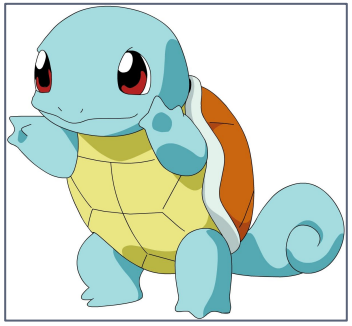
CS 2261: Media Device Architecture - Week 9

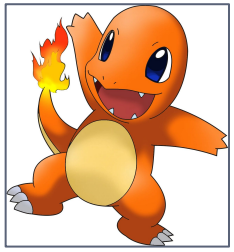
Overview

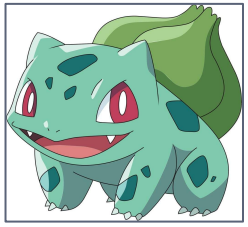
- Sprites
- Mode 0 Intro

Sprites

This is the relative ranking of Pokémon favorites for Quiz 2 (by size):

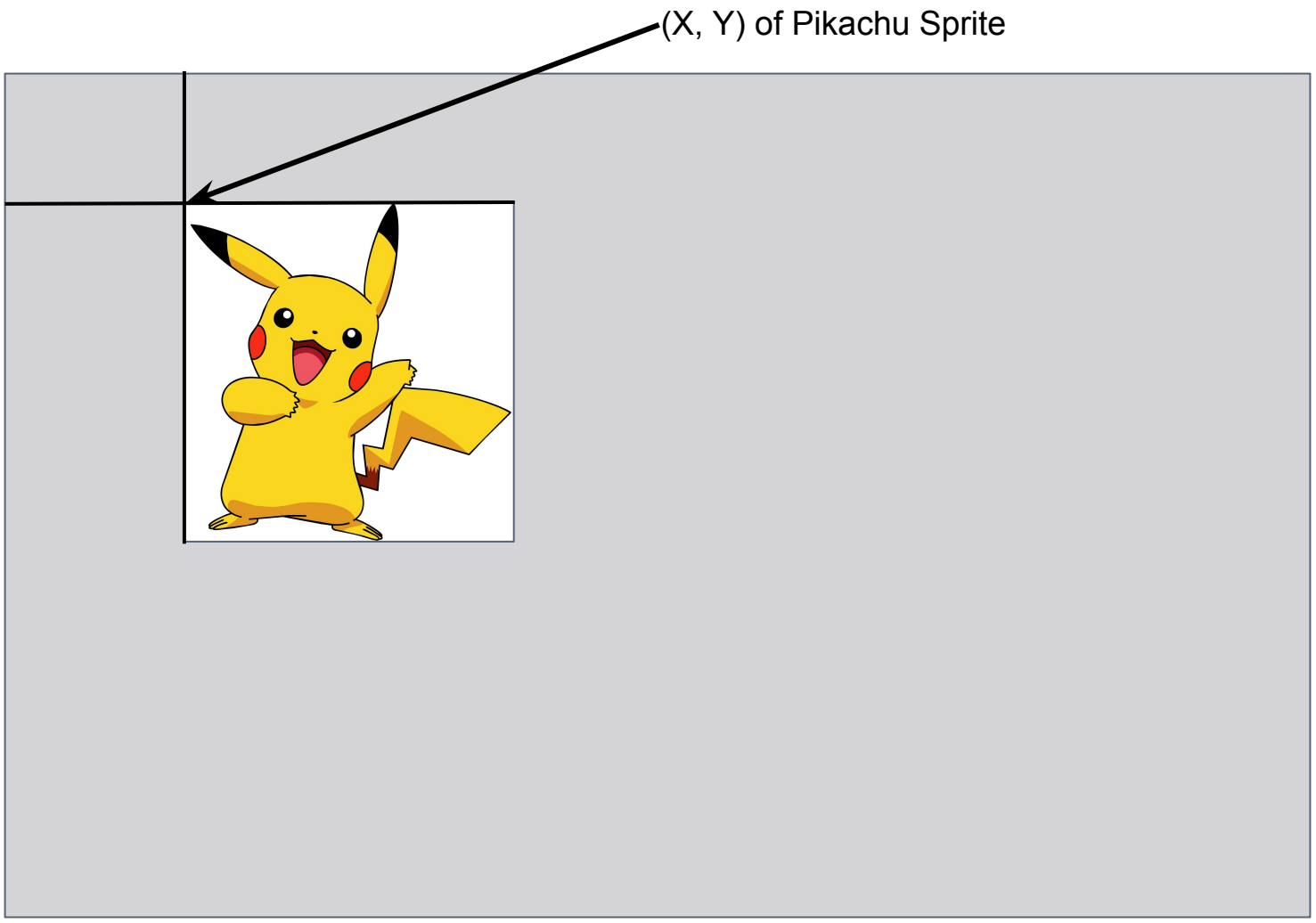






You voted for 29 different Pokémon, but all but 4 got a single vote.

