INTRODUCCIÓN A 8BP

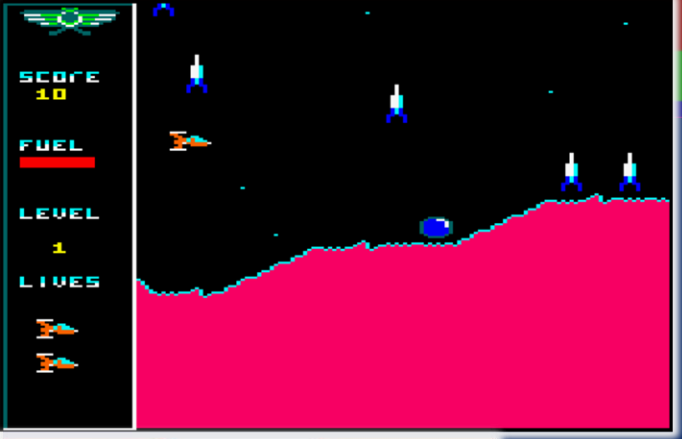
consejos y trucos

https://8bitsdepoder.blogspot.com
https://github.com/jjaranda13/8BP

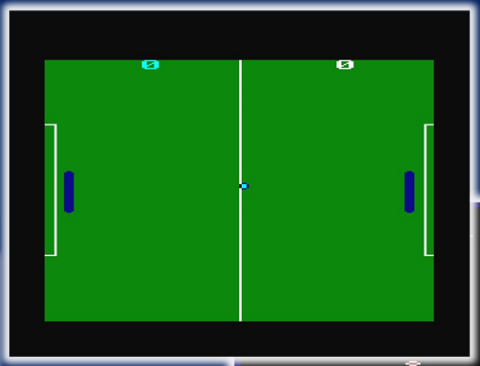
@8bitsdepoder

Agenda

1. Lo que puedes lograr con 8BP
2. Introducción a 8BP y lógicas masivas
3. Consejos y trucos con 8BP



Algunas de las cosas que puedes lograr con 8BP



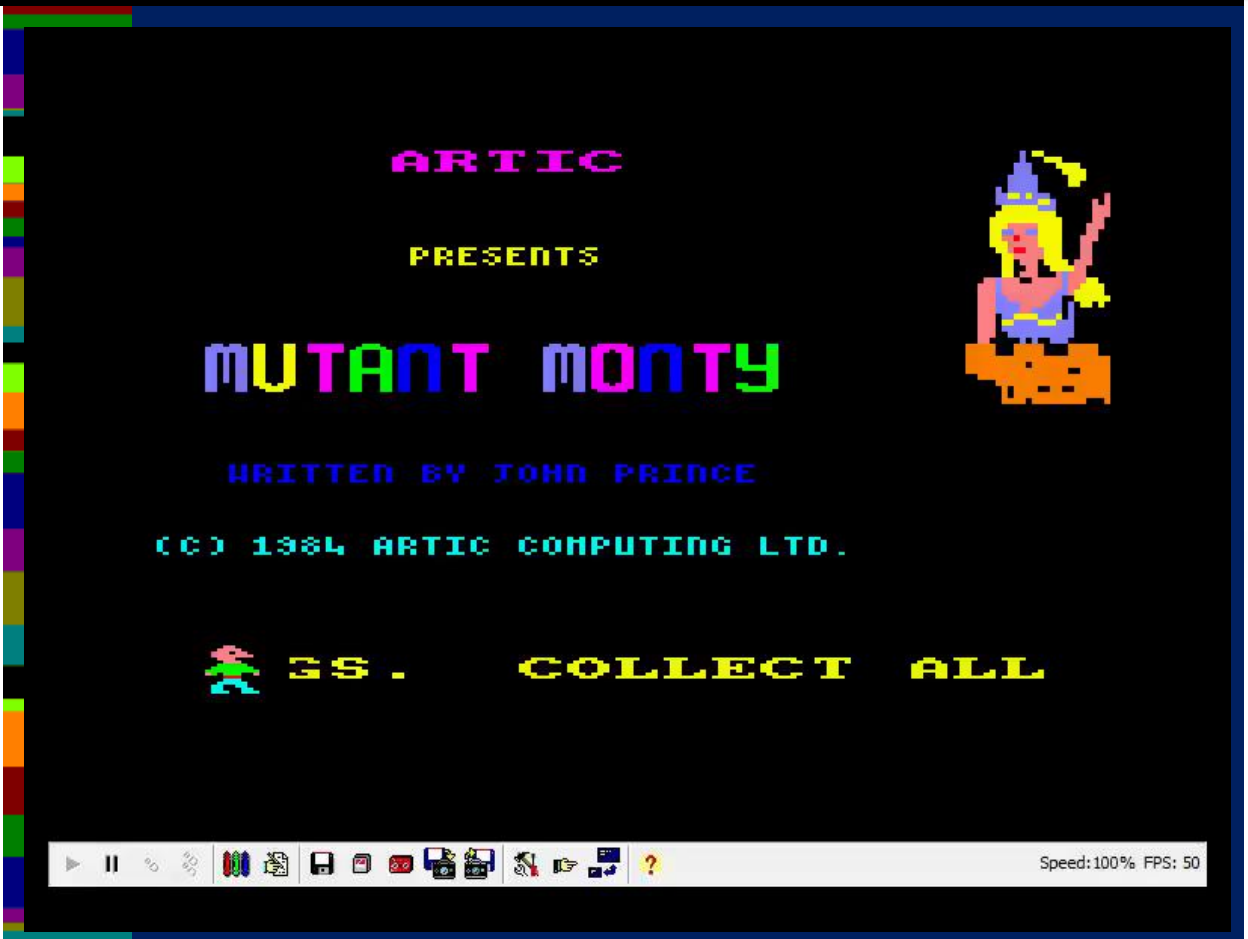
Historia de Scramble

Scramble fue creado en 1981 por Konami. Considerado entre los mejores videojuegos de todos los tiempos según Killer List of Videogames (KLOV) .

Primer juego con mecanismo de “fuel” y primer arcade multinivel


ERIDU
THE SPACE PORT
massive logics
SPONSORED BY
AUALL
HTTPS://AUAMSTRAD.ES
JJGA
2019

Mutant Monty fue creado en 1984 por John Price (Artic computing). Es un juego sencillo pero divertido, con una jugabilidad muy buena, y una música fabulosa, llegando a convertirse en un verdadero clásico. John Price también hizo otros famosos juegos como "world cup" (US gold) , "Wong loopy aundry" (Amsoft) o "indiana jones and the temple of doom"



MUTANT MONTY

(AMSTRAD)

Como casi todo el mundo, Monty tiene dos ambiciones en la vida, enriquecerse rápidamente y convertirse en un héroe. Aquí él tiene la oportunidad de conseguir ambas.

Sólo tien que atravesar 40 habitaciones recogiendo todo el oro a su paso y rescatar a la "Damisela en peligro". De lo que no es consciente Monty es de los innumerables alienígenas dispuestos a detenerles desde los temidos Parallelians hasta la bella pero rara Quantum Leapies y de que tiene que pasar a través del agujero de las moscas, del interior del acelerador de Newton y cruzar el desolado territorio del Monstruo antes de que pueda rescatar a la damisela.

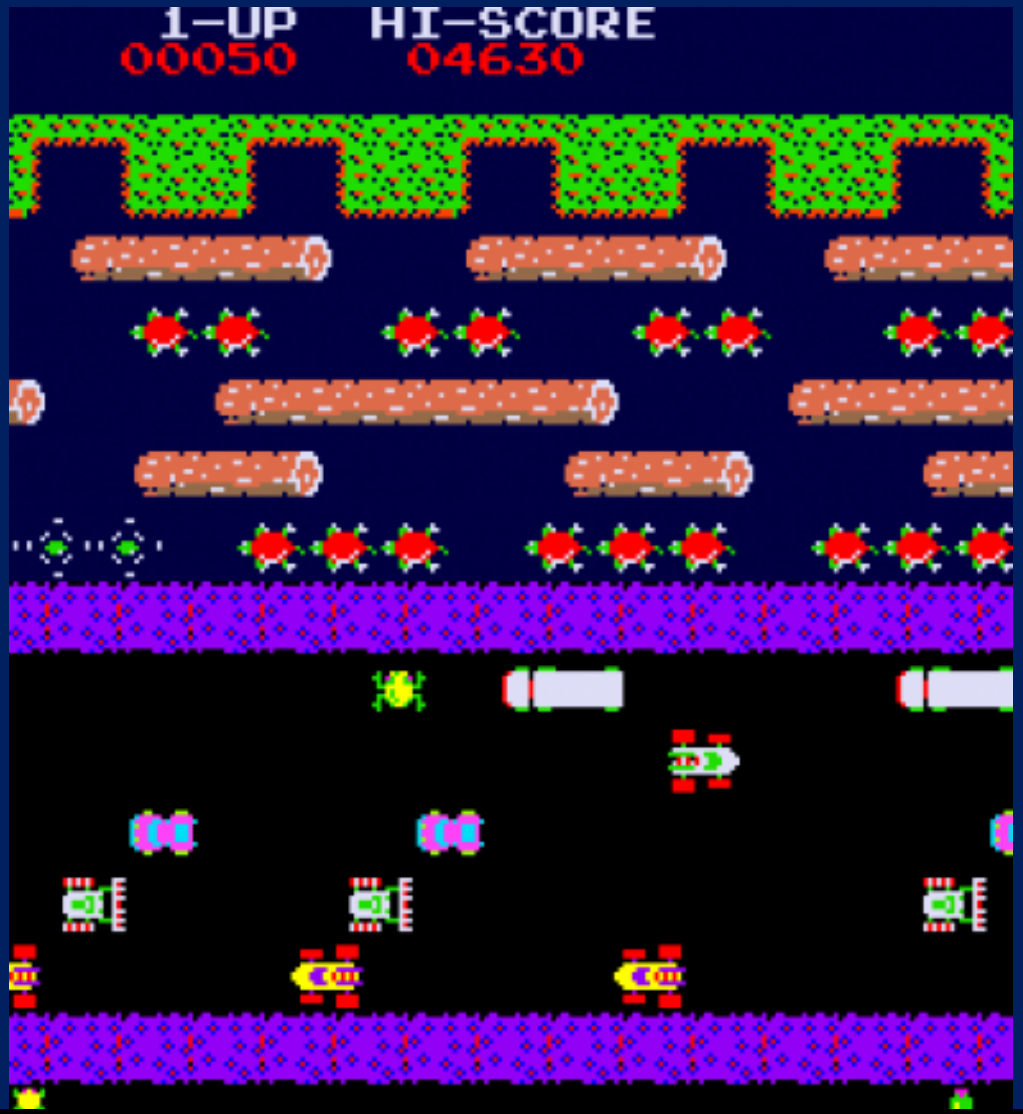
Y todo esto con sólamente 5 vidas y el reloj marcando el tiempo en su contra. ¿Debe Monty continuar, o debe en cambio detenerse a beber una copa en el bar? ¡Pobre Monty, sólo los valientes deben jugar!

TECLAS

1 = detiene el juego.
Q = mueve Monty hacia arriba.
A = mueve Monty hacia abajo.
O = mueve Monty hacia la izquierda.
P = mueve Monty hacia la derecha.

o Amsoft Joystick JY1.

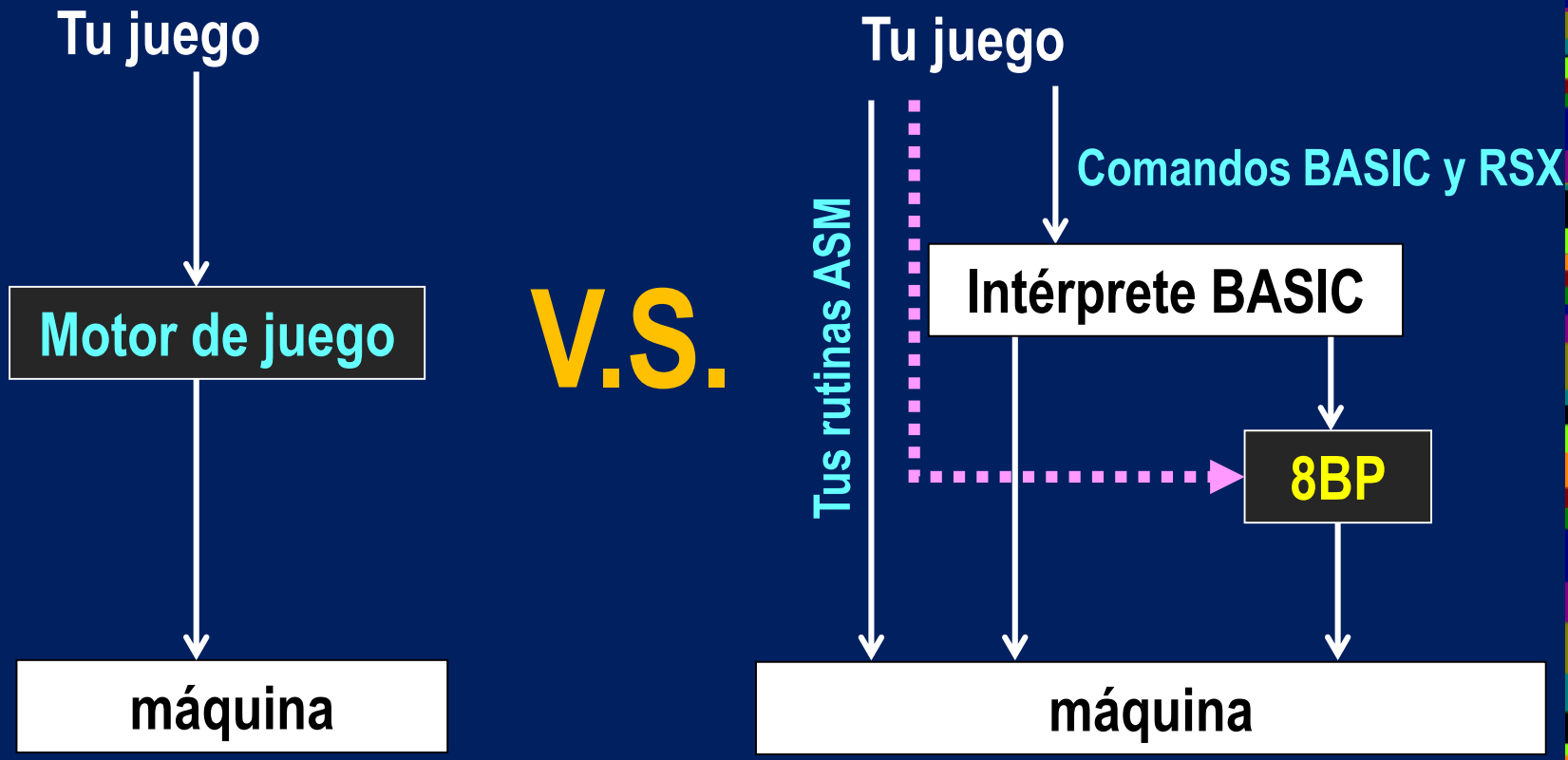
Historia de frogger



Frogger es considerado como uno de los 10 mejores videojuegos de todos los tiempos según Killer List of Videogames (KLOV)

Introducción a 8BP y lógicas masivas

8BP es una librería de rutinas útiles para videojuegos y accesibles desde BASIC mediante comandos RSX



RSX (Resident System eXtensions)

