

# 8 BITS DE PODER

En tu AMSTRAD CPC

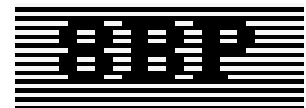
THE ULTIMATE RSX LIBRARY FOR GAMES



Jose Javier Garcia Aranda

[8bitsdepoder.blogspot.com](http://8bitsdepoder.blogspot.com)

<http://github.com/jjaranda13/8bp>



29 Abril 2017

Retro  
madrid



# AGENDA



1 | Introducción a 8BP

2 | lógicas masivas

3 | crear un proyecto

4 | sprites

5 | layout (tile map)

6 | scroll

7 | música

8 | recomendaciones

# 1 | Introducción a 8BP

¿Por qué programar en 2017 una máquina de 1984?

CPU	Número de transistores	MIPS (millones de instrucciones por segundo)	Ordenadores y consolas que lo incorporan
6502	3.500	0.43 @1Mhz	COMMODORE 64, NES, ATARI 800...
Z80	8.500	0.58 @4Mhz	AMSTRAD, COLECOVISION, SPECTRUM, MSX...
Motorola 68000	68.000	2.188 @ 12.5 Mhz	AMIGA, SINCLAIR QL, ATARI ST...
Intel 386DX	275.000	2.1 @16Mhz	PC
Intel 486DX	1.180.000	11 @ 33 Mhz	PC
Pentium	3.100.000	188 @ 100Mhz	PC
ARM1176		4744 @ 1Ghz (1186 por core)	Raspberry pi 2, Nintendo 3DS, Samsung galaxy, ...
Intel i7	2.600.000.000	238310 @ 3Ghz (¡casi 500.000 veces más rápido que un Z80 !)	PC

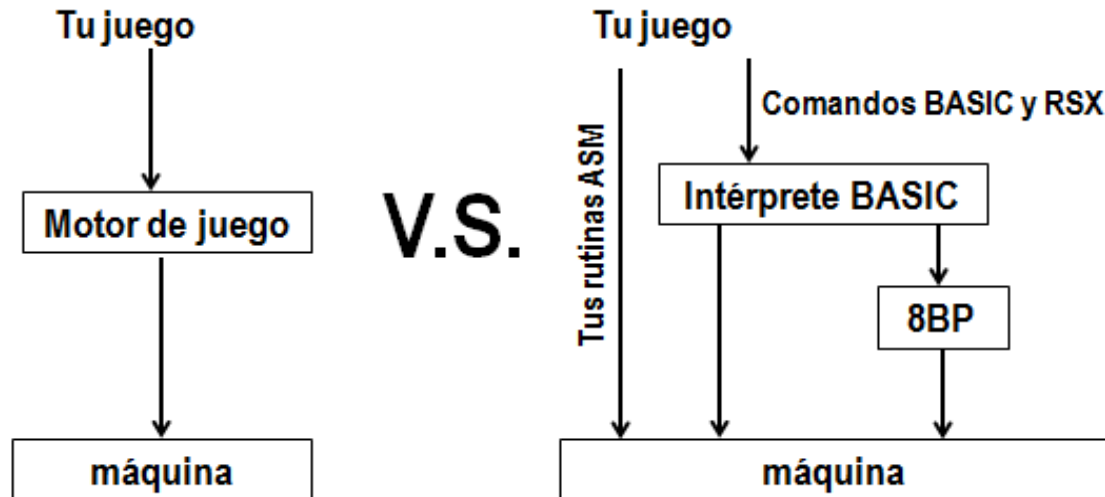


Porque:

“Las limitaciones no son un problema, sino una fuente de inspiración”

JJGA.

# 1 | Introducción a 8BP



RSX es el acrónimo de Resident System eXtensions. Las librerías como 8BP que proporcionan comandos para extender el BASIC se les llama librerías RSX

*¿Puedo usar 8BP desde Locomotive BASIC? si*  
*¿Puedo usar 8BP desde lenguaje C? Si*  
*¿Puedo usar 8BP desde ensamblador? Si*  
*¿Puedo usar 8BP desde CPC Basic Compiler? si*

# ¿Qué es 8BP?

Es una librería RSX.  
Añade nuevos comandos  
a LOCOMOTIVE BASIC  
para hacer juegos de  
Calidad profesional. Y  
ocupa solo 6KB!!

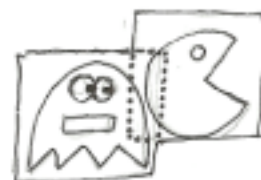


Imprime 32 sprites  
simultáneamente y  
controla su posición,  
velocidad, secuencia de  
animación, trayectoria...  
**|PRINTSPALL**

Mueve grupos de enemigos  
en bloque **|MOVERALL,**  
**|AUTOALL**

Sprite	x	y	vx	vy
0				
1				
2				
...				
31				

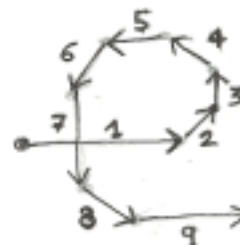
Crea secuencias de  
animación y  
"macrosecuencias"  
**|SETUPSP, |SETUPSQ**



Detecta colisiones entre  
tus sprites  
**|COLSP, |COLSPALL**

Soporta  
sobreescritura

Soporta clipping **|SETLIMITS**



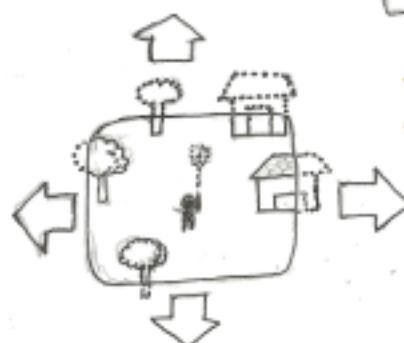
Crea tus trayectorias y  
asígnaselas a tus  
sprites para que las  
recorran  
**|ROUTEALL, |AUTOALL**



Crea tus layouts para  
juegos de pantallas con  
laberintos  
**|LAYOUT, |COLAY**



Soporta scroll  
multidireccional  
**|MAP2SP**



Mueve grupos de  
estrellas para crear  
efectos de espacio,  
lluvia, tierra,...  
**|STARS**



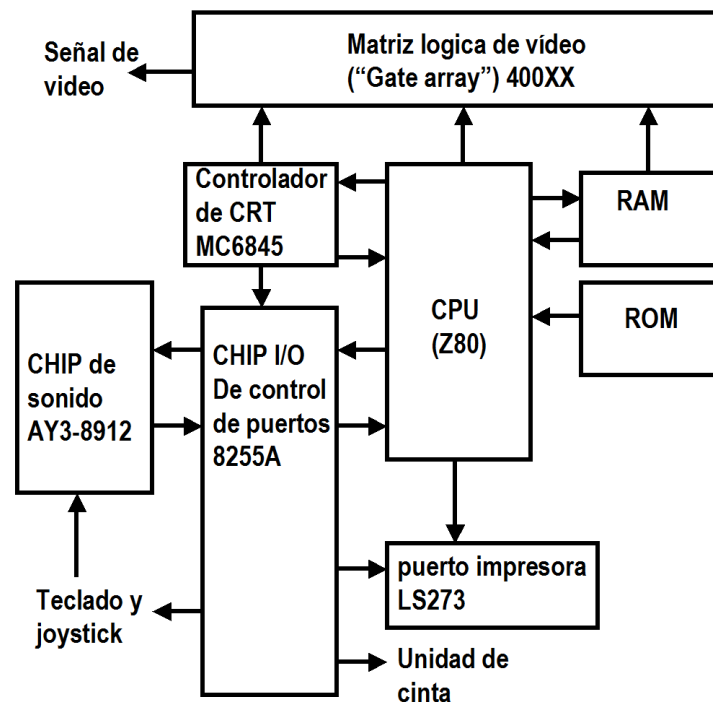
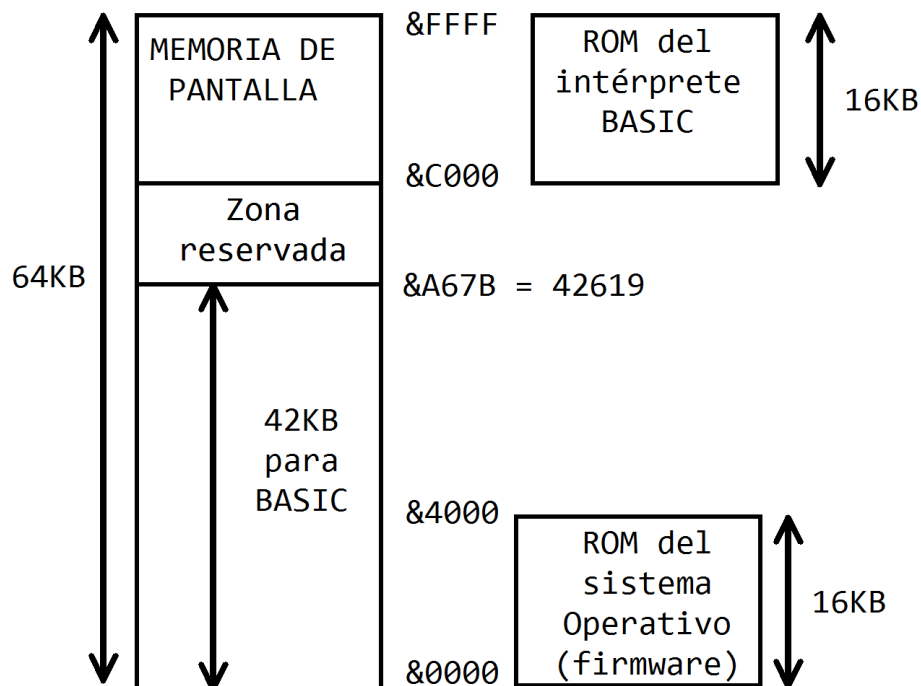
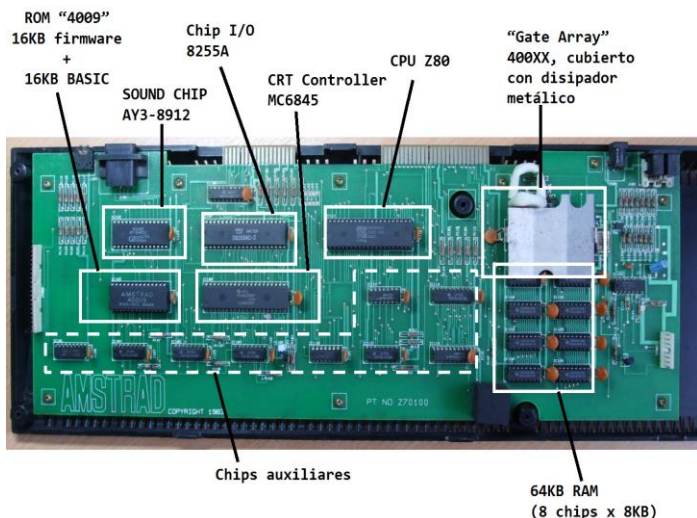
Edita y haz sonar tu música mientras  
juegas! **|MUSIC, |MUSICOFF**



