Making of

WRECKING BALL

by Francesc Alcaucer

Contenido

Introducción

Wrecking Ball es un videojuego tipo *machaca-ladrillos* desarrollado en ***Locomotive Basic*** para los ordenadores ***Amstrad CPC***. Este juego utiliza una técnica de gráficos (que yo creía habitual en juegos en código máquina), que consigue **mediante UDG y algunos *pokes* a direcciones de control del firmware crear gráficos coloridos sin utilizar el efecto de transparencia de texto**.

Dado que el Basic en el *Amstrad CPC* es un lenguaje de programación interpretado, en el código no he añadido comentarios con la pretensión de no ralentizar innecesariamente su ejecución, pero aquí se tratará de separar las partes principales y dar algunas explicaciones de su funcionamiento.

La concepción de *Wrecking Ball* tuvo lugar en los estadios finales de mi anterior desarrollo, también en *Locomotive Basic*, ***CPC Invaders***, mientras pensaba la mejor manera de que el fondo pudiera ser repuesto durante la acción del juego. Es cuando pensé en un juego tipo *Arkanoid* y si no sería posible conservar el fondo de la pantalla pintando un carácter almacenado en un array tipo texto, escogiendo el carácter *N* a partir de la coordenada *X* de la bola y siendo el elemento *N'* del array la coordenada *Y*, ejecutándose a una buena velocidad.

Lo primero que hice fue programar una zona de juego con obstáculos simples y una pelota rebotando (el carácter 231 del *Amstrad CPC*). Con 4 colores del modo 1 del CPC no me pareció que fuera a ser muy vistoso, dedicando uno de esos colores al fondo (aunque pensaba utilizar un patrón repetitivo con tramado al estilo de Arkanoid), sólo quedaban 2 colores más para el resto de la acción, necesitaba más colorido, que obtendría con el modo 0.

Como decía más arriba, **se supone** que para conseguir gráficos UDG multicolores hay que utilizar el modo de transparencia de texto y diferentes caracteres con puntos definidos de acuerdo con los diferentes colores que se quieren mostrar, **pero con este proyecto he descubierto que esto no es así**.

Recordando las pruebas que había hecho 30 años atrás con el modo 0, en las que ***pokeaba* los registros del CPC para que el firmware 'creyera' que estaba trabajando en modo 1** (esto lo había utilizado en algún juego sencillo, para que los marcadores no se vieran tan enormes con esa horrible fuente achatada del modo 0, al estilo del marcador del juego *Rampage* de *Activision* en CPC), **se consigue un sorprendente efecto secundario: los caracteres se muestran en varios colores**; en lugar de sólo el color del primer plano y el fondo, aparecen otros... ¿Sería posible definir qué colores debían aparecer o era sólo un *bug* incontrolable?

Otro de los objetivos que me propuse abordar con este proyecto es el de tener especial cuidado en el sonido y la música, dotándolo de sonido posicional (*ejem*, más o menos) y de una melodía más elaborada que en mi anterior proyecto, en el que sólo se emitían tonos repetitivos y sonidos a modo de batería en momentos fijos.

Argumento

Estamos en el año 3021 y la basura espacial alrededor de la tierra ha llegado a ser tan numerosa que se están formando bloques enormes de material en circulación en las inmediaciones. Estos bloques suponen no sólo un riesgo de colisión contra estaciones espaciales, naves tripuladas o satélites alrededor de la órbita, sino que podrían descender de altitud y llegar a la superficie del planeta, con fatales resultados. Se te ha designado para ser el primer operador de bola de demolición espacial, una enorme (y peligrosa) masa esférica resultado de un proyecto secreto con una gran capacidad de desintegración…

Gráficos

Hack de texto en modo 0

Empezaré aclarando que **el desarrollo lo inicié emulando el modelo 6128 del Amstrad CPC**. Como recurso de consulta cuando tenía el CPC, me compré (y aún conservo) el libro publicado en 1987 por **Anaya Multimedia**, ***Amstrad CPC 464/664/6128. Manual de referencia avanzado***, de **Rafael Sarmiento de Sotomayor**, en este, entre otros temas, se tratan las rutinas y direcciones del firmware de los CPC. En la parte central de la página 153, aparecen las direcciones de memoria de control del bloque referente a la pantalla.

Haciendo *MODE 0:POKE &B7C3,1* (esta dirección controla en el 6128 el modo de pantalla activo), el cursor de texto pasa de rectangular a ser cuadrado, como en el modo 1, y haciendo luego *PEN 2*, obtenemos un interesante y colorido *Ready*:

Si probamos con diferentes valores para *PEN,* observamos que el comportamiento es similar al del modo 1, mostrándose texto con los valores de 1 a 3 y con valor 4 equivale a *PEN 0* y la secuencia se repite en valores superiores. Cada vez que cambiamos a un valor en los que se muestra texto, si afinamos la vista podemos ver que lo hace mostrando puntos en 4 tintas diferentes (contando el color de fondo):

Bien, el valor de tinta del fondo siempre es el mismo, pero si los otros varían y 3 valores son válidos, tenemos 9 tintas, ¿qué pasa con el resto? Hagamos la siguiente prueba:

```
10 MODE 0:PEN #4,6:PEN #5,13:PEN #6,11:PEN #7,14
20 POKE &B7C3,1:PEN #1,1:PEN #2,2:PEN #3,3
30 FOR n=1 to 7:LOCATE #n,1,10+n:PRINT #n,"Ready":NEXT
```

Con esto establecemos tintas para *PEN* antes y después del *POKE*. El resultado es el siguiente:

Vemos que han ocurrido dos cosas: mientras tecleamos el cursor aparece en la siguiente línea cuando va más allá de la posición horizontal 20 y tenemos textos con diferentes combinaciones de tintas. Para solventar lo de las 20 columnas lo que hago es redefinir todas las ventanas con

WINDOW #n,1,40,1,25 y con eso ya tengo las 40 columnas. Para la ventana de gráficos ocurre algo parecido, pero se soluciona con la instrucción: *ORIGIN 0,0,0,640,400,0*

Tras diferentes pruebas, obtengo la siguiente tabla:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X			X	X										
2		X						X		X					
3			X									X			X
4				X											
5	X			X	X										
6		X		X		X									
7			X	X			X								
8								X							
9	X							X	X						
10		X						X		X					
11			X					X			X				
12												X			
13	X											X	X		
14		X										X		X	
15			X									X			X

En el eje vertical tenemos los valores posibles para *PEN* y en el eje horizontal tenemos con qué tintas se mostrarán los caracteres que aparecerán con dada uno. Como podemos ver, hay dos pequeños problemas y es que hay tintas que se repiten con diferentes valores de *PEN* y que los valores 4, 8 y 12 sólo muestran 1 tinta.