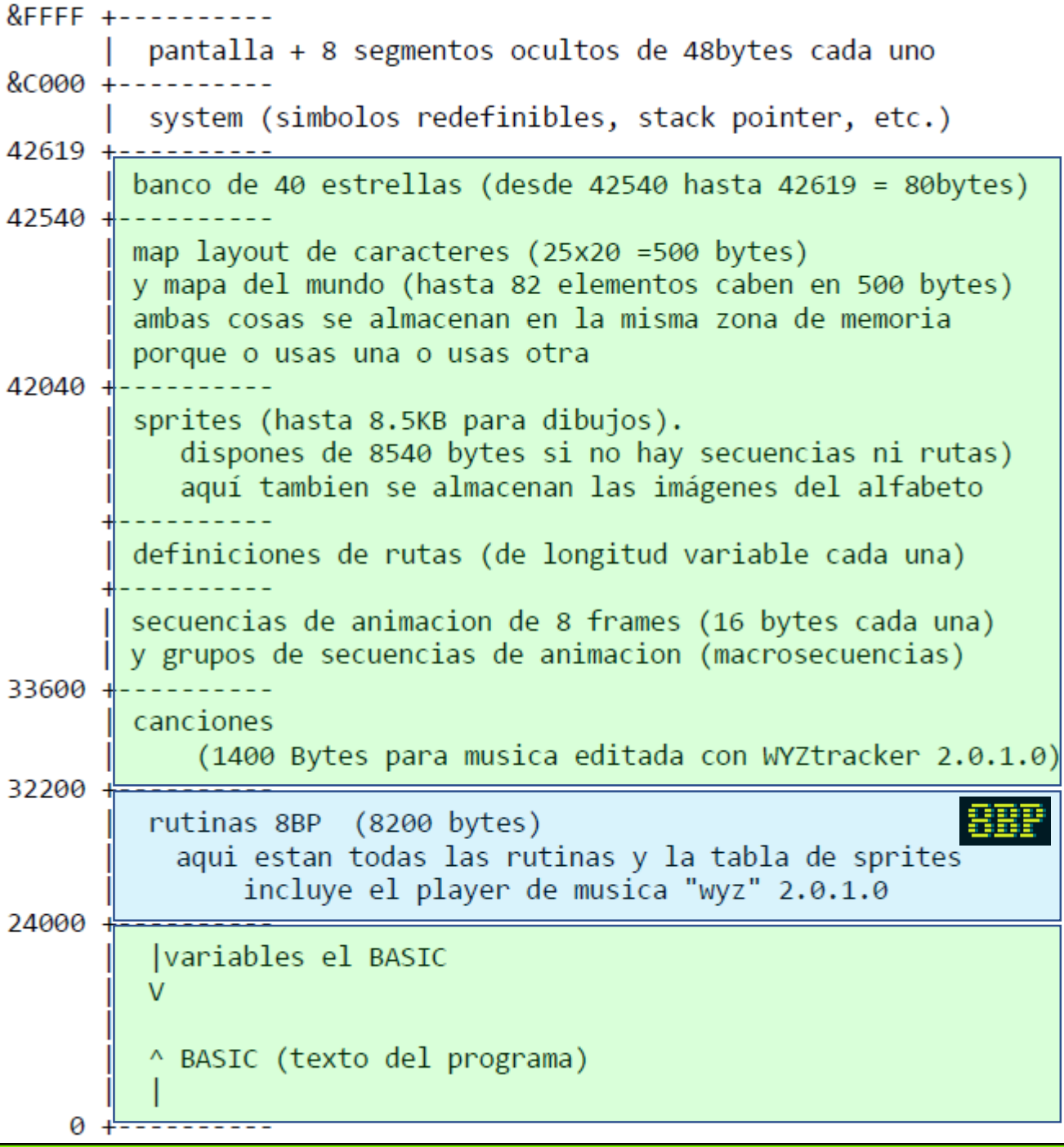
8BP Memory map

8BP Sólo ocupa 8 KB y te proporciona 27 comandos

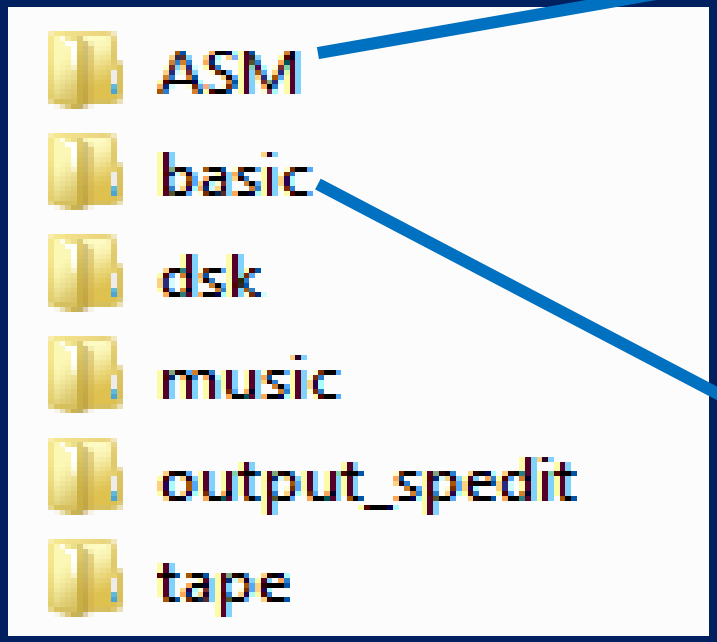
24 KB libres para BASIC

1.4 KB musica

8.5 KB sprites



Carpetas de un juego 8BP



Make_all.asm
Images_mygame.asm
Routes_my_game.asm
Secuences_my_game.asm
...

Loader.bas
tujuego.bas

Tujuego.bin incluye a 8BP y sólo lo crearás en el .dsk o en el .cdt
En total habrá 3 ficheros

capacidades

- 32 sprites con clipping (SETLIMITS), sobreescritura, ordenación y detección de colision (COLSPALL).
- Comandos para mover N sprites a la vez (MOVERALL, AUTOALL, ROUTEALL...)
- Secuencias de animacion y macrosecuencias (cualquier sprite puede cambiar su secuencia de animacion dependiendo de su Vx,Vy)
- Enrutado de sprites automatico con rutas definibles (loops, saltos,…)
- Scroll multidireccional (comandos MAP2SP y UMAP)
- Permite musica in-game basada en WYZtracker 2.0.1.0(comandos MUSIC and MUSICOFF)
- Capacidad de juegos con layout ("tile map"), con detección de colisión.
- Capacidad de animación por tintas (RINK)
- Set de Minicaracteres definibles para usar en tus juegos (PRINTAT)
- Comando STAR para efectos de estrellas, tiera, Lluvia…
- Capacidad PSEUDO-3D
- Sólo ocupa 8 KB y reserva 8.5KB para sprites y 1.4KB para musica, dejando 24 KB para lógica BASIC.

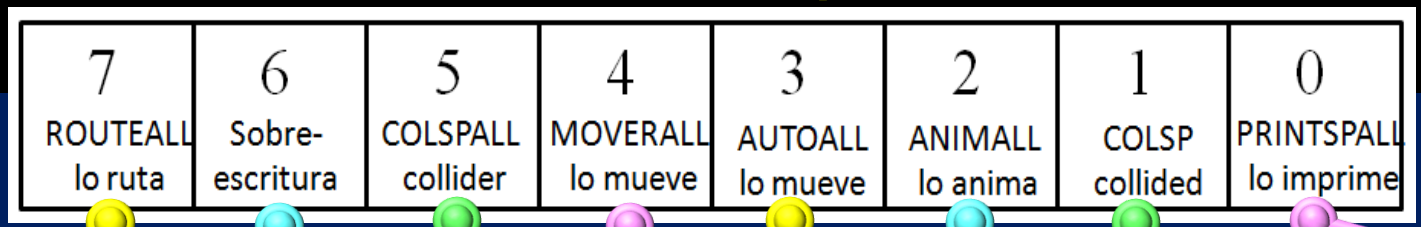
Sprites

- 8BP soporta 32 sprites de cualquier tamaño
- 16 bytes por sprite (9 parámetros)
- Comienzan en 27000
- Podemos leer con PEEK o con |PEEK
- Podemos escribir sus parámetros con POKE o con |POKE o con |SETUPSP
- El primer byte es el de estado
- Los puedes colocar con |LOCATESP

	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
7	27112	27113	27115	27117	27118	27119	27120	27121	27127
8	27128	27129	27131	27133	27134	27135	27136	27137	27143
9	27144	27145	27147	27149	27150	27151	27152	27153	27159
10	27160	27161	27163	27165	27166	27167	27168	27169	27175
11	27176	27177	27179	27181	27182	27183	27184	27185	27191
12	27192	27193	27195	27197	27198	27199	27200	27201	27207
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

Cada sprite tiene un byte de status:

SETUPSP,sp,0,status



visible

colisionable

Puede tener una secuencia de animación
ANIMALL o PRINTSPALL,1

Puede tener velocidad
AUTOALL o AUTOALL,1

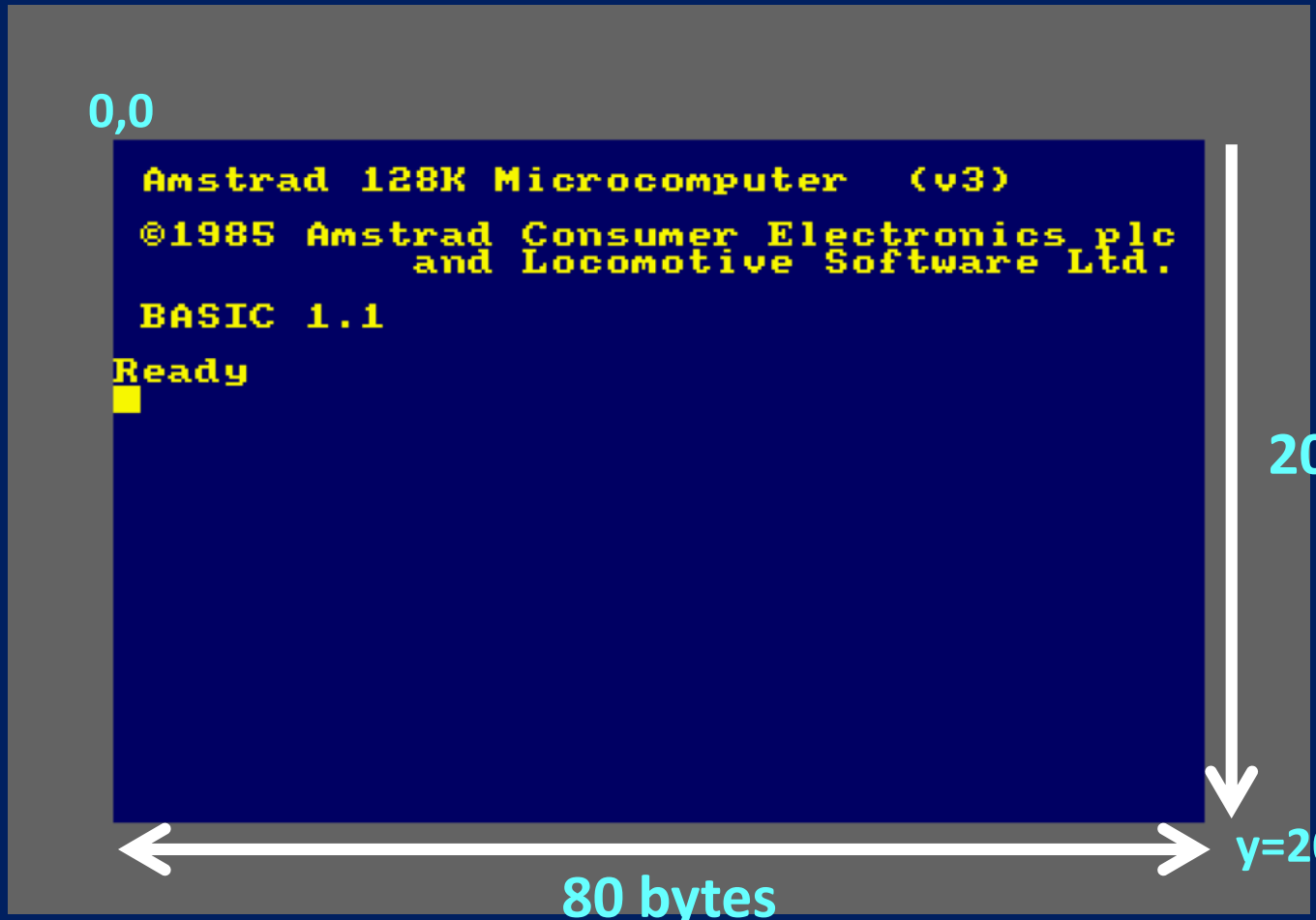
Puede moverse de forma relativa solidariamente con otros sprites
MOVERALL,dy,dx

Puede ser colisionador
COLSPALL,@collider,@collided

Puede tener sobreescritura

Puede tener una ruta asignada
SETUPSP, sp, 15, ruta

Coordenadas en 8BP



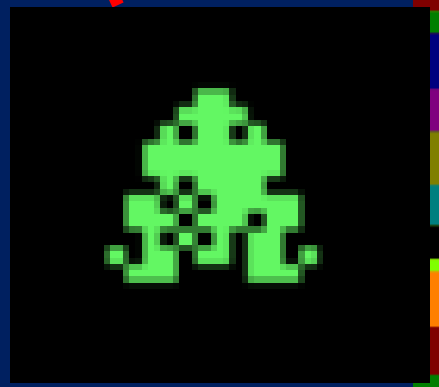
1 BYTE = 2 pixels de MODE 0

1 BYTE = 4 pixels de MODE 1

Primer ejemplo:

```
Ready
list
10 MEMORY 23999
20 CALL &6B78
30 DEFINT A-Z
40 |SETUPSP,31,0,1
50 |SETUPSP,31,9,16
55 x=40:y=100
60 |LOCATESP,31,y,x
70 |PRINTSPALL,0,0,0,0
Ready
run
Ready
```

```
10 MEMORY 23999
20 CALL &6B78
30 DEFINT A-Z
35 frogimg = 16
40 |SETUPSP,31,0,1
50 |SETUPSP,31,9,frogimg
55 x=40,y=100
60 |LOCATESP,31,y,x
70 |PRINTSPALL,0,0,0,0
```

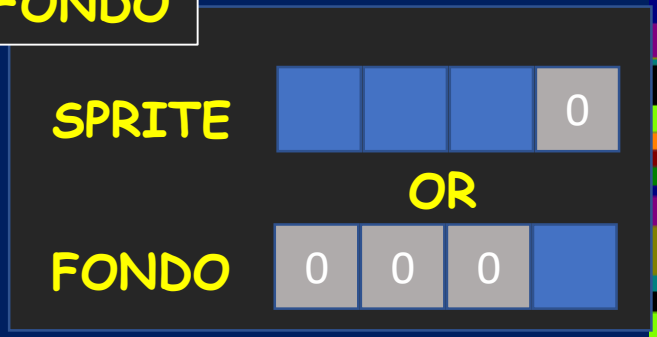


Sprites: sobreescritura

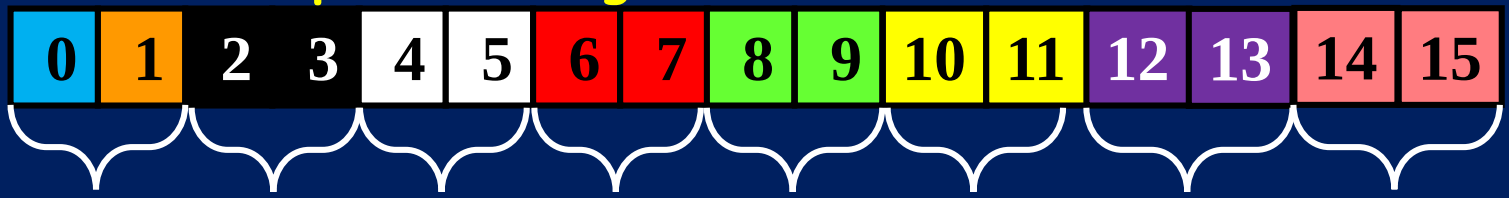
No usa doble-buffer asi que no gasta memoria y es muy rápido
Jamás destruye el fondo.
Técnica usada en juegos como "mision genocide" y "wonderboy"

FONDO= PIXEL AND 0001
New PIXEL= SPRITE OR FONDO

Se reduce el número de colores
9 en mode 0
3 en mode 1

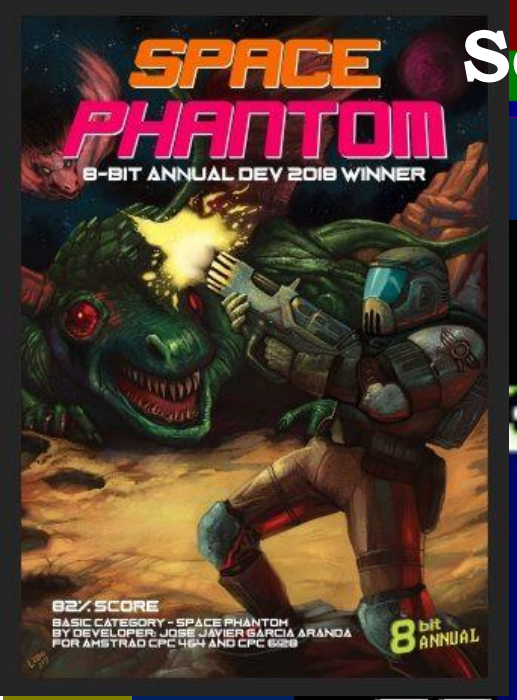


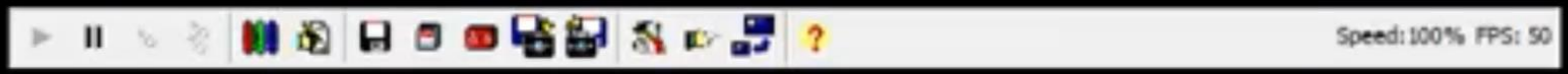
- Hay truco para usar mas de 9 colores en MODE 0 con sobreescritura
- 8BP también permite elegir 1 o 2 bit de fondo



Sprites: sobreescritura

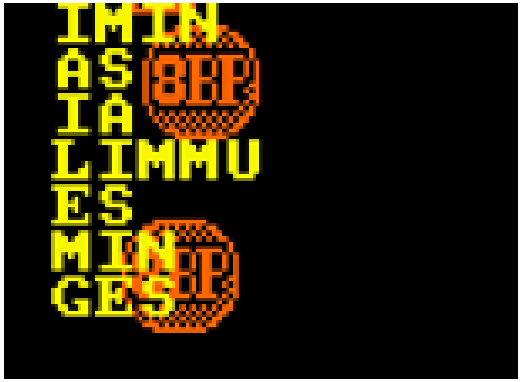
Sobreescritura y animación por rotación de tintas (RINK)

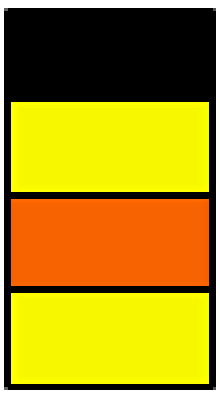




Sprites: sobreescritura

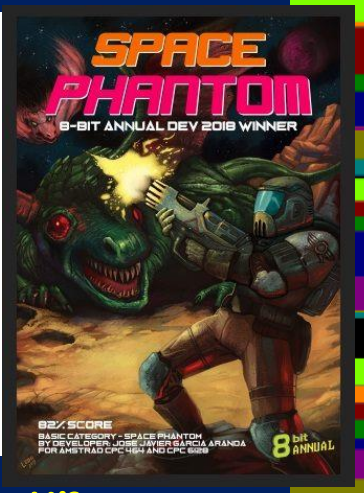
Con la misma técnica podemos pintar por detrás del fondo





00
01
Colores de fondo

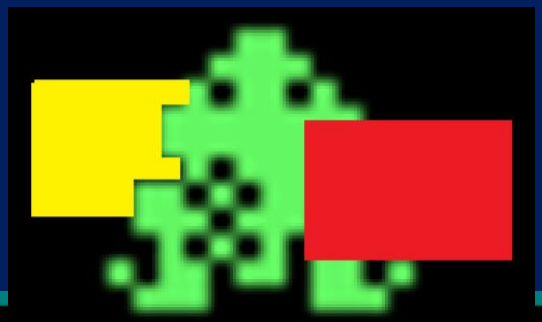
10
11
Color de sprite



En mode 0 incluso podemos hacer objetos por los que un personaje pasa por delante y otros por los que pasa por detrás

