8 BITS DE PODER

Como programar

“PACO, el hombre”

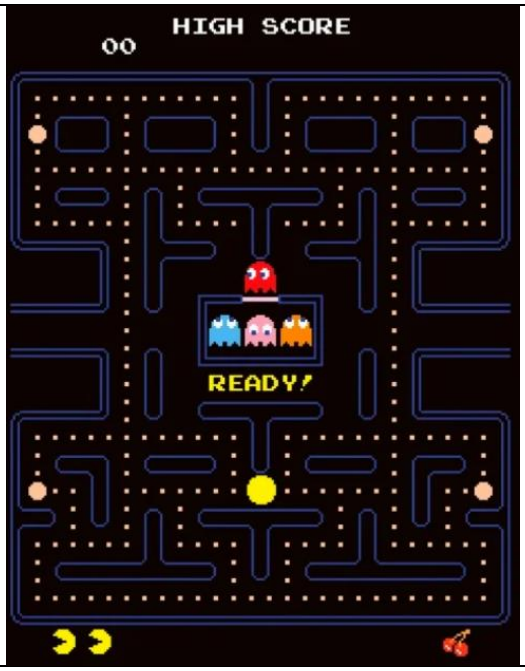
V41

Jose Javier García Aranda

Contenido

1 Historia de comecocos (“PACMAN”)

Juego creado en 1980 creado por por el diseñador de videojuegos Toru Iwatani de la empresa **Namco**, y distribuido por Midway Games.

PACMAN fue un éxito desde el principio y se ha convertido en un gran clásico, un título de culto. Simboliza gráficamente el concepto de videojuegos. Ver el comecocos y los fantasmas es como leer la palabra “videojuegos”. PACMAN fue el juego que más monedas recaudó en los 80 (14 mil millones de dólares). Entre 1981 y 1987 se vendieron 293.822 máquinas arcade con todas las variantes de este videojuego	

	PACMAN original mente se llamaba “puckman” y su nombre viene de “paku”, onomatopeya japonesa para el sonido de abrir y cerrar la boca. Los distribuidores americanos (la empresa MIDWAY) decidieron cambiarle el nombre a PACMAN por las bromas que podría generar una palabra que en ingles se parece a fuckman (que más o menos significa hijo puta en inglés)

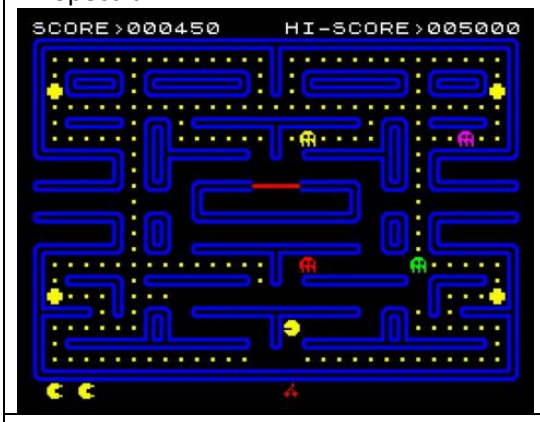
En el juego original cada fantasma tiene una personalidad según su color:

- El rojo es el mas peligroso y persigue a pacman
- El rosa a veces persigue, a veces no. Sus decisiones son aleatoriamente inteligentes
- El naranja tiene un movimiento aleatorio, no persigue a pacman
- El azul es “torpe” y a veces huye de pacman, es decir aleatoriamente a veces toma decisiones de huida

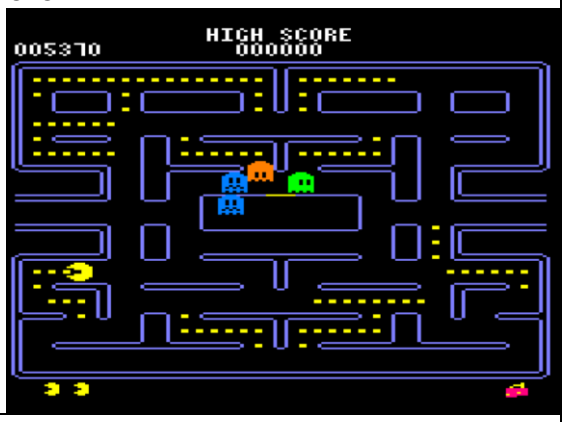
Los fantasmas deciden en cada bifurcación del laberinto que nuevo camino seguir, y siempre tienen “prohibido” retroceder. Cuando pacman entra por el pasadizo, aparece al otro lado y si los fantasmas le siguen, reducen su velocidad al pasar por dicho pasadizo.

Hay muchas versiones del juego para todas las plataformas, he aquí algunas:

ZX Spectrum



CPC

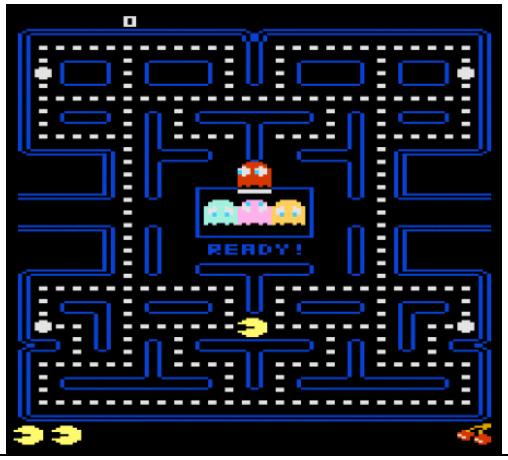


C64

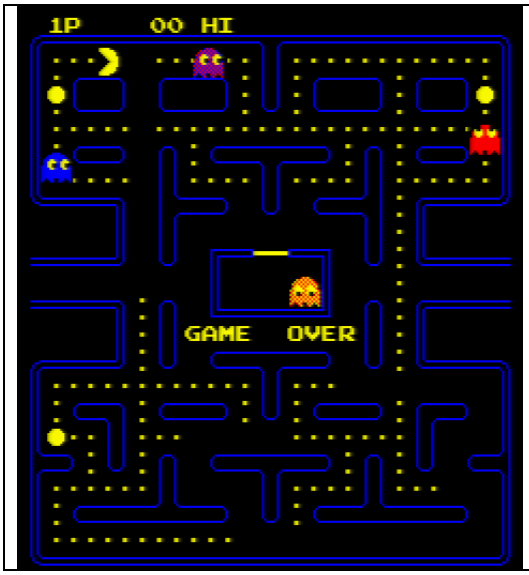
MSX

Bbc micro

Atari 7800



Recientemente (año 2020) fue publicada la que probablemente sea la mejor versión de pacman para un ordenador de 8 Bit, se trata del “PACMAN emulator” que corre en Amstrad de 128 KB (no funciona en CPC464). Su similitud con la maquina arcade es magnífica



Esta versión para CPC6128 usa los ROMS originales de la maquina arcade, de modo que se trata de un auténtico emulador de la maquina original. De hecho, es necesario incluir los ficheros originales del arcade en un disco de Amstrad, además del emulador capaz de interpretarlos.

El juego consume mucha memoria y usa mezclas de color en mode 1 para lograr un aparente numero de colores mayor de 4.

