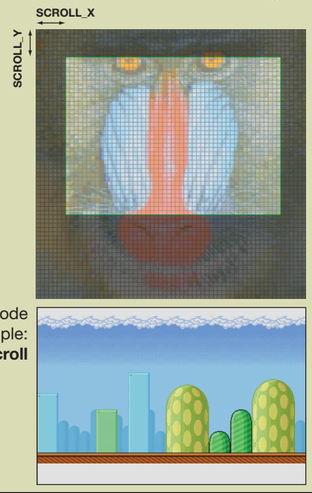
GAMEDUINO quick reference

screen

Screen RAM is a grid of 64×64 bytes, each byte is a single character index. This byte controls the character image and palette used for that 8×8 pixel cell. The total size of the background screen is 512×512 pixels, but only 400×300 pixels are visible. The **SCROLL_X** and **SCROLL_Y** registers control the position of this 400×300 pixel window within the larger screen area.



code sample: **scroll**

characters

Characters are 8×8 grids of pixels, defined by the values in the character data and palette RAMs. The character data RAM holds the 64 pixels of the character image, encoded using two bits per pixel. The hardware uses these two bits to look up the final color in the character's 4-entry palette.

For example, a character with a palette of blue, yellow, red and white might appear as shown below. In the left-hand square, the pixel values 0–3 are shown. In the middle square, these pixel values in binary are listed. In the right hand column are the hex values, as they appear in memory for this character.

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

00	01	01	01	01	01	01	01	15 55
00	01	10	10	10	01	01	01	1A 95
01	01	10	10	10	01	01	00	5A 94
00	01	10	11	11	11	11	11	1B FF
01	01	01	01	10	01	01	01	55 95
00	01	01	01	10	01	01	01	15 95
01	01	01	01	10	01	01	01	55 95
00	01	01	01	10	01	01	01	15 95

memory map

Gameduino has 32 kbytes of memory, organized into different functions. The background section controls background character graphics. The sprite section controls the foreground sprite graphics.

background	0x0000 0x0FFF	64x64 character screen
	0x1000 0x1FFF	character data 256 characters of 8x8 pixels
	0x2000 0x27FF	character palettes 256 characters of 4 colors
control registers	0x2800 0x2FFF	control registers collision RAM
sprites	0x3000 0x37FF	sprite control 256x4 bytes
	0x3800 0x3FFF	sprite palette 4 palettes of 256 colors
	0x2800 0x7FFF	sprite images 64 images of 16x16 bytes

registers

Registers control some simple functions of the Gameduino. Gameduino is little-endian, so 16-bit registers have their lower 8 bits at the lower address in memory.

address	bytes	name	description	access	reset value
0x2800	1	IDENT	Gameduino identification—always reads as 0x6D	r	0x6d
0x2801	1	REV	Hardware revision number. High 4 bits are major revision, low 4 bits are minor	r	0x10
0x2802	1	FRAME	Frame counter, increments at the end of each displayed frame	r	0
0x2803	1	VBLANK	Set to 1 during the video blanking period	r	0
0x2804	2	SCROLL_X	Horizontal background scrolling register, 0–511	r/w	0
0x2806	2	SCROLL_Y	Vertical background scrolling register, 0–511	r/w	0
0x2808	1	JK_MODE	Sprite collision class mode enable 0–1	r/w	0
0x280A	1	SPR_DISABLE	Sprite control: 0 enable sprite display, 1 disable sprite display	r/w	0
0x280B	1	SPR_PAGE	Sprite page select: 0 display from locations 0x3000–0x33FF, 1 from 0x3400–0x37FF	r/w	0
0x280C	1	IOMODE	Pin 2 mode: 0=disconnect, 0x46=flash enable, 0x4A=coprocessor control	r/w	0
0x280E	2	BG_COLOR	Background color	r/w	0
0x2810	2	SAMPLE_L	Audio left sample value, 16 bit signed -32768 to +32767	r/w	0
0x2812	2	SAMPLE_R	Audio right sample value, 16 bit signed -32768 to +32767	r/w	0
0x281E	2	SCREENSHOT_Y	Screenshot line select 0–299	r/w	0
0x2840	32	PALETTE 16A	16-color sprite A palette	r/w	0000 (black)
0x2860	32	PALETTE 16B	16-color sprite B palette	r/w	0000 (black)
0x2880	8	PALETTE 4A	4-color sprite A palette	r/w	0000 (black)
0x2888	8	PALETTE 4B	4-color sprite B palette	r/w	0000 (black)
0x2900	256	COLLISION	Collision RAM	r	0
0x2A00	256	VOICES	Audio voice controls	r/w	0
0x2B00	800	SCREENSHOT	Screenshot line RAM	r	0

sprite control

Gameduino has 256 hardware sprites: 16×16 pixel images that can appear anywhere on the screen. Sprites are drawn from back-to-front, so higher-numbered sprites cover up lower-numbered ones. Each sprite's appearance is controlled by a 32-bit word:

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0				
C	IMAGE	Y	PAL	ROT	X

X X coordinate, 0-511. 0 is the left edge of the screen, 399 is the right edge.

Y Y coordinate, 0-511. 0 is the top edge of the screen, 299 is the bottom edge.

IMAGE Spriteimageselect,0-63. Selects which source image the sprite uses.

PAL Sprite palette select, 4 bits. Controls how pixel data is turned into color.

ROT Sprite rotate, 3 bits. Rotates a sprite by quarter-turns.

C Collision class, 0 is J, 1 is K.

To hide a sprite park it off the screen by setting its Y coordinate to 400.

