ZT1 Graphics

# refers to index, numbering in this case starts with 0.

# Goals

Understanding how ZT1 graphic files work. We will use **N** and **N.pal** for reference.

* + Location of palette
  + Width/height
  + Transparent pixel
  + Adjusting positions (x/y)
  + Frames
  + Animation speed
  + Color replacements

# Experiments and research

## Palettes

### Blue8.pal

Blue16.pal has a similar structure.

4 bytes: 09 00 00 00 . First byte determines how many series of 4 bytes follow in this palette file. Basically this means: number of colors -1.

4 bytes: FF 00 FF 00 . Filler bytes? That was our original thought. It is however likely that we might have 9 colors, although we have to figure out what the first one means.

+ 4 bytes per color: 00 11 22 FF .  
00 = red  
11 = green  
22 = blue

### Red\_100x50.pal, red\_100x100.pal, red\_50x100.pal

Identical files.

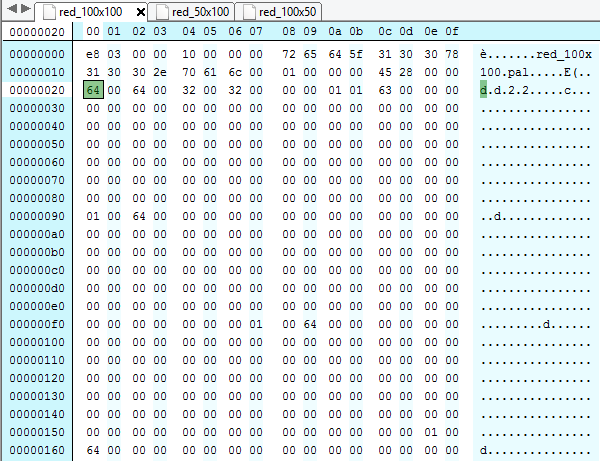
4 bytes: 00 01 00 00 . Either this means one color, or it means 256 colors (reverse bytes: 01 00 = 256). It should be noted that there are lots of “empty” colors listed in the file ( 00 00 00 FF ), and it would be the correct length of the file.

After the first four bytes, the color series start!

## Graphic files

### Red\_100x100

One red color, 100x100, **but one transparent pixel (top left, white)**



E8 03 00 00 : ? At this point it remains a mystery. It does **not** seem to be the length of the file, or the number of byte blocks. **It seems to be the animation speed, in milliseconds**, based on experiments with ZOOT**.**  
  
10 00 00 00 : Defines the length of the location (amount of characters), starting from the next block. HEX 10 = DEC **16**. For some reason, red\_100x100.pal is only **15** characters. So there’s +1 for some reason.

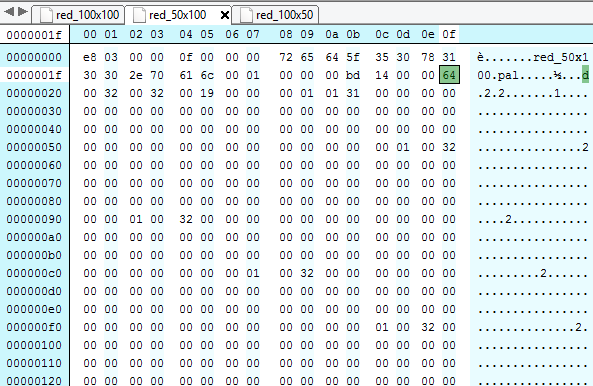
The marked byte (‘64’) and the next one (‘64’) define the image **height** and the next one defines the **width**.

I’m currently unsure about most other bytes which follow, until you have at some point repetitive blocks. It probably is a row or column of points, with 64 indicating the start or end.

ZOOT lists the default offsets as 50x50 . Translated, this means: half of the height, half of the width. Which results in our bytes **32 00 32 00**.

By manipulation, we alter this to 50x-50 . The bytes then change to **CE FF 32 00** .This confirms: first height, then width. We also changed the animation speed, but this speed is not changed by ZOOT (although it might be present) if we only have 1 frame.

