bt Documentation

Release 0.1alpha

btdevel

October 18, 2012

CONTENTS

1 File Formats **3**
- 1.1 General . . . 3
- 1.2 Formats . . . 4

2 MSDOS **7**
- 2.1 File list . . . 7

3 Amiga **9**
- 3.1 File list . . . 9
- 3.2 Tools . . . 9

4 Indices and tables **11**

Contents:

CHAPTER

ONE

FILE FORMATS

1.1 General

1.1.1 Endianness

Long integers in BT are sometimes saved in big and sometimes in little endian format (see [http://en.wikipedia.org/wiki/Endianness]_). In big endian format the most significant byte comes first, e.g. `01 02 5E AE` means `0x01<<24 + 0x02<<16 + 0x5E<<8 + 0xAE` in big endian and `0x01 + 0x02<<8 + 0x5E<<16 + 0xAE<<24` in little endian. Big endian is what humans are wont to read and what is used on PCs. In BT usage is a bit arbitrary, e.g. in the MSDOS version the file `dpics0` starts with `00 00 56 58 57 5a 01 00`, i.e. the first long is big endian and the second one little endian, while `dpics1` starts with `00 00 56 58 00 01 6a a2`, where both longs are big endian.

In BT often the following strategy works: decode the long value as big endian and as little endian and take the smaller of both. Only in cases where you expect the values to be really large, specify endianness explicitly. Code example:

```
def read_long_big(byte_arr, offset):
    return  b[0] << 24 | b[1] << 16 | b[2] << 8 | b[3]

def read_long_little(byte_arr, offset):
    return  b[0] | b[1] << 8 | b[2] << 16 | b[3] << 14

def read_long(byte_arr, offset, endian=GUESS):
    return min(read_long_big(byte_arr, offset),
               read_long_little(byte_arr, offset)
```

1.1.2 Compressed Files (Huffman encoding)

Compression in BT is mostly done with Huffman encoding [http://en.wikipedia.org/wiki/Huffman_coding]_. Any chunk of data that is Huffman encoded, has the following structure:

- a long (4 bytes, mostly big endian) describing the number of bytes

of decompressed data * a long (4 bytes, big or little endian) describing the number of `'bits'` of compressed data including the Huffman tree * the Huffman tree * the compressed data

The Huffman tree and the compressed data must be read bit by bit always started with the highest bit of each byte. Both are directly adjacent.

Note: Description of Huffman encoding goes here

1.1.3 Indexed Files

Description of indexed files goes here

1.2 Formats

1.2.1 Levels

In BT1:

- Stored in 'levs' file, as indexed/compressed file, contains 15 chunks of data, with:
  - 0 = Wine Cellar
  - 1 = Sewers 1
  - 2 = Sewers 2
  - 3 = Sewers 3
  - 4 = Catacombs 1
  - 5 = Catacombs 2
  - 6 = Catacombs 3
  - 7 = Harkyn's Castle 1
  - 8 = Harkyn's Castle 2
  - 9 = Harkyn's Castle 3
  - 10 = Kylearan's Tower
  - 11 = Mangar's Tower 1
  - 12 = Mangar's Tower 2
  - 13 = Mangar's Tower 3
  - 14 = Mangar's Tower 4
  - 15 = Mangar's Tower 5
- First 22*22 byte: wall data. Ordering of bytes: west to east (fast), south to north (slow), zeroth byte (0N, 0E), first byte (0N, 1E), 22nd byte (1N, 0E),
- Each byte contains 4 pairs of bits in the ordering: W, E, S, N. Meaning:
  - 00 -> Nothing
  - 01 -> Wall
  - 10 -> Door
  - 11 -> Secret Door
- Flags (following 484 bytes):
  - bit 0 is set if there are stairs up.
  - bit 1 is set if there are stairs down.
  - bit 2 is set if there is a special (this includes spinners, magic squares, messages, magic mouths, etc. Everything that's not covered by one of the other bits.)

  - bit 3 is set if there's darkness
  - bit 4 is set if there's a trap.
  - bit 5 is set if there's a portal down
  - bit 6 is set if there's a portal up
  - bit 7 is set if there's a random encounter scheduled for this tile.

- Thread: http://brotherhood.de/Bardstale/talefiles/board/viewtopic.php?t=788

1.2.2 City

1.2.3 Graphics

CHAPTER

TWO

MSDOS

2.1 File list

The MSDOS version contains the following files:

47 Probably the guild images

b0.huf Image for House type 0, compressed, using CGA palette

b1.huf Image for House type 1, compressed, using CGA palette

b2.huf Image for House type 2, compressed, using CGA palette

b3.huf Image for House type 3, compressed, using CGA palette

bard.exe The executable, can be used to locate game strings, other game data is difficult to obtain as the exe file is compressed (Does anybody know the decompression alg? Otherwise memdumps can be used to access that data; use the debug version of dosbox and the do a memdumpbin)

bardscr The main game screen, compressed, using CGA palette, the image is color separated: first come all blue pixel values, then all green pixel values, then red, and then the one for highlight.

bardtit The title screen. Format is the same as bardscr.

bigpic Indexed/compressed file containing all the monsters, images inside buildings, probably also city pictures except the big images of the four house types.

cga_scr Not checked yet

city.nam Contains an index to the street names. Compressed.

city.pat Contains an index to the house numbers and specials in Skara Brae.

citypics.bin Not checked yet.

color.cmp No clue.

color.rgb No clue.

comp No clue.

comp_tit No clue.

dpics0 Contains the dungeon pictures for the standard dungeons. Image data is partitioned with partitioning like:

[(56, 88), (192, 86), (120, 54), (80, 33), (48, 17), (16, 40), (32, 88), (48, 85), (32, 52), (16, 31), (16, 15), (48, 32), (40, 18), (80, 8)]

dpics1 Dungeon pictures for the dungeon levels in Mangar's tower. Otherwise like dpics0 (endianess is a bit different for the first two longs here, than in dpics0 and dpics2

dpics2 Dungeon pictures for the dungeon levels in the Catacombs. Otherwise like dpics0.

graphics.drv Probably uninteresting

icons.bin Not checked.

items Contains the items available in Garth's equipment store. The value of byte x corresponds to the number of items Garth has to sell of item type x. A value of `0xFF` means an infinte supply.

levs Contains the levels. File is indexed and compressed. Longer description needs to be given elsewhere.

rgb Not checked.

rgb_tit Not checked.

tdy Not checked.

tdy_scr Not checked.

tdy_tit Not checked.

CHAPTER

THREE

AMIGA

Description of Amiga files goes here

3.1 File list

3.2 Tools

Useful tools for the amiga: * UAE * unadf * uae_readdisk * http://www.amigaemulator.org/useful * adflib * adfopus (windows-only I guess)

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*