INTRODUCCION A 8BP
Y como programar:

Durante la charla taller de 8BP sobre el juego ERIDU, podrás conocer la "historia prohibida" que no te han contado sobre el origen extraterrestre de la humanidad…

Agenda

```
Ready
list
1 Historia de Scramble
2 Introduccion a 8BP y logicas masivas
3 la "historia prohibida" de Eridu
4 Programacion de Eridu
5 Fase 5
Ready

RUN
```

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

“scrambler” de PuzCPC, 2019 (AMSTRAD). excelente

“Star avenger” de Kuma computers (AMSTRAD)
Scroll brusco

“killer cobra” (AMSTRAD)
Brusco y rapidisimo

“Penetrator” (ZX spectrum)
Suave y jugable

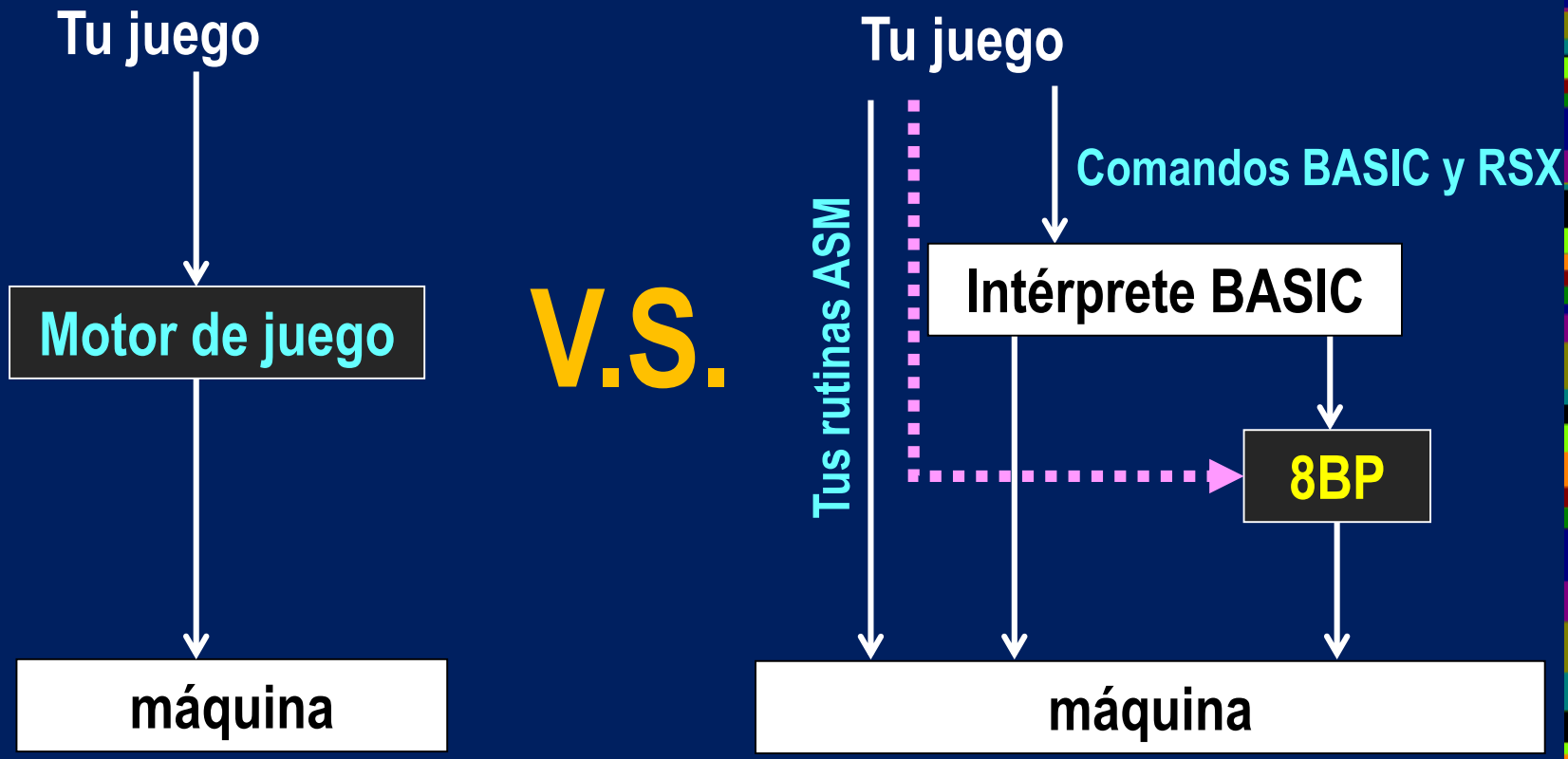
“SKRAMBLER” (C64)
Brusco a pesar del HW

“SKRAMBLER” (MSX)
suave y jugable


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

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)

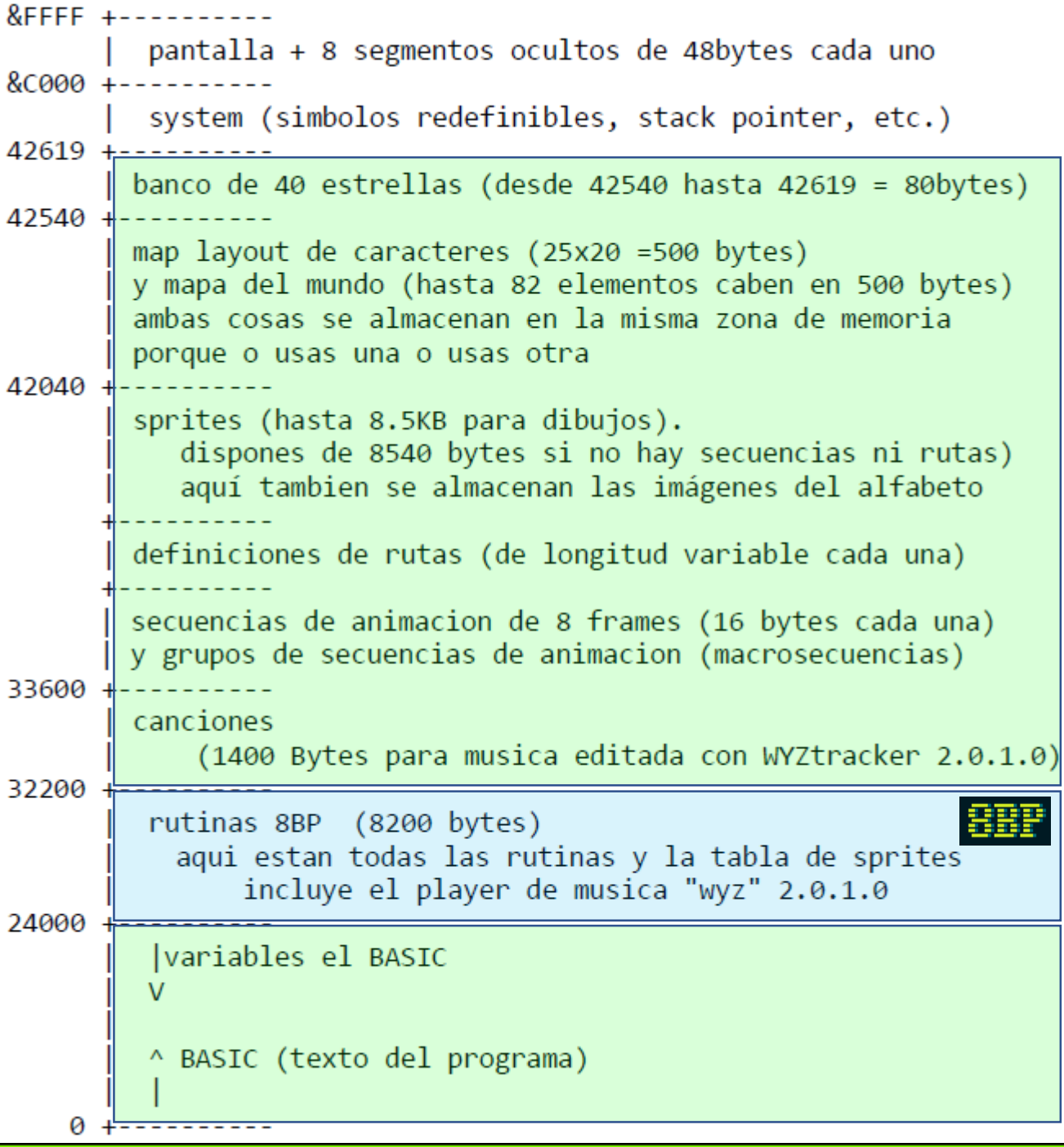
8BP Memory map

8BP Sólo ocupa 8 KB y te proporciona 27 comandos

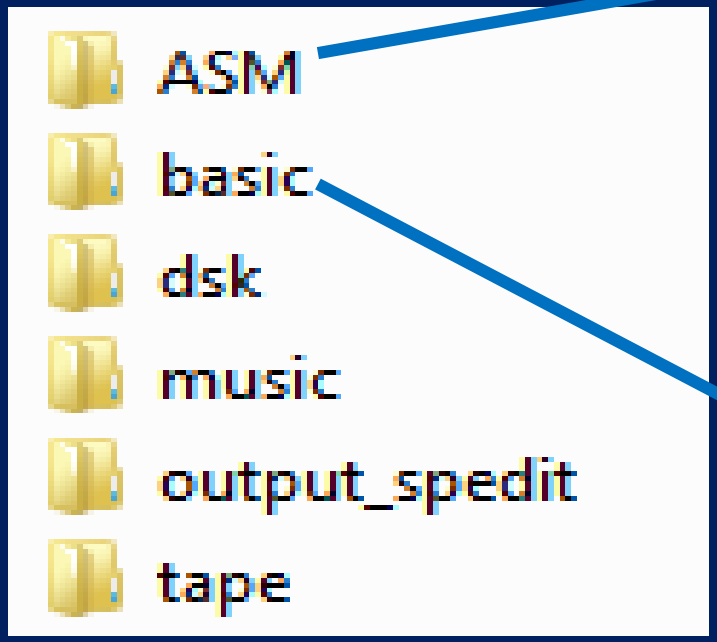
24 KB libre para BASIC

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
Frogger.bas

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 animacion 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.

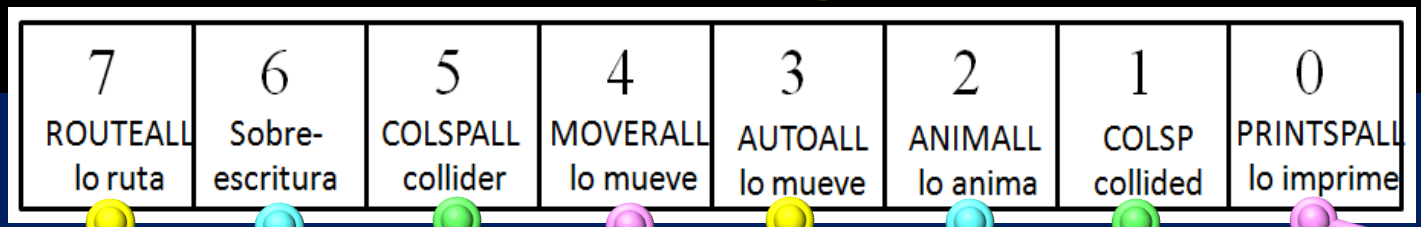
Sprites

- 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

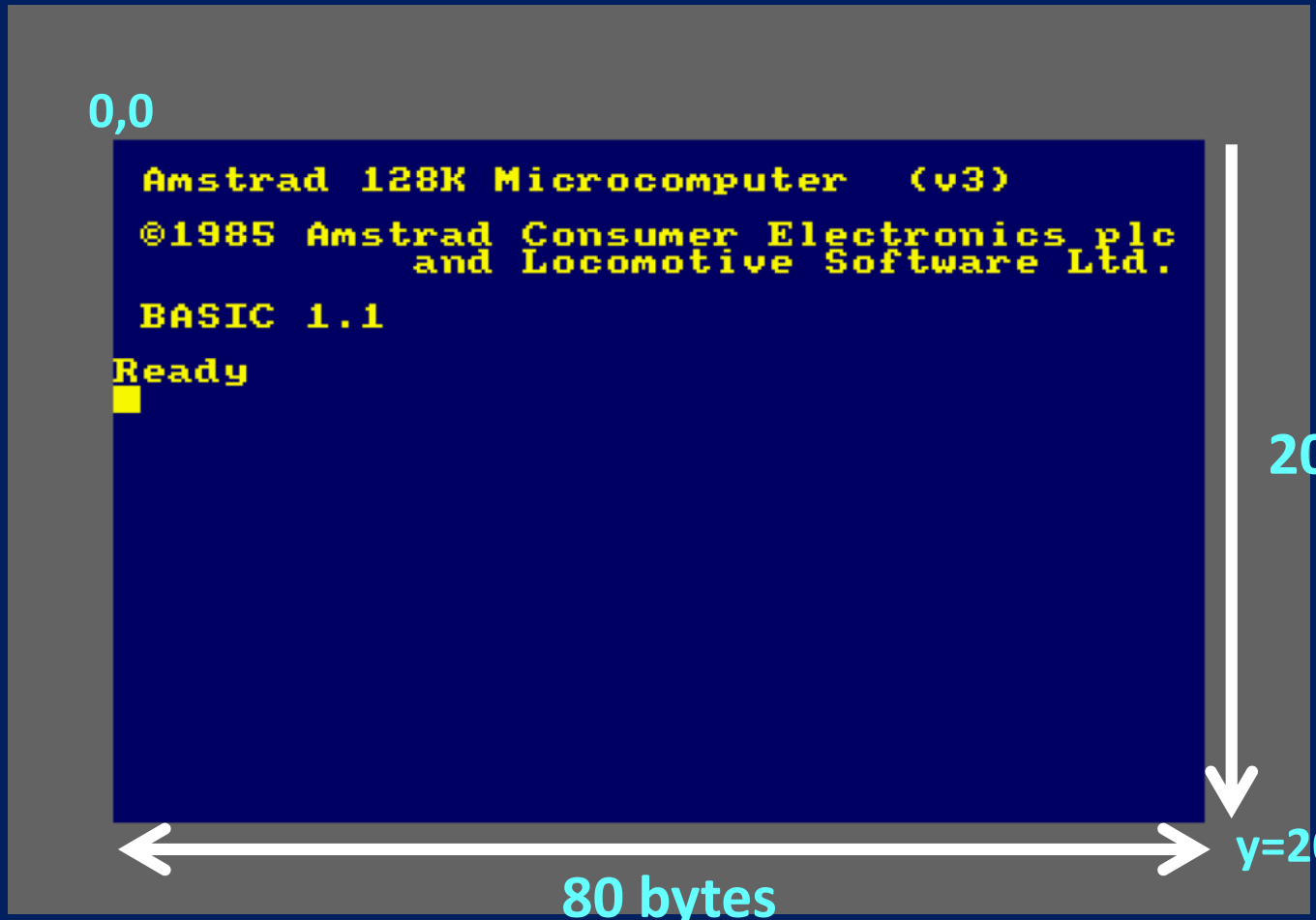
Puede moverse de forma relativa solidariamente con otros sprites
MOVERALL,dy,dx