También usa un modo de video especial, sin borde, aprovechando toda la longitud vertical de la pantalla

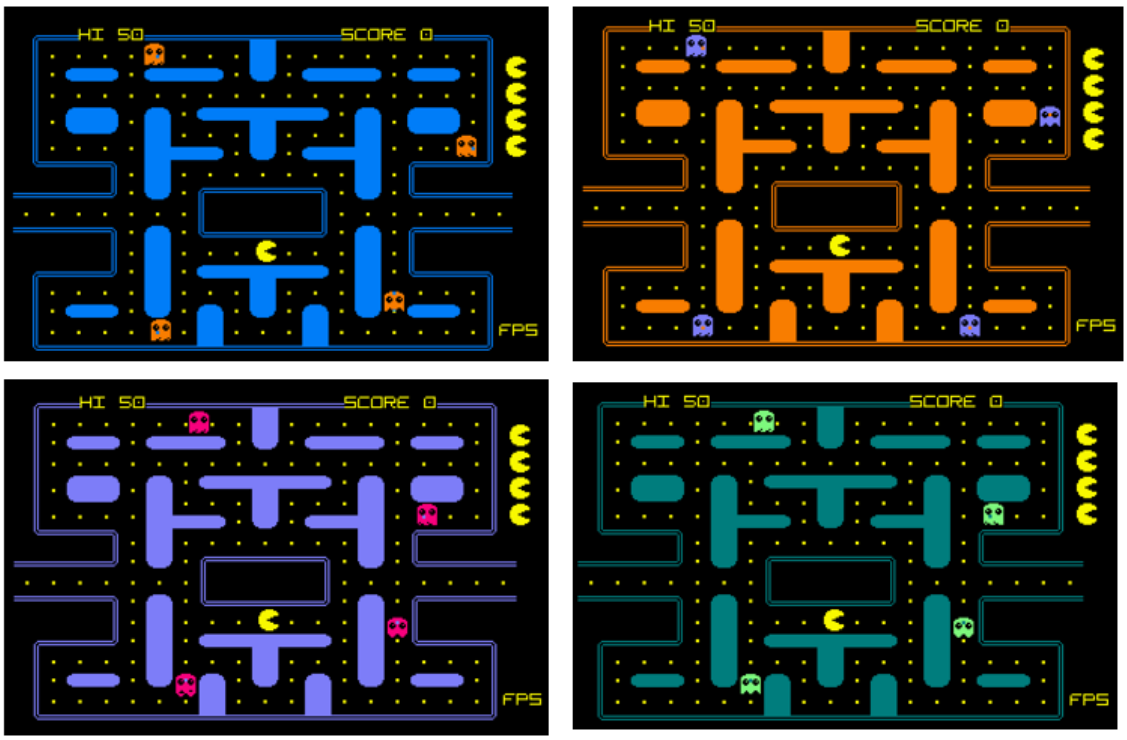
2 Versión programada con 8BP

La versión programada con 8BP se titula casi igual que el original, “Paco, el hombre”, que viene a ser una traducción “simpática” de PACMAN. Es un juego con la esencia del mítico comecocos pero simplificado. No se han incluido frutas ni te puedes comer a los fantasmas.

Consta de 4 niveles y en cada uno la velocidad es mayor. En cada nivel el numero de fotogramas por segundo aumenta y la velocidad llega a complicar mucho el juego

- Nivel 1: 15 FPS
- Nivel 2: 19 FPS
- Nivel 3: 30 FPS
- Nivel 4: 33 FPS

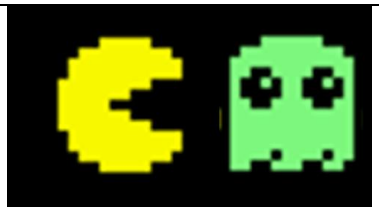
Cada nivel tiene un diseño de colores diferentes al resto, pero el laberinto es el mismo.



Para lograr esas velocidades se ha hecho un uso muy completo de la técnica de **lógicas masivas**. Los dos primeros niveles se ejecutan en BASIC y los dos segundos se ejecutan con ciclo compilado.

Un juego en BASIC con 4 enemigos capaces de moverse por un laberinto, sprites con sobreescritura (los fantasmas no destruyen los cocos al pasar por encima), detección de colisiones con los muros y los fantasmas, etc **funcionando a casi 20 FPS en BASIC** es algo imposible sin la técnica de lógicas masivas.

Otra novedad técnica de este juego es el movimiento de dos en dos pixeles de mode 1 (es decir, **medio byte**) logrando un desplazamiento bastante suave de todos los personajes. Normalmente en 8BP los juegos se mueven byte a byte (2 pixeles de mode 0 , o lo que es lo mismo 4 de mode 1) sin embargo en este juego se ha empleado una técnica que describiré en este documento para lograr un movimiento más suave

Los sprites han sido diseñados con el spedit15 (herramienta que viene con 8BP) . En este caso al tratarse de la versión 15, te permite desplazar una imagen medio byte para obtener una versión de la imagen ligeramente desplazada	

La música se ha creado con el WYZtracker, creado por augusto ruiz, que también viene con 8BP (y el player de música está integrado en la librería a través del comando MUSIC)

Controles del juego:

Para comenzar a jugar, pulsa espacio (o el disparo del joystick)

Q (arriba), A (abajo), O (izquierda), P (derecha)

M: activar/desactivar la musica

3 Diseño general

En este apartado vamos a ver los sprites que usamos para cada elemento del juego.

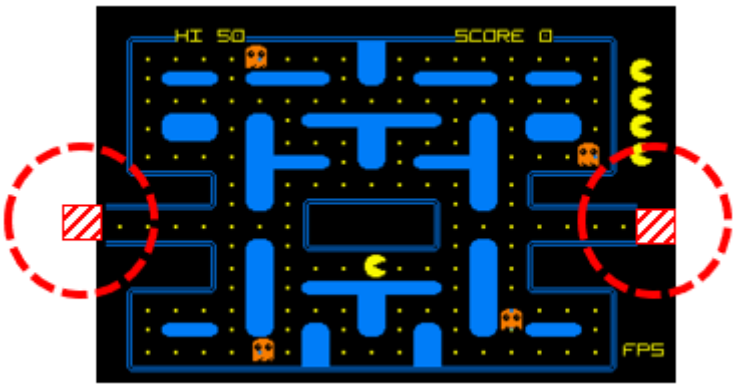
Distribución de Sprites:

La distribución de sprites es importante para identificar en la rutina de colisión si te has chocado contra un muro del decorado, o un disparo enemigo. También es necesario para elegir un Sprite ID cuando se va a crear un disparo nuevo, por ejemplo. Definir este reparto es fundamental. Una cosa que se deduce de esta tabla es que Joe puede tener hasta 6 disparos "vivos" simultáneos en pantalla.

Sprite iD	Uso
31	Paco
27,30	Fantasmas (4)
25,26	Sprites invisibles para forzar el cambio de posición al atravesar el túnel

Los sprites "invisibles" (sin flag de impresión en su estado) se colocan en los extremos del túnel. Cuando paco colisiona con ellos, se mueve de posición, 80 bytes a la derecha o a la izquierda (según choque con el situado a la izquierda o derecha). Los fantasmas no usan este mecanismo, simplemente cuando su coordenada excede de un umbral, se mueven al otro lado. El motivo de la elección de este mecanismo para paco es el de evitar sentencias "IF" que penalicen el rendimiento, y aprovechando que hay una llamada a la detección de colisiones de 8BP, se puede aplicar este método. En realidad, es más complejo, pero la detección de colisión se hace en código maquina y es muy rápida, mucho mas que una sentencia IF. En cuanto a los fantasmas, estos no tienen detección de colisión en su lógica y por lo tanto es necesario comprobar sus coordenadas.

Como ves, el Sprite invisible de la izquierda está fuera de la pantalla, en una coordenada negativa, lo cual no supone ningún problema.



