THE GAC ADVENTURE WRITERS HANDBOOK

CONTENTS

Grateful thanks to all contributors :

Simon Lipscomb, Tim Walsha, Pat Winstanley, John Gwinnell & Lesley Anderson

HOW TO WRITE A GOOD ADVENTURE

The best adventures all contain the following features:

1. A well defined and interesting plot.
2. A good atmosphere.
3. Originality and inventiveness in the problems - but with a good deal of logic.
4. A lot of responses to player input.
5. A degree of humour.

These features are described more fully below.

1. PLOT

Try to make your plot original, not just another "collect the treasures" or "rescue the princess" clone. You must be careful that your plot will work in adventure format - a hijacked plane scenario would probably be a bit limited.
A plot should neither be too simple nor too complex as the player will either become too bored or too confused.
The plot can be presented on the inlay or accompanying literature on a loading screen or at the game start, but these methods allow for gradually less plot to be shown. Plot on the inlay can be read in the shop and may incite the customer to buy the game.
Also any nasty pirates won't know what's going on!

2. ATMOSPHERE

Room descriptions do not have to be big to be atmospheric. Good use of adjectives and adverbs usually does the trick.
For example:

GOOD: You're in a cold damp east-west corridor. The walls are bathed in flickering torch light.
BAD : You're in a corridor. Torches are on the walls. East or West.
Aim to include available exits, but try to make them more interesting than "Exits lead North, East and West". "The path winds it's way Northwards into the mountains", is more interesting.

3. PROBLEMS

Problems consist either of having to get past some obstacle or interacting with other characters or objects. A problem should always have a reason for solving it, usually to reach some new locations or to get an essential object. Problems are usually successive, for example, solving one problem, gets an object needed to solve the next.
When designing a problem keep it relevant to the plot, for example, coping with technology/getting past robots in a science fiction game.
The problem should not be so obscure as to make it impossible, unless some clue is given elsewhere.

The solution to a problem should be relevant to the actual problem. For example, if the player meets a vampire they will assume that the way to defeat it is one of the old vampire methods, (e.g. garlic, cross, holy water and so on). If it turns out that the way past the vampire is to "REMOVE SHOES" then the player will feel cheated.

Gets Harder

Adventures should get harder as they progress, so that the beginners get some easy problems to start them off and more experienced players will not sail straight through to the end.
Easy problems at the beginning should have plenty of clues as to what the player should do and plenty of responses, (see below).
They will also give the player a feel of your adventure writing technique so they will know what to expect later on.
DON'T put any "Instant death" high priorities in right at the start, unless they are for walking off a cliff or something similar. In fact the player should never be killed unless he/she deserves it - moving into a new location and finding out it is quicksand will annoy the player unless it is specifically stated that there is quicksand in that direction.
A player attacking a lion with bare hands is asking for trouble.
A player walking along a path is not.

4. RESPONSES

A game should have lots relevant responses to wrong inputs, instead of saying "You can't", it will say why you can't.
An example of this would be:

The player is on the south bank of a river in a copse of trees.
The way across the river is to build a raft using the axe and the rope.
Error messages can be written for:

a) Trying to cross the river without the raft -
"Full of crocodiles", "Can't swim", etc.
b) Building the raft without the axe -
"You try to fell the tree using karate, but fail".
c) Building a raft having cut down the trees, but not having the twine -
"The logs don't seem to want to stay together".
d) Finally two messages for building the raft successfully, and crossing the river.

5. HUMOUR

In general, humour found in adventure games is the wry comment or sarcasm. Humourous responses are usually those for when the player does something wrong or stupid. Total all out silly adventures and spoofs, (e.g. "Bored of The Rings"), should only be attempted if you are above averagely witty or the player will fed up with the same old jokes every time.
No game should take itself too seriously, it is after all only a game!

GRAPHICS

Graphics are becoming an important part of adventures these days, at least as far as the commercial aspect is concerned. A text-only adventure written on GAC is only utilising half the features, but it is best to leave the graphics until last and use up the remaining memory rather than trying to cram an adventure into memory left after drawing the graphics.

If you try to have a graphic for every location, you will end up either with a small adventure or poor graphics. The best tactic is to use various merged elements, and to place graphics evenly throughout the adventure, so that the player goes for roughly the same amount of time between illustrated locations. It is best to draw graphics of the more interesting places in your game. A simple east west tunnel is not the most inspiring of places, but the 50 foot statue of the 12 armed warrior deity could make an impressive picture.

If you are not particularly good at drawing, you can always get a friend to do the graphics. Failing that make the pictures informative rather than artistic.
Some pointers to follow as regards to graphics are to keep the paper colour dark and pen colour light. Choose similar colours as opposed to wildly different ones, in the hope of getting lots of shades.
As a final point, remember that most adventure reviews do not have graphics ratings.

BELIEVABILITY

This may seem like a strange thing to ask of a game where magic works, or aliens frequently pop in for tea, but your game should have an internal logic.
Believability is enhanced by relevant placement of locations and objects, for example, a snowfield next to a desert, or an ironing board in a forest, do not help believability.
The best way to make an adventure seem realistic is to ask the question why? of all features. If you have no answer, change that feature.

PLAYTESTERS

Always get someone else, (at least one other), to play your game. This way you will be able to find out if your problems are too easy or too difficult, which responses to put in and any mistakes in spelling or syntax, (other people tend to spot your mistakes more easily!). And when your playtester gets stuck or stops the game, don't just tell them the way round it, but note things down and adapt your adventure to accept his responses and/or put in more clues etc.

WHAT IS A CONDITION?

In an adventure the player has freedom to type anything at the keyboard. Only a tiny minority of the possible inputs has relevance to the adventure, so some method must be used to tell the interpreter what these are and how to deal with them. The method used in GAC is the use of four different types of conditions.

a) Connections
b) Local Conditions
c) Low Priority Conditions
d) High Priority Conditions

Every command that the player can type which causes some action to be taken by the interpreter must have a condition of one type or other, as the conditions are what cause action to be taken. It isn't always necessary to have a separate condition for every single relevant input. Very often a group of similar commands can be lumped together into one 'blanket' condition which will deal with them all (e.g. GET, DROP, EXAMINE etc). But remember..... no condition....no action!

The term condition is a little misleading as this implies only a test for truth. In GAC the term condition is used to cover actions too..... sometimes with no 'condition' attached.

The general construction of conditions is as follows:

IF (whatever is inside the brackets is <u>all</u> true) (Then, and <u>only then</u>) perform <u>all</u> the actions between the end of the brackets and the 'END' command.

The conditional part " IF (.....) " can be omitted altogether if it is required that a particular action be performed every turn. Normally, this type of entry would go in the High Priority section.

When the conditional part is used, what it does is test to see whether the player's input and any other relevant information is true. If everything in the brackets is true, all the actions after the brackets will be performed. If anything within the brackets is not true, none of the actions after the brackets will be performed, so the condition will be ignored. The search through further conditions will continue until either a condition which is true is found and acted upon, or no true condition at all is found in which case no action will be taken, and the player will be asked to try again.

Order of Priorities

Each condition is looked at in turn and acted upon if appropriate until either a command to stop looking at any further Local or Low Priority conditions is found (WAIT or OKAY at the end of a true condition which jumps the player to the beginning of the High Priority conditions), or all conditions have been checked and the player is asked for the next command. The order in which conditions are checked is predetermined and very important..... Connections, Local, Low, High.

CONNECTIONS .. are conditions which the writer can adjust only to the extent of which movement verb leads to which location. Any input by the player which matches a valid connection in the room where the player is will be acted upon, the player moved to a new room, The High Priority conditions checked and acted upon if appropriate, and the "What now?" message printed....WITHOUT LOOKING AT ANY LOCAL OR LOW PRIORITY CONDITIONS!!! So if you want a move from one room to another to be conditional upon (say) possession of the correct key, that particular room connection must not be entered in the room description, otherwise the player will move before any check is made as to whether or not the key is available. Instead, the connection must be ommitted from the room description connections and entered elsewhere, usually as a LOCAL condition. For instance, say you wanted the player to be able to move north from room 6 to room 7 if, and only if a key was being carried, the entry would be as follows:.

IF (VERB (north) AND AVAI (key)) GOTO 7 WAIT END

This would be entered as a Local condition for room 6, but no en[illegible] would be made for 'north' in the room connections.