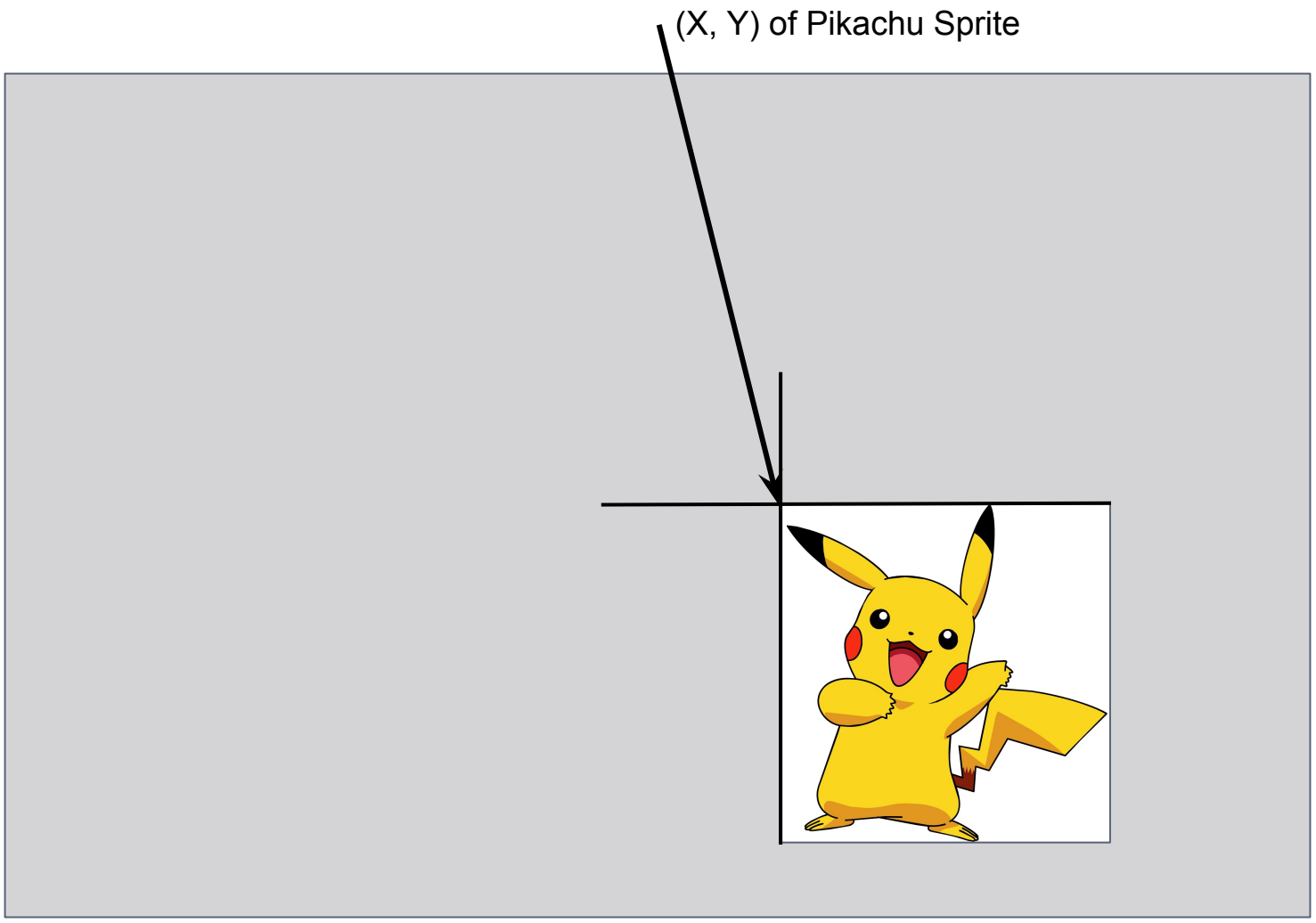
Pikachu Sprite



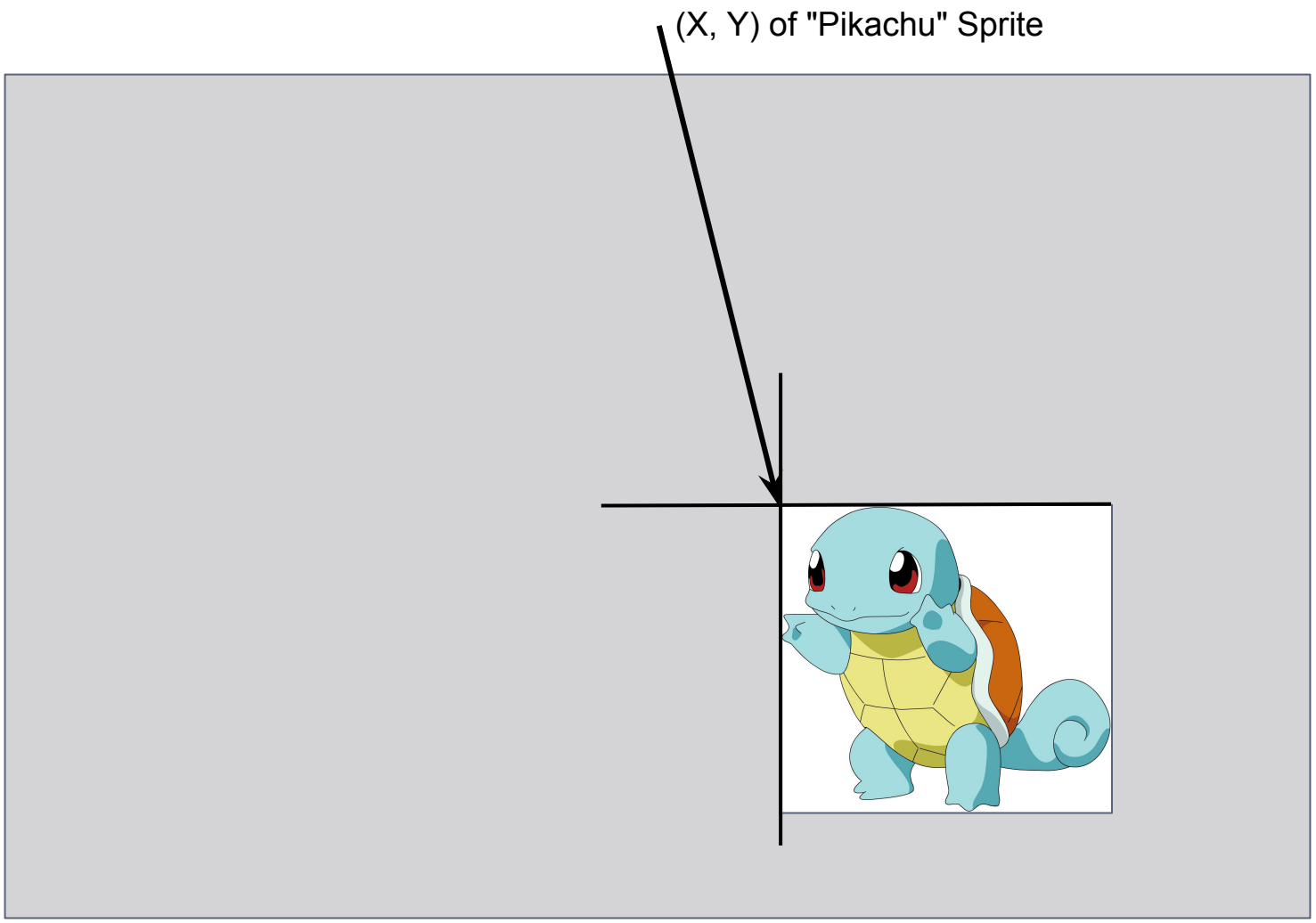
Sprites

- These days, "sprite" is a generic term for an image that can be placed into a larger scene.
  - Originally, sprites were images that could be inserted without disturbing the scene. In this sense, they are implemented with hardware.
    - GBA sprites fall under the original definition
      - They sit "on top of" the background behind them.
      - They can move independently of the background!
  - Examples of sprites:
    - Balls, paddles, pacmen, pikachus…
    - Basically anything that's not part of the background is likely best implemented as a sprite.

Moved within the scene



Replaced by Updating Underlying Memory



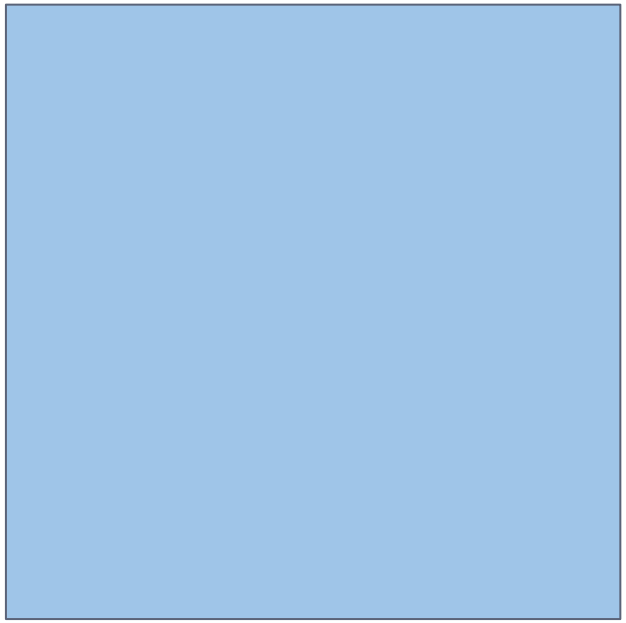
GBA Sprite Overview

- Standard component of games
  - Basically anything that's not part of the background should probably be a sprite.
- Composed of 8x8 tiles
- Hardware-supported movement and drawing (including transforms) that doesn't change the background.
- The GBA supports up to 128 sprites at a given time, ranging in size from 8x8 to 64x64 pixels.
- Sometimes referred to as "video objects", especially in GBA documentation (see some of the following slides).

Size comparisons

Single Pixel: Smallest Sprite: 8x8

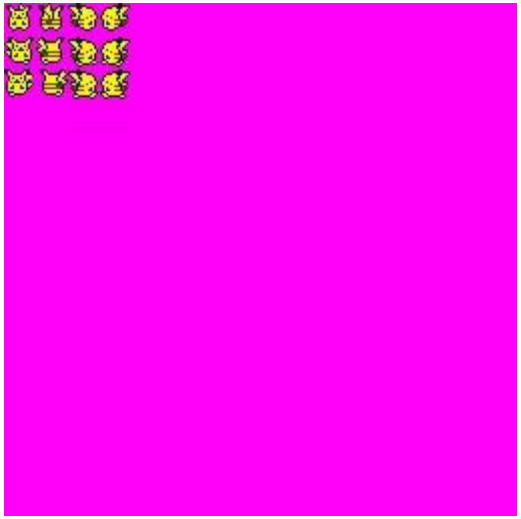
Largest Sprite: 64x64



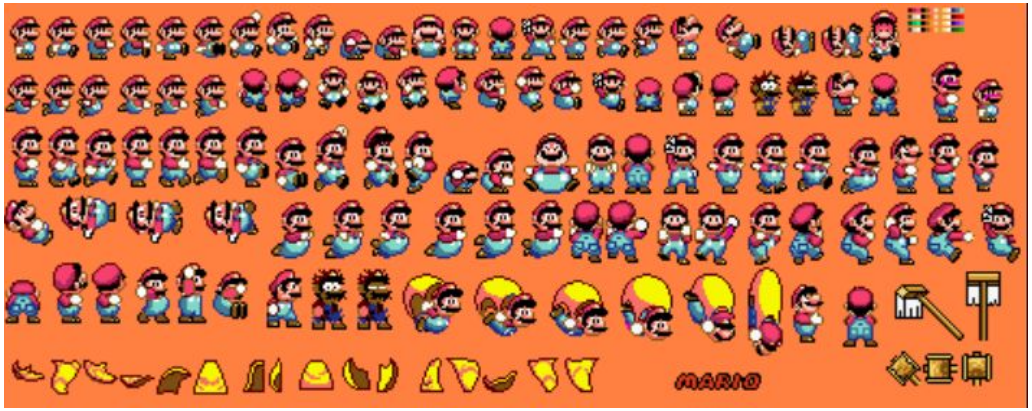
Screen Size: ~20% wider than this entire screen, at this scale.

