DT1 format

Paul SIRAMY, 30 December 2002

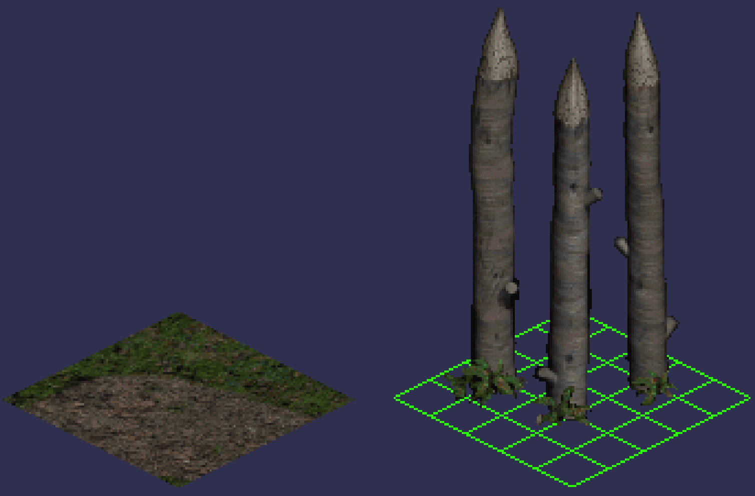
Before anything else, I really want to thanks **Clannad** for his original documentation of the DT1 format. Without the doc he made in the first place, in the old Phrozen Keep's Forums, I shouldn't have be able to decode further more the DT1 format, as I surely have give up. He made a DT1 viewer, and he could have kept his infos for himself, to not beeing afraid of rivals. Thanks Clannad to have share your knowledge with the world (as I'm doing right now), the current state of the Moding scene is also a part of your work (personal note : this is *this* doc that really throw me into the Diablo 2 Modding scene, and this was the start of many tools I have made since. If you hadn't made it, I shouldn't even be here today).

Introduction

Ok, so what are the DT1 files ? They are all the **D**iablo 2 **T**iles that are used for the floors and walls of the maps (a Tile beeing the gfx element of a map). Or maybe the **T** stand in fact for **T**extures ? Well, that's not important. There are 256 DT1 files in the mpq, for a total of 157 MB (if you don't have LOD, that's less of course). But note that some are not used by the game in fact. Each DT1 is a collection of bitmaps of the same theme, like **Tristram**, **Catacombs**, **Crypt**, and so on. Their main path is from **Data\Global\Tiles**. In this directory there is another sub-division : 1 directory for each act. And again in each Act directory, there's a directory structure (different for each acts). For instance, the file **Data\Global\Tiles\Act1\Tristram\Town.dt1** have all the Tiles specific to Tristram.

One thing to remember is that a DT1 is usually used by many maps. Let's take the stone walls you see in the Rogue Encampment. There are some such walls in this town. But you can see the exact same walls in the Cold Plains, in Stony Field, in Tristram... So if you want to edit a DT1, don't forget your changes will alter more than 1 map, so be carefull.

Another usefull things to know : Diablo 2 is a 3D-isometric game, and therefore the Tiles of the DT1 are also in a 3D-isometric shape. The walls are too in a 3D-isometric shape, even if it's less obvious for some of them, like this fence, this is why I have added a green rule under it for this example :



Concepts

Since we'll discuss the format of a DT1 in all its parts, you need to know more.

- The *order* of the Tiles in a DT1 is unsignificant. You don't have to worry about *this* part. But what identify a Tile in a DT1 is 3 indexes, and they **are** a source of troubles. This part will be explain later.

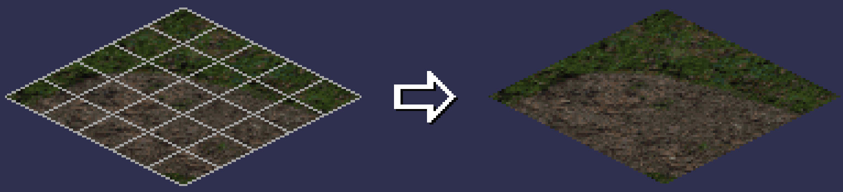
- There are many kind of Tiles in the game : Floors, Animated Floors, Walls, Walls that go into the Abyss (these one are draw *below* the Floors), Roofs, Shadows, and even some Special Tiles that are used for Warping and such. All these different types of Tiles can be put into 1 DT1 with no problems, there's no restrictions for that.

- A Tile is divided into many more sub-tiles. Take the fence of the precedent example, the one with the green rule. I have draw a rule of 5 by 5 sub-tiles. It was in purpose, this is to show you the tiniest element of a Tile. EACH one of this sub-tile can be walkable or not, can let the light go thru or not, can let the player jump over it or not... The DT1 format not only have the gfx data of a Tile, it also have all of its sub-tiles infos, so don't expect to make a Tile from scratch with no problems (at least given the current Tools we have at our diposal today). In the other hand, that means you can change the way a Tile is working by just changing some flags of its sub-tiles... Always dream to walk on the water, or go thru a wall ? By editing a DT1 you can, this is just some flags to modify.