**Y mucho mas,  
con 8BP!!!!**



# 1 | Introducción a 8BP



## AMSTRAD CPC464 MAPA DE MEMORIA de 8BP

```

;
; &FFFF +-----
; | pantalla + 8 segmentos ocultos de 48bytes cada uno
; &C000 +-----
; | system (simbolos redefinibles, etc)
; 42619 +-----
; | banco de 40 estrellas (desde 42540 hasta 42619 = 80bytes)
; 42540 +-----
; | map layout de caracteres (25x20 =500 bytes)
; | y mapa del mundo (hasta 82 elementos caben en 500 bytes)
; | ambas cosas se almacenan en la misma zona de memoria
; | porque o usas una o usas otra
; 42040 +-----
; | sprites (hasta 8.5KB para dibujos.
; | dispones de 8540 bytes si no hay secuencias ni rutas)
; +-----
; | definiciones de rutas (de longitud variable cada una)
; +-----
; | secuencias de animacion de 8 frames (16 bytes cada una)
; | y grupos de secuencias de animacion (macrosecuencias)
; 33500 +-----
; | canciones
; | (1.25 KB para musica editada con WYZtracker 2.0.1.0)
; 32250 +-----
; | rutinas 8BP (6250 bytes)
; | aqui estan todas las rutinas y la tabla de sprites
; | incluye el player de musica "wyz" 2.0.1.0
; 26000 +-----
; | variables el BASIC
; | V
; |
; | ^ BASIC (texto del programa)
; | |
; 0 +-----

```

# 1 | Introducción a 8BP

## Herramientas:

 **WINAPE**  
DOT NET  
Emulador/editor/ensamblador

**WYZTracker**  
Secuenciador de  
música



SPEDIT: editor de  
sprites (viene con  
8BP)



RGAS: otro editor  
de sprites



CPCDiskXP: editor de discos

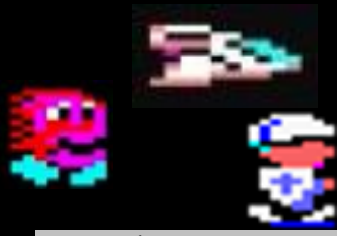


CPC basic: un compilador de  
locomotive basic (no 100%  
compatible)



Convimg: editor de  
pantallas de carga





# AGENDA



1 | Introducción a 8BP

2 | *lógicas masivas*

3 | crear un proyecto

4 | sprites

5 | layout (tile map)

6 | scroll

7 | musica

8 | recomendaciones

## 2| lógicas masivas

*Ciclo de juego: conjunto de instrucciones que se ejecutan por cada fotograma*

### **INICIO**

Inicialización de variables globales: vidas, etc  
Espera a que el usuario pulse tecla de comienzo de juego  
GOSUB pantalla1  
GOSUB pantalla 2  
...  
GOSUB pantalla N  
GOTO INICIO

### **Código de PANTALLA N (siendo N cualquier pantalla)**

Inicialización de coordenadas de enemigos y personaje  
Pintado de la pantalla (layout), si procede

### **BUCLE PRINCIPAL (ciclo del juego en esta pantalla)**

Lógica de personaje : Lectura de teclado y actualización de coordenadas del personaje y si procede, actualización de su secuencia de animación

Ejecución de lógica de enemigo 1

Ejecución de lógica de enemigo 2

...

Ejecución de lógica de enemigo n

Impresión de todos los sprites

Condición de salida de esta pantalla (IF ... THEN RETURN)

GOTO BUCLE PRINCIPAL

## 2| lógicas masivas

Es reducir la complejidad computacional de orden  $N$  a orden 1, con astucia, imponiendo restricciones que no sean perceptibles, aparte del uso de comandos que afectan a varios sprites.

### Escuadrones:

|MOVERALL,dy,dx en lugar de |MOVER  
|AUTOALL en lugar de |AUTO  
|COLSPALL en lugar de |COLSP  
|ROUTEALL

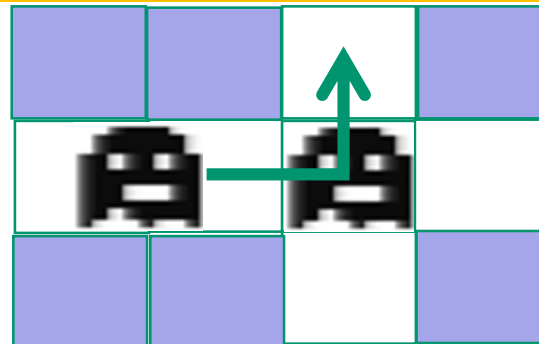
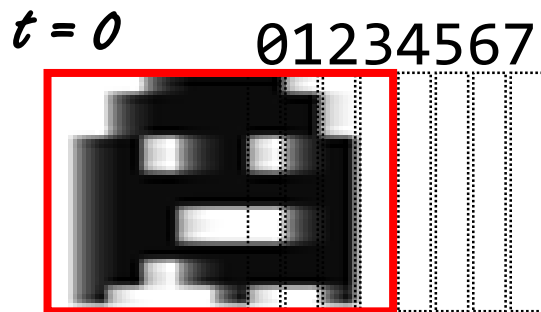


### Ejecución de menos comprobaciones

- En cada ciclo de juego ejecutar un subconjunto de comprobaciones
- Todas las comprobaciones tardarán varios frames pero no importa

### Ejecución de una sola lógica

- En cada ciclo de juego ejecutar la “Inteligencia” de un solo enemigo
- En juegos de laberintos podemos adaptar el mapa a la inteligencia



$t = 5$



# AGENDA



1 | Introducción a 8BP

2 | lógicas masivas

3 | *crear un proyecto*

4 | sprites

5 | layout (tile map)