Distribución memoria:

El programa BASIC empieza con una instrucción MEMORY 17499 para que las variables no excedan de esa dirección. EL listado basic ocupa 13 KB. Contiene el ciclo de juego porque aunque los niveles 3 y 4 usan ciclo compilado, los niveles 1 y 2 funcionan en BASIC.

Rango memoria	Uso	comentarios
0-17499	BASIC y variables BASIC	El listado basic ocupa 13 KB
17500-23663	Ciclo compilado	Usado en niveles 3 y 4
24000-32200	Rutinas 8BP	8,2KB
32200-33600	Canciones	1 cancion (solo se usan 0.9 KB)
33600-42040	Rutas y gráficos	8.4 KB (solo se usan 3.1 KB)
40000-42000	Decisiones precalculadas	Se usa en el nivel 2 y en el 4
42040-42540	layout	

Todos los sprites han sido creados con speditV15, el cual permite desplazar medio byte cualquier imagen. Esto se ha usado para lograr el movimiento suave de todos los personajes (paco y los fantasmas)

Ficheros:

Los ficheros de que consta el juego tienen una correspondencia directa con las tablas de rangos de memoria que acabas de ver.

Rango memoria	**ficheros**	**Uso**	**aclaraciones**
no consume memoria durante el juego	"LOADER.BAS"	Carga todos los ficheros del juego. El ultimo que carga es "PACO.BAS", el cual entra en ejecución	Tiene pocas líneas, incluye el cambio de tintas para que el grafico inicial cargue con los colores adecuados
0-13KB	"PACO.BAS"	Listado BASIC y variables BASIC	Es el juego
17500-42620	8BP.BIN	Librería 8BP, ciclo compilado, música y gráficos y decisiones precalculadas	Un solo fichero binario que contiene todo.

Para generar 8BP.bin se han seguido los siguientes pasos

Ensamblado de librería gráficos y música (desde winape ensamblamos make_all.asm)

Seleccionamos como drive A el disco donde tenemos el ciclo de juego compilado (ciclo.dsk) y lo cargamos. El fichero disco.dsk lo genera el script "compila.bat" una vez que hemos escrito el programa C.

LOAD "ciclo.bin"

En este momento ya tenemos casi todo lo "binario" necesario en memoria RAM. Falta cargar las decisiones precalculadas para los niveles 2 y 4. Para ello hice una modificación en el programa BASIC que almacenaba cada decisión de fantasma en cada ciclo de juego. Hay fotogramas en los que la decisión es "seguir" y otros en los que la decisión es "tomar el camino de la derecha", etc. Pues bien, ejecutando esa "variante" del programa BASIC, se van guardando las decisiones desde la dirección de memoria 40.000 hasta la 42.000 (2KB) y con ello se almacenan las decisiones correspondientes a 6712 ciclos (una cantidad muy grande).

A continuación, guardé en disco todo lo binario con un solo comando

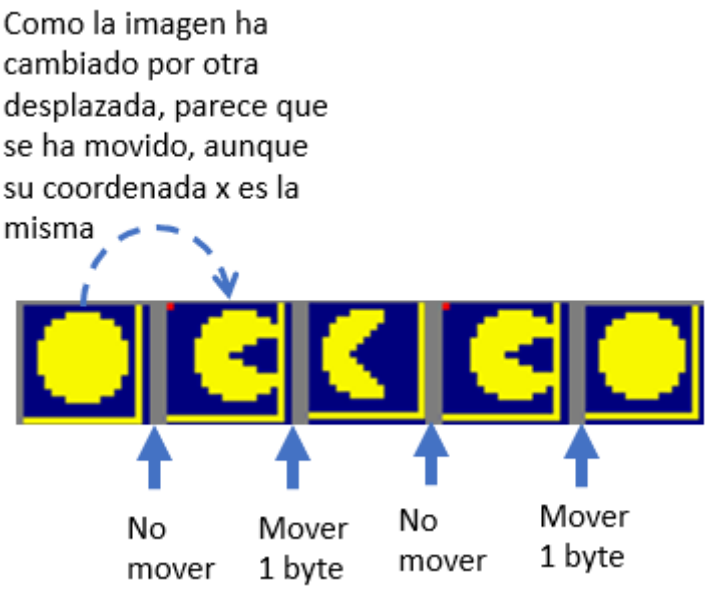
SAVE “8BP.bin”, b, 17500, 25120

Es cierto que este fichero binario contiene muchos espacios no usados, pues sobra mucho espacio de música y gráficos, pero simplifica la carga al tener que cargar únicamente un fichero. Otra desventaja es que esos KB “vacíos” van a hacer que la carga en cinta de cassette tarde un poco más, pero en total hablamos de unas 4KB vacías, que son un par de bloques de carga en cinta, no es demasiado.

4 Movimiento suave de medio byte

8BP mueve los sprites byte a byte con comandos como MOVER, y su sistema de coordenadas es de bytes, no de pixeles. Por lo tanto hablamos de 80 posiciones en el eje horizontal y de 200 en el vertical.

Un byte contiene 2 pixels en mode 0 , o bien 4 pixeles en mode 1. Puede que queramos un movimiento más suave (pixel a pixel en mode cero o de dos pixels en mode 1). Para ello hay un sencillo truco que podemos emplear. Se trata de tener una imagen del personaje desplazada medio byte y simplemente asignársela al Sprite. Aunque se imprima en la misma coordenada, parecerá que se ha movido.



En esta secuencia de movimiento, hay veces que el comecocos no se mueve, pero como cambiamos la imagen es como si se moviese 2 pixeles (medio byte). En los momentos en los que se mueve 1 byte, la imagen aparece desplazada hacia la izquierda medio byte, de modo que el resultado “neto” también es como si se moviese 2 pixeles (medio byte) hacia la derecha

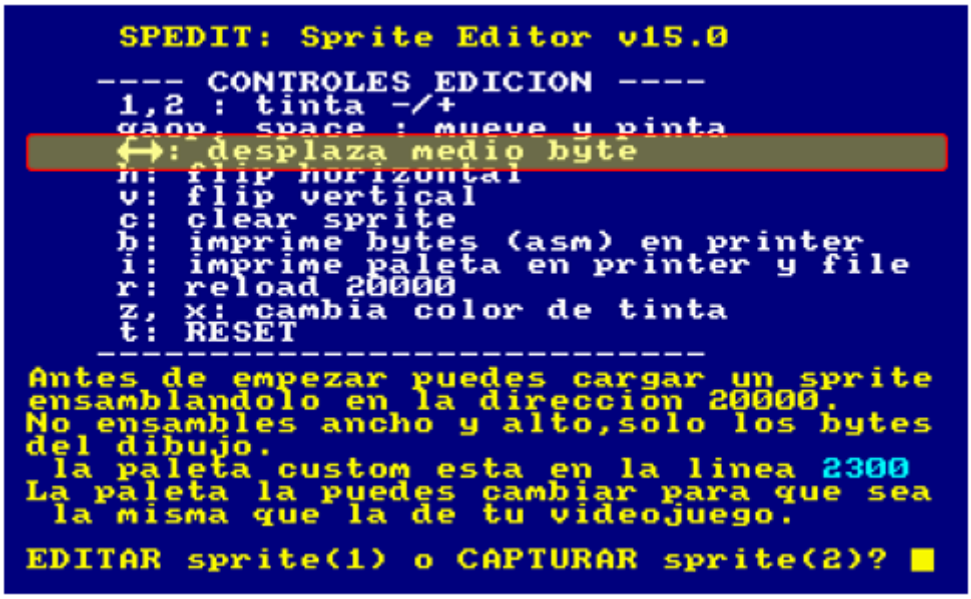
Para definir el movimiento de este comecocos podemos usar el mecanismo de las rutas de 8BP. Este seria el ejemplo de la ruta de movimiento hacia la derecha

```
ROUTE0; derecha
        db 253
        dw COCO_R1
```

```
      db 1,0,0 ; no se mueve
      db 253
      dw COCO_R2
      db 1,0,1  ; se mueve
      db 253
      dw COCO_R1
      db 1,0,0  ; no se mueve
      db 253
      dw COCO_R0
      db 1,0,1  ; se mueve
      db 0
```