Puede ser colisionador
COLSPALL,@collider,@collided

Puede tener sobreescritura

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

Coordenadas en 8BP



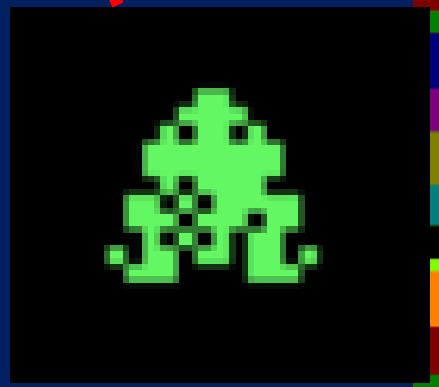
1 BYTE = 2 pixels de MODE 0

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
```

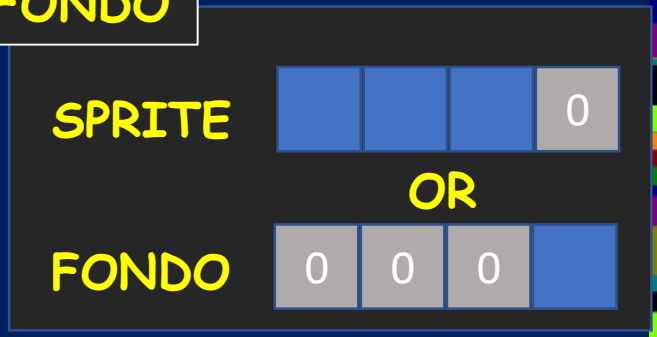


Sprites: sobreescritura

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

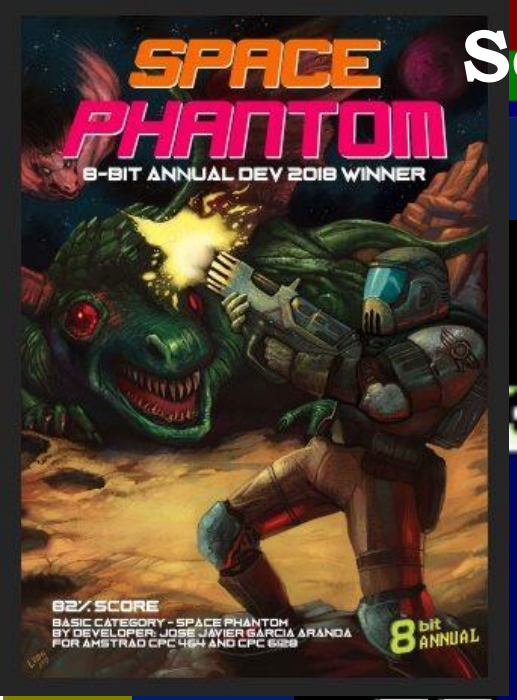
Se reduce el número de colores
9 en mode 0
3 en mode 1

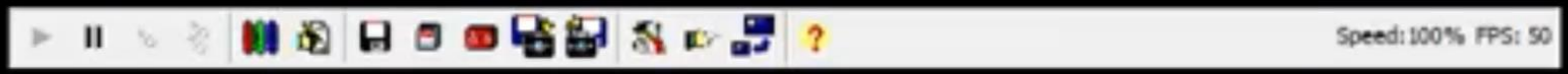


- Hay truco para usar mas de 9 colores en MODE 0 con sobreescritura
- 8BP también permite elegir 1 o 2 bit de fondo

Sprites: sobreescritura

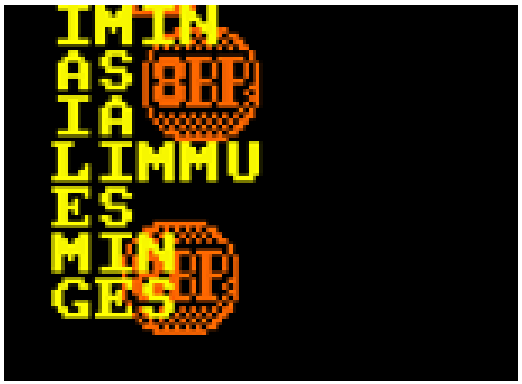
Sobreescritura y animación por rotación de tintas

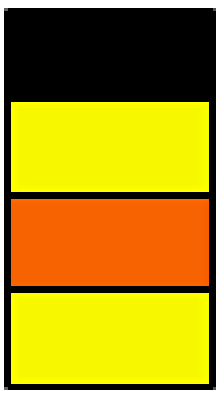




Sprites: sobreescritura

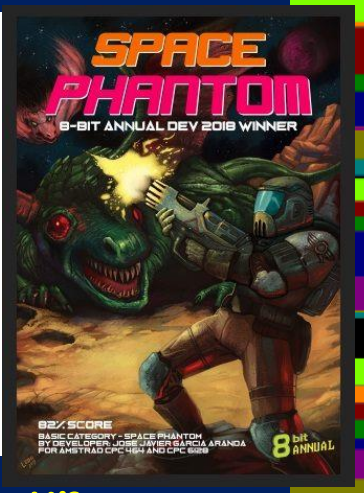
Con la misma técnica podemos pintar por detrás del fondo





00
01
Colores de fondo

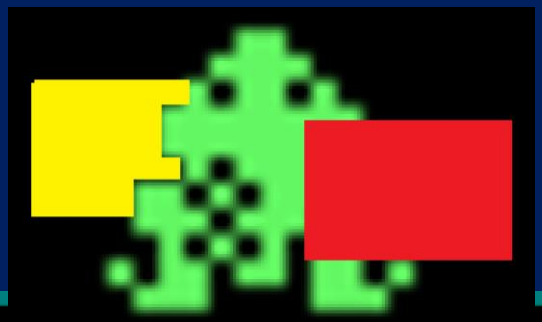
10
11
Color de sprite



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

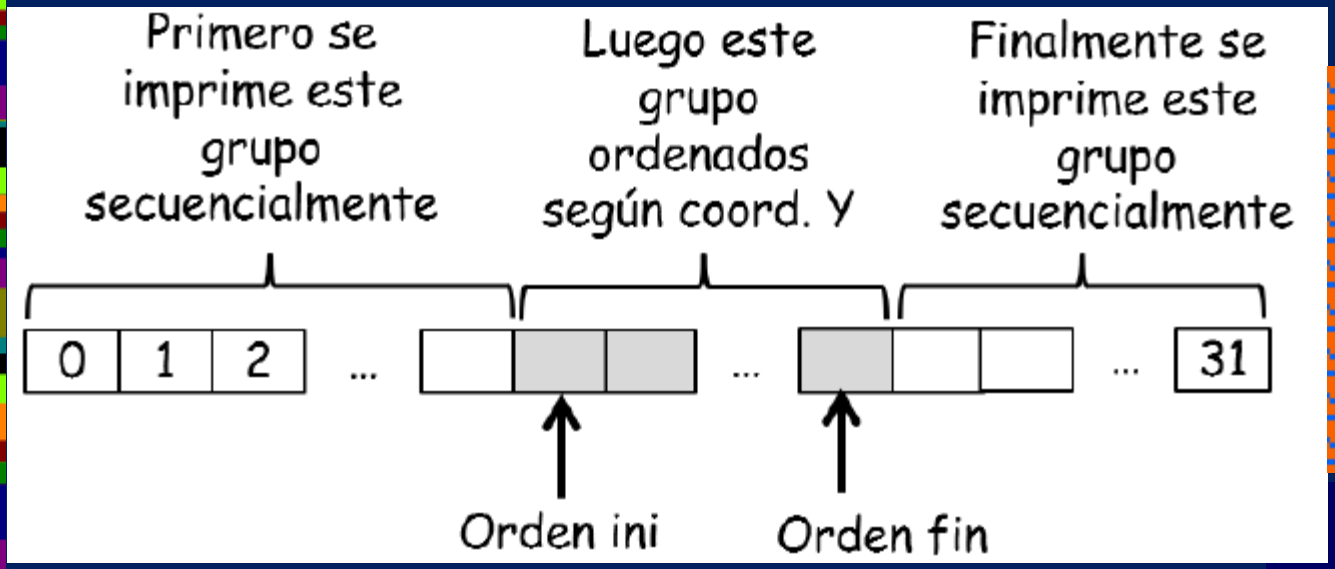
En este ejemplo de paleta con 4 colores de fondo y 3 para sprites, el color verde puede pasar por delante del color amarillo pero a la vez pasa por detrás del color rojo



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

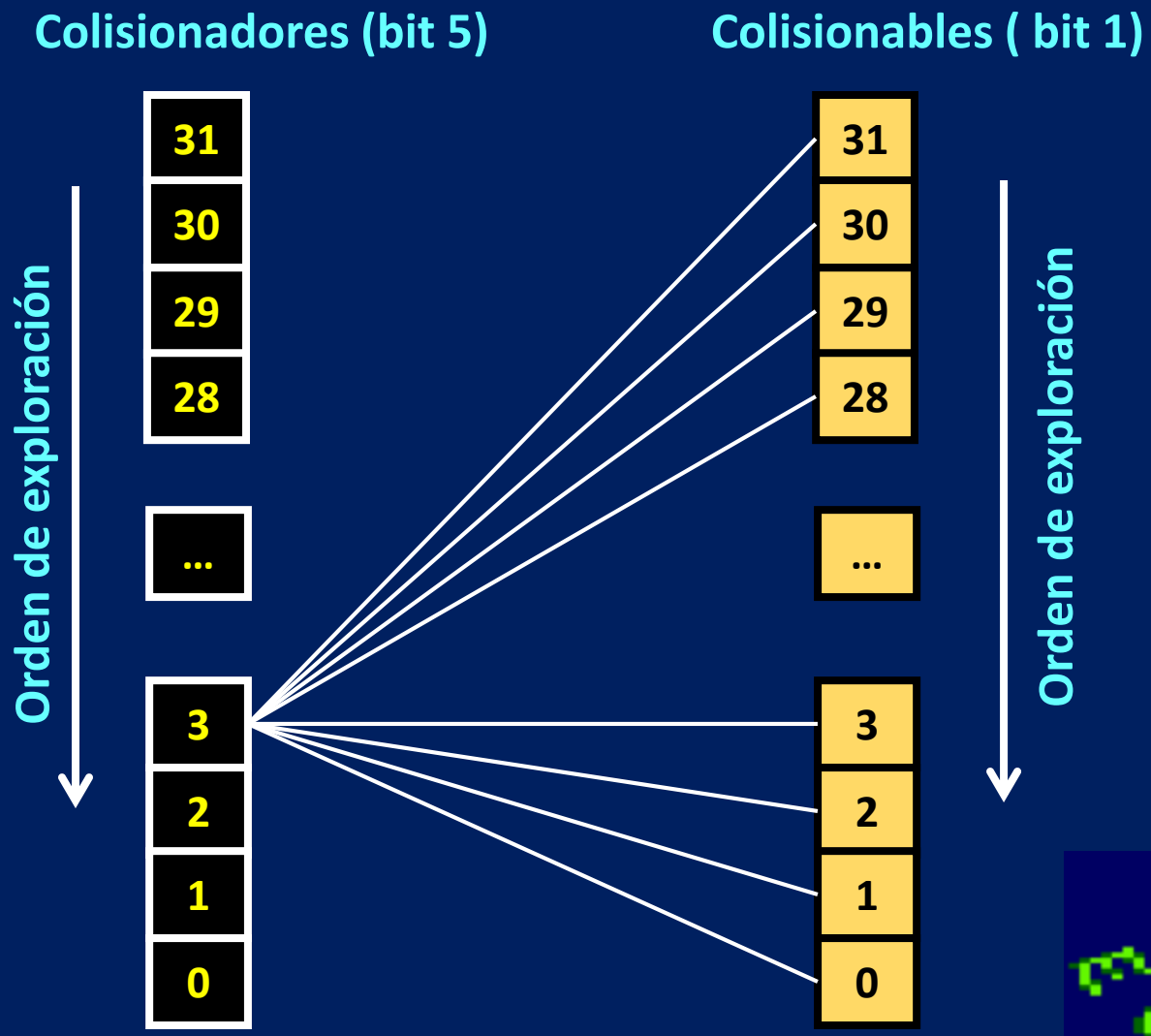
PRINTSPALL,2 : ordenacion parcial de sprites basado en Ymax

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

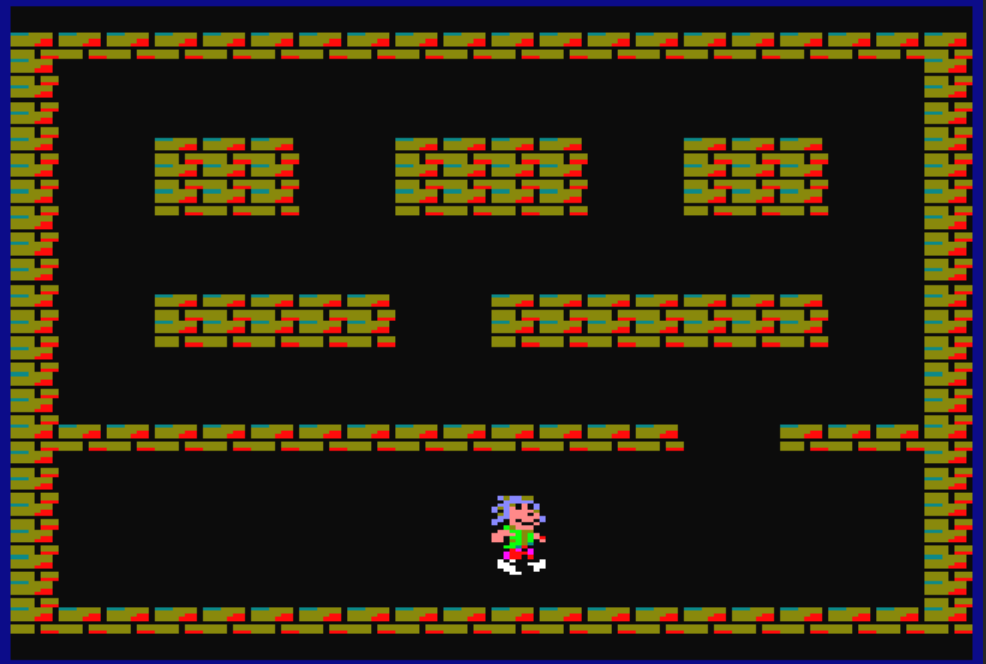
Sprites: colisiones

Funcionamiento de |COLSPALL



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, sp, @col
```

0 = no hay colisión

1 = hay colisión

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

32 = no hay colisión
<32 hay colision

32 = no hay colisión
<32 hay colision

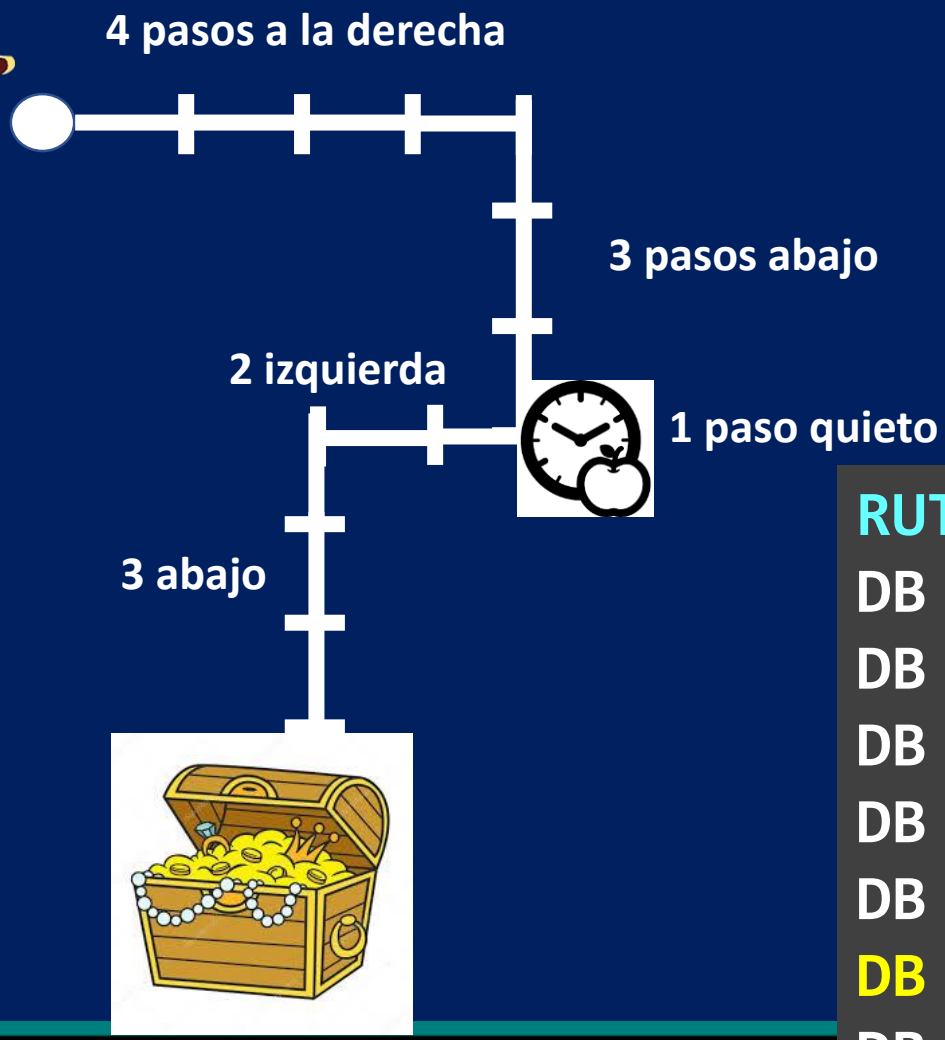
Sprites: secuencias de animacion

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

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
```

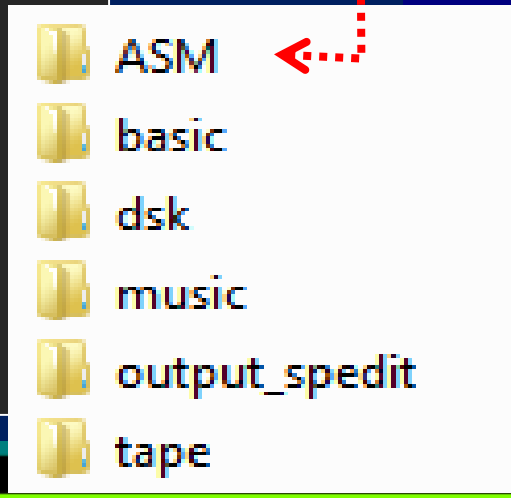
Sprites: concepto de ruta

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
```