FONDO (2 bit)	SPRITE (2 bit)
0000	0100
0001	0101
0010	0110
0011	0111

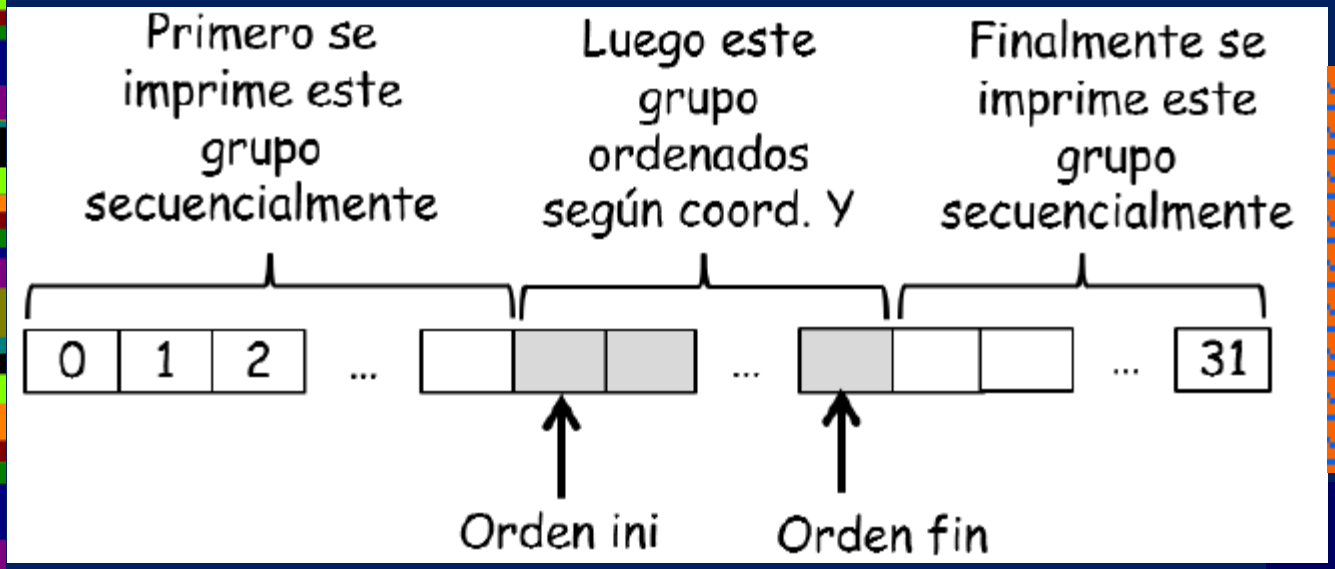
En este ejemplo de paleta con 4 colores de fondo y 2 para sprites, el color verde puede pasar por delante del color amarillo y azul pero pasa por detrás del color rojo



Sprites: ordenamiento

La "burbuja restringida a un solo cambio" es lo mas rápido si casi están ordenados

```
|PRINTSPALL, ordenini, ordenfin, anima, sync
```



inicializacion del comando (solo una vez)

PRINTSPALL,0 : ordenacion parcial de sprites basado en Ymin

PRINTSPALL,1 : ordenacion total de sprites basado en Ymin

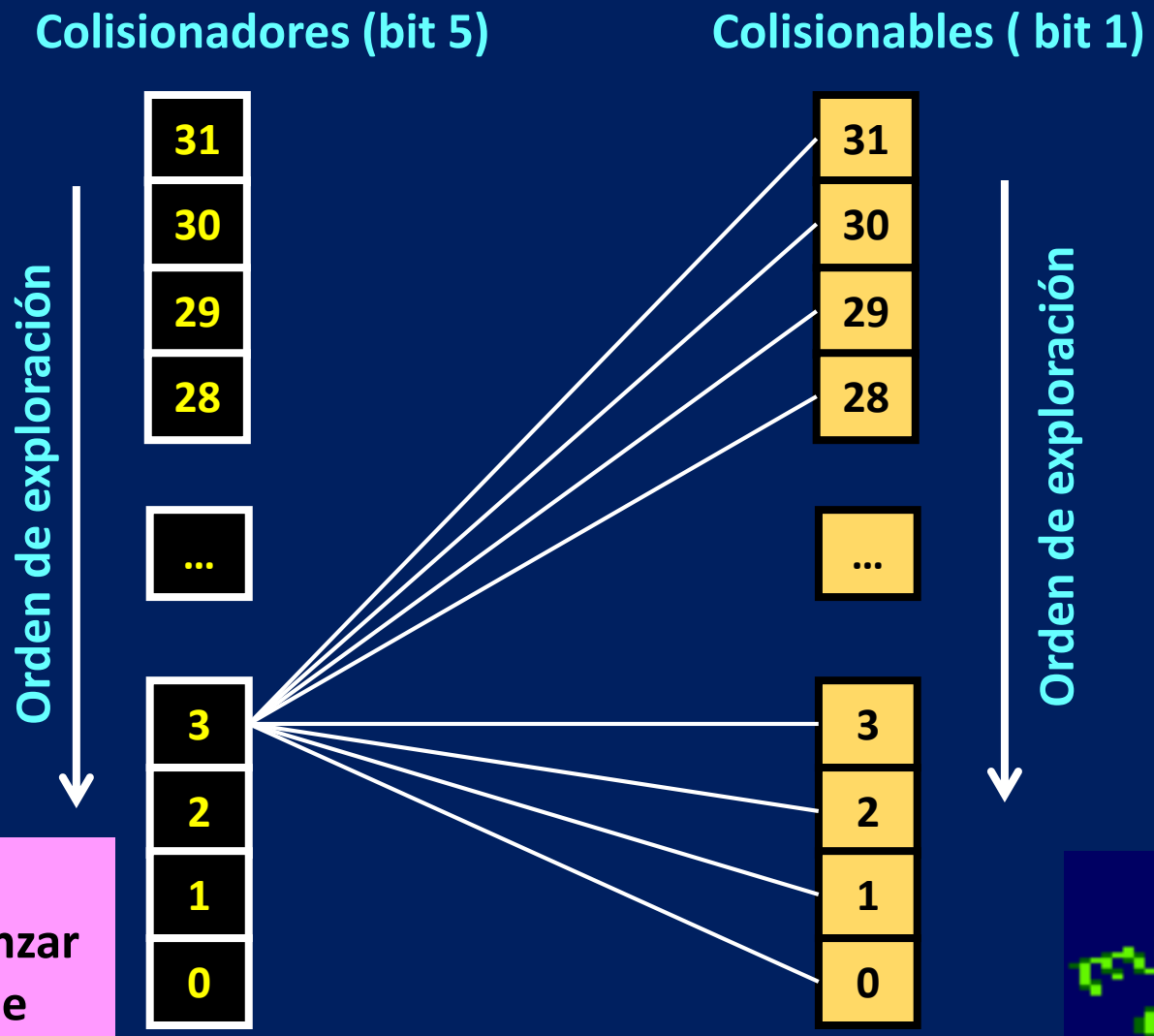
PRINTSPALL,2 : ordenacion parcial de sprites basado en Ymax

PRINTSPALL,3 : ordenacion total de sprites basado en Ymax


DEMO 8BP V36
0 : ORDEN PARCIAL EN Y MINIMA
1 : ORDEN COMPLETO EN Y MINIMA
2 : ORDEN PARCIAL EN Y MAXIMA
3 : ORDEN COMPLETO EN Y MAXIMA
order type (0-3)? 2
cesped no ordenado
resto ordenados
demo
Speed: 100% FPS: 50

Sprites: colisiones

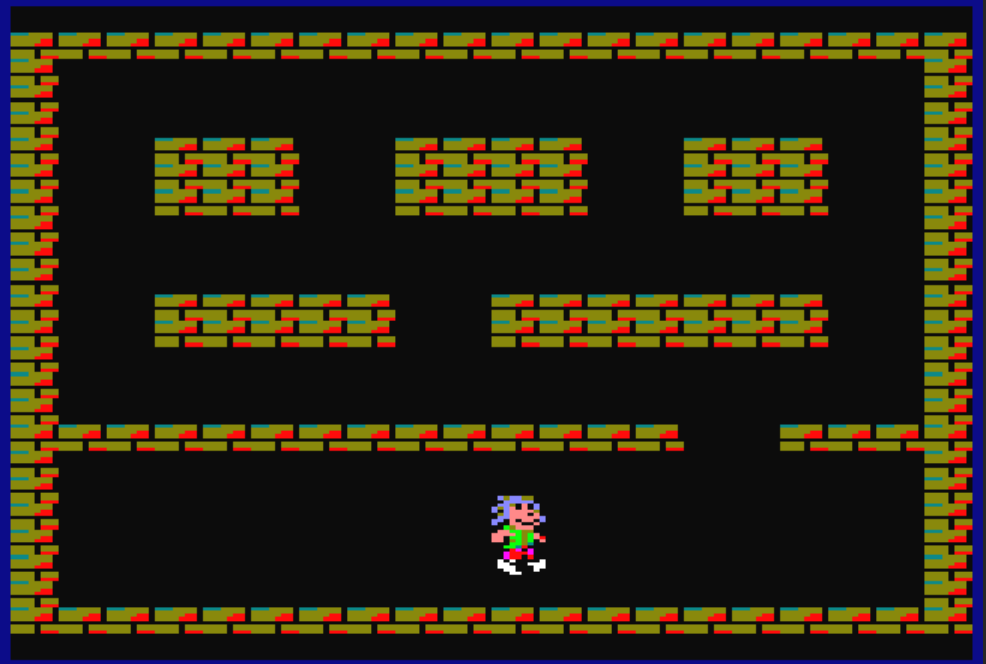
Funcionamiento de |COLSPALL



A partir de v37 podemos comenzar en el collider que queramos

Mapa de tiles (layout)

```
100 c$(1)= "ZZZZZZZZZZZZZZZZZZZZ"
110 c$(2)= "Z                  Z"
120 c$(3)= "Z                  Z"
125 c$(4)= "Z                  Z"
130 c$(5)= "Z  ZZZ  ZZZZ  ZZZ  Z"
140 c$(6)= "Z  ZZZ  ZZZZ  ZZZ  Z"
150 c$(7)= "Z  ZZZ  ZZZZ  ZZZ  Z"
160 c$(8)= "Z                  Z"
170 c$(9)= "Z                  Z"
190 c$(10)="Z                  Z"
195 c$(11)="Z  ZZZZZ  ZZZZZZZ  Z"
200 c$(12)="Z  ZZZZZ  ZZZZZZZ  Z"
210 c$(13)="Z                  Z"
220 c$(14)="Z                  Z"
230 c$(15)="Z                  Z"
240 c$(16)="ZZZZZZZZZZZZZ  ZZZZ"
250 c$(17)="Z                  Z"
260 c$(18)="Z                  Z"
270 c$(19)="Z                  Z"
271 c$(20)="Z                  Z"
272 c$(21)="Z                  Z"
273 c$(22)="Z                  Z"
274 c$(23)="ZZZZZZZZZZZZZZZZZZZZ“
' print layout
560 FOR i=0 TO 23:|LAYOUT,i,0,@c$(i):NEXT
```



El comando LAYOUT interpreta cada letra como un sprite.

En este caso la “Z” es el sprite 31 y le hemos asignado una imagen de ladrillos

El espacio es el sprite NINGUNO

Colisión con mapa de tiles (layout) y colisión entre sprites

```
COLAY, umbral, @col, sp
```

0 = no hay colisión

1 = hay colisión

En v37 se ha cambiado el orden de parámetros de COLAY y se puede invocar sin parametros

```
COLSPALL,@collider , @collided
```

32 = no hay colisión
<32 hay colision

32 = no hay colisión
<32 hay colision

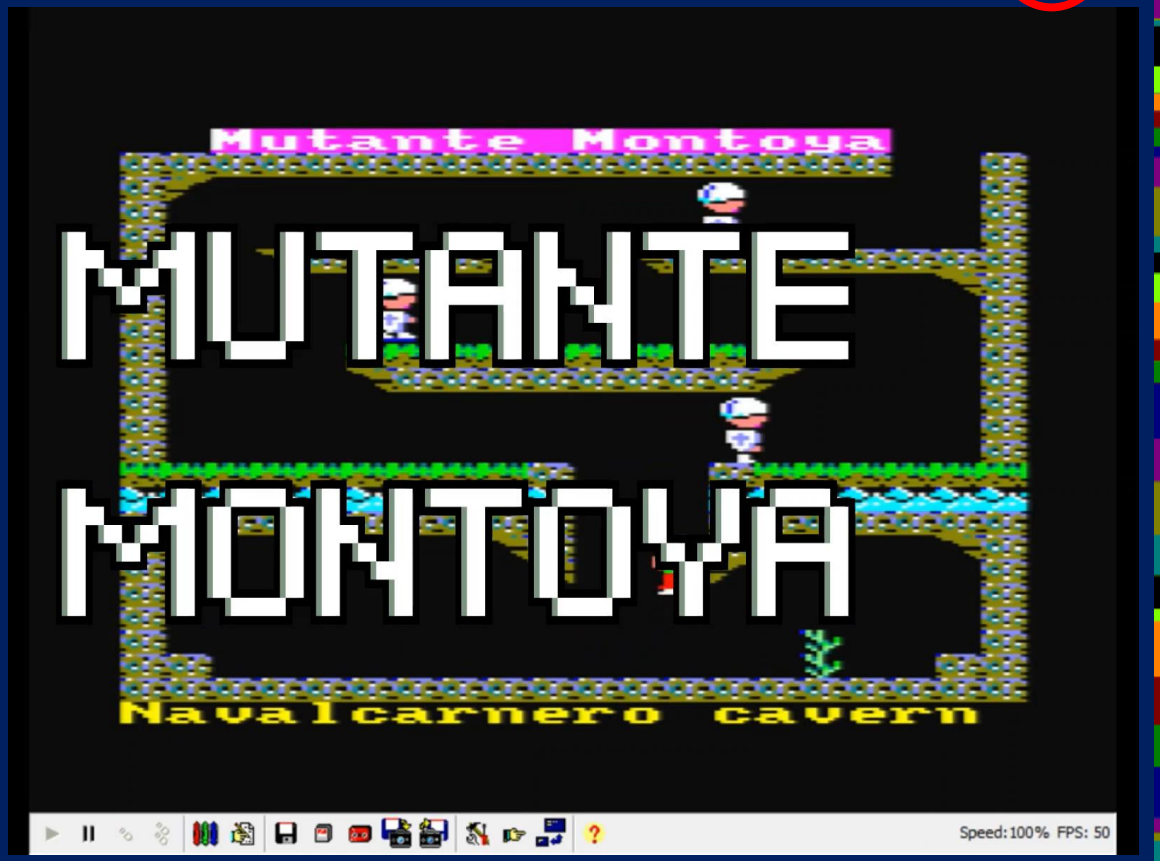
```
COLSPALL, collider
```
: Comienza a explorar en collider

Sprites: secuencias de animación

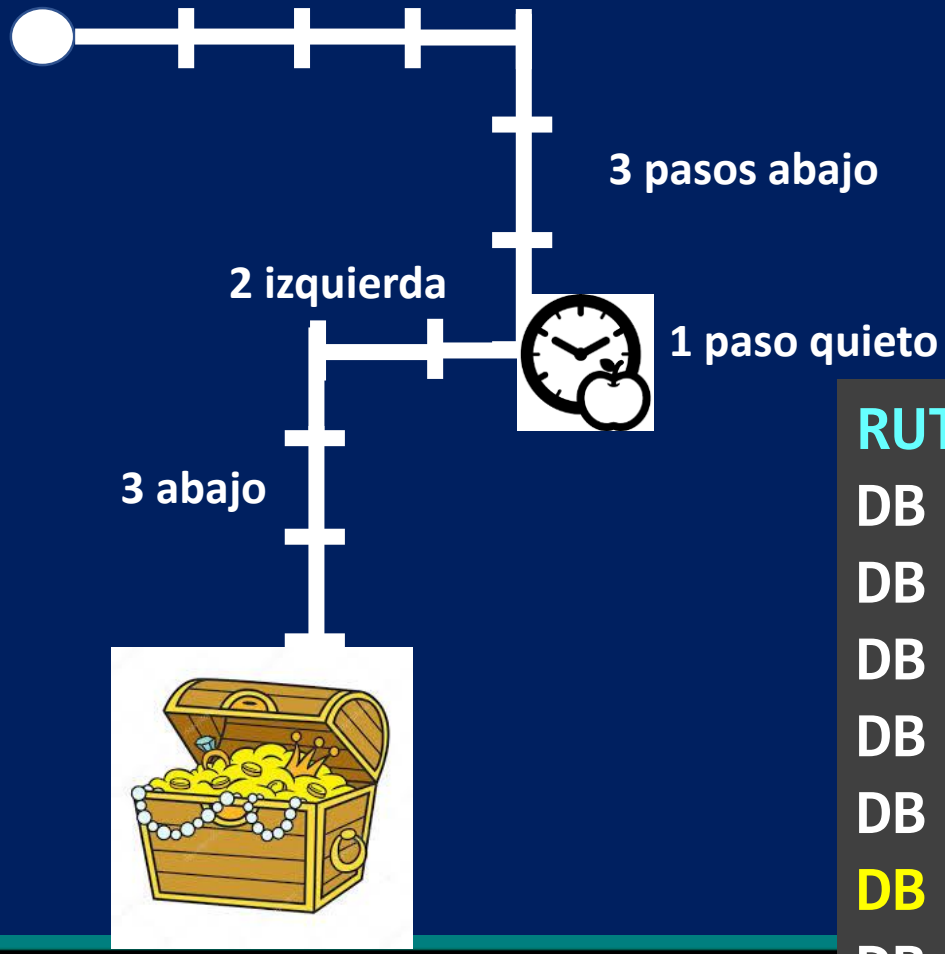
```
|SETUPSP, <sprite_id>, 7,  <sequence number>

|PRINTSPALL,0,0,1,0
```

sequences_mygame.asm

```
_SEQUENCES_LIST
dw MONTOYA_R0,MONTOYA_R1,MONTOYA_R2,MONTOYA_R1,0,0,0,0    ;1
```