6 | scroll

7 | música

8 | recomendaciones

### 3 | crear un proyecto

NIBIRU\_V26

ASM

basic

dsk

music

output\_spedit

tape

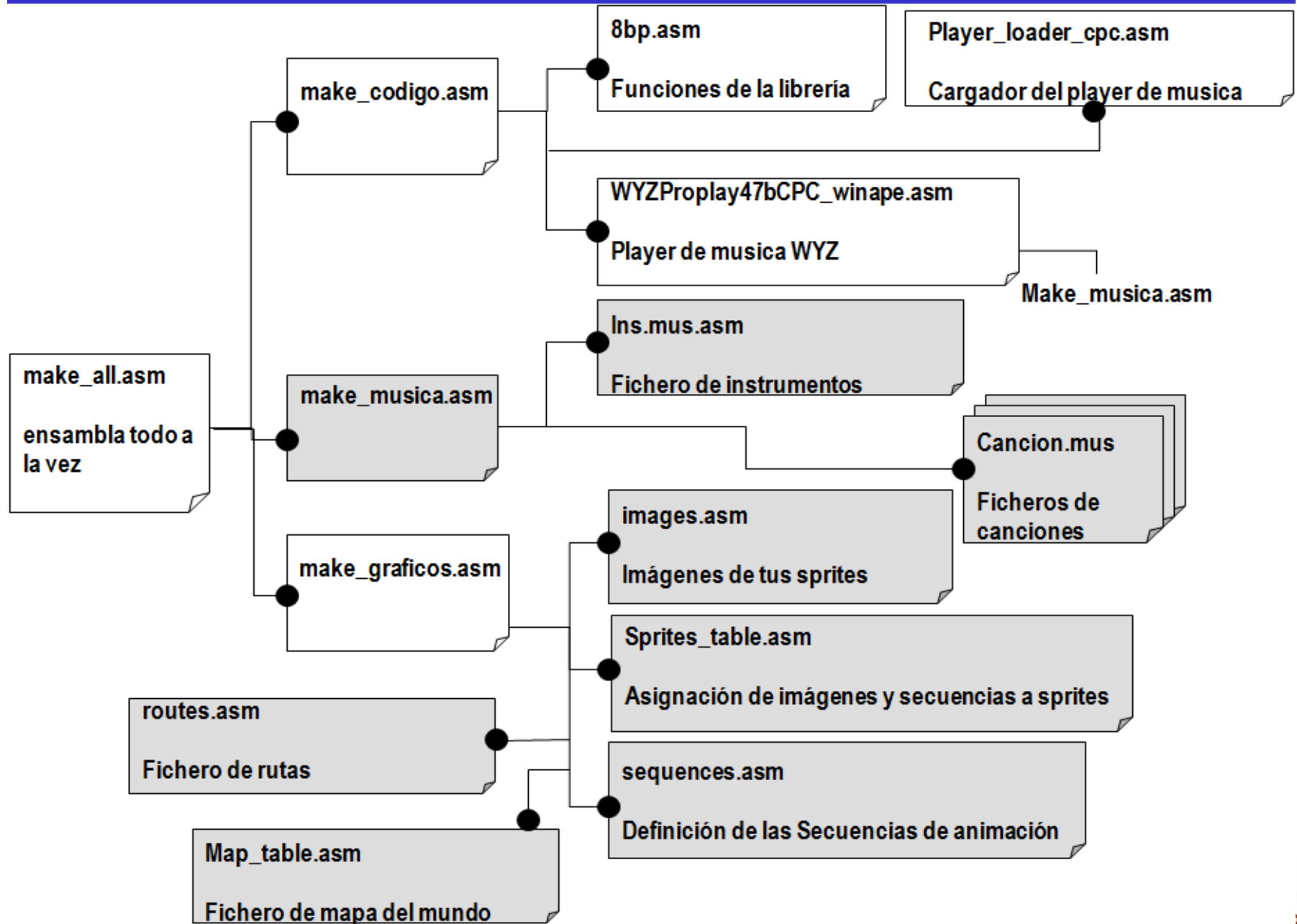
No tienes que saber ensamblador  
pero necesitas saber lo que significa  
“ensamblar”



En windows tienes muchos ficheros pero tras  
ensamblar, sólo tendras que salvar 5 ficheros en el  
Amstrad



### 3| crear un proyecto



### 3| crear un proyecto



Tu juego en 5 ficheros:

- Loader.bas
- Imagen.scr
- 8bp.lib
- Music.bin
- Sprites.bin
- Maps.bin
- tujuego.bas



Salva todo en orden!

#### Loader.bas

```
10 MEMORY 25999
15 LOAD "!imagen.scr",&C000
20 LOAD "!8bp.lib"
30 LOAD "!music.bin"
40 LOAD "!sprites.bin"
45 LOAD "!maps.bin": REM esto solo si tu juego tiene mapas
50 RUN "!tujuego.bas"
```

### 3| crear un proyecto

#### AMSTRAD CPC464 MAPA DE MEMORIA de 8BP

```
;
; &FFFF +-----
; | pantalla + 8 segmentos ocultos de 48bytes cada uno
; &C000 +-----
; | system (simbolos redefinibles, etc)
; 42619 +-----
; | banco de 40 estrellas (desde 42540 hasta 42619 = 80bytes)
; 42540 +-----
; | map layout de caracteres (25x20 =500 bytes)
; | y mapa del mundo (hasta 82 elementos caben en 500 bytes)
; | ambas cosas se almacenan en la misma zona de memoria
; | porque o usas una o usas otra
; 42040 +-----
; | sprites (hasta 8.5KB para dibujos.
; | dispones de 8540 bytes si no hay secuencias ni rutas)
; +-----
; | definiciones de rutas (de longitud variable cada una)
; +-----
; | secuencias de animacion de 8 frames (16 bytes cada una)
; | y grupos de secuencias de a
; 33500 +-----
; | canciones
; | (1.25 KB para musica ed
; 32250 +-----
; | rutinas 8BP (6250 bytes)
; | aqui estan todas las rut
; | incluye el player de
; 26000 +-----
; | variables el BASIC
; | V
;
; ^ BASIC (texto del program
;
; 0 +-----
```



Save "loader.bas"

Save "imagen.scr",b,16000,16384 : rem por ejemplo

Save "8bp.lib",b,26000,6250

Save "Music.bin",b,32250,1250

Save "sprites.bin",b,33500,8540

Save "maps.bin",b,25000,1000

Save "tujuego.bas"



# AGENDA



1 | Introducción a 8BP

2 | lógicas masivas

3 | crear un proyecto

4 | **sprites**

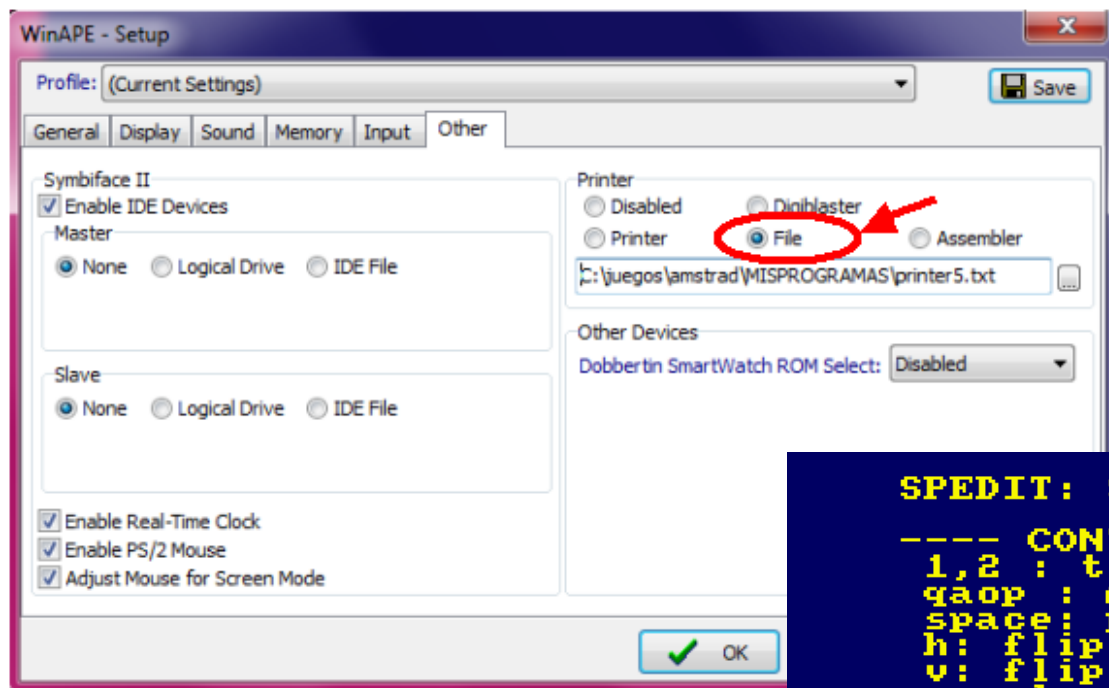
5 | layout (tile map)

6 | scroll

7 | música

8 | recomendaciones

# 4 | sprites



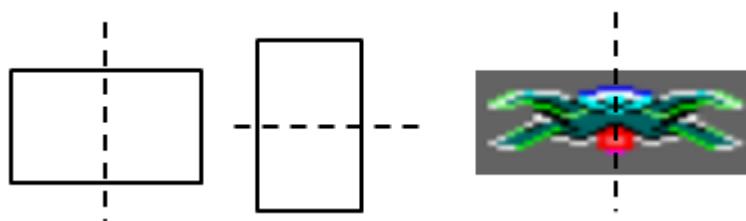
## SPEDIT: Sprite Editor v2.0

### ---- CONTROLES EDICION----

```
1,2 : tinta -/+  
qaop : mueve cursor pixel  
space: pinta  
h: flip horizontal  
v: flip vertical  
c: clear sprite  
b: imprime bytes (asm) en printer  
i: imprime paleta en printer  
r: reload 24000  
z, x: cambia color de tinta  
t: RESET
```

Antes de empezar puedes cargar un sprite ensamblandolo en la direccion 24000. No ensambles ancho y alto, solo los bytes del dibujo.

