The File Formats of WarCraft: Orcs & Humans

December 4, 2015

Contents

1 The Archive Format

WarCraft: Orcs & Humans uses a custom archive format called .WAR, possibly short for **War**Craft or **W**arCraft **Ar**chive. The specifications of the format changed between versions of the game, so it is important to take a look at various releases.

1.1 Structure of a .WAR file

War files consist of 3 parts: Header, File Table and Data.

1.1.1 The Header

The header is pretty short and consists of an Archive ID and the number of file entries that are to be found in the File Table. PreRelease archives to not feature the Archive ID. Following is a list of all known archive headers:

Version	Archive ID	Number of Entries
PreRelease		`EF 01 00 00`
DOS Retail	`18 00 00 00`	`47 02 00 00`
DOS Shareware	`19 00 00 00`	`47 02 00 00`
Mac Retail	`00 00 00 1A`	`00 00 02 47`
Mac Shareware	`00 00 00 19`	`00 00 02 47`

1.1.2 The File Table

The File Table holds offsets to all the files that are stored inside the Data section. Files are not named, so their index in the File Table has to be fixed, which means that stripped down versions of the game have placeholders.

Placeholders in the PreRelease demos and DOS Shareware are `FF FF FF FF` and under Mac they are `00 00 00 00`.

In the retail version they are marked by a follwing offset just 1 greater.

1.1.3 The Data

Each non-placeholder data entry begins with its unpacked size as a 4 byte integer. If the third highest bit (`20 00 00 00`) is set, the file is compressed, else it's just raw data. `filesize & 0x1FFFFFFF` returns the correct size of the file. The length of the data is calculated as `offsets[n+1]-offsets[n]-4`, using the size of the .WAR file as final offset, as usual.

1.2 Decompression

The DOS version archives of WarCraft are compressed using a sort of LZ compression. This means that at compression time, the algorithm checked if there was the exact same sequence of bytes previously written, as is being written now.
WarCraft's exact algorithm for decompressing LZ packed data is this the following:

```
for o:=0 to 4095 do
        bufwin[o]:=0; \\init our 4096 byte buffer with zero
i:=0;

while (i<filesize) do
begin;
        warfile.read(cmask,1);
        i:=i+1;
        for a:=0 to 7 do
        begin;
                if (cmask mod 2=1) then //uncompressed byte
                begin;
                warfile.read(bufbyte,1);
                bufwin[tmp.position mod 4096]:=bufbyte;
                tmp.write(bufbyte,1);
                i:=i+1;
                end
                else //compressed block
                begin;
                        warfile.read(offset,2);
                        numbytes:=offset div 4096;
                        offset:=offset mod 4096;
                        i:=i+2;
                        for m:=0 to numbytes+2 do
                        begin;
                        bufbyte:=bufwin[(offset+m)mod 4096];
                        bufwin[(tmp.position)mod 4096]:=bufbyte;
                        tmp.write(bufbyte,1);
                        end;
                end;
        cmask:=cmask div 2;
        end;
end;
tmp.size:=finalsize; \\Crop the file, just in case
```

1.3 List Files

Since the .WAR files don't contain file names, an external list of names, descriptions or at least type detection heuristics have to be provided in order to make sense of them. Both, WarDraft and Hallfiry's Warcraft 1 extractor use external List Files to identify the contents of .WAR files (Hallfiry's extractor also has heuristics for the archives that haven't been manually indexed, yet).

Not all versions have been examined so far, but this is what is known:

Version	List File
Oct04Demo Oct06Demo Interplay Demo	Demo
DOS Retail DOS Shareware DOS CD Release 1 DOS CD Release 2 FormGen Shareware Creative Labs Shareware	Retail
Mac Retail Mac Shareware	Mac
TEN Version	N/A
MPlayer Version	N/A

2 Image Formats

WarCraft uses three major image formats for regular images, sprite sheets and cursors. I've dubbed those .IMG, .SPR and .CUR for easy reference and naming. All images require a 256 color palette to be displayed.

2.1 Palettes

There are three palette formats used by different versions of the game. The DOS versions' palettes have no header and are to be read as RGB byte values between 0 and 64, each, so they need to be multiplied by 4 to get full 8 bit values for each channel. Those DOS palettes can also be halved and contain only 128 colors instead of 256.

Mac palettes always have 256 colors. They begin with an 8 byte header, followed by a list of all colors. Each entry in that list consists of the color index as a 2 byte integer, followed by RRGGBB, with each color value written twice, so one byte for each channel can be ignored. An image will typically require two palettes, one for the lower half of indices and one for the greater half. The lower half is usually a terrain palette.

2.2 The .IMG Format

Image files start with a 2 byte width and a 2 byte height, followed by the pixels of the image as 1 byte color indices from the corresponding palette.

2.3 The .CUR Format

Cursors start with x and y offset, each as 2 byte integer, followed by the usual structure of an .IMG file.

2.4 The .SPR Format

Sprite sheet files start with a 2 byte integer telling the number of frames inside the file, followed by the sprite dimensions as 1 byte width and height. Next is a list of all frames, starting with their y and x offset, followed by width and height, each as 1 byte value. Last comes the offset of the frame inside the file, stored as 4 byte integer.

If the width times height is greater than the difference between this and the next offset, then the frame is compressed as specified below. Else it is to be read as a usual indexed 256 color bitmap.

2.4.1 Sprite Decompression

Sprites of the Mac version are often compressed with an RLE method. Only transparency is compressed and the compression is linewise.

Lines are assembled by reading them blockwise, where each block starts with a single byte Head, that gives further instructions:

- `0x00` → EndOfLine
- `0xFF` → EndOfFrame
- `0x80 & Head = 0` → Head-many uncompressed pixels follow
- `0x80 & Head != 0` → ((0x7F & Head) + 1)-many transparent pixels

3 The Map Format

Todo

4 The Dialog Format

Todo