not call a special condition which is already in the nest. If you do, it will be ignored.

TRICKS AND TIPS

There are several useful hints and tips which you can use to either "short cut" conditions, or to do things that are quite common in adventures.

The first of these is using ***noun1*** to bypass a lot of ***drop*** and ***get*** conditions. As long as the number of an object and the number of the noun describing it are the same (ie the rock is object two and noun 2 is "rock"), you can check for a whole range of objects in one condition :

```
if verb "get" and noun1 > 0 and noun1 < 11 then
                                      get noun1 ok
```

This will allow you to ***get*** all objects with numbers greater than 0 and less than 11, ie objects 1 to 10. The reason it works is because the value of ***noun1*** is the same as the object number it refers to. Obviously, if you have more or less objects, just change the *11* to one more than the last object number you have defined (21 for 20 objects, 6 for 5 objects etc.).

Wearing things is another useful trick. To wear something you already have, just move it to an unused room (say 19999 for the sake of argument). Since you haven't actually ***drop***ped it, the weight of stuff you are carrying will not be updated, so the weight of the object will still register. To take the object off, just move it back to your standard inventory. Here are examples of wearing and taking off object 2 (a hat) :

if verb "wear" and noun "hat" then 2 to 19999 ok
if verb "remove" and noun "hat" then 2 to with ok

You could have used the ***swap*** objects trick like we used for lighting the lamp, but swapping "a hat" for "a hat (worn)" and then doing a ***dropall*** will allow you to drop that object and you can get nonsense messages like :

You are in a cave. You can also see a hat (worn).

How can you wear a hat that is on the ground ? This trick places it safely out of the range of ***dropall***. To check whether an object is being worn (in this case the hat), use ***in*** with the room number you decided to put worn objects in :

if 2 in 19999 then { the hat is worn } *ok*

The same trick allows you to make use of bags, pockets, etc. To give a list of what you are wearing, use ***list*** with your chosen room number, in our case ***list 19999***.

If you want to be particularly kind to your player, you can build in an "Oops" command to take him back to before his last command. This will use up two of the ramsave positions, leaving one more for normal ramsave and ramload options.

You will need to use a marker (say marker 4 for sake of argument, although you could use any one you want). This will be a record of which ramsave holds your "Oops" position. Then include the following lines:
In special condition 13, add

if set? 4 then ramsave 1
if reset? 4 then ramsave 2
change 4

In the low priority conditions, add

if verb "oops" and set? 4 then ramload 1 look wait
if verb "oops" then ramload 2 look wait

This leaves ramsave position three for the player's ramsave/ramload option.

ERRORS

When entering a line of conditions, several errors can occur. These are accompanied by a beep and a short message at the bottom of the screen. The possible ones are:

Unknown operator - the STAC cannot understand one of the words you have typed. This is most commonly because of a typing error. Example: ***vreb*** instead of ***verb***, or by not leaving a space in between two words, or between a word and a bracket. Example: ***print(2 + 2) * 2*** - you must leave a space between words.

Mismatched brackets - the number of brackets open does not match the number of brackets closed. Example: ***print (2 + (3 * 5)*** [one too many "("]

Mismatched numbers - either there is a result not doing anything or there are not enough results. Example: ***print set 1*** - set 1 does not return a result so ***print*** has nothing to work with. Also, ***weigh 1*** - what do we do with the weight of object 1 ?

Unknown noun/verb/adverb - the word in "quotes" is not in the vocabulary. Example: ***if verb "aardvark" then...*** will give this error if "aardvark" isn't a defined verb.

CHAPTER FOUR

GRAPHICS

The STAC is designed to allow you to accompany your adventure with 16-colour graphic pictures. These may be created using the inbuilt graphic editor, or they may be imported from either **NEOCHROME™** or **DEGAS™** drawing programs.

To get to the graphic editor from the main menu, press *G*. The menu will be replaced by the drawing screen (see next page).

This consists of a large, blank window filling the top two thirds of the screen, and a smaller window filling the lower third of the screen which contains your drawing tools.

You must enter a picture number before you can draw on the screen. This is so that you can recall the picture at a later time. Enter your number (it will appear on the right hand side), and press *RETURN*. You are now ready.

There are a bewildering array of icons on the left hand side of the screen. The first row represent the 16 colours you can use. One will be outlined in a different colour, and this is the one you will be drawing in. To change to a different colour, just move the mouse pointer over the colour you wish to draw with, and press the left mouse button.

The second row represent the different drawing tools you can use. They are, from left to right, airbrush, paintbrush, line draw, rectangle, circle, ellipse, fill area, filled rectangle, filled circle, filled ellipse, merge picture, import picture, resize window, undo, clear screen, and drop back to main menu. Again, selection of these is by positioning the mouse pointer over one of them and pressing the left hand mouse button.

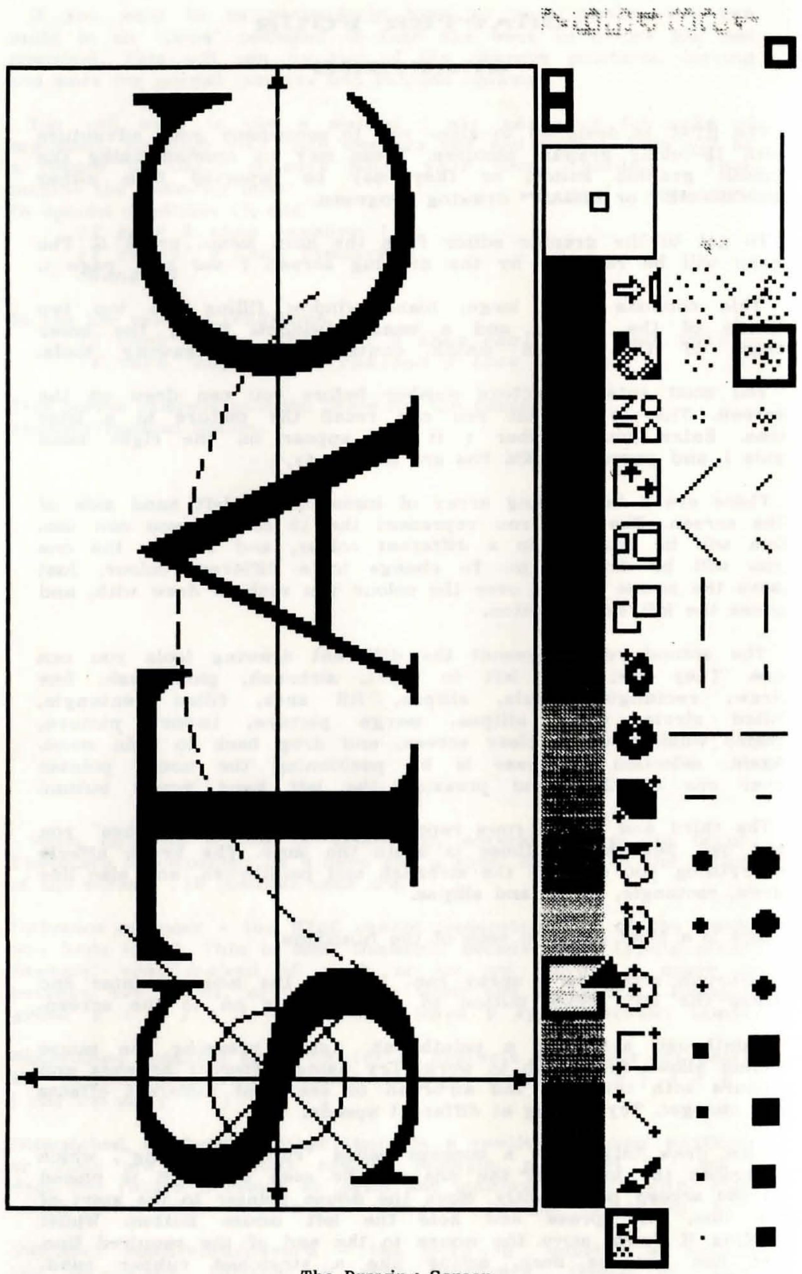
The third and fourth rows represent the different "brushes" you can use. Selection of these is again the same. The brush affects everything you do with the airbrush and paintbrush, and also line draw, rectangle, circle, and ellipse.

Here is a brief guide to each of the functions:

Airbrush acts like a spray can. Position the mouse pointer and press the left mouse button to spray colour on to the screen.

Paintbrush acts like a paintbrush. Again, pressing the mouse button allows the brush to work. Try using different brushes and colours with this and the airbrush to see what different effects you can get. Try moving at different speeds.

Line draw introduces a concept called "rubber banding", which describes the way that the line can be seen before it is placed on the screen permanently. Move the mouse pointer to the start of the line, and press *and hold* the left mouse button. Whilst holding it down, move the mouse to the end of the required line. The line can be seen, acting like a stretched rubber band, allowing you to position it exactly. Release the mouse button, and the line will be drawn permanently.


The Drawing Screen

Rectangle uses the same technique. Move the mouse to one corner, hold the left button down, and move to the diagonally opposite corner, and release.

Circle and ellipse require you to start with the mouse at the centre, then drag the outline of the figure. Don't worry if the rubber band figure seems incomplete - it is merely a guideline. The real thing will be perfect (!) .

By now you have probably got a messy screen, so we will skip ahead to the clear screen option (the icon is meant to look like an eraser, and is the second from the right hand end). To use this, position the mouse over it and click the left hand mouse button, then move the mouse into the drawing area and click it again. The window will clear, and you will be returned to whatever tool you were using before.

The next function is area fill (the paint can icon). Using lines, or the paintbrush, draw an enclosed area. Any shape will do. Then select the fill icon, and click the left hand mouse button inside the closed area. The area will then fill up with colour. Be sure to leave not even the tiniest gap, otherwise the colour will leak out. Clicking on the UNDO icon will solve that, cancelling the last thing you did.

Filled rectangles, circles, and ellipses are managed in the same way as normal ones, but their outline is not affected by the brush being used, and, of course, they are filled in with solid colour.

Merge picture requires that you have another previous picture to merge with this one, so to preserve your current masterpiece, click on the picture number you typed in at the start. After a couple of seconds, which is STAC compressing your picture to save memory, it will be replaced by a cursor and you should then type a different number. The screen will clear, and you will be able to draw another picture. We are now ready to merge pictures. Click on the icon (two overlapping boxes), and you will be asked to enter a picture number to be merged, on the right in the usual place. Type in the number of your first picture. It will be loaded onto the screen, overlapping your current picture. This is useful for adding subtle detail to a sequence of largely similar screens.

