EXPLORER (1986, for 48k ZX Spectrum)

Explorer was my second open world program, after The Forest. The same algorithm was used in both programs to generate limitless terrain from a few hundred bytes of Z80 assembler, in real time as the player moves around. Details of the method were described in design notes for The Forest, under the heading of The Complex Forest.

The present document aims to explain how to use Explorer successfully (37 years too late?) and describes some aspects of the design which were innovative in 1986.

Keypress actions

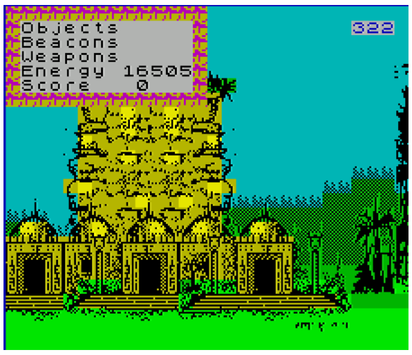
If a joystick is in use it replaces the arrow keys and F for firing a weapon.

On the ground, stationary, no menu:

←	turn left 11°
→	turn right 11°
↓	turn back 180°
↑	move forward if no cliff blocks the way, and keep moving
U	up, to aerial view
other	display main menu

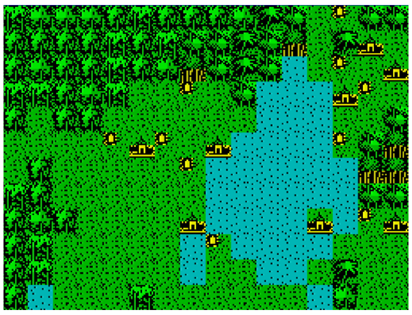
On the ground, moving forward:

←	turn left 11°
→	turn right 11°
↑	move faster
↓	move slower/stop
other	stop



On the ground, main menu displayed:

O	show objects menu (about the missing fragments)
B	show beacons menu (you are carrying some at start)
W	show weapons menu (you have a gun)
X	show credits - cleared again by any key
other	clear this menu



Aerial view:

←	move west
→	move east
↑	move north
↓	move south
U	go higher (once)
D	go down (crash if not on grass)
other	ignored

Objects menu:

E	Echo scan for direction of nearest object
S	Send a found object to a beacon (by number)
other	clear this menu

Beacons menu:

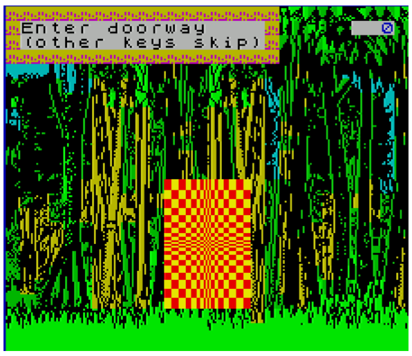
D	drop a beacon
P	pick up a beacon
F	fix 3 nearest beacons
other	clear this menu

Weapons menu (pops up if a beast appears):

←↑→↓	aim
F	fire
other	clear this menu (only possible if there is no beast - you have to kill it)

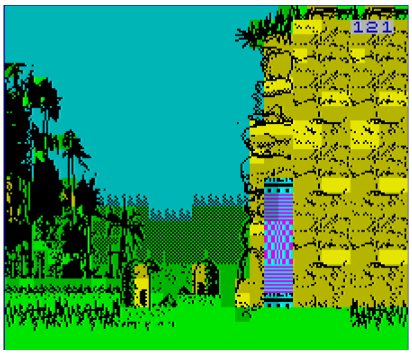
Strategy - some hints and tips

When you have recovered from the initial crash landing of your spaceship the aim is to find the 9 objects which are widely scattered fragments of the ship, so it can be reassembled.



After wandering for a while you will encounter examples of “doorways”. These are teleportation machines. You will be asked for the name of a place you wish to go to. It is important to remember the name you give, so you can always go there again.

You are carrying some beacons. These can be dropped wherever you like. When you find an object (part of the ship) you can send it to a dropped beacon.



In aerial views a dropped beacon is visible as a flashing square.

You can move around more rapidly in the aerial view but you will not see certain things on the ground.

One of the menus has an option to do an “Echo scan” to discover the direction to the nearest object. You will need to move in that direction (bearing, top right of the screen) but you will not be told how far.

You will need to get the 9 objects together in one place to rebuild the ship.