Sprites: concepto de ruta



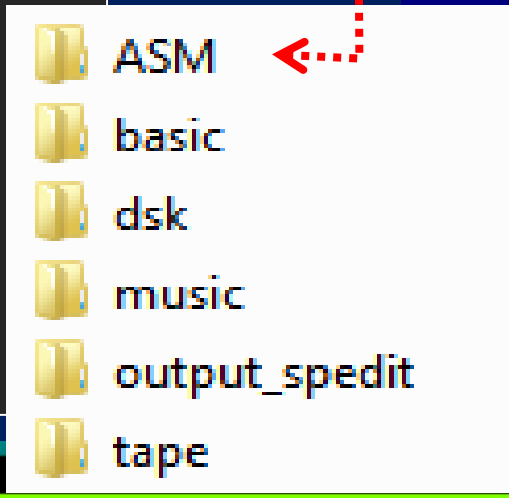

```
RUTA DEL TESORO
DB 4, 0, 1
DB 3, 1, 0
DB 1, 0, 0
DB 2, 0, -1
DB 3, 1, 0
DB 1,-6,-2
DB 0
```

Sprites: concepto de ruta

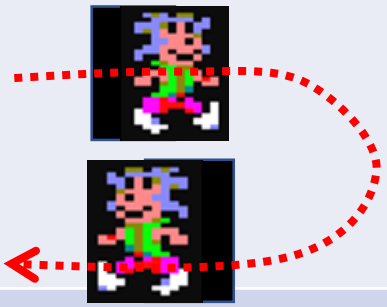
Routes_mygame.asm

```
; LISTA DE RUTAS
;================
;pon aqui los nombres de todas las rutas que hagas
ROUTE_LIST
    dw ROUTE0
    dw ROUTE1
    dw ROUTE2
    dw ROUTE3
…

ROUTE0 ; RUTA DEL TESORO
DB 4, 0, 1
DB 3, 1, 0
DB 1, 0, 0
DB 2, 0, -1
DB 3, 1, 0
DB 1,-6,-2
DB 0
```

```
|SETUPSP,pirata,15, 0
```



Sprites: concepto de ruta

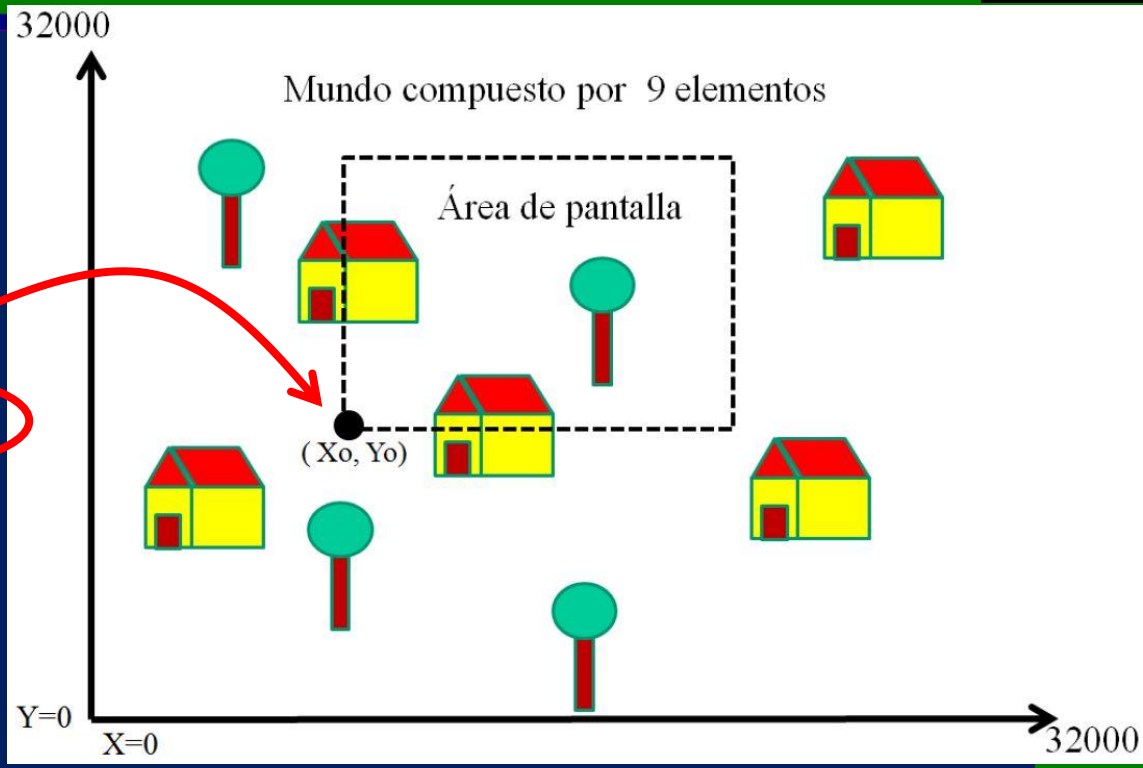
code	Descripción	Ejemplo
255	Cambio de estado	DB 255,3,0 Estado pasa a valor 3. El cero del final es de relleno
254	Cambio de secuencia de animación 	DB 254,10,0 Se asocia la secuencia 10. El cero es de relleno Si la secuencia asignada es la que ya tiene el sprite, entonces es inocuo (no se reinicia la secuencia de animación
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
251	Pasa al siguiente frame de la animación. 	DB 251,0,0 Se anima el Sprite. Los dos ceros son de relleno

Scroll multidireccional

Scroll basado en una "ventana deslizante"

Comando

`|MAP2SP, Yo, Xo`



OJO: las coordenadas para imprimir sprites van al revés en el eje vertical pero la traducción es automática, no tienes que hacer nada

El mundo mide 32000 x 32000

Scroll multidireccional

31
30
29
28

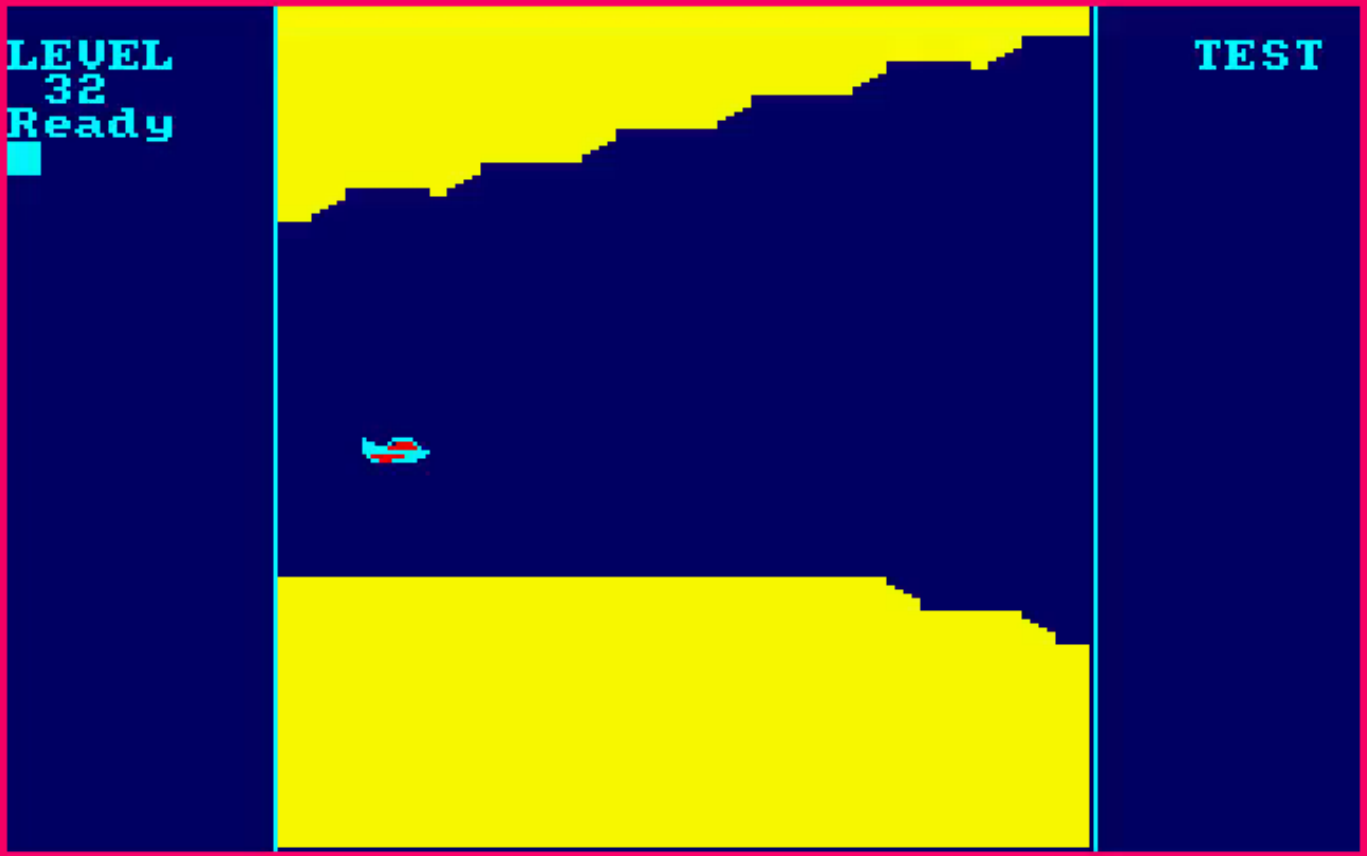
muñecos

...

3
2
1
0

Generados por **MAP2SP** Montañas , casas...

Scroll multidireccional: 32 fps!

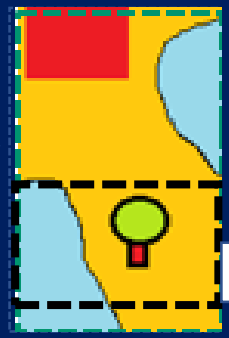


Speed:100% FPS: 50

Scroll multidireccional

Mapa completo en dirección 23000 (por ejemplo)

Mapa parcial ubicado en la dirección 42040

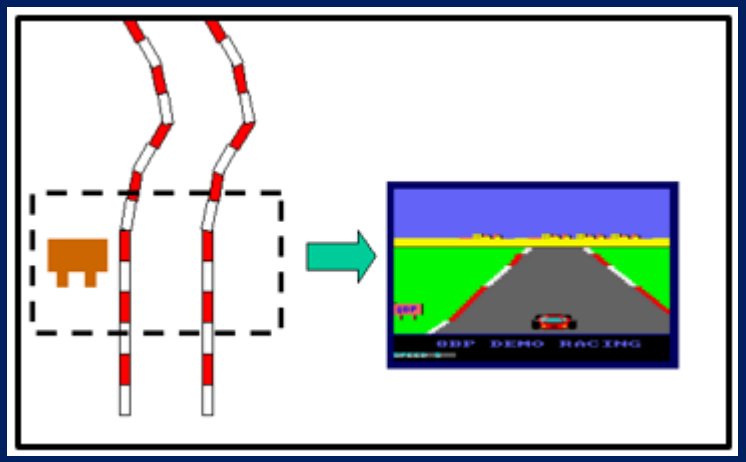


UMAP,23000,24000,200,500,0,80

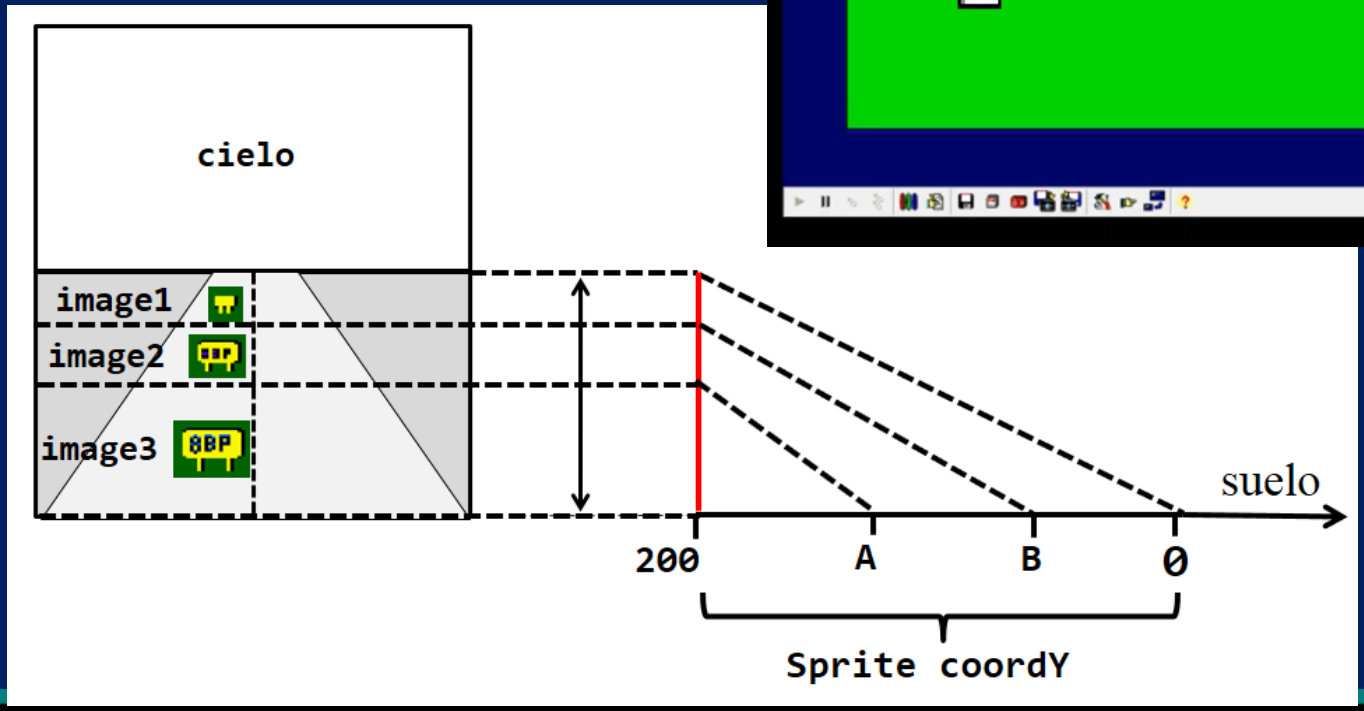
MAP2SP,y,x

MAP2SP se invoca en cada ciclo de juego. Cuando nos acerquemos a la frontera del mapa parcial invocaremos UMAP para que actualice el mapa parcial

Pseudo-3D



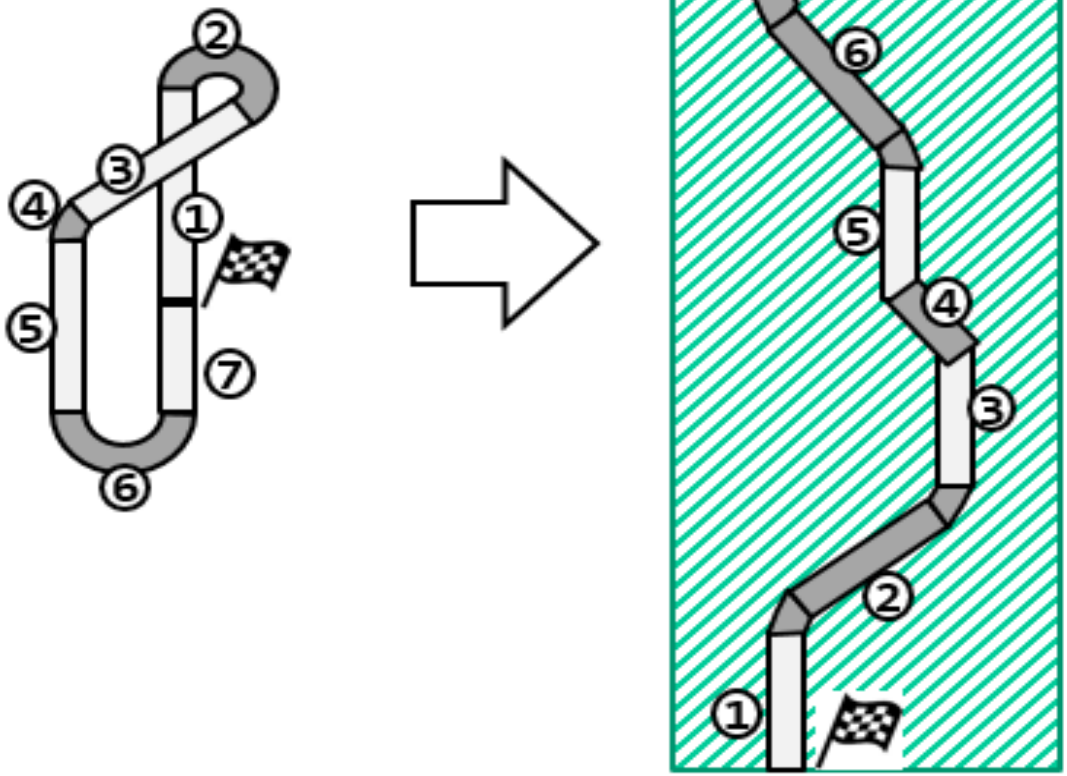
El comando 3D nos permite recorrer un mundo 2D proyectado

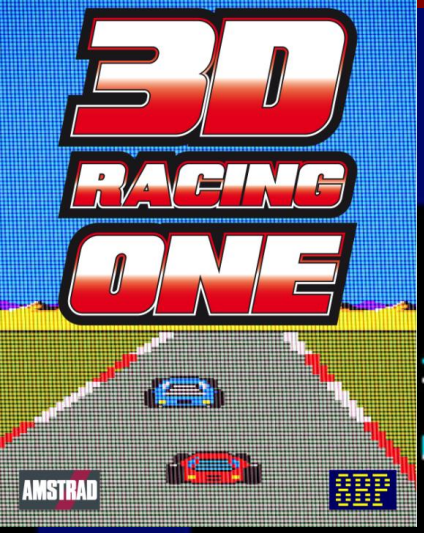


CONCEPTO:

Sólo es 3D el plano del suelo.