Dibujar las imágenes desplazadas puede resultar algo tedioso. Por ello desde la versión V15 de spedit posees un mecanismo de desplazamiento de una imagen que la desplaza medio byte (2 pixels de mode 1 o un pixel de mode 0) a la derecha o a la izquierda

Como ves en el menú de SPEDIT te aparece una nueva opción: con las flechas de cursor puedes producir un desplazamiento de la imagen que has dibujado



Esto nos permite fácilmente desplazar una imagen para aplicar la técnica descrita de movimiento suave

5 Lógica del programa principal

El programa principal es muy sencillo. Además, no hay tabla de puntuaciones, tan solo almacenamos el valor de la mayor puntuación (similar al juego “happy Monty”).

La lógica del juego (**no confundir con el ciclo de juego**) comienza en la línea 280. La lógica del programa principal es simplemente la que construye el layout y lanza el ciclo de juego, del cual retornamos cuando nos matan.

```
280 '--- BUCLE PROGRAMA PRINCIPAL ---
290 '================================
```

```
291 c=0:co=0:lastop=-1:newr=0:op=0: GOSUB 450:'construccion nivel y
recolocacion
292 if level=1 then ink 3,11:ink 2,15
293 if level=2 then ink 3,15:ink 2,14
294 if level=3 then ink 3,14:ink 2,7
295 if level >3 then ink 3,10:ink 2,22
301 if music then |MUSIC,0,0,0,6
302 GOSUB 1070:'reset timer
310 IF level=1 THEN GOSUB 1120: goto 320 'ciclo basic
311 if level=2 THEN GOSUB 1120: goto 320 'ciclo basic precomputed
312 if level=3 THEN GOSUB 3000: goto 320 'ciclo C
313 IF level>3 THEN GOSUB 3000: goto 320 'ciclo C precomputed
320 IF vidas=0 THEN 340:'fin
330 GOTO 291
```

Si quieres “crackear” el programa, en la línea 130 se asignan las vidas iniciales. Puedes ponerte 10 vidas, por ejemplo

En el programa principal hay una rutina importante, la que invoca el GOSUB 450. Es una rutina que crea el layout, coloca a los cuatro fantasmas en posiciones concretas y también a Paco.

El layout se construye usando un conjunto de variables de cadena L$(0) … L(24) donde cada carácter representa un Sprite id (tal y como se explica en la documentación de 8BP). Las imágenes con las piezas que se usan para construir el laberinto se cargan en los sprites 0 a 27 en la lines 450.

A partir de la línea 870 se colocan los fantasmas y paco. A los fantasmas los inicializamos en posiciones y rutas concretas. La posición inicial es muy importante y tiene que ver con las lógicas masivas, como veremos más adelante.

```
440 '---construccion nivel---
450 FOR i=16 TO 43:|SETUPSP,i-16,9,i:NEXT:' imágenes de construccion
460 |SETUPSP,31,9,16
470 |SETLIMITS,0,76,0,200
480 L$(0)= "=AAAAAAAADAAAAAAA> "
490 L$(1)= "F        O        E "
500 L$(2)= "F;;;;;;;;O;;;;;;;;E "
510 L$(3)= "F<RS<ROS<Q<ROS<RS<E "
520 L$(4)= "F                 E "
530 L$(5)= "F;;;;;;;;;;;;;;;;E "
540 L$(6)= "F KL P<ROOOS<P KL E "
550 L$(7)= "F;MN;O   O   O;MN;E "
560 L$(8)= "F    O;;;O;;;O    E "
570 L$(9)= "F;;;;OOS<Q<ROO;;;;E "
580 L$(10)="?BBG O       O HBB@ "
590 L$(11)="   F;O;;;;;;;O;E    "
600 L$(12)="AAAI<Q<HBBBG<Q<JAAAA"
610 L$(13)="       E   F        "
620 L$(14)=";;;;;;;E   F;;;;;;; ":'el ultimo esta fuera de limits
630 L$(15)="BBBG<P<JAAAI<P<HBBBB"
640 L$(16)="   F O       O E    "
650 L$(17)="   F;O;;; ;;;O;E    "
660 L$(18)="=AAI<O<ROOOS<O<JAA> "
```

```
670 L$(19)="F    O   O   O    E "
680 L$(20)="F;;;;O;;;O;;;O;;;;E "
690 L$(21)="F<RS<Q<P<Q<P<Q<RS<E "
700 L$(22)="F      O   O      E "
710 L$(23)="F;;;;;;O;;;O;;;;;;E "
720 L$(24)="?BBBBBBCBBBCBBBBBB@ "
730 IF newlevel THEN CLS:FOR y=0 TO 24: |LAYOUT,y,0,@L$(y):NEXT: '
solo la primera vez
740 newlevel=0
750 |SETLIMITS,0,80,0,200
760 |SETUPSP,31,9,45: FOR i=1 TO 5:|PRINTSP,31,i*16,74:NEXT
770 |SETUPSP,31,9,41: FOR i=1 TO vidas:|PRINTSP,31,i*16,74:NEXT
780 |SETLIMITS,0,76,0,200
790 col=0
800 FOR i=0 TO 31 :|SETUPSP,i,0,0:NEXT:'reset
810 |AUTOALL,1:|PRINTSPALL,0,0,0,0:|COLAY,62, @col,31 ' config
820 |LOCATESP,31,16*8,36:|SETUPSP,31,0,128+32+8+1:
|SETUPSP,31,15,4:|PRINTSP,31:'quieto

830 'setup fantasmas
840 FOR n=0 TO 3: dfx(n)=27003+(27+n)*16:NEXT:'x
850 FOR n=0 TO 3: dfy(n)=27001+(27+n)*16:NEXT:'y
860 GOSUB 880: GOTO 930
870 'colocacion fantasmas y paco
880 |LOCATESP,27,1*8,4:|SETUPSP,27,0,128+64+2+8+1:
|SETUPSP,27,15,5:|SETUPSP,27,9,43:rf(0)=5 'derecha
890 |LOCATESP,28,1*8,4*17:|SETUPSP,28,0,128+64+2+8+1:
|SETUPSP,28,15,6:|SETUPSP,28,9,43:rf(1)=6 'izq

900 |LOCATESP,29,22*8,4*17-1:|SETUPSP,29,0,128+64+2+8+1:
|SETUPSP,29,15,6:|SETUPSP,29,9,43:rf(2)=6 'izq
910 |LOCATESP,30,22*8,4+1:|SETUPSP,30,0,128+64+2+8+1:
|SETUPSP,30,15,5:|SETUPSP,30,9,43: rf(3)=5 'derecha
920 |SETUPSP,31,9,42:|LOCATESP,31,16*8,36:
|SETUPSP,31,0,128+32+8+1:|SETUPSP,31,15,4:dir=5:|PRINTSP,31
921 RETURN
```

El lanzamiento del ciclo de juego depende del nivel. En los dos primeros niveles se lanza con el **gosub 1120** y en los niveles 3 y 4 (ciclo compilado) se lanza con el **gosub 3000**