LOCAL CONDITIONS ..... If no valid connections are found or the player has not typed a movement verb, the next set of conditions will be searched (in number order) for a match with the player's input and status. These will be the Local conditions relating to the room where the player currently is. (If the player is in room 15, only room 15 conditions will be checked, regardless of whether another room's local conditions match). If and when a true condition is found it is acted upon, and if a 'WAIT' or 'OKAY' is encountered in that condition, a jump will be made to the High Priority conditions, then to the player's next command as above. If no true condition is found, or one is found and acted upon which has no 'WAIT' or 'OKAY' command in its actions, further searching continues. Only when all relevant Local conditions have been looked at are the Low Priority conditions checked.

LOW PRIORITY .... Assuming no jump has been made, the Low Priority conditions are checked for a match in exactly the same way as the Local ones. Whereas only the Local conditions pertaining to the current room are available for checking, all the Low Priority conditions are available. Again, true conditions are looked for and acted upon, and if no jump is made, High Priority conditions are checked.

HIGH PRIORITY CONDITIONS ... These are always checked and acted u[illegible] after every turn, regardless of whether any Connections, Local or Low conditions have been performed. A 'WAIT' or 'OKAY' command which stops further checking of Low or Local conditions jumps to the beginning of the High Priority table after incrementing the turns counter. Once The High Priority conditions have been 'done' the "What now?" message is printed (if the player is still in the game!).

Use of Conditions

CONNECTIONS
Used ONLY for unconditional moves between rooms.

LOCAL CONDITIONS
Used for actions which will only be performed in one location

LOW PRIORITY
Used for actions which can take place anywhere

HIGH PRIORITY
Used for conditions and actions to occur before the players input.

Drafting Conditions

When working out the form of a new condition, it is useful to ignore noun, verb, object, marker numbers, and write the logic out in plain English. Think of the condition as split into two parts;

1) TRUE OR FALSE....... what needs to be true for the actions to be performed?

2) WHAT TO DO.......... what actions are required for the desired effect?

e.g. IF (VERB {hit} AND NOUN {granny} AND HERE {granny}) ....
[then] MESS 'she hits you back harder!' WAIT END

Only when the logic of the condition is clear on paper should you insert the relevant numbers. Don't stop in the middle of a condition to enter a new message or verb etc, as doing so will probably ruin your train of thought. Also, keep your draft on paper until the condition has been entered and debugged. It's a lot easier to read and understand English than algebra!

MARKERS

While writing an adventure you will come across many situations where, depending upon the player's actions, the environment can be in one of two states. For instance..... imagine the following text greeting the player on entering a location;

"You are in a farmhouse kitchen. You can also see a table."

Now suppose you wanted the player to examine the table, and in doing so find a bottle. This is simple to set up, merely by having a condition on the lines of;

A) IF (VERB (examine) AND NOUN (table)) BRIN (bottle) LOOK WAIT END

On typing "exam table" the player would be rewarded with the following text;

"You are in a farmhouse kitchen. You can also see a table, a bottl

There is just one snag with this routine. If the player again types "exam table" (which is perfectly reasonable since occasionally adventures provide for a sequence of objects to be found) the bottle will be acted upon by the BRIN command again. Since you won't want the player to discover the bottle every time he examines the table, something must be done to let GAC know that the bottle has already been found.

This is where the use of a marker comes in! A marker can be imagined as a switch which can be either on or off, but no other state is possible. In the situation described above you would allocate a marker to 'mark' whether or not the bottle had been discovered. If the player types "exam table" and the marker says that the bottle has not yet been found, GAC needs to know that it can bring the bottle. If however the bottle has already been found, GAC needs to know that further examination of the table will NOT result in the bottle being brought. And since examining the table after the bottle has been found ought to give some description of the table, two conditions need to be set up, (B) & (C);

B) IF (VERB (examine) AND NOUN (table) <u>AND RES? x</u>) BRIN (bottle) <u>SET x</u> LOOK WAIT END

C) IF (VERB (examine) AND NOUN (table) <u>AND SET? x</u>) MESS (see nothing special) WAIT END

So if the bottle has not been found, condition (B) will be acted up[illegible].. because marker x is "reset". (All markers are reset initially unless you set them, except markers 0 & 1).

If condition (B) is acted upon, one of the things that will happen is the "setting" of marker x in the action part of the condition. So if, at a later stage, the player tries to examine the table again, condition (B) will no longer be true. GAC, on reaching "res? x" will check marker x and find that it is "set". This being so, it will move on to look for a true condition.... condition (C). Here GAC is told to check for marker x being "set ?", which it is. So condition (C) will be acted upon giving the message "You see nothing special".

Once a marker is set, or reset it will remain in that state unless GAC acts upon a condition which contains a command to switch it to the opposite state.

There are only four possible expressions which can be used regarding markers:

<u>SET? x</u> - is used in the 'IF' part of a condition, ie within the brackets to ask GAC to check if the marker x is "set". If marker x is not "set" the condition is not true, as so will not be acted upon.

<u>RES? x</u> - is also used in the 'IF' part, to ask GAC to check if marker x is "reset". If marker x is not "reset", the condition will not be acted upon.

<u>SET x</u> - is used in the 'action' part of the condition ie after the brackets. If marker x is reset, "set x" will change it to being "set". If marker x is already "set" it will remain "set.

<u>RESE x</u> - is used in the 'action part. If marker x is "set", "rese x" will change it to being "reset". If marker x is already "reset" is will remain "reset".

So if you want a condition to include a check on the state of a marker, use RES? or SET? inside the brackets.

If you want to change the state of a marker, use SET or RESE after the brackets.

LIGHT AND DARK

To indicate a light or dark room, a special marker is set aside... marker 1.

If the player is in a light room and moves into one you have designated as being dark, marker 1 is changed from being set to being reset. In this special case only the action words SET and RESE are needed. The way to acheive a change of light conditions is as follows.

For this example, imagine two rooms;

```
 ------------           ------------
:            :         :            :
:   ROOM 1   :         :   ROOM 2   :
:            :---------:            :
:  (LIGHT)   :         :   (DARK)   :
:            :         :            :
 ------------           ------------
```

a) Make sure there is NO connection E 2 from Room 1, and NO connection W 1 from Room 2, in the movement line of the room description. If you don't, typing the relevant direction will cause a move to be made without looking at any conditions.

b) Make local conditions as follows;

```
ROOM 1.... IF ( VERB (east) ) RESE 1 GOTO 2 WAIT END
ROOM 2.... IF ( VERB (west) ) SET 1 GOTO 1 WAIT END
```

What happens is this: If you are in room 1 and type "east", GAC looks through the connections, doesn't find an east connection, so moves on to the Local Conditions for room 1. Here it finds a condition for moving east and acts upon it, in the process 'resetting' marker 1. Now GAC knows (because it is already built in) that if marker 1 is reset, the current room is dark. The current room in this case is room 2 since the GOTO moves the player there. Since marker is signifying darkness, the room description is not printed. Instead the appropriate message, "It is dark. You can't see." greets the player.

If the player is in room 2 and types "west", GAC moves the player to room 1, "setting" marker 1 in the process. Now that marker 1 is set, GAC knows the player is in a light room and prints the description of room 1.

If you have dark rooms in your adventure, you will probably want to provide the player with a lamp. Again, a special marker is set aside by GAC to cope with lamps... marker 2.

If the player is in a room and both marker 1 and marker 2 are "reset", GAC will refuse to describe the room, instead giving the "It is dark" message. So marker 2 is used to tell GAC whether a lit lamp is available.

There are three possibilities for the circumstances in which the player may move east from room 1 to room 2:

a) Carrying a lit lamp with him.
b) The lit lamp is already in room 2 having been left there earlier.
c) The lit lamp is neither carried nor in room 2 already.

Now if either (a) or (b) is the case, GAC needs to know that light is available in the dark room when the player reaches it, ie marker 2 must be "set" when the player gets there. This is achieved with the following line:

Room 1: Line 1:

```
IF ( 2 IN (lit lamp) OR CARR (lit lamp) AND VERB (east) ) RESE 1 SET 2
GOTO 2 WAIT END
```

This line tells GAC that if the lamp is in room 2 or the player is carrying it and east is typed, rese marker 1 to say the room is dark, and set marker 2 to say there is a light source available.... then move to room 2 .