Las curvas son una ilusión.

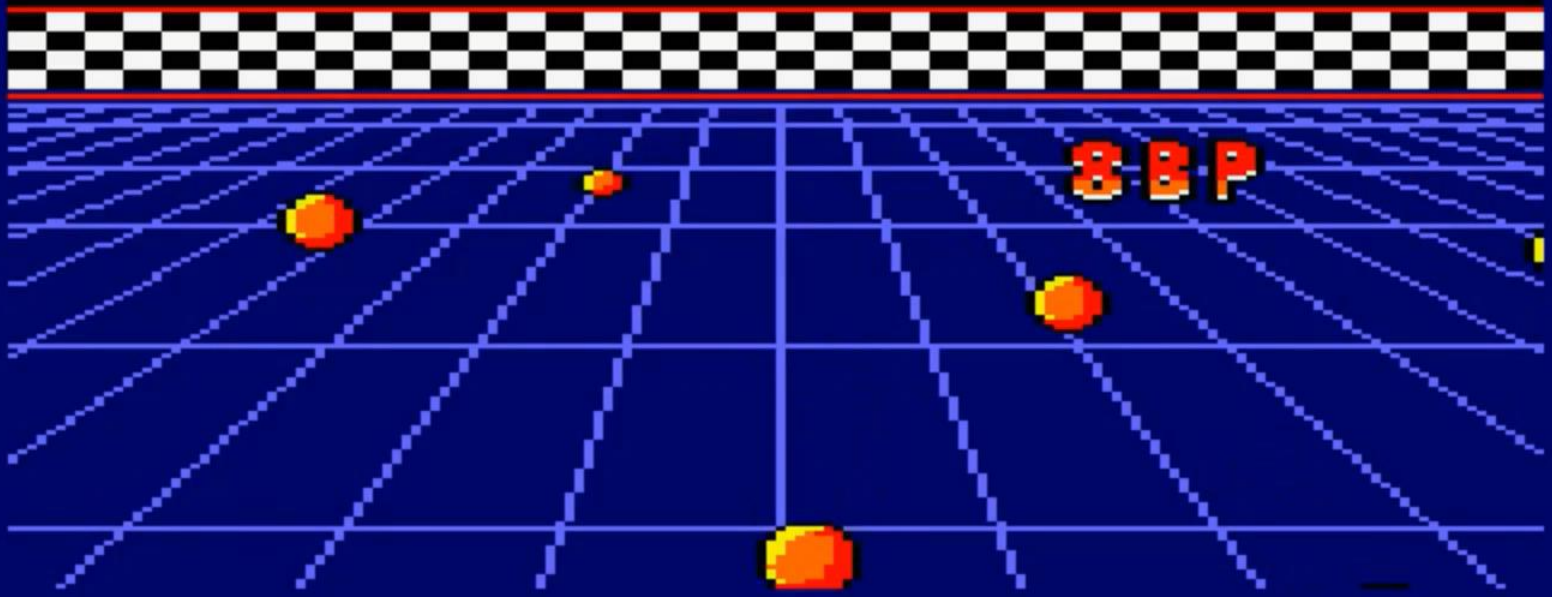




3D RACING ONE

JOSE JAVIER GARCIA ARANDA 2018

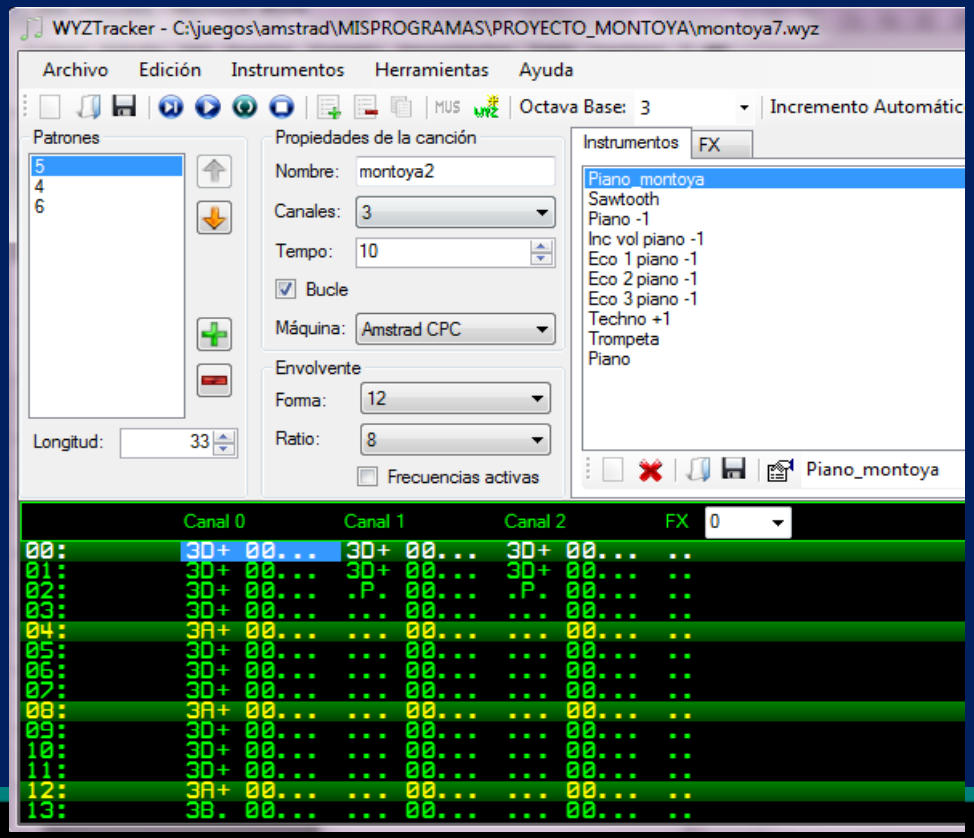
LY BASIC PROGRAM CREATED WITH BBP



Speed:100% FPS: 50

Musica on-game

```
10 |MUSIC,3,6 : comienza a sonar la melodia 3 a velocidad 6

. . .

2000 |MUSICOFF: ' deja de sonar la musica
```



Compones las canciones con el WYZ tracker

El WYZ player esta integrado en la librería

Lógicas masivas

Algunos comandos la llevan "por dentro":

|COLSPALL (estrategia de 1 colision por frame)
|PRINTSPALL (ordenamiento burbuja restringida)
|AUTOALL, |MOVERALL, |ROUTEALL, |PRINTSPALL… (grupos)

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 o |AUTOALL,1

Ejecución de menos comprobaciones

- En cada ciclo de juego ejecutar un subconjunto de comprobaciones usando aritmética modular en cascada. Todas las comprobaciones tardarán varios frames pero no importa

Ejecución de una sola lógica

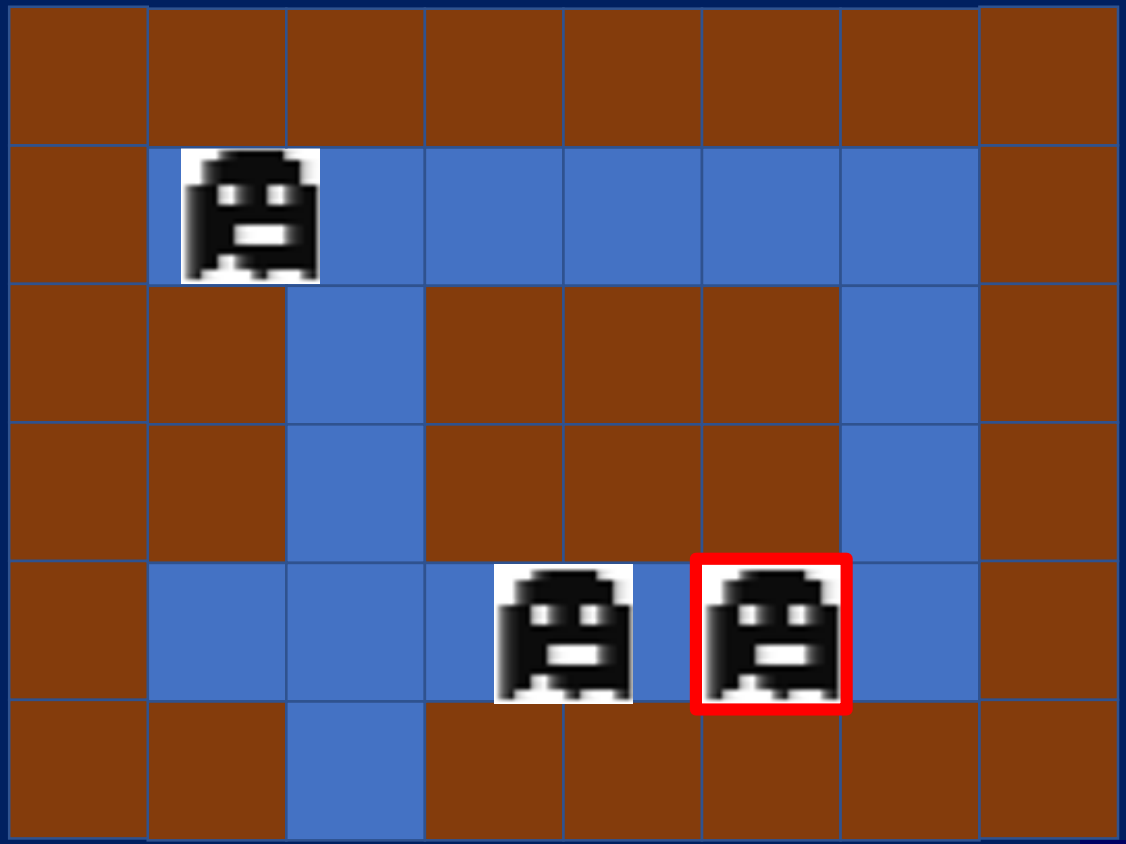
- En cada ciclo de juego ejecutar la "Inteligencia" de un solo enemigo
- En juegos de laberintos podemos "adaptar el mapa a la inteligencia"

limitaciones:

Introduce limitaciones "indetectables" en tu juego
Ejemplo: solo muere un enemigo en cada fotograma
Ejemplo: solo coges la moneda en uno de cada 2 fotogramas
Ejemplo: en cada ciclo de juego no lees todas las teclas
Ejemplo : nuevo enemigo solo en frames multiplos de 8