Decido entonces tomar la siguiente estrategia: cargar las tintas de manera que pueda representar caracteres con colores con cierta gradación, colores en tintas de 0 a 15:

0 4 3 2 17 8 6 16 24 12 15 18 26 13 7 11

Trasladando el color a la tabla anterior, tenemos las siguientes combinaciones multicolores:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X			X	X										
2		X						X		X					
3			X									X			X
4				X											
5	X			X	X										
W6		X		X		X									
7			X	X			X								
8								X							
9	X							X	X						
10		X						X		X					
11			X					X			X				
12												X			
13	X											X	X		
14		X										X		X	
15			X									X			X

Editor UDG para el *hack* de texto

Para facilitarme la tarea de investigar cómo se presenta el texto en pantalla cuando utilizo este pequeño *hack* y para ir creando los primeros gráficos, me hago un editor UDG sencillito, quedando de la siguiente manera:

En la parte superior-izquierda tenemos la matriz del carácter de 8x8 puntos, justo a su derecha tenemos el mismo carácter cuando tenemos el *hack* de texto, mostrando un carácter de 4x8 puntos.

Si nos fijamos bien, podemos ver que cada fila del carácter que definimos se divide en dos *nibbles* que representan los dos píxeles de la izquierda y los dos de la derecha del carácter, consiguiendo una tinta u otra, diremos que (con la paleta de colores que tenemos aplicada) de intensidad baja, media o alta, dependiendo de qué bits activemos en cada *nibble.*

Con un poco más de esfuerzo y evolucionando la herramienta, consigo el siguiente resultado y ahora me permite exportar a fichero .BAS con formato de texto ASCII con todos los comandos *SYMBOL*:

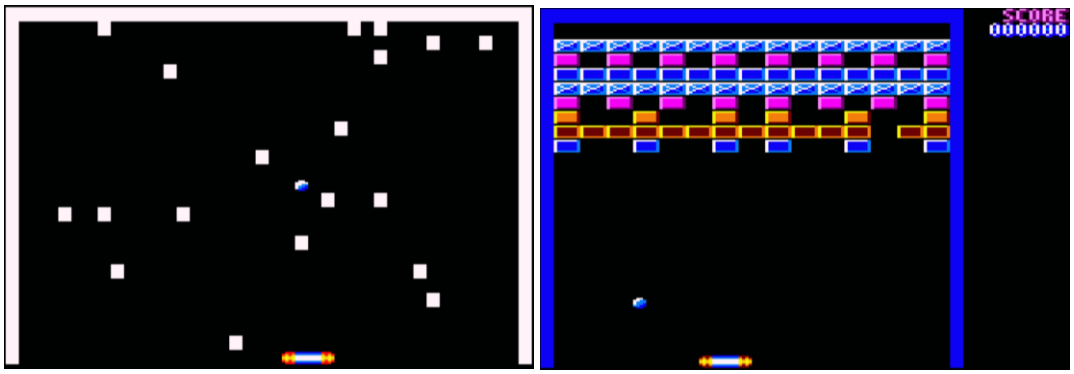
En conclusión, lo que tengo es un editor UDG que me permite generar gráficos en 4 colores (3 de primer plano además del fondo) gracias al *hack* de texto del modo 0, para mostrar '*sprites*' imprimiendo en pantalla un único carácter con 4 colores, en lugar de tener que imprimir 3 con el modo de transparencia de texto. Según mis pruebas, imprimir un único carácter es 1.9 veces

más rápido. No lo he probado, pero imagino que con cadenas de texto más largas habrá más diferencia.

Hay que tener en cuenta que como es lógico pensar, este truco se puede utilizar también desde programas escritos en otros lenguajes, siempre que el firmware no sea desactivado.

Evolución de los gráficos del juego

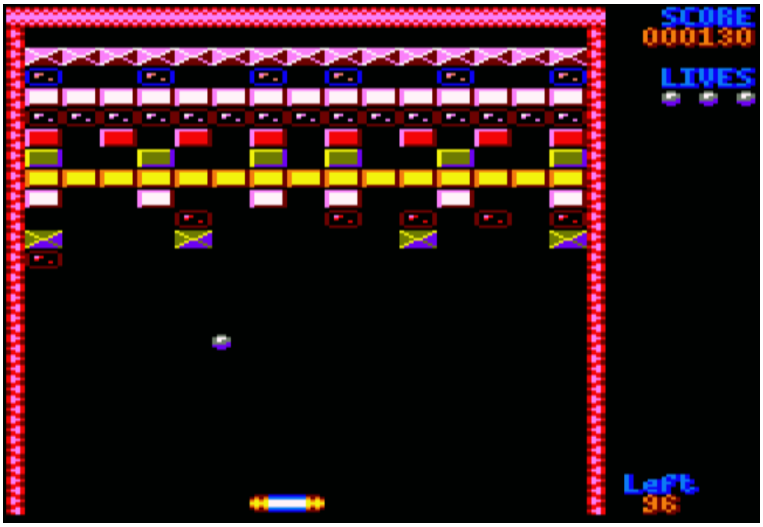
Aquí la primera y segunda maqueta 😊:



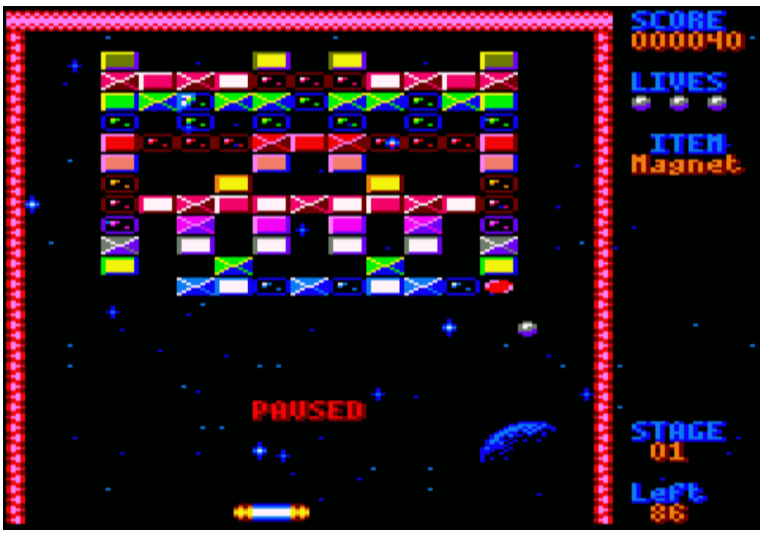
Realizo un nuevo diseño UDG de la bola, pasa de la v1 a la v3, que me agrada más y dejo definitivamente ya que me parece que tiene aspecto metálico:

v1  v2  v3 

Rediseño también el borde de la zona de juego y los ***ladrillos*** que se habrán de destruir con diferentes características de *dureza,* añado contador de los que restan por eliminar y doy vida al marcador de puntos. De momento la bola nunca se pierde, por lo que no descienden las vidas:



La cosa va pintando bien, incorporo la pausa, más elementos del marcador y ladrillos transparentes (el *hack* de texto produce un curioso efecto de transparencia que no queda mal), tras los que se muestra el fondo, fondo que consiste en una variable tipo texto *DIM*ensionada de 40 caracteres de longitud por cada elemento y que se redibuja con el movimiento de la bola o al destruir ladrillos, obteniendo el carácter con *MID$* de la variable dimensionada, sin que la velocidad se resienta.