If neither of the above are true, ie the lit lamp is not in room 2 and the player is not carrying it, when east is typed in room 1, the condition will be ignored. But we still want the player to be able to get to room 2, even though he will get a "It's dark" message, so the following line must be inserted after the first:

Room 1: Line 2:

```
IF ( VERB (east) ) RESE 1 RESE 2 GOTO 11 WAIT END
```

This line will take care of possibility (c)

When setting up the "light lamp" command, ignore the setting of markers and let the local movement conditions take care of it. Simply by having two objects , a lit lamp and an unlit lamp which are swapped as required will be enough for descriptive purposes.

Don't forget that similar conditions must be set up for each move from dark to light or vice versa, but moving light to light or dark to dark can be set up using normal connections.

COUNTERS

While markers are useful to store information about the status of the game, counters are far more versatile. The ability to store any number between 0 and 255 opens immense possibilities, not only for counting turns and scoring!

Almost anything can be stored in a counter as long as it is in the form of a number. This includes terms such as TURN, ROOM, RAND x, VBNO, NO1 and NO2, which all return a number when used in a condition.

Counters can also be used to manipulate numbers, by using them as temporary stores, and their contents as the values used in arithmetic expressions.

Scoring

This is one of the simplest uses of a counter. Counter 0 is used t hold the score, since this is where GAC looks when asked to print th score at the end of a game, or when the player is killed.

Imagine you had the following puzzles and scores for solving them in your game;

1) Find a key.........20 points
2) Kill a giant.......30 points
3) Cross a river......50 points

To increase the score when the relevant puzzle is solved, the condition which covers solving the puzzle needs to add the correct number of points to counter 0. All counters are set to 0 initially, and only the 'turns' counters (126 & 127) change without any intervention.

So, Counter 0 holds the value 0 at the start. There are three ways to change the value held in a counter:

1) x CSET n - stores the value 'x' in the counter numbered 'n'. Thus to store 25 in counter 0 you would use " 25 CSET 0 " in the action part of a condition.

2) INCR n - adds one to the number held in counter n. If counter 0 held the number 25 from the last example, using "INCR 0" in the action part of a condition would result in counter 0 then storing the number 26. Using "INCR 0" again would result in counter 0 storing and so on.

3) DECR n - subtracts one from the number held in counter n.... exactly the reverse of "INCR n".

Returning to our 'scoring' using counter 0, this is how to do it;

A) IF ((the player does something to find a key)) CTR 0 + 20 CSET 0 (and anything else that happens when the key is found)

The 'scoring' part is "(CTR 0 + 20) CSET 0 "

Taking the part within the brackets first, "CTR 0" gets the value currently held in Counter 0, and "+ 20" adds 20 to that value. The result of the 'sum' in the brackets then REPLACES the original value held in counter 0 by the action "CSET 0". This might seem a bit long winded; why not simply use "20 CSET 0"? The problem with doing that is that since the result of the calculation in the brackets replaces whatever was originally held in counter 0, "20 CSET 0" would only give the correct result if counter 0 happened to be holding the number "0". A look at the condition for killing the giant will demonstrate why the simpler version is inappropriate!

B) IF ((player kills giant)) CTR 0 + 30 CSET 0 (and any other actions required)

Again, "CTR 0 + 30" is taking the value of counter 0 and adding to it, but assuming the key has already been found, counter 0 will contain not 0 but 20 from condition (A). If we simply used "30 CSET 0", counter 0 would end up holding only 30, not 50 (20 + 30)!

C) IF ((player crosses river)) CTR 0 + 50 CSET 0 (and other actions reqired)

Assuming the key has been found and the giant killed, counter 0 will be holding 50, so "CTR 0 + 50" will be the equivalent of "50 + 50" giving a total of 100 to store in counter 0. If we assume that crossing the river completes the adventure, the score (ie the value of counter 0) will be displayed (if marker 3 which can disable this is reset). The puzzles could have been solved in any order, but the final result will still be 100.

Some games penalise the player for 'saving' a game position. If you want to do this, simply alter the routine for saving to include "CTR 0 - x CSET 0" where 'x' is the amount of the penalty.

Turns

The number of turns taken by the player are automatically stored in counters 126 & 127. This can be useful if you want to introduce a routine where the player will die of hunger if he doesn't find and eat food within a certain number of turns. Assume the player is allowed 20 turns from the start to find and eat the food. The following condition will kill him off after the 20th turn if he has not eaten. Here we will use a marker to 'mark' that food has been eaten. Any marker (except the reserved ones) will do...we'll use marker 5.

LOW.. IF ((player eats food)) SET 5 (and any other action required)

HIGH.. IF (TURN > 20 and RES? 5) (kill player)

In this routine, marker 5 will be reset until the player eats the food, when it will cahnge to being set. If marker 5 is set, the High Priority condition will be ignored. If marker 5 is reset and the player has not yet taken 21 turns, the High Priority condition will be ignored. But if the player has not eaten (res? 5), and more than 20 turns have been taken, he will be killed. 'TURNS' is used here as it is a short way of saying 'counter 126'.

If later on in the game you wish the player to have a limited number of moves in a desert without drinking, 'TURNS' is not much use, since you don't know what turn the player will reach the desert in, so have no idea what turn to kill him off in. To overcome this, we can use a marker to check when the player enters the desert, and use a spare counter as a base for our calculation as follows. (We'll assume 10 turns are allowed without water, marker 6 to say the player has entered the desert, marker 5 for drinking, and counter 5 to keep track of turns after entering the desert).

```
LOCAL...( Instead of a connection to the desert location)
IF ( go into desert ) 10 CSET 5 SET 6 enter desert location)

LOW... IF ( (drink water) ) SET 5 ( any other actions)

HIGH... IF ( RES? 5 AND SET? 6) DECR 5 END

HIGH... IF ( CTR 5 = 0 AND SET? 6 ) (kill player off)
```

In this sequence, as the player enters the desert, counter 5 is loaded with the value 10 (turns allowed without water) using "10 CSET 5", and the fact that the player has entered the desert is noted by setting marker 6. The first 'high' condition then subtracts one from counter 5 every turn unless marker 5 is set (ie water has been drunk). If marker 5 is reset, counter 5 will continue to decrease, until eventually (after 10 turns) it reaches 0. When this happens, the second 'high' condition kills the player off.

Money

Sometimes you will want your player to be able to obtain and spend money, tokens etc. To keep track of the player's solvency (or otherwise!) use a counter to store the value of his holding. Things can get tricky if you have more than one value of currency, eg 100p = £1 so initially stick to a single unit. You will need to be able to reduce the value of the counter when the player spends money (or perhaps has it stolen) and increase the value of the counter if he finds (or steals) extra cash. If possible, use a counter with the same number as the object it counts....it makes keeping track and de-bugging easier!

Assuming counter 100 will hold the money:

To get money;
IF ((player gains money)) CTR 100 + x CSET 100 (and anything else)....

To spend/lose money;
IF ((player spends money)) CTR 100 - x CSET 100 (and anything else)...

To count money;
IF ((player counts money)) PRIN CTR 100 (with messages to make it read well)..

Time

This can be controlled by allocating a set number of turns to represent a unit of elapsed time. If we assume 60 turns equals one 'hour', the following routine will give the player a time-check on request. (We'll use counter 10 to store 'minutes' and counter 11 'hours).

```
HIGH.(A) INCR 10 END

HIGH (B) IF ( CTR 10 = 60 ) INCR 11 0 CSET 10 END

LOW .... IF (( player asks time)) PRIN CTR 11 PRIN CTR 10 (with
suitable messages to make it read well)
```

Using this, minutes will never be greater than 60, as when counter 10 reaches 60, 1 is added to the hours and counter 10 loaded with 0 to start counting again. You could improve this routine by adding an extra counter (say counter 12) to count days. Just add this:

```
HIGH (C) IF ( CTR 11 = 24 ) INCR 12 0 CSET 11 END
```

Also add "PRIN CTR 12" to the low condition to have days displayed too.

These are just a few examples of the ways counters can be used. In conjunction with the other commands GAC provides, you can build up virtually any 'counting' routine you wish..........EXPERIMENT!!