Some pokes

(If you must, in an emulator…)

1. To get back to the main BASIC interpreter loop in ROM (JP 12A2):

```
POKE 65469,195
POKE 65470,162
POKE 65471,18
```

then use the X key when the main menu is displayed

2. When the program was being tested it was necessary to know where certain kinds of things could be found, to see that they worked properly. For example, one of the objects from the spaceship is at coordinates x = 784, y = 30304. Knowing that the

player’s position is held in the addresses 57656 for x and 57658 for y should enable you go to that place to see it. It you do it in BASIC you can GOTO 50 to resume the program. (Did you know that Windows 11 calculator has a programmer setting that is very handy for decimal/hex conversions?)

About the graphics

The scene display is built as 4 layers:

- Sky
- Hazy distant trees
- Mid-ground trees and features
- Foreground trees and features

There are more than 60 component graphics available to be assembled in those layers. All of them can appear either as originally drawn or left-right reversed. There is a grammar for determining how graphics and their reversals may be combined together.

Each component graphic is stored in memory as a string of 10-byte blocks. Each block has the following format.

Byte 1 = attributes

Based on Sinclair standard except for some special cases. The flash and bright bits have a different interpretation in Explorer too.

			Paper			Ink			
			G	R	B	G	R	B	
Bit	7	6	5	4	3	2	1	0	
	0	1	1	1	1	1	1	1	127: skip this character
	1	0	x	x	x	x	x	x	Left edge
	1	1	x	x	x	x	x	x	Right edge
	x	x	1	0	1	0	0	0	Leave previous attribute
	x	x	0	0	1	1	1	1	Ripple water

The value 127 means skip this character, only using byte 10 (see below) to gain offsets bigger than -128..127.

When left and right edges are indicated, the pixel bytes (see below) are handled differently. The left and right edge characters combine with previous contents of the screen when drawn. Pixel bytes are masked by using 256-byte look-up tables (LUTs) held in memory: one for left, another for right. The mask leaves a 1-pixel wide band of paper colour just outside the edge so that black objects on black objects are delineated.

Water rippling is done by a programmed loop while waiting for a keypress.

The bit for flashing has been reused as shown above but sometimes things in the program do flash or are animated in some way. That is all done by program code, redrawing graphics as necessary.

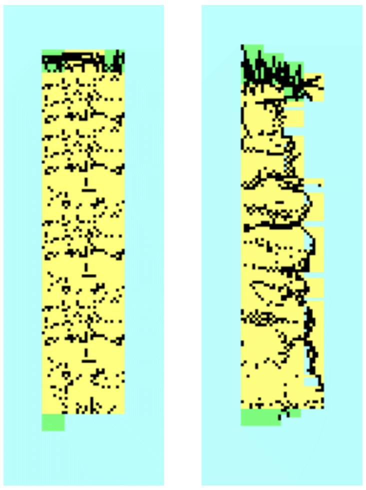
Bytes 2..9 = pixels

Each has 8 pixel bits, as in Sinclair standard: paper 1 or ink 0.

Byte 10 = offset

Offset (-128..+127) to the position in the attributes file (32 x 24) for drawing the next 10-byte block. A value of 0 means the end of the graphic.

Reversing graphics



Often graphics are drawn reversed left to right. This is done by another 256-byte LUT in memory, applied to each of the bytes 1-9 in each block.

For example, foreground cliffs are drawn using the 2 graphics shown here. The left one is represented by character H, the right one by G. A cliff may be drawn in a scene as G* (* means left-right reversed) followed by any number of H followed by a G (not reversed). This is a simple example of the “grammar” for combining graphics.

G clearly has right edge attributes on most of its 81 ten-byte blocks. When reversed the left edge LUT is used.

For those who wish to poke around, the graphics are stored beginning at address 40000. Having saved that memory by using the emulator ZXSpin, I have written a JavaScript program to redraw each graphic in an HTML canvas. That is how I produced the cliff examples here.

Further information

Many sections of code in Explorer are the same as were written for The Forest:

- Terrain is generated the same as for the Complex Forest

- Bearings, clockwise from north, use 1-byte format (256 b-degrees to the circle). Sines and cosines are looked up from a table.

- Players can move freely in any direction unless blocked by a cliff. Coordinates are 3-byte fixed-point numbers with resolution 1/256 metre. Terrain is not tiled, even though the aerial view looks that way.

More details of all these points can be found in TheForest_DesignMaterial.pdf and a link to a copy of that is in the Additional File Downloads section of this page:

https://spectrumcomputing.co.uk/entry/1843/ZX-Spectrum/The_Forest

A higher level, more descriptive explanation of how the terrain generator works can be found by downloading the file TerrainGeneration.pdf from here:

https://grelf.itch.io/terrain

All of this information is freely available to use and perhaps extend the ideas further.