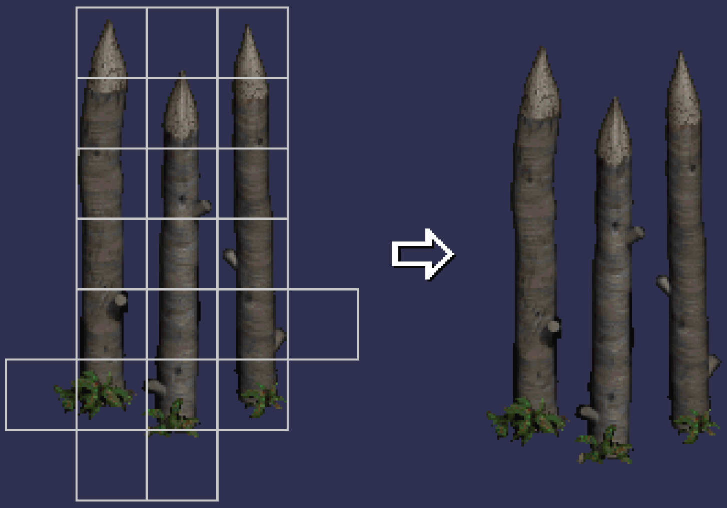
- In addition of these sub-tiles data, a Tile have some other data, like the Sound index to use when the player walk / run over it, and maybe some infos to tell which are the environment effects the game can make over them (like river effect, rain...). It may have a sprite index to use for the minimap too. So just modify a gfx of a Tile is really not enough. If you change a wood floor by your own gfx of stone floor, and only the gfx, don't be surprised when you'll walk over this Tile : you'll still hear the wood sound. So again be carefull when you want to modify a Tile : think of all the datas of this Tile that you may have to edit as well.

- Before we look at the file format itself, you have to know how the Tiles are handle in there : the Tiles are split into several smaller parts. And these smaller parts are not always of the same type. From a graphic point of view, there are 2 main different types, one for Floors, the other for Walls :

    - Floors are split into a maximum of 25 blocks, which are the 25 sub-tiles :



    - Walls are split into a variable amount of blocks of 32 by 32 pixels each :



Each one of this 2 type is coded into its own format. Wall blocks are coded into a kind of simple RLE (Run Length Encoding) format, something like the PCX format : this is to handle the transparency of the walls. As for the Floors blocks... usually a Floor is not transparent, and when 2 successives pixels are of the same color that's an exception. So the Floor blocks are coded into a RAW format, no compression at all. But what happens if the Floor have some transparent area in it ? There are some such Floors, like the borders of a floor near the animated lava in act 4. In this case we'll have a Floor Tile wich have the 2 types of blocks in its data, and you'll see later that it isn't a problem.

In summary, as far as I know, Wall Tiles are only using Wall blocks (32 by 32 RLE pixels), and Floor Tiles are using both type of blocks : Walls blocks and RAW blocks (these RAW Floor blocks are in a 3d-isometric shape too, as you can see).

- Just to make a thing clear : Walls have 25 sub-tiles datas, just like the Floors, even if you don't see some floor gfx on them ! I remind you that these flags tells if a sub-tile can let the player walk over it or not, if it block the light or not, and such. So a wall still need theses infos for each one of its 25 sub-tiles : this is the only way to know where are the parts of the wall that are all solid (no walk, no jump, no light ...) and where are the others where the player can still walk (a door for instance). What about a barrel in the left part of a Tile ? It's an 'object' that can be placed on many floors, but it don't block all the area of the Tile, only 1 sub-tile. Don't mix up the 25 sub-tiles of Floors (and Walls), with the gfx block of a Wall (variable amount). For Floors it's almost the same, but not for the Walls.

File Format

Structure of a DT1

- File Header
- X Tile Headers
- X Tile Data, each one structured like that :
    - Y Block Headers
    - Y Block Data

File Header (276 bytes)

The file start as folow :

# Bytes	Description
4	Version (= 7)
4	Version (= 6)
260	All zero (reserved for future use ?)
4	Number of Tiles
4	Pointer in file to Tile Headers (= 276)

There are some DT1 with the first 2 DWORD not equal to 7 and 6, they are in a variation of the format I expose here. It seems they are some preliminary works that have been left in the mpq, but it's just a guess as I never be able to decode them completly. In any case, if a DT1 don't start with the DWORD 7 and 6, obviously don't try to read the datas with the current doc, as you'll get garbage.