El ciclo de juego en C es una traducción casi directa del ciclo de juego en BASIC que vamos a ver en el siguiente apartado. La única diferencia es que antes de llamar al ciclo de juego en C

```
3000 '---ciclo en C
3001'=============
3010 'pasamos direc de variables para poder escribirlas desde c
3020 |POKE,17900,@scr
3030 |POKE,17902,@target
3031 |POKE,17904,@music
3032 |POKE,17906,@hi
3033 |POKE,17908,@flaghi
3034 if level>=4 then |POKE,17914,1 else |POKE,17914,0
3040 call &4d0d:'llamada al ciclo compilado
```

```
3050 'recogida resultados
3055 |peek,17912, @c:'ciclos consumidos
3060 result=0:|peek, 17910, @result:'muerte(0) o target logrado(1)
3070 if result =0 then 1820 ' muerte
3071 if result =1 then 2060 ' fin nivel
```

6 Lógica del ciclo de juego

El ciclo de juego de "Paco el hombre" usa intensivamente lógicas masivas. Funciona como un motor de 4 tiempos, es decir, cada 4 ciclos todo vuelve a empezar y en cada uno de los 4 ciclos se hacen cosas diferentes.

La siguiente tabla muestra con una "X" las tareas que se llevan a cabo en cada ciclo. En un programa convencional (sin lógicas masivas) durante un solo ciclo de juego se realizarían las 8 tareas. Sin embargo, en este videojuego se ejecutan solo 2. Es decir, de una forma "simplista" se puede decir que funciona 4 veces más rápido que un juego programado de un modo convencional

	ciclos			
Tarea	**0**	**1**	**2**	**3**
Lectura teclado	X			
Detección colisión con fantasmas y con sprites invisibles		X		
Detección colisión con layout y recolocación si Paco choca			X	
Detección de puntos (cocos)				X
Lógica fantasma 1	X			
Lógica fantasma 2		X		
Lógica fantasma 3			X	
Lógica fantasma 4				X

Como puedes ver en cada ciclo se ejecuta una tarea que concierne a Paco y una lógica de uno de los 4 fantasmas.

En el momento en que un fantasma se encuentra en una bifurcación es importante que se ejecute la lógica de dicho fantasma, de lo contrario no podrá decidir qué camino escoger. Podemos colocar inicialmente los fantasmas de forma astuta para que cada vez que se encuentren en una bifurcación, se ejecute su lógica.

Primero vamos a tratar de explicar esto con el movimiento horizontal y después con el vertical

Horizontal:

Las bifurcaciones del laberinto siempre se encuentran en coordenadas múltiplos de 4.

La siguiente tabla resume la estrategia. Aparecen las posiciones de cada fantasma y los cuadros verdes indican en que ciclos se ejecuta su lógica. Como puedes comprobar cada vez que un fantasma se encuentra en un múltiplo de 4, su lógica se ejecuta. Y todo es gracias a la posición inicial que se le ha dado a cada fantasma

Para comprender la tabla debes tener en cuenta que mediante la técnica de movimiento de medio byte, un fantasma se va a mover 1 byte cada dos ciclos de juego. Eso significa que, por ejemplo, el fantasma 1 lo podemos ubicar inicialmente en una coordenada múltiplo de 4 y tardará 8 ciclos en llegar a otra posición múltiplo de 4. En ese momento se ejecutará la lógica del fantasma 1 pues de hecho se ejecuta cada 4 ciclos

Es importante considerar tanto la posición como la dirección del movimiento inicial para rellenar la tabla

ciclo	**Fant 1**	**Fant2**	**Fant3**	**Fant4**
C=0	4 (hacia derecha)	4*17 (hacia izq)	4*17-1 (hacia izq)	4+1 (hacia dere)
C=1	4	4*17	4*17-1	4+1
C=2	5	4*17+1	4*17-2	4+2
C=3	5	4*17+1	4*17-2	4+2
C=4	6	4*17+2	4*17-3	4+3
C=5	6	4*17+2	4*17-3	4+3
C=6	7	4*17+3	**4*17-4**	4+4
C=7	7	4*17+3	4*17-4	**4+4**
C=8	**8**	4*17+4	4*17-5	4+5
C=9		**4*17+4**	4*17-5	4+5
C=10			4*17-6	4+6
C=11			4*17-6	4+5

Vertical

En este caso las bifurcaciones del laberinto siempre se encuentran en coordenadas múltiplos de 8.

Cualquier fantasma en movimiento vertical tarda 8 ciclos en alcanzar la siguiente posible bifurcación ya que, aunque se mueve cada dos ciclos, lo hace dos líneas cada vez.

Puesto que todos los fantasmas comienzan en dirección horizontal, habrá un primer fantasma que decida cambiar de rumbo y escoja dirección vertical en una bifurcación. Esto no puede ocurrir simultáneamente en dos fantasmas porque solo se ejecuta una lógica a la vez, de modo que cuando otro fantasma entre en dirección vertical, el primero al menos le llevará un ciclo de ventaja y así sucesivamente, de modo que si la dirección inicial de todos los fantasmas es horizontal, entonces pueden empezar los 4 fantasmas en una posición Y múltiplo de 8

Vamos a suponer que el fant1 es el primero en moverse en vertical, después el fant2, después el 3 y después el 4. En rojo he puesto los momentos en los que el fantasma tiene movimiento vertical y además la coordenada Y es múltiplo de 8. Como ves, siempre coincide con un ciclo en el que se ejecuta su lógica. El fantasma 3 pasa a dirección vertical en el ciclo 6 porque antes es imposible que se encuentre en bifurcación (debe tener x múltiplo de 4). Como puedes comprobar, siempre que la coordenada Y es múltiplo de 8, se ejecuta una lógica de ese fantasma. Ten en cuenta que un fantasma pasa dos ciclos en cada coordenada, de modo que, si no es en el primer ciclo, se ejecuta en el segundo.

ciclo	Fant 1	Fant2	Fant3	Fant4
C=0	**X=4 , Y=8**	X=4*17 , y=8	X=4*17-1, y=22*8	X=4+1 ,y=22*8
C=1	X=4, y=8	**X=4*17, Y=8**	X=4*17-1, y=22*8	X=4 +1, y=22*8
C=2	X=4, y=10	X=4*17, Y=8	X=4*17-2, y=22*8	X=4+2, y=22*8
C=3	X=4, y=10	X=4*17,y=10	X=4*17-2, y=22*8	X=4+2, y=22*8
C=4	X=4, y=12	X=4*17,y=10	X=4*17-3, y=22*8	X=4+3, y=22*8
C=5	X=4, y=12	X=4*17, y=12	X=4*17-3, y=22*8	X=4+3, y=22*8
C=6	X=4, y=14	X=4*17, y=12	**X=4*17-4, y=22*8**	X=4+4, y=22*8
C=7	X=4, y=16	X=4*17, y=14	X=4*17-4, y=22*8	**X=4+4, y=22*8**
C=8	**X=4, y=16**	X=4*17,y=14	X=4*17-4, y=22*8+2	X=4+4, y=22*8
C=9	X=4, y=18	**X=4*17, y=16**	X=4*17-4, y=22*8+2	X=4+4, y=22*8+2
C=10	X=4, y=18	X=4*17, y=16	X=4*17-4, y=22*8+4	X=4+4, y=22*8+2
C=11	X=4, y=20	X=4*17, y=18	X=4*17-4, y=22*8+4	X=4+4, y=22*8+4
C=12	X=4, y=20	X=4*17, y=18	X=4*17-4, y=22*8+6	X=4+4, y=22*8+4
C=13	X=4, y=22	X=4*17, y=20	X=4*17-4, y=22*8+6	X=4+4, y=22*8+6
C=14	X=4, y=22	X=4*17, y=20	**X=4*17-4, y=22*8+8**	X=4+4, y=22*8+6
C=15	X=4, y=24	X=4*17, y=22	X=4*17-4, y=22*8+8	**X=4+4, y=22*8+8**
C=16	**X=4, y=24**	X=4*17, y=22	X=4*17-4, y=22*8+10	X=4+4, y=22*8+8

Gracias a este esquema de reparto de tareas combinado con la colocación astuta de fantasmas en coordenadas iniciales es posible hacer que este "reloj" de 4 tiempos funcione sin problemas a la máxima velocidad que nos puede ofrecer el BASIC.

