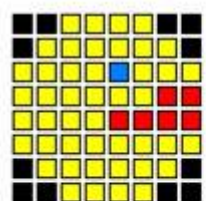


Tile background

A background that is made up from tiles consists of small 8x8 characters (tiles). These tiles are defined by the user the same way as an 8x8 sprite in a 1D fashion, and can either get its colors from a palette of 16 or 256 colors. These tiles can be placed on the screen wherever you want them, and the same tile can be used as many times you wish. By putting different tile numbers in a tile map, you can make your own backgrounds from tiles.

You probably remember this illustration from the lesson about sprites; this could just as well have been a tile. The format of data for tiles and 8x8 sprites are exactly the same.



This is what the tile will look like in its true size 🍷

Color palette

0	■
1	■
2	■
3	■

These are the values we need to store in the Char Base Block to make the tile look like the pacman at the left.

0	0	3	3	3	3	0	0
0	3	3	3	3	3	3	0
3	3	3	3	2	3	3	3
3	3	3	3	3	3	1	1
3	3	3	3	1	1	1	1
3	3	3	3	3	3	3	3
0	3	3	3	3	3	3	0
0	0	3	3	3	3	0	0

A very important thing to do when using tile backgrounds is to know where in VRAM you wish to put the different tile data that describe how the tiles look. This is called Char Base Block. Another necessary thing to know is where you want to put the info about where the different tiles are to be placed on the screen. This is called map data and is put in a Screen Base Block.

Here is a table that shows how the Char Base Blocks and Screen Base Blocks are spread out in memory. We just have 4 available Char Base Blocks (0-3) and 32 available Screen Base Blocks (0-31). It is important that these memory areas don't overlap or your background will display incorrectly.

VRAM	Address	Screen Base Block
------	---------	-------------------

Char Base Block 3 (16Kb)	0x600F800 (2Kb)	31
	0x600F000 (2Kb)	30
	0x600E800 (2Kb)	29
	0x600E000 (2Kb)	28
	0x600D800 (2Kb)	27
	0x600D000 (2Kb)	26
	0x600C800 (2Kb)	25
	0x600C000 (2Kb)	24
Char Base Block 2 (16Kb)	0x600B800 (2Kb)	23
	0x600B000 (2Kb)	22
	0x600A800 (2Kb)	21
	0x600A000 (2Kb)	20
	0x6009800 (2Kb)	19
	0x6009000 (2Kb)	18
	0x6008800 (2Kb)	17
	0x6008000 (2Kb)	16
Char Base Block 1 (16Kb)	0x6007800 (2Kb)	15
	0x6007000 (2Kb)	14
	0x6006800 (2Kb)	13
	0x6006000 (2Kb)	12
	0x6005800 (2Kb)	11
	0x6005000 (2Kb)	10
	0x6004800 (2Kb)	9
	0x6004000 (2Kb)	8
Char Base Block 0 (16Kb)	0x6003800 (2Kb)	7
	0x6003000 (2Kb)	6
	0x6002800 (2Kb)	5
	0x6002000 (2Kb)	4
	0x6001800 (2Kb)	3
	0x6001000 (2Kb)	2
	0x6000800 (2Kb)	1
	0x6000000 (2Kb)	0

We have two kinds of backgrounds on the gba: text backgrounds and rotation/scaling backgrounds. This table shows which sizes the backgrounds can have when we use both of these backgrounds.

In this table you can also see how many (kilo)bytes each of these backgrounds need. This might be nice to know when you want to figure out how many Screen Base Blocks you need for storing map data. What is important to remember is that Screen Base Blocks and Char Base Blocks cannot use the same memory area, or else you will get garbage on the screen.

Size Bits 14-15	Text background		Rotation/scaling background	
	Number of pixels	Size	Number of pixels	Size
00	256x256	2K bytes	128x128	256 Bytes
01	512x256	4K bytes	256x256	1K byte
10	256x512	4K bytes	512x512	4K bytes
11	512x512	8K bytes	1024x1024	16K bytes

The first thing to do when making a background is to choose if we want to use a text background or a rotation/scaling background. From the table underneath we can find information about which modes support the different backgrounds.