Lógicas masivas: ejemplos

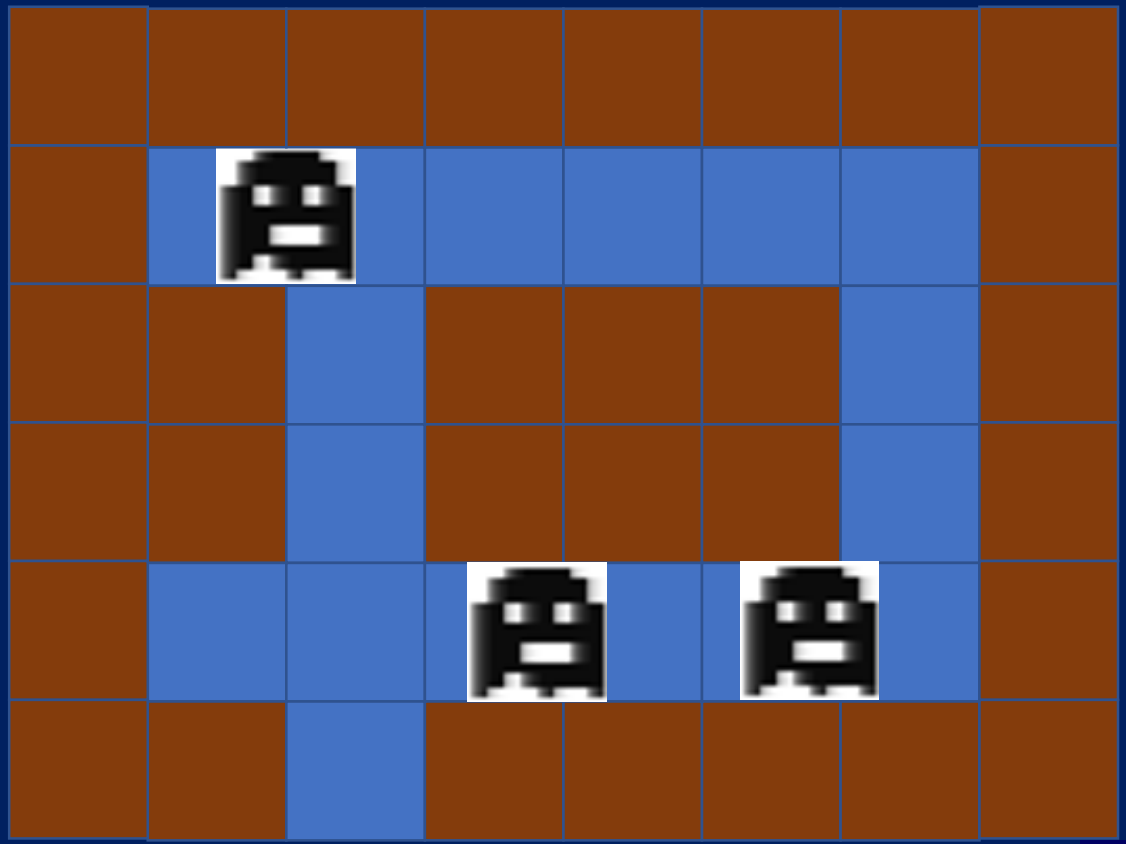
Ciclo= 0



massive logics

Lógicas masivas: ejemplos

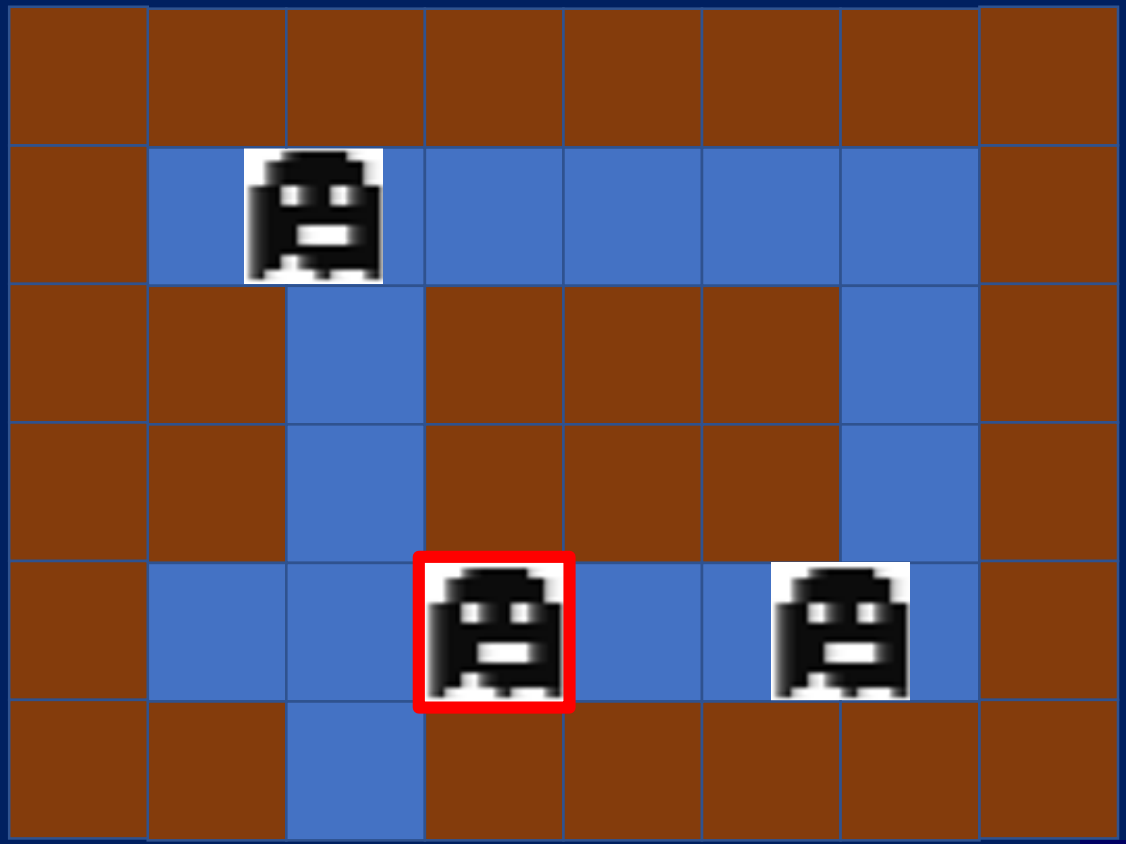
Ciclo= 1



massive logics

Lógicas masivas: ejemplos

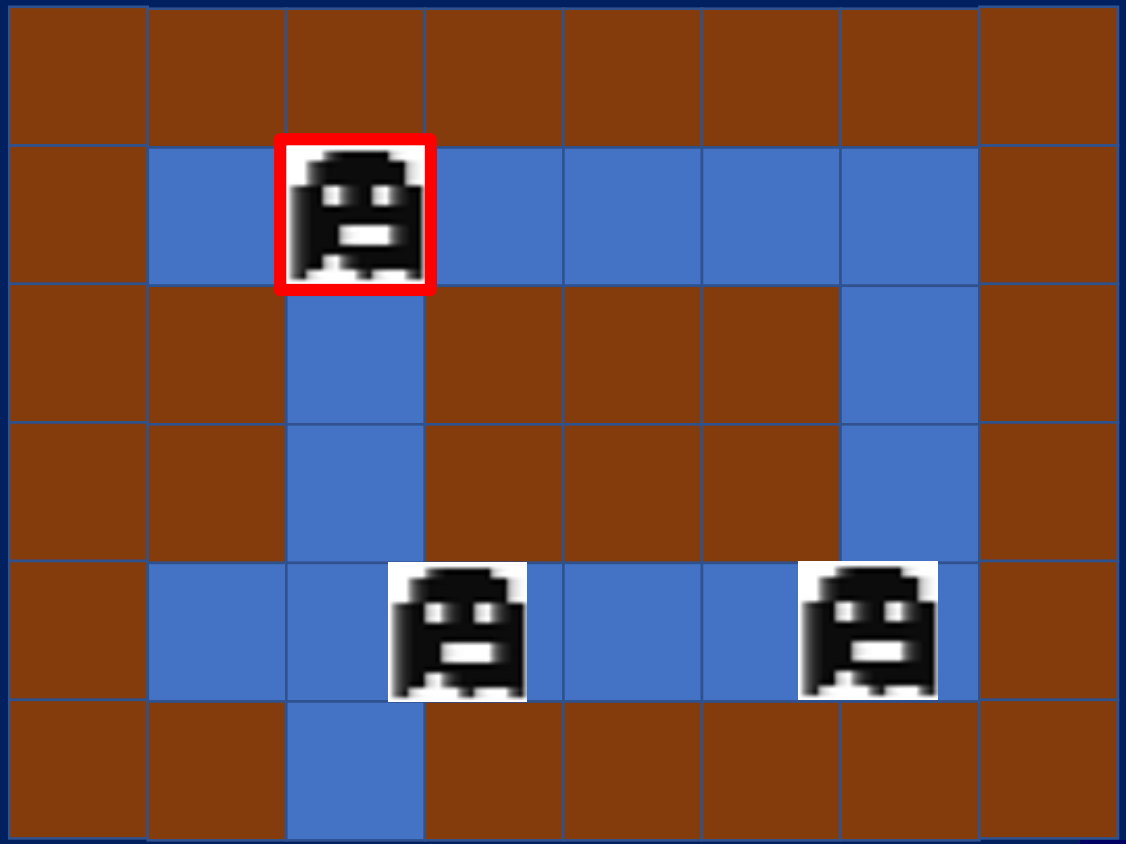
Ciclo= 2



massive logics

Lógicas masivas: ejemplos

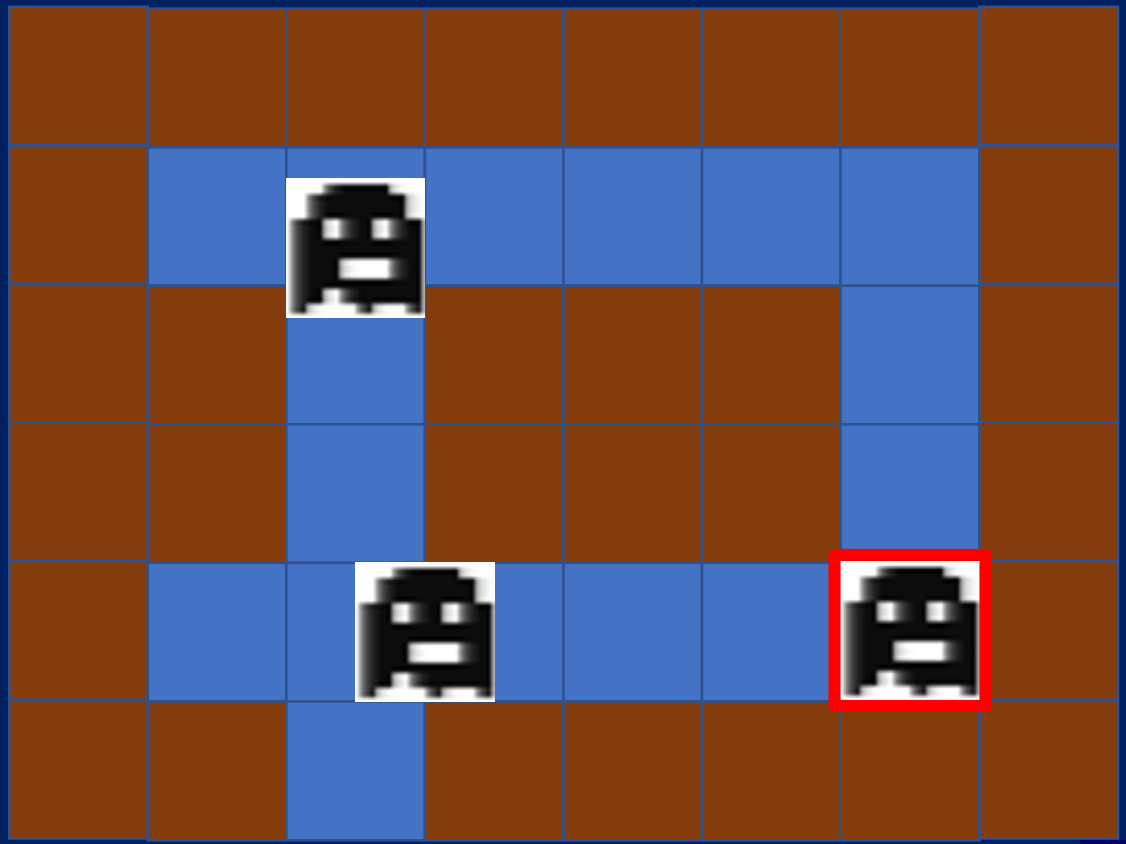
Ciclo= 3



massive logics

Lógicas masivas: ejemplos

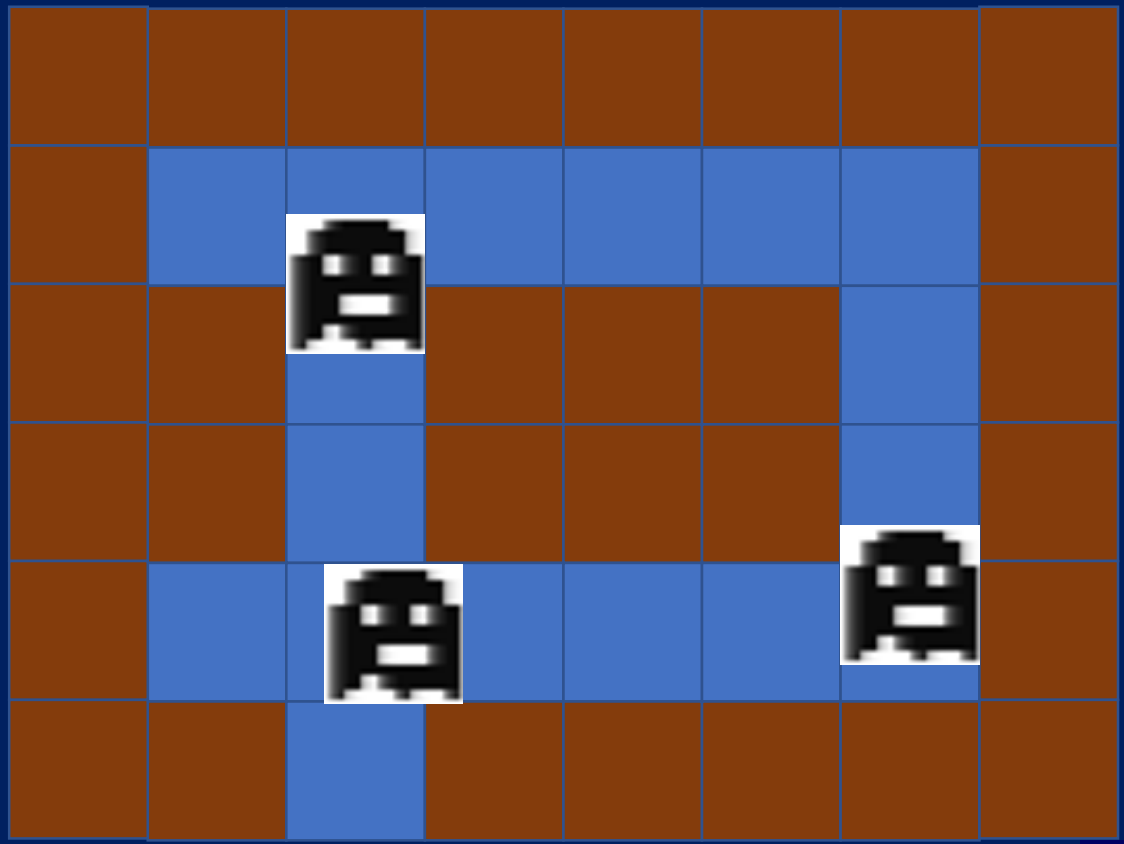
Ciclo= 4



massive logics

Lógicas masivas: ejemplos

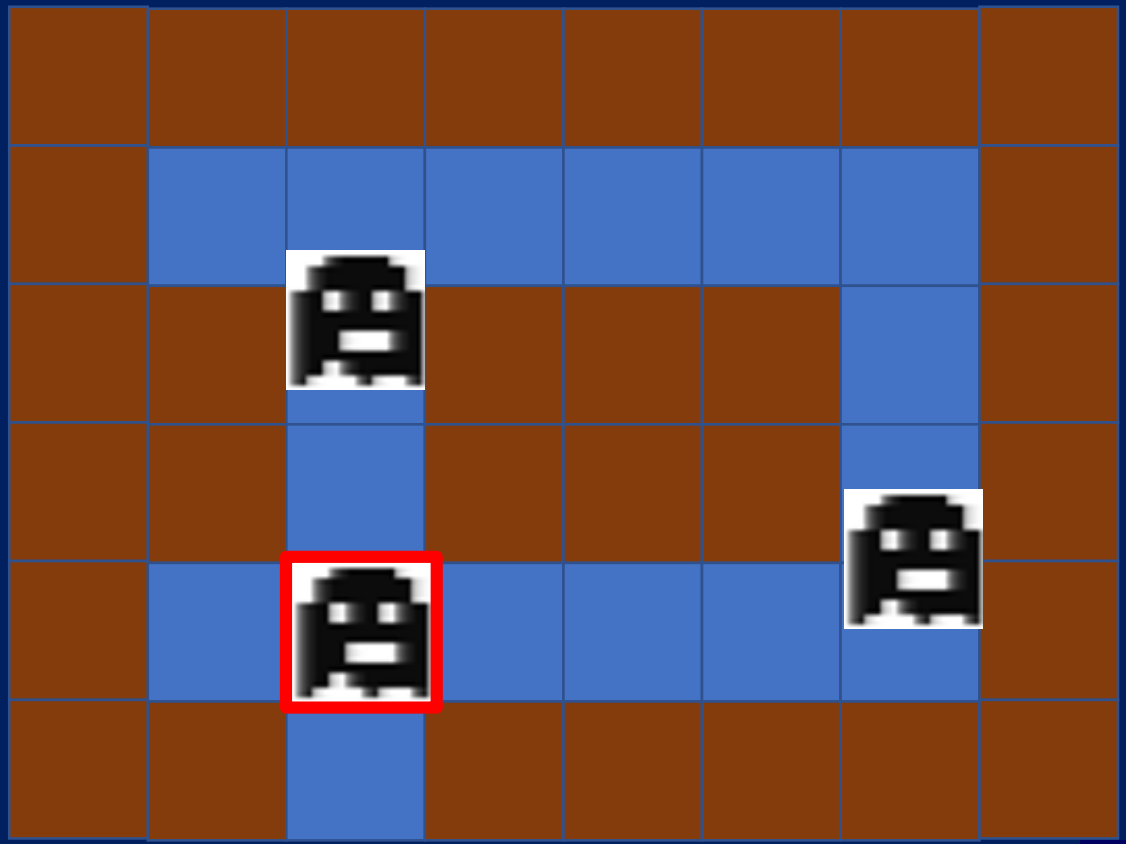
Ciclo= 5



massive logics

Lógicas masivas: ejemplos

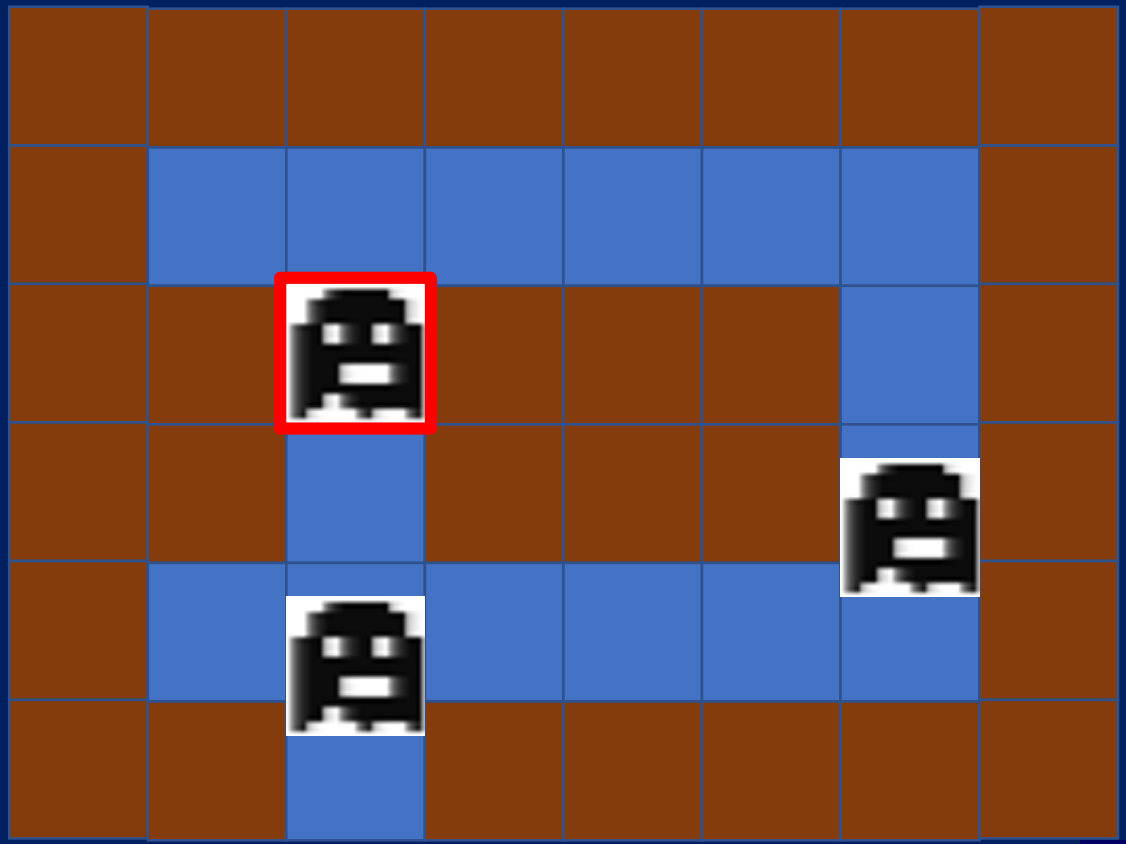
Ciclo= 6



massive logics

Lógicas masivas: ejemplos

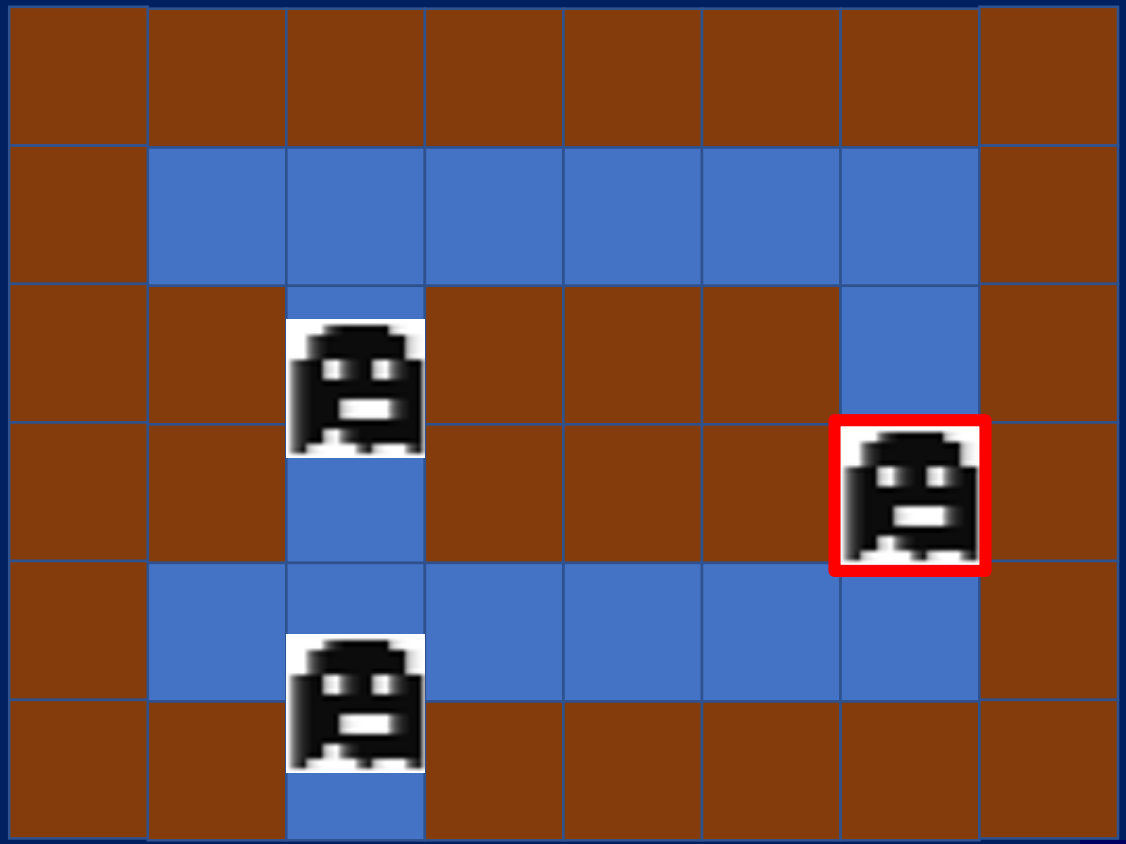
Ciclo= 7



massive logics

Lógicas masivas: ejemplos

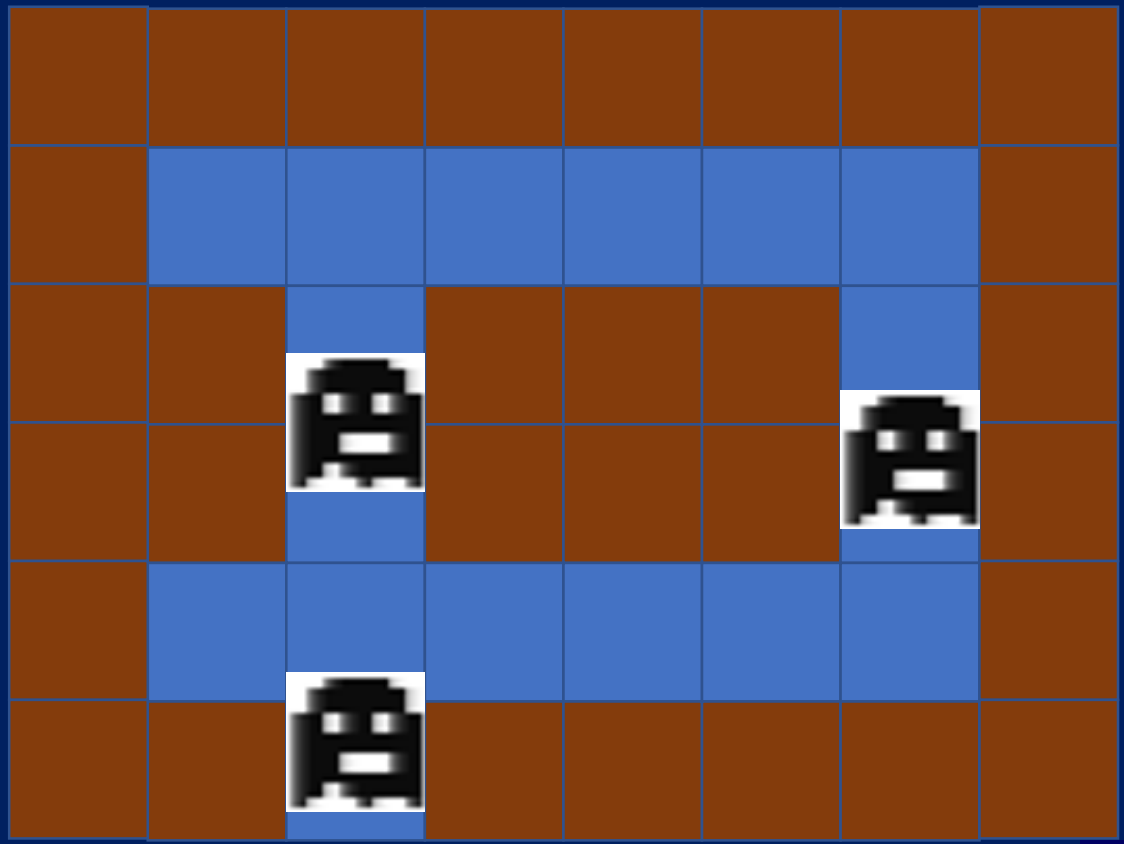
Ciclo= 8



massive logics

Lógicas masivas: ejemplos

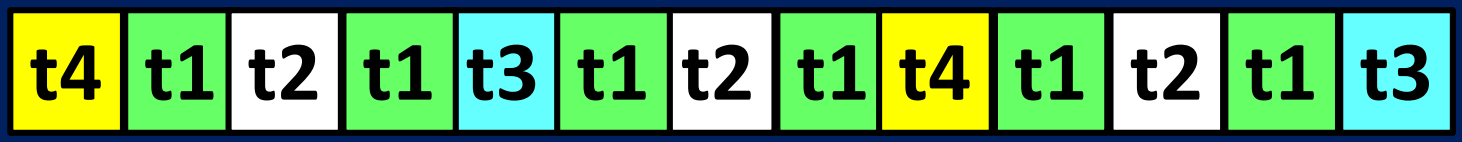
Ciclo= 9



massive logics

Lógicas masivas: lógica modular en cascada

4 tareas, pero tan sólo una se ejecuta cada la vez




```
10 IF ciclo AND 1 THEN 90
20 REM cada dos ciclos entramos aquí
25 IF ciclo AND 3 THEN 80
30 REM cada 4 ciclos entramos aquí
35 IF ciclo AND 7 THEN 70
40 REM cada 8 ciclos entramos aquí
50 <tarea 4> : GOTO 100
70 <tarea 3> : GOTO 100
80 <tarea 2> : GOTO 100
90 <tarea 1>
100 REM --- fin de tareas ---
```

Lógicas masivas: lógica modular en cascada

En este ejemplo hemos elegido los intervalos 2,4 y 8
AND 1 : es cero cada 2 ciclos
AND 3 : es cero cada 4 ciclos
AND 7 : es cero cada 8 ciclos

Gracias a que hemos elegido operaciones en intervalos multiplo, los IF se ejecutan "en cascada": solo entramos en un IF si hemos entrado en el anterior:
50% de los ciclos solo ejecutan un IF (línea 10)
50% ejecutan 2 IF (líneas 10 y 25) de los cuales la mitad (el 25%) ejecutan 3 IF (líneas 10, 25 y 35)

En media se ejecutan 1*50%+2*25%+3*25%=1.75 sentencias IF por ciclo

Consejos y trucos

Consejos

trucos

Consejos

- Primeras líneas de tu programa
- Linea fundamental de tu ciclo
- Crea tus sprites con spedit
- Flipea para ahorrar memoria
- Una sola lógica para todo el juego
- Juegos de pantallas vs Juegos con scroll
- El memory map de tu juego
- Ahorra lógica con Secuencias de muerte
- Elige tus sprites id para colisionar en orden

Primeras líneas de tu programa

```
10 MEMORY 23999
20 CALL &6B78
30 DEFINT A-Z
```

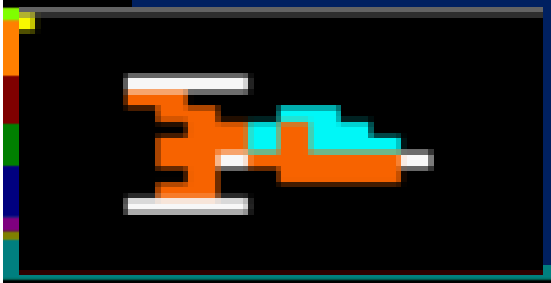
Línea fundamental de tu ciclo de juego

```
100 |AUTOALL,1:|PRINTSPALL: |COLSPALL
```

O bien, en juegos con scroll:

```
100 |MAP2SP,y,x:|AUTOALL,1:|PRINTSPALL: |COLSPALL
```

Diseña tus sprites con SPEDIT11
(es fácil y viene con 8BP)



.txt

```
;------ BEGIN IMAGE --------
db 9 ; ancho
db 17 ; alto
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 48 , 48 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 252 , 0 , 0 , 160 , 0 , 0 , 0
db 0 , 0 , 84 , 248 , 164 , 88 , 0 , 0 , 0
db 0 , 0 , 0 , 252 , 92 , 12 , 240 , 0 , 0
db 0 , 0 , 84 , 252 , 92 , 12 , 8 , 0 , 0
db 0 , 0 , 84 , 184 , 252 , 252 , 184 , 0 , 0
db 0 , 0 , 0 , 248 , 84 , 252 , 248 , 0 , 0
db 0 , 0 , 84 , 168 , 80 , 240 , 0 , 0 , 0
db 0 , 0 , 48 , 48 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
;------ END IMAGE --------
```

.asm

```
;------ BEGIN IMAGE ----
NAVE
db 9 ; ancho
db 17 ; alto
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 48 , 48 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 252 , 0 , 0 , 160 , 0 , 0 , 0