En un apartado posterior veremos cómo se puede lograr acelerar la lógica de los fantasmas, basándonos en precálculos.

6.1 Ciclo de juego: control de tareas

La lógica del ciclo de juego comienza en la línea 1100 y lo primero que hace es incrementar el 1 el número de ciclos. Después calcula una variable "n" que la usamos para saber en qué tiempo de los 4 tiempos estamos

```
1100 '--- CICLO DE JUEGO ---
1110 '======================
1120 'ciclo de juego n=0-> teclado + xy, n=1->colision fantasmas,
n=2->colision layout , n=3-->deteccion puntos + xy
1130 c=c+1:n=c AND 3
1140 CALL &71CE :'|AUTOALL
1150 IF n AND 1 THEN 1340 :' colision fantasmas(n=1) y puntos (n=3)
1160 IF n THEN 1190 ELSE 1260: 'teclado (n=0) y colay (n=2)
```

La misma variable "n" nos servirá para elegir que fantasma mover en la rutina de logica de fantasmas, en la línea 1480. El Sprite id del fantasma se calcula sumando 27 + n

```
1480 '--- rutina fantasmas---(cada frame)
1481 '===================================
1490 if level=2 then 5000 else id=27+n
```

6.2 Control del personaje (n=0)

La lectura del teclado se realiza cada 4 ciclos, de modo que a veces la respuesta es mas instantánea que otras, sin embargo ante una bifurcación la respuesta es inmediata porque el ciclo en que se ejecuta la lectura del teclado coincide con los momentos en los que paco puede estar en una bifurcación (n=0)

Esta tarea la encontramos a partir de la línea 1250. Lo primero que vemos es el calculo de FPS que se realiza cada 64 ciclos, para interferir lo mínimo con el juego y no restarle velocidad.

En función de la variable "dir" que contiene la dirección del personaje, leemos el teclado. Por ejemplo si paco va hacia la derecha (dir=0) entonces no leemos la tecla "P" que precisamente sirve para movernos a la derecha. Asi ahorramos instrucciones.

- Dir=0 : derecha
- Dir=1 : izquierda
- Dir=2: arriba
- Dir=3: abajo
- Dir=4 : parado

```
1250 '---rutina teclado --- (n=0)
1251 '============================
1260 IF c AND 63 THEN 1270 ELSE FPS=0.5+64*300/(TIME -
A):cfps$=STR$(FPS):|PRINTAT,0,190,73,@cfps$:A=TIME

1270 IF dir THEN IF INKEY(27)=0 THEN CALL &753C,31,0,1:CALL
&71FD,31:CALL &753C,31,0,-1:IF col=0 THEN |SETUPSP,31,15,0:
dir=0:GOTO 1490 ELSE 1490

1280 IF dir <>1 THEN IF INKEY(34)=0 THEN CALL &753C,31,0,-1:CALL
&71FD,31:CALL &753C,31,0,1:IF col=0 THEN |SETUPSP,31,15,1:
dir=1:GOTO 1490 else 1490

1290 IF dir <>2 THEN IF INKEY(67)=0 THEN CALL &753C,31,-1,0:CALL
&71FD,31:CALL &753C,31,1,0:IF col=0 THEN |SETUPSP,31,15,2:
dir=2:GOTO 1490 ELSE 1490

1300 IF dir <3 THEN IF INKEY(69)=0 THEN CALL &753C,31, 1,0:CALL
&71FD,31:CALL &753C,31,-1,0: IF col=0 THEN |SETUPSP,31,15,3:
dir=3:GOTO 1490 else 1490
1310 GOTO 1490
```

6.3 Detección colisiones (n=1)

La detección de colisiones es sencilla y la tenemos en la línea 1340. En caso de colision nos vamos a la 1810

```
1320 '--- detecta puntos (n=3) y colision fantasmas (n=1)
1321 '================================================
```

```
1330 ' deteccion puntos tiene en cuenta dir para no borrar el punto
sin comerlo o comerlo sin borrarlo
1340 IF n=1 THEN CALL &73A8,31:IF cod<32 THEN 1810 ELSE 1490'
colision cada 4 frames (n=1)
```

En la línea 1810 nos encontramos con la rutina de tratamiento de la colision. Se diferencia entre la colision con un Sprite invisible que nos hace desplazarnos mágicamente al otro extremo de la pantalla o bien la colision con un fantasma que nos causa la muerte de Paco

```
1800 '--- rutina colision y muerte---
1801 '================================
1810 IF cod<27 THEN IF cod=26 THEN |MOVER,31,0,-80:GOTO 1490 ELSE
|MOVER,31,0,80:GOTO 1490
```

A partir de la línea 1820 comienza la parte del programa dedicada a la muerte, en la que Paco da vueltas sobre si mismo y pierde una vida. No la voy a comentar porque es muy sencilla

6.4 Colision con layout y recolocación (n=2)

Esta tarea chequea la colision con el layout y en caso de producirse, recoloca a Paco para que no destruya el laberinto. Obviamente se ejecuta ANTES de la instrucción que imprime todos los sprites y que se encuentra en la línea 1780 (el PRINTSPALL).

Como puedes ver la recolocación depende de la dirección en la que se mueve paco y es de 1 byte (movimiento horizontal) o 4 lineas (movimiento vertical)

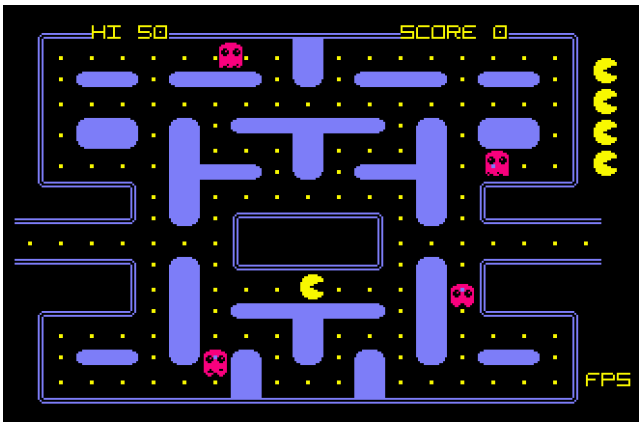
```
1170 '--- rutina recolocacion --- (n=2)
1171 '================================
1190 CALL &71FD,31:IF col THEN 1200 ELSE 1490 :'COLAY
1200 |SETUPSP,31,15,4: ON dir GOTO 1220,1230,1240 :' ruta 4 es
parado
1210 |MOVER,31,0,-1:GOTO 1490
1220 |MOVER,31,0,1:GOTO 1490
1230 |MOVER,31,4,0:GOTO 1490
1240 |MOVER,31,-4,0:GOTO 1490
```