Primera prueba de gráficos en el modelo CPC 464 (no sale bien)

Me dispongo a hacer la primera prueba en el modelo *464*. Genero el CDT y lo cargo, cambio la dirección del *POKE de* B7C3 *a* B1C8, que corresponde al *464,* y me quedo de piedra cuando veo que aparece esto:

El horror. El firmware del 464 es diferente, aparte de por el juego de instrucciones del Basic. Ya había notado ciertas diferencias por ejemplo al imprimir un carácter en la esquina inferior-derecha, que en el 464 se produce un retorno de carro y en el *664/6128* no ocurre así y cosas por el estilo, pero esto ya… en fin, ¿por qué hace esto? Frustración y desesperación a partes iguales…

Búsqueda de ayuda profesional

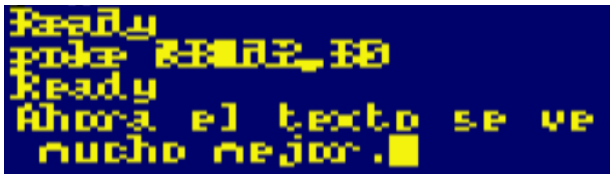
Un desastre, sin el *hack* de texto no hay juego, el impacto visual que suponía era decisivo y en el 464 no parecía funcionar… en este modelo es diferente, hace cosas raras, no hay texto multicolor, los píxeles de las letras parecen desplazados:

Entonces recupero mi bloc de notas de la época, una carpeta de anillas con más de 30 o 35 años y que aún conservo. En ese bloc guardo el diseño de las letras del *Arkanoid II* que reproduje en *CPC Invaders* y por el que me había dejado la vista copiando píxel a píxel desde la pantalla al papel, también contiene algunas contraseñas de juegos y anotaciones de fórmulas, pequeños listados en Basic, trucos con la instrucción *OUT* y *POKES*, entre los que está anotado lo siguiente:

Mode 0/1 special text design STR

```
10 mode 0
20 poke &B1C8,1:poke &B1D0,0:poke &B1D2,60
```

El modelo de CPC que yo tenía de pequeño era el de cinta, y todas las pruebas que hacía eran para ese modelo. Por aquel entonces ya había estado jugando con esto, pero no lo había explotado, no había investigado lo suficiente sobre lo que ocurría con el texto y las diferentes tintas porque no supe darme cuenta de nada. Lo que sí hice es jugar con esas otras direcciones (B1D0 y B1D2) porque parecía que el texto se veía mejor:



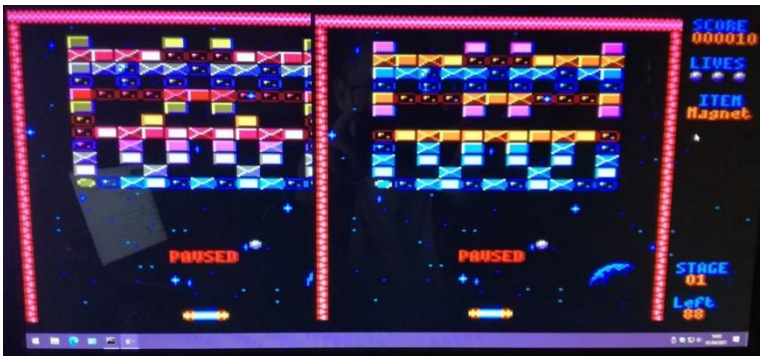
No lo recordaba, pero el texto lo podía ver también en colores y me gustaba como quedaba, pero ahí quedaba la cosa:

De vuelta al manual del firmware, ¿qué controlaban esas direcciones de memoria? Según dice el libro, las máscaras de puntos y cierto es que los puntos de los caracteres parecían desplazados antes de tocar las máscaras y tras hacerlo el texto multicolor también era más vistoso, por lo que algo tenían que ver en todo esto.

Las direcciones de memoria de control de máscara son las que van de B1CF a B1D6, pero al parecer sólo necesitaba las 4 primeras. Estuve probando diferentes valores y no conseguía que se viera igual que en el *6128*, modelo en el que además las máscaras no tenían efecto alguno y de hecho sus valores están a cero, así que no servía como referencia. Parecía que en el *464* los valores que podía asignar a *PEN* se comportaban como si estuviera en el modo 1, es decir, de 0 a 3 y en 4 se repetía la secuencia, con lo que el juego mostraría un colorido inferior al del modelo *6128* (recordemos la tabla de colores de la página 4). De nuevo, situación de frustración…

Busco en Internet y no parece que nadie haya hecho algo parecido con el firmware del *CPC*, estoy en un callejón sin salida, hasta que decido contactar con "*el profe*", Fran Gallego… "*si él no me soluciona esto nadie más me lo va a solucionar*", me dije.

Me pongo en contacto a través del grupo ***CPCtelera*** en ***Telegram***, explico mi problema al grupo y el profe muestra inmediatamente su interés por el asunto, doy un respingo y paso por privado una foto de la pantalla donde se ve el proyecto en los dos modelos, *6128* (izquierda) y *464* (derecha):



El *profe* me va preguntando cosas y pasa a hacer sus propias pruebas y ver el efecto, e incluso repasa las rutinas de CM del firmware que se encargan de imprimir los caracteres en pantalla para ver cómo trabajan... Según va explicando voy viendo y probando cosas, hasta que, ¡sorpresa! …resulta que la máscara por defecto cuando está puesto el *MODE 1* (*&88, &44, &22, &11*) es la que tenía que utilizar en *MODE 0*, pero para establecer la tinta que hay que utilizar no debo utilizar la instrucción *PEN*, sino un ***POKE &B28F,tinta***

Para establecer las tintas relativas a otra ventana (*WINDOW #n)*, lo que tengo que hacer es cambiar el ***cauce actual*** de texto, dirección de memoria ***B20C:***

```
POKE &B20C,ventana:POKE &B28F,tinta
```

Bueno, ya está todo casi solucionado, ¡mil gracias al profe por iluminarme!