### Red\_50x100

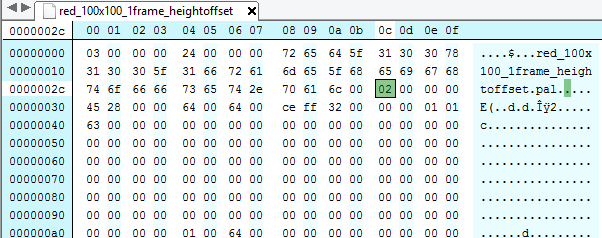


DEC 100 = HEX 64 = height.  
DEC 50 = HEX 32 = width.

A bit further, we also see 00 32 00 19 , or in the red\_100x50-version: 00 19 00 32 . So this has to do with **width/height too, or with position**.

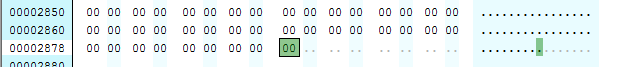
Here we see the same pattern, but it’s **32** (the width) which returns. So it most likely signals the beginning or end of **a line**. It’s most likely the beginning. There’s a byte, value 31 at the beginning of the file, which in the previous example has 63 as value. This could be a reference to skip our first transparent pixel. Which leads to a new question: how do we define other transparent stuff? Would it be merely color based? Is our transparent color (FF FF FF) listed in here explicitly or by an index number?  
  
**00 01 00 64** or **00 01 00 32** in this example seems to indicate the beginning of the row. For the first row (with the transparent pixel), this is **00 01 01 63** or **00 01 01 63**.

### Red\_100x100 with 2 frames

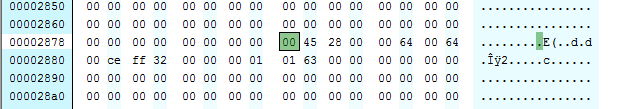


This seems to be the byte indicating the number of frames.

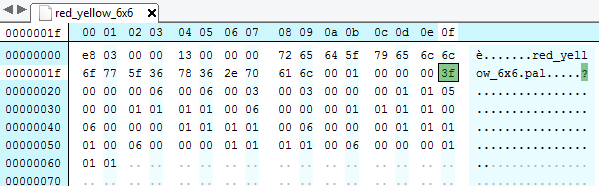
If we compare the file with 1 frame, it ends here:



Now, it seems to restart with this. We again have the height, width, x, y. But more importantly, there are more bytes which seem to matter. In our case: **00 45 28 00** . Their use is still unknown at this stage.



#### Red\_yellow\_6x6

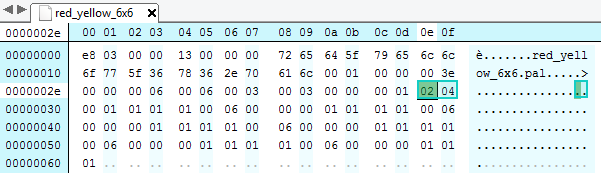


**HEX 3F = DEC 63**. No idea yet what this refers to. Not our amount of pixels (6x6 = 36 – 1 transparent pixel).

What we know: **00 03 00 03** = offset. After that, we have   
**00 00 00 01 01 05** . Some of this is unknown, 05 refers to the fact that the next 5 bytes are pixel colors. 01 most likely to the place where we start drawing (= ‘skip the first pixel, move to the second spot’).

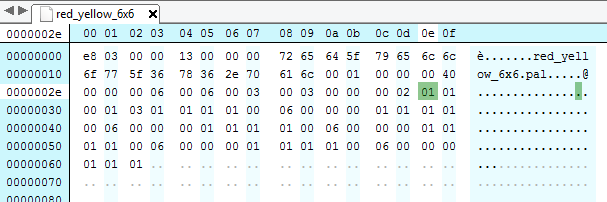
Starting from below: we have an instruction to write a line (left to right), with 6 pixels: **00 00 00** (=3 pixels with color #0) **01 01 01** (=3 pixels with color #1). The **06** in front probably determines where to start. **00** might signal: all the way to the left.

We will now create the same image, but with the top left pixel + the one to the right of it, as transparent.