6.5 Detección y control de puntos (n=3)

La detección y control de puntos se basa en una consulta al layout , que es una matriz de caracteres, para lo cual averiguamos que posición del layout debemos consultar en función de las coordenadas de Paco

Un punto en el layout son dos tipos de caracteres. Concretamente el carácter **";"** para los puntos situados en caminos horizontales y el carácter **"<"** para los puntos situados en caminos verticales

Puedes ver la correspondencia entre el laberinto y sus cocos y el texto con el que se ha construido el layout




```
"=AAAAAAAADAAAAAAAA> "
"F        O        E "
"F;;;;;;;;O;;;;;;;;E "
"F<RS<ROS<Q<ROS<RS<E "
"F                 E "
"F;;;;;;;;;;;;;;;;;E "
"F KL P<ROOOS<P KL E "
"F;MN;O   O   O;MN;E "
"F    O;;;O;;;O    E "
"F;;;;OOS<Q<ROO;;;;E "
"?BBG O       O HBB@ "
"   F;O;;;;;;;O;E    "
"AAAI<Q<HBBBG<Q<JAAAA"
"       E   F        "
";;;;;;;E   F;;;;;;; "
"BBBG<P<JAAAI<P<HBBBB"
"   F O       O E    "
"   F;O;;; ;;;O;E    "
"=AAI<O<ROOOS<O<JAA> "
"F    O   O   O    E "
"F;;;;O;;;O;;;O;;;;E "
"F<RS<Q<P<Q<P<Q<RS<E "
"F      O   O      E "
"F;;;;;;O;;;O;;;;;;E "
"?BBBBBBCBBBCBBBBBB@ "
```

Para acelerar el calculo usamos una tabla llamada chache que se inicializa al principio del programa. En caso de encontrar un punto, lo reemplazamos escribiendo un espacio en su lugar (con un poke d, 32)

```
1320 '--- detecta puntos (n=3) y colision fantasmas (n=1)
1321 '================================================
1330 ' deteccion puntos tiene en cuenta dir para no borrar el punto
sin comerlo o comerlo sin borrarlo
1340 IF n=1 THEN CALL &73A8,31:IF cod<32 THEN 1810 ELSE 1490 '
colision cada 4 frames (n=1)
1350 |PEEK,27499,@x:y=PEEK(27497)
1360 ON dir GOTO 1370,1380,1390
1369 y2=y+8:x2=x+2:GOTO 1400
1370 y2=y+8:x2=x:GOTO 1400
1380 y2=y:x2=x:GOTO 1400
1390 y2=y+12:x2=x
1400 d=cache(y2)+x2/4
1410 dato=PEEK(d)
1411 if inkey(38)=0 then if music then music=0:|MUSIC else
music=1:|MUSIC,0,0,0,6
1420 IF dato<60 THEN IF dato>32 THEN 1440 ELSE 1490
1430 '--rutina incremento puntos---
1440 POKE d,32:scr=scr+1:cad$=STR$(scr):|PRINTAT,0,0,60,@cad$: if
music then 1450 else SOUND 4,400,10,15,0,1
1441 'if dant<>(d-&a438) then print (d-&a438);y2;",";x;",";dir:
dant=d-&a438
'1442 locate 1,1:print d,dato, chr$(dato)
1450 IF scr=target THEN 2060
```

```
1460 IF hi>scr THEN 1490 ELSE 1470
1470 LOCATE 18,14:PEN 2:PRINT "NEW":LOCATE 16,15::PRINT "HI-
SCORE":hi=30000:flaghi=1
```

Hay una variable llamada “target” que contiene el número de puntos al que tenemos que llegar para pasar de nivel. Ese target se calcula simplemente sumando el numero de cocos del laberinto (que es un numero conocido =141) a los puntos que tenia Paco antes de entrar a cada nivel.

Para el primer nivel la variable target se inicializa precisamente a 141 en la línea 120

```
90  'VARIABLES GLOBALES
100 '====================
110 DIM L$(25)
120 hi=0:|PEEK,42038,@hi:flaghi=0:target=141:lasthi=hi:
130 level=1: vidas=4: newlevel=1:scr=0
131 music=0
```

6.6 Lógica de fantasmas (n= cualquier valor)

Esta tarea se ejecuta en todos los ciclos de juego pero en cada ciclo se ejecuta la lógica de un solo fantasma. Ya hemos visto como se escoge el fantasma (sumando 27 + n)

La lógica del fantasma es determinista. Cada vez que encuentra una bifurcación, se calcula la ruta a escoger con la formula

newr=1+newr AND 3

siendo newr un numero entre 1 y 4 (hay 4 posibles rutas: derecha , izquierda, arriba y abajo). Si es imposible escoger esa ruta por haber un muro, se intenta con la siguiente. Lo que nunca se permite es volver atrás, igual que en el juego original.

Las rutas de los fantasmas no son la variable newr, sino que newr nos permite decidir la ruta. La verdadera ruta es un numero entre 5 y 8 y se almacena en el array rf(n), es decir para cada valor de n (= para cada fantasma) el valor de su ruta es rf(n).

Al final de la lógica de fantasma se imprimen todos los sprites y se vuelve al inicio del ciclo de juego

```
1480 '--- rutina fantasmas---(cada frame)
1481 '==================================
1490 id=27+n
1490 locate 1,1:print n: input l:goto 4000
1490 goto 4000
1490 locate 1,1:print n: input l:goto 5000
1490 if level=2 then 5000 else id=27+n

1500 IF rf(n)<7 THEN xf=PEEK(dfx(n)):IF xf AND 3 THEN 1780 ELSE ELSE
1650:'acelera un poco. dir fant x
1510 IF rf(n)>5 THEN 1590
1520 newr=1+newr AND 3
1530 ON newr GOTO 1550,1560:'ruta 5 (derecha)
```

```
1540 CALL &753C,id,0,1:CALL &71FD,id:CALL &753C,id,0,-1: IF col THEN
1550 ELSE 1780:' sigue
1550 CALL &753C,id,-1,0:CALL &71FD,id:CALL &753C,id,1,0: IF col THEN
1560 ELSE rf(n)=7:CALL &70FD,id,15,7:GOTO 1780
1560 CALL &753C,id,1,0:CALL &71FD,id:CALL &753C,id,-1,0: IF col THEN
1570 ELSE rf(n)=8:CALL &70FD,id,15,8:GOTO 1780
1570 IF xf >75 THEN |MOVER,id,0,-80:GOTO 1780
1580 GOTO 1540
1590 IF xf<200 THEN newr=1+newr AND 3 ELSE |MOVER,id,0,80:GOTO 1780
1600 ON newr GOTO 1620,1630: 'rutas 6 izq
1610 CALL &753C,id,0,-1:CALL &71FD,id:CALL &753C,id,0,1: IF col=0
THEN 1780:' sigue
1620 CALL &753C,id,-1,0:CALL &71FD,id:CALL &753C,id,1,0: IF col=0
THEN rf(n)=7:CALL &70FD,id,15,7:GOTO 1780
1630 CALL &753C,id,1,0:CALL &71FD,id:CALL &753C,id,-1,0: IF col=0
THEN rf(n)=8:CALL &70FD,id,15,8:GOTO 1780
1640 GOTO 1610
1650 yf=PEEK(dfy(n)):IF yf AND 7 THEN 1780 ELSE IF rf(n)=8 THEN 1720
1660 newr=1+newr AND 3
1670 ON newr GOTO 1690,1700: 'rutas 7 up
1680 CALL &753C,id,-1,0:CALL &71FD,id:CALL &753C,id,1,0: IF col=0
THEN 1780:' sigue
1690 CALL &753C,id,0,1:CALL &71FD,id:CALL &753C,id,0,-1: IF col=0
THEN rf(n)=5:CALL &70FD,id,15,5:GOTO 1780
1700 CALL &753C,id,0,-1:CALL &71FD,id:CALL &753C,id,0,1: IF col=0
THEN rf(n)=6:CALL &70FD,id,15,6:GOTO 1780
1710 GOTO 1680
1720 newr=1+newr AND 3
1730 ON newr GOTO 1750,1760: 'rutas 7 down
1740 CALL &753C,id,1,0:CALL &71FD,id:CALL &753C,id,-1,0: IF col=0
THEN 1780:' sigue
1750 CALL &753C,id,0,1:CALL &71FD,id:CALL &753C,id,0,-1: IF col=0
THEN rf(n)=5:CALL &70FD,id,15,5:GOTO 1780
1760 CALL &753C,id,0,-1:CALL &71FD,id:CALL &753C,id,0,1: IF col=0
THEN rf(n)=6:CALL &70FD,id,15,6:GOTO 1780
1770 GOTO 1740
1780 CALL &62A6:'printspall
1790 GOTO 1130:'inicio ciclo
```