La paleta la puedes cambiar pero debera ser la misma en tu programa.  
Selecciona paleta (1,2,3,4)  
1:default 2:custom >=3:overwrite  
? ■





## 4| sprites

Los sprites los guardamos en *images.asm*

**SPEDIT:** Para hacer reload y seguir editándolos debemos comentar ancho y alto y ensamblar en la dirección 24000

```
org 24000

;----- BEGIN SPRITE -----
SOLDADO_R0
;db 6 ; ancho  OJO comentamos esta linea
;db 24 ; alto   OJO comentamos esta linea
db 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 48 , 48 , 0 , 0
db 0 , 16 , 56 , 48 , 32 , 0
db 0 , 52 , 48 , 48 , 48 , 0
db 0 , 52 , 48 , 48 , 48 , 0
db 0 , 52 , 48 , 48 , 48 , 0
db 0 , 12 , 36 , 112 , 240 , 0
db 0 , 164 , 240 , 229 , 207 , 0
db 0 , 69 , 207 , 207 , 207 , 0
db 0 , 80 , 207 , 207 , 218 , 0
db 0 , 0 , 229 , 207 , 248 , 0
db 0 , 0 , 80 , 240 , 240 , 0
db 0 , 16 , 48 , 48 , 80 , 0
db 0 , 16 , 37 , 48 , 90 , 0
db 0 , 16 , 15 , 26 , 79 , 0
db 0 , 16 , 37 , 48 , 79 , 0
db 0 , 0 , 37 , 37 , 0 , 0
db 0 , 0 , 48 , 37 , 0 , 0
db 0 , 0 , 16 , 15 , 0 , 0
db 0 , 0 , 32 , 16 , 32 , 0
db 0 , 0 , 48 , 16 , 48 , 0
db 0 , 0 , 60 , 60 , 60 , 0
db 0 , 0 , 0 , 0 , 0 , 0
;----- END SPRITE -----
```



Assembler Symbols		
Filter:		
soldado_12	8678	✓
soldado_r0	839E	✓
soldado_r1	8430	✓
soldado_r2	84C2	✓
song	7CF8	✓
song_0	7F3C	✓
song_0_end	802C	✗
.	----	.

# 4| sprites con sobreescritura

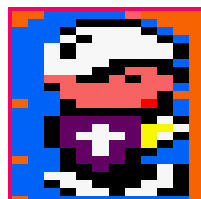


```
2300 REM ----- PALETA sprites transparentes MODE 0-----
2301 INK 0,11: REM azul claro
2302 INK 1,15: REM naranja
2303 INK 2,0 : REM negro
2304 INK 3,0:
2305 INK 4,26: REM blanco
2306 INK 5,26:
2307 INK 6,6: REM rojo
2308 INK 7,6:
2309 INK 8,18: REM verde
2310 INK 9,18:
2311 INK 10,24: REM amarillo
2312 INK 11,24 :
2313 INK 12,4: REM magenta
2314 INK 13,4 :
2315 INK 14,16 : REM naranja
2316 INK 15,16:
2317 AMARILLO=10
2420 RETURN
```



code	significado
0000	Color 1 de fondo. Si un sprite lo usa y le activas el flag de sobreescritura, significa "transparencia", es decir, imprimir restableciendo el fondo
0001	Color 2 de fondo. Si un sprite lo usa y le activas el flag de sobreescritura, deja de significar un color para significar "no imprimir". Se respeta lo que haya en ese pixel, por ejemplo, un pixel coloreado por un sprite anteriormente impreso con el que nos estamos solapando.
0010	color 1 de sprite
0011	
0100	color 2 de sprite
0101	

code	significado
0110	color 3 de sprite
0111	
1000	color 4 de sprite
1001	
1010	color 5 de sprite
1011	
1100	color 6 de sprite
1101	
1110	color 7 de sprite
1111	



9 colores en total:

- 2 de fondo
- 7 para sprites (en realidad 8 pero uno -000- significa transparencia)
- Los elementos ornamentales pueden usar los 9.

## 4 | sprites con sobreescritura


000	1
-----	---


 =  Color de fondo


110	0
-----	---


 =  Color de sprite

Paleta ejemplo


0000 = 

0001 = 


1100 = 

1101 = 

*Cuando el sprite se imprime:*

Fondo OR sprite = 1101 = 

*Cuando el sprite se marcha:*

Pixel OR 0001 = 0001 = 

*El fondo nunca fue destruido, estaba “escondido” en el sprite*

# 4| sprites

	1byte	2 bytes	2 bytes	1byte	1byte	1byte	1byte	2 bytes	1byte
sprite	status	coordy	coordx	vy	vx	seq	frame	imagen	ruta
0	27000	27001	27003	27005	27006	27007	27008	27009	27015
1	27016	27017	27019	27021	27022	27023	27024	27025	27031
2	27032	27033	27035	27037	27038	27039	27040	27041	27047
3	27048	27049	27051	27053	27054	27055	27056	27057	27063
4	27064	27065	27067	27069	27070	27071	27072	27073	27079
5	27080	27081	27083	27085	27086	27087	27088	27089	27095
6	27096	27097	27099	27101	27102	27103	27104	27105	27111

Byte de estado indica para cada sprite sus flags activos

7	6	5	4	3	2	1	0
ROUTEALL	Sobre-	COLSPALL	MOVERALL	AUTOALL	ANIMALL	COLSP	PRINTSPALL
lo ruta	escritura	collider	lo mueve	lo mueve	lo anima	collided	lo imprime

13	27208	27209	27211	27213	27214	27215	27216	27217	27223
14	27224	27225	27227	27229	27230	27231	27232	27233	27239
15	27240	27241	27243	27245	27246	27247	27248	27249	27255
16	27256	27257	27259	27261	27262	27263	27264	27265	27271
17	27272	27273	27275	27277	27278	27279	27280	27281	27287
18	27288	27289	27291	27293	27294	27295	27296	27297	27303
19	27304	27305	27307	27309	27310	27311	27312	27313	27319
20	27320	27321	27323	27325	27326	27327	27328	27329	27335
21	27336	27337	27339	27341	27342	27343	27344	27345	27351
22	27352	27353	27355	27357	27358	27359	27360	27361	27367
23	27368	27369	27371	27373	27374	27375	27376	27377	27383
24	27384	27385	27387	27389	27390	27391	27392	27393	27399
25	27400	27401	27403	27405	27406	27407	27408	27409	27415
26	27416	27417	27419	27421	27422	27423	27424	27425	27431
27	27432	27433	27435	27437	27438	27439	27440	27441	27447
28	27448	27449	27451	27453	27454	27455	27456	27457	27463
29	27464	27465	27467	27469	27470	27471	27472	27473	27479
30	27480	27481	27483	27485	27486	27487	27488	27489	27495
31	27496	27497	27499	27501	27502	27503	27504	27505	27511