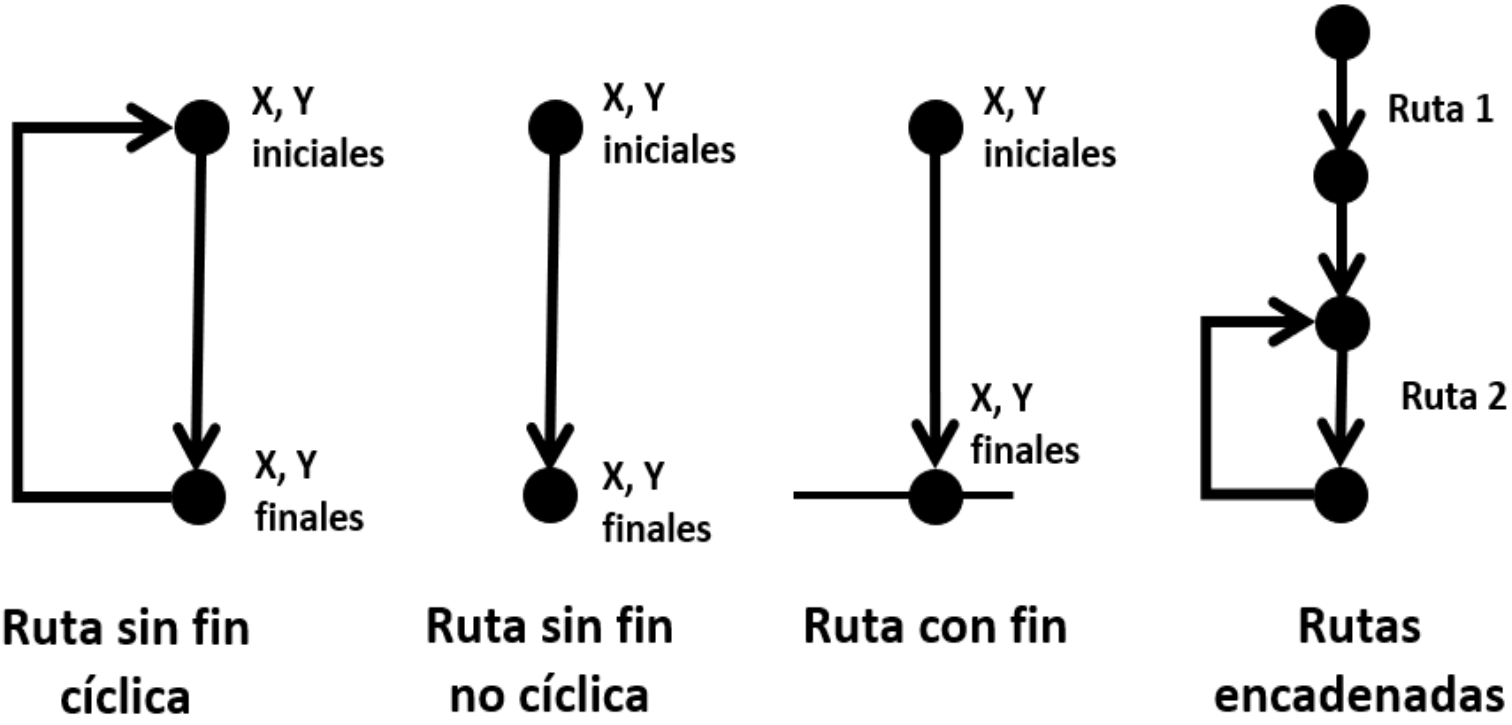
```
|SETUPSP,pirata,15, 0
```



Sprites: concepto de ruta

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
251	Pasa al siguiente frame de la animación.	DB 251,0,0 Se anima el Sprite. Los dos ceros son de relleno

Tipos de rutas de sprites



Ruta sin fin cíclica

Ruta sin fin no cíclica

Ruta con fin

Rutas encadenadas

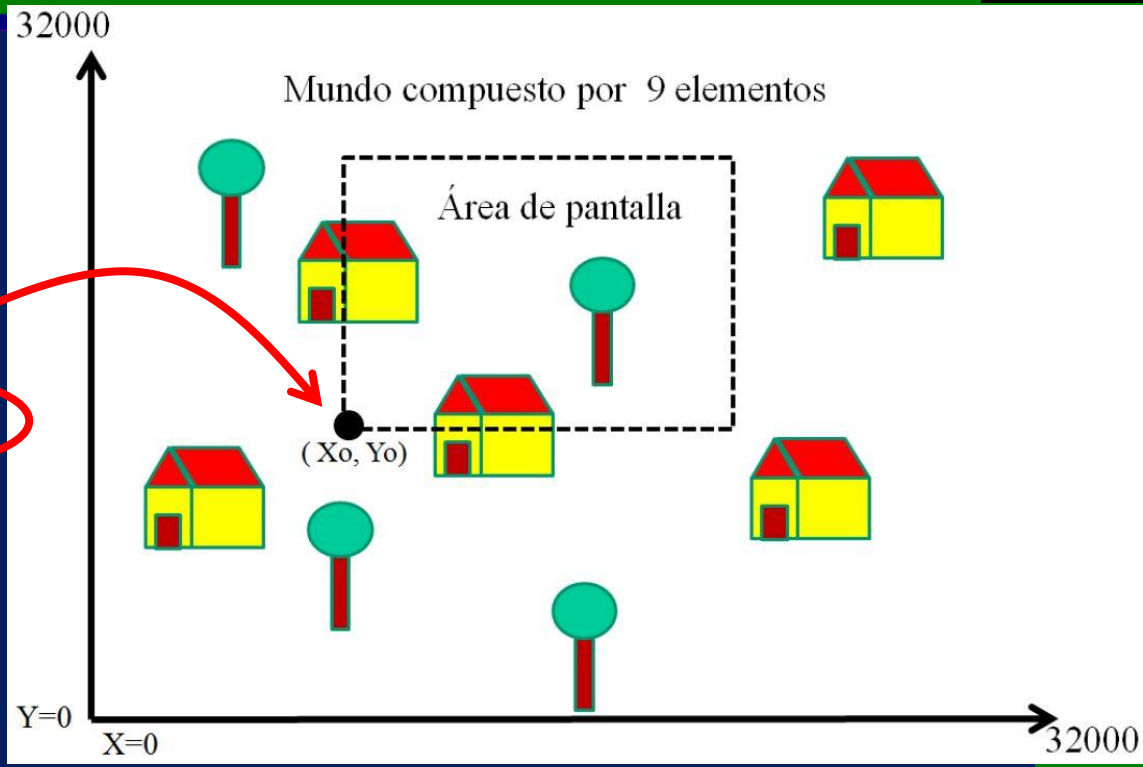
Una ruta puede ocupar 255 bytes como mucho de modo que puede llegar a interesarte construir rutas mas largas encadenándolas

Scroll multidireccional

Scroll basado en una "ventana deslizante"

Comando

`|MAP2SP, Yo, Xo`



El mundo mide 32000 x 32000

OJO: las coordenadas para imprimir sprites van al revés en el eje vertical

Scroll multidireccional

31
30
29
28

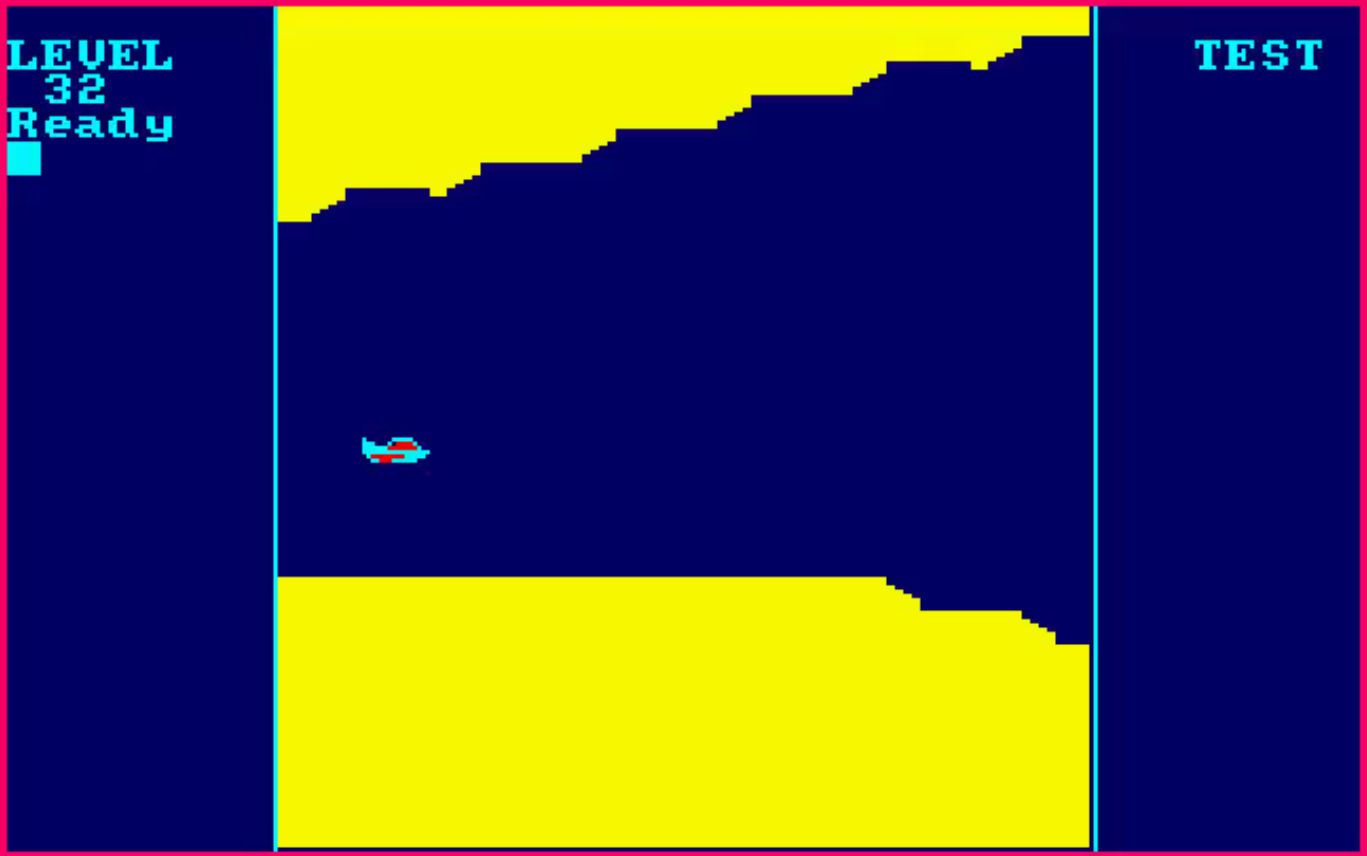
muñecos

...

3
2
1
0

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

Scroll multidireccional: 32 fps!

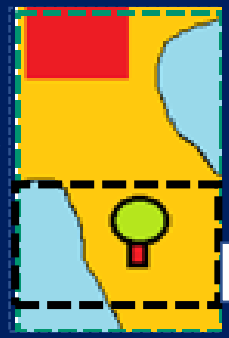


Speed:100% FPS: 50

Scroll multidireccional

Mapa completo en dirección 23000 (por ejemplo)

Mapa parcial ubicado en la dirección 42040

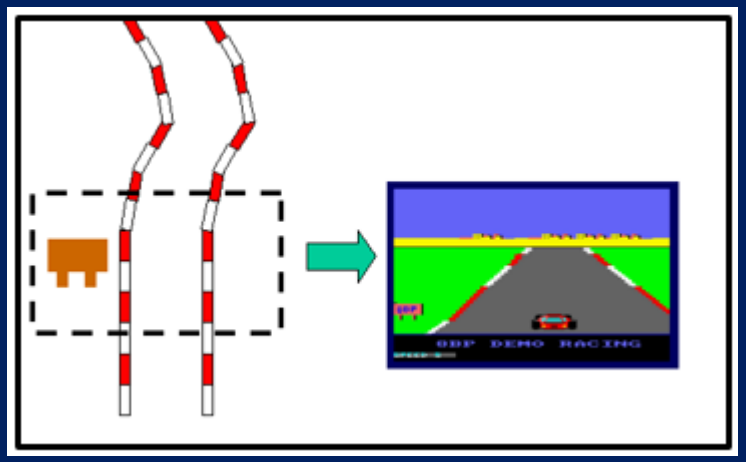


UMAP,23000,24000,200,500,0,80

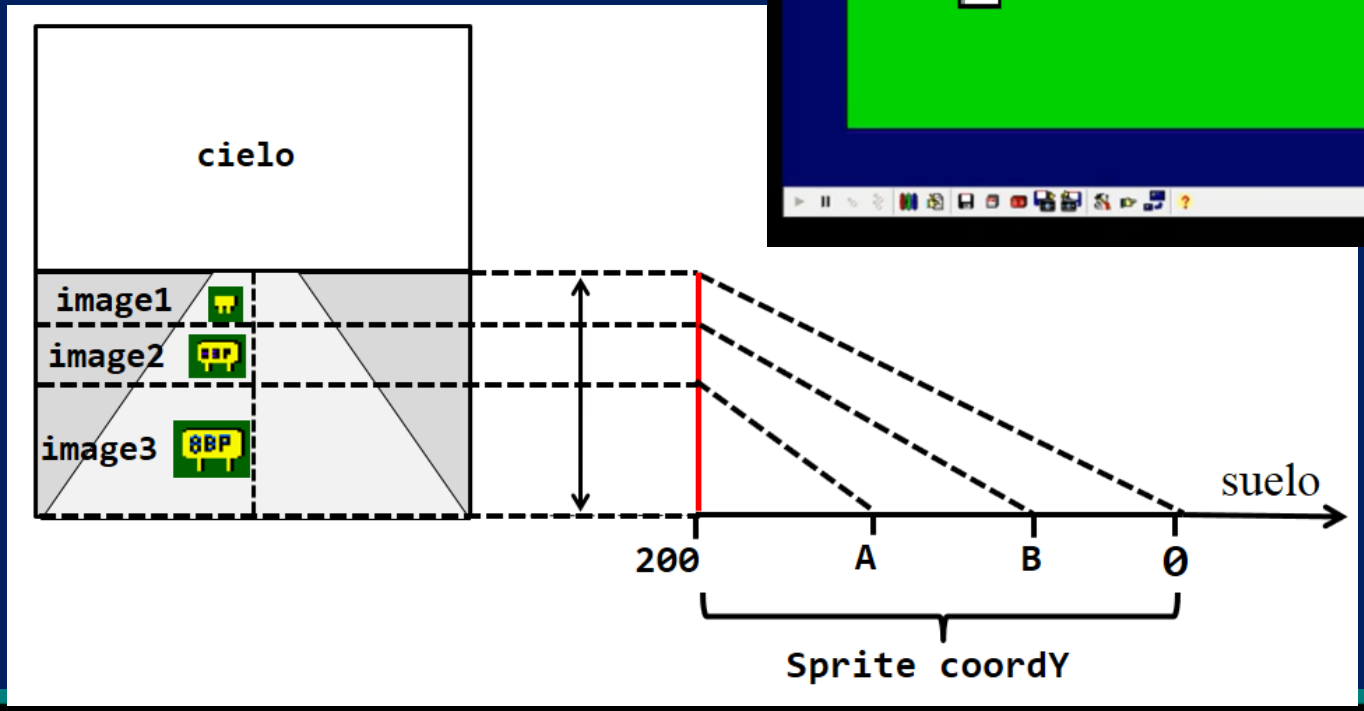
MAP2SP,y,x

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



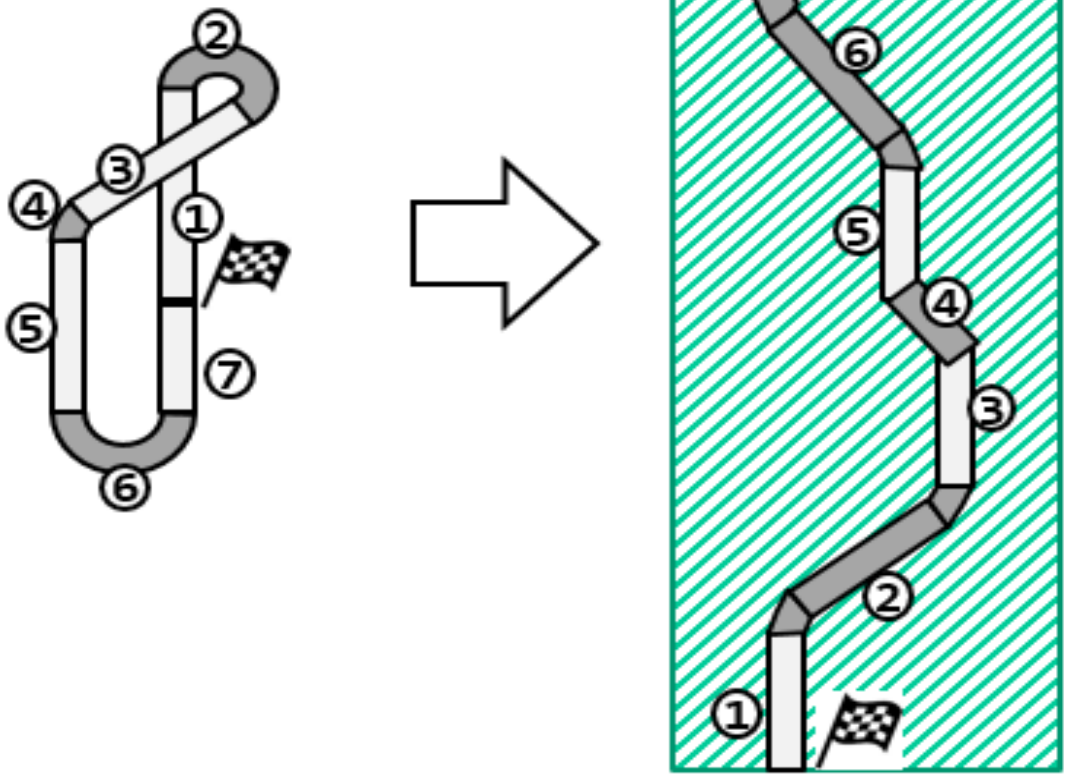
El comando 3D nos permite recorrer un mundo 2D proyectado

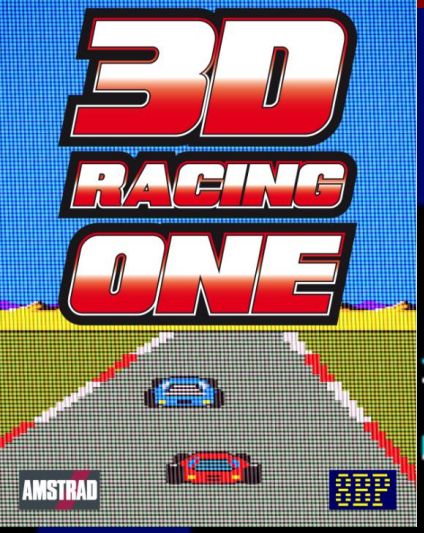


CONCEPTO:

Sólo es 3D el plano del suelo.

Las curvas son una ilusión.