After this File Header follow a variable amount of Tile Headers. Of course there is 1 Tile Header for each Tile there is in the DT1 (4th data of the File Header). Each one of this header is structured as follow :

Tile Header (96 bytes)

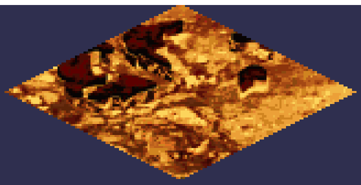
# Bytes	Description	Comment
4	Direction	'General' orientation
2	Roof height	In pixels
1	Sound index	
1	Animated ?	Flag
4	Height	in pixels, always power of 32, always a negative number
4	Width	in pixels, always power of 32
4	Zeros	Unused
4	Orientation	The 3 indexes that identify a Tile
4	Main Index	
4	Sub Index	
4	Rarity / Frame index	Only Frame index in an Animated Floor Tile
1	Unknown 1	Seems to always be the same for all the Tiles of the DT1
1	Unknown 2	
1	Unknown 3	
1	Unknown 4	
25	Sub-tiles flags	Left to Right, and Bottom to Up
7	Zeros	Unused
4	Block Headers Pointer	Pointer in file to Block Headers for this Tile
4	Block Datas Length	Block Headers + Block Datas of this Tile
4	# of Blocks	
12	Zeros	Unused

Direction : As far as I'm concerned, it's useless. I prefer to check the **Orientation** instead of the Direction. Direction of a tile is a sort of a "main orientation", while the Orientation data itself is very more accurate. The values I have found : 1, 2, 3, 4 and 5. Maybe it has something to do while playing the game in Direct 3D mode, using the Perspective effect ? In this mode, the Tiles are slightly oriented to the left or to the right, and it have nothing to do with the fact they are drawing on the left half or the right half of the screen.

Roof height : A Roof tile is almost the same thing as a Floor tile, except that they have different Orientations, and that a Roof have this data not equal to zero. This height is used when the game need to draw the Roof : it tells in how many pixels to the up the sprite must be draw above the floor.

Sound index : This is a number that is used when the player is walking / running over a tile. There are differents sound : Wood, Stone, Mud, Hearth... I don't know the relation between this index and the wav files, as I have never check.

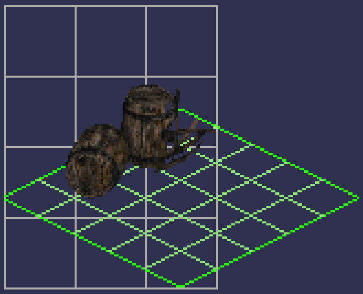
Animated ? : This is a flag. I sometimes wonder if this is not a bitfield, with more than 1 bit of this byte having a meaning. In any case, when a Floor tile is a part of a Floor animation, the lowest bit is set. In this case, the **Rarity / Frame index** data of this tile is not used as a Rarity, but as a Frame index. As far as I know there are only Floor that are animated, but I have never try to make an animated wall. Check the lava tiles in act 4 : for 1 animated floor, there are 10 tiles of the same **Orientation / Main Index / Sub-index** in the DT1, but with this Frame index that go from 0 to 9. The speed of the animation is hardcoded to be of 10 frames per sec, so each frame have a duration of 1/10 of a second, like in this example :



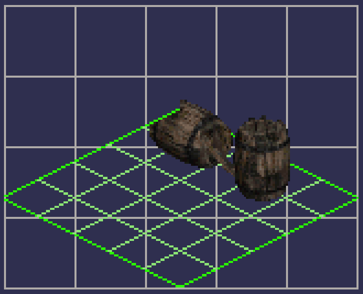
Height and **Width** : they are the size of the "box" where the final Tile fit whitin. They are always in power of 32 pixels (32, 64, 96, 128, 160 ...), and as a very special case, the Height is ALWAYS a negative number, whichever the type of the Tile, as if the game was starting to draw the pixels from the bottom of the image, instead of the top. Now remember that the size of this box have nothing to do with the coordinates system within. The Width and Height are just needed to create a bitmap in memory with enough space (and usually you'll have more space that you really need). The decoding process within this box will be discuss later.

About the Width of a Tile... It can be less than 160, but never more. If it's less, that's because the Tile have only the left part of the "box" that have pixels, so it's not necessary to have a bigger box if the gfx can fit into a smaller one. But in the case there's only gfx in the right part, the box will still be at its maximum size, since this Width is always referenced from the full left side. 2 examples will help :

Smaller box, because only the left part is used



Normal box, because the Width is from the left side



Beware : some Tiles have both their Width and Height set to 0, meaning theses Tiles are completly empty. You must check these 2 datas for the case they are equal to zero, because it's easy to have a bug in a program with that (trust me, I have try for you). **Data\Global\Tiles\Act1\Town\Trees.dt1** is such a DT1 with some empty Tiles.

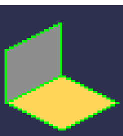
Orientation, **Main Index** and **Sub-index** : As said before, they are the 3 indexes that identify a Tile. There can be more than 1 tile with this combination in a DT1. In fact there are some Tiles that **are** using the same 3 indexes of another Tile, either a Tile of the same DT1, or the Tile of another. In this case, the game choose 1 Tile randomly whithin the same, according to the Rarity data (this very important point will be discuss later, as there are some very special cases).