The marked bytes confirm our theory: in the first row, skip the first two pixels (=transparent). Color the third pixel. Four colors follow: 1x red (color 00) and 3x yellow (color 01).

Another variation: the first and third pixel of the first row are transparent.

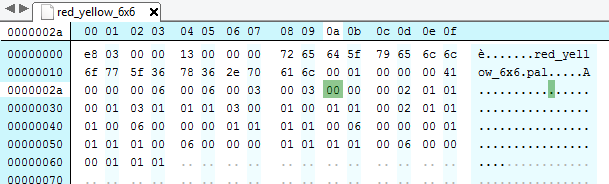


Observations: we have 2 additional bytes.   
This could be explained by instructions to skip drawing of pixels.

**(00 02) 01 01 00 01 03 01 01 01 01 (00) (06)  
  
  
00 02: unknown. Does 02 refer to the amount of transparent pixels in this row?  
  
01: Skip 1 pixel (transparent). It’s likely that this one is always skipped.  
01 00: Draw 1 pixel with color #0.  
01: Skip 1 pixel (transparent?).   
03 01 01 01: Draw 3 pixels with color #1.**The last part might be a closing part? **AFTER** highlighted byte: instruction: ‘move to index 1 = ‘pixel 2’ in the row (skipping index 0). Color this with color #0.’   
**00:** might mean the end of this instruction? This means that **01**

#### Alternative

6x6. Pixels #0 to #2 are red, rest are yellow, but:  
Row #0: pixels #0, #2 are transparent.  
Row #1: pixels #1, #3, #4 are transparent.



Let’s start with our last row. It reads like this. I’m not sure yet whether the (00) is still with the last row or not. It most likely refers to a potential offset. In the last case: start with pixel #0.

**HEX 41 = DEC 65** is the mystery byte here.  
**It’s likely that it refers to the number of bytes which come after it? Probably we should not take the next 00 (00 00) into account). It’s likely that this tells us how many bytes are dedicated to a frame.  
00 41 00 00.** In one of our other examples, this is **00 BD 14 00.**

**(00) 06 00 00 00 01 01 01 –** draw 6 pixels. 3x color #0, 3x color #1.

Before that, we should spot the same pattern twice:  
  
**(00) 06 00 00 00 01 01 01 01** (or **01 00**. This seems to signal ‘end of row’, although I was actually expecting that might be derived from the width of the image. One explanation might be that you couldn’t possibly know how many pixels are non-transparent in this line and that it can’t be found anywhere in this graphics file. But in that last case: what if the last pixel drawn was just one pixel with one color? Then we would also have 01 + color index).

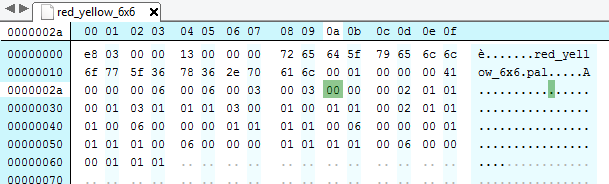
**03** : there are 3 transparent pixels.  
**00**

**00** : draw transparent pixel? **02 01 01 (01)** : draw 2 pixels, color #1. (End of line).

**Pixel row 1.**(00) **02** : there are 2 transparent pixels.  
**01** : move to pixel #1 / skip 1 pixel.  
01 00 : color on, 1x color #0  
01 : skip 1 pixel.  
**03** 01 01 01 : color on, 3x color #1.  
  
2 + 1 + 3 = width 6. ZT1 knows the line ends here.  
  
(01 00) : end of line?

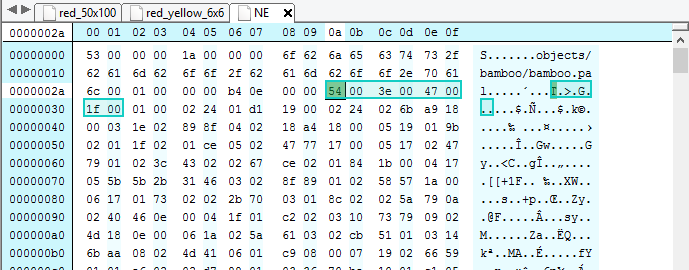
**Pixel row 2.  
03** : there are 3 transparent pixels.  
00 : don’t skip pixels, just start.  
01 00 : color on, 1x color #0.  
 : = draw 1st pixel in red  
**01** : skip 1 pixel.  
01 00 : color on, 1x color #0.  
 : = draw 3rd pixel in red  
**02** : skip two pixels.  
 : = transparent pixels  
01 01 : color on, 1x color #1.  
 : = draw 6th pixel in yellow  
  
3 + 1 + 2 = width 6.