Import picture allows you to cut a rectangular piece from an uncompressed **NEOCHROME™** or **DEGAS™** slide which has been saved to disk previously. Click on the disk icon, and you will be asked "Neo or Degas ?". Press *D* for Degas, or *N* for Neochrome. A file selector box will then appear, from which you can make your slide selection. Click on *OK* to load the slide, or *NO* to cancel the function. If you selected *OK*, the slide will be loaded in, and the colours set up. You can then draw a rectangle around any part of the screen, which will be cut out and used as your picture. Draw the rectangle as if you were using the rectangle function.

Resize allows you to change the size of the window which you wish to use for drawing in. Note that it also clears the picture - you cannot get it back. Click on the resize icon (rectangle with arrows), and draw a rectangle in the usual way on the upper part of the screen. It will then be centred, and you can use it for drawing in.

Undo allows you to undo the last thing that you did, and only that one last thing.

The final icon drops you back to the main menu.

By this time, you will probably have noticed that there is a small window towards the right of the screen, in which appear coloured squares. This is a magnify window, and with it you can see the area around your mouse position with pixel accuracy. It can be used to pinpoint the position of your mouse for sensitive things, like joining lines together, or small area fills.

Underneath this, at the bottom of the screen, is a small horizontal line. Clicking on this with the left mouse button will change the selected line style. There are 16 different styles, so clicking 16 times will bring you back to the start pattern again. The line style affects lines, of course, but also the paintbrush and the draw rectangle options. There are many useful effects which can be achieved using a combination of brushes and linestyles.

At the extreme right are three boxes which slide up and down a scale marked 0 to 7. These represent the amount of red, green and blue that go to make up the currently selected colour. You can move them by clicking the mouse at the position you want them to move to. The colour "palette" that you set up will be remembered when you leave the picture. This is an almost indispensible feature - the default palette of colours is useful for only a very few pictures. You can get quite subtle shade variations using this feature, since there are 512 colours available.

When they are printed on the screen during an adventure, the pictures can either sit snug against the top of the screen, or space can be left at the top for an extra line of text. This space at the top of the screen is set up in pixels from the Begin where? option on the Main Menu. They are not surrounded by the frame shown on the drawing screen - you can draw your own if one is needed.

The pictures are stored in a compressed format, to save on memory. If a picture cannot be compressed (as is the case for some digitized pictures, and for highly detailed or random pictures), then it is stored as a complete section of screen. This is why you may notice that some pictures are loaded more quickly than others. A full sized, uncompressed screen will take about 18000 bytes of memory. Compression will take this figure down by a significant factor, as will reducing the size of the window.

You may have up to 9999 pictures, memory permitting, but in practice this will not be possible. The number of pictures you can fit in varies according to the amount of detail you include.

The simpler the pictures, the less memory they take up. In the worst case, 16 full size very detailed pictures will fill 300 000 bytes of available memory.

One thing it is worth remembering is that a picture is not necessarily worth a thousand words. Many of the best adventures ever produced were purely textual (ie no pictures !). It is better to have a sparsely illustrated adventure that has many locations and is fun to play than one which has a lot of brilliant pictures but which is unplayable.

Many adventurers maintain that pictures are a waste of time, effort, and memory. I disagree. There is plenty of scope for including pictures in adventures, but as I have said, don't neglect the adventure to produce the pictures.

CHAPTER FIVE

MUSIC IN MESSAGES

The music system used in **STAC** is quite basic in construction, but needs a separate chapter to explain - so here it is !

The tunes are entered in messages, and are played when the message is printed. They can even be mixed with normal text. But how does the **STAC** know what to play and what to print ?

Tunes are started by typing *Cntrl-T* (T for tunes). Now anything after this will be interpreted as music, until either the end of the message is reached, or a *Cntrl-U* character. The first thing to do, usually, is to set the volume and tempo of the tune. This is done using the *v* and *t* commands. Each of these is followed by a number, then a space (remember the space on the end !). So, to set the volume to 15 (loudest), and the tempo to 5 (10 beats per second), use

v15 t5

Ok. So far so good, but what about playing notes ? Notes are represented by the letters abcdef and g, as in normal music. You have two octaves for use immediately - to get the upper octave use upper case (ABCDEFG). So, to play a scale of C, you might try

v15 t5 cdefgABC

Remember to include the *Cntrl-T* and *Cntrl-U* to show that this is a tune. (You will have to actually print the message to hear the tune). All the notes here are in the key of C. To sharpen a note, append "#". Here is a scale in the key of C#:

v15 t5 c#d#ff#g#A#CC#

To change the durations of the notes, add a number afterwards. Notes with no number afterward will sound for the amount of time dictated by the last number. Here is a scale with an odd rhythm:

v15 t5 c2 de1 f.g2A.BC4

Notice the use of "." as a rest here. The duration can be altered just as if it was another note.

Two octaves is a bit restrictive. You can access the full range provided by the sound chip by using + and -. These transpose up and down one octave respectively. Here is an example of an extended scale in C:

v15 t5 cdefgAB+cdefgAB+cdefgABC

Repeats can be handled easily. The start of the repeat is marked by an *r* with the number of repeats. The end is marked by a colon (: because it looks a bit like a repeat mark). Hence to play a scale 4 times, use:

v15 t5 r4 cdefgABC :

Finally, chords can be played like ordinary notes. Just replace the note with curly brackets *{}* around the notes you want to play as a chord. Here is an example:

v15 t5 cdefgAB{Cec}

which is a scale with a chord at the end. You may have up to 3 notes in a chord. You can change durations in the same way as a normal note:

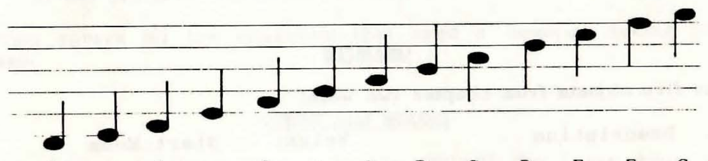
v15 t5 cdefgAB{Cec}4

will carry the chord on for four "beats".

The relation of tempo to "beats" per second is given here. Also, if we assume that 4 "beats" is a crotchet note, then the tempo in beats per minute is also given.

Tempo	"Beats"/sec	Crotchets/min
1	50	750
2	25	375
3	16.7	250
4	12.5	187
5	10	150
6	8.3	125
7	7.1	107
8	6.2	94
9	5.6	83
10	5	75
11	4.5	68
12	4.2	62
13	3.8	58
14	3.6	53
15	3.3	50

Here are the names of the notes on the treble stave:



Notice that the **STAC** does not wait for the tune to finish before going on to the next thing. If another tune is to be played before this one is finished, then the old tune is stopped, and the new one takes over.

CHAPTER SIX
OUR SMALL ADVENTURE

Having seen how to set out an adventure in chapter two, and how to use the conditions in chapter four, we will now attempt to get a complete adventure running. I am assuming that you have already typed in the room and object descriptions from chapter two.

As a recap, here they are again.

ROOM DESCRIPTIONS

Room 1: Above Ground Connections: None
You are standing outside a large cave entrance, which runs into the cliff face to the east.

Room 2: Large Cave Connections: east 3
You are inside a large cave. A lit tunnel leads west, and two dim passages lead east and south.

Room 3: Passage Connections: west 2
You are in an east-west passage. A snake is asleep in a corner at the eastern end.

Room 4: Lakeside Park Connections: None
You are in a small park by a lake, which surrounds you on all sides except the west, where there is a cave entrance. The lake looks too cold to swim in.

Room 5: Outside Castle Connections: None
You are outside a giant castle built into the cliff. A cave entrance leads north and there is a door to the east.

Room 6: Strong Room Connections: West 5
You are in the castle strong room. It has obviously not been used for some time, and the only way out is to the west.

OBJECTS

The five objects from chapter two were:

No.	Description	Weight	Start Room
1	a lamp	10	1
2	a dead rat	10	5
3	a key	1	4
4	a bar of gold	100	6
5	a lit lamp	10	0

MESSAGES

The next thing to do is to enter the messages we will need to alert the user to what is happening. Press *M* from the main menu to get into the message editor. This is essentially similar to the editor used for the room descriptions, and the object descriptions. There will already be some messages in there,

numbered 9900 upwards. These are the ones that are used in most adventures, and were loaded along with everything else in the *QSTART* file. To insert a new message, press *Insert*, then alter the message number if necessary. You can now type in your message in the normal way. Here are the messages we will need (don't type in the comments in *italics*):

1. It lights up.
When the player lights the lamp.
2. It goes out.
When the player extinguishes the lamp.
3. It tastes even better than it looks !
If the player eats the rat. Think of everything !
4. The snake wakes up, comes over, and bites you.
If the player tries to go past the snake without the rat.
5. The snake wakes up, comes over, and eats the rat. It then falls asleep again.
When the player goes past the snake with the rat.
6. You hear shuffling footsteps nearby...
After two moves in the dark.
7. An eight foot spider with glowing red eyes jumps from the shadows and neatly severs your head from your twitching body.
After three moves in the dark. Gruesome stuff, eh ?
8. The door unlocks and swings open.
When the player unlocks the door.
9. You bump your nose on the door. Ouch !
If the player tries to walk through a closed door.
10.You got out with the gold !
When the player succeeds.
11.The snake does not like being touched and bites your hand. The venom courses through your veins like liquid fire.
If the player tries to pick up the snake.
12.You find nothing much.
If the player tries to examine an unknown object.
13.I'm afraid I don't know what that is.
If the player tries to examine something.
14.It is a large snake, which looks poisonous.
If the player examines the snake.

This covers all the situations that need a message during the game.

VERBS and NOUNS

In addition to those present in the *QSTART* file (see appendix D), we will need some verbs and nouns to cover specific actions in this adventure. The nouns are quite straightforward, and refer to the objects in the adventure. Also, the player will need to unlock the door, so "door" must be included. Notice that nouns which mean the same have the same number, and the nouns for the actual objects have the same number as the object they refer to... ie "rat" is noun 2, and object 2 is the rat. This is so we can use a short cut later on.

Here are the extra verbs and nouns we need:

NOUNS	VERBS
1 lamp	20 extinguish
1 torch	20 off
2 rat	21 feed
3 key	22 score
4 gold	23 eat
4 bar	23 devour
4 treasure	24 unlock
5 door	25 light
6 snake	25 on

To enter these, press *N* or *V* from the main menu. You will be presented with a familiar list of items. To insert a new item, use *Insert* as usual, then type the number and word on the bottom line of the screen. As usual, the Space Bar will allow you to edit a word, and *F1* will delete it. The words are inserted in alphabetical order, so you may not actually see your word go in. You can scroll, as usual, with the arrow keys.