Note that despite the Main-index and the Sub-Index datas in the .dt1 appear to be DWORDS (32 bits), in fact in a .ds1 (a map of the game) it can only use numbers of 6 bits for these 2 datas. This means that in all the .dt1 of the game you'll only find values that range from 0 to 63, no more. It's also very important to remember if you're planing to make your own .dt1 later. So, you have 64 possible numbers for the Main-index, and 64 possible numbers for the Sub-index, that means that for a map you can "only" use 64 * 64 different tiles (4096), but that should be enough ;)

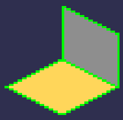
The **Orientation** is the data to check to know the type of the Tile :

- Floors, either static or animated, have an orientation of 0
- Special Tiles have 10 or 11 (special tiles are Warps, TP location, Map entries...)
- Shadows have 13
- "Walls" (better say "Objects") that have the precedent Shadows have an Orientation of 14
- Roofs have 15
- Lower walls have > 15
- Normal walls have all the other values which are < 15, as descibed here :

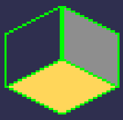
1. Left Wall



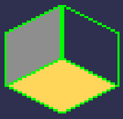
2. Upper Wall



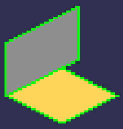
3. Upper part of an Upper-Left corner



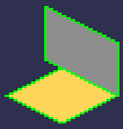
4. Left part of an Upper-Left corner



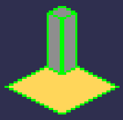
5. Upper-Right corner



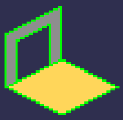
6. Lower-Left corner



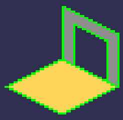
7. Lower-Right corner



8. ??? Seems to be Left Wall with Door object, but not always

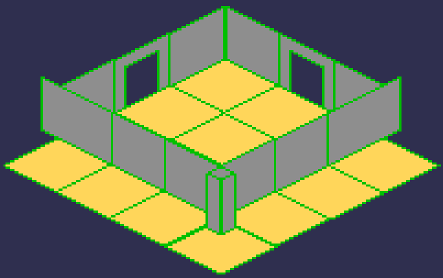


9. ??? Seems to be Upper Wall with Door object, but not always



Here's an exemple that show how the game usually use that Upper Wall Tiles. There is exeptions sometimes, but I think that this exemple is the generic layout. They're put together to make a 3 * 3 tiles room (but due to the design of the tiles in the game, we need 4 * 4 tiles) :

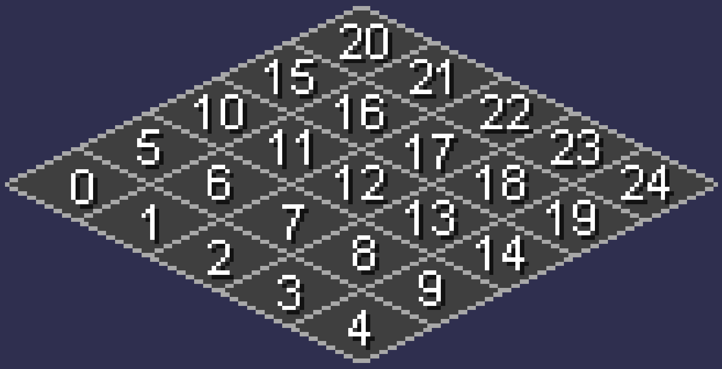
```
3925
8001
1001
6227
```



Rarity / Frame Index : As said before, this data is either the Rarity of a Tile, or the Frame Index in the case of an Animated Floor.

Unknown 1 to **Unknown 4** : I still don't know what theses datas ar for. It appear they have the same values for all the Tiles of a DT1, which is strange : why repeating the same values again & again for each Tiles if they're all the same ? Maybe they are some index that tells which sprite to used for the minimap, or maybe they tells which are the environment effects the game can make with the Tiles (like Rain), but I doubt they are any one of this 2 possibilities after all. They're really unknown datas for now.

Sub-tiles flags : They are the flags of the 25 Sub-tiles of a Tile. Let's say the North of the Tile is the upper-right border, the order of the flags is from bottom to up and left to right, as follow :



As far as I know, here's the bits of such a Sub-tile flag, and their meaning when I know them :

- bit 0 : block walk
- bit 1 : block light + block Line Of Sight (the possibility to see monsters)
- bit 2 : block jump (and teleport I believe)
- bit 3 : block Player's walk but not Mercenary's walk (weird)
- bit 4 : ?
- bit 5 : block light only (not LOS)
- bit 6 : ?
- bit 7 : ?

Block Headers Pointer : Pointer in file to the Block Headers of this Tile. There is a variable amount of Blocks for a Tile. If the Width and Height of the Tile are both equal to zero (no Blocks), this pointer is unsignificant (in originals DT1 it point to the Block Headers of the next valid Tile).