## 4| sprites

Modificar parámetros de sprites : |SETUPSP

|SETUPSP, <id\_sprite>, <param\_number>, <valor>

Ejemplo

|SETUPSP, 3, 7, 2

- param\_number= 0 → cambia el status (ocupa 1 byte)
- param\_number= 5 → cambia Vy (ocupa 1byte, valor en lineas verticales).  
También se puede modificar Vx a la vez si lo añadimos al fnal como parámetro
- param\_number= 6 → cambia Vx (ocupa 1byte, valor en bytes horizontales)
- param\_number= 7 → cambia secuencia (ocupa 1byte, toma valores 0..31)
- param\_number= 8 → cambia frame\_id (ocupa 1byte, toma valores 0..7)
- param\_number= 9 → cambia dir imagen (ocupa 2bytes)
- param\_number= 15 → cambia la ruta (ocupa 1bytes)

clipping de sprites : |SETLIMITS, xmin,xmax,ymin,ymax

Ejemplo |SETLIMITS,0,80,0,200



## 4| sprites

*Impresión de un solo sprite*

`|PRINTSP,12`

*Impresión ordenada de todos los sprites*

`|PRINTSPALL,31,0,0`

*Impresión sincronizada y ordenada de 8 sprites*

`|PRINTSPALL,7,0,1`

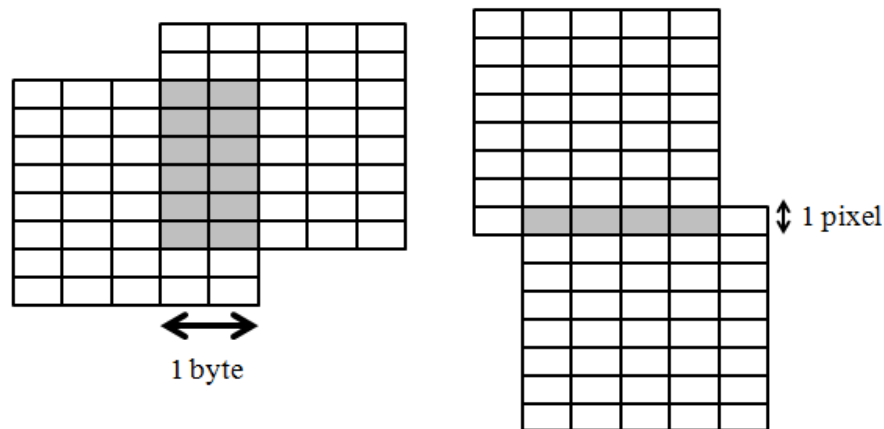
*Impresión secuencial y animada*

`|PRINTSPALL,0,1,0`



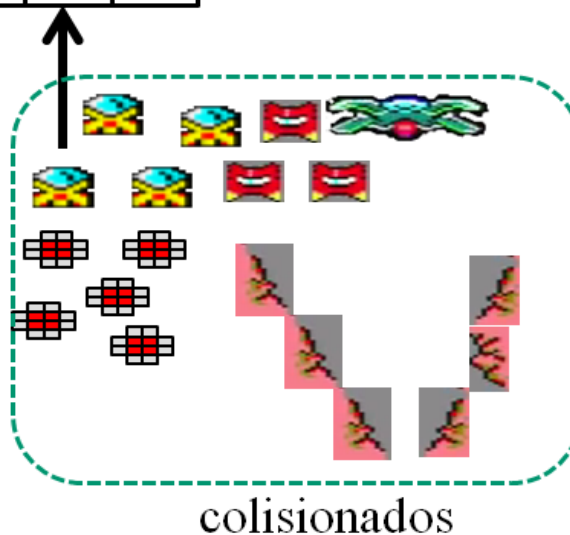
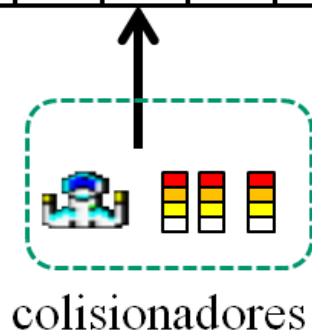
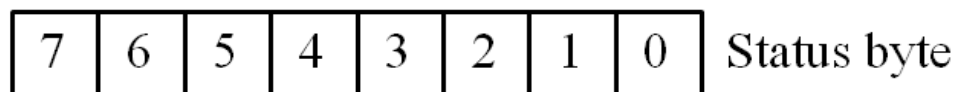
*Aquí el algoritmo de burbuja es el mejor!*

## 4 | sprites: colisiones



|COLSP, <sprite>  
|COLSPALL  
|COLAY

|COLSPALL, @colisionador%, @colisionado%



COLSP 32, ini, fin  
COLSP,33,@colisionado  
COLSP,34,dy,dx

## 4| sprites: secuencias y macrosecuencias

Sequences.asm

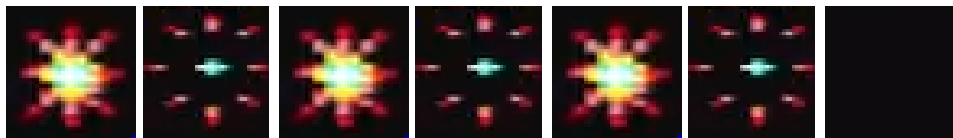


```
dw MONTOYA_R0,MONTOYA_R1,MONTOYA_R2,MONTOYA_R1,0,0,0,0
```

el equivalente en BASIC usando la librería 8BP es:

```
SETUPSQ, 1, &926c,&92FE,&9390 ,&02fe ,0,0,0,0
```

Secuencias de muerte acaban en 1



Macrosecuencias: una secuencia de secuencias. Se asignan automáticamente en función de la Vy,Vx del sprite



Macrosecuencias comienzan en 32

## 4| sprites: secuencias y macrosecuencias

```
org 33500;
;=====
; secuencias de animacion
;=====
; cada secuencia contiene las direcciones de frames de animacion
; cíclica.cada secuencia son 8 direcciones de memoria de imagen
; numero par porque las animaciones suelen ser un numero par
; un cero significa fin de secuencia, aunque siempre se gastan 8 words por
; cada secuencia
; al encontrar un cero se comienza de nuevo.
; si no hay cero, tras el frame 8 se comienza de nuevo
; la secuencia cero es que no hay secuencia.
; empezamos desde la secuencia 1

;-----secuencias de animacion -----
_SEQUENCES_LIST
dw NAVE,0,0,0,0,0,0,0;1
dw JOE1,JOE2,0,0,0,0,0,0;2 UP JOE
dw JOE7,JOE8,0,0,0,0,0,0;3 DW JOE
dw JOE3,JOE4,0,0,0,0,0,0;4 R JOE
dw JOE5,JOE6,0,0,0,0,0,0;5 L JOE

_MACRO_SEQUENCES
;-----MACRO SECUENCIAS -----
; son grupos de secuencias, una para cada dirección. el significado es:
; still, left, right, up, up-left, up-right, down, down-left, down-right
; se numeran desde 32 en adelante
db 0,5,4,2,5,4,3,5,4;la secuencia 32 contiene las secuencias del soldado Joe
```

*Cada secuencia tiene un identificador, que es su posición en esta lista*

## 4| sprites: rutas

Cada ruta (en el fichero `routes.asm`) posee un número indeterminado de segmentos y cada segmento tiene tres parámetros:

- Cuantos pasos vamos a dar en ese segmento
- Qué velocidad  $V_y$  se va a mantener durante el segmento
- Qué velocidad  $V_x$  se va a mantener durante el segmento

Al final de la especificación de segmentos debemos poner un cero para indicar que la ruta se ha terminado y que el sprite debe comenzar a recorrer la ruta desde el principio

```
ROUTE1; izquierda-derecha  
;-----  
      db 10,0,-1  
      db 10,0,1  
      db 0
```

Ojo con  $V_y, V_x$ , que deben estar entre -127 y +127

Código de escape (campo "número de pasos")	Descripción	Ejemplo
255	Cambio de estado del sprite.	DB 255,3,0 Estado pasa a valor 3. El cero del final es de relleno
254	Cambio de secuencia de animación del sprite	DB 254,10,0 Se asocia la secuencia 10. El cero es de relleno
253	Cambio de imagen	DB 253 DW new_img Se asocia la imagen "new_img" que debe ser una dirección de memoria
252	Cambio de ruta	DB 252,2,0 Se asocia la ruta 2

|AUTOALL, 1 invoca internamente a |ROUTEALL



## 4 | sprites: rutas





# AGENDA



1 | Introducción a 8BP

2 | lógicas masivas

3 | crear un proyecto

4 | sprites

5 | layout (tile map)