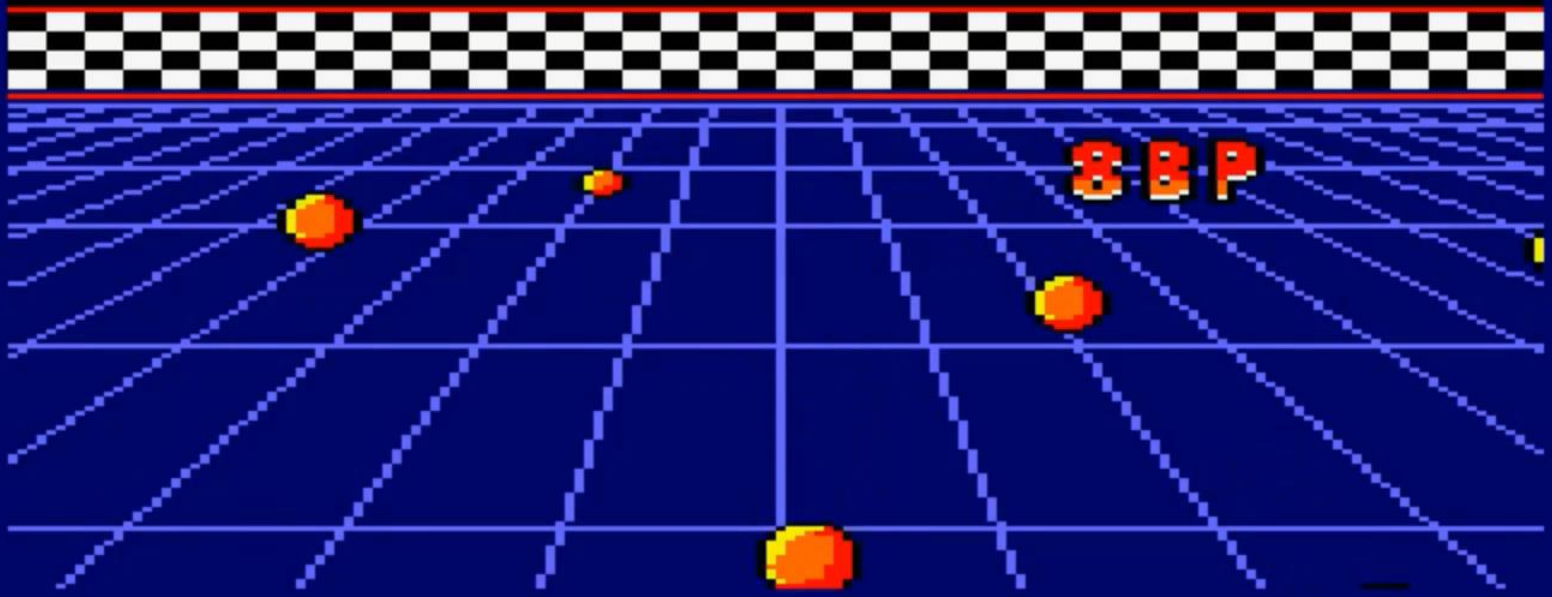




3D RACING ONE

JOSE JAVIER GARCIA ARANDA 2018

LY BASIC PROGRAM CREATED WITH BBP



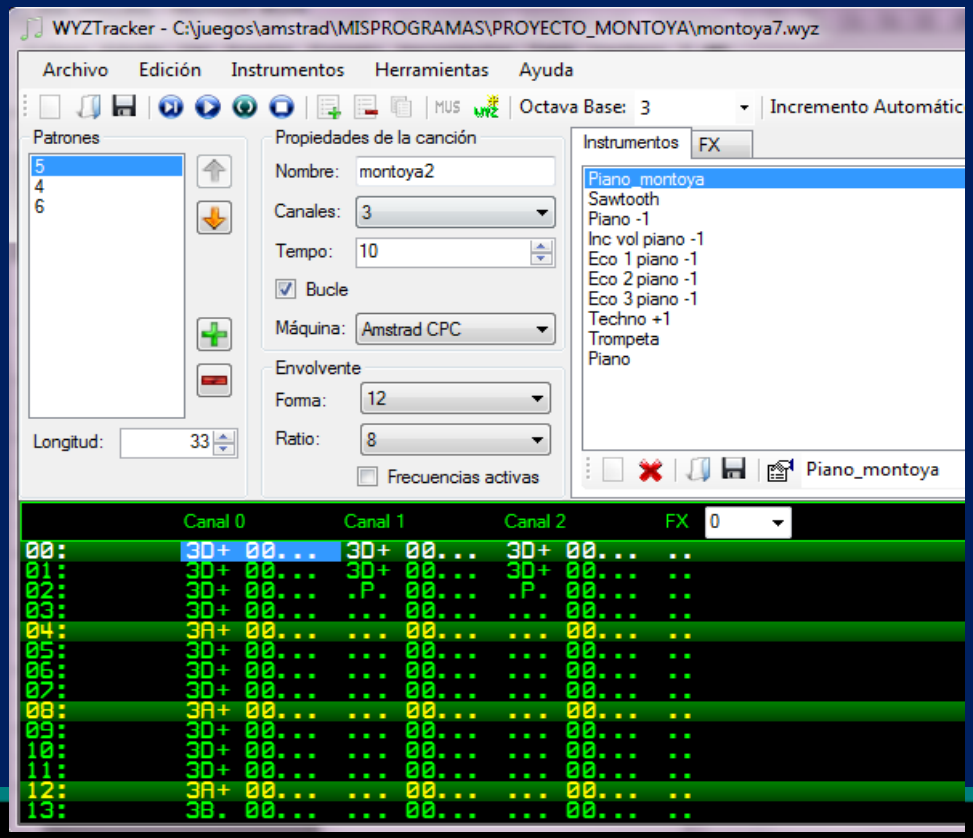
Speed:100% FPS: 50

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

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

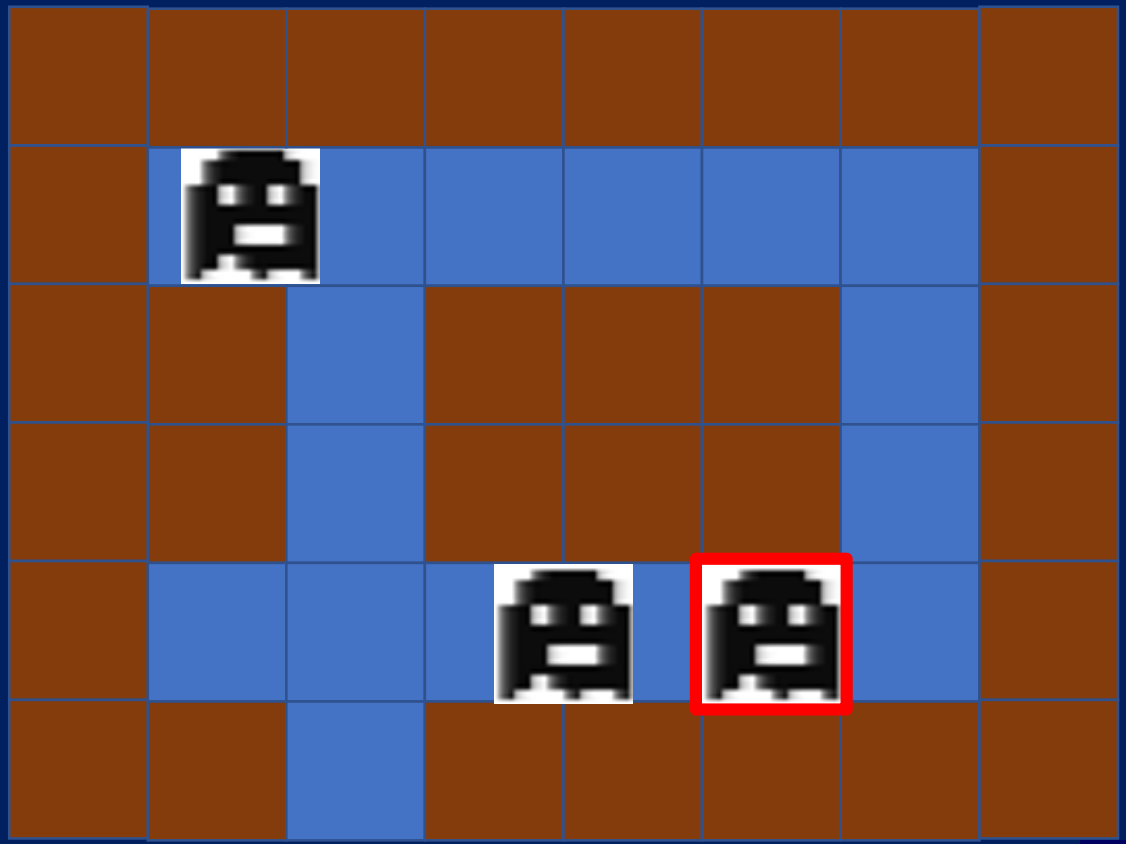
- En cada ciclo de juego ejecutar un subconjunto de comprobaciones
- Todas las comprobaciones tardarán varios frames pero no importa

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**"