LOW PRIORITY CONDITIONS

At this point we have all the raw data we need for the adventure, but nothing to say how the adventure will react to player commands. These are mostly handled by the low priority conditions.

To enter these, press *L* from the main menu, which will give the familiar scrolling list. *Insert* will insert a new line of conditions before (above) the line you are currently on. All the lines after this will then be shuffled up one. For example, if the conditions are listed as:

```
1    print 1
2    print 2
3
```

and you are on line 1, then inserting a line "print 7777" will result in:

```
1    print 7777
2    print 1
3    print 2
4
```

You will be asked to type the conditions in a line at a time. Here they are (again, comments are in *italics* - don't type them in !):

Line Condition

1 if verb "score" then message 9902 print counter 0 message 9903 print turns message 9904 wait
If you typed "score", then print score and number of turns taken. Score is held in counter 0.

2 if zero? noun1 and verb "get" then message 13 newcom
if you typed "get" but no recognized noun, say "I don't know what that is"

3 if zero? noun1 and verb "drop" then message 13 newcom
and the same for drop

4 if zero? noun1 and verb "examine" then message 12 newcom
and again for examine

5 if verb "get" and noun "gold" then get 4 20 +count 0 ok
give player 20 points for getting the gold

6 if verb "drop" and noun "gold" then drop 4 20 -count 0 ok
and minus 20 points for dropping it again !

7 if (noun1 < 5) and verb "get" then get noun1 ok
this is the short cut I mentioned. If you typed "get", and the noun number was less than 5 (ie 1,2,3 or 4) then get object 1,2,3 or 4. This only works if the nouns and objects have the same numbers.

8 if verb "drop" and noun "lamp" and carried 5 then drop 5 1 swap 5 message 14 20 -count 0 reset 2 wait
if you typed "drop lamp", when you are carrying a lit lamp, then drop the lit lamp and make it go out, printing a message to that effect.

9 if (noun1 < 5) and verb "drop" then drop noun1 ok
the same short cut for drop

10 if verb "examine" and noun 1 and avail 5 then objlng 5 wait
if you typed "examine lamp" and the lit lamp is available then describe that

11 if (noun1 < 5) and verb "examine" then if avail noun1 then objlng noun1 wait
useful thing, this short cut ! Here it is again for examining things.

12 if verb "eat" and noun "rat" and carried 2 then drop 2 2 to 0 message 3 wait
if you eat the rat, and it is being carried, then drop it, move it to room 0 (ie destroy it) and print the relevant message

13 if verb "light" and noun "lamp" and carried 1 then 1 swap 5 20 +count 0 message 1 set 2 wait
if you typed "light lamp", and the unlit lamp is available, then swap it for the lit lamp, add 20 to your score, print a relevant message, set marker 2 to show you now have a source of light, and wait for a new command.

14 if verb "extinguish" and noun "lamp" and avail 5 then 5 swap 1 20 -count 0 message 2 reset 2 wait
and vice versa for extinguishing the lamp.

The rest of the low priority conditions are to be found in the *QSTART* file.

HIGH PRIORITY CONDITIONS

These take care of events the player cannot control, for the most part. They are accessed from the main menu by pressing *H*. Here are the ones we will need (with *comments* as usual):

No. Condition

1 if set? 1 and reset? 2 then dec 1

if it is dark, (marker 1 set) and we have no source of light (marker 2 reset), then decrease the "turns-left-in-the-dark" counter.

2 if (1 =count 1) and set? 1 and reset? 2 then lf message 6

if you have only one turn left in the dark, then print the "scuffling footsteps" message to warn then player that something nasty is about to happen.

3 if (0 =count 1) then lf message 7 death

if the player has no moves left in the dark, then kill him off !

4 if at 1 and carried 4 then lf message 10 20 +count 0 success

if the player is in room 1 with the gold, then he has won ! Give him another 20 points !

Those are all the high priority conditions needed. The turns-in-the-dark counter is set up at the start of the game by special condition 17 (the start of adventure condition that is executed only when you first enter the adventure). This is available from the main menu under key *S*. Enter 17 for the condition number, and insert line one as follows:

1 3 setcntr 1 setstr 110

This gives you 3 moves in the dark before being eaten. Marker 1 is already reset, therefore it is not dark. Also, set the strength so you cannot carry everything !

To get back to the main menu, press *Esc*, then enter a condition number of 0 when asked.

LOCAL CONDITIONS

These are conditions specific to one place only, and are used for movement between rooms that cannot be handled by the connections, and also for interaction with scenery, such as unlocking the door to the strong room, and dealing with the snake.

They are found on key *C* from the main menu (key *L* was already taken up by the low priority conditions...). Enter the number of the room in which the conditions apply, then you will be asked to enter the actual conditions in the usual way. Here they are:

Room 1

1 if verb "east" then set 1 goto 2 wait

if you typed "east" then set the dark marker, and move to room 2

Room 2

1 if verb "west" then reset 1 goto 1 wait

if you typed "west", then reset the dark marker and go back to room 1. Most of the other local conditions are of this form - setting or resetting the dark marker then moving.

2 if verb "south" then reset 1 goto 5 wait

Room 3

1 if verb "get" and noun "snake" then message 11 death

if you tried to get the snake, then print a message and kill the player off.

2 if verb "east" and set? 4 then reset 1 goto 4 wait

marker 4 is set if the snake has already been fed and will let you past. If so, and you typed "east", then you will get outside again

3 if verb "east" then message 8 death

if you typed "east", and the snake had not been fed (if it had been, the last condition would be true and we would not have got this far), then print a message and kill the player off.

4 if verb "feed" and noun "snake" and carried 2 then message 5 set 4 drop 2 2 to 0 20 +count 0 wait

if you typed "feed snake" then say the snake has eaten the rat, and set marker 4 to record that it has been fed. Then destroy the rat and give the player 20 points.

5 if verb "feed" and noun "snake" then message 8 death

if you typed "feed snake" and you don't have the rat, then it will come over and bite YOU !

6 if verb "examine" and noun "snake" then message 14 wait

if you examine the snake then print its description (in message 14) and wait for a new command.

Room 4

1 if verb "west" then set 1 goto 3 wait

Room 5

1 if verb "north" then set 1 goto 2 wait

2 if verb "east" and reset? 3 then message 6 newcom

if you try to go east, and the door (mrkr 3) is closed, then print the appropriate message and wait for a new command line.

3 if verb "east" then goto 6 wait

if the door is open, goto room 6

4 if verb "unlock" and noun "door" and carried 3 then message 8 set 3 20 +count 0 wait

if you type "unlock door", and you have the key, then print an appropriate message, mark the door as being unlocked, and give the player another 20 points.

Room 6

No local conditions

To get back to the main menu, press *Esc* twice.

BEGIN WHERE ?

On the main menu there is an option *Begin where ?*, accessed by pressing the *B* key. This sets up where in the adventure you will start, space at top of screen, and keys to use for yes and no, and asks you to supply a name for a loading picture. If you do not wish to have a loading picture for your runnable adventure, just click on the *NO* button on the file selector box when it appears.

SAVE THE ADVENTURE

Now would be a good time to save the adventure. This is done from the disk menu (press *D* from the main menu). On the disk menu is the option *Save adventure data.* Press key *S*, and a file selector will come up, showing the files that are on the disk currently in your drive.

Replace this with a disk which you don't mind cluttering up with adventure data (preferably a blank, formatted disk). To check what files are on this disk, click on the drive letter (it will usually be A:) on the right of the file selector box.

To enter a name for your adventure data, click on the filename at the bottom of the file selector, and type in the name of your adventure data. This can be up to eight letters long, and you should follow it by ".ADV" to show that it is an adventure file. Press *Enter* when you have a satisfactory name, then click on the OK box to start saving the data.

The disk drive will whirr for a short while while the data is being saved. When it has finished, you can return to the main menu by pressing *Esc* as usual.

PLAYING THE ADVENTURE

You can test the adventure from within the STAC by pressing *Enter* from the main menu. You should then be able to play the adventure. If anything goes wrong, you can call up a help screen by pressing *Help* then *Enter* when the system asks "What now ?". This shows the state of all the markers and counters, as well as where all the objects are, and various other useful information. You can scroll up and down the using the arrow keys as usual, and return to the game using *Esc.*

If anything really major goes wrong, then you will be given an error message, and a listing of the line of conditions where the error occurred. You can force an error by pressing both *Shift* keys at the same time in case you get stuck in an infinitely repeating loop. This will return you to the main menu. Alternatively, pressing the **Esc** key when asked for a command will do the same.

The errors are listed in appendix E, with a brief explanation of what is likely to have caused them.

Remember to test everything the user is likely to say - don't just test that the adventure is solvable. Attempt to be as stupid as possible. If a room description says:

```
    You are in a cave. Entrances lead north and east.
```

Try going south and west too, and up, and try examining the elderberry bush that is not there, and so on. This can uncover many a mistake !

If, however, everything went all right, you can save the adventure as a *stand-alone* program. This means that you can just start it up in the usual way from the desktop, and other people will be able to play it even if they do not own the STAC. Note

that this will require around 30K of extra space on the disk, since the part of STAC that is responsible for running the adventure must also be saved.

If you wish to sell your adventures, that is fine - we ask for no licensing or royalty payments. However, the adventure should contain a credit along the lines of:

Developed using the ST Adventure Creator by Sean Ellis for Incentive Software 1988

preferably on starting the adventure.

To do this, go to the disk menu and press key *R* for *Runnable adventure program.* Then follow the same procedure as for saving an adventure data file, except give the file a ".PRG" extension rather than ".ADV".

If you have one of the larger ST computers, with double sided disks and over 512K of memory, it is a good idea to remember that most people who will want to play your adventure will probably have single sided disk drives and 512K of memory, so format your disks single sided and try not to use more than about 290K for your adventure. This will ensure the widest possible market.

CHAPTER SEVEN

THE DISK AND PRINTER MENUS

So far, we have glossed over the printer and disk menus. This chapter explains what the options in each menu do.

DISK MENU

The disk menu contains the options:

- Encode and save link file
- Format a disk
- Load adventure data
- Merge data section
- Output data section
- Runnable adventure program
- Save adventure data
- X - erase a file
- ? Disk information

as well as *Esc* to return to the main menu. We have already met *Load,Save,* and *Runnable. Format* sets up a completely blank disk so that you can save data on it, *Output* allows you to save a single section of data on to the disk, such as just the verbs, or just the pictures. *Merge* allows you to merge these saved sections back into your adventure. This is useful for building up, say, a library of pictures which can be merged in after an adventure is written and tested. The *X* option erases a file from the disk, and the *Disk information* option gives you an indication of how much space is free on a disk.