Sprite sheet

- Often, a mapping of many sprites are combined into a single bitmap
  - Typically for space/performance reasons.
  - Can also help keep related sprite values together, logically (as with sprite animations).



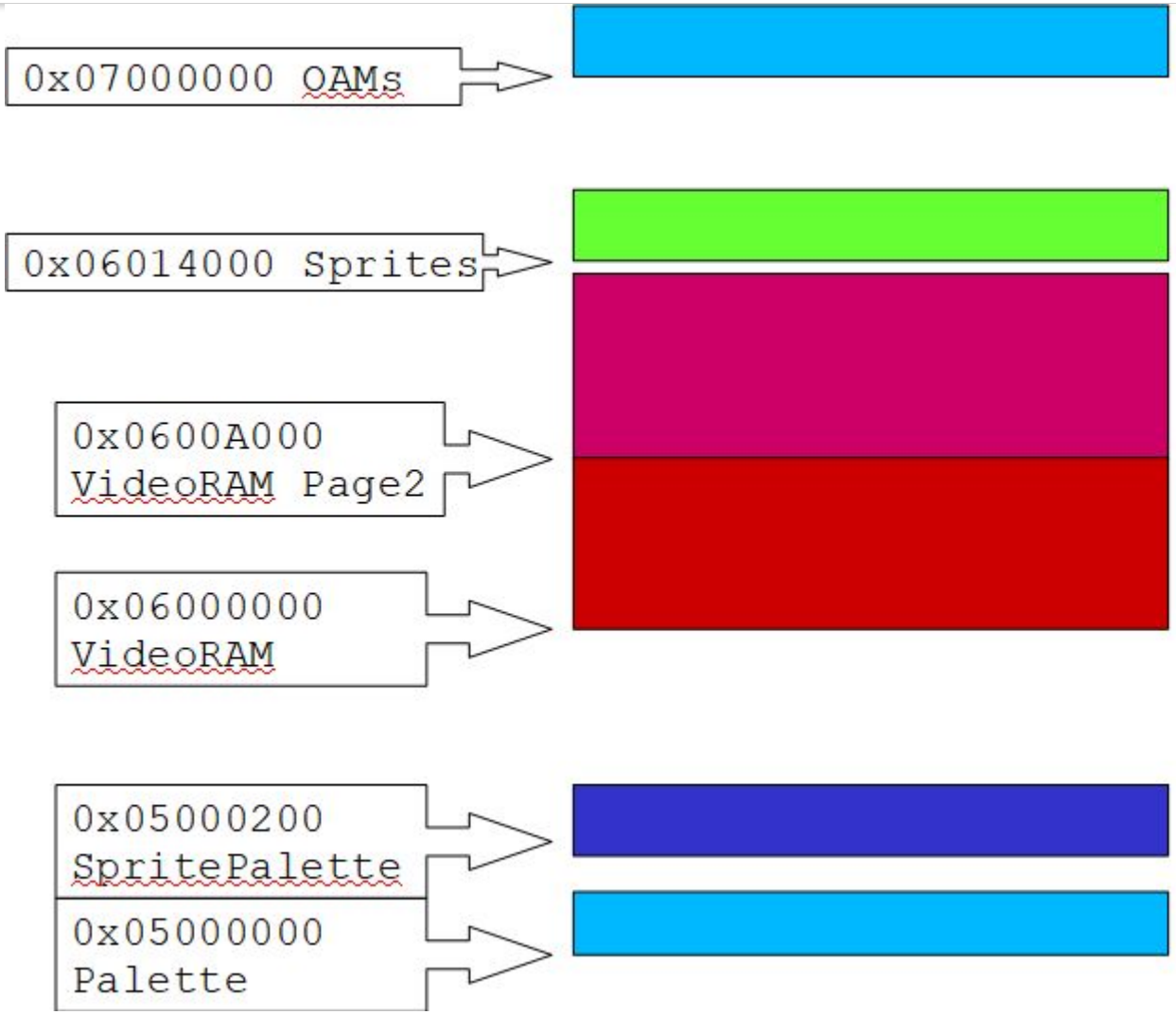
- Such a grouping is typically called a "sprite sheet" (also sometimes called a "texture atlas").



Essential Sprite Steps

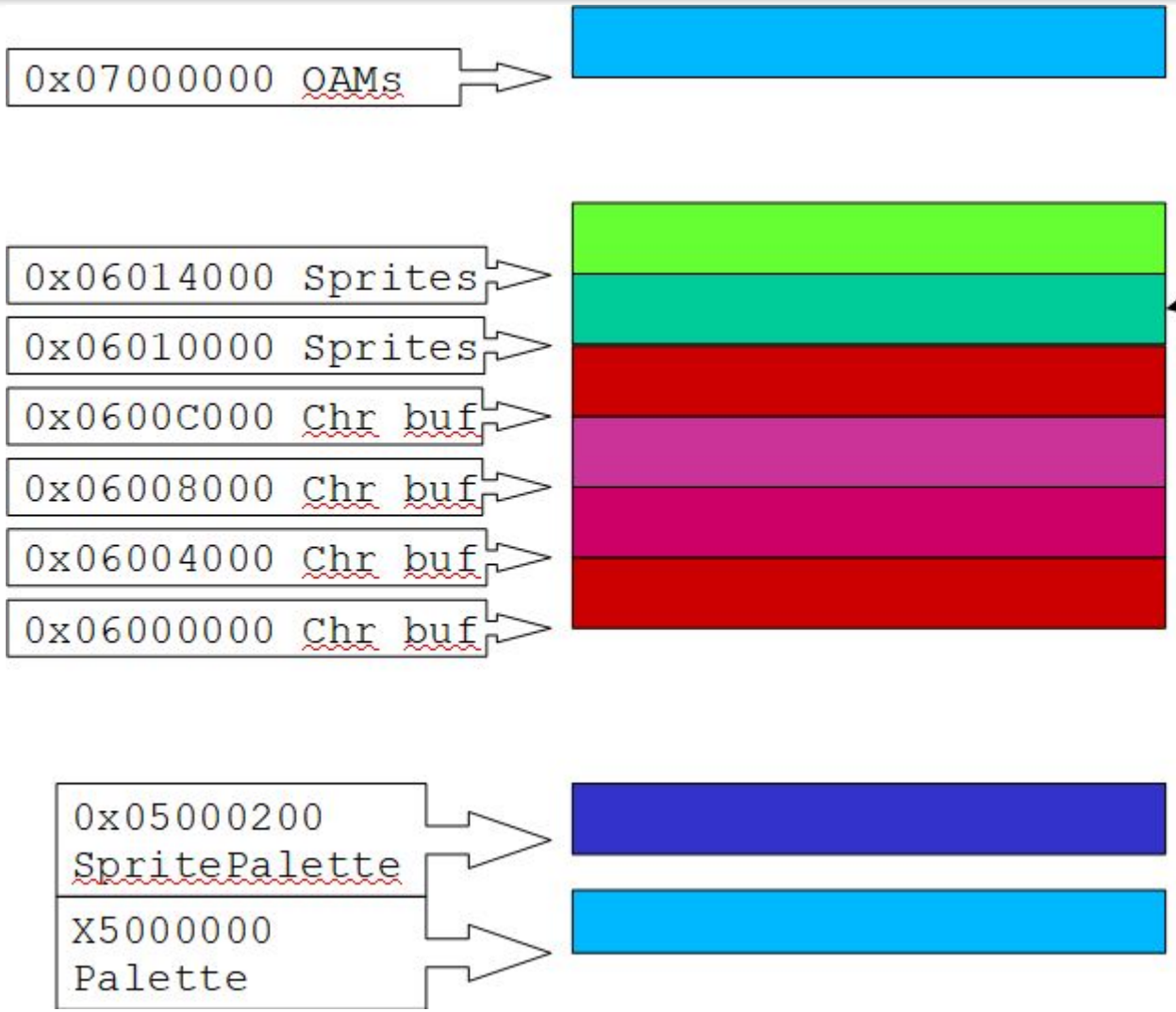
- Load sprite palette into sprite palette memory
  - Just after the Mode 4 pallet at 0x0500200
- Load sprite graphics into OVRAM (Object Video RAM)
- Set sprite attributes in OAM (Object Attribute Memory)
- Enable sprite objects and select correct mapping mode in REG_DISPCNT

Video Buffer Layout for Bitmap Display Modes & Sprites



Note:
Not to Scale!