(01 00) : end of line?



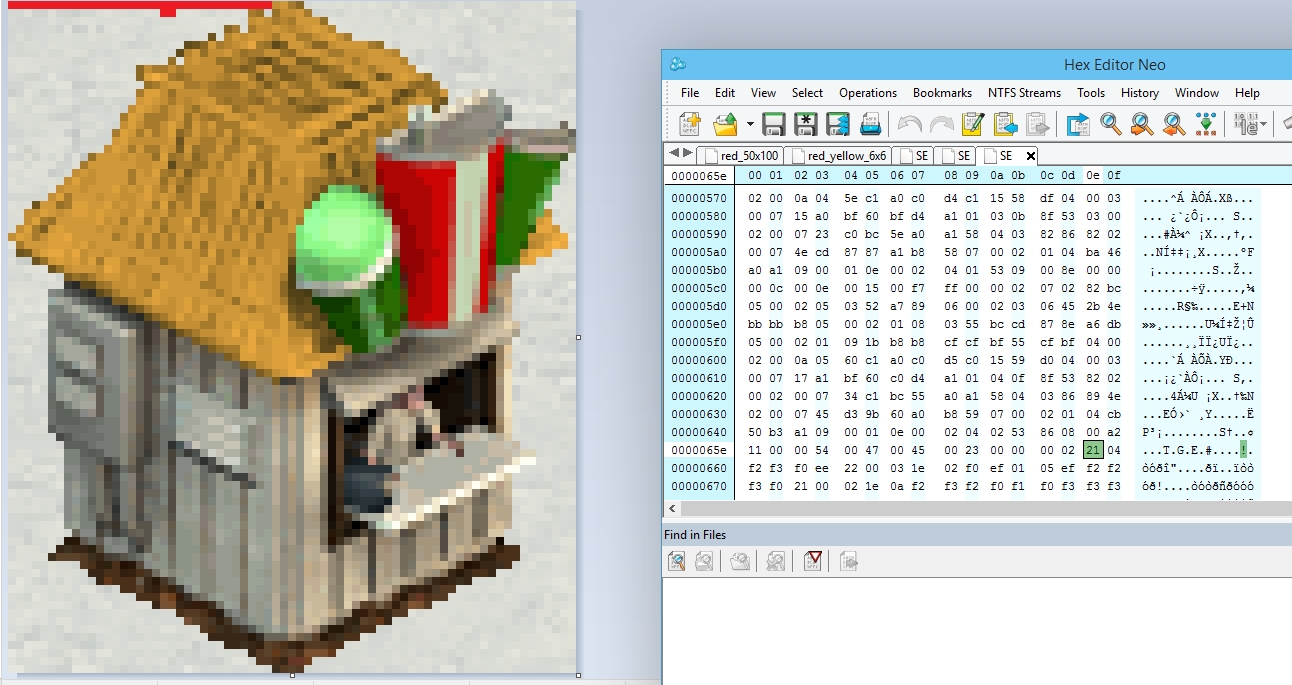
#### Bamboo.

This one seems to be different, we can’t render it properly.



We’ve marked the width, height, offsets.  
Width: 3E (HEX) = 62 (DEC),   
Height: 54 (HEX) = 84 (DEC).

#### Drinkstand



02 21 04 : 2 blocks. 21 offset. 4 color bytes.  
This also returns at the end of the file (bottom line here).

HEX 21 = DEC 33  
+ 4 colors   
HEX 22 = DEC 34

DEC 71 = HEX 47 (width), which matches the length of the building **without** (???) the sign. Although the 71 only makes sense **with** it.