The *Encode* option is used to create files to *link* to in multi-disk adventures. It would not be a good idea to let other people just load these link files into the STAC and look at them, so these files are *encoded*, using a sort of secret password known only to the program. Only if you know the password and how it is used can you get the original data back again. This is only an analogy; you will not be able to load these files by typing in a password, so I thought I would save everyone a lot of wasted time and energy looking for it !

All of these use the file selector, which we have come across before, but not all of its features have been mentioned.

At the top is the *pathname*, which indicates which area on the disk in which to search for files. The name at the bottom is the *filename*, the name of the file to search for. In the middle is the *directory window*, which shows a list of files that are on the disk in the area specified by the pathname. You can select one of these by clicking on it, or enter a new filename by clicking on the old filename at the bottom. On the left hand edge of the directory window are a *close box*, and four arrows, up, down, fast up and fast down. If there are more files on the disk than can fit in the window, you can scroll up and down by clicking on the arrows.

Sometimes you will find that a seemingly normal file will be displayed with a small mark alongside it. This is not a file, but a *folder*. It can contain more files and folders, and is used to organize the disks better. To open a folder and see what is inside, click on it. The close box closes it again. Note how the pathname at the top changes when using folders. You can alter the pathname if you like by clicking on it.

There are several *disk selector* buttons on the right hand side of the file selector. These will only be of any use if you have more than one disk drive, as they allow you to switch between drives quickly and easily. Clicking on the disk selector button for the disk you are already on will update the directory - if you have just changed disks you should do this to see what is on the new disk.

Finally, there are the *OK* and *NO* buttons. Clicking on OK will confirm that you want to continue with what you are doing, and clicking on NO will abandon it. Thus if you select the *Delete file* option by accident, click on NO and nothing will happen. A short cut for selecting a file and then clicking on OK is to double-click on the file you want in the window. Also, the OK button can be activated by pressing *Enter* or *Return*, and the NO button by pressing *Esc*.

If you try to load a file which is obviously wrong - like selecting a file called *TIMES.FNT* (a font file) when asked to provide a name for an adventure file - the STAC will realize that the file is the wrong type, and a box will appear in the middle of the screen saying *Invalid file format*. To continue, press *Return*. This prevents you loading incorrect types of data and confusing the STAC !

PRINTER MENU

Many of you will have printers, but those that don't will still find the printer menu useful. With it you can list any part of the data to the screen or to the printer. You can also send it to the serial port or down the midi lines if you like - this might be useful for transferring data between computers.

To use the printer menu, press *P* from the main menu, then the letter for the data area you want to use - *R* for room descriptions, *O* for objects, etc. Then the program will ask whether you want to send the output to the screen, the printer, the RS-232 port, or the midi. Respond with either *S,P,R* or *M*.

You will then be asked to supply a range of items to be sent, from one value to another. Thus you can list only messages 1 to 10, for example. The list will then appear on the screen or the printer (or be sent to the serial port or the midi).

This is invaluable when you are writing adventures, as you can print out all your messages and not have to swap between messages and conditions all the time whilst looking for the right message number.

In addition, option *X* allows you to set up the program to recognize your printer's needs. Some printers need to be sent a *linefeed* signal to advance to the next line - others do it themselves, hence the *Auto-linefeed* question. If your printer does this itself, press *Y* for yes, otherwise *N* for no.

Many of the newer high quality printers are 24 pin dot matrix printers, so the next questions asks *24 pin ?* Again, answer *Y* or *N*. This is for the graphics printing - it uses a higher quality screen dump for 24 pin printers. **NOTE** - the graphics screen dump will only work if you have an *Epson* compatible printer. This is true of most printers which can handle graphics.

The new printer dump routine is also patched into the normal screen dump. This is activated by pressing *Alternate* and *Help* at the same time. The contents of the entire screen are then sent to the printer. If you do not have a printer and hit this option by accident, be patient. It takes the computer nearly a minute to realize that the printer is not attached and return to normal operation. Also note that since this feature was included primarily to print out pictures, it will not give a faithful reproduction of text from a medium resolution (80 character) screen, or a split mode screen when running an adventure.

The *Printer width* question is used to determine whether a picture will fit across the paper or not (if not, then it is printed along the paper). Try experimenting with different values for this - the higher the number, the wider the paper. Since it is in printer units, it will vary from printer to printer. I find a value of about 128 works well with an *Epson LQ800*, which is what I use.

The linefeed length is again used for graphics, and represents the distance between successive scans of the print head whilst printing graphics. If you get white stripes across your picture, try reducing it. Dark stripes suggest increasing this spacing.

Since control codes are widely used on screen, but can cause havoc on a printer, they are translated on printing to a more readable form. The abbreviations for each can be found in appendix B, along with what the control characters actually do on the screen.

CHAPTER EIGHT

THE FONT EDITOR

Using the STAC, it is possible to change the style in which text is printed on the screen (though not to the printer). This is known as the *font*, and the *font editor* can be called up from the main menu with key *F*.

There are 256 characters in the character set, including letters, numbers and punctuation. The first 32 are reserved, and are not actually printed. These are known as *control characters*, and cause different effects on the screen (see appendix B). The other 224, however, are printable, and they can be redesigned by you to add a personal touch to the adventure.

When you call up the font editor, the 256 characters are displayed on the left, and an 8 by 8 grid on the right. There are also three buttons at the bottom, marked *Exit*, *Save* and *Load*. By the grid are four rectangles of colour, which are black, red, white and blue.

You can draw in the grid using the mouse. Click on one of the coloured rectangles (not white for the moment - it will not show up...), and click in a square on the grid. It will be filled with the colour you selected. This will allow you to construct and change characters.

To see how a character is constructed, click on the character you want using the *left* mouse button. The grid will show a magnified view of the character, which you can then edit by selecting colours and clicking in the grid. This is easier to do than to explain. The character will also be displayed life size in the top left hand corner of the screen.

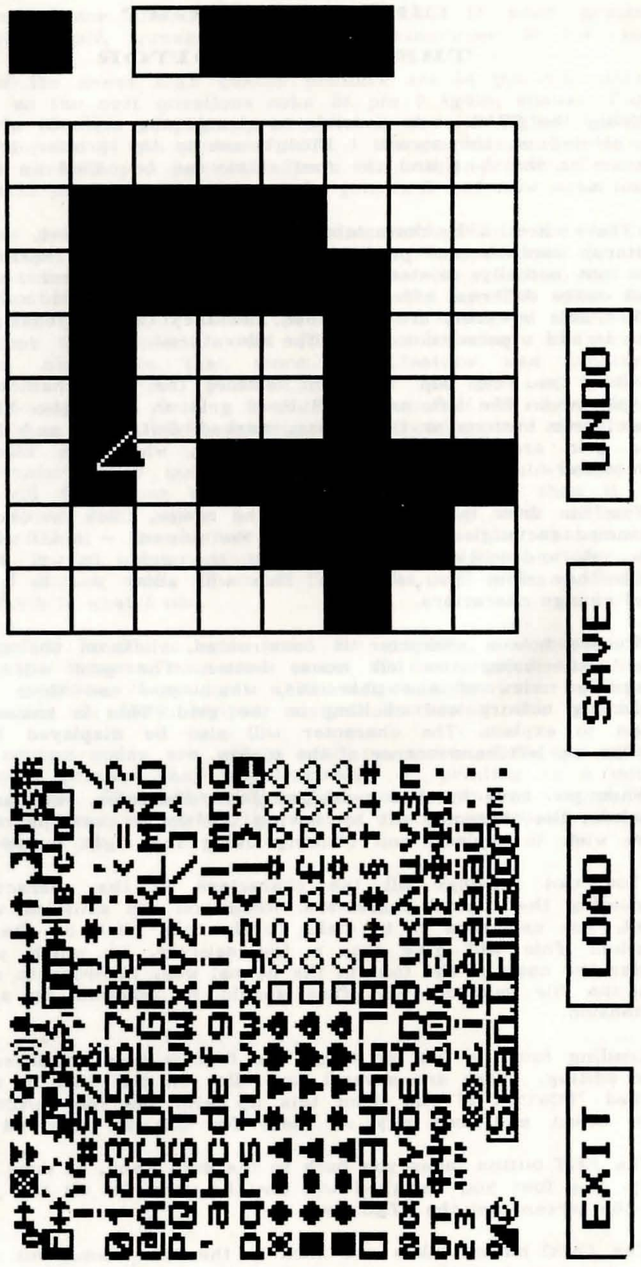
When you have finished redesigning the character, you can put it back in the character set by moving the mouse over the character you wish to replace, and clicking using the *right* mouse button.

You can redesign all the characters in the character set, including the control characters. When you are satisfied with the font, you can save it to disk. To do this, click on then *SAVE* button. This will give you a file selector, in which you can enter the name of the font in the normal way. In order to show up on the file selector, font files should be followed by a ".FNT" extension.

Loading fonts is just as easy. The font is then displayed ready for editing. There are several font files on the disk in a folder called "FONTS". If you open this up with the file selector, you can select and load any of these for use in your adventure.

The *EXIT* button takes you back to the main menu, as does the *Esc* key. The font you designed will now be used for all the printing to the screen that the STAC does.

The *UNDO* button takes you back to the main menu, but restores the previous font that you had, just in case you make a complete mess.



The Font Editor Screen

You may notice that the font is broadly divided into two sections: the first half, which you can for the most part see on the keyboard, and the second half, which you cannot. How do we type in the characters in the second half ? This is relatively straightforward. When typing in a message or a room description, for instance, pressing the **Insert** key will let you use the second half of the character set. If you press a key on the keyboard, it will now return the character directly underneath it by 8 lines on the character set (ie it adds 128 to the character code for those of you who do a little programming). For a full list of how to get all the characters see Appendix E.

To access control characters that also do things to the line you are typing (like *control-M* which acts like return), press *Shift-Help* and then the control character. *Shift-Help* means, in effect, "put the next character into the line no matter what it is". This is useful for inserting additional control characters into messages etc. for special effects.

Successful fonts follow some general rules. Firstly, try to make the letters look distinct from each other, or there is a danger of the text becoming unreadable (try loading *BLOCK.FNT* to see what I mean - all the letters look more or less the same). If all the vertical lines are double width, then this makes the font look darker and it is consequently more readable.