Mode	BG0CNT	BG1CNT	BG2CNT	BG3CNT
0	Background 0 Text	Background 1 Text	Background 2 Text	Background 3 Text
1	Background 0 Text	Background 1 Text	Background 2 Rotation/scaling	

2			Background 2 Rotation/scaling	Background 3 Rotation/scaling
---	--	--	----------------------------------	----------------------------------

Next comes an overview of the different background control registers, and an explanation of the different bits in the registers.

It's in these registers we decide where in VRAM we want to store the char data that describes the look of the tiles (Char Base Block), and where we want to put the information about where on the screen the different tiles are supposed to be placed, also known as map data (Screen Base Block).

Text background control register. (supported by BG0, BG1, BG2 and BG3).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE	XX	SBB						C	M	XX	XX	CBB	P		

Bits 0-1 Priority of the background
 00: 1. priority
 01: 2. priority
 10: 3. priority
 11: 4. priority

Bits 2-3 Character Base Block (0-3), choose where in memory we are going to put char data for tiles (4 different intervals of 16K bytes each).

Bits 4-5 Not used.

Bit 6 Set mosaic.

Bit 7 Choose between 16/256 color palette
 0 = 16 colors x 16 palettes
 1 = 256 colors x 1 palette

Bits 8-12 Screen Base Block (0-31), choose where in memory we are supposed to put the map data for tiles (intervals of 2K bytes).

Bit 13 Not used.

Bits 14-15 Choose which size the background is supposed to be
 00: 256x256 pixels
 01: 512x256 pixels
 10: 256x512 pixels
 11: 512x512 pixels

Text and rotation/scaling background control register (only supported by BG2 and BG3).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIZE	W	SBB						C	M	XX	XX	CBB	P		

Bits 0-1 Priority of the background
 00: 1. priority
 01: 2. priority
 10: 3. priority
 11: 4. priority

Bits 2-3 Character Base Block (0-3), choose where in memory we are going to put char data for tiles (4 different intervals of 16K bytes each).

Bits 4-5 Not used.

Bit 6 Set mosaic.

Bit 7 Choose between 16/256 color palette
 0 = 16 colors x 16 palettes
 1 = 256 colors x 1 palette

Bits 8-12 Screen Base Block (0-31), choose where in memory we are supposed to put the map data for tiles (intervals of 2K bytes).

Bit 13 Set wraparound.

Bits 14-15 Choose which size the background is supposed to be

- 00: 128x128 pixels
- 01: 256x256 pixels
- 10: 512x512 pixels
- 11: 1024x1024 pixels

What is important when using tile backgrounds is that you know where in VRAM you wish to put the data describing the tiles' appearance (Char Base Block), and where you wish to put the information about where you want the tiles to be placed on the screen (Screen Base Block).

When we have set the background control register for our chosen background, we have to input the data describing how the different tiles look in the char base block chosen in the background control register. When we have input the palette at 0x5000000 (BG_PaletteMem) and the image information for all tiles (char data), there is only one thing left to do: to tell the gba where on the screen we want to put the different tiles. This is done by storing information in the Screen Base Block we chose in the background control register. There are two different formats to store the map data.

If we use a text background, the information about every tile's position takes 16 bits, whereas when using a rotation/scaling background we only need 8 bits. Here is an explanation of the two formats:

Text background.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Palette				VF	HF	Tile number(0-1023)									

Bits 0-9 The tile number for the tile used (max 1024 different tiles). Remember that if you're using 256 color tiles each tile takes 2 spaces in memory, which means that tile 0 will start at 0, tile 1 starts at 2, tile 2 starts at 4...

Bit 10 Horizontal flip: the tile will be flipped on the y-axis.

Bit 11 Vertical flip, the tile will be flipped on the x-axis.

Bit 12-15 When using 16 palettes with 16 colors each, we can choose which palette to use for the specific tile. When using 1 palette with 256 colors these bits have no meaning.

Rotation/scaling background.

7	6	5	4	3	2	1	0
Tile number(0-255)							

Bits 0-7 These bits choose the tile number; there are only 255 available tiles in rotation/scaling backgrounds. These backgrounds always use a 256 color palette, with no options for flipping tiles. We always have to write 16 bits at the time to VRAM, and because of that we have to write two 8-bit tile numbers to memory at the same time.