DIAGNOSTICS SCREEN

This shows the state of markers and counters during the game. They can be called up at any point in the game, by pressing ESC followed by D, and are useful for finding out why something is not going quite as expected.
The first table shows markers -

MKR	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
0																				
20																				
40			1																	
60																				
80																				
100																				
120																				
140																				
160													1							
180																				
200																				
220																				
240																				

e.g. Marker 42 has been set and so has 172

Note: Markers start off Reset, except for No.0 and 1, which are used by the system.

The second table shows the state of the counters at the time of access. Note that a counter can only hold a number from 0 to 255. The GAC counter system works in Modulo 255, therefore if a number is raised more than 255, it will wrap around back to 0 and up, whilst the opposite happens if a counter is lowered below 0.
N.B. This only applies when a counter is changed by:

CTR [counter No.] + [No.] CSET [counter No.]

Not when it is changed by the INCR and DECR commands. With these when the number reaches a limit (255 or 0), it will stay as it was.

CTR	+0	+1	+2	etc.............
10	0	0	0	
20	0	0	254	
30	0	0	0	
40	0	55	0	
50	0	0	0	
etc.				

e.g. In the above diagram, counter 22 has a value of 254 whilst counter 41 has a value of 55.

TEXT

STORAGE OF TEXT AND COMPRESSION

As you write your adventure all the words you type, whether as messages or nouns for example, are stored in a dictionary. Each new word takes up a number of bytes equal to its length, plus another three for a pointer showing its position in memory. When, (and if), the word is deleted, part of the pointer remains so that if you constantly enter and delete words, you would gradually run out of memory. The old words remain in memory, the program merely ignores them.
When a word is entered that has already been used, the GAC will refer to its pointer, thus only using up three bytes.
Points to remember are:

a) Spelling mistakes and typing errors are all stored in memory, so check a line before entering it.
b) If you are running low on memory, try using words that you have used before.
c) Words, messages etc are stored in memory in the order you enter them, which is why when you do a printer listing, (except for the Commodore), local conditions and messages may not be in numerical order.

NOUNS AND VERBS

When entering nouns, verbs and adverbs, try to use as many synonyms as possible, but be careful of overuse, for example, "SCRUTINISE" for "EXAMINE".
With an object such as "A CREDIT CARD", the associated nouns should be "CREDIT" and "CARD".
It is a good idea to feature "THEM" as noun 255 as well as "IT", if you have objects such as "a pair of shoes". "HIM" and "HER" can also be included if other people feature strongly in your adventure, (although the program will not distinguish between them!).

The GAC handles abbreviations automatically, looking through verbs, nouns and adverb tables until it meets a word that matches up with the players input. To explain this more fully we have included a sample verb table:

20	EAT
3	EAST
16	EXAMINE
7	GET
13	GRAPHICS
10	INVENTORY
9	L
15	LOAD
9	LOOK

In the adventure with this verb table, the player could type "INVENTORY", "INVEN", "INV" or even just "I", because it is the only verb beginning with "I". To go EAST, however, the player must type "EAST" or "EAS" - simply typing "E" or "EA" would cause the program to try to "EAT", due to the fact EAT is before EAST alphabetically. To make the program accept "E" for east, you would need to enter "3 E". A similar thing has been done with "LOOK" and "LOAD".

DIFFERENT BUT THE SAME

If you have different types of the same object, for example, "A RED CARD", "A GREEN CARD", etc., there are several ways of telling the difference.

a) Colour as Noun

```
i.e.    Obj 1  =  "a red card"
       Noun 1  =  "RED"
        Obj 2  =  "a green card"
       Noun 2  =  "GREEN"
```

If you are doing it this way, it is a good idea to have a noun "CARD", (separate from the actual cards), and have a line that checks to see if the player has just typed "CARD". If this is so, a message along the lines of "which one?" is printed up.

b) Use of Adverbs

Probably the best method, but it does mean the player has more typing to do, is:

```
     Obj 1  =  "a red card"
     Obj 2  =  "a green card"
  Adverb 1  =  "RED"
  Adverb 2  =  "GREEN"
  Noun 100  =  "CARD"
```

So a line would read as follows:

```
IF ( VERB [TAKE] AND NOUN [CARD] AND ADVE [RED] AND HERE [RED
CARD] ) GET [RED CARD] OKAY END.
```

c) Priority of Objects

The player need only type "GET CARD" and the program will first check whether the Red Card is there and get it, if it is. If not the program will check for the Green Card and so on.
This method can be used for cases where there are two objects that represent one object in two different states and that cannot exist at the same time, for example a lamp and a lit lamp.

SYSTEM MESSAGES

These are the messages from 239, (238 for BBC computers), onwards. They are automatically printed up by GAC if the situation arises. The messages and situations are as follows:

238	"nothing." (BBC ONLY)	This is printed when a LIST command and the area listed, (either WITH or a room), is empty of objects.
239	"You are carrying'. "	Should always have a space at the end, or a colon. The only one GAC does not automatically use.
240	"What now ?...."	Printed up after all High Priorities are carried out. WAIT and OKAY return to this section of the program.
241	"You Can't."	When the players input contains at least one word in the GAC's dictionary, but no conditions are met by it.
242	"Pardon ?"	None of the input is recognised.
243	"Press a key...."	Printed after EXIT or if the player answers Yes to Mess 244.
244	"Are you sure ?..."	Printed after a QUIT command has been executed.
245	"You've already got that"	If a GET command is carried out and the player already has the object.
246	"You haven't got that"	If a DROP command is carried out and the object is not carried.
247	"You can't see that"	If a GET command is carried out and the object is not HERE.
248	"Carrying too much."	If the objects weight + weight of currently carried objects exceeds STRE.
249	"Your score was"	Leave a space at the end. If the player answers Yes to 244 or if an Exit is met.
250	" and you took"	Leave a space at each end. Printed after Mess 249 and CTR 0. NOTE: It is a good idea to print the maximum score possible.
251	"It is dark. You can't see"	Automatically printed up each move, if marker 0 is reset.
252	"I can't find that anywhere"	If a BRIN is carried out and the object in question a) doesn't exist or b) is in room 0.
253	"You can also see"	Needs a space, colon etc. Printed after room descriptions if objects are present.
254	"Okay"	OKAY prints this and performs a WAIT. It can be printed without a WAIT by MESS 254.
255	" turns"	Needs a space at the front. Printed after Mess 250 and TURNS.

NOTE: Messages 249, 250 and 255 can be used for scoring during the game, but their wording must be such that they mean "So far in this adventure" and "In the adventure you have just played".

GRAPHICS

CARTOON FILMS ON GAC! (Amstrad, BBC and Commodore only)

Yes, simple animated cartoons are possible on GAC, although their uses are limited.
To draw your cartoon, simply draw one frame of animation, redraw over the parts to be moved in Ink 0, then draw on a new frame.
It is best to stick to lines, as filling can slow the process down and make the "film" seem jerky. If the line drawing goes too fast however, a few fills in the background colour, (0,0), will slow the action down to the speed required.

ADVANCED GRAPHICS - TECHNICAL

Memory Used (Bytes)

Function	BBC	Amstrad	Spectrum	Commodore
/	5	5	5	5
D	3	3	3	3
R	5	5	5	5
E	5	5	5	5
I	1	1	1	1
S	3	3	3	3
F	3	3	3	3
P (m)	3	3	3	3
AOT	*	*	*	1
MIRROR	*	*	*	1

HOW TO SAVE MEMORY IN THE GRAPHICS SECTION

a) Draw a rectangle instead of 4 lines.
b) To draw an arc, first draw an ellipse, then redraw the part you want with lines. Step back through the picture and delete the ellipse.
c) Fill an area first and then add details instead of filling around them. If adding detail to a stippled fill, try to draw it in a colour that is not one of the two comprising the shade, this makes it much bolder.
d) Use the step back and forwards facilities to delete mistakes made, instead of merely drawing over them.

LOOK - If you have a problem whereby EXAMINING or OPENING something BRINGS an object, it is user friendly to put a LOOK command after the "A-HA, FOUND SOMETHING" message, so that the player can see what thay have found. Of course if you have a message that reads "YOU FIND A FROG", for example, this is not necessary.

DESC - A neat use of DESC is to have illustrated Examine messages. The way to do this is to have a room instead of a message, with the Examine reply in it, i.e. "It is a large, rusty key". Thus if you have a room, you can also have a picture of the object. The conditional line for Examine would read something like:

IF (NO1 is within examinable object range AND VERB EXAMINE) DESC NO1 WAIT END.