Don't be tempted to use the whole 8 by 8 grid for a character - if you do all the characters will butt up against one another and the result will again be unreadable.

You can put special characters in the top half of the character set since they can be typed in by pressing the *Insert* key. This could be useful for, say, *italics*, or **bold**, or maybe double width characters or monograms. You could even have mirror writing for puzzles ! The possibilities are endless, and by using four colours you can get some great effects.

CHAPTER NINE

SPECIAL CONDITIONS AND COMMANDS

SPECIAL CONDITIONS

The special conditions are a part of the **STAC** that can be loaded with the *QSTART* file and safely left alone. They control the adventure's responses to errors and other extraordinary situations. A full list of the default conditions can be found in Appendix D, but here I would like to show what each special condition does, and how you can tailor the responses to your own needs

There are 18 special conditions that are used by the system. You can use the others by using the *special* word in conditions, and they can be useful for often repeated sections in conditions.

Here are the first 18 special conditions.

1. Death

This condition is called when the user dies (ie when the *death* word is executed in a condition). It should at the very least exit the game, possibly giving a message as well.

2. Success

This is similar to 1, but is called when the player wins. Again, it should at least exit the player from the game.

3. Quit

This should ask the player whether he really wants to quit or not. If he answers yes, he should exit the game. Special conditions will always return to where they were called from unless they include *wait, ok, newcom,* or *byebye,* which will be executed as normal.

4. Ok

This is called when the *ok* word is executed. It should print a message "Okay", and *wait* for a new command.

5. Can't get an object that is not here

This is called when the player attempts to *get* an object that is not here. It should print a message to that effect, then either *wait* for a new command, or scrap the rest of the commands on the line and wait for the player to type a new one by using *newcom.* In the first case, if the player had typed a whole string of commands, then any after where he could not pick up the object will be executed. In the second case it gives him a chance to correct his mistake before continuing. This is a common thing to do in situations that are errors.

6. Carrying too much to get object

Is similar to 5, and is handled in the same way.

7. Already got that object

Is again similar to 5.

8. Haven't got that object to drop.

This is called when you try to drop an object you do not have, and is handled in much the same way as 5.

9. Dropall

This is what is actually executed when the **STAC** comes across a *dropall* command. It should include a loop that goes through each object that you are carrying and drops it, repeating until you are carrying nothing. The default action also gives a list of items dropped.

10. Getall

This is like dropall, cycling through all the objects in the room and attempting to pick them up. It will stop if you try to pick up too much.

11. You can't

This is called when you have asked the system to do something it does not recognize (ie it has gone through all the local and low priority conditions and has not recognized anything). It should just print a message "You can't" and then scrap the command line with *newcom.* Again, similar to 5.

12. Pardon ?

This is called if you enter a command containing no verbs that are recognized. Yet again, another one similar to 5.

13. What now ?

This is called when a prompt is required to tell the player he is expected to give a command. The default also prints a status line at the top of the screen showing where you are, your score, and the number of moves taken.

14. Look

This is called when the *look* word is executed. It should at least display the long description of the room you are in, and a list of any objects that are there, or a message to show that it is dark if marker 1 is set (ie it is actually dark) and marker 2 is reset (ie you do not have a lamp).

15. Describe room on entry

When moving from one room to another, you will want to be given a description of the room you have just entered. This is what this condition is for. The default action is to check if it has been visited, and if so only print the short description, otherwise to print the long description. The room is then marked as having been visited. Again, if dark, a suitable message is printed.

16. Strength reduced to below load

When using *setstr*, it is possible to set your strength to 1 whilst carrying a combined weight of 10. If this happens, this condition is called. The default is to drop items one by one until the amount you are carrying is less than or equal to your strength.

17. Start up

This is called once, at the start of the adventure, and is used to set up things. It is too specific to be included in *QSTART*, but typically it will be used to set your strength, ask for your name if necessary, set light or dark markers, and moves-in-the-dark counters.

18. Load prompt

This is called if *load* cannot find the link file it needs on the current disk. This condition should print a prompt to tell the player to change disks, and then press a key. It should also wait for the keypress !

COMMANDS

The commands that the player enters are processed by the **STAC** so that they can be passed to the conditions in usable form. This process is quite complex, but will allow you to see how a player command is interpreted, and thus what sort of commands you can use.

Let us take a very small vocabulary:

NOUNS	VERBS	ADVERBS
1 lamp	1 take	1 green
2 box	2 put	2 red
255 it		3 in
		254 ,

and look at the player's command

Take the lamp, put it in the green box

The **STAC** goes through the command one word at a time, checking through the verb, noun and adverb tables, and filling in the recognized numbers in *verb1, noun1, noun2, adverb1* and *adverb2.* If it comes across an adverb with number 254 or 255, then it stops and saves its position, having got a complete command. Let's see what happens:

```
Word       Take      the        lamp        ,           put
Verb?      YES       NO         NO          NO
Noun?       |        NO         YES         NO
Adverb?     |        NO          |          YES
            |                    |           |
       verb1=1     ignore     noun1=1    254 so end command
```

So we have: verb1 = 1 TAKE
noun1 = 1 LAMP

as our first command. Having dealt with this, we come back to the next word in the line:

```
Word       put        it          in        the       green
Verb?      YES        NO          NO        NO        NO
Noun?       |         YES         NO        NO        NO
Adverb?     |          |          YES       NO        YES
            |          |           |                   |
       verb1=1    noun1=255    advb1=3    ignore    advb2=2
                  becomes 1
```

```
Word       box
Verb?      NO
Noun?      YES
Adverb?     |
        noun2=2
```

Any occurrence of noun 255 (*it*) is immediately replaced by the last noun referred to (in this case noun 1, the lamp).

```
Thus the command is: verb1 = 2    PUT
                     noun1 = 1    LAMP
                     noun2 = 2    BOX
                     advb1 = 3    IN
                     advb2 = 2    GREEN
```

We then come to the end of the line, so processing stops and the command is passed through to the conditions. Since there are no more commands on this line, the next time through a new line will be requested from the player.

The difference between adverbs with numbers 255 and those with 254 is a subtle one. In a command which was separated by a 254 adverb, if there is no verb in the command then the one from the previous command is used. This allows such things as (assuming adverb *and* had number 254)

get the lamp and the rat

since it preserves the *get* from one part to the next. Of course, if there is a verb in the next part, the previous verb is abandoned. It does, however, add a useful additional touch to the command interpreter.

In addition to all this, a verb with number 255 is recognized as an "again" feature, and recalls the relevant information about the last command. Even if your last command was *throw the green pot at the monster*, a verb 255 *again* will repeat it. This can be very useful.

APPENDIX A

EDITING KEYS AND CONTROL CHARACTERS

KEYS USED WHEN TYPING

If you make a mistake, or leave something out when typing a line into the STAC, then the following keys will help you to correct it. The *cursor* is a black rectangle which shows where your typing will appear. If it is before something, then what you type will be inserted, shuffling the rest of the text up rather than overwriting it.

Key	Action
Backspace	Erase character to left of cursor
Delete	Erase character under cursor
Arrow Keys	Move cursor up,down,left,right if possible
Shift-Up	Move to start of what you typed
Shift-Down	Move to end of what you typed
Shift-Left	Move to left hand screen edge
Shift-Right	Move to right hand screen edge
Home	Move to start of what you typed
Shift-Home	Clear everything after cursor
Shift-Undo	Clear everything you typed
Insert	Toggle between first and second halves of character set (press again to cancel)
Shift-Help	Insert next character no matter what it is

CONTROL CHARACTERS FOR USE IN MESSAGES

These are accessed by holding down the *Control* key and then pressing the letter key indicated. Note that some of these mimic the actions of certain keys (these are given in *Italic* after the description of the key), and should thus be preceded by pressing *Shift-Help* before typing them in. Also, you can quite legally include any of the characters marked "ignored" if you like, but as it says, they will be completely ignored. They will not even leave a space.

Character	Prints as	Action taken
Cntrl-A	[TAB4]	Tab across by 4-character tab
Cntrl-B	[CT-B]	Ignored
Cntrl-C	[CT-C]	Ignored
Cntrl-D	[CT-D]	Ignored
Cntrl-E	[BUZZ]	Sound a buzz
Cntrl-F	[BEEP]	Sound a beep
Cntrl-G	[PING]	Sound a ping
Cntrl-H	[LEFT]	Move left one character *Backspace*
Cntrl-I	[RIGH]	Move right one character *Tab*
Cntrl-J	[DOWN]	Move down one character
Cntrl-K	[UP]	Move up one character
Cntrl-L	[CLRS]	Clear the screen *Shift-Home*

Key	Prints as	Action
Cntrl-M	[CR]	Move to left edge of screen *Return/Enter*
Cntrl-N	[SAVE]	Save this cursor position
Cntrl-O	[REST]	Move back to saved position
Cntrl-P	[CT-P]	Ignored
Cntrl-Q	[CLRL]	Clear this line
Cntrl-R	[LARG]	Large window size enabled The large window size covers the entire screen
Cntrl-S	[SMAL]	Small window enabled The small window size covers the area below the picture if there is one.
Cntrl-T	[TUNE]	Tune starts here
Cntrl-U	[TEXT]	Tune ends here
Cntrl-V	[STR0]	Print string 0
Cntrl-W	[STR1]	Print string
Cntrl-X	[INVS]	Inverse on/off
Cntrl-Y	[NORM]	Inverse off
Cntrl-Z	[TB10]	Tab to 10 characters short of right margin
Cntrl-[	[ESC]	Ignored *Esc*
Cntrl-\	[CT-\]	Ignored
Cntrl-]	[CT-]]	Ignored
Cntrl-_	[HOME]	Home cursor to top left corner *Home*
Cntrl-?	[CT-?]	Ignored

APPENDIX B

GLOSSARY OF TERMS

This section gives the meanings of some of the less common phrases and words in the manual.

ADVENTURE UNIVERSE - is the setting for the adventure, and includes all the locations and objects that the player will visit or use.

ADVENTURER - simply the person playing the adventure.

AUTHOR - the person who actually wrote the adventure.

BRUSH - on the screen, the shape you are drawing with. This need not look anything like a "real" brush !

CHARACTER - any number, letter, piece of punctuation, or symbol that the computer can display.

CHARACTER SET - the sum total of all the characters the computer can display.

CHORD - playing more than one note at once (in music).

CLICK - to click on something, move the mouse over it and hit the left-hand mouse button quickly.

COMMAND - anything the player types to get the adventure to do something. Typical commands are "Go north" or "Get the gold".