Lógicas masivas: ejemplos

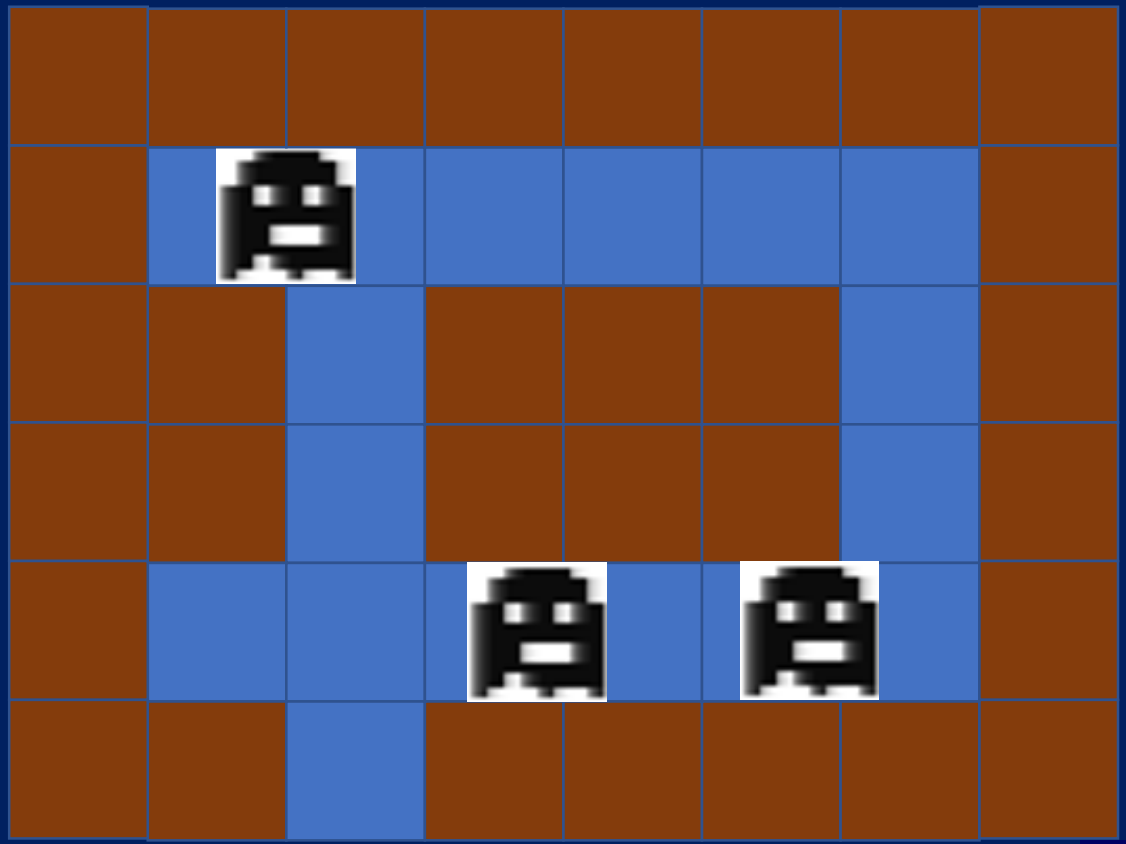
Ciclo= 0



massive logics

Lógicas masivas: ejemplos

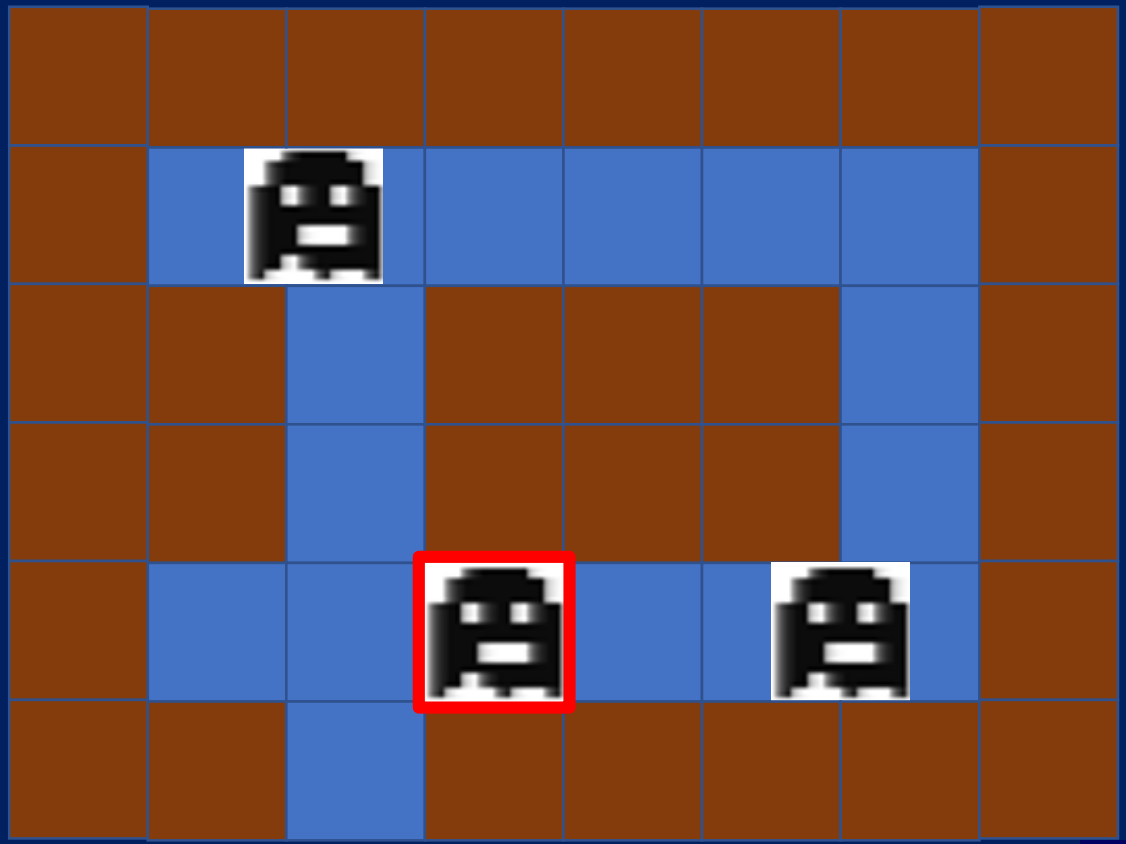
Ciclo= 1



massive logics

Lógicas masivas: ejemplos

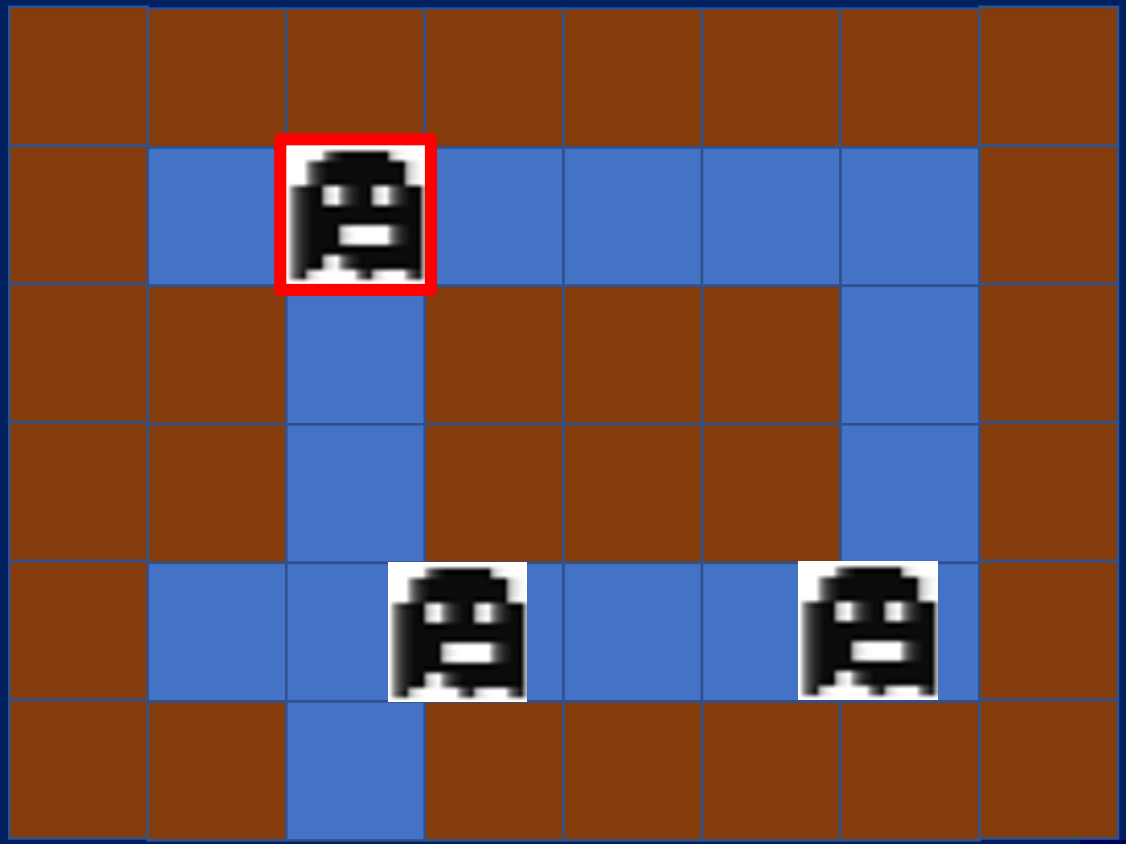
Ciclo= 2



massive logics

Lógicas masivas: ejemplos

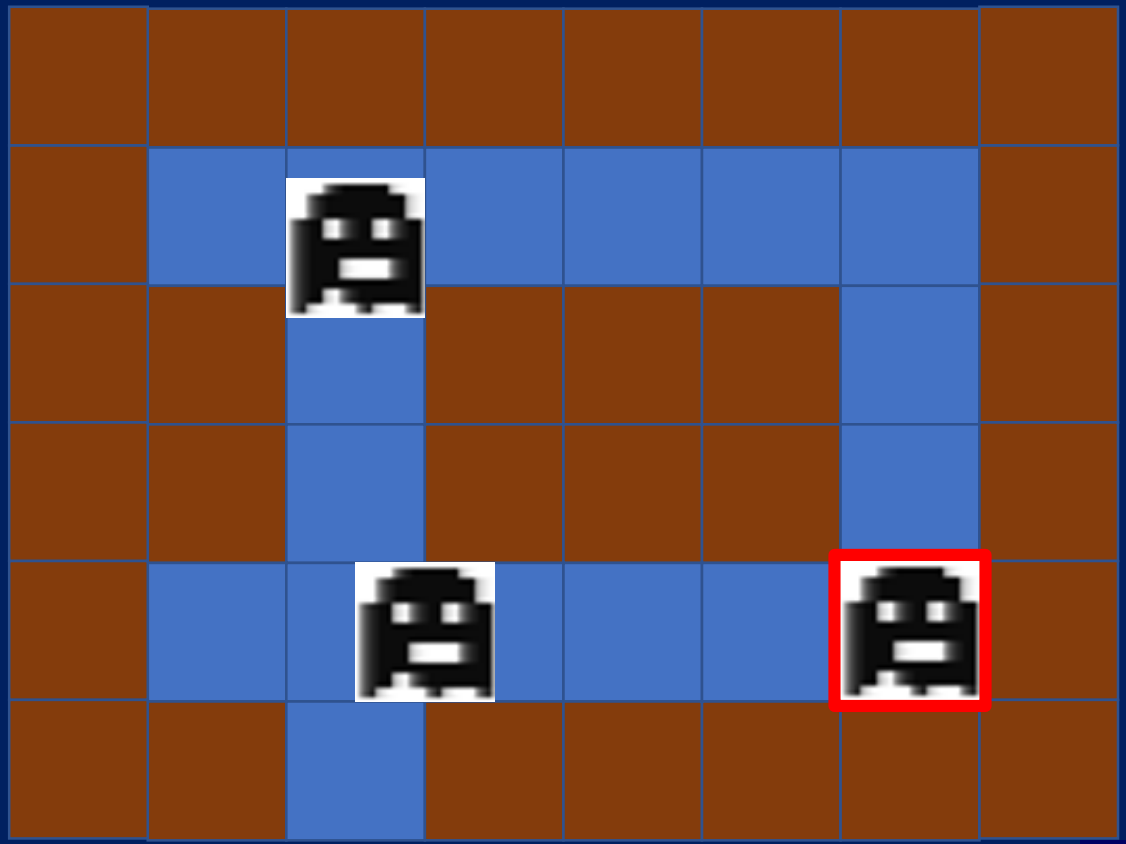
Ciclo= 3



massive logics

Lógicas masivas: ejemplos

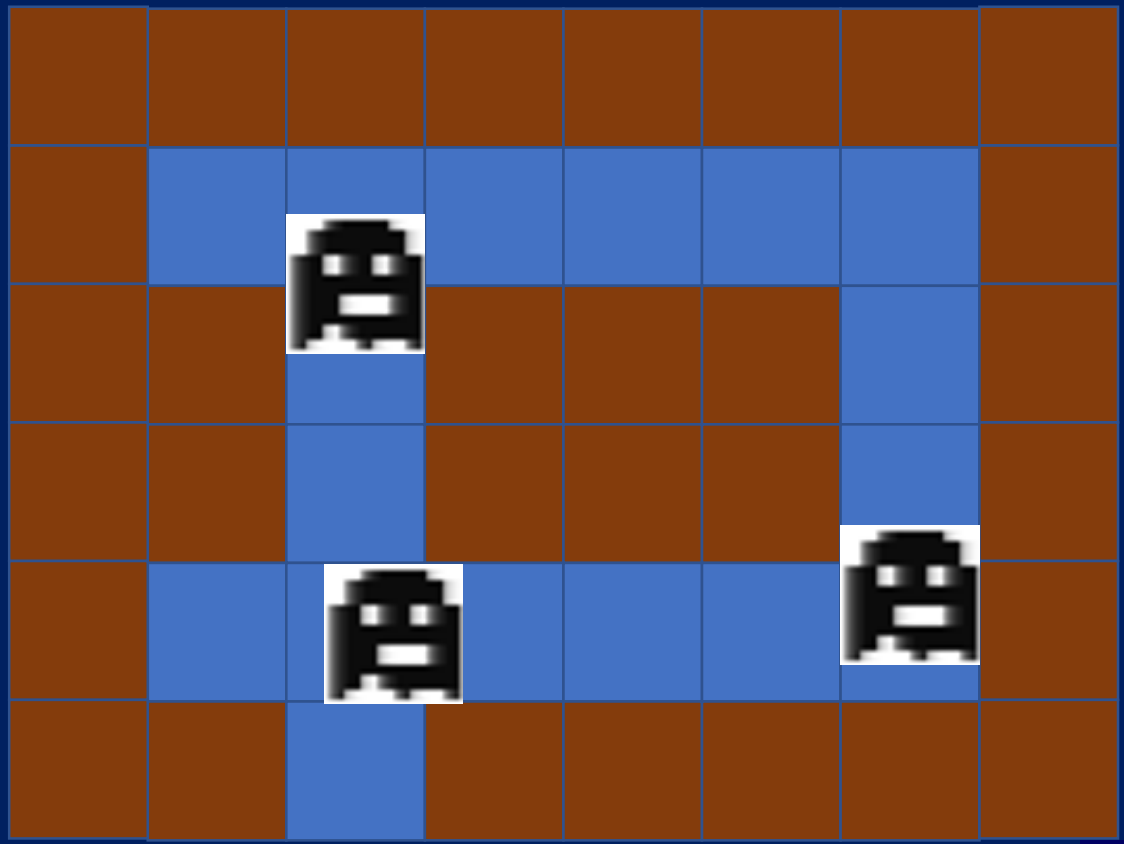
Ciclo= 4



massive logics

Lógicas masivas: ejemplos

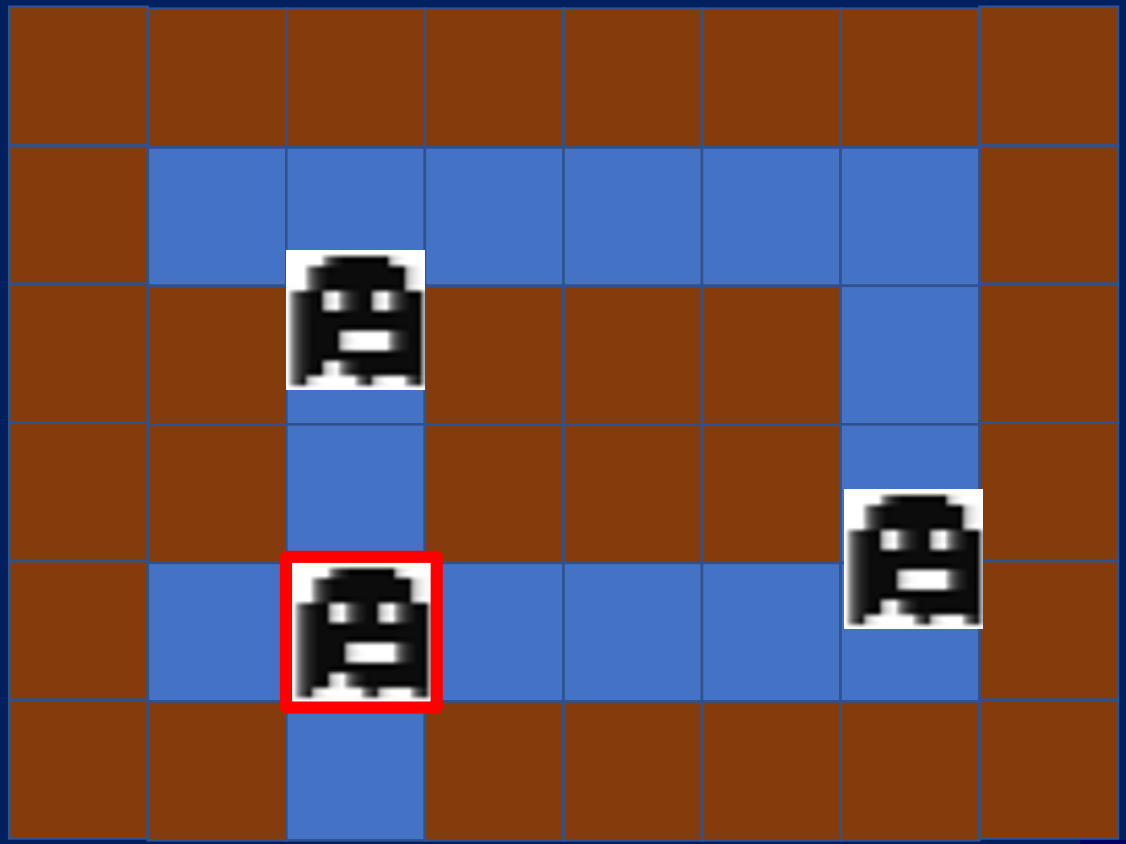
Ciclo= 5



massive logics

Lógicas masivas: ejemplos

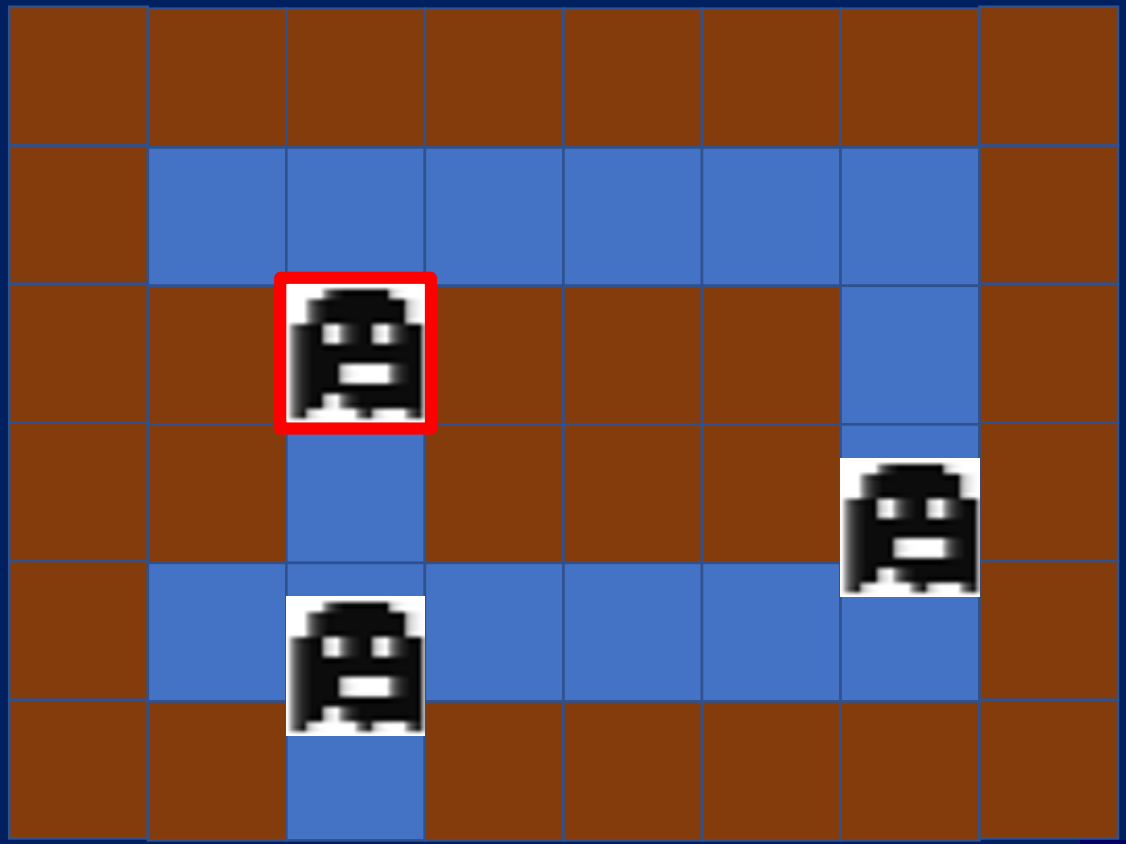
Ciclo= 6



massive logics

Lógicas masivas: ejemplos

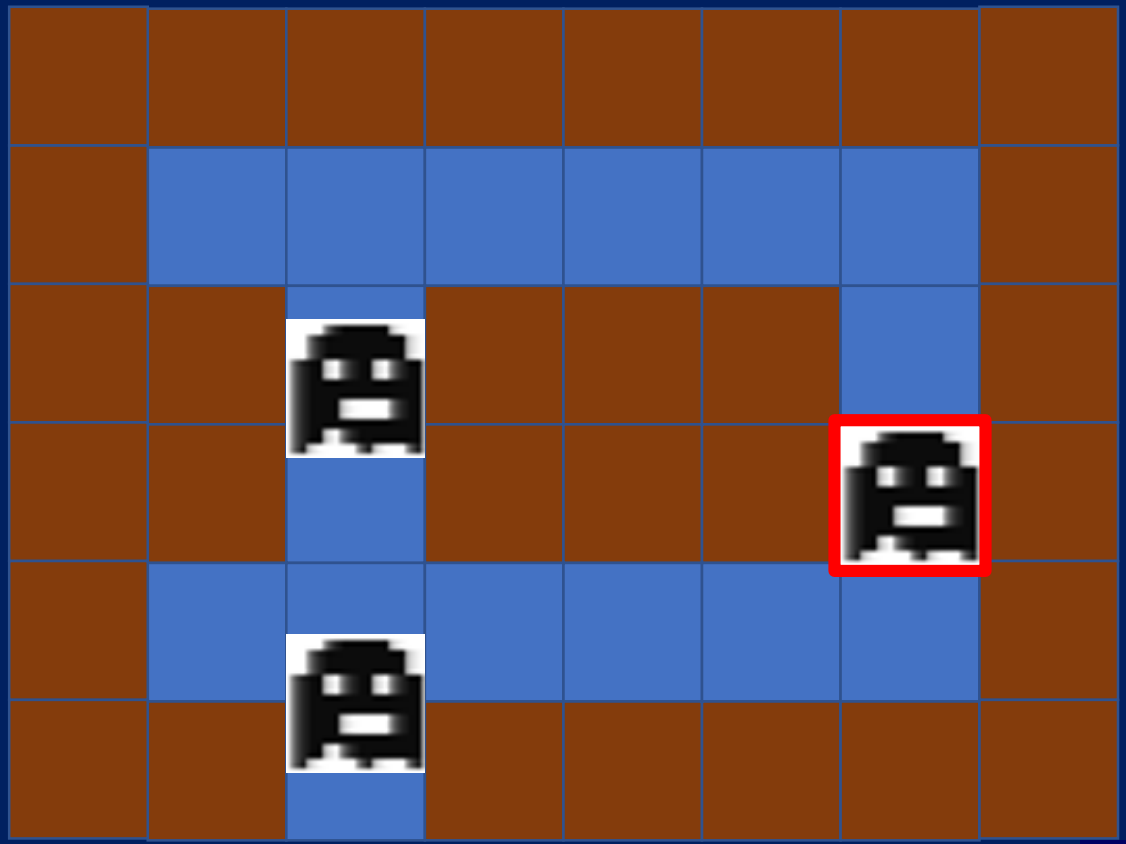
Ciclo= 7



massive logics

Lógicas masivas: ejemplos

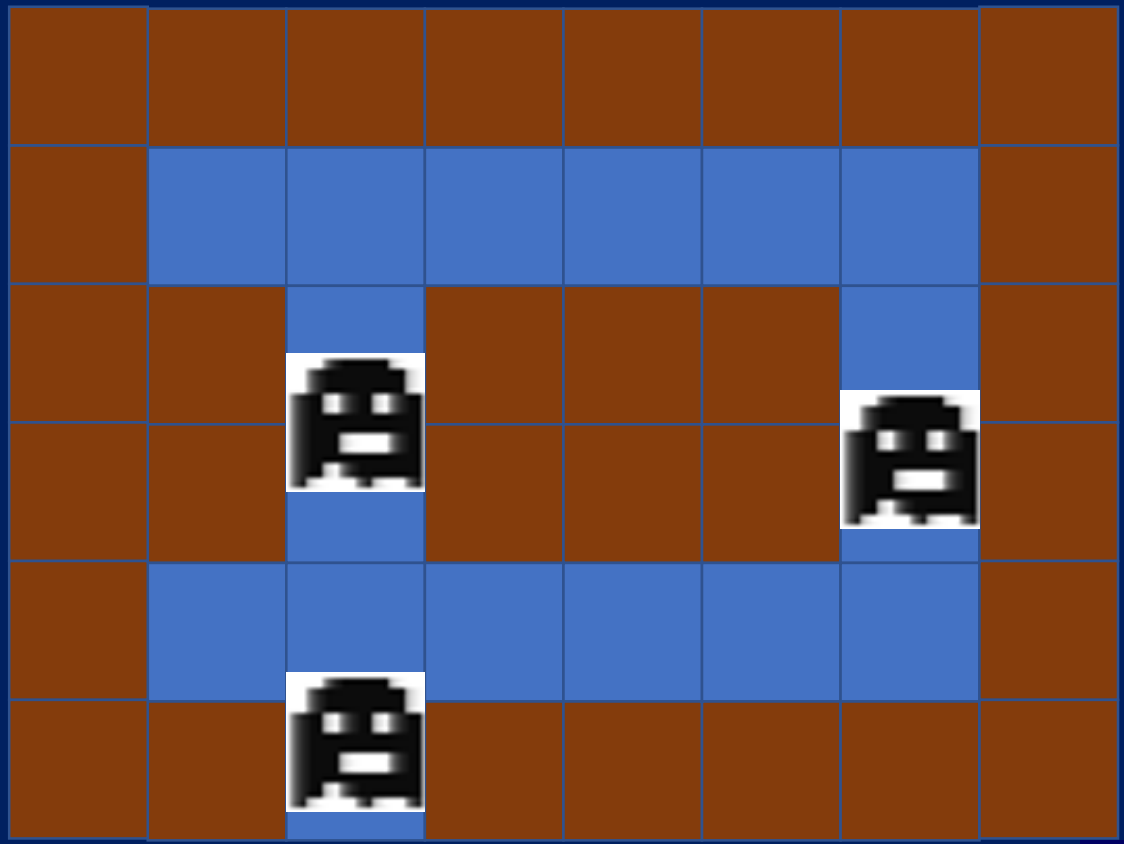
Ciclo= 8



massive logics

Lógicas masivas: ejemplos

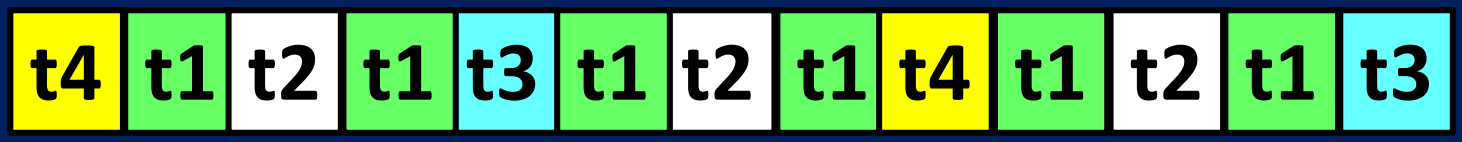
Ciclo= 9



massive logics

Lógicas masivas: ejemplos

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 ---
```


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

La historia “prohibida” que no te han contado sobre el origen de la humanidad

¿Y si existió una civilización “madre” anterior a todo lo que nos han contado?

III dinastía

Saqqara (Egipto).
Pirámide
Escalonada de Zoser

Maidum . Pirámide de
Huni-Snefru

IV dinastía

Pirámides de Giza.

Giza. Gran Pirámide de Keops

2.600.000 bloques
de 5 Tn, a bloque por
cada 3 minutos
durante 20 años???

V dinastía

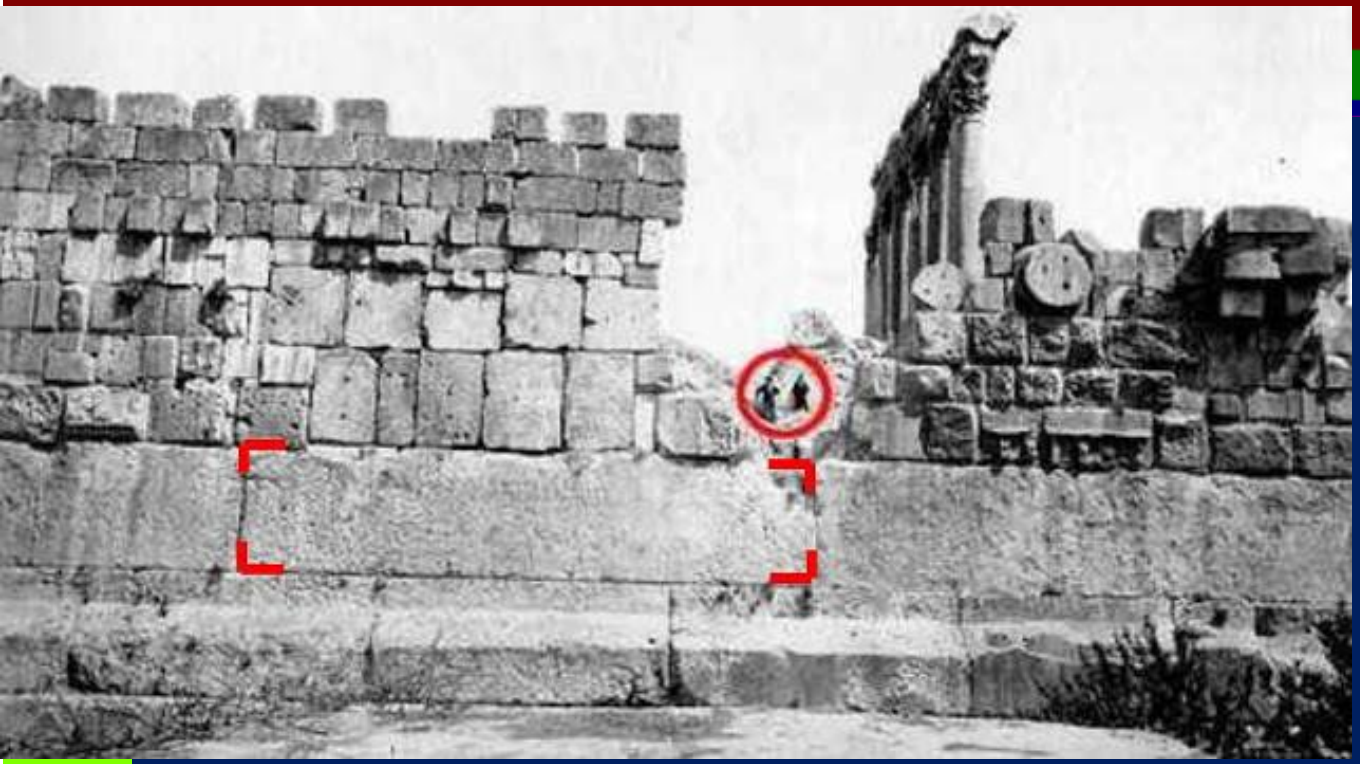
Abusir. Pirámide de Neferirkare

VI dinastía

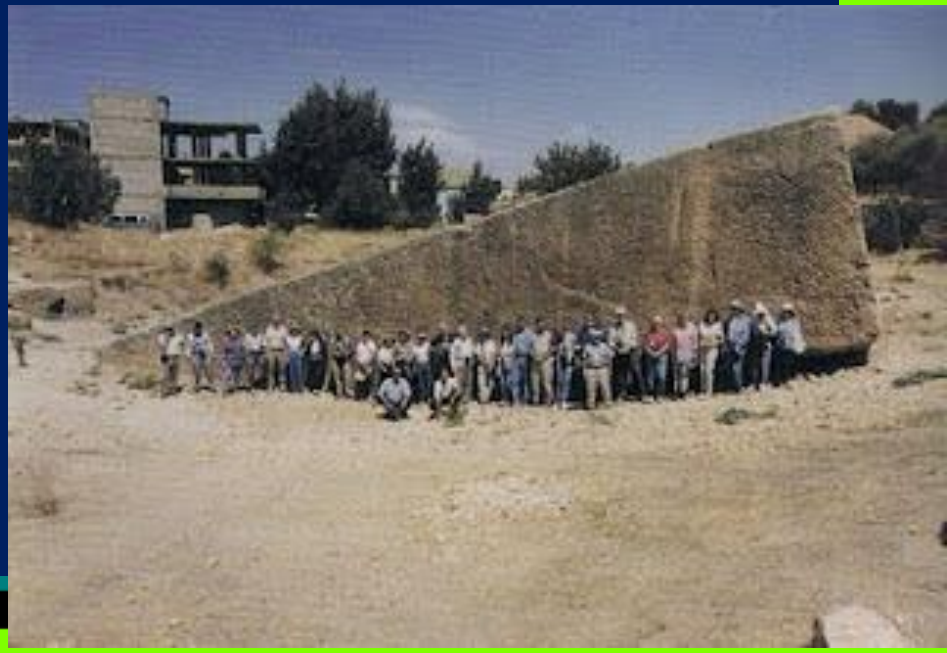
Saqqara sur. Pirámide de Pepi II

XII dinastía

Dashur. Pirámide de
adobe de Amenemhat III.



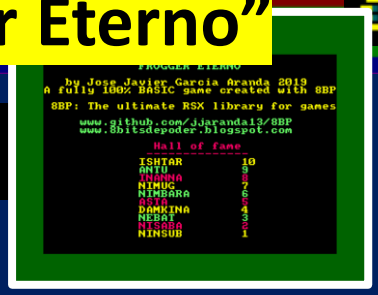
Baalbeck: ¿posible plataforma de despegue de naves extraterrestres?. Cada piedra pesa 1200 toneladas, imposibles de transportar hoy en dia