To make a large sprite draw several sprites together in a grid pattern. For example, four 16x16 sprites can be arranged to make a single 32x32 sprite.

To spin a sprite use the ROT field to rotate it, and use extra animation frames for finer rotations.

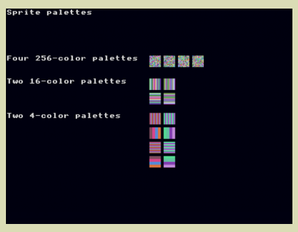
To animate a sprite you can

- change the IMAGE field
- change the PAL field to do simple color animation
- change the ROT field to apply flips and rotates of 90, 180 and 270 degrees
- load new data to the source image.

sprite palette select

Each pixel of the sprite image is fetched and looked up in a sprite palette. This palette is a list of colors. Gameduino gives you several palette options: a 256-color palette, a 16-color palette and a 4-color palette. Why not always use the 256-color palette? Because using the smaller palette options lets you squeeze more images into memory. 256 bytes of sprite image RAM can hold one 16x16 sprite image in 256-color mode, two images in 4-bit mode (with a 16 color palette), or four images in 2-bit mode (4 color palette).

3	2	1	0	
0	0	P		**8–bit mode** Each byte indexes into 256-color palette, P
0	1	N	AB	**4–bit mode** The high (N=1) or low (N=0) 4 bits from each byte index into 16-color palette A (AB=0) or B (AB=1)
1	N		AB	**2–bit mode** Two bits from each byte (N=3 is highest, N=0 is lowest) index into 4-color palette A (AB=0) or B (AB=1)



code sample: **palettes**

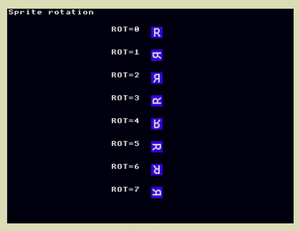
sprite rotate

Each sprite has a 3-bit ROT field that applies a simple rotation and flip to the sprite image.

2	1	0
Y flip	X flip	XY swap

Y flip flip the image top-to-bottom
X flip flip the image left-to-right
XY swap flip the image diagonally
By using these in combination, the sprite image can be rotated:

ROT	Y flip	X flip	XY swap	results
0	0	0	0	R
1	0	0	1	R
2	0	1	0	R
3	0	1	1	R
4	1	0	0	R
5	1	0	1	R
6	1	1	0	R
7	1	1	1	R



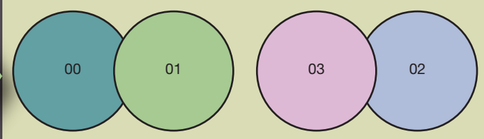
code sample: **rotation**

sprite collision detection

The Gameduino has a special memory area that you can use to detect when sprites overlap. As it draws the image, Gameduino keeps track of which pixels cover others, and writes the results into the collision RAM.

Each byte in the collision RAM corresponds with the same numbered sprite. If the sprite does not cover another then the byte's value is 0xFF. But if the sprite covers any part of another sprite, then the value is the number of the other sprite.

For example, if sprites 00–03 are arranged like this:



address	value	meaning
0x2900	FF	sprite 00 did not cover up any other sprite
0x2901	00	sprite 01 covered up some pixels from sprite 00
0x2902	FF	sprite 02 did not cover up any other sprite
0x2903	02	sprite 03 covered some pixels from sprite 02

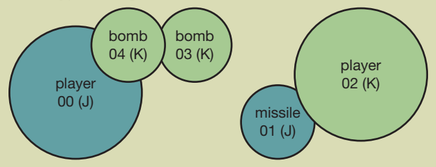
code sample: **collision**

sprite collision class

In a game you might have the following rules:

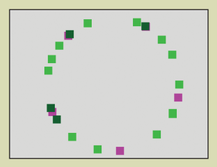
- when the player touches an enemy bomb, the player dies
- when a player's missile touches an enemy, that enemy dies.

Here is a typical in-game situation:



Notice that bomb 04 covers both bomb 03 and player 00. In this situation we're much more interested that bomb 04 is covering player 00. For this reason, the hardware has a mode JK_MODE where it ignores "friendly" collisions. In this mode, each sprite belongs a collision class J or K. Collision notifications only happen when a J sprite covers up a K sprite, or when a K sprite covers up a J sprite.

address	value	meaning
0x2900	FF	sprite 00 no collision
0x2901	FF	sprite 01 no collision
0x2902	01	sprite 02 covers some pixels from sprite 01
0x2903	FF	sprite 03 no collision
0x2904	00	sprite 04 covers some pixels from sprite 00



code sample: **jkcollision**

colors

Gameduino stores colors in an ARGB1555 format:

15	14 13 12 11 10	9 8 7 6 5	4 3 2 1 0
A	R	G	B

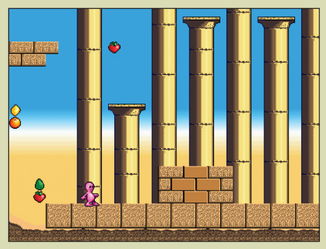
Each color field red (**R**) green (**G**) and blue (**B**) has a range 0–31.

Gameduino is little-endian, so a color stored in two bytes stored starting at address is:

	7	6	5	4	3	2	1	0
address	G_2	G_1	G_0	B_4	B_3	B_2	B_1	B_0
address+1	A	R_4	R_3	R_2	R_1	R_0	G_4	G_3

The **A** bit controls transparency. When A=1 the pixel is transparent and the other fields are ignored.

For sprites, transparent pixels show through the background layer. For the background layer, transparent pixels show BG_COLOR.



code sample: **bgcolor**