Video Buffer Layout for Tiles Display Modes & Sprites



Sprites are composed of Tiles, not just bitmaps

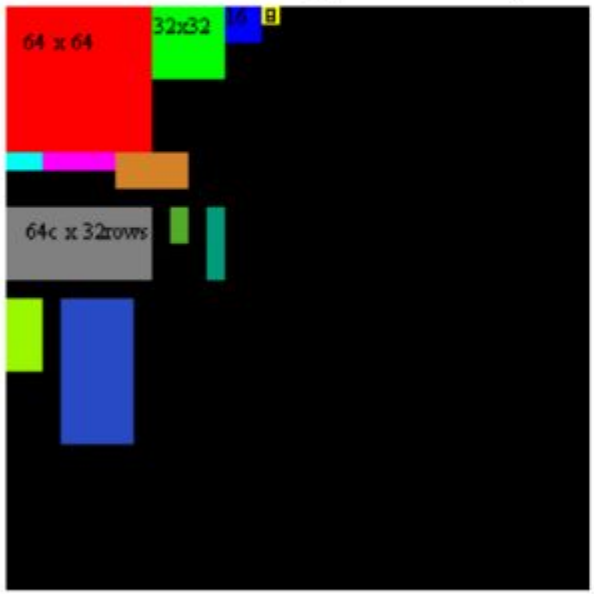
- Tiles themselves have either 4bits per byte (16 colors) or 8 bits per byte (256 colors)
- How many tiles and at what bpp is best?
- Sprite sheet of 32 tiles x 32 tiles = 1024 total tiles
  - 1024 x 64 pixels/tile = 65536 total pixels
  - With 8bpp that's 65536 bytes
  - With 4bpp that's half or 32768 bytes
    - That's 32KB, which is the size of Sprite Memory
    - So a 32 tile x 32 tile sprite sheet at 4bpp fits nicely (tightly) into
- Usenti will still export as shorts

Color Modes

- 4bpp
  - 16 colors
  - 32 bytes per 8x8 sprite (smallest)
- 8bpp
  - 256 colors
  - 64 bytes per 8x8 sprite
- For most applications,16 color sprites are
- preferred
  - Tiles are smaller
  - Sprites are smaller
  - More to work with!

Visually

- Sprite image area 16 color sprites (4bpp)
- 256x256 (pixels)



- Sprite image area 256 color sprites (8bpp)
- 128x256 (pixels)

Indices

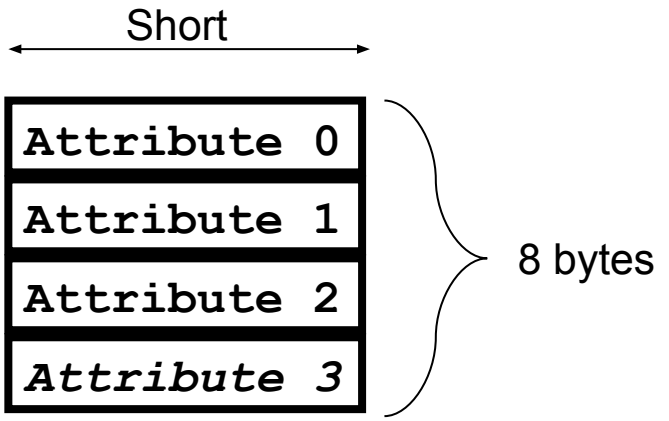
- The two character blocks dedicated to sprite images are divided into 32 byte long chunks.
- When telling the game boy where to find a sprite's image you give it the index of the chunk where the image starts.

Sizing

- Character blocks are 16 kb long
- Smallest sprite (8x8, 16 colors) uses 32 bytes
- Therefore a character block can hold 512 of them
- Switch to 256 colors and reduce the number per character block to 256
- The largest sprite is 64x64 and at 256 colors takes up 4096 bytes so a character block can hold 4 of them!

OAM

- Object Attribute Memory
- Location: 0x7,000,000
- Size: 1Kb



$$\frac{1Kb}{8b} = 128 sprites$$

Attribute 0

0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Shape		256			α	OM		Row Location							

- Bits 0-7: Row location (Top of Sprite)
- Bits 8-9: Object Mode:
  - 00: Regular, 01: Affine, 10: Hide
- Bit 10: Enable alpha blending
- Bit 13: 16 colors if cleared, 256 colors if set
- Bits 14-15: Sprite shape:
  - 00: Square, 01: Wide, 10: Tall

Attribute 1

0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Size		Flip					Column Location								

- Bits 0-8: Column location (Left side of sprite)
- Bits 12-13: Horizontal/Vertical Flip
  - 00: No Flip, 01: Horizontal Flip, 10: Vertical Flip
- Bits 14-15: Sprite Size
  - 00: 8 pixels, 01: 16, 10: 32, 11: 64

Attribute 2

0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Sub-Palette				Priority		Base index of sprite image									

- Bits 0-9: Base index of sprite image
- Bits 10-11: Draw priority
  - Higher priorities drawn first
  - For same priority sprites, higher OAM entry number is drawn first
- Bits 12-15: Sub-palette for 16 color sprites

Size and Shape

Size		00	01	10	11
Shape					
	00	8x8	16x16	32x32	64x64
Wide	01	16x8	32x8	32x16	64x32
Tall	10	8x16	8x32	16x32	32x64
			col x row		

Attribute 3

- There is no Attribute 3 but you have to leave space for it nevertheless!

OBJ_ATTR Struct

```
#define ALIGN(x) __attribute__((aligned(x)))

typedef struct
{
  u16 attr0;
  u16 attr1;
  u16 attr2;
  short fill;
} ALIGN(4) OBJ_ATTR;
```

ShadowOAM

Note: You can't write to OAM during VDraw, so you have to set up an equivalent space in memory somewhere else, then DMA that into OAM during VBlank.

We'll call this thing the ShadowOAM:

```
OBJ_ATTR shadowOAM[128];
```

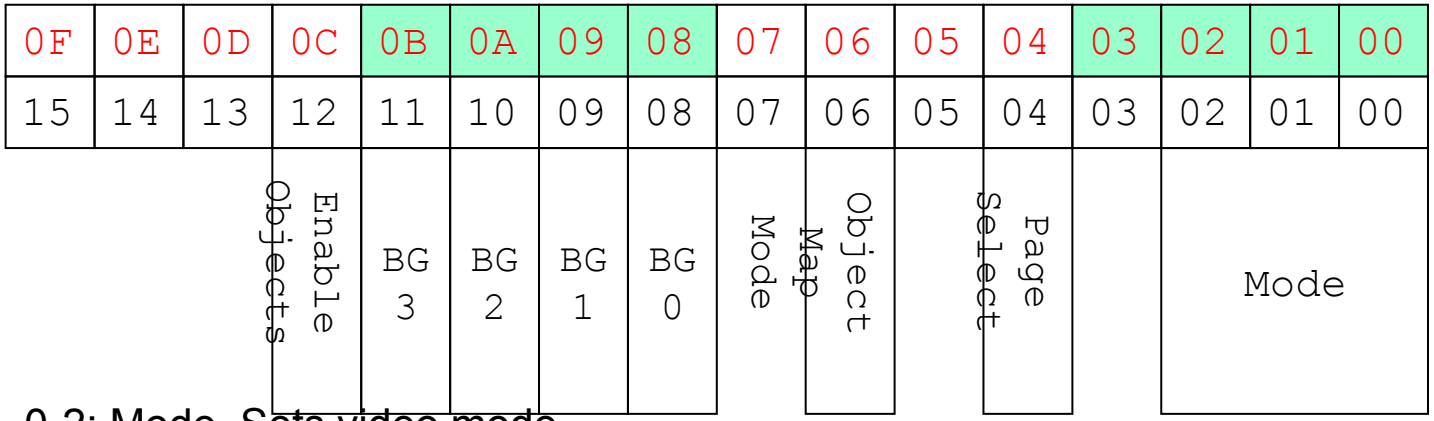
Sprite Setup

- Copy palette into palette memory (DMA!)
- Copy image into OVRAM (DMA, again!)
- Setup shadowOAM
  - Hide all the sprites you won't be using
  - Update shadowOAM whenever you want, later
- Copy shadowOAM actual OAM (DMA...)
  - You'll be doing this again during every VBlank!
- Set appropriate bits in REG_DISPCNT

Sprite Use

- Change/update location or tile index of all updated sprites in the shadowOAM buffer
- On vertical blank, DMA copy the buffer into the OAM location

REG_DISPCNT



- 0-2: Mode. Sets video mode.
  - 0, 1, 2 are tiled modes; 3, 4, 5 are bitmap modes.
- 4: Page select. Modes 4 and 5 can use page flipping for smoother animation. This bit selects the displayed page (and allowing the other one to be drawn on without artifacts).
- 6: Object mapping mode.
  - 0 is 2D, 1 is 1D
- 8, 9, 10, 11 BG0, BG1, BG2, BG3
- 12 Objects Enable (Set to 1 to enable sprites)