6 | scroll

7 | música

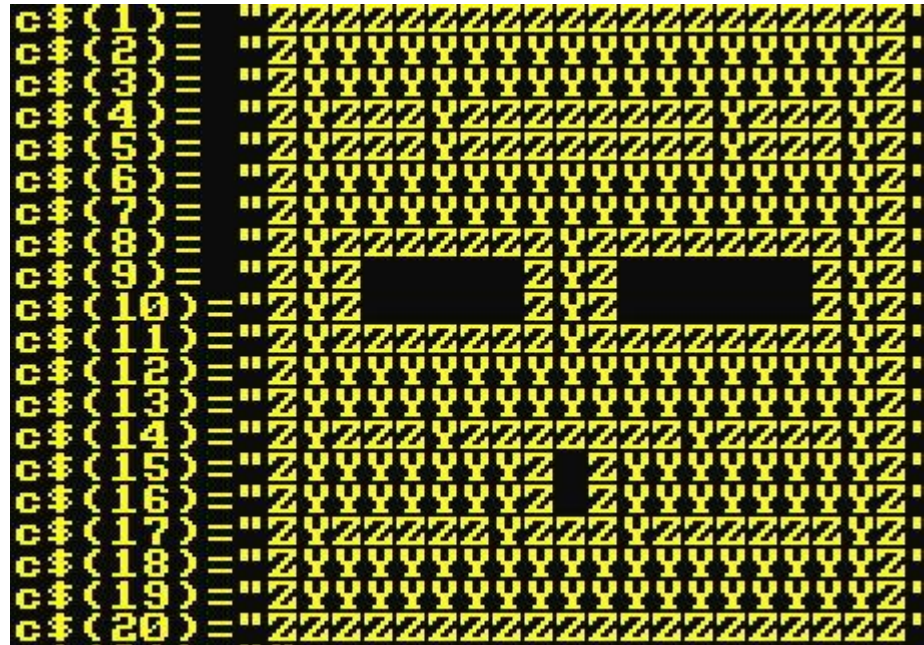
8 | recomendaciones

## 5| layout (tile map)

El mapa de tiles de 8BP ( lo llamamos layout) se basa en el mecanismo de sprites, asociando un carácter a cada spriteid

Caracter	Sprite id	Codigo ASCII
" "	NINGUNO	32
"0"	0	59
"1"	1	60
"2"	2	61
"3"	3	62
"4"	4	63
"@"	5	64
"A"	6	65
"B"	7	66
"C"	8	67
"D"	9	68
"E"	10	69
"F"	11	70
"G"	12	71
"H"	13	72
"I"	14	73
"J"	15	74
"K"	16	75
"L"	17	76
"M"	18	77
"N"	19	78
"O"	20	79
"P"	21	80
"Q"	22	81
"R"	23	82
"S"	24	83
"T"	25	84
"U"	26	85
"V"	27	86
"W"	28	87
"X"	29	88
"Y"	30	89
"Z"	31	90

|LAYOUT,<y>,<x>,@string



```
550 'rutina print layout
```

```
560 FOR i=0 TO 23:|LAYOUT,i,0,@c$(i):NEXT
```

```
570 RETURN
```

Detección de colisión con el layout:

|COLAY<umbral ASCII>, <sprite number>, @colision%

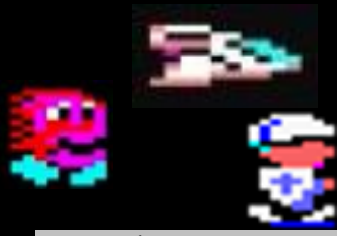
## 5| layout (tile map)



al y  
cción  
al



COEF camera ASCI, <sprite number>, <consion>



# AGENDA



1 | Introducción a 8BP

2 | lógicas masivas

3 | crear un proyecto

4 | sprites

5 | layout (tile map)

6 | scroll

7 | música

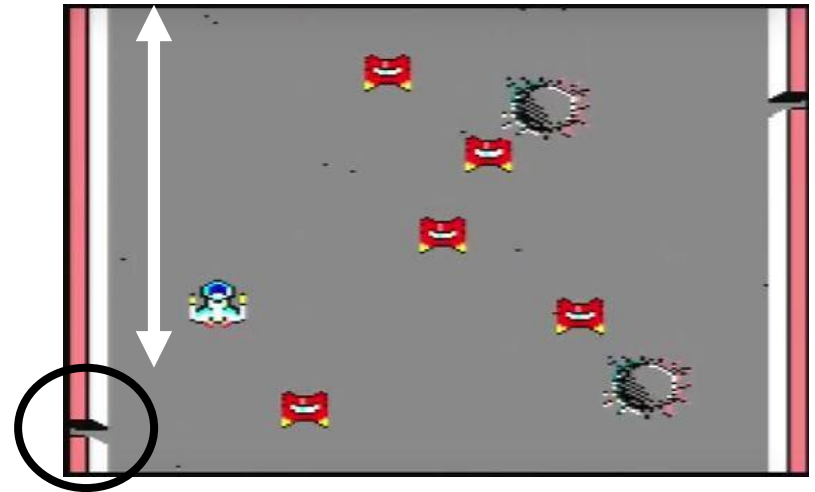
8 | recomendaciones



## 6 | scroll

Técnicas de scroll:

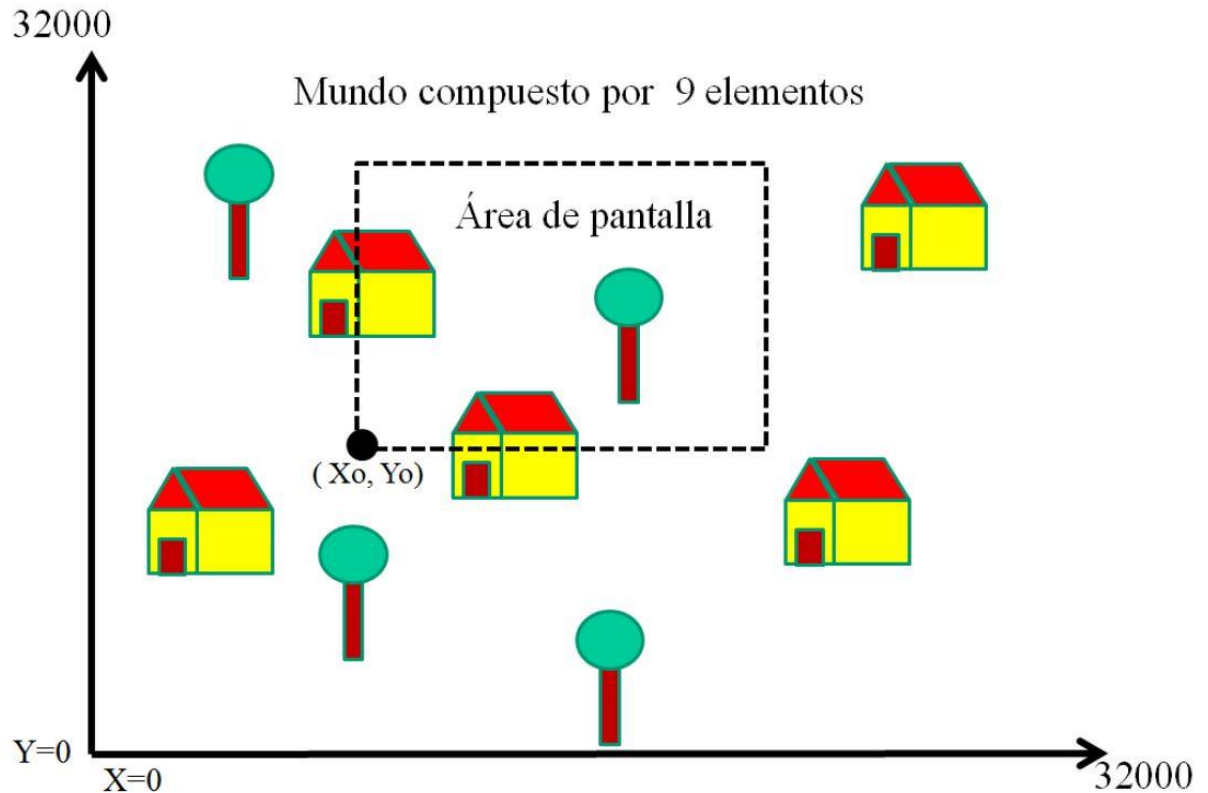
- |MOVERALL, |AUTOALL
- |STARS
- Truco de "Manchado"
- |MAP2SP
- animación |RINK





## 6 | scroll: comando |MAP2SP

Comando  
|MAP2SP, Yo, Xo



```
;MAP TABLE
```

```
; primero 3 parametros antes de la lista de "map items"
```

```
dw 50; maximo alto de un sprite por si se cuelga por arriba y ya hay que pintar parte de el
```

```
dw -40; máximo ancho de un sprite por si se cuelga por la izquierda (numero negativo)
```

```
db 64; numero de elementos del mapa.como mucho debe ser 64
```

```
; a partir de aqui comienzan los items
```

```
dw 100,10,CASA; 1
```

```
dw 50,-10,CACTUS; 2
```

```
dw 210,0,CASA; 3
```

```
dw 200,20,CACTUS; 4
```

```
dw 100,40,CASA; 5
```

```
dw 160,60,CASA; 6
```

```
dw 70,70,CASA; 7
```

```
dw 175,40,CACTUS; 8
```

```
dw 10,50,CASA; 9
```

```
dw 250,50,CASA; 10
```

```
dw 260,70,CASA; 11
```

```
dw 290,60,CACTUS; 12
```

```
dw 180,90,CASA; 13
```