DEC 84 = HEX 54 (height)

 Render of last part of drink stand.

## Unknown formats

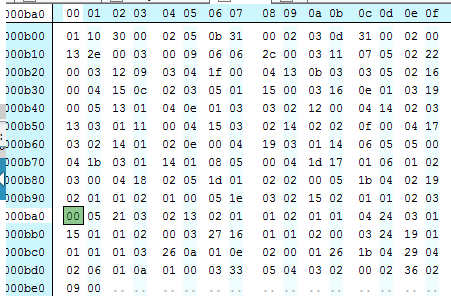
* Icons
* (list icons?)
* Shadows MM (1 color, so compressed format?)

Arctic Wolf, guest male, attack animation:   
#color 163 = yellowish. In Arctic Wolf. In color.pal, same.

#### Different type

The submerged graphics for the dolphin have weird values in the spots in the bytes which come along with the width and height. Contrary to the Beluga, these graphics are different than the usual “basic” images. The beluga’s files even don’t have the FATZ-type of animation. The beluga’s shadows are opaque black, and the color is altered by the game engine. The dolphins have the FATZ-type, with no special value at the beginning.

2C, at the end of the file, could be a key byte. The color would refer to a green color in dolphin.pal. So we could assume it’s either an offset or a number of pixels to expect.



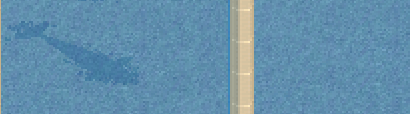
Second row, middle: 2C. We could assume it is not an instruction to skip X pixels, since we can’t draw “00” pixels afterwards. Conclusion: it is the number of bytes which represent a drawing. So 06 is our number of pixels we skip, before we draw 44 pixels. The 06 in front of that, **could** be the number of blocks.

This would make 44 pixels out of a total of about 70 pixels we’re looking for. The theory does not make sense if we also consider the 2E in the second row, which is also a green color. 2D is grayish.

Could it be that the format is entirely different, and that we’re having to deal with transparency as well (a color “00”, or levels of transparency defined?)

**Another approach** is starting from below. The use of 00 to show transparency, is unlikely, or there would only be 1 transparent pixel in that row. 3rd row from the bottom: 26 is a brown color. It’s unlikely to occur in a shadow image. We’re assuming an offset for this one. 0A would be 10 pixels, which would lead us to the next pattern of:  
  
 1 offset, 10 colors; = 11  
 2 offset, 6 colors; = 8  
 5 offset; 4 colors; = 9  
36 offset, 2 colors. = 38

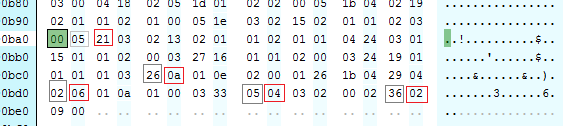
We need to get to a sum of about 70 pixels, based on the **SE**-image we see in **Marine Mania**. Without analyzing every pixel, but at glance,

  
Left: with shadow, right: without

Another approach. Our previous gues seems weird, in the last row, we aren’t expecting that many pixels. Still, the color 00 does not make sense at this point (it’s displayed as pink).

33 05 04 03 02 00 02  
36 02 09 00

Sort of makes sense:



Beluga has 25 frames, all black.

Could this make sense?01 26 1B **1 block,** +26px, +1B black pixels (nothing else!)   
= 41 HEX = 65 DEC  
  
  
04 29 04 02 06 01 0A 01 00  
**4 blocks,** +29px, +04 black, +02px, +06 black, +01px, +0A black, +01, 0px  
= 41 HEX = 65 DEC

03 33 05 04 03 02 00  
**3 blocks**, +33px, +5 black pixels, +4px, + 3 black pixels, (+2px), EOL  
= 41 HEX = 65 DEC  
  
02 36 02 09 00  
**2 blocks**, +36px, +2 black pixels, (+9px), EOL  
= 41 HEX = 65 DEC

#### Mystery byte

We read the SE-view of bamboo.  
We wrote it with Zoot. There’s a difference of a byte.   
The significance is unknown. Could it refer to the number of the frame?