Lo siguiente, dado que el mismo programa debe poderse ejecutar en todos los modelos, es identificar en qué versión de firmware se está ejecutando el juego y establecer variables para las direcciones de memoria que importan, cosa fácil:

```
IF &39=PEEK(&39)THEN
MODE.=&B1C8:CH.=&B20C:MASK.=&B1CF:PEN.=&B28F:REM 1.0
ELSE MODE.=&B7C3:CH.=&B6B5:MASK.=&B7CA:PEN.=&B72F:REM 1.1
```

Defino un array de 15 elementos (*DIM PEN.(15)*) con los valores de tinta que me interesan (y alguno extra) para el *POKE* a la dirección que controla la tinta, de 1 a 15:

```
&F0, &3C, &FC, &C3, &F, &CF, &F3, &3F, &FF, &3, &C, &CC, &C0, &30, &33
```

Sólo tengo que hacer esto para cambiar la tinta del cauce actual:

Desplazamiento de pantalla vertical al píxel

Ya desde antes de ponerme con *Wrecking ball*, mientras trabajaba en *CPC Invaders*, había hecho un pequeño programa que presentaba un *scroll* vertical con fondo estrellado que se movía al píxel (hacia arriba, todas las estrellas a la misma velocidad). El truco es muy sencillo, sólo hay que manipular los registros del CRTC debidamente para variar la posición de la pantalla.

Por ejemplo:

```
10 MODE 1
20 WHILE INKEY$=””:GOSUB 100
30 PRINT#7,CHR$(11):FOR n=1 TO 2:LOCATE#7,1+RND*40,1:PRINT#7,”.”;
40 WEND
50 END
100 FOR y=1 TO 7:CALL &BD19:OUT &BC00,5:OUT &BD00,y
110 FOR n=1 TO 50:NEXT
120 NEXT:OUT &BC00,5:OUT &BD00,0:RETURN
```

Para *CPC Invaders* no lo utilicé, porque se desplaza toda la pantalla y no tenía oportunidad de utilizarlo durante la acción, pero esta vez sí que lo he introducido en la secuencia de introducción al juego y que da paso a la pantalla de menú.

Pantalla de título

Para la pantalla del menú donde también se muestra el título del juego, quería que éste fuera con letras grandes, sin utilizar la misma fuente de letra del resto del juego. Había pensado en hacer algún texto chulo con efecto 3D en Gimp, pasarlo luego al Amstrad, y esta vez sí, presentarlo en pantalla con la ayuda del *hack* de texto y transparencias para utilizar todos los colores que fueran necesarios. Hice alguna prueba y rápida y me pareció que con la baja resolución del modo 0 no se iba a ver suficientemente claro y por otra parte, tenía que modificar mi editor UDG para que interpretara una imagen cargada y la convirtiera a matrices de caracteres con los píxeles que se necesitara mostrar para cada color… un derroche.

Me plantee entonces presentar el título con caracteres de manera que, unidos entre sí, conformaran las letras. La primera prueba para ver si podía caber todo fue tal que así:

Aparte de que las letras se ven bastante feas, el menú también se veía demasiado recargado. Reduje el número de opciones a las estrictamente necesarias y diseñé los caracteres a modo de piezas que tenían que dar forma a la fuente de 3x5 o 4x5, dependiendo de qué letra se trate:

Pantalla de carga y presentación

Se trata este de un juego tipo *machaca-ladrillos,* pero ¿qué sentido podía tener que la acción estuviera localizada en el espacio? Se me ocurrió que lo que se hubiera de destruir fuera basura espacial.

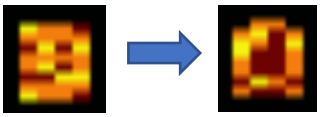
Durante su desarrollo y de casualidad me he enterado de noticias, etc. relativas al problema real con la basura especial. Resulta que ya existen empresas que se encargan de limpiar la órbita de desechos (por ej.: *Airbus*, *Astroscale* y *ClearSpace*). En la fase final del proyecto, justo antes de la presentación a la *CPCRetroDev*, tenía la pretensión de incluir una pantalla de carga y se me ocurrió visitar la página web de la Agencia Espacial Europea (ESA) y más en concreto un artículo publicado hablando del problema de la basura espacial (en él se comenta que ya ha tenido lugar la primera colisión en febrero de 2009).

Una de las imágenes que lo ilustran es la que he utilizado como base para la pantalla de carga. Con Paint.net he aumentado la saturación color, he subido los niveles de componente verde al globo terráqueo, he agregado la bola y convertido la imagen resultante con ***ConvImgCPC***. Ya en el CPC he añadido los textos:


Notar que las letras del título son las mismas que en la pantalla de menú del juego con el modo de transparencia de texto activado y que, con el *hack* de texto y modo, se consigue que se vean realmente como si fueran translúcidas (además, por las tintas resultantes de la mezcla la paleta de color aplicada a las tintas parece intencionada, pero es pura casualidad).

Elementos móviles durante la partida

Hacia la mitad del desarrollo introduje elementos que se movían por la pantalla a fin de entorpecer al jugador, desviando la trayectoria de la bola y que no fuera tan monótono. Utilicé el carácter 224, que por defecto es una carita sonriente. Esto se quedó así hasta prácticamente el final del proyecto, ya que no tenía ni idea de qué historia dar a esos objetos. Al final se ha quedado en que son cápsulas de salvamento perdidas (no voy a entrar en si están ocupadas o no) y el gráfico ha surgido por aprovechar uno de los bytes de la carita, ya que utilizo el comando *PEEK* para detectar colisión entre la bola y el objeto.



Velocidad y jugabilidad

Tests de velocidad de impresión de caracteres

Hago este sencillo programa para testear el número de caracteres por segundo que se pueden imprimir con el hack de texto y con el modo de transparencias (a efectos de la prueba, en realidad no importa en qué modo este la pantalla):

```
10 DEFINT a-z:Print#2,CHR$(22)CHR$(1)
20 A$=CHR$(231)+CHR$(8)
30 B$=CHR$(15)+CHR$(1)+CHR$(231)+CHR$(8)+CHR$(15)+CHR$(2)+
     CHR$(232)+CHR$(8)+CHR$(15)+CHR$(3)+CHR$(233)
40 n=0:t!=TIME+3000:WHILE t!>TIME:PRINT#1,A$;:n=n+1:WEND
50 Locate 1,10:PRINT"Chars en 10 seg. con hack:";n
60 n=0:t!=TIME+3000:WHILE t!>TIME:PRINT#2,B$;:n=n+1:WEND
70 Locate 1,11:PRINT"Chars en 10 seg. con transparencia:";n
```

Resultado:

```
Chars en 10 seg. con hack: 1038
Chars en 10 seg. Con transparencia: 527
```

Lo que da un resultado de 17.28 cps en el primer caso y 8.78 cps en el segundo caso.

El movimiento se demuestra andando

En los primeros estadios del desarrollo, tengo únicamente una bola que rebota, una paleta que se mueve de lado a lado y algunos ***obstáculos***, embrión de lo que serán luego los ***ladrillos*** del juego. El patrón de rebote de la bola en este momento es el siguiente:

- Cuando la bola choca lateral y/o verticalmente con algo cambia la dirección *X* y/o *Y*, respectivamente.
- Cuando la bola choca diagonalmente cambian ambas direcciones, *X* e *Y*.