Here is a program to show how to make a simple tile background.

```
#include "gba_h.h"
```

```
int main()
```

```
{
```

```
    u8 i, x, y;
```

```
    u16* ScreenBB = (u16*)ScreenBaseBlock(8);
```

```
    u16* CharBB = (u16*) CharBaseBlock(0);
```

```
    REG_BG0CNT = BG_COLOR_256 | TEXTBG_SIZE_256x256 | (8 << SCREEN_SHIFT) | (0 << CHAR_SHIFT);
```

```
    SetMode(MODE_0 | BG0_ENABLE);
```

```
// input 3 colors into the palette
```

```
    BG_PaletteMem[0] = RGB(31,31,31);//white
```

```
    BG_PaletteMem[1] = RGB(0,30,0);//green
```

```
    BG_PaletteMem[2] = RGB(0,0,25);//blue
```

```
// input tiles(char data)
```

```
    for(i=0; i<32; i++) //tile 0 (background)
```

```
    {
```

```
        CharBB[i] = 0x0000; // color from palette 0(!! We have to write 2 byte at a time !!)
```

```
    }
```

```
    for(i=32; i<64; i++) //tile 1
```

```
    {
```

```
        CharBB[i] = 0x0101; // color from palette 1(!! We have to write 2 byte at a time !!)
```

```
    }
```

```
    for(i=64; i<96; i++)//tile 2
```

```
    {
```

```
        CharBB[i] = 0x0202; // color from palette 2(!! We have to write 2 byte at a time !!)
```

```
    }
```

```
// 2 tiles are placed on the screen
```

```
    x = 5;
```

```
    y = 3;
```

```
    ScreenBB[x+(32*y) ] = 1;//tile 1 at location (5, 3)
```

```
    x = 15;
```

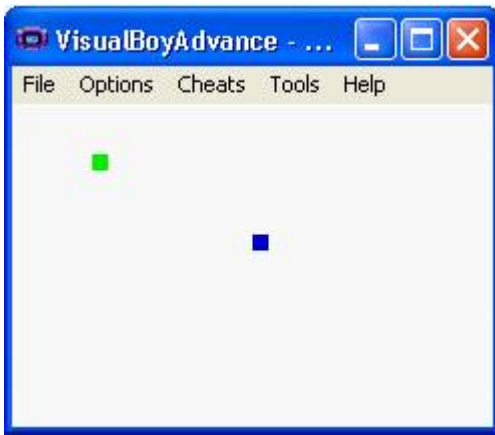
```
    y = 8;
```

```
    ScreenBB[x+(32*y) ] = 2; //tile 2 at location (15, 8)
```

```
    while(1) {}
```

```
}
```

The result from this program should look something like this.



There is a lot of new code in this program, and probably the first thing to grab your attention will be these two lines:

```
u16* ScreenBB = (u16*)ScreenBaseBlock(8);
u16* CharBB = (u16*) CharBaseBlock(0);
```

These lines use the macros CharBaseBlock(n) and ScreenBaseBlock(n) from gba_bg.h

```
#define CharBaseBlock(n)      (((n)*0x4000)+0x6000000)
#define ScreenBaseBlock(n)   (((n)*0x800)+0x6000000)
```

These macros figure out the correct memory address according to which ScreenBaseBlock and CharBaseBlock we choose. These addresses are then cast to the pointers ScreenBB and CharBB that we use when we store char and map data.

Another new thing is probably the way we set the background control register. We choose to use background 0, with 256 colors (1 palette), and we want it to be a text background with size 256x256 pixels (32x32 tiles). The next thing we do is set the value 8 (which is the ScreenBaseBlock we choose to use) to position 8 (which is the number replaced by the macro SCREEN_SHIFT) in the control register (the number 8 is also moved 8 places to the left in the control register). The last thing we do is set the value 0 (which is the CharBaseBlock we choose to use) to position 2 in REG_BG0CNT.

This last thing “|(0 << CHAR_SHIFT)” is not actually necessary because it already says 0 as the constant. It’s just here to show how it’s done in case you want to change the CharBaseBlock.

```
REG_BG0CNT = BG_COLOR_256 | TEXTBG_SIZE_256x256 | (8 << SCREEN_SHIFT) | (0 <<
CHAR_SHIFT);
```

In these lines we used 2 macros from gba_bg.h:

```
#define CHAR_SHIFT 2
#define SCREEN_SHIFT 8
```

These macros decide how many places we move the ScreenBaseBlock and CharBaseBlock values to the left in the background control register (BG0CNT).