If you examine object 1, (with Noun 1 presumably), room 1 will be displayed. If room 1 is part of your adventure, start all Examine rooms at 200 for example and DESC NO1 + 199.

x SWAP n If two objects are to be swapped whilst being carried, they must be of the same weight, or problems with the inventory limit will occur. If they are not the same weight, simply drop them first.

OBJ n - This can be used to 'spice up' messages, for example: "A rat steals" OBJ n.
Note that as most objects are "A gun" or "SOME seed", clever wording must be used to avoid "you can't see the a gun".

LIST - List can be used to do backpacks, chests, other people's inventories and so on. A room must be assigned to the backpack, for example, as well as the actual object. Objects placed in the backpack are sent to this room. When the player EXAMINES or LOOKS into the backpack, print a message "very roomy. It contains: " and LIST [Backpack room].
It makes the game look neater, if after LIST, another room or "WITH", you have a message "." printed at the end.

x to n - If moving an object that the player is carrying, make sure it is DROPPED first or problems with the inventory will occur as with SWAP.

CTRx + CSETx - Ctr 0 holds your score. Ctrs 126 and 127 hold the turns and should be left well alone.

CONN - This can be used to do a "You can't go that way" routine. IF CONN x = 0, it means you cannot go VERB x, so a line :

IF (CONN [x] = 0 AND VERB x) MESS "BLOCKED" WAIT END.

is required. This is more user friendly than merely repeating "You can't".

STRE - This can be altered in the course of the adventure by potions of healing and so on.

BRIN - Brin x uses 1 less byte than x TO n, so if you have a situation where a new object is brought to the players location, for example, a vending machine, it may be better to use BRIN x if this situation occurs frequently. NOTE: objects cannot be brought from room 0, so use some other room such as 500 as an "object sanctuary", where all uncreated and destroyed objects reside.

MESS	-	Several can be strung together, i.e. built up from a stock of short multiple use message.

DEBUGGING NOTE: If debugging your adventure, make up a verb and put it in the verb table, i.e. 254 x. Then put the following line in low priorities:

```
        IF ( VERB 254 ) BRIN NO1 GET NO1 OKAY END.
```

This will not work if your uncreated objects start in room 0. To use the line, begin at the room you wish to test and type "x object".

FIND	-	This can be used for 'teleports', where the player is returned to an object either static, (although GOTO would work for that), or movable.
WAIT	-	WAIT returns to players input, so if a line containing WAIT is true, any following lines will be ignored.
OKAY	-	Basically MESS 254 WAIT.
PRIN	-	Useful for printing CTR values for coins, stamina etc. Can also be used in debugging by using PRIN ROOM.
HOLD	-	By having a value such as HOLD 65000, a press a key situation can be achieved, as HOLD is stopped when a key is pressed.
GET/DROP		These check automatically for the conditions needed for GET, for example, is it here?/are you carrying it already?. They are also the only commands that alter the value of your inventory limit, so if you move an object directly to 0 without dropping it first, its weight will still be in your inventory.

Condition lines do not exist with 'gaps' between line numbers. For example, if you have lines 1, 2, 3, and type in a line 10, it will be moved down to become line 4.
New lines can be inserted between existing lines as long as END is put at the end. For example:

```
Line #10 is......
              IF ( VERB 10 ) GOTO 1 WAIT END
```

If we wanted to enter a line between 10 and 11, we could either place it at the end of line 10:

```
   IF ( VERB 10 ) MESS 1 WAIT END if verb 11 mess 15 wait end.
```

Or before line 11:

```
   If verb 11 mess is wait end IF ( VERB 50 ) GOTO 2 WAIT END.
```

In either case our new line would become 11, 11 and all following lines would be shunted up 1.

The GAC will check through all lines of conditions until it finds one that is true. It will then carry out all actions in the line. If it encounters WAIT, it will return to the input routine and ignore any following conditions.
OKAY is the same as WAIT, except that it prints up message 254 first, so OKAY WAIT END is superfluous.

Use of WAIT can mean that you do not need to check a marker twice.
For Example: Local Condition Room 1

```
Line 1.....
           IF ( RES? 10 [TROLL ALIVE] AND VERB KILL AND NOUN
          TROLL ) TROLL TO 0 MESS "YOU KILL TROLL" SET 10 WAIT END
Line 2.....
           IF ( VERB KILL AND NOUN TROLL ) MESS "ALREADY DEAD"
           WAIT END
```

In line 2 marker 10 is not checked because it has to be set if "KILL TROLL" has been typed and this line is being checked by GAC. If 10 was reset and "Kill Troll" was typed, line 1 would be carried out, the wait in it would return the GAC to input mode and line 2 would never be reached. This works best with local conditions.
NOTE that line 2 has a wait on it as well. This is because local conditions are checked before low and there is a line in low conditions that is:

```
 IF ( VERB KILL ) MESS "VIOLENCE WILL NOT HELP YOU HERE" WAIT END
```

This will be printed up if the player tries to kill something other than the troll.
If line 2 did not have the wait, it would print "already dead. Violence will not help you here". This actually makes sense! But imagine a hermit who you give an apple, and an error trap for give with no one there, it would come up with; "Thank'ee kindly. No one wants that".

If a line takes up 255 characters without brackets, when it is entered the GAC will include brackets and spacing which will cause the line to be more than 255 characters long, which may cause undesirable effects.

ADVANCED USE OF CONDITIONS

NOUN, VERB, ADVE

It is worth noting that the GAC will understand four words from an input sentence - 2 nouns, a verb and an adverb. These of course, do not have to be used as nouns, verbs and adverbs. Nouns can be stored under verbs and so forth, for example, Noun 1 could be "Take", but it is easier to keep to the correct catagories.

HERE

If you have monsters/obstacles as objects, HERE can be used instead of a marker. For example, instead of marker 10 being the troll - set means dead, 2 objects can be used. "A large ugly troll" and "a dead troll". Then conditions could be placed in the program to control whether the troll is alive or dead.
A line would read:

```
    IF ( VERB NORTH AND HERE TROLL ) MESS "TROLL BLOCKS PATH"
    WAIT END
```

followed by:

```
    IF ( VERB NORTH AND HERE DEAD TROLL ) GOTO NORTH ROOM WAIT
    END
```

HOWEVER, CARE MUST BE TAKEN - If there is a connection between the troll room and the NORTH ROOM, GAC will by-pass the above instructions and go directly to the north room, ignoring the fact that the troll is there. (See flowchart)

X IN n   (See Errata Information also)

This can be used in a similar way to HERE. For example, in a shop, all objects for sale can be stored in an inaccessable room, say 200. Instead of having a marker to determine whether an item has been purchased, check whether it is in 200 using 200 IN object. If it is then the player may buy it.
NOTE: IN must go at the start of a conditional expression.

Set? & Res?

Used to check the state of markers. Markers are used for something that has two possible forms, for example, a chest open or closed/whether a button has been pressed etc. Physical objects such as doors, chests, monsters and so on, can be done using HERE, but markers are useful for things such as determining whether you have initiated the self destruct sequence or are under an invisibility spell.
NOTE: Remember the ? on the Set? !!

Some built in markers are:

Mkr 0 -	If set, it means a room has been described since last reset, (either by moving, looking or DESC). This is useful for high priorities relating to the colour of the sky for instance. A RESE 0 command must be placed after all checks for SET? 0 in high priorities.
Mkr 1 -	If set means room is light. If reset means room is dark, (i.e. print mess 251). To do this, have local conditions when moving from Dark to Light/Light to Dark, (i.e. from the outside to the inside of a cave), which set or reset 1 depending on the direction of movement, current status of mkr 1, (so as not to get stack empty), and any light source available.
Mkr 2 -	If set means you are carrying a light source, for instance a torch. Note that Carr Torch has the same effect as Set? 2.
Mkr 6 -	If reset means it is the first move. It can be used to set, stre, print "thanks to GAC" etc. For example: IF (Res? 6) MESS WELCOME TO *** QUEST STRE 100 SET 6 END. Effectively any marker can be used for 2 & 6.

CTR

These are used in a similar way to markers, but for things that have several possible values, for example, money. Further information can be found under INCR, DECR, CSET and x EQU? n.

x EQU? n

Does ctr n EQUAL x? This condition must go at the beginning of a set of conditions.