Enseguida se hace notar un problema de movimiento: cuando se mueve la paleta, la bola parece moverse más lentamente y así es. Para salvar esto, el movimiento de la bola deberá gestionarse mediante interrupciones, para que sea todo lo fluido y constante que sea posible.

Cuando agrego el marcador de puntos, y el contador de ladrillos que faltan por destruir, el hecho de tener que borrar el ladrillo, poniendo en su lugar los 2 caracteres que corresponden al fondo que hay detrás y luego actualizar los marcadores ponen de manifiesto un nuevo problema: actualizar todo eso causa que se ralentice el movimiento de la bola en exceso.

Nuevamente hago uso de interrupciones para actualizar los marcadores de forma asíncrona, esto es, cuando se destruye un ladrillo se ejecuta una instrucción *AFTER para* que al poco tiempo se actualice el número de ladrillos y desde este, otro más que actualice el marcador de puntos. Si varios ladrillos son destruidos muy seguidos, no dará tiempo a que se ejecute la actualización de los marcadores, con lo que esta tarea se pospone hasta un momento más propicio (cuando el juego estará más desahogado).

Bucle principal

El bucle principal mientras se ejecuta la acción del juego es lo más sencillo posible, persiguiendo una respuesta de teclado lo más rápida que al alcance esté:

```
100 IF NOT INKEY(kr)THEN 200
110 IF NOT INKEY(kl)THEN 210
```

```
120 IF 1>CBR THEN 260
130 IF pw AND 2 THEN 170
150 GOSUB 300:IF INKEY(kp)THEN 100
160 GOSUB 900:GOTO 100
```

La línea 150 hace un *GOSUB* a la rutina de movimiento de los *enemigos,* por así decir, que aparecerán por la pantalla para entorpecernos y salta a la línea 100 si el jugador no ha pulsado la tecla de ***PAUSA*** (rutina que está localizada en la línea 900).

Hay que comentar que la variable *CBR* (*counter brick)* no la utilizo para decidir cuándo se termina la fase, sino también para hacer saltar el juego cuando hay que restar una vida u otras circunstancias, ya que por el hecho de utilizar interrupciones he notado que realizar determinados cambios de situación de juego desde dentro de una rutina llamada por una interrupción tiene efectos indeseados, por ejemplo: movimientos de la bola más rápidos por tiempo determinado, actualizaciones de objetos de pantalla fuera de lugar, etc.

La variable *PW* la utilizo activando y desactivando sus bits para establecer el estado del jugador en cuanto a los ítems que se hayan recolectado, pero también la utilizo en ciertos momentos en que la bola, por su trayectoria, se coloca sobre la paleta del jugador y esta necesita ser repintada, cuando no se estuviera moviendo.

Se obtiene también un *rendimiento extra* cuando, después de desplazar la paleta, se continuará ejecutando la misma línea de programa mientras el jugador mantiene la tecla de dirección pulsada (y no se haya alcanzado el extremo de la zona de juego):

```
200 IF pm>px THEN LOCATE#7,px,25:px=px+1:PRINT#7,pr$;:GOTO 100
ELSE 120
210 IF 2<px THEN px=px-1:LOCATE#7,px,25:PRINT#7,pl$;:GOTO 110
ELSE 120
```

Patrones de rebote

El esquema de comprobaciones que realizaba la rutina de movimiento de la bola (a partir de la línea 3000) en cada iteración inicialmente era el siguiente (suponemos que se desplaza hacia arriba y a la derecha):

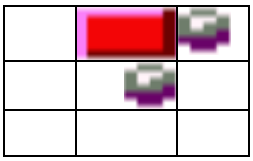
	T1	T2
		T3

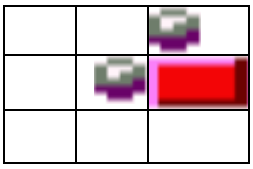
Se comprueba colisión (los ladrillos, como el fondo, también están almacenados en un array de cadenas de texto) en tres posiciones posibles: arriba, diagonal y lateral. Si hay ladrillo en cada una de esas posiciones, se procesa la colisión y ocurre lo que tenga que ocurrir en consecuencia (líneas 4000, 4100 y 4200).

Estas comprobaciones inicialmente no eran demasiado problema, pero cuando agregué más contenido al juego y objetos que circulan por la pantalla para entorpecer al jugador, dado que la verificación de colisión con uno de estos objetos la realizo convirtiendo las coordenadas *X* e Y de la bola a dirección de memoria de video y examinando el contenido, se hacía demasiado pesada, por así decir, la rutina de movimiento de la bola y la reacción a las pulsaciones de tecla del jugador ya no resultaban todo lo fluidas que podía desear.

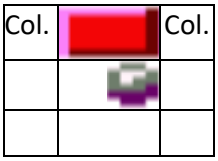
Para reducir el número de comprobaciones que se han de realizar en cada iteración (pasar de 3 a 1), planteo la posibilidad de comprobar únicamente la posición diagonal, en primera instancia, si existe obstáculo se comprobarán también las posiciones vertical y lateral:

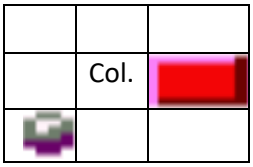
Primero se detecta el ladrillo en la posición en la que se encuentra el verde, caso de existir, se comprueban también las posiciones vertical y lateral. Pero, si de da un caso como estos, no rebotaría (recordemos, en el ejemplo, la bola se mueve en sentido arriba/derecha):





La forma que se me ocurrió para paliar este problema es incluir en el array de los ladrillos un valor que indique colisión en los lados de los ladrillos en los que no haya nada, pero en pantalla no se muestra, es decir: C C





Hay que notar que, en el ejemplo de la derecha, se detectará la colisión justo una posición antes de llegar al ladrillo que la causa y quedará guardada para procesarse en la siguiente iteración.

Otro pequeño cambio que introduje es que la bola al colisionar diagonalmente no invertirá su movimiento en ambos ejes, sino que se invertirá siempre el vertical y aleatoriamente el horizontal, obligando más a menudo al jugador a moverse, ya que de lo contrario me parecía que se hacía algo aburrido porque había casos en los que podías mantener la paleta en determinada posición inmóvil enviando la bola siempre en la misma dirección hasta destruir el ladrillo pretendido.

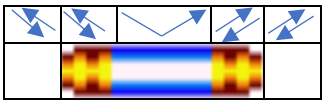
Movimiento y acción de la paleta

La paleta, como suele ser habitual en este tipo de juegos, se mueve sólo lateralmente y sólo obedece una de las teclas direccionales simultáneamente.