Better Sprite Structure

- Direct access to key variables like x, y, and tile index
- Functions to compress variables into the attributes
- More object oriented approach, easier to handle many sprites at once

Mode 0 Intro / Overview

Atari

- **196?:** As an engineering student at the University of Utah, Nolan Bushnell liked to sneak into the computer labs late at night to play computer games on the university's $7 million mainframes.
- **1972:** Bushnell founded Atari with $250 of his own money and another $250 from business partner Ted Dabney. They then created and commercialized the world's first commercial video game, Pong. Bushnell was 27 years old.
- **1976:** Warner Communications buys Atari from Bushnell for $28 million.
- **1977:** Atari introduces the Atari Video Computer System (VCS), later renamed the Atari 2600
- **1978: December - Atari announces the Atari 400 and 800 personal computers.**
- **1979: October - Atari begins shipping the Atari 400 and Atari 800 personal computers.**


ATARI CXB53 16K RAM
ATARI CXB01 10K ROM


ATARI 800


ATARI 800

Typical Atari 800 Screenshot

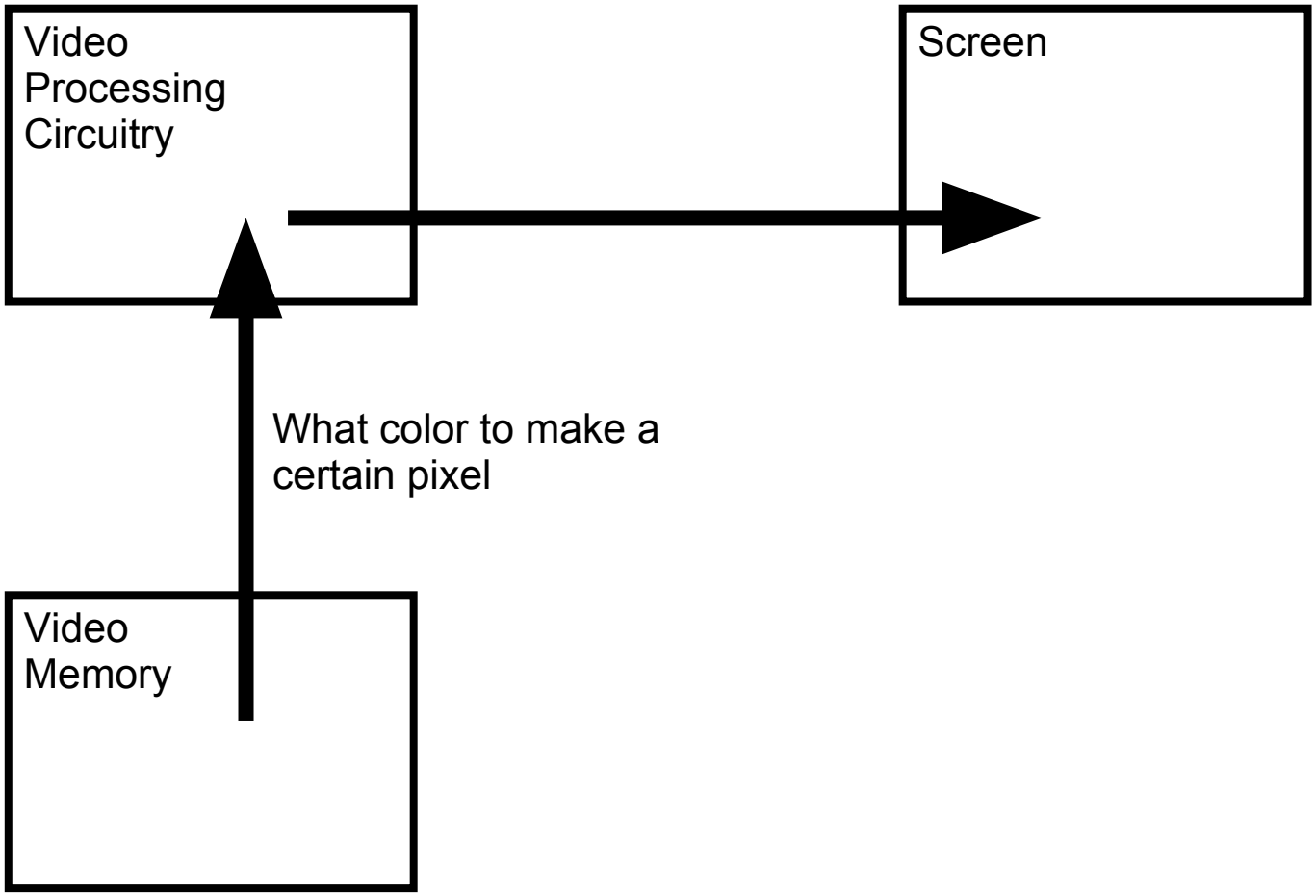
Atari Graphics

- Graphics and Text Modes
- Display List
- Player/Missle Graphics
- Customizable Fonts
  - Area in memory that described fonts (characters)
  - Area in memory that was what went on *screen*

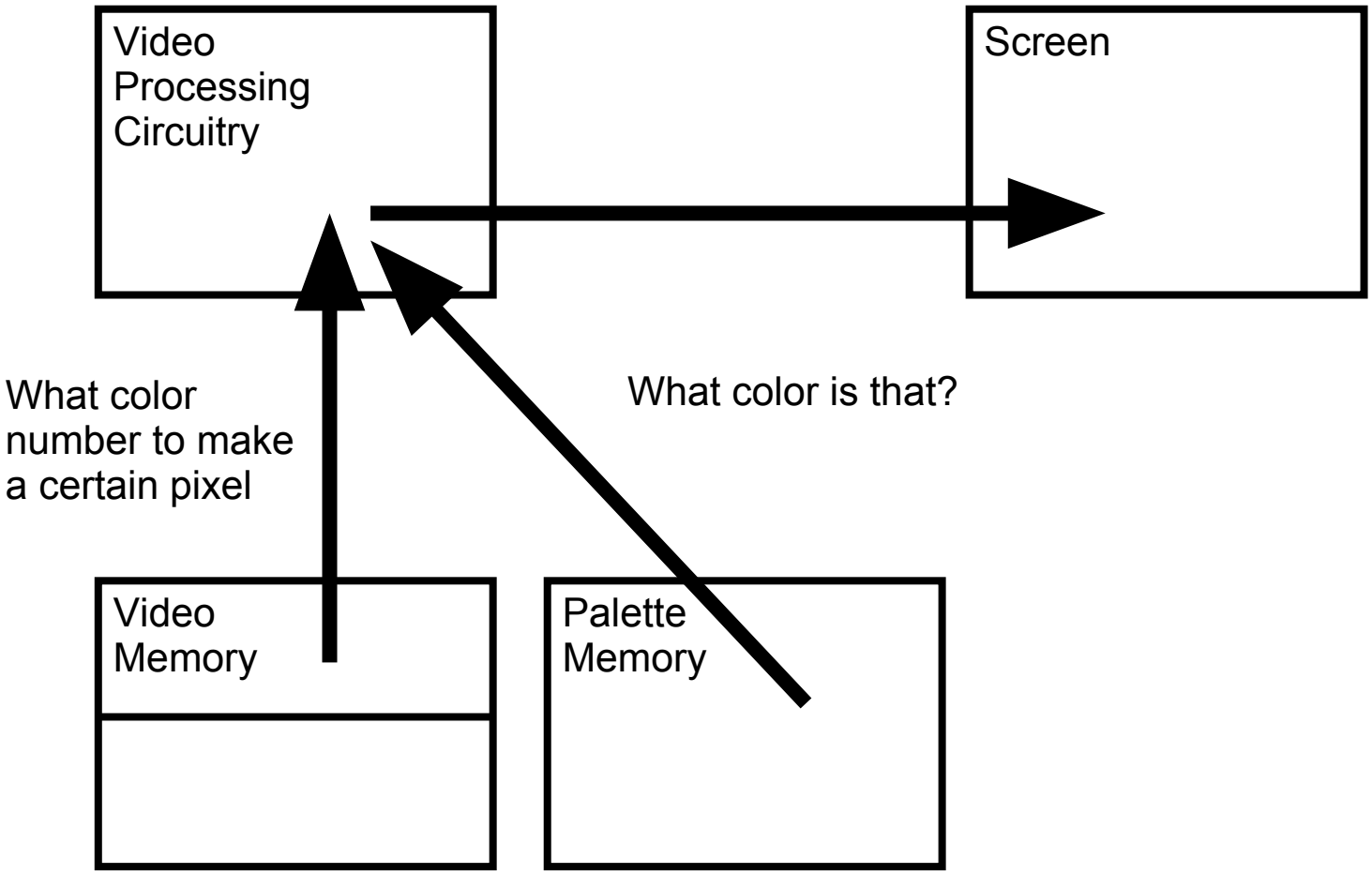
GBA Tiles

- Logical extension of the customizable character sets. Some of the terms used today are based on the historical derivation of the technology
- (The logical extension of the player/missle graphics is sprites)