COMMAND LINE - what the player types in. It may contain several commands, separated by punctuation or words like *and* or *then*.

COMMAND INTERPRETER - the part of the **STAC** that extracts meaning from the player's commands. Also known as a "parser". The **STAC** parser can make sense of multi- part commands and complex actions such as: "Get the screwdriver, the axe and the dog then hit the log with the axe and examine the dog."

CONDITIONS - are the part of the adventure which make the decisions. See chapter 3 for a full explanation.

CONNECTIONS - the means by which different rooms are connected to each other.

CONTROL CHARACTER - is a character which is not printed on the screen, but which instead controls some action on the screen, such as clearing it or moving on to the next line. These are for the most part typed by holding the **Control** key down and typing a letter.

CURSOR - a small rectangle which shows where the next thing you type will appear.

CURSOR KEYS - the four keys with arrows on to the right of the main keyboard. They move the cursor in the direction of the arrow on the key.

DEFAULT - the state something is in before you do anything to it. The default message number, for instance, is one more than the last one you used.

DOUBLE CLICK - to click twice very quickly.

ELLIPSE - a squashed circle; an oval.

EDIT - changing information about something is called editing.

ENTER - when asked to enter information, type the information and then press either the *Enter* key or the *Return* key.

FILE - everything stored on a disk is in the form of files. Each one stores a separate set of information, much like files in a normal filing cabinet.

FILENAME - the name of a file on the disk. This consists of two parts: a name and a type. The name can be up to 8 letters long, and the type up to 3 letters. They are separated by a dot. It is customary to give files that hold the same sort of information the same type (also called its extension). An example is SANSERIF.FNT , whose type of ".FNT" proclaims it as a font file.

FILE SELECTOR - is a useful tool for selecting files to save on to disk, or to load from it. Rather than having to remember names of files and type them in, the file selector allows you to pick one from the disk using the mouse. It also handles opening and closing folders, and changing disks.

FONT - when printing characters on the screen, the font specifies exactly how they look.

FOLDER - files can be grouped together in folders for convenience. Rather than having to look through a whole load of files for the one you want, you can just open the relevant folder. All the font files on the **STAC** disk come in a folder called FONTS to keep them all together. See also "pathname".

GAC - short for Graphic Adventure Creator, an adventure writing system for many 8-bit computers, upon which the **STAC** was based. See also "**STAC**"

GRAPHICS - pictures, as opposed to text.

ICON - a small picture representing a function to be performed. To "do" the function, click on the icon with the mouse. These are used mostly in the graphic editor - see also chapter 4.

IMPORT - use a picture which has been produced using a different program (either **NEOCHROME™** or **DEGAS™**).

INTELLIGENCE - is generally the decision making property of an adventure, rather than its ability to think. We are still a very long way from a true "thinking" computer program.

INVENTORY - a list of objects, usually the ones you are carrying.

LINESTYLE - how a line is drawn. Examples are dotted, dashed, continuous.

LOOP - a programming term for something which is done over and over again. It is called a loop because on flowcharts (a convenient programming aid consisting of little boxes connected to each other with squiggly lines), the lines all join up in a loop. Infinite loops just go round and round forever - for an example of this see "LOOP".

MAGIC - a convenient cheat for letting the adventurer do something very difficult (if not impossible).

OBJECTS - generally, anything in an adventure which you can pick up and move.

OPERANDS - the pieces of information that are manipulated by operators.

OPERATOR - (in conditions) is a word that does something to operands. For example, in *1 + 2*, the operands are 1 and 2, and the operator is +, which simply adds them together.

PALETTE - the set of colours that you can use on the screen. You can use 16 colours chosen from the 512 available on the ST.

PARSER - see "command interpreter".

PATHNAME - this specifies in which folder(s) a file may be found. A pathname that looks like A:\FONTS\TIMES.FNT means "file TIMES.FNT in folder FONTS on disk A". See also "filename", "folder".

PIXEL - the smallest dot you can produce on the screen.

PLAYER - the person who plays the adventure.

RANDOM - an action for which you do not know the outcome. The action of rolling a dice provides a random number between 1 and 6.

RUBBER BAND - not a pop group ! A rubber band object is one that can be stretched using the mouse until it is the proper shape.

SPLIT MODE SCREEN - the ST can display three screen modes, usually referred to as high, medium, and low resolution. The **STAC** allows a split screen so that you can have full colour low-resolution pictures in the top half, and 80 character medium resolution text in the bottom part of the screen, something which is not usually possible.

STAC - short for the ST Adventure Creator.

STRING - a string is a sequence of characters (letters or numbers). It may be up to 39 characters long. Here are some examples of strings: "Hello", "99-64", "Aardvarks are not common in Iceland".

TELEPORT - to move from point A to point B instantly without going through any point in between. Star Trek and Larry Niven fans will know all about teleporting.

TEMPO - how fast a piece of music is played. Most music by Motorhead, for example, has a faster tempo than that favoured by Val Doonican.

TEXT - words and numbers, as opposed to graphics.

USER - the person using the **STAC**, as opposed to the adventurer.

VOCABULARY - the sum total of all the words that the adventure game will respond to.

VOLUME - how loud a piece of music is played. Again, this might merit another Motorhead / Val Doonican comparison.

APPENDIX C

CONTENTS OF QSTART FILE

The *QSTART* file supplied with the STAC contains many things which are used in almost all adventures, no matter what the story or the precise nature of the adventure.

Here is a list of exactly what you get when you load the *QSTART* file, with additional comments in *italics*.

ADVERBS

254	,	*Adverbs with number 254/5 are taken to be*
255	.	*separators between different commands on*
255	?	*the same line.*
255	!	
254	and	
255	then	

LOW PRIORITY CONDITIONS

1 if verb "drop" and noun "all" then dropall wait
2 if verb "get" and noun "all" then getall wait
Get and drop all objects.
3 if verb "l" then look wait
Describe room when player types "look".
4 if verb "i" and notzer? cntobj with then message 9916 list with wait
Inventory when you are carrying something.
5 if verb "i" and zero? cntobj with then message 9916 message 9917 wait
Inventory when you are carrying nothing.
6 if verb "quit" then quit ok
7 if verb "text" then text draw 0 ok
8 if verb "graphics" then pict draw pictof room ok
9 if verb "save" then save ok
10 if verb "load" then load look wait
11 if verb "split" then split lf message 9932 wait
12 if verb "ramsave" then ramsave 1 ok
13 if verb "ramload" then ramload 1 look ok

MESSAGES

9900	You have died	*When the player dies*
9901	Well done !	*When the player wins*
9902	You scored a total of	
9903	points in	
9904	moves.	*To show the score*
9905	Okay	*Message printed for Ok.*
9906	I can't see that !	*Can't find object to get.*
9907	You are carrying too much already.	
9908	Why? You've already got it.	*When you try to get something you already have.*
9909	You don't have that.	*Drop something you don't have.*

MESSAGES ctd

9910	You can't do that !	
9911	Pardon ????	
9912	What now ?	
9913	You can also see	*When listing objects in a room.*
9914	.	
9915	Are you sure (Y/N) ?	*Ask when quitting*
9916	You are carrying	*For inventory*
9917	nothing.	
9918	You feel weak, and stumble.	*When strength reduced too much to carry load.*
9919	You drop	*For dropall*
9920	taken.	*For getall*
9921	You can't examine that, I'm afraid.	
9922	You see nothing special	*For examining things*
9923	[SAVE][LARG][HOME][INVS][CLRL]	*Move to top line, make it black with white letters.*
9924	[TB10]	*Move across to print score*
9925	/	*Between score and turns*
9926	[NORM][SMAL][REST]	*Go back to cursor position*
9927	Dark in here, isn't it?	*For when you enter a dark room.*
9928	It's no use looking, it's dark !	
9929	You don't have anything to drop.	
9930	There's nothing here to pick up.	
9931	:	
9932	Text size changed.	
9933	Please insert disk containing	

NOUNS

254	all	
254	everything	
255	it	*"it" must always be noun 255*
255	them	

SPECIAL CONDITIONS

1. Death
1 lf message 9900 lf
print "You have died"
2 message 9902 print counter 0 message 9903 print turns message 9904 pause 5000 byebye
print score and number of turns, wait for keypress, then end the game

2. Success
1 lf message 9901 lf
print "Well done"
2 message 9902 print counter 0 message 9903 print turns message 9904 pause 5000 byebye

3. Quit

1 message 9915 if yesno then lf message 9902 print counter 0 message 9903 print turns message 9904 pause 5000 byebye

4. Ok

1 message 9905 wait

print "Okay", wait for new command

5. Get failed - object not here

1 message 9906 newcom

print "I can't see that", scrap rest of command line, wait for new command

6. Get failed - carrying too much

1 message 9907 newcom

7. Get failed - already got object

1 message 9908 newcom

8. Drop failed - not carrying object

1 message 9909 newcom

9. Dropall

1 if zero? firstob with then message 9929 newcom

If not carrying anything, say so.

2 if firstob with then repeat lf message 9919 itis firstob with objsht firstob with drop firstob with until zero? firstob with

A little complex, this one. Print "You drop ", then describe the first object you have with you, set "it" to refer to this object, and drop it. Repeat until all objects have been dropped.

10. Getall

1 if zero? firstob room then message 9930 newcom

2 if firstob room then repeat lf caps objsht firstob room itis firstob room message 9931 get firstob room message 9920 until zero? firstob room

Similar to above. Describe object you are about to pick up, then make "it" refer to it. Attempt to get the object (this may fail if you are carrying too much already).If it does not fail then print "taken", and repeat until there are no objects left in the room.

11. You can't !

1 message 9910 newcom

12. Pardon ??

1 message 9911 newcom

13. What now ?

1 lf message 9912

print "What now ?"

2 message 9923 descsht room message 9924 print counter 0 message 9925 print turns message 9926

print the status bar at the top of the screen.

14. Look

1 if set? 1 and reset? 2 then message 9928 return

if it is dark and you have no source of light, print "It's dark" and return to where we were.

2 set 0 desclng room draw pictof room

otherwise print room description and picture if any, and set marker 0 as confirmation of this.

3 if firstob room then message 9913 list room message 9914

if there are any objects here, list them

15. Describe room when entering it

1 lf

2 if set? 1 and reset? 2 then message 9927 return

test if it is dark as above

3 set 0

4 if visit? then descsht room draw pictof room else desclng room draw pictof room

if this room has already been visited, use the short description, otherwise use the long one.