VBNO, NO1, NO2

VBNO = 1 is the same as VERB 1. These conditions return the value of the verb/noun etc. and can be used to check whether the player has tried to take a movable object or an obstacle, i.e. all takeable objects have nouns below 50.

```
          IF ( NO1 < 50 AND VERB TAKE ) GET NO1 OKAY END.
```

If VBNO, NO1 or NO2 = 0, it means that the player has typed in something that is not in the games vocabularly. This means you can do lines such as:

```
    IF ( VBNO = 0 ) MESS "I DON'T KNOW HOW TO DO THAT" WAIT END.
```

ALGEBRA

Any condition containing <, >, +, -, must go at the FRONT of a set of conditions. This also applies to RAND and IN. For example:

```
               IF ( VERB 7 AND NO1 > 8 ) MESS 1 END
```

will not work properly, whereas

```
IF ( NO1 > 8 AND VERB 7 ) MESS 1 END
```

is correct.
Because EQU?, IN and ALGEBRA must go at the beginning of conditional lines, if you wish to check two of them, for example, IF NO1 < 8 and 5 EQU? 7, you must use two lines and a marker:

```
1 IF ( NO1 < 8 ) SET 40 END

2 IF ( 5 EQU? 7 AND SET? 40 ) MESS 1 END
```

High priority line 1 must be RESE 40, otherwise line 2 will happen without line 1 being true. The way these lines work is as follows:

1 IF NOUN was less than 8, SET 40

2 IF Ctr 7 = 5 and 40 is SET (line 1 must have been true therefore NOUN less than 8) MESS 1 END.

If you remember that Room, VBNO, NO1, NO2, TURN, WEIG and CTR (X) are all numbers disguised as words, you can carry out sums with them, set Ctrs to them, GOTO them and all sorts of things.

MULTIPLE LOADING TECHNIQUES

There are several ways in which to implement multiple loading techniques.

The first is to simply have each part as a stand alone adventure, for example, "EUREKA".

The second is the password method, whereby completing one part gives you the password for the next. A point to note is that if this method is used, a word the same as the password, minus one letter, eg. PASSWORD & PASSWOR must be defined to ensure the whole password is typed.

The final method is the load/save method. This entails saving a status out at the end of one part and loading it in at the beginning of the next.
SAVE command saves out status of markers, counters, objects, relevant nouns and your current room, so <u>ALL</u> these must match for all parts.
It is a good idea to set a marker at the end of one part and check for it in the next part to ensure it is a save from a completed game.

ERROR MESSAGES (Additional)

ROOM NOT FOUND	The program has been sent to a room that has not been defined, (either by beginning there, by use of GOTO or by an exit)
MESSAGE NOT FOUND	The program has been told to print an undefined message. Use the line that is printed up to determine which. If the line contains no undefined message, not found is a system message, (239-255, 238 for BBC - See System Messages).
STACK EMPTY	Usually because a "?" is left out or a space left between the SET or RES conditions.
ILLEGAL VALUE	A value that is either negative or exceeds the maximum value. (See Appendix A - Amstrad, Appendix B(v) - BBC, Commodore, Spectrum for the range of numbers for maximum values possible). It will usually be a number greater than 255 that is causing the trouble.

ERRATA INFORMATION

Advinman messages (Chapter 5 - Amstrad, App.A - Others)

1. It is a small battery operated lamp.

2. It is a freshly dead rat. Looks tasty!

3. It is a small silver key.

4. It looks valuable.

5. It is a small lamp, shining brightly.

6. You can't walk through doors you know!

8. The snake wakes up, comes over and bites you. The poison takes hold immediately.

9. Well done! You got out with the gold. You are rich!

10. The door unlocks and swings open.

13. The snake wakes up, comes over and eats the rat. It then returns to its position.

14. The snake, not surprisingly takes a nasty turn and bites you. The venom kills you.

15. It lights up brightly.

16. It goes out.

17. It tastes even better than it looks! Yum Yum!

18. You find nothing much.

19. I'm afraid I don't know what that is.

20. You hear scuffling footsteps nearby....

21. A giant spider with glowing red eyes leaps from the shadows and neatly severs your head with powerful jaws, before devouring you.

Conditions

Amstrad

x IN n	Is object n in room x? If object 1 is in room 3, as above, then 3 IN 1 will be true, but 4 IN 1 will be false.

Others

r IN O	Is object O in room r? If object 1 is in room 3, as above, then 3 IN 1 will br true, but 4 IN 1 will be false.

QS verbs (All except Amstrad)

For 12 EXITS, read 12 TEXT.

KEY - A = Amstrad
B = BBC
C = C64
S = Spectrum

Appendix B - Words Used In Conditions, Objects

(iii) BBC + Spectrum/(iv) C64

BCS For o in r, read r in o

Appendix A - Table iii - Conditions High Priority

BCS For Res ? read Res?
For Equ ? read Equ?

Verbs - 2.2

BS Para.3, line 1 should read:

"To enter a verb, simply type in a number then a space followed by your verb".

Graphics - 3.2

S G - Displays Attribute grill. Must be held down.

A Section 2

You may have as many connections from a single room as you can fit into 255 characters, all on a single line and followed by RETURN. For example: EAST 20 WEST 18 NORTH 19 SOUTH 21 jump 49 (RETURN).

Appendices A & B

S Where it reads "Tape or Disc", ignore "or Disc".

AMSTRAD HINTS AND TIPS

Control Characters

The Control Characters explained in Chapter 9 of your User Manual can be incorporated into your GAC programs to produce effects on screen and in game.
Control Characters can be entered as rooms or messages using the following notation:

[CTRL + KEY], <PARAMETERS>.

Some useful control characters to remember are:

SET PEN NO.
[CTRL] + [O] n (n is the pen number [0-3])

SET PAPER NO.
[CTRL] + [N] n (n is the paper number [0-3])

Clear screen.

[CTRL] + [L]

Move cursor

[CTRL] + [J]	Down one line (LF)
[CTRL] + [K]	Up 1 line
[CTRL] + [H]	Left 1 character
[CTRL] + [I]	Right 1 character (space)

Inverse Video on/off

[CTRL] + [X] Toggles inverse video on or off

Beep

[CTRL] + [G] Can be used if something important happens.

Change pen colour (INK)

[CTRL] + [\] n xx

n is the pen number (0-3).
x is a letter representing a colour (as in Appendix B of your GAC manual).
NOTE: If two different x's are specified, the ink will be flashing between them.

[CTRL] + []] xx Set border colour

Where x is a letter representing a colour (as above).

Change screen mode.

[CTRL] + [D] n n is mode number (0-2)

This is not advisable with graphics on screen! Note that the Auto-formatter in GAC is set to 40 columns, so in mode 2 only half the screen will be used.

Other Control Characters can be found in Chapter 9 of the User Manual, but their use in GAC is limited.

Graphics

Due to differing ROM routines for line drawing across the CPC range, your graphics may differ on all Amstrad machines. The answer is simple - avoid where at all possible filling with the cursor adjacent to a line, or "fill leakage" may occur when the game is transferred from 664/6128 to 464 or vice versa.
If you own a 464 without a disc drive and are thinking of making your game commercially available, your game will reach a wider audience if you leave 1400 bytes free for the disc drive inherent to other CPC's.

Conditions

With some complicated commands, for example: NO1 SWAP (NO1 + 10) the brackets are removed when the line is entered and the line cannot be re-edited. Replace the missing brackets each time, and everything will work okay!

Advinman Messages (Chapter 5)

1. It is a small battery operated lamp.
2. It is a freshly dead rat. Looks tasty!
3. It is a small silver key.
4. It looks valuable.
5. It is a small lamp, shining brightly.
6. You can't walk through doors you know!
8. The snake wakes up, comes over and bites you. The poison takes hold immediately.
9. Well done! You got out with the gold. You are rich!
10. The door unlocks and swings open.
13. The snake wakes up, comes over and eats the rat. It then returns to its position.
14. The snake, not surprisingly takes a nasty turn and bites you. The venom kills you.
15. It lights up brightly.
16. It goes out.
17. It tastes even better than it looks! Yum Yum!
18. You find nothing much.
19. I'm afraid I don't know what that is.
20. You hear scuffling footsteps nearby.....
21. A giant spider with glowing red eyes leaps from the shadows and neatly severs your head with powerful jaws, before devouring you.