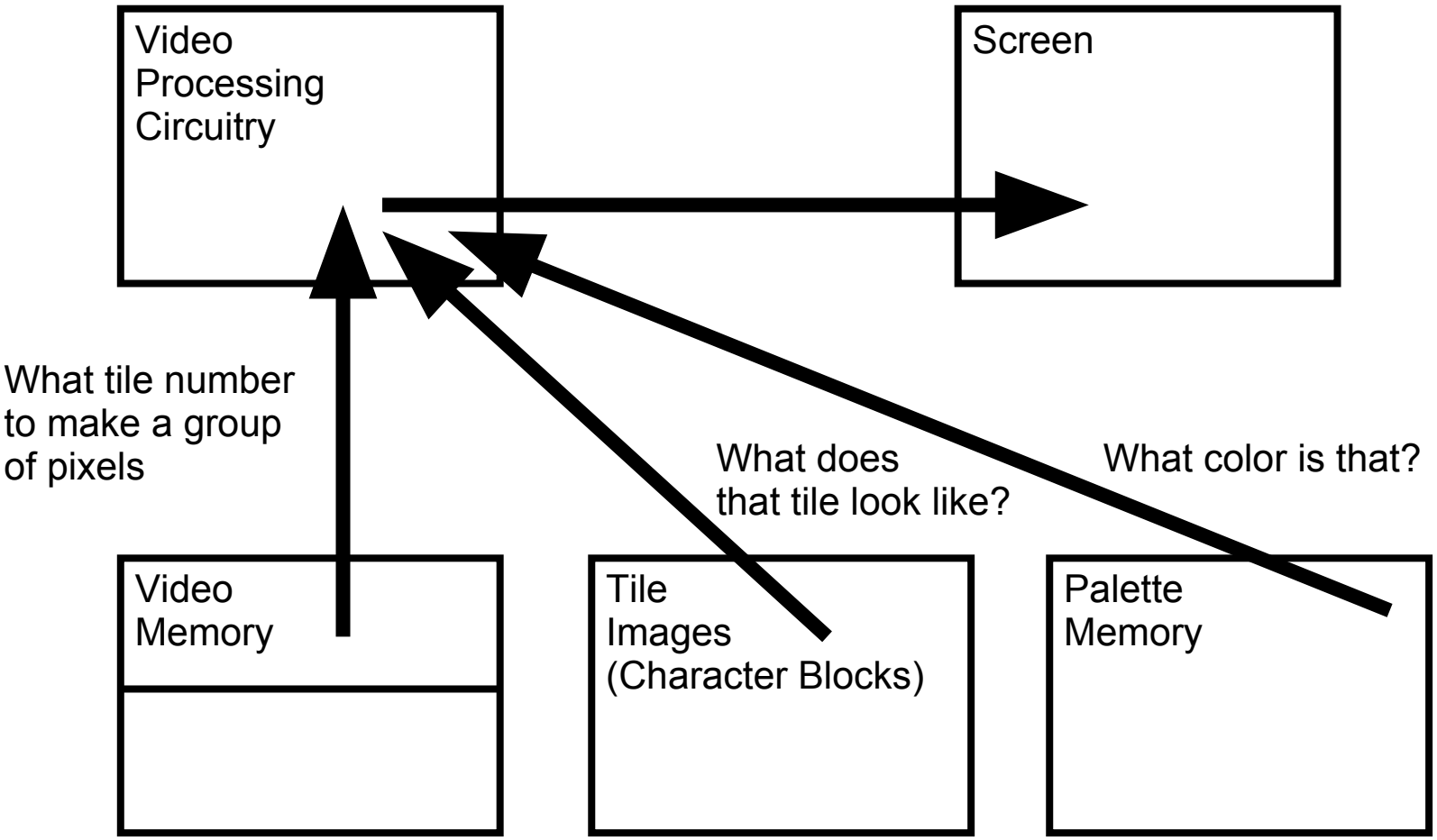
Mode 3 Bitmaps



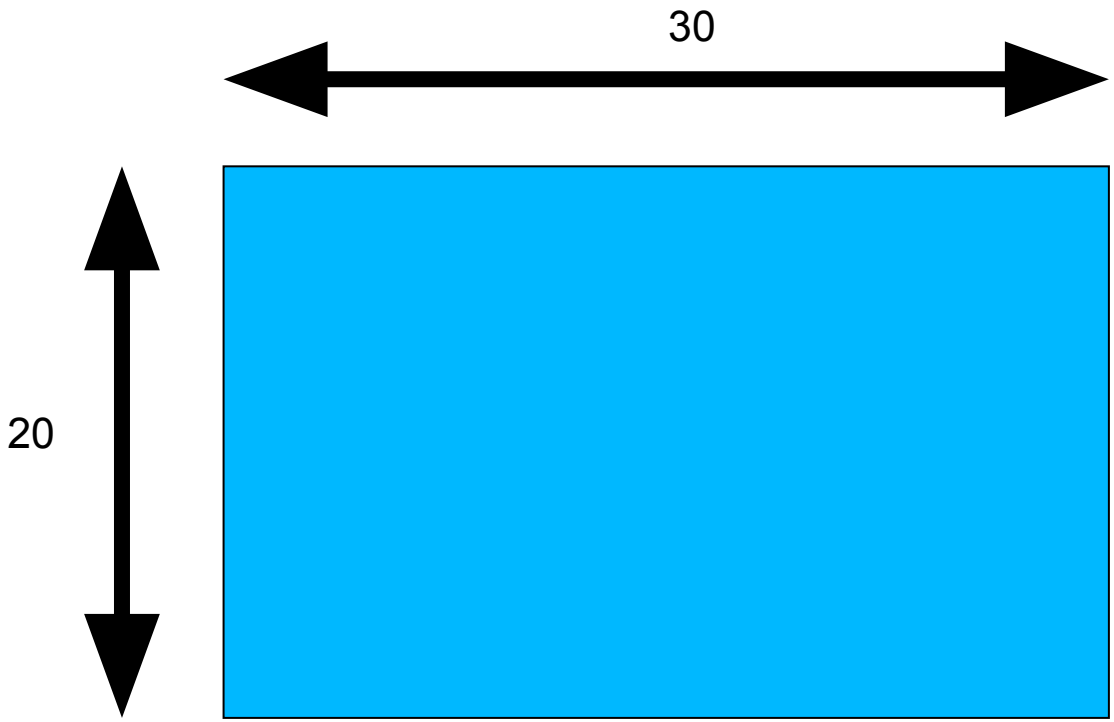
Mode 4 Bitmaps



Mode 0 Tilemaps



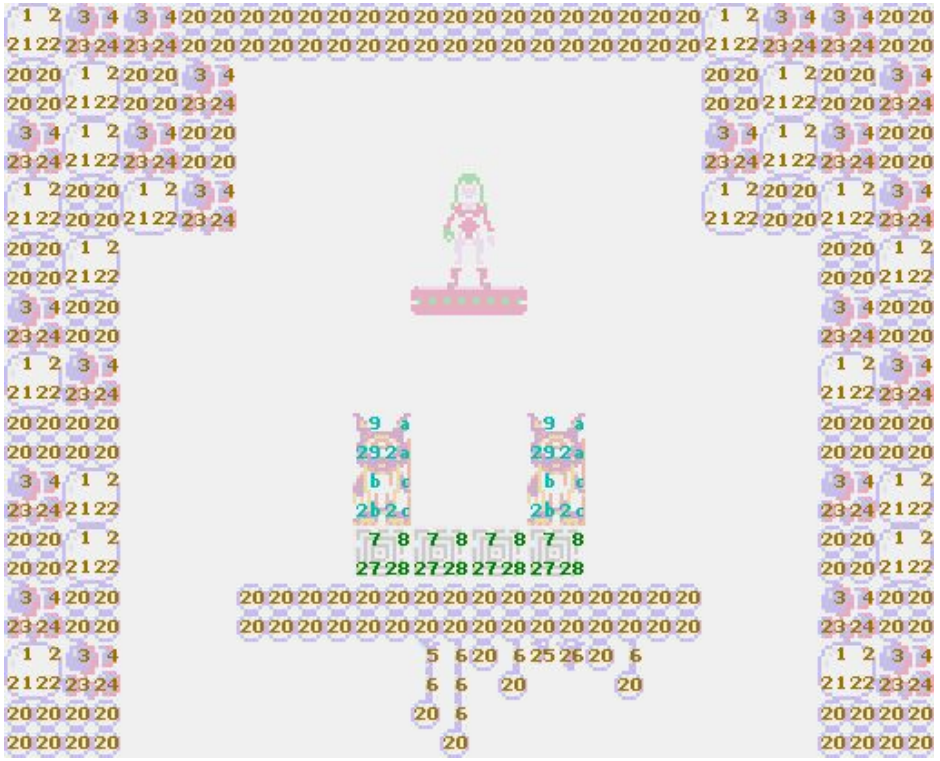
Tiles On Screen



Tiles & Tilemaps

- Tile
  - 8x8 bitmap image
    - Treated like big pixels, as the building block of an image
  - 16 or 256 colors
- Tile Map
  - 2D array of tile indexes
  - Tile : Tilemap :: Pixel : Bitmap