Cuando la bola se encuentra en la coordenada *Y* 24 o superior, se comprueba su posición *X* respecto a la de la paleta y se decidirá su rebote en función de:

- Si la bola se aproxima por la izquierda y golpea el extremo izquierdo, rebotará en dirección contraria (lo mismo se aplica a dirección y lado contrario).

- Si la bola se aproxima por la izquierda y golpea la parte central, saldrá rebotada en la misma dirección.



Como digo, el rebote de la bola también tiene lugar cuando esta se encuentra en la coordenada vertical 25. La ventaja que tiene esto es, que, dado que la paleta se desplaza más rápido que la bola, es posible *rescatar* la bola de la caída *in extremis*, es decir, llegar con la paleta y colocarla sobre la bola justo antes de que esta llegue a caer para salvar la pérdida de una bola.

Movimiento cápsulas de salvamento

Las cápsulas que aparecen y se desplazan por la pantalla son siempre cuatro y tienen dos patrones de movimiento:

- Dos de ellas se desplazarán según una dirección escogida aleatoriamente, nunca en diagonales. Cuando colisionan con alguna pared, se replantea una nueva dirección de movimiento. Cuando alcanzan una determinada "altura" por encima de la paleta, permanecerán allí y se desplazarán lateralmente o bien se alejarán.
- Las otras dos se desplazarán siempre en diagonal, buscando la interceptación de la bola y no tienen límites de altura.

Cuando las cápsulas y la bola colisionan entre sí, la cápsula de destruye y aparecerá una nueva por la parte superior de la pantalla.

Recolección de cápsulas de efectos

Cuando un ladrillo es eliminado, entre un 25 y 35% de las veces puede (y digo puede porque luego se comprueba el número de ladrillos que faltan por eliminar o el tipo de cápsula) aparecer una cápsula de diferentes colores que altera el comportamiento de la bola o de la paleta. Estas capsulas aparecen y permanecen estáticas, hasta que transcurridos unos segundos desaparecen. Hacer que cayeran hasta abajo para recogerlas con la paleta suponía demasiada carga de trabajo, por lo que descarté hacerlo así. En lugar de esto, es la propia bola la que las recolecta al colisionar con ellas.

Los efectos son aquellos que no supongan excesivas cargas de trabajo adicionales, por ejemplo: no hay efecto *multiball*. La lista es la siguiente:

	Aumentar la velocidad de desplazamiento de la bola.
	Reducir el tamaño de la paleta (hasta cierto punto).
	La bola se puede "reconducir".
	Invertir los controles de movimiento de la paleta.
	Bonificar puntuación con 1000 pts. y pasar nivel (se muestra el guiño propuesto).
	Aumentar el tamaño de la paleta (hasta cierto punto).
	Hace que la paleta tenga cierta adherencia (*magnetismo*) cuando esta se mueve.
	Bola super destructiva (un único impacto basta para eliminar los ladrillos).
	Reducir la velocidad de desplazamiento de la bola.

Sonido y música

Efectos de sonido

La premisa es que cualquier cosa de las que ocurran en la pantalla debería producir algún efecto de sonido, de lo contrario quedaría la cosa bastante sosa. Prácticamente todo produce algún efecto, excepto el movimiento de la paleta y de los objetos que pululan por la pantalla, ya que tampoco me iba a pasar sobrecargándolo.

Es cierto que en cierta manera los efectos de golpeo de la bola están inspirados en cómo suenan en el juego *Arkanoid II*, pero es que este tipo de sonido me resultaba de alguna manera más natural que cualquier otro. Estas son las envolventes configuradas:

```
ENV 1,15,-1,5:ENV 2,3,-5,1
ENT-1,1,-100,3,1,50,2,1,50,2
ENT-2,1,-100,3,2,25,2,2,25,2
```

Efecto de rebote en límites de la zona de juego (*fadeout* de volumen corto, envolvente de tono tipo metálico):

```
SOUND 1,200,200,15,1,1
```

Efecto de rebote en ladrillos (diferentes tonos en función de la dirección de movimiento vertical de la bola):

```
SOUND 1,200+50*sy,80,15,1,1
```

Efecto de rebote en paleta (mismo efecto, con tono más grave):

```
SOUND 1,300,200,15,1,1
```

Efecto de ladrillo eliminado (el más agudo de todos, variación en la envolvente de tono, con cierto arpegio):

```
SOUND 1,100,80,15,1,2
```

Efecto estereofónico “posicional”

He dotado al programa de cierta estereofonía para los efectos de sonido, de acuerdo con la posición de la bola, de dos tipos:

- Número de canal (derecho, izquierdo o ambos)
- Nivel de volumen (mayor volumen en el canal izquierdo según la acción se encuentra más cerca del extremo izquierdo de la zona de juego y lo mismo para el caso contrario), al mismo tiempo que emito el mismo sonido por el otro canal con algo menos de volumen para dar algo de espacialidad.

Para la elección del canal de sonido utilizo una función definida de usuario:

```
DEF FNst(n)=128+ABS(20>n)+4*ABS(10<n)
```

Para la emisión del sonido, que lo hará por uno u otro canal o ambos, según la función anterior:

```
SOUND FNst(x),200,80,15,1,3
```

Para la emisión de sonido basándome en el volumen, por ejemplo:

```
v=x\6:REM X es la coordenada de la bola
SOUND 129,200,5*(15-v),15-v,1,1:SOUND 132,200,5*(10+v),10+v,1,1
```

Música

Este nuevo proyecto, como en el anterior, está programado 100% en Basic, por lo que no quería utilizar ninguna clase de rutinas en CM para gráficos ni tampoco para la música (aparte de las propias herramientas que te da el firmware del CPC. Además, las bases del concurso recientemente publicadas lo prohíben). Desde el primer momento tenía claro que para la incorporación de melodía me iba a programar una herramienta de composición musical, aunque ni soy músico ni tengo demasiada idea de solfeo aparte de qué es un pentagrama.

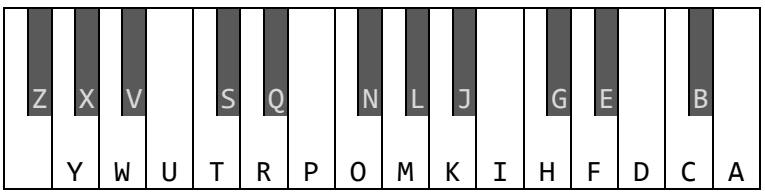
Aunque diga que no soy músico sí que tengo algo de oído y toco algo a una mano (o tocaba, ahora me falta mucha práctica). Me viene de la misma época en la que tenía el *Amstrad* y un excuñado al que le gustaban los teclados musicales y me dejaba cacharrear con ellos. Algunos de estos sintetizadores disponían de caja de ritmos, recursos más que divertidos de manejar dadas mis limitaciones. Más tarde me compré un pequeño sintetizador monofónico *Yamaha CS01II* (no MIDI) y un teclado MIDI para el PC, con el que también he tocado algo de secuenciadores y *MOD trackers*.