db 0 , 0 , 84 , 248 , 164 , 88 , 0 , 0 , 0
db 0 , 0 , 0 , 252 , 92 , 12 , 240 , 0 , 0
db 0 , 0 , 84 , 252 , 92 , 12 , 8 , 0 , 0
db 0 , 0 , 84 , 184 , 252 , 252 , 184 , 0 , 0
db 0 , 0 , 0 , 248 , 84 , 252 , 248 , 0 , 0
db 0 , 0 , 84 , 168 , 80 , 240 , 0 , 0 , 0
db 0 , 0 , 48 , 48 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
db 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
;------ END IMAGE ------
```

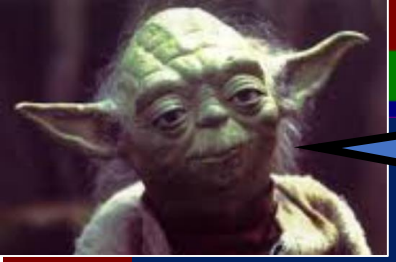
Ves que fácil es?

Images_mygame.asm

```
IMAGE_LIST
;-----------------------------------------------------------
; pondremos aqui una lista de las imagenes que queremos usar
; se empiezan a numerar en 16
; podemos usar hasta 255 imagenes especificadas de este modo
;-----------------------------------------------------------
DW AUA_A;16
DW AUA_U;17
DW AUA_RED;18
DW AUA_GREEN;19
DW AUA_BLUE;20
DW NAVE;21
DW COHETE;22
DW ALA_L;23
DW ALA_R;24
DW ERIDU;25
DW LOGO_8BP;26
```

En el listado BASIC:

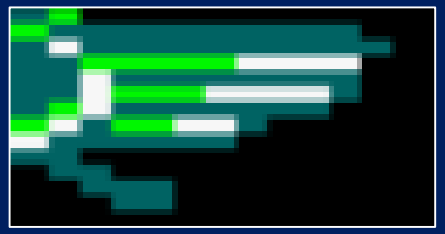
310 |SETUPSP,0,9,26:|PRINTSP,0,162,68



ahorra memoria con flipeo horizontal

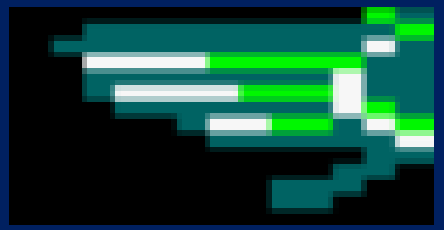
Images_mygame.asm

```
;================================================
_BEGIN_FLIP_IMAGES
;================================================
; aqui pon las imagenes que se definen como otras
; existentes pero flipeadas horizontalmente.
ALA_R    dw ALA_L
;================================================
_END_FLIP_IMAGES
;================================================
```



ALA_R

Se obtiene flipeando:



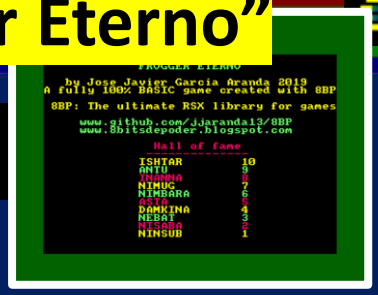
ALA_L

Es muy importante que programes una única lógica de ciclo de juego y la apliques a todas las pantallas.

Recuerda que solo tienes 24 KB para la lógica de tu juego, presentación, etc.

Happy Monty: una sola lógica, 25 niveles

Construcción de juegos de pantallas: “Frogger Eterno”



Inicio y Presentación

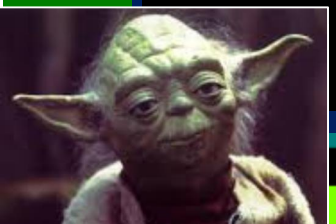
Lógica del programa principal

- GOSUB pantalla 1
- GOSUB pantalla 2
- GOTO FIN

Pantalla N:

- inicialización de sprites (enemigos)
- pintado de escenario
- ciclo de juego:
  - leer teclado y mover personaje
  - [crear enemigos cada X ciclos]
  - decisiones lógicas
  - mover sprites,imprimir sprites
  - detectar colisiones y lógica asociada

Opcional, según el juego



Construcción de juegos con scroll: “Eridu: the space port”

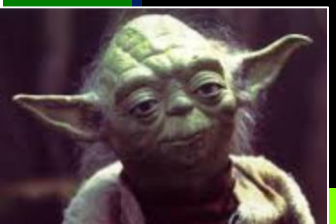
Inicio y Presentación

Lógica del programa principal

- GOSUB fase 1
- GOSUB fase 2
- GOTO FIN

Fase N:

- inicialización de sprites
- pintado de escenario, marcadores
- ciclo de juego:
  - leer teclado y mover personaje
  - Mover mapa
  - crear enemigos según posición mapa
  - decisiones lógicas
  - mover sprites, imprimir sprites
  - detectar colisiones y lógica asociada

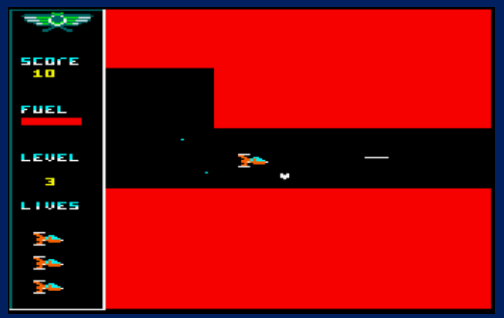


En juegos con scroll haz ficheros de mapas y ficheros de enemigos

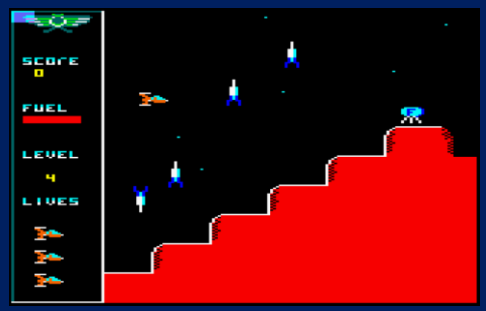
Map_titulo_eridu.asm

Map_fase1.asm
Enemigos_fase1.asm

Sin mapa ni fichero de enemigos



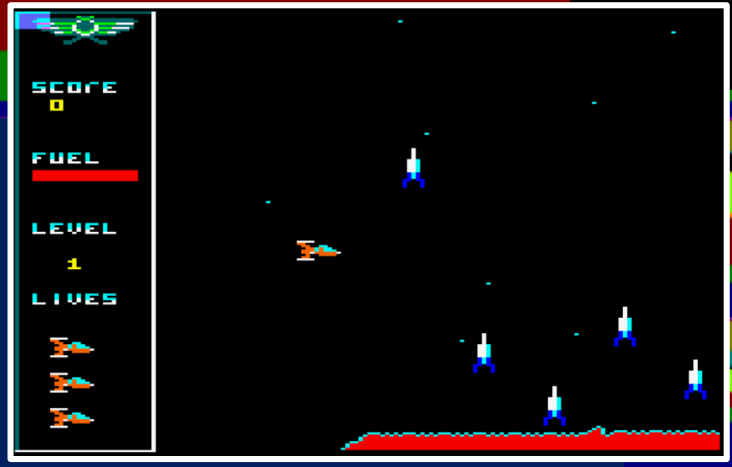
Map_fase3.asm
Enemigos_fase3.asm



Map_fase4.asm
Enemigos_fase4.asm

Map_fase5.asm
Enemigos_fase5.asm

Construye tus mapas uniendo trocitos



Y X

```
org 23300

_MAP_TABLE_FASE1

dw  8,80,SUP
dw  8,84,SHOR2
dw  8,88,SHOR2
dw  8,92,SHOR2
dw  8,96,SHOR2
dw  8,100,SHOR2
dw  8,104,SHOR2
dw  12,108,SHOR
dw  9,112,SHOR2
dw  9,116,SHOR2
dw  9,120,SHOR2
dw  12,124,SHOR
dw  9,128,SHOR2
```

...