Illustrations from Tonc



GBA Tile Mode Features

- Up to 4 tiled background layers
  - Up to 1024x1024 pixels each
- Hardware scrolling
  - Maps can be larger than the screen
- Hardware Parallax (optional)
  - Layers scroll at different speeds to simulate depth
- Affine transformation (optional)
  - Rotation and scaling effects

Steps to Tiles

- Create tile images store them in a character block
- Create screen image and store it in one or more screen blocks
- Create a palette
- Set image control buffer

Steps to Tiles

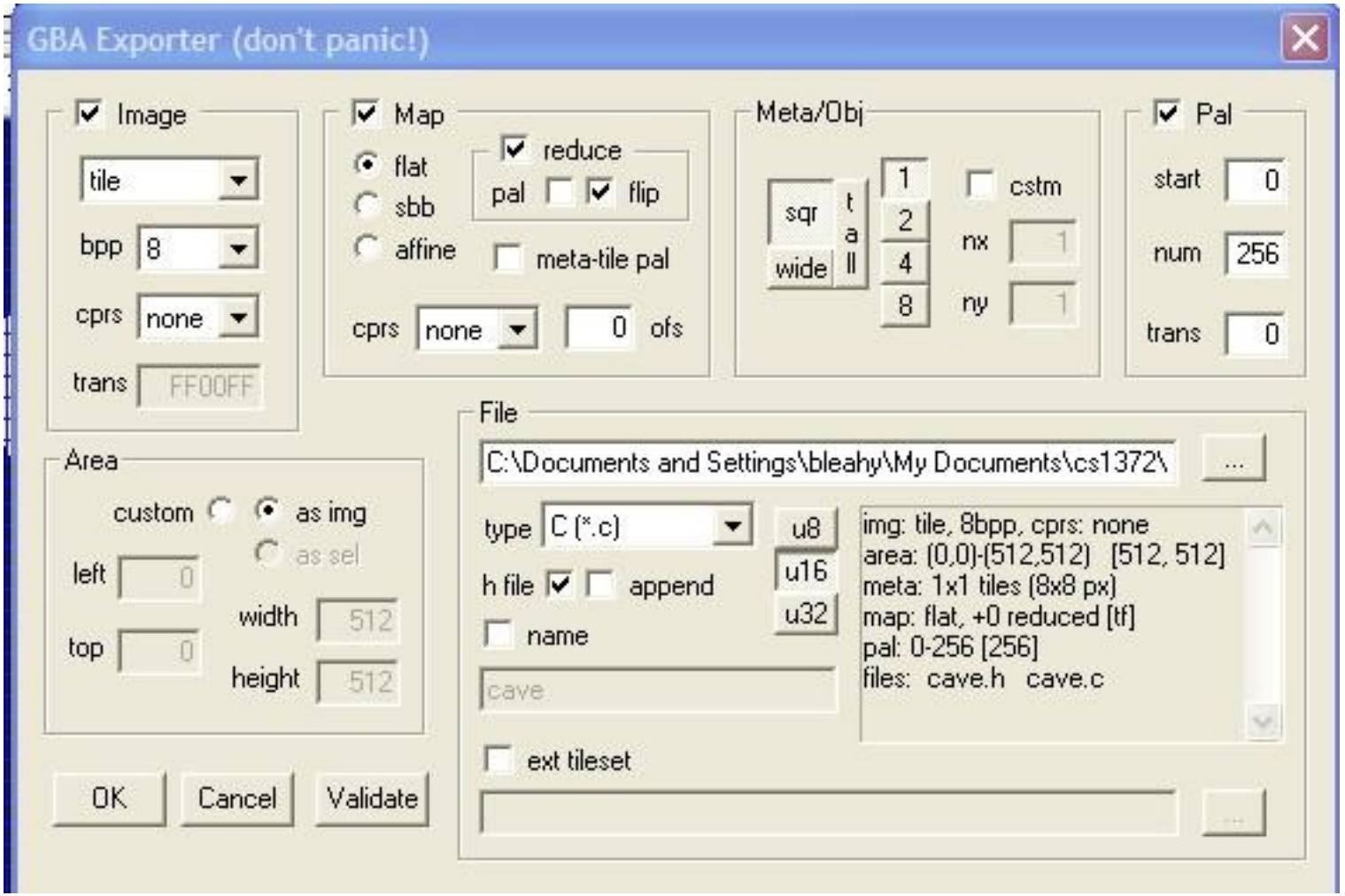
- Create tile images
- Create screen image
- Create a palette
- Store tile images in a character block
- Store screen image in one or more screen blocks
- Store palette in palette memory area
- Set image control buffer and Mode

Steps to Tiles

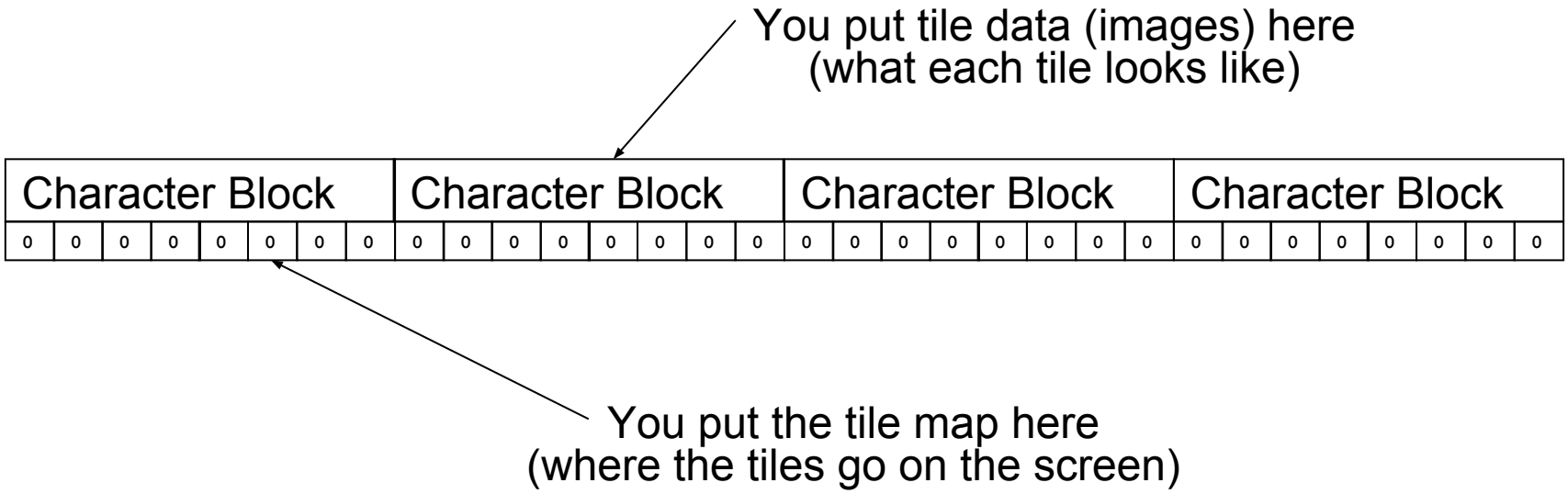
- Create tile images
- Create screen image
- Create a palette