5 visit

mark this room as having been visited

6 if firstob room then message 9913 list room message 9914

list any objects that are here

16. Strength reduced to below load

1 message 9918

print "You stumble"

2 repeat lf message 9919 objsht firstob with drop firstob with until stren? >= amount newcom

drop objects until the amount carried is less than your strength.

17. Start of adventure

Not included - adventure start is very specific

18. Load prompt

1 lf message 9933 asslink 0 pause 5000

print "Please insert disk containing " then link file name, then wait for keypress.

VERBS

255	a	9	l	2	s
255	again	10	list	14	save
6	d	15	load	2	south
6	down	9	look	17	split
8	drop	1	n	17	t
3	e	1	north	7	take
3	east	13	pictures	12	text
16	exam	11	quit	17	textsize
16	examine	18	ramload	5	u
7	get	19	ramsave	5	up
13	graphics	14	restore	4	w
10	i	18	rl	4	west
10	inventory	19	rs	12	words

APPENDIX D

CHARACTER KEY SEQUENCES

When using the Font Editor, it can be seen that there are considerably more characters than are actually displayed on the keyboard. How do you type these in ?

This generally requires use of the insert and shift-help keys mentioned in appendix B. Here is a list of how to get all the characters. Since they may change in appearance from font to font, they are organized by row and column as they appear on the font editor.

In order to compact the list, the following keys have been abbreviated:

```
Control     = Cnt          - means "with" ( eg ! is Shf-1 )
Insert      = Ins          / means "then" ( eg row 1 column
Shift-Help  = Hlp            9 is Hlp/Backspace )
Shift       = Shf
```

Row	Col	Key sequence
1	1	Not available
1	2	Cnt-A
1	3	Cnt-B
1	4	Cnt-C
1	5	Cnt-D
1	6	Cnt-E
1	7	Cnt-F
1	8	Cnt-G
1	9	Hlp/Backspace
1	10	Cnt-I
1	11	Cnt-J
1	12	Cnt-K
1	13	Hlp/Cnt-L
1	14	Hlp/Return
1	15	Cnt-N
1	16	Cnt-O
2	1	Cnt-P
2	2	Cnt-Q
2	3	Cnt-R
2	4	Cnt-S
2	5	Cnt-T
2	6	Cnt-U
2	7	Cnt-V
2	8	Cnt-W
2	9	Cnt-X
2	10	Cnt-Y
2	11	Cnt-Z
2	12	Esc
2	13	Cnt-\
2	14	Cnt-]
2	15	Hlp/Home
2	16	Hlp/Cnt- -
3	1	Space bar
3	2	Shift-1
3	3	Shift-2

Row	Col	Key Sequence
3	4	#
3	5	Shf-4
3	6	Shf-5
3	7	Shf-7
3	8	’
3	9	Shf-9
3	10	Shf-0
3	11	Shf-8
3	12	Shf-=
3	13	,
3	14	-
3	15	.
3	16	/
4	1	0
4	2	1
4	3	2
4	4	3
4	5	4
4	6	5
4	7	6
4	8	7
4	9	8
4	10	9
4	11	Shf-;
4	12	;
4	13	Shf-,
4	14	=
4	15	Shf-.
4	16	Shf-/
5	1	Shf-’
5	2	Shf-A
5	3	Shf-B
5	4	Shf-C
5	5	Shf-D
5	6	Shf-E

Row	Col	Key sequence
5	7	Shf-F
5	8	Shf-G
5	9	Shf-H
5	10	Shf-I
5	11	Shf-J
5	12	Shf-K
5	13	Shf-L
5	14	Shf-M
5	15	Shf-N
5	16	Shf-O
6	1	Shf-P
6	2	Shf-Q
6	3	Shf-R
6	4	Shf-S
6	5	Shf-T
6	6	Shf-U
6	7	Shf-V
6	8	Shf-W
6	9	Shf-X
6	10	Shf-Y
6	11	Shf-Z
6	12	[
6	13	\
6	14	]
6	15	Shf-6
6	16	Shf- -
7	1	‘
7	2	A
7	3	B
7	4	C
7	5	D
7	6	E
7	7	F

Row	Col	Key sequence
7	8	G
7	9	H
7	10	I
7	11	J
7	12	K
7	13	L
7	14	M
7	15	N
7	16	O
8	1	P
8	2	Q
8	3	R
8	4	S
8	5	T
8	6	U
8	7	V
8	8	W
8	9	X
8	10	Y
8	11	Z
8	12	Shf-[
8	13	Shf-\
8	14	Shf-]
8	15	Shf-#
8	16	Hlp-Delete
9	..	Insert, see row 1
10	..	Insert, see row 2
11	..	Insert, see row 3
12	..	Insert, see row 4
13	..	Insert, see row 5
14	..	Insert, see row 6
15	..	Insert, see row 7
16	..	Insert, see row 8

For rows 8 to 16, press Insert, then follow the key sequence 8 rows above (eg row 16 column 1 would be Insert then row 8 column 1 which is P giving the sequence Insert/P).

APPENDIX E

CONDITIONAL WORDS

For a full description of the words in each section, see also the page numbers indicated.

DECISION MAKING [p9,24]

if...then...else	Tests if a condition is true or false. If true, performs action after then, otherwise performs that after else, if present.

PLAYER COMMANDS [p4,9,21,22,25,35,50,64]

verb v	Was verb v typed ?
noun n	Was noun n typed ?
adverb a	Was adverb a typed ?
noun1 noun2	Return numbers of first and second nouns typed by the player.
advb1 advb2	Return numbers of first and second adverbs typed.
verb1	Return number of verb typed.
itis n	Force "it" to refer to noun n.

ROOMS [p13-15,39]

goto r	Move player to room r and redescribe.
moveto r	Move player to room r.
desclng r	Print long description of room r.
descsht r	Print short description of room r.
look	Print description of this room, with picture if present, using short description if already visited.
draw p	Draw picture number p.
pictof r	Get picture number associated with room r.
split	Turn split mode screen on and off.
c colour xxx	Change colour c in bottom part of screen to RGB value xxx.
c topcol xxx	Change colour in top part of screen.
visit	Mark this room as already visited.
visit?	Has this room been visited before ?
room	Get number of this room.
at r	Is the player at room number r ?

COMMENTS [p21]

; \	Either of these signals that the rest of the line is to be treated as a comment and ignored.

OBJECTS [p10-12,22-25,48-49]

get o	Get object o.
drop o	Drop object o.
getall	Get every object in this room.
dropall	Drop every object you are carrying.
o to r	Move object o to room r.
o swap O	Swap object o and O.
bring o	Bring object o here.
find o	Move player to object o.
objlng o	Print long description of object o.
objsht o	Print short description of object o.
o in r	Is object o in room r ?
carried o	Is object o being carried ?
here o	Is object o in this room ?
avail o	Is object o available (ie. here or being carried) ?
weight o	Return the weight of object o.
whereis o	Return location of object o.
stren?	Return your current strength.
amount	Return the total weight of everything you are carrying.
setstr e	Set player's strength to e.
firstob r	Return number of first object in room number r.
cntobj r	Return the number of objects in room number r.
list r	List all objects in room r.
list with	List all objects with you (being carried).

MARKERS [p15,38]

set m	Set marker m.
reset m	Reset marker m.
change m	Change state of marker m (ie if it was set before then reset it, and vice versa).
set? m	Is marker m set ?
reset? m	Is marker m reset ?

COUNTERS [p16,24,38]

e setcntr c	Place value e in counter c.
counter c	Return value stored in counter c.
inc c	Add one to value of counter c.
dec c	Subtract one from counter c.
e +count c	Add e to counter c and put the value back in the counter.
e -count c	Subtract e from counter c.
e =count c	Is the value e the same as the value in counter c ?

COMBINATIONS [p23]

t and T	Are both t and T true ?
&	
&&	(Both same as and)
t or T	Is either t, or T, or both, true ?
¦	
¦¦	(Both same as or)
t xor T	Is either t, or T, but not both, true?
^	
^^	(Both same as xor)
not t	Is t false ?
~	(Same as not)

ARITHMETIC [p11]

e + E	Return result of adding e to E.
e - E	Return result of subtracting e from E
e * E	Return result of multiplying e by E.
e / E	Return result of dividing e by E. Note that this returns whole numbers only. So 9 / 2 is 4, not 4.5 .
e mod E	Returns the remainder when e is divided by E. 9 mod 2 is 1.

COMPARISONS [p11-12]

e < E	Is e less than E ?
e > E	Is e greater than E ?
e = E	Is e equal to E ?
e <= E	Is e less than or equal to E ?
e >= E	Is e greater than or equal to E ?
e <> E	Is e not equal to E ?
zero? e	Is e zero? (e = 0)
pos? e	Is e positive ? (e > 0)
neg? e	Is e negative ? (e < 0)
notzer? e	Is e not zero ? (e <> 0)
notpos? e	Is e not positive ? (e <= 0)
notneg? e	Is e not negative ? (e >= 0)

LIFE and DEATH [p16,38,39,48]

death	Kill the player and stop the game.
success	End the game since the player has succeeded.
quit	Ask the player if he is sure he wants to quit, and if yes, end the game.
byebye	Exit from the game immediately.

DISKS [p17,18,50]

save	Save game position to disk.
load	Load a previously saved game position.

DISKS (ctd.)

ramsave n	Save game position to memory slot n.
ramload n	Load a previously ramsaved position.
link m	Load and play an extension file. Note that this will only actually work in a runnable adventure. The filename to load is put in message m.
asslink	Prints the filename of the expected link file.

MESSAGES and STRINGS [p18-21,36]

message m	Print message m to the screen.
caps	Make sure first character of next message is a capital letter.
get$ s	Get string s from the player.
print$ s	Print string s to the screen.
edit$ s	Allow player to edit string s.
value s	Get numeric value of string s.
n number$ s	Put numeric value n in string s.
s add$ S	Add string s on to end of string S.
s copy$ S	Copy string s to string S.
s swap$ S	Swap strings s and S.
m mess$ s	Copy message m into string s.
n cutst$ s	Cut n characters off start of string s.
n cutend$ s	Cut n characters off end of string s.
length$ s	Return length of string s.
c addchr$ s	Add character c to end of string s.
c first$ s	Find first occurrence of character c in string s.
c last$ s	Find last occurrence of character c.
ascii$ s	Get character code of 1st character of string s.
obey$ s	Obey string s as if the player typed it in.
parse$ s	Fill in additional nouns, verbs and adverbs from string s.
comm$ s	Put the rest of the command line into string s.
s =$ S	Is string s equal to string S ?
s <$ S	Is string s less than string S ?
s >$ S	Is string s greater than string S ?
s <>$ S	Is string s not equal to string S ?
s <=$ S	Is string s less than or equal to string S ?
s >=$ S	Is string s greater than or equal to string S ?
print e	Print number e.