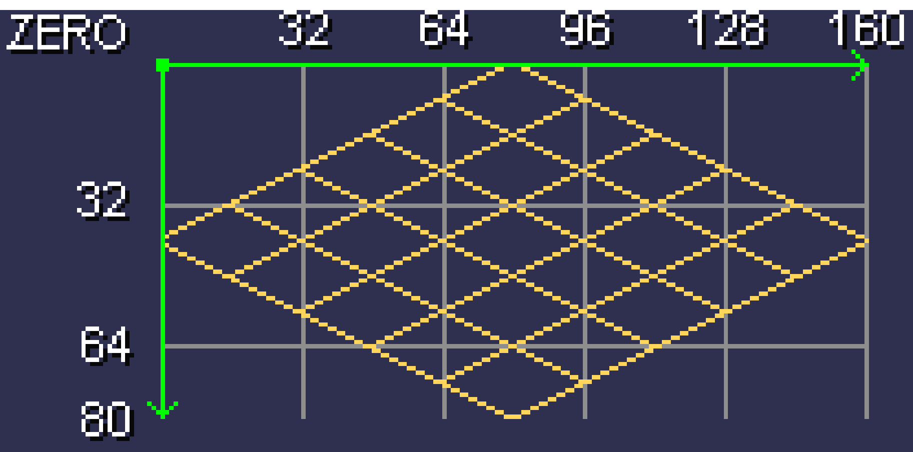
Block Datas Length : Length of the Block Headers + Block datas of this Tile (in Bytes). If the Width and Height of the Tile are both equal to zero (no Blocks), this length is set to zero.

# of Blocks : Number of Blocks which composed this Tile. This is of course both the number of Block Headers and the number of Block Datas. If the Width and Height of the Tile are both equal to zero, this number is set to zero.

Coordinates systems

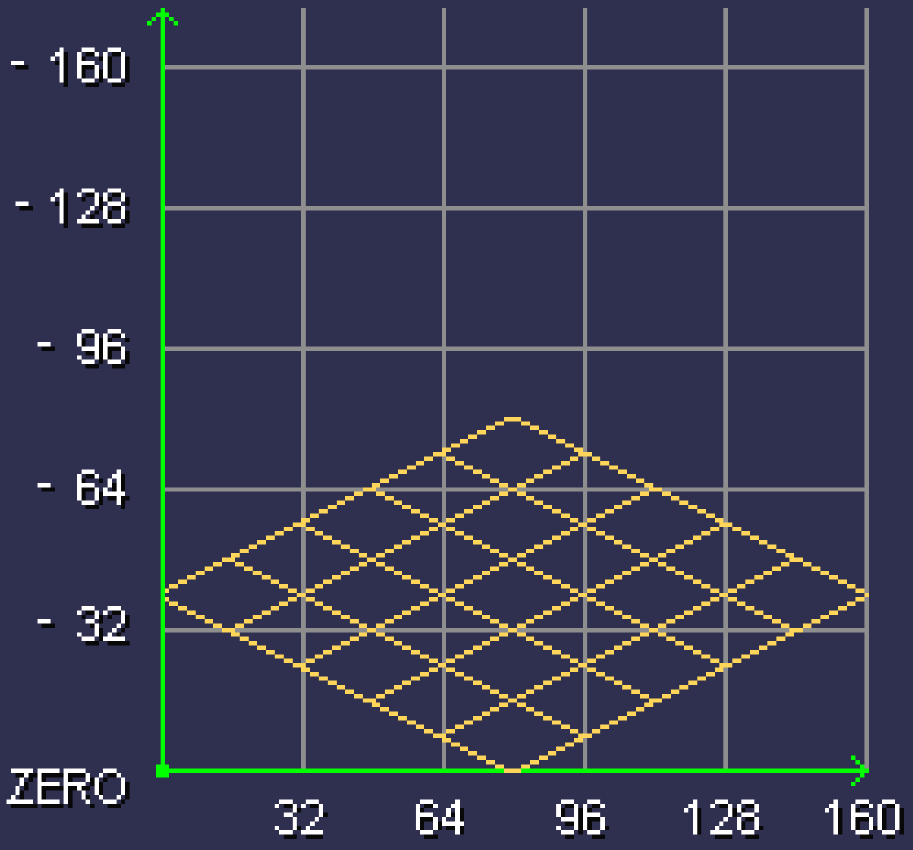
Before we look at the process of making a Tile with its Blocks, you must know the different coordinates system of each type of Tile. There are 3 differents systems, one for the Floors and Roofs, another one for the Upper Walls, the Shadows and the Specials Tiles, and the last for the Lower Walls. Each one of this system have the X axis working the same way, but NOT the Y axis. In summary, each blocks have its own coordinates to tells where to place it in the bitmap. So you must know where are theses coordinates in the bitmap, because it's dependant of the type of the Tile.

- **Floors and Roofs system** :



There's a unused area below the Floor. Despite the Height of a Floor says it's 128 pixels (-128 in the file, but +128 for us) you can safely assume the Floor is a box of 160 * 80 pixels. It's 79 to be very accurate, but 80 being a muliple of 2, it's better to keep 80 than 79 (come on, that's just 1 empty line).