1 mapa= 82 trocitos

En juegos con scroll, haz un fichero de enemigos para cada fase

- **No es aıgo especifico de 8BP**
- **los he programado asi en Eridu, no es la única opción posible**
- **Cada 8 fotogramas voy a crear un enemigo en el ciclo de juego**
- **Cada enemigo lo he descrito con 4 bytes:**

Ruta, Y, X, pasos dados en su ruta al nacer

db 2,45,78,0; esto es una gema

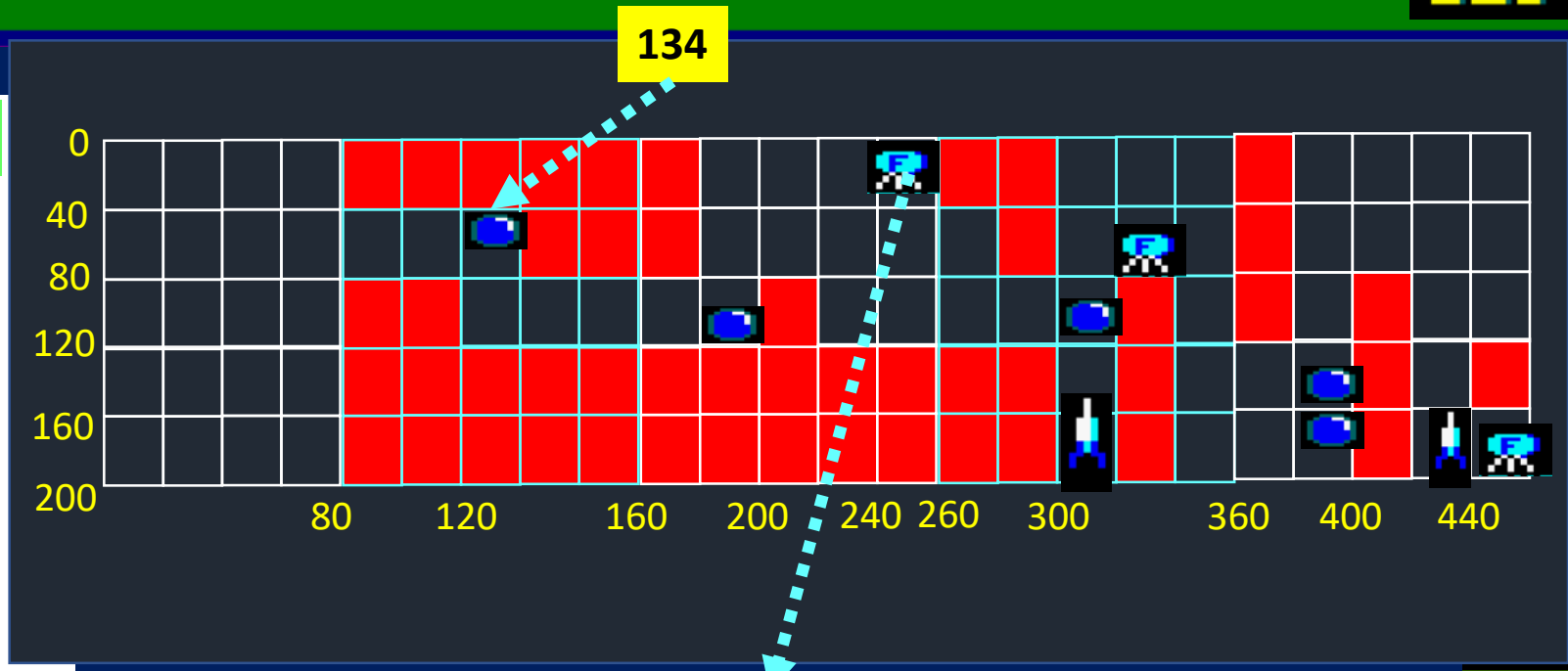
```
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 2,45,78,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
db 2,90,78,0
db 0,0,0,0
db 1,0,83,0;fuel
db 0,0,0,0
```

1

8

13

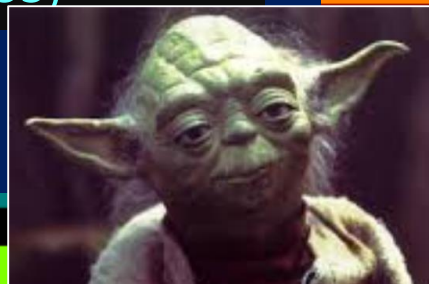
19



Cada entrada es un enemigo y se van a crear cada 8 fotogramas, de modo que, teniendo en cuenta que ancho pantalla es 80:

Xmap=(240-80)/8=20 → lo he ajustado en posición 19 y coordenada 83. Es decir , 19 * 8 +83 = 235

(el objeto se sitúa en la coordenada de mapa 235)



Ciclo de juego de Eridu


```
680 GOSUB 860
700 |MAP2SP,0,m:|AUTOALL,1:|PRINTSPALL:|COLSPALL:IF cor<32 THEN GOSUB 1020:IF
reiniciar THEN RETURN
701 ciclo=ciclo+1:m=m+1
710 IF ciclo AND 7 THEN 680
720 IF e(n) THEN cosp=cosp mod 7+20: |SETUPSP,cosp,0,139: |SETUPSP,cosp,15,e(n):
|ROUTESP,cosp,e(n+3): |LOCATESP,cosp,e(n+1),e(n+2)
730 n=n+4
740 IF ciclo AND 15 THEN 680
750 |STARS
760 IF ciclo AND 63 THEN 680
770 c=color(c):INK 1,c
780 fuel=fuel-1:c$=" ":|PRINTAT,72,1+fuel,@c$
790 IF fuel THEN 820
800 c$="NO FUEL !":|PRINTAT,y-10,x,@c$
810 cicloaux=ciclo:ciclo=ciclomax:level=level-1:GOSUB 1160:'fuel=0
820 IF ciclo>=ciclomax THEN level=level+1:IF fuel>0 THEN cicloaux=0:RETURN ELSE RETURN
'820 IF ciclo>=200 THEN fps=10*ciclo*300/(time-a):print fps:end:RETURN
830 IF m>maxm THEN m=0:n=0:GOSUB 2450
840 GOTO 680
```

20,26,25,24,23,22,21, 20...

LÓGICAS MASIVAS:
4 tareas en cascada
rinde 18 fps en fase2

Haz el MEMORY MAP de tu juego y prepara tu comando memory

Save “eridu.bin”,b,20700,42040-23700

Mapa de memoria de ERIDU

24000

200 bytes: enemigos fase 1

23800

500 bytes: map fase 1

23300

200 bytes: enemigos fase 3

23100

500 bytes: map fase 3

22600

200 bytes: enemigos fase 4

22400

500 bytes: map fase 4

21900

21900

500 bytes: ERIDU TITULO

21400

200 bytes: enemigos fase 5

21200

500 bytes: map fase 5

20700

Libre para BASIC

MEMORY 20699

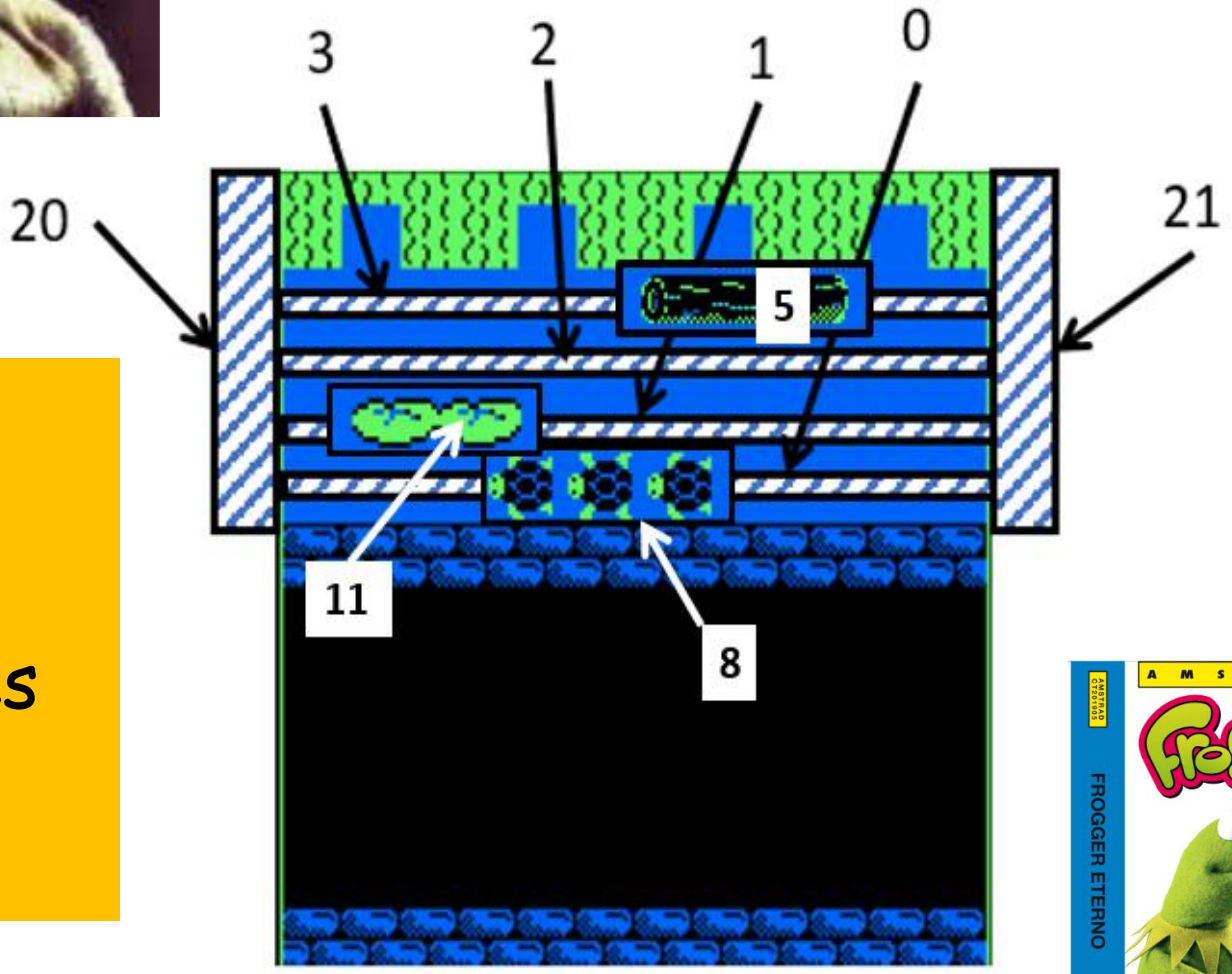
0000

Usa secuencias de Muerte para ahorrar lógica

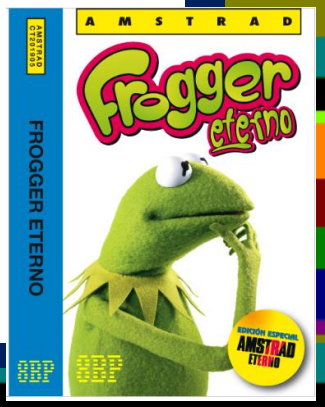
fichero secuences_mygame.asm

```
;-----secuencias de animacion  ----
_SEQUENCES_LIST
dw EXPLO1,EXPLO2,EXPLO3,EXPLO4,BORRA,1,0,0;
```

Elige los sprite ID para que el orden de las colisiones te ayude



Frogger detecta colisión con tronco antes que colisión con rio



Trucos:

- Lectura eficiente de teclado
- disparo múltiple
- Elegir un sprite ID
- Medir los FPS
- Reacción suave (pong)
- Mapas de pantallas compactos
- Colisionadores inversores
- Flags de estado con otro uso
- Fisicas simuladas con rutas
- Mas color con sobreescritura
- Invocar sin parámetros y/o con CALL
- Alterar estado en rutas
- Te falta memoria

Teclado: método eficiente

```
850 '---RUTINA LECTURA CONTROLES
860 IF INKEY(69) THEN 880
870 <instrucciones si pulsas Q>:GOTO 900
880 IF INKEY(67) THEN 900
890 <instrucciones si pilsas A>
900 IF INKEY(34) THEN 920
910 <instrucciones si pulsas O>:GOTO 940
920 IF INKEY(27) THEN 940
930 <instrucciones si pulsas P>
940 IF demora THEN demora=demora-1:RETURN ELSE IF INKEY(47) THEN RETURN
950 disp=(disp AND 1)+29    29, 30, 29, 30...
960 |SETUPSP,disp,0,233: |SETUPSP,disp,15,0: |LOCATESP,disp,y+6,x-8:
demora=7
```

- La demora permite disparos separados en el tiempo
- Este disparo multiple consume dos sprites calculados con aritmética modular

Elegir un sprite ID para crear un enemigo

sp=20 + sp mod 7 1.88 ms

Es la forma mas rápida de recorrer un intervalo de valores cíclicamente

22,21,20,26,25,24,23, 22…

Mide tus FPS

1 Reset Timer:

En un CPC 6128

```
POKE &b8b4,0: POKE &b8b5,0: POKE &b8b6,0: POKE &b8b7,0
```

En un CPC 464

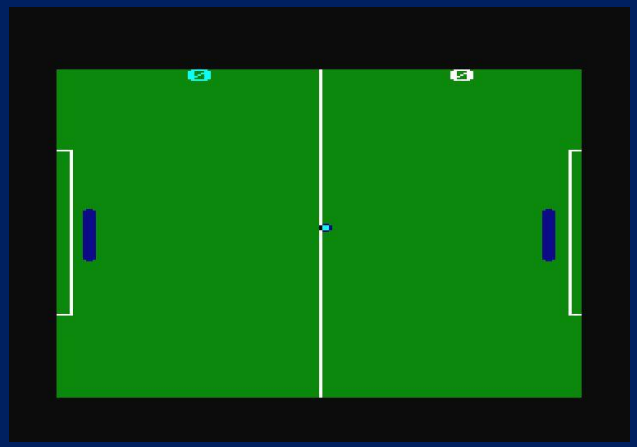
```
POKE &b187,0: POKE &b188,0: POKE &b189,0: POKE &b18a,0
```

2 Ejecuta N ciclos

3 Tras N ciclos , obtén el valor de TIME y calcula FPS

```
If ciclo = N then A=TIME :FPS= ciclo * 300/ (TIME – A): PRINT fps: rem
TIME nos da el tiempo en unidades de duracion 1/300 seg
```

Reacción suave



```
200 ' rutina de movimiento barra
220 IF INKEY(67)=0 THEN vy=-5:GOTO 250
230 IF INKEY(69)=0 THEN vy=5:GOTO 250
240 if vy>0 then vy=vy-1 else if vy<0 then vy=vy+1
250 |SETUPSP,31,5,vy
260 RETURN
```

Mapas compactos y sprites colisionadores inversores

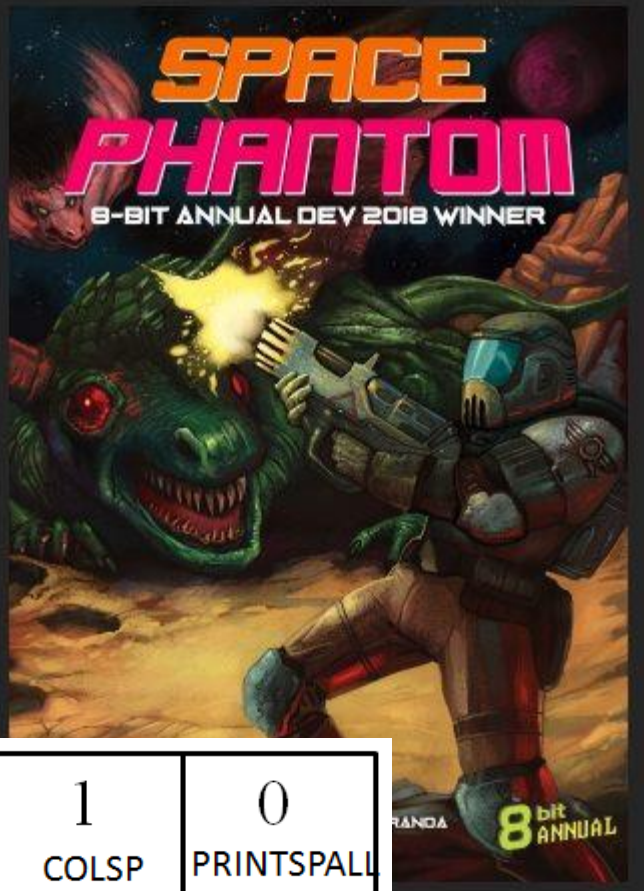

```
"IK m     o      JG"
" IGGGGGH IGGGGHq'
" z   c      xb   '
" C CCGK          d'
" F oC IDD DDDDD  '
" F   C  F        F'
" Fv C JHz b xoF  '
" F   C IGGGGGGGH '
" IGGH z     a   xw'
"        EEEEEoEEE "
```

Un nivel de happy Monty ocupa 160 bytes, incluyendo enemigos, inversores, oro...

Flags de estado con otros usos

El enemigo esta lejos pero es colisionable porque le puedes matar
Sin embargo al colisionar contigo no te mata hasta que esta cerca

En la ruta puedes cambiar un flag de estatus que no uses, como el de MOVER, para indicarle a la rutina de colision si debe o no matarte



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

Físicas simuladas con rutas

Saltos:

En lugar de usar la ecuación de Newton puedes definir una ruta en la que la Vy comienza en -5 y va disminuyendo fotograma a fotograma. Al llegar a la posición cenital, se cambia la imagen para que se borre a si mismo en su parte superior y la velocidad Vy se torna positiva, y poco a poco va aumentando. Es como aplicar la ecuación de Newton, pero sin cálculos.

Mas color con sobreescritura

2 tintas para el fondo (2 colores)
\+ 14 tintas para sprites con sobreescritura (7 colores)

TOTAL: 16 tintas, 9 colores

2 tintas para el fondo (2 colores)
6 tintas para sprites con sobreescritura (3 colores)
\+ 8 tintas para los demás sprites (8 colores)

TOTAL: 16 tintas, 13 colores

4 tintas para el fondo (4 colores)
\+ 12 tintas para sprites con sobreescritura (3 colores)

TOTAL: 16 tintas, 7 colores

Invocar sin parámetros

PRINTSPALL, COLAY, MOVER, MOVERALL, COLSP, STARS

Invocar con CALL

Alterar estado en rutas

Modifica el estado de un sprite para que no sea colisionable en ciclos pares o impares, de modo que aligeres el trabajo de COLSPALL y con 8 sprites solo tenga que calcular las colisiones con 4 de ellos

Tendras que inicializar a 4 de los 8 enemigos con |ROUTESP,<id>, 1

```
ROUTE1; derecha
;------------------------
    db 1,0,1
    db 255,128+8+2+1,0 ; cambio de estado a colisionable
    db 1,0,1
    db 255,128+8+1,0 ; cambio de estado a no colisionable
    db 0
```

El mismo truco sirve para aligerar PRINTSPALL si se usa en el flag de impresión

Te falta memoria

BASIC te da memory full si no le das 5KB de margen pero a lo mejor no necesitas tanta RAM para tus variables

```
10 MEMORY 19999
20  LOAD  "!juego.bin":  rem  carga  datos  a
partir de la 20000
30 CLEAR: MEMORY 25000 : rem asi le damos mas
margen, por ejemplo.
40 RUN “!juego.bas”: rem la primera línea de
juego.bas debe ser memory 19999
```

Empieza
la aventura de programar!

2 consejos:

1. No escatimes tiempo en la creación de gráficos
2. Empieza por hacer algo sencillo y hazlo crecer


https://8bitsdepoder.blogspot.com
https://github.com/jjaranda13/8BP
FROGGER
LIVES
SCORE
TIME
LEVEL
HAPPY MONTY
EXIT
SCORE
HIGH:
LEVEL: 0
FPS:
SCORE 9 LIVES: 3
3D RACING ONE
SPEED
MOTOR
TIME: 52
SCORE
10
FUEL
LEVEL
1
LIVES
8BP
Polvoranca Forrest