En el nivel 2 de BASIC (y también en el nivel 4 de ciclo compilado) se usa una técnica para acelerar las decisiones de los fantasmas. Consiste en haber guardado previamente las decisiones de los fantasmas en cada ciclo de juego, con una versión modificada de esta lógica. Con las decisiones guardadas en memoria, podemos ejecutar la lógica de fantasmas simplemente leyendo lo que toca hacer en cada ciclo, sin necesidad de detectar colisión con layout para saber si se puede mover en determinada dirección. Gracias a esta técnica podemos pasar de 14 fps a 19 fps en BASIC.

El almacenamiento de las decisiones usa un mecanismo para comprimir la información. Consiste en que como la decisión de “seguir en la misma dirección” es la decisión mas frecuente, usamos un solo dato para almacenar muchas decisiones. Es decir

- Dato = 1 : decisión= asignar ruta derecha

- Dato = 2 : decisión= asignar ruta izquierda
- Dato = 3 : decisión= túnel , mover 80 bytes a la izquierda
- Dato = 4 : decisión= túnel , mover 80 bytes a la derecha
- Dato= 5 : decisión= asignar ruta arriba
- Dato= 6: decisión = asignar ruta abajo
- Dato= 7 : decisión = no cambiar nada , es decir, seguir en ruta actual
- Dato= 8 : dos decisiones seguidas sin cambiar nada
- Dato= 9 : tres decisiones seguidas sin cambiar nada
- Dato= 10 : cuatro decisiones seguidas sin cambiar nada
  Y asi sucesivamente
- Dato = 100 : 94 decisiones seguidas sin cambiar nada

Con este mecanismo en 2000 bytes almacenamos 6712 decisiones, es decir 6712 ciclos de juego. Una vez consumidas estas 6712 decisiones, Paco se muere, pero es muchísimo tiempo.

```
4999 'decisiones precomputadas logica fantasmas
5000 if op<8 then op=peek (&9c40+co) else op=op-1:CALL &62A6:GOTO
1130
5001 co=co+1:if co=2000 then 1820:'muerte (co=2000 ->6712 ciclos)
5002 if op>6 then CALL &62A6:GOTO 1130
5010 id=27+n:if op>4 then CALL &70FD,id,15,op:CALL &62A6:GOTO 1130
'5 y 6
5011 on op goto 5020,5030
5012 if op=3 then |MOVER,id,0,-80:CALL &62A6:GOTO 1130 '3
5013 |MOVER,id,0,80:CALL &62A6:GOTO 1130 '4
5020 CALL &70FD,id,15,7:CALL &62A6:GOTO 1130 '1
5030 CALL &70FD,id,15,8:CALL &62A6:GOTO 1130 '2
```

7 Invocación del ciclo de juego compilado

El ciclo de juego en C es prácticamente idéntico al de BASIC, y no vamos a comentarlo, pero al menos la forma en la que se invoca y se recogen datos a su terminación es interesante entenderlo

Para pasarle variables que desde C se puedan modificar lo que hacemos es pasar la dirección de memoria donde están almacenadas, con el modificador “@”

```
3000 '---ciclo en C
3001 '=============
3010 'pasamos direcciones de variables para escribirlas desde c
3020 |POKE,17900,@scr
3030 |POKE,17902,@target
3031 |POKE,17904,@music
3032 |POKE,17906,@hi
3033 |POKE,17908,@flaghi
3034 if level>=4 then |POKE,17914,1 else |POKE,17914,0
3040 call &4d0d:'llamada al ciclo compilado
3050 'recogida resultados
```

```
3055 |peek,17912,@c:'ciclos consumidos
3060 result=0:|peek,17910,@result:'muerte (0) o target alcanzado (1)
3070 if result =0 then 1820 ' muerte
3071 if result =1 then 2060 ' fin nivel
```

En cuanto a la recogida de datos, como los ciclos consumidos o el resultado (muerte o misión cumplida) los recogemos con PEEK

8 Rutas

Vamos a revisar las rutas de sprites del videojuego con una tabla, donde encontraras todas las rutas de enemigos y de Paco.

En total el juego tiene 13 rutas definidas. Se encuentran en el fichero routes_mygame.asm

rutas	descripción
0-4	Rutas de Paco (5 rutas, la numero 4 significa quieto)
5-8	Rutas de los fantasmas
9-12	Rutas para dibujar el texto paco en la presentación antes de jugar
13	Ruta para las bolas cayendo en la presentación

Vamos a ver unas rutas interesantes, la de Paco hacia la derecha y la del fantasma hacia la derecha. Son parecidas, usan movimiento suave de medio byte con la técnica anteriormente explicada

```
ROUTE0; derecha
      db 253
      dw COCO_R1
      db 1,0,0 ; no se mueve
      db 253
      dw COCO_R2
      db 1,0,1  ; se mueve
      db 253
      dw COCO_R1
      db 1,0,0  ; no se mueve
      db 253
      dw COCO_R0
      db 1,0,1  ; se mueve
      db 0
```

```
ROUTE5
      db 253
      dw FANT_R1
      db 1,0,0; no se mueve
      db 253
      dw FANT_R0
      db 1,0,1; se mueve
      db 0
```

Y ahora vamos a ver la ruta arriba de paco y la de un fantasma

```
ROUTE2;UP
      db 253
      dw COCO_UP1
      db 1,0,0; no se mueve
      db 253
```

```
ROUTE7;UP
      db 253
      dw FANT_UP1
      db 1,0,0; no se mueve
      db 253
```

```
        dw COCO_UP2
        db 1,-4,0; se mueve
        db 253
        dw COCO_UP1
        db 1,0,0; no se mueve
        db 253
        dw COCO_UP0
        db 1,-4,0; se mueve
        db 0
```

```
        dw FANT_UP0
        db 1,-4,0; se mueve
        db 0
```

9 ¡A Jugar y a programar!

Hemos visto como está estructurado el videojuego “Paco el hombre” y cuáles son sus mecanismos de funcionamiento. El control de tareas para alcanzar la máxima velocidad, el movimiento suave de medio byte y la aceleración con decisiones de fantasmas precalculadas son las cosas más interesantes de programar que caracterizan este videojuego.

Es el momento de comenzar tu propio juego. Un consejo: No olvides dedicarles tiempo a los gráficos, pues un juego con dibujos atractivos es mucho más agradable y hasta divertido.

¡A jugar y a programar!