Eridu, la primera ciudad del mundo, creada por los "Anunnaki", una raza extraterrestre según las tablillas sumerias, hace 400.000 años


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

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

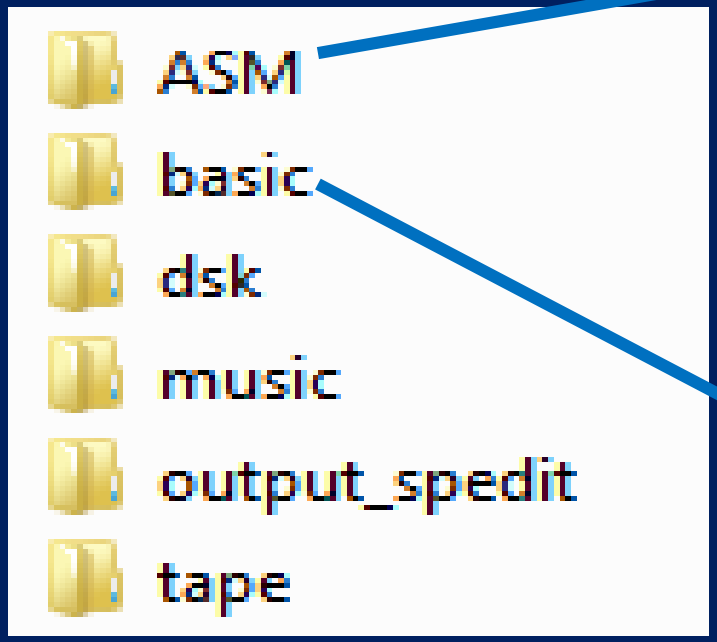
Lógica del programa principal

- 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

Carpetas de un juego 8BP



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

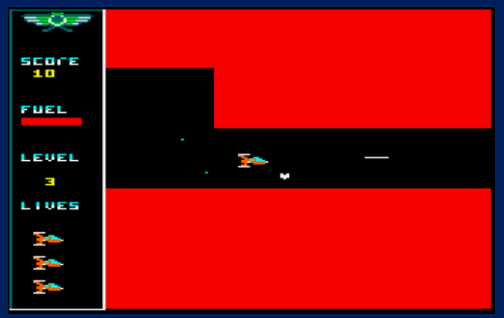
Loader.bas
eridu.bas

Ficheros de mapas y enemigos de Eridu

Map_titulo_eridu.asm

Map_fase1.asm
Enemigos_fase1.asm

Sin mapa ni fichero de enemigos

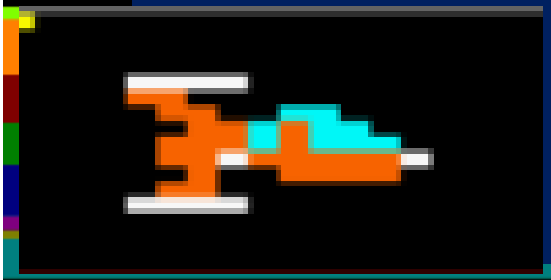


Map_fase3.asm
Enemigos_fase3.asm

Map_fase4.asm
Enemigos_fase4.asm

Fase 5

Diseño de sprites : SPEDIT11



.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 ------
```

Diseño de sprites: fichero “images_mygame.asm”

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

Diseño de sprites: flipeo horizontal

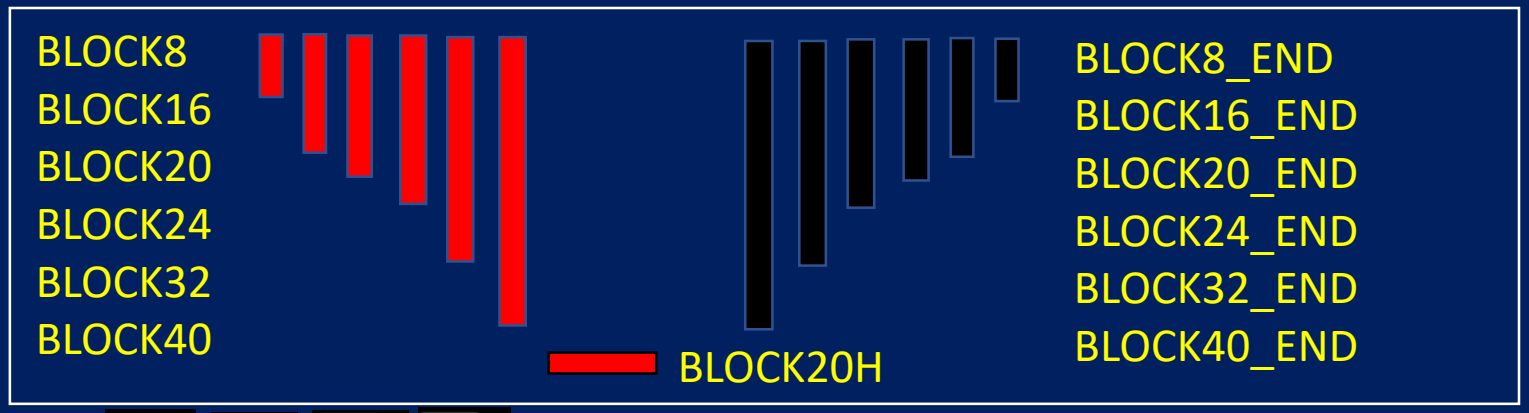
Images_mygame.asm

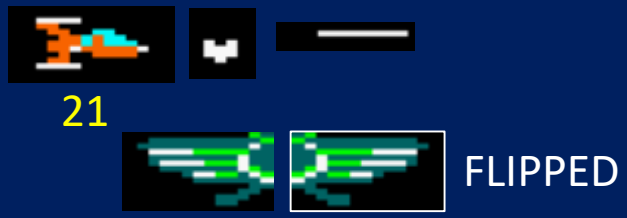
```
;================================================
_BEGIN_FLIP_IMAGES
;================================================
; aqui pon las imagenes que se definen como otras
; existentes pero flipeadas horizontalmente.
ALA_R    dw ALA_L
;================================================
_END_FLIP_IMAGES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

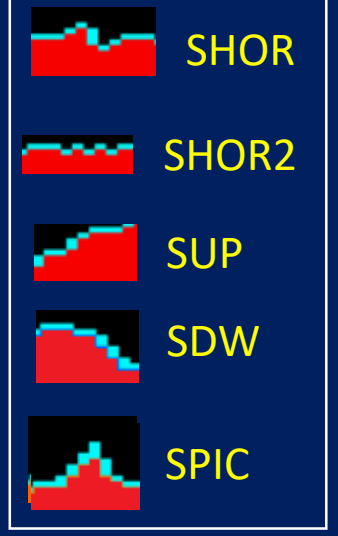
ALA_R

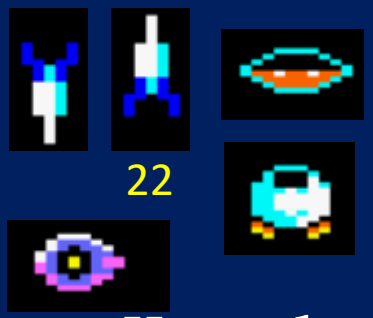
Se obtiene flipeando:

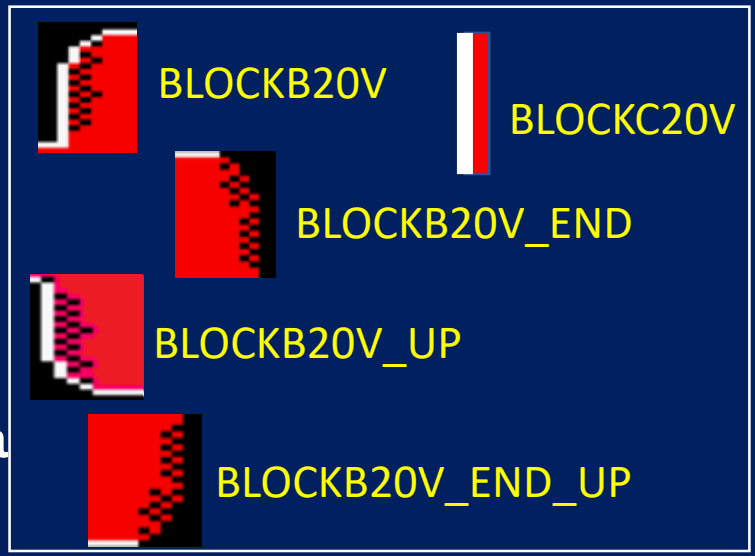
ALA_L











No todas necesitan identificador numérico, basta con un nombre para usarlas en mapas y rutas

Construcción de un mapa

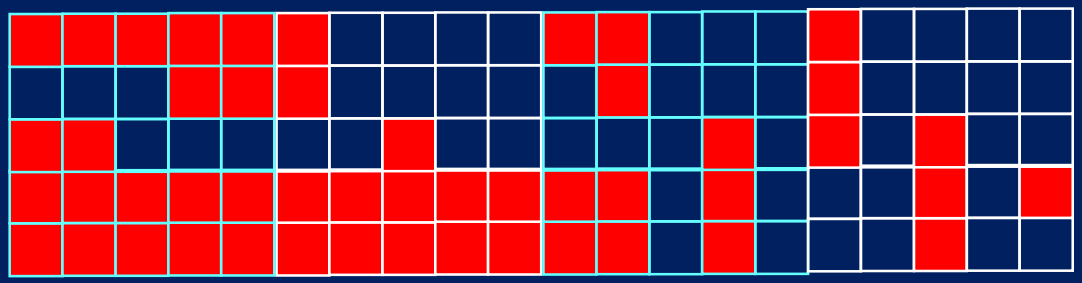
Y X