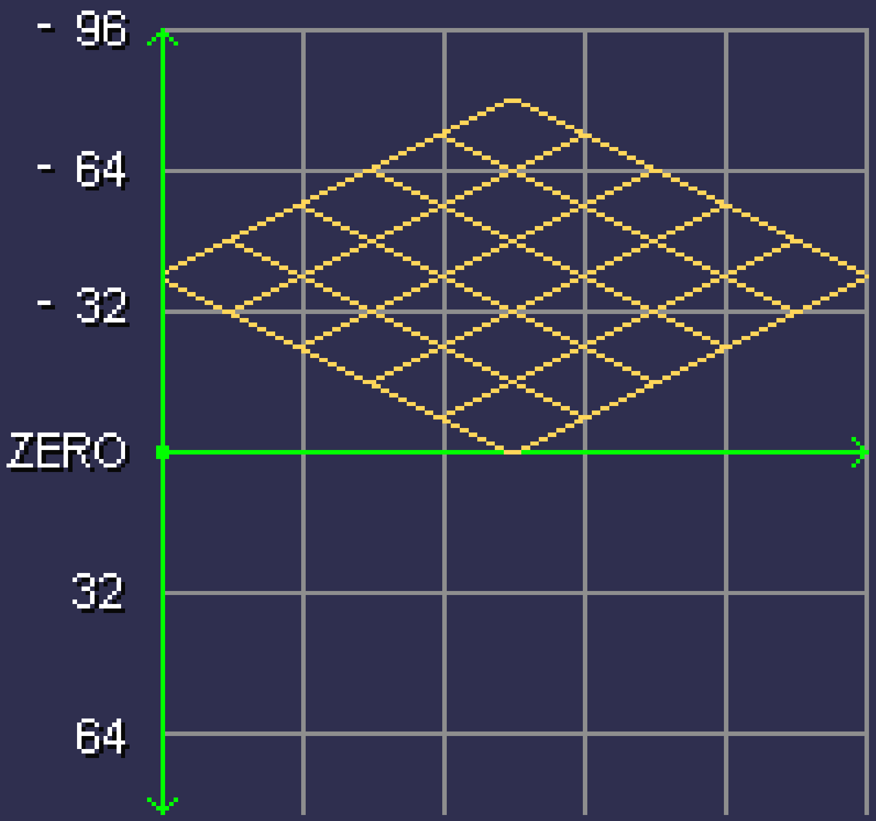
- **Upper Walls, Shadows and Special Tiles system** :



In this system, we know what the maximum coordinates are (zero), but not the minimum. As an exemple, the Worldstone have a Upper Wall Tile with an Height of 704 pixels. The floor grid which is draw is just for reference : Upper Walls usually don't have floor gfx.

Strangely, ALL Upper Wall Tiles I have checked so far have a line of blank blocks at the top. So you can assume that if the Tile is not an empty Tile, the Height is 32 pixels too much, and therefore you can minus the Height by 32 when creating the bitmap in memory.

- **Lower Walls system** :



And in this last system, we know what the minimum coordinates are (- 96), but not the maximum. As an exemple, the Worldstone have a Lower Wall Tile with an Height of 960 pixels. The floor grid which is draw is usually NOT just for reference : Lower Walls usually have a floor gfx in there, with their normal lower walls.

Also, be carefull of 2 tricks :

- In this system, we don't necessary have the Block's Height data beeing the negative value of the maximum coordinate. In fact, in this case that Height value is the minimum number of pixel (in power of 32) necessary to draw the Tile. For instance, imagine that we have a Lower Wall that only use 2 blocks, at coordinates (0, 64) and (0, 96). In this case the Block's Height data will be set to -64 (only 2 blocks needed).

- The 2nd trick is that, have you have just see, in this system there is no useless line of blocks, the Block's Height data is the **exact** amount of blocks needed to make the bitmap in memory, so don't minus that value by 32 like you did in the Upper Walls system, and be careful when you want to draw that blocks in your bitmap.

After all the Tile Headers come the Tile Datas. Each one of this Tile Data is structure like that :

- a variable amount of Block Headers
- the same amount of Block Datas

A Block Header contain all the informations of a Block, while the Block Data is the encoded pixels of this Block. Each one of the Block Header is structured as follow :

Block Header (20 bytes)

# Bytes	Description	Comment
2	X position	Position in the bitmap
2	Y position	
2	Zeros	Unused
1	Grid X	Position in the Sub-tile Grid
1	Grid Y	
2	Format	Type of encoding
4	Length	Length in bytes of the encoding data
2	Zeros	Unused
4	File offset	Offset in file of the encoding data

X position and **Y position** : They are the coordinates of the upper / left corner of the Block, according to the Coordinates System of the Tile, as explain before.