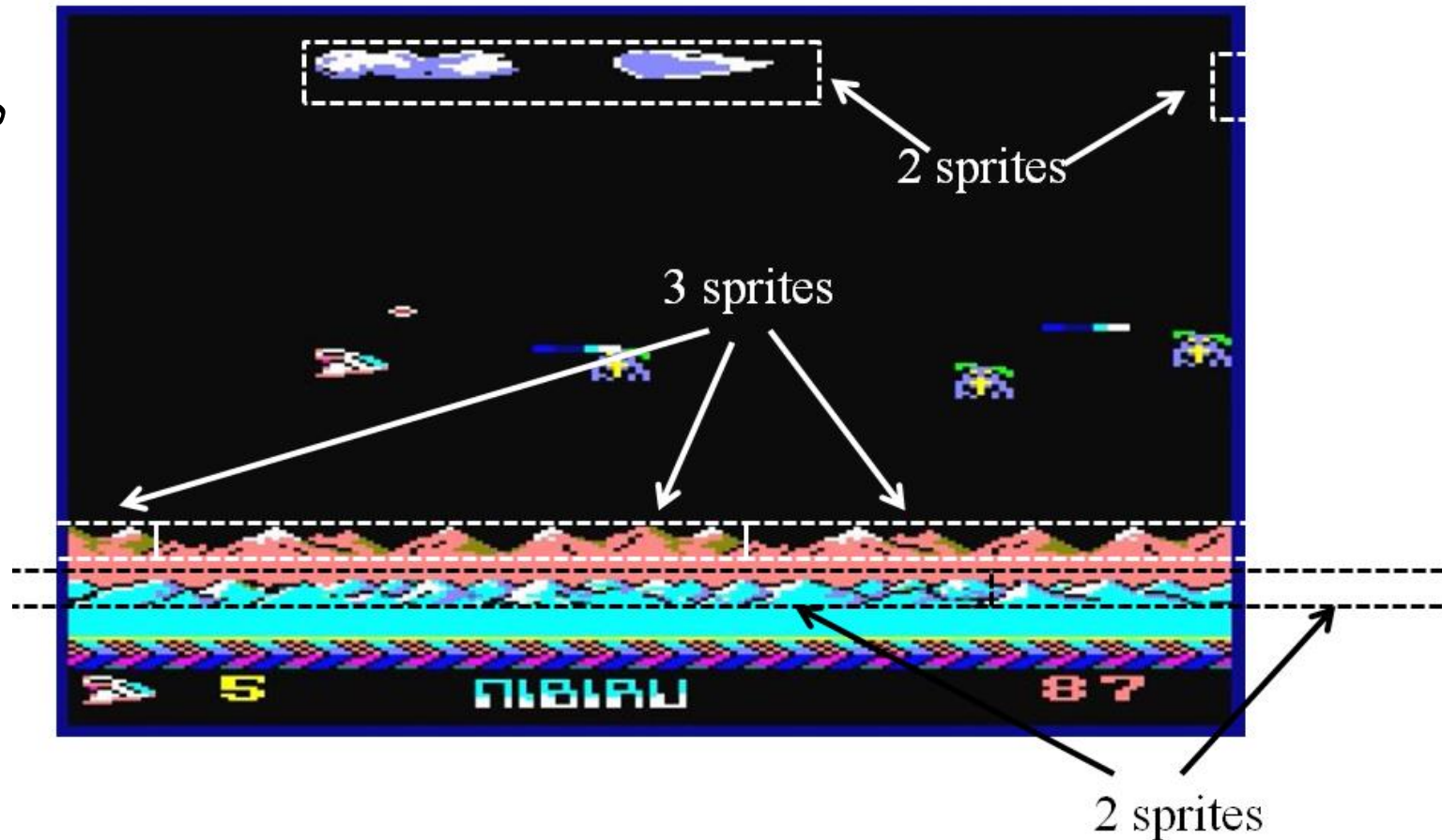
```
dw 60,100,CASA; 14
```

### Mapa del mundo

Puedes tener muchas fases y pokear el mapa al entrar en cada fase

## 6 | scroll: comando | MAP2SP

Scroll  
"Parallax"



Para las montañas usé 3 sprites normales, fuera del mapa del mundo. Les di movimiento automático en su flag de status y les hice moverse hacia la izquierda. Pero en los ciclos impares les desactivo tanto el flag de impresión como el flag de movimiento automático, de modo que solo se mueven e imprimen uno de cada dos ciclos de juego, logrando una velocidad mitad que la que lleva el agua

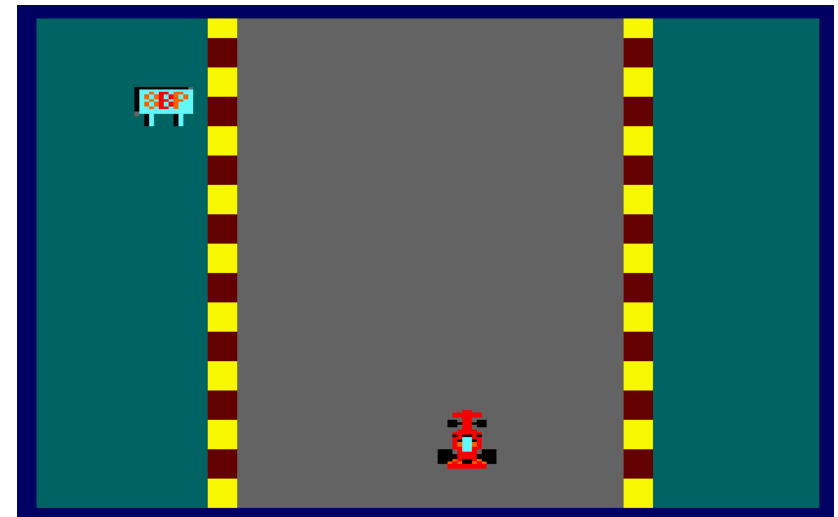
## 6 | scroll:



## 6 | scroll: comando |RINK



0							
8	9	10	11	12	13	14	15



Fotograma 1	Fotograma 2	Fotograma 3	Fotograma 4
t1			
t2			
t3			
t4			
t5			
t6			
t7			
t8			

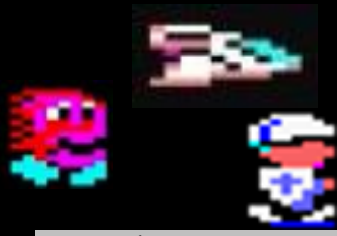
|RINK,tinta\_inicial, color1,color2, color3, color4, color4, color5, color6, color7,color 8

|RINK,tinta\_inicial, color1,color2, color3, color4

|RINK, step

## 6 | scroll: comando | RINK





# AGENDA



1 | Introducción a 8BP

2 | lógicas masivas

3 | crear un proyecto

4 | sprites

5 | layout (tile map)