- Store tile images in a character block
- Store screen map in one or more screen blocks
- Store palette in palette memory area
- Set image control buffer and Mode

usenti

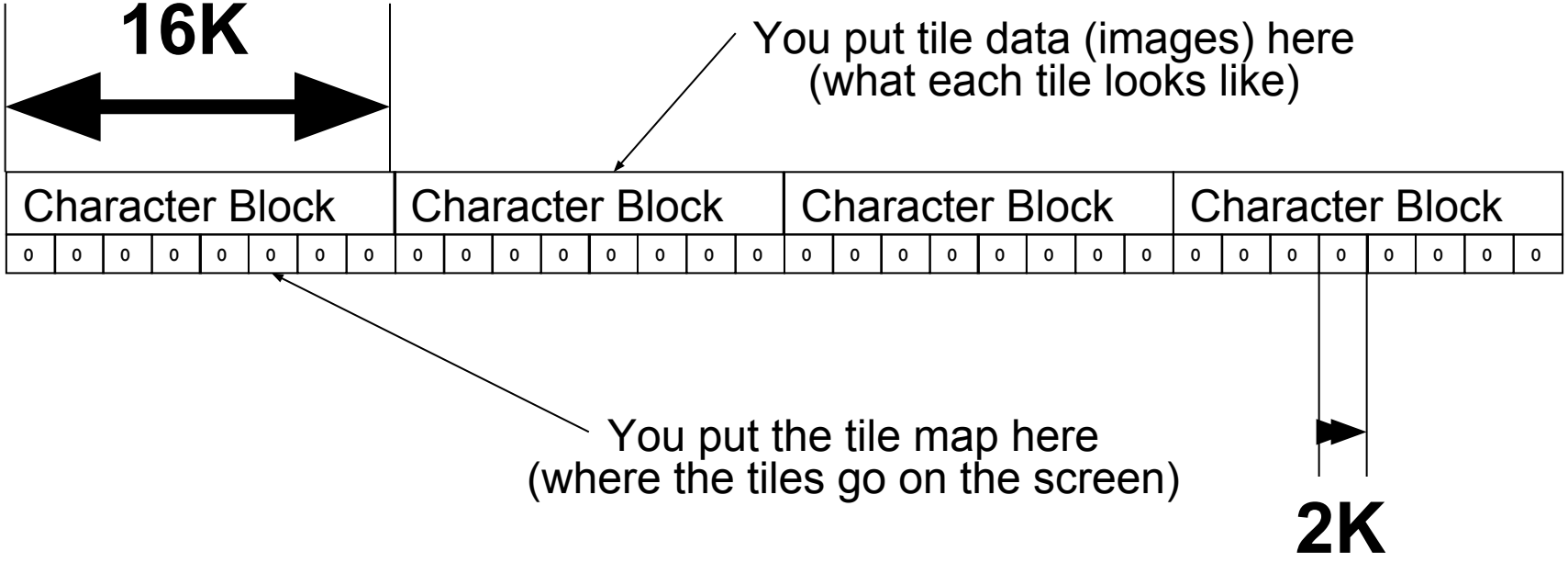

GBA Exporter (don't panic!)
Image
tile
bpp 8
cprs none
trans FF00FF
Map
flat
sbb
affine
reduce
pal
flip
meta-tile pal
cprs none
0 ofs
Meta/Obj
sqr
wide
tall
1
2
4
8
cstm
nx 1
ny 1
Pal
start 0
num 256
trans 0
Area
custom
as img
as sel
left 0
top 0
width 512
height 512
File
C:\Documents and Settings\bleahy\My Documents\cs1372\
...
type C (*.c)
u8
u16
u32
h file
append
name
cave
img: tile, 8bpp, cprs: none
area: (0,0)-(512,512) [512, 512]
meta: 1x1 tiles (8x8 px)
map: flat, +0 reduced [tf]
pal: 0-256 [256]
files: cave.h cave.c
ext tileset
OK
Cancel
Validate

Video Buffer Layout for Tiles & Sprites



Sprite Data	Sprite Data

Video Buffer Layout for Tiles & Sprites



Sprite Data	Sprite Data

Mode & BG Specifics

Mode	Backgrounds	Rotation/Scaling
0	0, 1, 2, 3	No
1	0, 1, 2	Yes (only background 2)
2	2, 3	Yes

Background	Resolution	Rotation/Scaling
0	512 x 512	No
1	512 x 512	No
2	128 to 1024	Yes
3	128 to 1024	Yes

Bg Layers

- 0 & 1
  - Max Resolution 512x512
  - Tile map of 16 bit numbers
- 2 & 3
  - Resolution from 128x128 to 1024x1024
  - Rotation & Scaling
  - Tile map of 8 bit numbers

Color

- 256 color tiles
  - 8 bits per pixel
  - Tiles share one 256 color palette
- 16 color tiles
  - 4 bits per pixel
  - Tiles use one of 16 palettes, 16 colors each

Correct

Incorrect: 256 color tiles being displayed as 16 color tiles

Tile Memory

- **Double the sprite memory compared with bitmap modes**
- Tile data video in video buffer
  - From 0x6000000 to 0x600FFFF
  - On 16k boundary (any of 4 spots)
- Tile map also stored in video buffer
  - On 2k boundary (any of 32 spots)
- Tile data and Tile map are in same space but normally would not overlap.

Tile Data and Tile Map

- The tile map is stored in the same location as the video buffer (in the bitmap video modes), an array of numbers that point to the tile images.
- In text backgrounds (0 and 1) the tile map is comprised of 16-bit numbers, while the rotation backgrounds (2 and 3) store 8-bit numbers in the tile map.
- When working with tile-based modes, video memory is divided into 4 logical char base blocks, which are made up of 32 smaller screen base blocks, as shown on the next slide.

Tile Setup

```
//background setup registers and data
#define REG_BG0CNT *(volatile unsigned short*)0x4000008
#define REG_BG1CNT *(volatile unsigned short*)0x400000A
#define REG_BG2CNT *(volatile unsigned short*)0x400000C
#define REG_BG3CNT *(volatile unsigned short*)0x400000E
#define BG_COLOR256 0x80
#define CHAR_SHIFT 2
#define SCREEN_SHIFT 8
#define WRAPAROUND 0x1

//background tile bitmap sizes
#define TEXTBG_SIZE_256x256 0x0
#define TEXTBG_SIZE_256x512 0x8000
#define TEXTBG_SIZE_512x256 0x4000
#define TEXTBG_SIZE_512x512 0xC000

//background memory offset macros
#define CharBaseBlock(n) (((n)*0x4000)+0x6000000)
#define ScreenBaseBlock(n) (((n)*0x800)+0x6000000)

//background mode identifiers
#define BG0_ENABLE 0x100
#define BG1_ENABLE 0x200
#define BG2_ENABLE 0x400
#define BG3_ENABLE 0x800
```

Tile setup (cont)

```
//video identifiers
#define REG_DISPCNT *(unsigned int*)0x4000000
#define BGPaletteMem ((unsigned short*)0x5000000)
#define SetMode(mode) REG_DISPCNT = (mode)

//vertical refresh register
#define REG_DISPSTAT   *(volatile unsigned short*)0x4000004

//button identifiers
#define BUTTON_RIGHT 16
#define BUTTON_LEFT  32
#define BUTTON_UP    64
#define BUTTON_DOWN  128
#define BUTTONS (*(volatile unsigned int*)0x04000130)
```

Tile setup(cont)

```
int main(void)
{

    //create a pointer to background 0 tilemap buffer
    unsigned short* bg0map =(unsigned short*)ScreenBaseBlock(31);

    //set up background 0
REG_BG0CNT = BG_COLOR256 | TEXTBG_SIZE_256x256 | (31 << SCREEN_SHIFT) | WRAPAROUND;

//set video mode 0 with background 0
SetMode(0 | BG0_ENABLE);

//copy the palette into the background palette memory
DMAFastCopy((void*)test_Palette, (void*)BGPaletteMem, 256, DMA_16NOW);

//copy the tile images into the tile memory
DMAFastCopy((void*)test_Tiles, (void*)CharBaseBlock(0), 57984/4, DMA_32NOW);

//copy the tile map into background 0
DMAFastCopy((void*)test_Map, (void*)bg0map, 512, DMA_32NOW);
```

Input & Tile Scrolling

```
//main game loop
while(1)
{
        //wait for vertical refresh
WaitVBlank();

//D-pad moves background
if(!(BUTTONS & BUTTON_LEFT)) x--;
if(!(BUTTONS & BUTTON_RIGHT)) x++;
if(!(BUTTONS & BUTTON_UP)) y--;
if(!(BUTTONS & BUTTON_DOWN)) y++;

//use hardware background scrolling
REG_BG0VOFS = y ;
REG_BG0HOFS = x ;

         //wait for vertical refresh
WaitVBlank();

for(n = 0; n < 4000; n++);
}
return 0;
}
```