Grid X and **Grid Y** : They range from 0 to 4 both, and they're really used ONLY for a Floor Tile. I think they're some kind of relations between the Sub-tile Flag and the Block position, but since they're ALL the same for all the Floor Tiles I have checked, you can forget them if you want (but keep that in mind).

Format : If this value is equal to 1, then it's a 3D-isometric Floor Block (RAW format, no transparency), else a regular one (RLE fomat, 32 by 32 pixels).

Length : Length in bytes of the encoding data of this Block.

File offset : Offset to **add** to the **Block Headers Pointer** of the Tile to have the position in the file of the encoding data of this Block.

After all theses Block Headers come the Block Datas. Each one of this encoding data must be decode in accordance of the **Format** code of the Block. Remember there is NO PALETTE in a DT1, since there is a global palette for each act. Here are 2 (lame) samples of code in C language that show how to decode the 2 possible types of Blocks :

3D-isometric Block :

1st line : draw a line of 4 pixels
2nd line : draw a line of 8 pixels
3rd line : draw a line of 12 pixels
and so on...

```
void draw_block_isometric (BITMAP * dst, int x0, int y0, const UBYTE * data, int length)
{
   UBYTE * ptr = data;
   int   x, y=0, n,
         xjump[15] = {14, 12, 10, 8, 6, 4, 2, 0, 2, 4, 6, 8, 10, 12, 14},
         nbpix[15] = {4, 8, 12, 16, 20, 24, 28, 32, 28, 24, 20, 16, 12, 8, 4};


   // 3d-isometric subtile is 256 bytes, no more, no less
   if (length != 256)
      return;
   // draw
   while (length > 0)
   {
      x = xjump[y];
      n = nbpix[y];
      length -= n;
      while (n)
      {
         putpixel(dst, x0+x, y0+y, * ptr);
         ptr++;
         x++;
         n--;
      }
      y++;
   }
}
```

RLE Block :

1st byte is pixels to "jump", 2nd is number of "solid" pixels, followed by the pixel color indexes.
when 1st and 2nd bytes are 0 and 0, next line.

```
void draw_block_normal (BITMAP * dst, int x0, int y0, const UBYTE * data, int length)
{
   UBYTE * ptr = data, b1, b2;
   int   x=0, y=0;


   // draw
   while (length > 0)
   {
      b1 = * ptr;
      b2 = * (ptr + 1);
      ptr += 2;
      length -= 2;
      if (b1 || b2)
      {
         x += b1;
         length -= b2;
         while (b2)
         {
            putpixel(dst, x0+x, y0+y, * ptr);
            ptr++;
            x++;
            b2--;
         }
      }
      else
      {
         x = 0;
         y++;
      }
   }
}
```

Rarity (frequence) of Tiles

A harder topic now. I'll assume you have already used my DS1 Editor (a DS1 is a map of the game). If it is not the case, you should leave this topic for now, and come back later.

So, what identify a Tile is 3 indexes : Orientation, Main-index and Sub-index. And you should already know that when some Tiles have the same 3 indexes, they are a part of the same **random set of Tiles**. Now, how the game handle this ?

For instance, in Tristram, there are 4 Floor Tiles which are looking as hearth half burnt, and they have the EXACT same Orientation / Main-index / Sub-index . This is why they are a part of a random set. When the game see in a map that it must use such a Tile it choose one randomly between the random set... but not equally. Each Tile have its own chance to appear. This certainly remind you of the "Rarity" you can see in some .txt. This work almost the same way, but with some particularity.

Let's take an easy exemple. Say that you have remake the town of act1 with only 1 Floor Tile, and no Walls (except the 3 Special ones of course). Now, let's make a custom DT1 : 2 Floor Tiles in it, having the same Orientation / Main-index / Sub-index as the one used in the map, and with their **Rarity** each set to 1. So, we have 2 Floors, with the same Rarity, from the same random set. If you test this map, you'll see the 2 Floors, and they'll share equally the area.