Conditions (Section 4.i)

The command "IN" is documented incorrectly. The corrected version is as follows:

"x IN n	Is object n in room x? If object 1 is in room 3 as above then 3 IN 1 will be true, but 4 IN 1 will be false.

Section 2

The section on exits should include the following:

"You may have as many connections from a single room as you can fit into 255 characters, all on a single line and followed by RETURN, i.e. EAST 20 WEST 18 NORTH 19 SOUTH 21 JUMP 49 (RETURN).

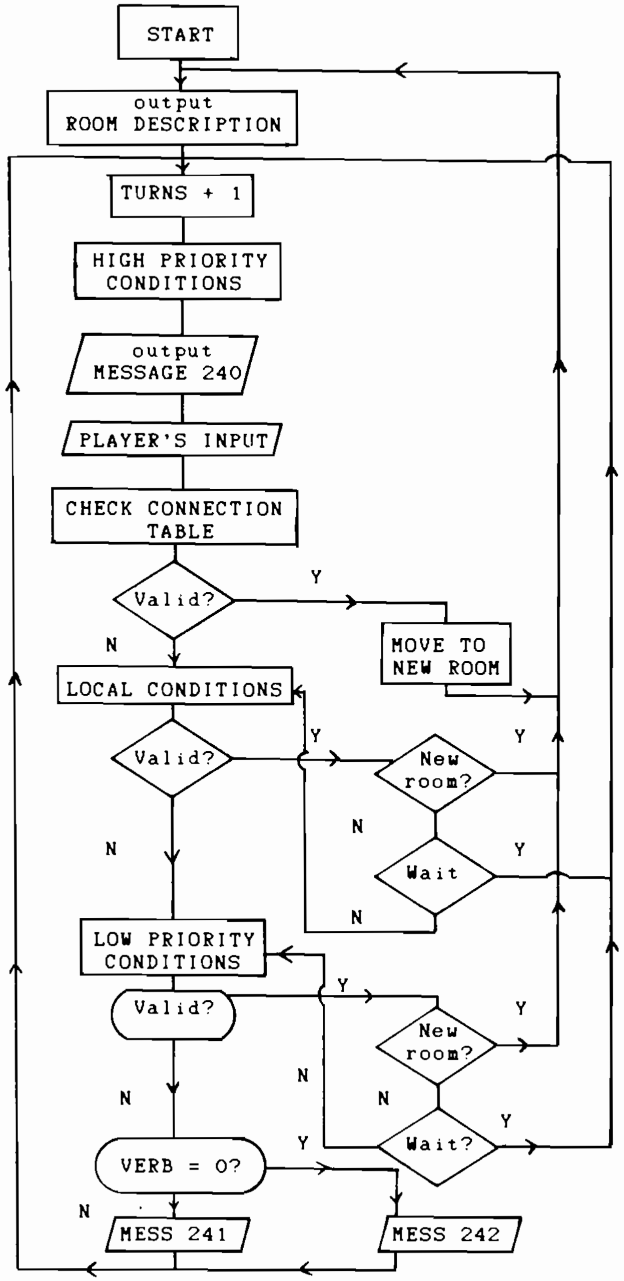
Section 4

The example line should read:

IF (VERB 9) LOOK WAIT END

(Verb 9 = LOOK)

240 : "What now?"



241 : "You can't"
242 : "Pardon?"

The internal commands QUIT and EXIT return the player to the first room with the program initialized. There is no end as such.

BBC HINTS AND TIPS

Message + Goto

When in graphics mode (PICT), the GAC clears the screen when moving from location to location, so if you wish to have a MESSage followed by a GOTO, you will need to have HOLD command of suitable length between the two.

Reset start

When a datafile is loaded into GAC, the Begin Where? value is not, you will therefore either have to begin your adventures in Room 1, (default), or remember to change the Begin Where? value before saving each time.

Border

The border around the Graphics area, whilst using the Graphics Editor will not be present in your runnable adventure. If you wish to have one, simply draw a rectangle encompassing the entire screen area.
Because GAC uses interrupts to produce two modes at the same time, the bottom two pixels may become corrupt. To prevent this, draw a border as above but move it up two pixels from the bottom.

Bug

Sometimes an error may occur with the formatter where the game will print a location description and then lock-up. To prevent this simply alter the length of the offending room description by adding or changing one or two words.

Memory free

There is approximately 18K free for use in your adventures. However a certain amount of memory must be assigned to each part, Graphics and Text.
There must be at least 2K used on Graphics and at least 3K on Text. As the memory is depleted on the GAC Main Menu, a message "available for graphics" will appear, showing how much K there is left for your pictures.

Deleting lines from your game

To delete whole condition lines, rooms etc. Press f0. Delete is only used to change part of a line. i.e.

```
IF ( VERB 9 ) LOOOK WAIT END
```

If we want to get rid of the extra "O" in "look", (so that the line can be entered), move the copy cursor up to the beginning of the line, copy out the line until the offending word is reached and either:
a) move the copy cursor past it before continuing copying, or
b) copy it out and use delete to erase the mistake or unwanted part.

FO will erase the whole thing.

BBC ERRATA

Advman Messages (Appendix A.v)

1. It is a small battery operated lamp.

2. It is a freshly dead rat. Looks tasty!

3. It is a small silver key.

4. It looks valuable.

5. It is a small lamp, shining brightly.

6. You can't walk through doors you know!

8. The snake wakes up, comes over and bites you. The poison takes hold immediately.

9. Well done! You got out with the gold. You are rich!

10. The door unlocks and swings open.

13. The snake wakes up, comes over and eats the rat. It then returns to its position.

14. The snake, not surprisingly takes a nasty turn and bites you. The venom kills you.

15. It lights up brightly.

16. It goes out.

17. It tastes even better than it looks! Yum yum!

18. You find nothing much.

19. I'm afraid I don't know what that is.

20. You hear scuffling footsteps nearby.....

21. A giant spider with glowing red eyes leaps from the shadows and neatly severs your head with powerful jaws, before devouring you.

Conditions (2.7.i)

The command "In" is incorrectly documented. The correct version is as follows:

"r IN o	Is object 0 in room r? If object 1 is in room 3 as above, then 3 IN 1 will be true, but 4 IN 1 will be false.

QS Verbs (Appendix B)

For "12 EXIT" read "12 TEXT".

Appendix B.iii

For "o IN r" read "r IN o".

Appendix A.iii. Conditions

For "RES ?" read "RES?".
For "EQU ?" read "EQU?".

Verbs (2.2)

Para.3 line 1 should read:

"To enter a verb, simply type in a number then a space followed by your verb."

COMMODORE - SPECIFIC HINTS

Message and goto

When in grpahics mode (PICT), the GAC clears the screen when moving from location to location, so if you wish to have a message followed by a GOTO, you will need to have a HOLD command of suitable length between the MESS and the GOTO.

Control Characters (Used in Descriptions and/or Messages)

The Control Characters in Appendix B section ii of the GAC manual can be used for various effects.
Their types and effects are given in the manual. Suggestions for their use are:
- i) To emphasise something - (CTRL G and REVS ON/OFF).
- ii) To change the screen/text colour either to highlight a word, (see above) or to fit the description without a picture, eg. A lava flow description could have red paper and orange/yellow text.

AOT/Mirror

"Mirror" REFLECTS left onto right or top onto bottom. Using Mirror you can save work and memory, especially on borders where only the top right hand corner need be drawn. After Mirroring, other features can be added to prevent the picture from looking symmetrical, (see the Castle picture in RANSOM).

AOT can be used to produce variety in similar pictures, or for views from different angles.
If you wish to mirror right onto left, or bottom onto top, AOT the screen first.
Slight animation can be achieved by "AOTing" virtually symmetrical pictures.

Colour Scheme

If you have the programmers reference guide, you may find the table on page 152 of some use when deciding on text colour etc.
If you haven't, Black and White are the best colours, but experimentation is always the best method.