MISCELLANEOUS

wait	Wait for a new command. [p10]
ok	Print "Ok", wait for a new command [p10,48]
newcom	Ask player for new command line. All commands not already done on the current line are discarded. [p23]
turns	Returns the number of turns since the start of the game. [p21-23]
false	Returns the same result as a false test. [p23]
true	Returns the same result as a true test. [p23]
yesno	Waits for the user to press "Y", which returns a true result, or "N", which returns false. [p22]
repeat..until t	Repeats conditions between repeat and until until condition t is true.[p24]
lf	Move printing on to a new line. [p13]
text	Disable pictures. [p14]
pict	Enable pictures. [p14]
e word n	Set word n to be equal to value e. The value of n may be 1 (noun1), 2 (noun2), 3 (verb1), 5 (adverb1) or 6 (adverb2). [p22]
connect v	Gives the number of the room which is connected to this room by verb v. [p14]
random e	Gives a random number between 1 and e inclusive. [p21]
special s	Executes special condition number s. [p23,24]
return	Return early from special condition. [p23]
pause e	Pause for e fiftieths of a second, or until a key is pressed. [p18,23]
setamnt e	Set amount to value of e. [p22]
setturn e	Set turns to value of e. [p22]
setwith e	Set with to value of e. [p22]
cursor e	Sets the height of the text cursor to e. e may be 0 to 7.

ERRORS

1. *Number out of range*
Occurs when you try to access counters, markers, or objects with numbers greater than 511, and when you try to use nouns, verbs, or adverbs with numbers greater than 255.

2. *Internal error*
Will only happen if a very complex expression is being evaluated and STAC runs out of space to store its results. This should never happen, as very complex is really *very* complex !

3. *Object not found*
Occurs when you try to describe an object which does not exist.

4. *Room not found*
Occurs when you try to describe a room which does not exist.

5. *Message not found*
Occurs when you try to print a message which does not exist.

6. *Break*
Occurs when both shift keys are depressed at the same time whilst executing the conditions.

7. *Not a valid link file*
Occurs when the file whose name was given to *link* is not actually a previously saved link file. Also occurs when testing an adventure - actually loading a link file would destroy all your carefully typed adventure data !

8. *No repeat for this until*
Occurs when an *until* is found which does not have a matching *repeat* before it.

DISK ERRORS

Several errors can occur whilst trying to access the disks. Some of these are not obvious, so here is a list:

1. *File not found*
There is not a file with the requested name on the disk.

2. *Pathname not found*
You have tried to enter a folder that is not there. Typing a pathname of "A:\FONTS\SANSERIF.FNT" effectively opens a folder called FONTS. If there is no folder on the disk with that name, then this error will occur.

3. *Disk write protected*
The disk you are trying to save to has its write protect tab removed.

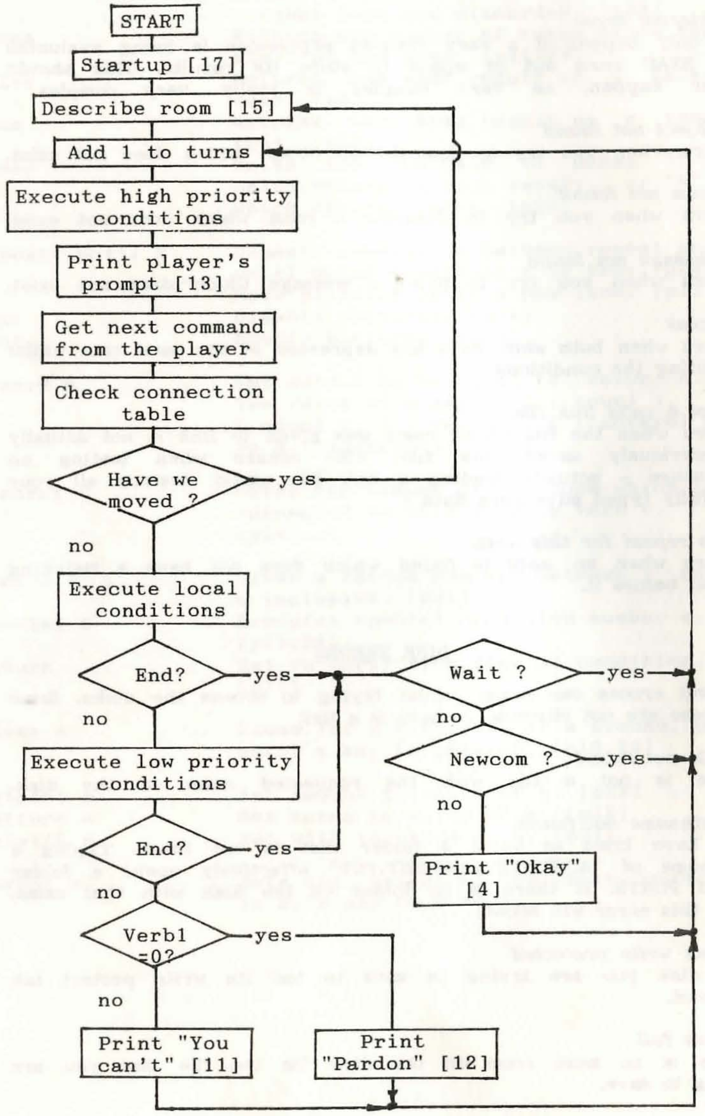
4. *Disk full*
There is no more room on the disk for the file that you are trying to save.

The warning box for all these file errors can be cleared by pressing a key.

APPENDIX F

ADVENTURE FLOWCHART

This is a flowchart showing what goes on when you play an adventure.



The numbers in [] brackets are special condition numbers.

APPENDIX G

HANDY REFERENCE SHEET

Ranges of numbers

Rooms: 1 to 9999
Messages: 1 to 9999
Pictures: 1 to 9999
Objects: 1 to 511

Special conditions: 1 to 255

Verbs: 1 to 255
Nouns: 1 to 255
Adverbs: 1 to 255

Markers: 0 to 511
Counters: 0 to 511
(which store -2,147,483,648
to +2,147,483,647)

Colours on screen: any 20 from 512
(16 in graphics screen, 4 in text)

Maximum picture size: 288 pixels horizontally
x 132 pixels vertically

Special Conditions

1. Death
2. Success
3. Quit
4. Ok
5. Can't get object (not here)
6. Can't get object (too heavy)
7. Can't get object (already got)
8. Can't drop object
9. Drop all
10. Get all
11. Print "You can't"
12. Print "Pardon ?"
13. Print "What now?"
14. Look
15. Describe room on entry
16. Strength reduced below load
17. Startup condition
18. Load prompt

Markers etc. used by STAC

Marker 0: Set when room described.
Marker 1: Set means dark.
Marker 2: Set means lamp available.

Counter 0: Score.

Noun 255: It
Verb 255: Again
Adverb 254/255: separators

INDEX

Item Page

Adventure Universe 3
Adverbs 9,64,71
Again 22,51
And,or,xor,not 66
Arithmetic 12,66

Begin where ? 39

Characters 45,58,68
Chords 33
Colour palette 30
Combinations 66
Commands 50
Comparisons 11,12,21,23,66,67
Conditions 8,64
.....high priority 37; local 38;
low priority 36,58;
special 48,59,70,71.
.....arithmetic 11,66; commands 9,64;
comments 21,64; disks 16,66;
markers/counters 15,65;
messages 12,67; objects 10,65;
pictures 14,64; rooms 13,64;
strings 18,67; testing 9,64;
.....other 21,68.
Connections 4,5,14,68
Contents of package 1
Control characters 45,58
Counters - conditional words 16,24,65
Counters - general 38-40,65

Dark marker 5,37,38,71
Decision making 9,24,64
Demonstration disk 1
Disk menu 6,42
Disks 6,40-43,55
Drawing screen 28
Dropping objects 10,25

Eating 37,39
Editing data 6,64,68
Encode 18,50,61
Equipment 1
Errors 26,40,57
Examine objects 37,39

Files 43
File selector 6,42
Flowchart 70
Folder 43
Fonts 45
Font editor 45,46
Format disk 42

Get objects 10,25
Getall 65
Glossary 60
Graphics 14,27

Help screen 40
High priority conditions 37

If-then-else 9,24,64
Importing pictures 29
Initialization 30,50,51
Inventory 13
It 51
Itis 64

Keys 58,68

LF (line feed) 68
Life and death - conditional words 16,21,23,66
Life and death - general 38,39,48,59,60,66
Light 5,37,38,71
Lighting a lamp 5
Link files 18,67
Load adventure 40
Load game position 16
Loading screen 39
Loading **STAC** 1
Local conditions 38
Loops 24,40,56,68
Low priority conditions 36,64

Main menu 6
Map 4
Markers - conditional words 15,26,65
Markers - general 5,38-40,65,71
Messages - conditional words 12,13,18-21,67
Messages - general 34,53,56,58,67
Mistakes 7,58
Music 32

Nesting 24
Newcom 68
Nouns 9,65,71
Numbering 25

Objects - conditional words 4,5,10-12,14,22-25,64
Objects - general 3,34,37,39,40,48-50,60,61,64
Ok 10,48,68
Oops 25

Palette 30
Pause 23,68
Pardon 49,60
Pictures 14,27
Plot 3
Printer 43
Program disk 1
Purpose of adventure 3

Questioning the player 21
Quick adventure 3
Quick start file 6,16,35,48,58

Ranges of numbers 71
Reference sheet 71
Repeat-until 24,68
Return 23,68
Rooms - conditional words 13-15,65
Rooms - general 34,38,39,49,56,61,65
Royalties 41
Runnable adventure program 40,41

Save adventure 40
Save game position 17,18,21,25,26,66
Scale 33
Scoring 36,37
Screen display 52
Screen dump 44
Separators 50,51,58,71
Short cuts 25
Special conditions 48,65,70,71
Starting the adventure 39,40
Strings - conditional words 12,13,18-21,67
Strings - general 34,53,56,58,67

Testing 40
Text 14,68
Tricks and tips 25
Turns counter 68

Verbs 9,67,71
Vocabulary - conditional words 4,9,21,22,25,64
Vocabulary - general 34-37,39,40,48-50,58-61,64,71

Wait 10,48,68
Wearing things 25
What now ? 49,60

Yes and no 22,68
You can't 49,60