The next line we have seen before:

```
SetMode(MODE_0 | BG0_ENABLE);
```

In this example we set mode 0, and background 0.

We only enter one color for each of the first 3 numbers in the palette (0, 1 and 2).

```
// input 3 colors into the palette
```

```

BG_PaletteMem[0] = RGB(31,31,31);//white
BG_PaletteMem[1] = RGB(0,30,0);//green
BG_PaletteMem[2] = RGB(0,0,25);//blue

```

We used several loops to store char data in memory; this way we don't have to make extra files with char data for the tiles. In this example I have only made 'boring' single color tiles by setting the same value for all 64 pixels in the tile ($8 \times 8 = 64$).

```
CharBB[i] = 0x0000;
```

2 palette values (each 8 bits) are stored into memory at a time (the gba can only store 16 bits at a time remember? This is done 32 times; that make a total of 64 pixels with the color from palette 0 (white). The same is done in the other loops as well: tile 0 is colored white, tile 1 is green, and tile 2 is blue.

```
CharBB[i] = 0x0101;
```

0x0101 is a 16 bits value, when we use hexadecimal numbers we can divide the numbers into two parts, this way we can get the two 8 bit numbers 01 and 01. These numbers are 8 bit (half of the original number); each of these points to palette entry 1. It has to be done this way because we can only write 16 bits at a time to VRAM. If we had written `CharBB[i] = 0x01;` only half of the tile pixels would be green. The other pixels would be white because the other part of the 16 bit number would be understood as 0 which is palette 0 and the pixel color white. The machine would actually see the number as `CharBB[i] = 0x0001`.

```

// input tiles(char data)
for(i=0; i<32; i++) //tile 0 (background)
{
    CharBB[i] = 0x0000; // color from palette 0 (!! We have to write 2 byte at a time !!)
}
for(i=32; i<64; i++) //tile 1
{
    CharBB[i] = 0x0101; // color from palette 1 (!! We have to write 2 byte at a time !!)
}
for(i=64; i<96; i++) //tile 2
{
    CharBB[i] = 0x0202; // color from palette 2 (!! We have to write 2 byte at a time !!)
}

```

I have not made a real map for this background, but instead just put some tiles directly on the screen. I hope you still remember the formula $x + (32 * y)$ from lesson 3 about bitmapped backgrounds. Tile number 0 was not set in this example, but as you probably did see when you run the program on an emulator the background turned white. The reason for this is that the emulator stores the value 0 in VRAM, and when the value 0 is everywhere in memory (including Screen Base Block 8) all entries in tile map will contain the value 0, which is the white tile. We are only writing two tiles to the tile map, so all other tiles will still have the color white (value 0).

```

// 2 tiles are placed on the screen
x = 5;
y = 3;
ScreenBB[x+(32*y)] = 1; //tile 1 at location (5, 3)

x = 15;
y = 8;
ScreenBB[x+(32*y)] = 2; //tile 2 at location (15, 8)

while(1) {}

```

Each of the locations in the tile map uses 16 bits (see below), and we write 16 bits to VRAM at a time so that the values we input (1 for tile 1 and 2 for tile 2) will be correct. We are not flipping the tiles or using 16 color palettes, so we won't have any complications with the settings.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Palette				VF	HF	Tile number (0-1023)									

If we were to use rotation/scaling backgrounds each of the entries in the map would be a 8 bit value., so we need to input two and two bits into memory at a time.

7	6	5	4	3	2	1	0
Tile number(0-255)							

There is one register called mosaic register that is used to manipulate the look of the tiles and sprites.

Mosaic register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sprite Mosaic V				Sprite Mosaic H				BG Mosaic V				BG Mosaic H			

If you set BG_MOSAIC_ENABLE in the initializaton of REG_BG0CNT, you are able to zoom in on the pixels both vertically (BG mosaic V) and horizontally (BG mosaic H) in your background. The value you write to BG mosaic V and BG mosaic H is proportional to the amount of zoom.