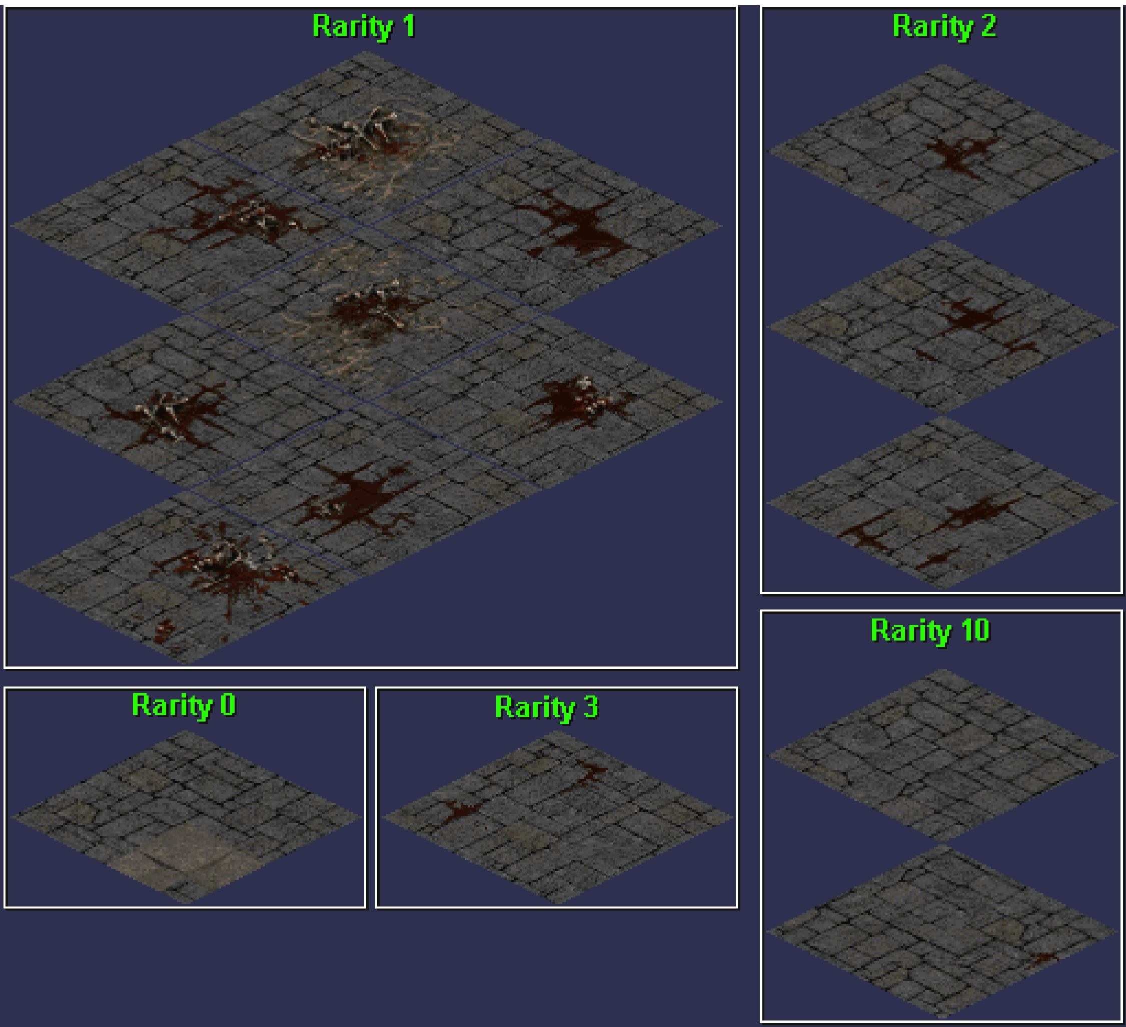
Now, set the first Floor Rarity to 1 and the second to 10. After remaking the DT1 and testing with the same map... the 1st Floor don't appear often. It appears 1/11 of the time : the Rarity of 1 Tile is for the distribution. If you set the first Floor Rarity to 2 and the second to 4, you'll have the first Tile appear 2 / 6 of the times, and the second 4 / 6 of the times.

Now, some special cases. What happens if in a Tile Random Set some tiles have a Rarity of 0 ? All to 0 ? Same random set in 2 different DT1 ? Some 0 in 1 DT1 and the other ?

Let's say we have 2 DT1, whith Tiles of the same random set in both. **First, the game compute the sum of the Rarity of all the Tiles in all the DT1 of the map for this random set**. If this sum is equal to zero, then you'll only see 1 Tile in the map, always the same : the Tile which is the last of this random set in the 1st DT1. It means that if all the Rarity are zero, you'll never see a Tile of another DT1 than in the 1st DT1, and for the 1st DT1 you'll only see the last Tile it found.

If the sum is NOT zero, then you'll see random Tiles of both DT1. But the game only use the Tiles which DO have a Rarity in this case. There's no priority of DT1 here, each Tile will appear according to its Rarity, as long as it is not zero (it's just normal to not take a Tile if it have 0 luck to appear after all).

Maybe an exemple will help to fix how it works :



This image show all the Tiles of 1 random set, used by **facade.ds1**, the Monastery. All these Floors are part of the same random set because they all have the same Orientation / main index / Sub-index. But their Rarity varies. Total of Rarity for this set = (8 x 1) + (1 x 0) + (3 x 2) + (1 x 3) + (2 x 10) = **37**. So for each 37 Tiles that will be draw, you'll see 8 Tiles that have many blood over them, 6 Tiles that have less blood, 3 Tiles that have very very few blood over them, and 20 Tiles that have no blood at all. You'll never see the Tile in the bottom-left corner of the image, because it have a Rarity of 0. It surely won't be this exact amount of Tiles, because of some randomness, but you've got the idea.

Last words

This doc is not yet complete, as there are still some unknown datas in the file format, but it should already be of a great help. If you find more informations, or if you have suggestions (or even if you just find typo errors ;)) you can contact me at siramy_paul@yahoo.com