6 | scroll

7 | música

8 | recomendaciones



## 7 | música (on-game)

|MUSIC,<numero\_melodía>,<velocidad>

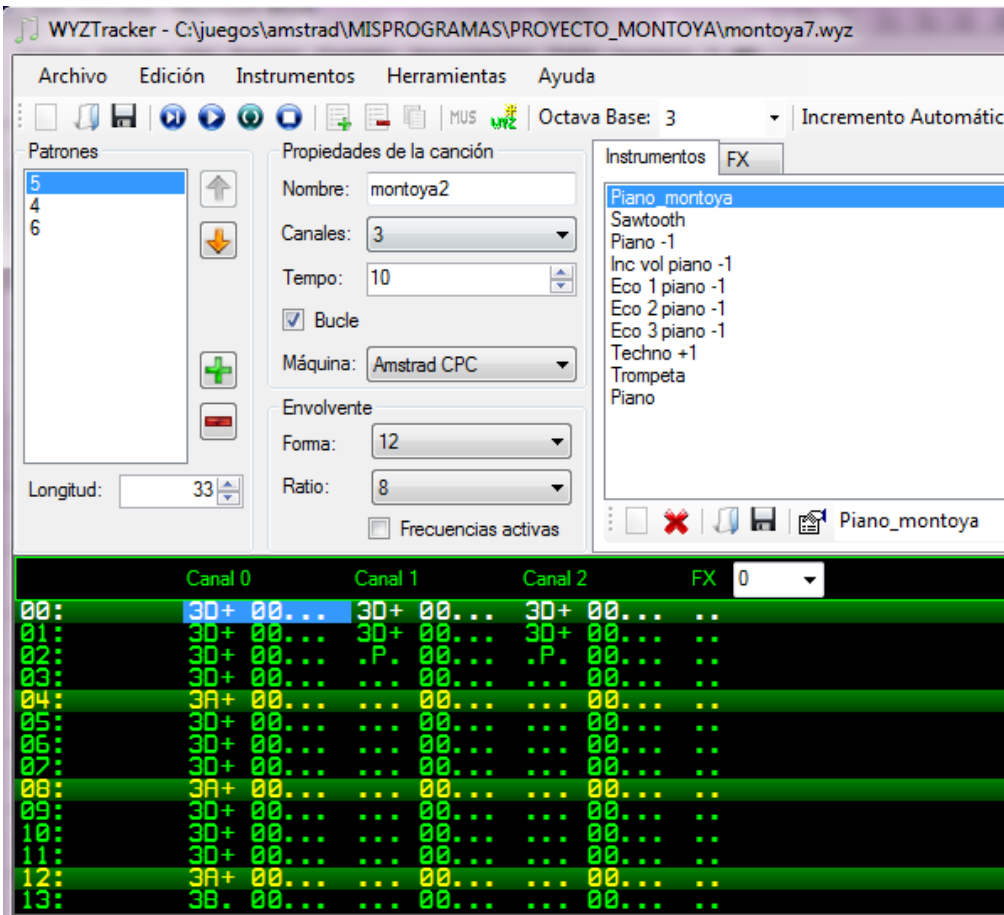
|MUSICOFF

# WYZTracker

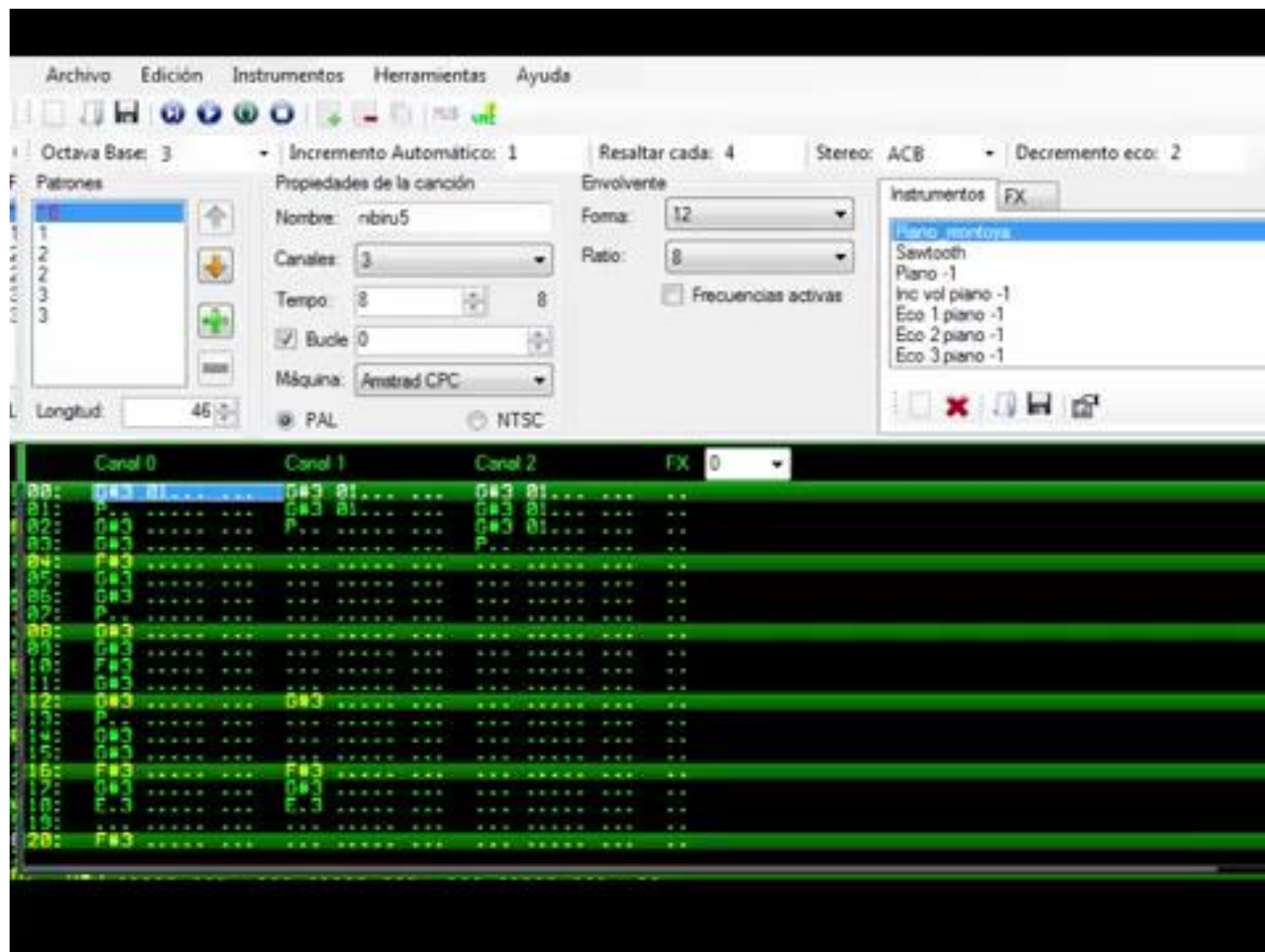
Esta herramienta es un secuenciador de música para el chip de sonido AY3-8912. Las músicas que genera se pueden exportar y dan como resultado dos archivos

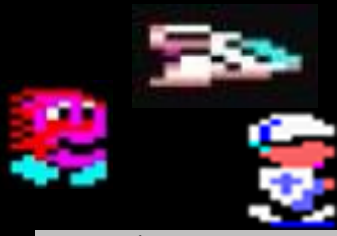
- Un archivo de instrumentos “.mus.asm” (debe ser el mismo para todas las músicas)
- Un archivo de notas musicales “.mus” (uno por cada música)

Gracias a Augusto Ruiz!



## 7 | música (on-game)





# AGENDA



1 | Introducción a 8BP

2 | lógicas masivas

3 | crear un proyecto

4 | sprites

5 | layout (tile map)

6 | scroll

7 | música

8 | recomendaciones

## 8 | recomendaciones

*Mide el tiempo que consume cada instrucción de tu programa y ante todo trata de ahorrar instrucciones en cada ciclo de juego*

```
10 MEMORY 25999
20 DEFINT a-z
30 a!= TIME
40 FOR i=1 TO 1000
50  <aqui pones un comando, por ejemplo PRINT "A">
60 NEXT
70 b!=TIME
80 PRINT (b!-a!)
900 c!=1000/((b!-a!)*1/300)
100 PRINT c, "fps"
110 d!=c!/60
120 PRINT "puedes ejecutar ",d!, "comandos por barrido (1/50 seg)"
125 rem si dejas la linea 50 vacia , tardara 0.47 milisegundos
130 PRINT "el comando tarda ";((b!-a!)/300-0.47);"milisegundos"
```

## 8 | recomendaciones

- Usa DEFINT A-Z ( y asigna en hexa si valor > 32000)
- No uses PRINT en cada ciclo
- Usa mucho GOTO, es mas rápido incluso que REM
- Evita el paso de parámetros (8BP lo permite)
- IF a > b and c > d then → IF a > b then if c > d then
- Usa POKE cuando puedas hacerlo en lugar de SETUPSP
- Si necesitas comprobar algo no lo hagas en todos los ciclos de juego. Usa lógica modular (MOD) y operaciones lógicas
- Aprovecha bien la memoria, reutiliza líneas

Técnica	Tiempo consumido
A = A+1: if A =5 then A=0: GOSUB <rutina>	2.6 ms
IF ciclo MOD 5 =0 THEN gosub rutina	1.84ms, suponiendo que ya tienes una variable llamada ciclo que se actualiza

Técnica	Tiempo consumido
If ciclo AND 15=0 then gosub rutina	1.6 ms (se ejecuta una de cada 16 veces)
If ciclo AND 1=0 then gosub rutina	1.6ms (Se ejecuta una de cada 2 veces)

Y usa lógicas masivas!!!!

# Empieza la aventura de programar!

Nibiru



Mini invaders



Anunnaki



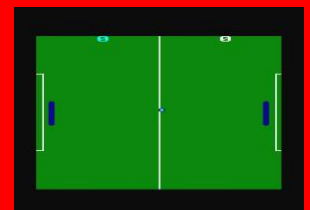
Mutante Montoya



Fresh Fruits & vegetables



Mini pong



*Todos los juegos mostrados están programados en Locomotive BASIC Y ejecutados sin necesidad de compilar*

Jose Javier Garcia Aranda

[8bitsdepoder.blogspot.com](http://8bitsdepoder.blogspot.com)

<http://github.com/jjaranda13/8bp>