El programa ultra simple que he creado para ayudarme en la composición (y al que he llamado *Music Box*), está orientado al funcionamiento de los MOD trackers, sólo que, a un nivel muy básico, sin instrumentos ni efectos.

De nuevo trabajo con *arrays* de cadenas de texto en las que almaceno valores numéricos (1 a 5) para lanzar sonidos de batería (la caja de ritmos) o caracteres alfabéticos para los tonos de las notas, en base a la siguiente fórmula (extraída del manual del CPC):

```
ma=ASC(char$)-65
SOUND 1,UNT(62500/(440*(2^((10-ma)/12)))),40,15
```

La distribución carácter alfabético - notas sería la siguiente, trasladada a un teclado de piano:



En *Music Box* los patrones están organizados en horizontal y en la pantalla (que parece un poco liosa) se muestra en la parte superior el indicador del punto de tick o reproducción (64 caracteres de longitud), a la derecha el *tempo* (valor que se le pasa al EVERY que invocará la interrupción para hacer sonar las notas) y la posición vertical del cursor (que marca el patrón actual, aunque en la captura no coincide dicho valor ya que está en modo reproducción y como se puede ver, el cursor se encuentra más abajo, posición que corresponde con el patrón que está sonando en ese momento).

La primera línea de patrón (en la que sólo se ven valores numéricos de 0 a 5) corresponde a la caja de ritmos, que sonará por el canal central y en adelante, cuando contienen caracteres alfabéticos, las líneas impares corresponden al canal izquierdo y las pares al derecho. Las tres

líneas inferiores corresponden a valores numéricos que harán sonar batería en ambos canales, derecho e izquierdo alternativamente y se reproducen individualmente.

El programa al igual que el editor UDG permite exportar el trabajo a fichero .BAS con formato ASCII, conteniendo las líneas DATA:

```
20000 DATA BCECEGGCECEGGMNONIK
20010 DATA
10003030200030301030103020304430100030302000303010301030203424 30
20020 DATA
Q000L0J0E0G0H0J0L0L0J0N0B0000000Q000L0J0E0G0H0J0L0J0G0N0Q0NQ0000
20030 DATA
X0X0Q0Q0X0X0Q0Q0X0X0Q0Q0X0X0Q0Q0X0X0Q0Q0X0X0Q0Q0X0X0Q0Q0X0X0Q0Q0
20040 DATA
S000N0L0G0I0J0L0N0N0L0P0D0000000S000N0L0G0I0J0L0N0N0L0P0D0PD0000
20050 DATA
Z0Z0S0S0Z0Z0S0S0Z0Z0S0S0Z0Z0S0S0Z0Z0S0S0Z0Z0S0S0Z0Z0S0S0Z0Z0S0S0
20060 DATA
L0L0E000L0L0X000L0L0E000L0L0X000N0N0G000N0N0B000N0N0G0G0N0N0B000
20070 DATA
0L0L0E000L0L0X000L0L0E000L0L0X000N0N0G000N0N0B000N0N0G0G0N0N0B00
20080 DATA
S000Z0Z0S0Z0Z0Z0S000Z0Z0S000Z0Z0Q000X0X0Q0X0X0X0Q000X0X0Q000X0X0
20090 DATA
SSSSZZZZSSZZZZZZSSSSZZZZSSSSZZZZQQQQXXXXQQXXXXXXQQQQXXXXQQQQXXXX
20100 DATA
0000000000000000000000000000000000E000G0H0J0L0J000G000H0J0L0J0Q000
20110 DATA
Q0Q0J000L0L0J000Q0Q0J000L0L0J000Q0Q0J000L0L0J000Q0Q0J000L0L0J000
20120 DATA
333000333000333554440003330003330003335552221122112212211111111
20130 DATA
434340040030400030004030040300004343400400304000300040300403000 0
20140 DATA
111111111111111222222220000000011111111111111112342342300000000
```

La primera línea *DATA* señala, mediante caracteres alfabéticos el número de patrón que se habrá de reproducir en cada momento, una vez leía al completo el último carácter de esa línea pasará a la primera posición. La ventaja que tiene utilizar caracteres alfabéticos es que se me ha hecho más fácil leer e introducir arreglos a la música así que si hubieran sido códigos numéricos.

El reproductor es tal que esto:

```
30000 IF tk=1 THEN my=ASC(MID$(MUS$(0),mx,1))-65:mx=mx+1:IF
mx>LEN(MUS$(0))THEN mx=2
30010 ma=ASC(MID$(MUS$(my),tk,1))-65:IF ma>=0 AND ma<58 THEN
GOSUB 30300 ELSE IF ma<>-17 THEN ON ma+17 GOSUB
30200,30210,30220,30230,30240:GOTO 30050
30020 ON VAL(MID$(MUS$(1),tk,1))GOSUB
30100,30110,30120,30130,30140
30030 IF 1=my OR 14=my THEN 30050
30040 mc=ASC(MID$(MUS$(my+1),tk,1))-65:IF 0<=mc AND 58>=mc THEN
GOSUB 30400
30050 tk=1+tk MOD 64:RETURN
30100 SOUND 130,500,20,14,7,6:RETURN
30110 SOUND 130,500,20,14,7,,1:RETURN
30120 SOUND 130,5,5,14,6,0,6:RETURN
30130 SOUND 130,10,60,14,8,,3:RETURN
30140 SOUND 130,100,200,14,3,7:RETURN
30200 SOUND 129+3*(tk MOD 2),500,20,14,7,6:RETURN
30210 SOUND 129+3*(tk MOD 2),500,20,14,7,,1:RETURN
30220 SOUND 129+3*(tk MOD 2),5,5,14,6,0,6:RETURN
30230 SOUND 129+3*(tk MOD 2),10,60,14,8,,3:RETURN
30240 SOUND 129+3*(tk MOD 2),0,10,0,5,,1:RETURN
30300 SOUND 129,UNT(62500/(440*(2^((10-ma)/12)))),40,15,8:RETURN
30400 SOUND 132,UNT(62500/(440*(2^((10-
mc)/12)))),40,13,8,8:RETURN
```

Y para poner en marcha y hacer sonar la música:

```
30500 DIM MUS$(140):RESTORE 20000:FOR n=0 TO 14:READ
MUS$(n):NEXT:tk=1:mx=1:EVERY 5 GOSUB 30000:RETURN
```

Como opciones en la reproducción tengo disponible (claro está) la posibilidad de reproducir más lenta o más rápidamente la melodía (aunque no demasiado rápido, ya que enseguida me quedo sin tiempo). Por ejemplo, cuando se termina la partida reproduzco cierta parte de la melodía más lentamente para que parezca más triste.

También puedo añadir notas que en realidad no están en las *DATA* a modo de acompañamiento a la melodía, unas octavas más graves, etc.