```
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
dw  9,132,SHOR2
```

Mapa de la fase 3



BLOCK40

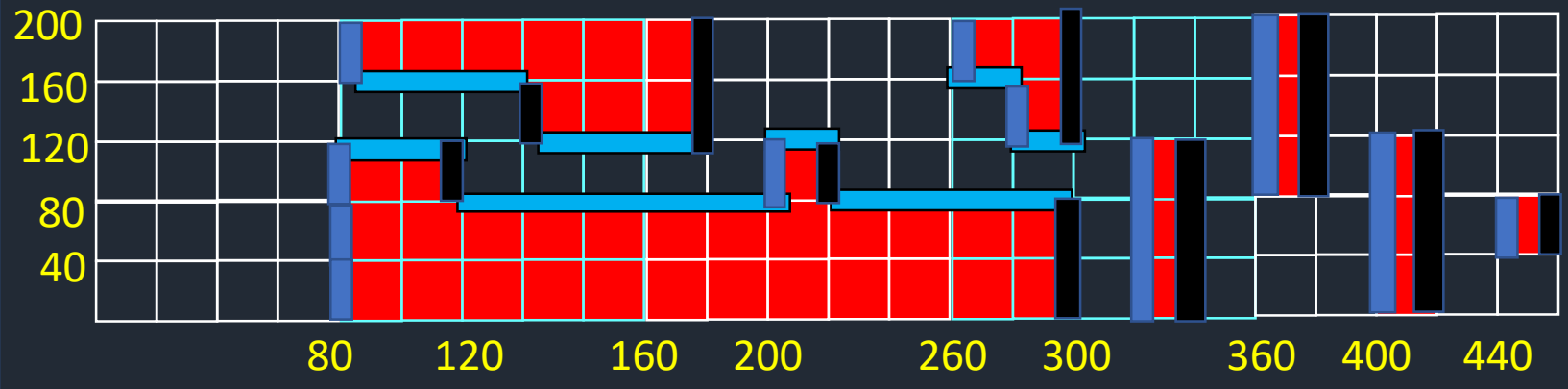
BLOCK20H

BLOCK40_END

Map_fase3.asm

```
ORG 22600

DW 40,80,BLOCK40
DW 80,80,BLOCK40
DW 120,80,BLOCK40
DW 200,80,BLOCK40
DW 120,80,BLOCK20H
DW 161,80,BLOCK20H
...
```



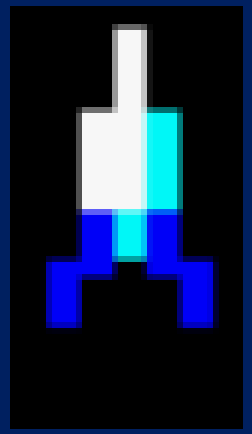
ERIDU ROUTES: fichero routes_mygame.asm

```
; LISTA DE RUTAS
;================
;pon aqui los nombres de todas las rutas que hagas
ROUTE_LIST
        dw ROUTE0;disparo
        dw ROUTE1;fuel
        dw ROUTE2;gema
        dw ROUTE3;cohete
        dw ROUTE4;disparo delete
        dw ROUTE5;ovni
        dw ROUTE6;explosion transparente
        dw ROUTE7;parte comun de ambas explosiones
        dw ROUTE8;explosion normal
        dw ROUTE9;bomba
        dw ROUTE10;bomba delete
        dw ROUTE11;cohete down
        dw ROUTE12;eye
        dw ROUTE13;alienball
```

```
; DEFINICION DE CADA RUTA
;========================
ROUTE0; disparo
        ;db 255,128+64+32+8+1,0
        db 253
        dw DISPARO
        db 20,0,5
        db 255,0,0
        db 1,0,0
        db 0

…

ROUTE3; cohete
        db 253
        dw COHETE
        db 100,0,-1;pausa
        db 128,-3,-1
        db 255,0,0
        db 1,0,0
        db 0
```




ERIDU: ficheros de enemigos





- **No es algo 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 

ERIDU fichero de enemigos fase 3

```
org 23100;&5cf8
; cada elemento es un enemigo que nacera en la posicion X=LINEA*8+Xini
; cada elemento se define con 4 parámetros (ruta, y, x, pasos)
; si ruta=0 entonces no hay enemigo en esa posicion
; he puesto 50 elementos de modo que como mucho map puede medir 50x8 = 400 de ancho
; formula para ubicar mas o menos un enemigo (calcular la x aqui) es
;  linea=(Xmap-80)/8 , tras probar corrijo un poco la Xini, o la línea o la Y
; lo hago asi (con coordenadas x divididas entre 8) por dos motivos
; 1) porque cada enemigo nuevo aparecera cada varios bytes, no cada byte.
; 2) porque usando bytes (en lugar de words) para las coordenadas ahorro memoria

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; esto es una gema
db 0,0,0,0
db 0,0,0,0
db 0,0,0,0
...
```

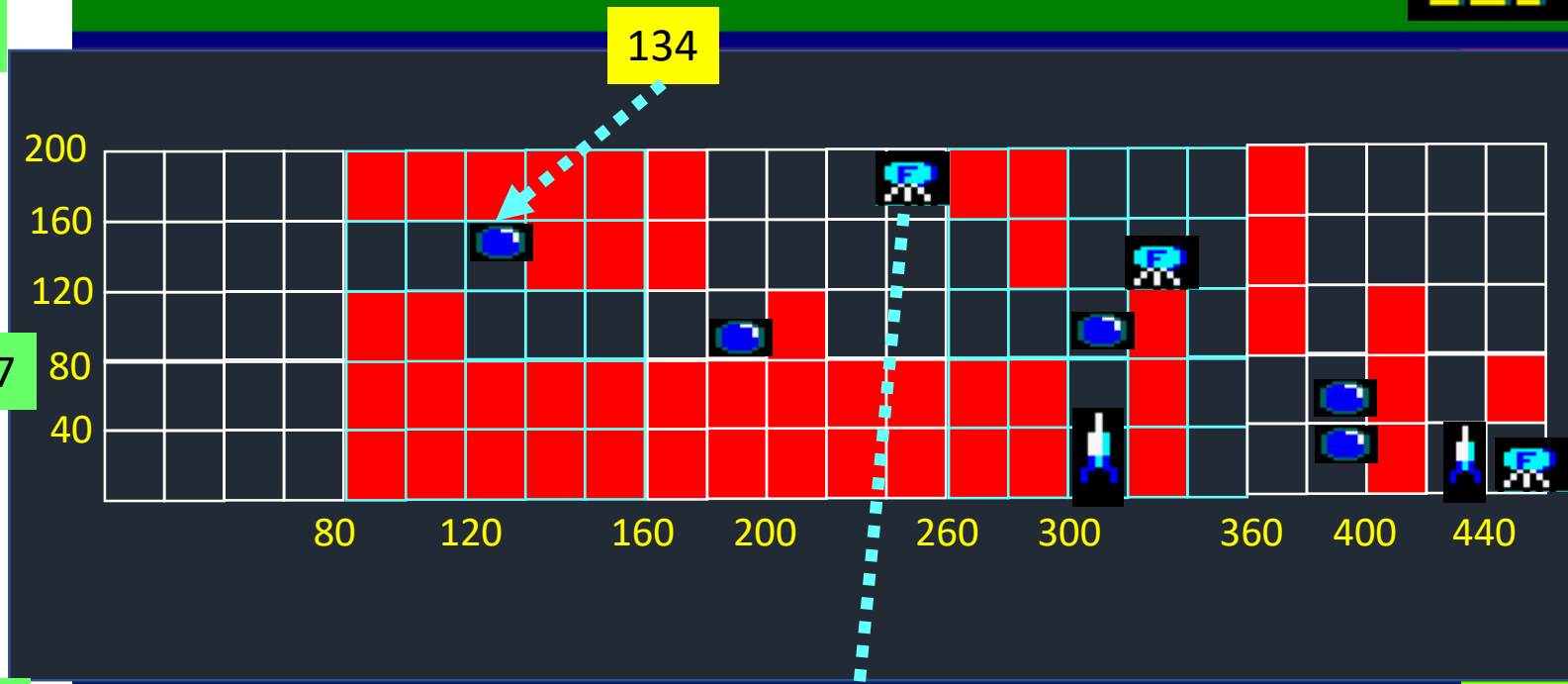
Cada entrada es un enemigo y se van a crear cada 8 fotogramas, de modo que la entrada 7 se corresponde con la coordenada de mapa:

7*8 + 78= 134

```
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
```

1

7




```
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
```

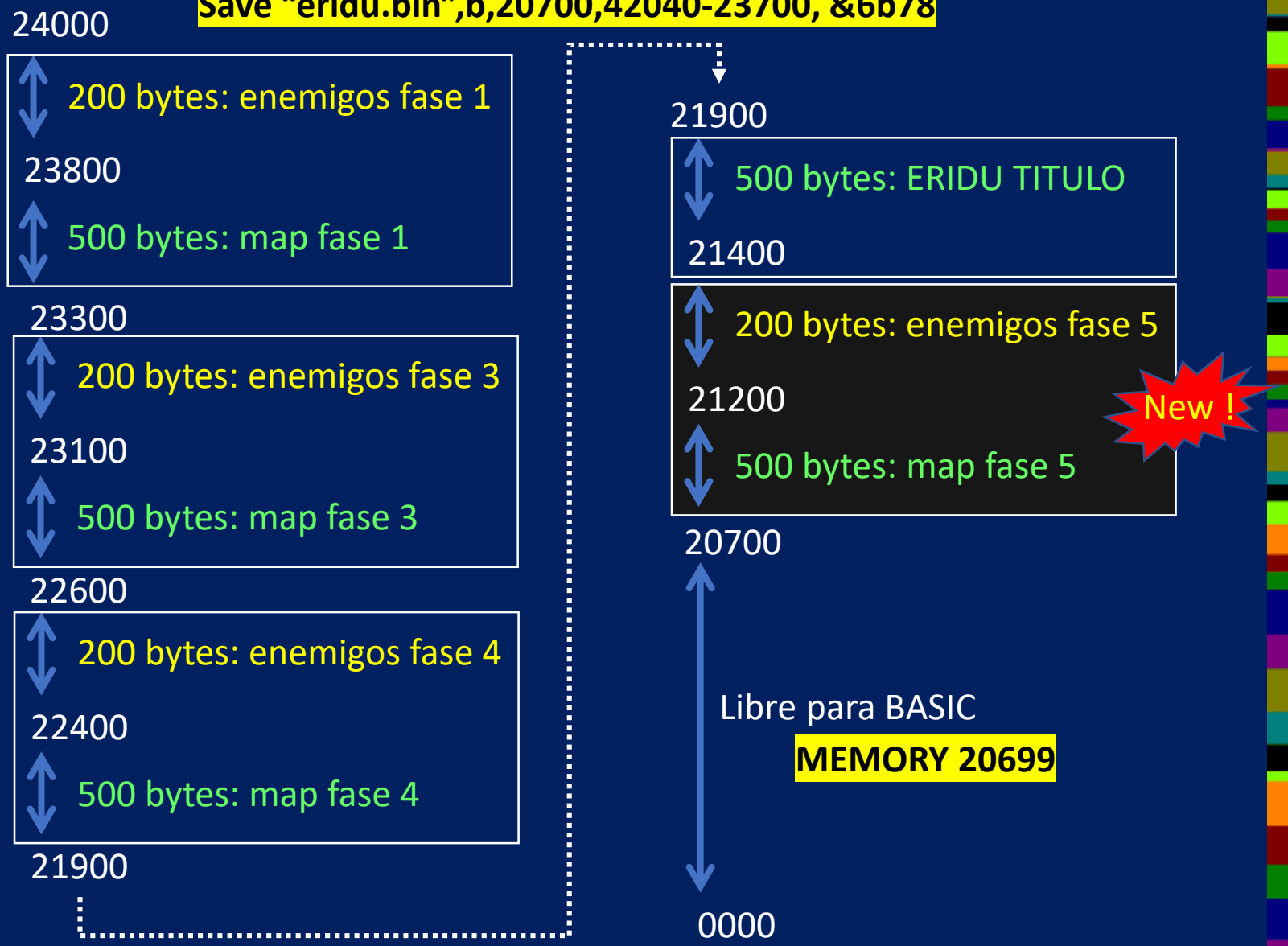
10

19

Xmap=(240-80)/8=20 → lo he ajustado en poscion 19 y coordenada 83
Es decir , 19 * 8 +83 = 235
(el objeto se sitúa en la coordenada de mapa 235)

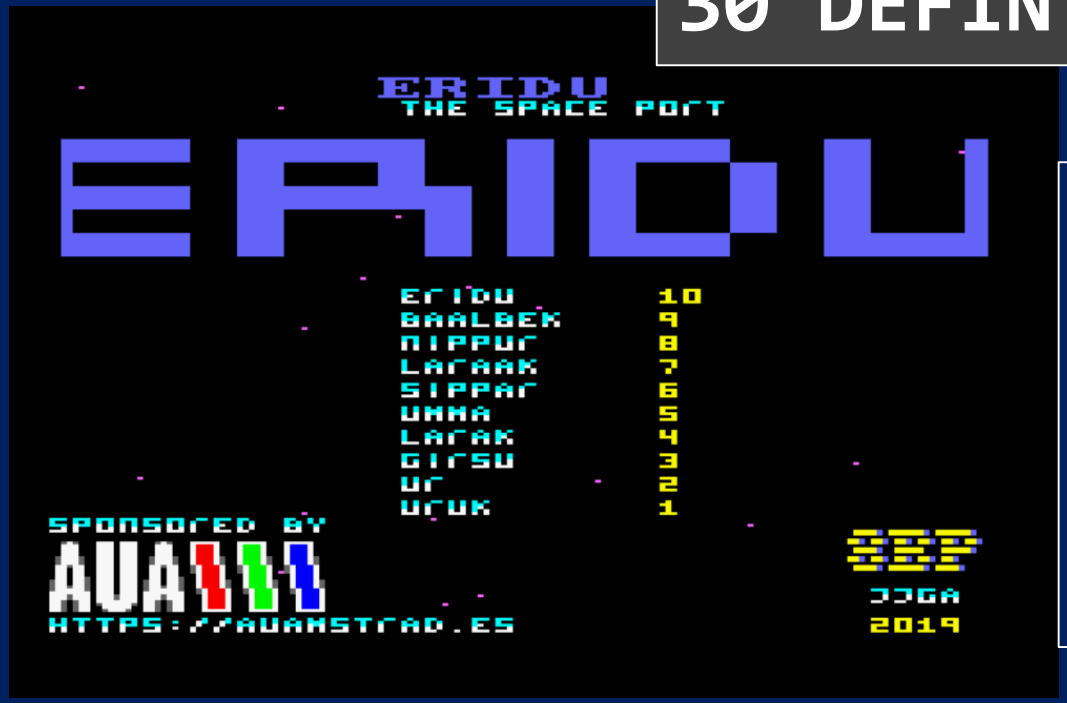
ERIDU MEMORY MAP

Save “eridu.bin”,b,20700,42040-23700, &6b78



Inicio y Presentación

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

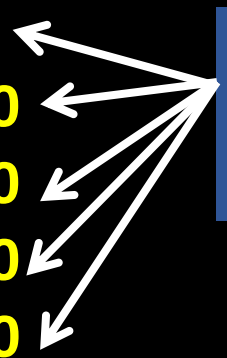


En eridu he puesto 20.699 para dar espacio a varios mapas de fases

Lógica del programa principal