COMMODORE ERRATA SHEETS

Advinman messages

1. It is a small battery operated lamp.
2. It is a freshly dead rat. Looks tasty!
3. It is a small silver key.
4. It looks valuable.
5. It is a small lamp, shining brightly.
6. You can't walk through doors you know!
8. The snake wakes up, comes over and bites you. The poison takes hold immediately.
9. Well done! You got out with the gold. You are rich!
10. The door unlocks and swings open.
13. The snake wakes up, comes over and eats the rat. It then returns to its position.
14. The snake, not surprisingly takes a nasty turn and bites you. The venom kills you.
15. It lights up brightly.
16. It goes out.
17. It tastes even better than it looks! Yum Yum!
18. You find nothing much.
19. I'm afraid I don't know what that is.
20. You hear scuffling footsteps nearby....
21. A giant spider with glowing red eyes leaps from the shadows and neatly severs your head with powerful jaws, before devouring you.

Conditions (2.8 i)

The command "IN" is incorrectly documented. The correct version is:

"r IN o   Is object o in room r? If object 1 is in room 3 as above then 3 IN 1 will be true, but 4 IN 1 will be false.

QS Verbs (Appendix B)

For "12 EXIT" read "12 TEXT".

Appendix B iv

For o IN r, read r IN o.

Appendix A iii - Conditions

For "Res ?" read "Res?".
For "Equ ?" read "Equ?".

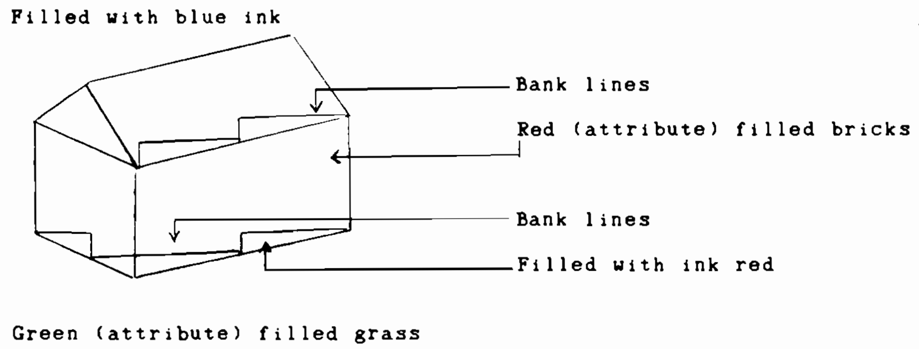
Loading QS and ADVINMAN

To load QS and ADVINMAN, the filenames MUST be typed in <u>CAPITALS without quotation marks.</u>

SPECTRUM HINTS AND TIPS

Graphics

Due to the way the Spectrum handles graphics on screen, only two colours can be present per character square...the paper colour - Attribute and the foreground colour - the ink.
Note: Once something has been drawn on screen in ink, there is no way of re-drawing over it.
With only two colours per square, you may think there isn't much scope for multi-coloured graphics in your adventure, but with careful positioning of your lines and nodes where lines join, many colours may be displayed adjacently.



"Bank lines" are to contain the ink Fill (F). This enables the area outside the bank line to be attribute filled (A) and detail drawn on it. For example, in the house picture above, all the side of the house is filled in red, but the bottom sections are filled in Ink as the other half of the character squares are filled in Attribute green. This enables windows and bricks to be drawn on the rest of the house, and bushes etc to be drawn on the grass in ink. It is a good idea to fill large areas in attribute so that detail may be put on to make it less boring.

It is helpful to draw and plan graphics on squared paper first, and use "G" whilst in graphics mode to bring up the character grid.
For more information on Bank lines and so on see picture 99 of Advinman.

Message and Goto

If you have a message followed by a Goto command, the room description will be printed one line up, obscuring the last line of the message. The answer is to have a line feed command (LF) between the MESS and the GOTO.

Should your datafile become corrupt use the following method of retrieving your datafile:

1. Note memory free
2. Turn Spectrum OFF then ON to clear
3. Clear 40000 (Enter)
4. Load in datafile: Load"" Code
5. Poke (65536 - memfree), 0
6. Poke (65537 - memfree), 0
7. Then Save "Datafile" CODE 42271, (23267 - memfree)
8. Reload GAC and load new file

SPECTRUM ERRATA SHEETS

Advinman messages (Appendix A v.)

1. It is a small battery operated lamp.
2. It is a freshly dead rat. Looks tasty!
3. It is a small silver key.
4. It looks valuable.
5. It is a small lamp, shining brightly.
6. You can't walk through doors you know!
8. The snake wakes up, comes over and bites you. The poison takes hold immediately.
9. Well done! You got out with the gold. You are rich!
10. The door unlocks and swings open.
13. The snake wakes up, comes over and eats the rat. It then returns to its position.
14. The snake, not surprisingly takes a nasty turn and bites you. The venom kills you.
15. It lights up brightly.
16. It goes out.
17. It tastes even better than it looks! Yum Yum!
18. You find nothing much.
19. I'm afraid I don't know what that is.
20. You hear scuffling footsteps nearby....
21. A giant spider with glowing red eyes leaps from the shadows and neatly severs your head with powerful jaws, before devouring you.

Conditions (2.7 i)

The command "IN" is incorrectly documented. The corrected version is as follows:

"r IN o	Is object o IN romm r? If object 1 is in room 3 as above then 3 in 1 will be TRUE, but 4 in 1 will be FALSE."

QS Verbs (Appendix B)

For "12 EXIT" read "12 TEXT".

Appendix B iii

For "o IN r" read "r IN o".

Appendix A iii - Conditions

For "Res ?" read "Res?"
For "Equ ?" read "Equ?"

Verbs 2.2

Para.3, line 1 should read:

"To enter a verb, simply type in a number then a space followed by by your verb."

Graphics 3.2

The command "G" should read:

"G - Displays a grid showing Attribute positions whilst key is held down."

Appendices A and B

Where the text reads "Tape or Disc", ignore the "or Disc".

DE-PROTECTING SPECTRUM ADVENTURES (& ADDING A NEW CHARACTER SET)

1. With an empty Spectrum save out a header on a blank tape.
   SAVE "filename" CODE 24000,41536 (save header only)

2. Position runnable adventure tape after the 2 headers. i.e. just before the large block of code.

3. CLEAR 23999 (enter)

4. LOAD "filename" CODE (to load in header)

5. Load in program code. Ignore any tape loading error that may occur at the end of the load.

6. You now have your program in memory.

7. LOAD "charset" CODE 38785,736 (To load in character set - SPACE to COPYRIGHT)

8. Save out program. SAVE "name" CODE 24000,41536

To load in your program with the new character set :-

```
CLEAR 23999
LOAD "name" CODE
POKE 23606,129
POKE 23607,150
RANDOMIZE USR 34104
```

This information has been kindly provided by Simon Kimberley of 115 Yelverton Road, Radford, Coventry. CV6 4AG. A tape is available from Simon containing a minimum of 9 different character sets varying from old style to ultra modern and a font creator with many features. The price is just 1.49!

ADVENTURE HELP/CLUBS AND MAGAZINES

When writing please help by enclosing a stamped addressed envelope for the reply.

Adventureline Club (The Guiding Light),
52 Micawber Way, Newlands Spring,
Chelmsford, Essex. CM1 4UG

The Adventurers Club Ltd.,
64c Menelik Road, London. NW2 3RH

Orcsbane,
84 Kendal Road, Hillsborough,
Sheffield. S6 4QH

Insight,
41 Union Court, Otley,
W.Yorkshire. LS21 3AS

Adventurers Anonymous,
Rivendale, Nethergate Street,
Bungay, Suffolk. NR3 1HE

Adventure Probe,
78 Merton Road, Wigan. WN3 6AT

Questline Adventure Help,
17 Headley Way, Headington,
Oxford. OX3 0LR

ADVENTURE WRITING HELP

Adventure Contact,
13 Hollington Way,
Wigan. WN3 6LS

ADVERTISEMENT!

Incentives Medallion Adventure Label has been set up specifically for the best GAC generated adventures!
Available now from all leading retailers, or order direct by Mail Order - all programs sent 1st Class post free.

The Legend of Apache Gold- Amstrad, Commodore 64 and Spectrum - £7.95

Winter Wonderland- Amstrad, BBC B, Commodore 64 and Spectrum - £7.95

Also available is a GAC Adventure Designer Pad - designed specifically for orderly information keeping, whilst generating adventures, (approx. 200 sheets) - £7.95

Incentive Software Limited 2 Minerva House - Calleva Park - Aldermaston Berkshire. RG7 4QW