```
180 cor=32:cod=32:|COLSPALL,@cor,@cod:|PRINTSPALL,0,0,0,0:|COLSP,32,0,26:|COLSP,34,0,1
```

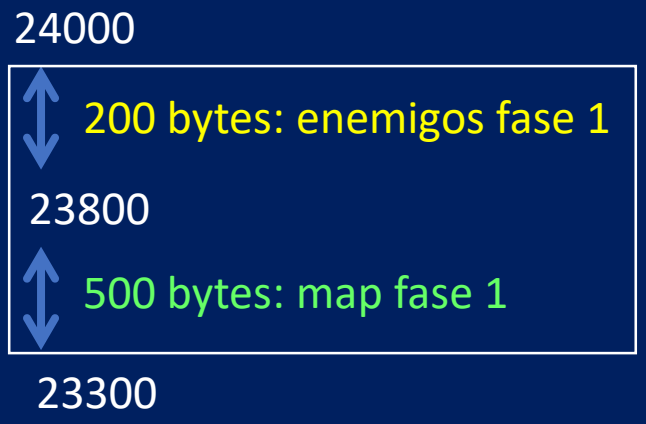
```
370 '---LOGICA FASES
380 vidas=3:fuel=12:score=0:level=1
400 WHILE (vidas>0)
410 IF (level=1) THEN GOSUB 530
420 IF (level=2) THEN GOSUB 1770
430 IF (level=3) THEN GOSUB 1980
440 IF (level=4) THEN GOSUB 2150
450 IF (level=5) THEN GOSUB 2290
460 IF (level=6) THEN 480:'fin
470 WEND
480 IF vidas=0 THEN 1280
490 IF vidas>0 THEN GOSUB 2630:'fin
500 GOTO 1280:'control score y run
```



De aquí retornas al perder una vida o al pasar de nivel

Inicializacion de fase

```
520 '--- FASE 1 ---
530 MODE 0:BORDER 0:INK 0,0:INK 7,15:INK 1,6: INK 15,24: INK 5,0
540 GOSUB 1590:'paint marcadores
550 maxm=390:dis=0
570 ciclomax=1250:eb=0:'enable bomba 0=enabled
580 fase=23800:GOSUB 1240:'carga enemigos de fase1
590 |UMAP,23300,23792,0,200,0,500:'carga map fase1
600 my=180:mx=64:ox=16:GOSUB 2140:' stars
610 |MUSIC,1,6
620 GOTO 640:' vamos al ciclo de juego
630 '--- END FASE1 ---
```



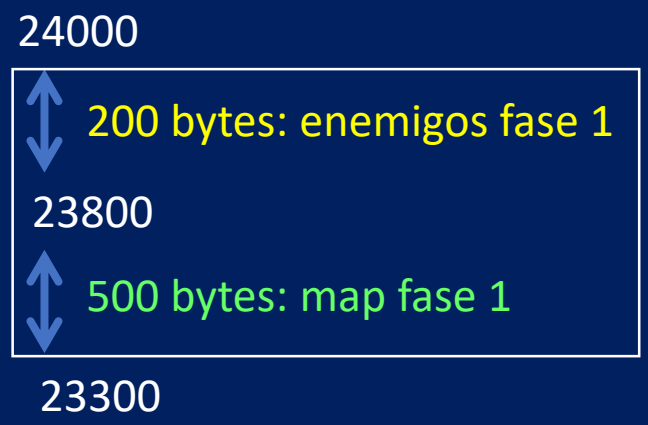
Carga del mapa

Rutina pintado marcadores


```
1590 '-- PAINT MARCADORES
1600 |MUSICOFF
1610 MODE
0:|SETLIMITS,0,80,0,200:|STARS,0,20,14,1,1:|STARS,20,40,11,1,1
1620 c$="READY LEVEL"+STR$(level)+" ":|PRINTAT,100,30,@c$
1630 FOR i=0 TO 40:|STARS,0,20,14,0,1:|STARS,20,40,11,0,2:NEXT
1640 MODE 0:FOR i=0 TO 31:|SETUPSP,i,0,0:NEXT:'reset sprites
1650 |SETUPSP,31,0,1+32:|SETUPSP,31,9,21
1660 |SETUPSP,30,0,1: |SETUPSP,30,9,23: |PRINTSP,30,2,2:
|SETUPSP,30,9,24: |PRINTSP,30,2,8
1670 PLOT 0,0,8:DRAW 0,398:DRAW 16*8-2,398: DRAW 16*8-2,0,4:DRAW
16*8-2,0:DRAW 0,0
1680 c$="SCORE":|PRINTAT,32,2,@c$:c$=STR$(score):|PRINTAT,40,2,@c$
1690 c$="FUEL":|PRINTAT,64,2,@c$
1700 c$=">>>>>>":|PRINTAT,72,2,@c$:fuel=12:'fuel max
1710 c$="LEVEL":|PRINTAT,96,2,@c$
1720 c$="LIVES":|PRINTAT,128,2,@c$
1730 |SETLIMITS,0,80,0,200:FOR i=1 TO vidas:
|PRINTSP,31,128+16*i,2:NEXT: |SETLIMITS,16,80,0,200
1740 c$=STR$(level):|PRINTAT,112,4,@c$
1750 y=100:x=30:|LOCATESP,31,y,x
1760 RETURN
```

Carga de enemigos

```
580 fase=23800:GOSUB 1240:'carga enemigos de fase1
```




```
db 3,170,80,100;cohete
db 3,170,80,80
db 3,170,80,70
db 3,170,80,100
db 3,170,80,100
db 3,170,80,80
db 1,178,80,0;fuel
db 0,0,0,0
db 0,0,0,0
db 2,136,80,0;gema
```

```
1240 '--- LOAD ENEMIGOS DE FASE
1250 FOR i=0 TO 49:FOR j=0 TO 3:e(i*4+j)=PEEK(fase+i*4+j):NEXT:NEXT:'enemigos
1260 FOR i=200 TO 240 STEP 4:e(i)=0:NEXT:'limpia (fase 2 llena mas)
```

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

Un array de una sola dimensión e(i) donde metemos los datos de enemigos que leemos con PEEK

Ciclo de juego


```
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

Teclado: método eficiente

```
850 '---RUTINA LECTURA CONTROLES
860 IF INKEY(69) THEN 880
870 IF y<180 THEN y=y+4:POKE 27497,y:GOTO 900
880 IF INKEY(67) THEN 900
890 IF y THEN y=y-4:POKE 27497,y
900 IF INKEY(34) THEN 920
910 IF x>16 THEN x=x-2:POKE 27499,x:GOTO 940
920 IF INKEY(27) THEN 940
930 IF x<70 THEN x=x+2:POKE 27499,x
940 IF demora THEN demora=demora-1:RETURN ELSE IF INKEY(47) THEN RETURN
950 disp=(disp AND 1)+29
960 |SETUPSP,disp,0,233: |SETUPSP,disp,15,0: |LOCATESP,disp,y+6,x-8:
demora=7
980 IF PEEK(27448)=0 THEN 990 ELSE IF PEEK(27432)=0 THEN 1000 ELSE
RETURN
990 |SETUPSP,28,0,233:|SETUPSP,28,15,9:|LOCATESP,28,y+6,x:RETURN
1000 |SETUPSP,27,0,233:|SETUPSP,15,9:|LOCATESP,27,y+6,x:RETURN
```

Muerte

```
1140 '--- MUERTE NAVE
1150 cicloaux=ciclo
1160 |MUSICOFF:SOUND 3,145,200,12,1,2,12
1170 |SETUPSP,31,4+1:|SETUPSP,31,7,1
1180 FOR i=0 TO 100
1190 BORDER RND*15:CALL &BD19
1200 |ANIMA,31:|PRINTSP,31
1210 NEXT
1220 vidas=vidas-1
1230 reiniciar=1:RETURN
```

fichero secuences_mygame.asm

```
;--------------secuencias de animacion  --------
_SEQUENCES_LIST
dw EXPLO1,EXPLO2,EXPLO3,EXPLO4,EXPLO5,0,0,0;1
```

Rutina de colisión

```
IF cor<32 THEN GOSUB 1020
```

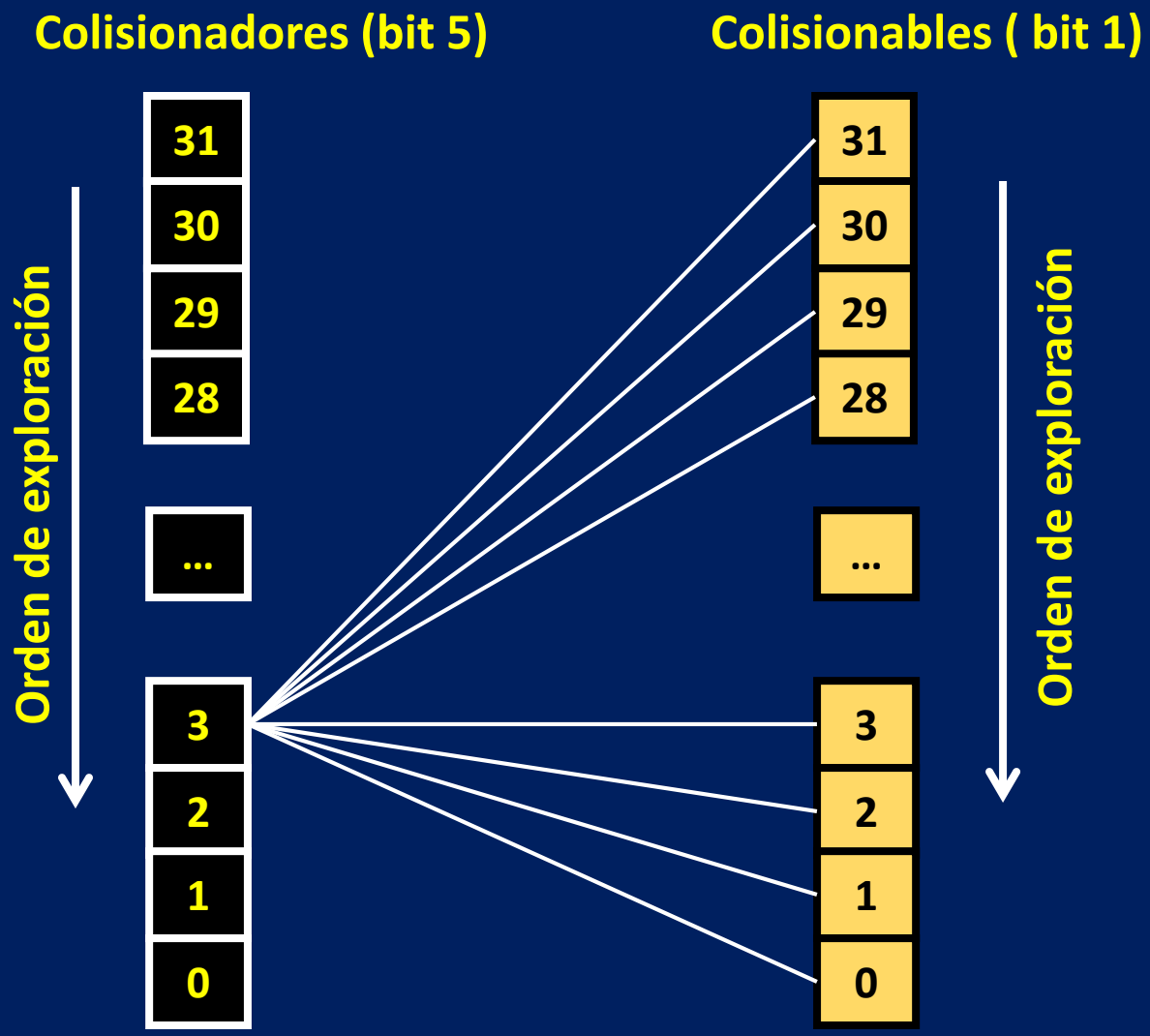
La rutina de colisión debe tener en cuenta todos los casos:

- Colisión de nave (cor=31): debe morir
- Colisión de disparo o bomba (cor>27):
  - Si cor>28: borrar disparo, else borrar bomba
  - Si cod<20 no debe pasar nada
  - Si cod>20
    - el enemigo debe morir
    - Score=score+10, imprimir nuevo score
    - Si score mod 500 =0 : nueva vida
    - Si cod=1: aumentar fuel

Rutina de colisión

```
1010 '--- RUTINA COLISION
1020 IF cor=31 THEN 1140
1030 IF cor>28 THEN |SETUPSP,cor,15,4 ELSE |SETUPSP,cor,15,10
1040 IF cod<20 THEN RETURN
1050 dir=27015+cod*16
1060 IF PEEK(dir)=1 THEN IF fuel<10 THEN
c$=">":|PRINTAT,72,1+fuel,@c$:fuel=fuel+3
1070 IF PEEK(dir)<3 THEN |SETUPSP,cod,15,6 ELSE |SETUPSP,cod,15,8
1080 score=score+10:c$=STR$(score):|PRINTAT,40,2,@c$
1090 IF score MOD 500 THEN RETURN
1100 IF vidas=3 THEN RETURN ELSE vidas=vidas+1:
|SETLIMITS,0,80,0,200: FOR i=1 TO
vidas:|PRINTSP,31,128+16*i,2:NEXT:|SETLIMITS,16,80,0,200
1110 |LOCATESP,31,y,x
1120 FOR i=0 TO 100 STEP 10:BORDER i MOD 16:SOUND 6,200-i,5,15:
NEXT: BORDER 0
1130 RETURN
```

Funcionamiento de |COLSPALL



Empieza
la aventura de programar!

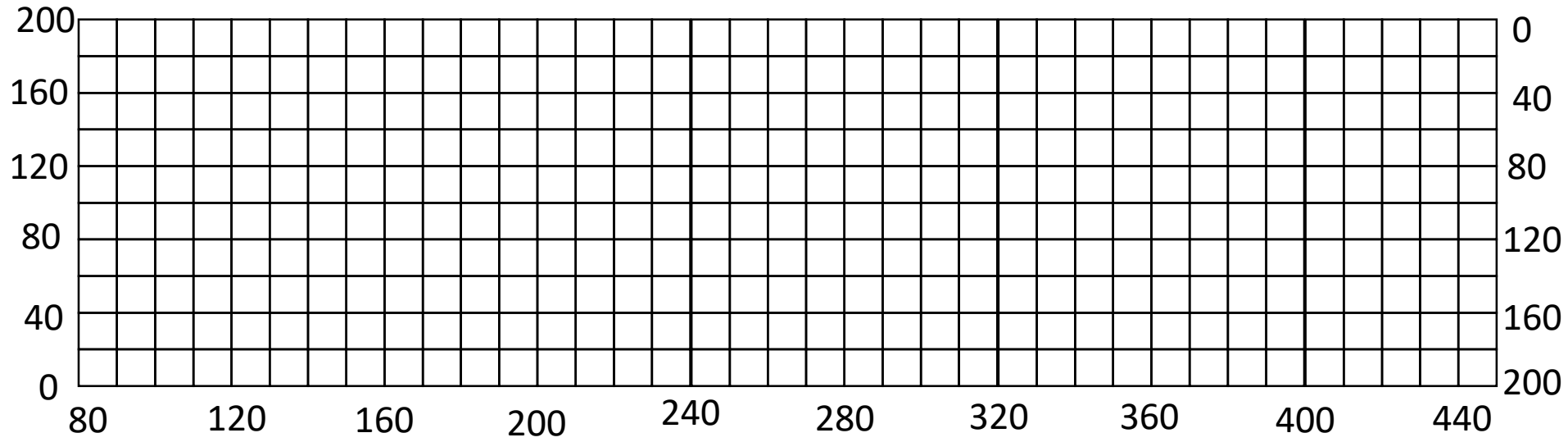
consejos:

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

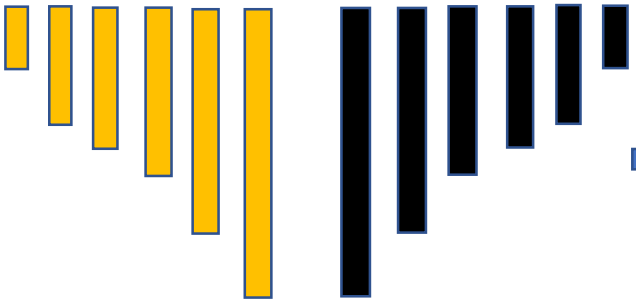
Tu propia fase !

- 401 level=5
- 2300 level=level+1:RETURN:'asi no hace nada → 2300 rem
- Crear el fichero de mapa de fase 5 map_fase5.asm similar a map_fase3.asm pero con org 20700
  - Para escribir el fichero de mapa, dibujar tu mapa en una cuadricula e ir traduciéndolo a imágenes y coordenadas. Puedes hacer algo sencillo usando los mismos tipos de bloques de la fase 3
  - Toma nota de donde termina tu mapa cuando lo termines, usando una etiqueta _END_MAP_FASE5 y consultando su dirección con la opción "symbols" de winape.
  - Usa esa dirección en el comando |UMAP, 20700,XXXX,0,201,0,500

- Crear el fichero de enemigos de fase 5 enemigos_fase5.asm similar a enemigos_fase3.asm pero con org 21200
  - Ubicalos en coordenadas de mapa que esten libres de bloques
  - Ten en cuenta que el sistema de coordenadas de mapa y el de sprites funcionan con la Y al reves
  - Usa los enemigos existentes o incluso dibuja un nuevo enemigo y/o diseña una nueva ruta



BLOCK8
BLOCK16
BLOCK20
BLOCK24
BLOCK32
BLOCK40

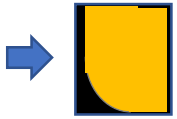


BLOCK20H

BLOCK8_END
BLOCK16_END
BLOCK20_END
BLOCK24_END
BLOCK32_END
BLOCK40_END

dibuja tu mapa

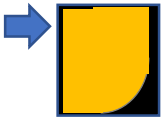
BLOCK20V



BLOCK20V_UP

BLOCKC20V

BLOCK20V_END



BLOCK20V_END_UP

Ensambla con make_graficos.asm cada vez que cambies algo
Toma nota de la etiqueta "_end_map_fase5" para el comando UMAP


https://8bitsdepoder.blogspot.com
https://github.com/jjaranda13/8BP
FRUTOMETER
LIVES
3D RACING ONE
Mutante Montoya
Polvoranca Forrest