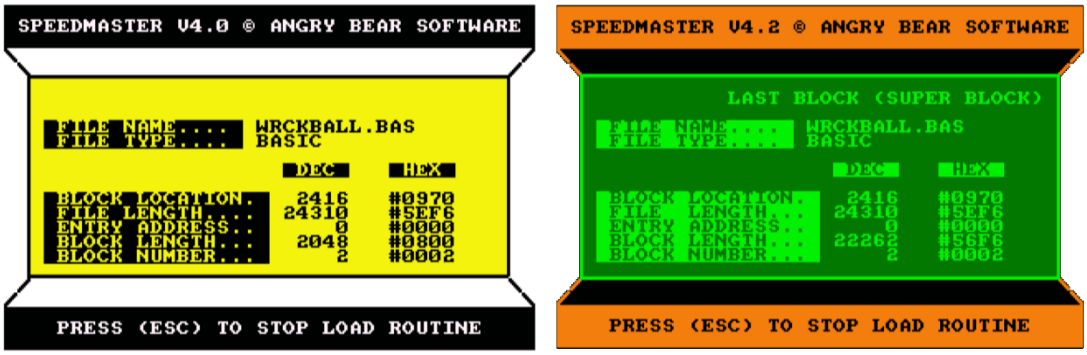
En total he creado 4 melodías distintas y 1 variación, para amenizar las siguientes partes del juego:

- Pantalla de menú y de mejores puntuaciones.
- Música al inicio de cada nivel.
- Música para el guiño.
- Música para el mensaje de fin de juego (extracto de la melodía del menú a menor velocidad).
- Música en la pantalla de introducción de iniciales al obtener una de las 10 mejores puntuaciones.

Sistema de carga desde cinta

Carga rápida de *bloques de tamaño extra*

Cuando tenía mi *Amstrad CPC 464* f.v., tenía cantidad de juegos originales (la mayoría), pero como todo hijo de vecino también tenía mis cintas copiadas, algunas copiadas con doble pletina, porque contenían juegos con carga turbo y no quedaba más remedio, pero también tenía juegos con carga normal, en bloques. Para copiar ese tipo de programas se podían utilizar copiones, en los que introducías la cinta original, se leían todos los bloques que podrían caber en memoria y luego introducías la cinta virgen y se volcaban allí los bloques leídos. El copión que yo utilizaba era ***SpeedMaster*** de *Angry Bear Software*.



Este copión estaba programado en Basic y la rutina de carga/grabación era CM. No sé exactamente cómo ocurrió, pero trasteando con él me di cuenta de que, mientras el primer bloque de cada archivo (que estuviera formado por varios bloques) debía ser de 2048 bytes, **a partir del segundo podía ser de mayor tamaño**. Esto ofrecía la ventaja de que para cargar cualquier programa me ahorraba el tiempo de cargar la cabecera de cada bloque y el espacio en blanco en la cinta que separa cada uno, tardando así menos tiempo en cargar.

Modifiqué *SpeedMaster* para que, en la fase de carga, a partir de los bloques terceros, se desplazaran los datos en memoria, justo al final del bloque segundo y modificando la cabecera de este para que correspondiera con la nueva longitud de bloque, al grabar este bloque en cinta se obtenía un bloque tan grande como hubiera podido caber en la fase de carga.

Todo esto lo he aplicado a la carga de *Wrecking Ball*: Es un programa en Basic de carga por bloques, pero salvo por los primeros, se carga como si fuera carga *turbo*, pero aparte de por su longitud, se puede considerar que son bloques normales, el firmware los puede cargar perfectamente por sí solo.

Tengo que comentar también que la velocidad a la que está grabado es 2000 b/s. (he realizado pruebas con los emuladores *Retro Virtual Machine* y con *WinAPE* y el segundo **no es capaz de interpretar correctamente** CDTs con bloques grabados de esta forma, por lo que he tenido que insertar un bloque de 1 byte entre la pantalla de presentación y el juego para que cargue correctamente.

Guiño

La referencia del presente año en la ***CPCRetroDev*** (2021) señala que debe aparecer un arcoíris durante el *gameplay*, como guiño al juego ***Rainbow Islands*** de *Ocean Software*.

Se ha incluido un efecto de arcoíris al recolectar una cápsula dorada ( , o **pulsando R+PAUSA**), que otorga 1000 puntos y una bola extra. Tras mostrar el arcoíris además suena (a mi manera) la parte final de la melodía principal del juego *Rainbow Islands*, que tiene lugar al terminar cada fase de dicho juego.

Descripción de las partes del programa

1-50	Licencia, inicialización y arranque del programa.
60-70	Inicio de partida
100-150	Bucle principal de la partida.
160	Salto a pausa de la partida.
170-250	Movimiento e impresión en pantalla de la paleta del jugador.
260-290	Saltos a fin de pantalla, pérdida de una bola o fin de partida.
300-470	Movimiento cápsulas que se mueven por la pantalla.
500-580	Borrado, colisión, impresión e inicialización cápsulas que se mueven.
600-640	Fin escenario actual, salto a generación de nuevo escenario y a bucle principal.
900-990	Entrada y salida del bucle de pausa. Iniciar interrupciones para mover bola, etc.
1000-1390	Generar escenario.
1400-1800	Inicializar variables para ladrillos y fondo. Imprimir fondo, marcadores, etc.
3000-3810	Movimiento bola, detección de colisión, rebote en paleta de jugador, etc.
4000-4260	Procesar colisión con ladrillos o cápsulas.
4300-4650	Puntuar al destruir ladrillo y decidir si aparece cápsula de poder.
4700-4930	Procesar colisión con cápsula de poder y aplicar efecto del poder adquirido.
5100-5950	Texto carácter por carácter en pantalla y otros efectos (arcoíris, etc.)
6000-7390	Definir mapa de caracteres.
7400-7930	Definir funciones, arrays de variables, envolventes y direcciones de memoria.
8000-8980	Pantalla de introducción y menú de opciones.
9000-9990	Pantalla de mejores puntuaciones y de introducción de iniciales.
10100-10240	Redefinir teclas de juego.
20000-20210	Datos de música
30000-30540	Reproductor de melodías e inicialización de melodía a reproducir.
64000-64040	Salida del programa.

Recursos y herramientas utilizadas

Microsoft Word 2019 (Microsoft 365) © Microsoft Corporation

Paint.net. Copyright © 2014 dotPDN LLC, Rick Brewster, and contributors.

ConvImgCPC. Demoniak. (http://ldeplanque.free.fr/ConvImgCpc/new/)

RetroVirtualMachine v2.0 © 2018-19 Juan Carlos González Amestoy

WinAPE 2.0 Beta 2 © Richard Wilson

SpeedMaster v4. Angry Bear Software

European Space Agency (ESA), artículo sobre basura espacial en la página de la ESA:
https://www.esa.int/Safety_Security/Space_Debris/About_space_debris
https://www.esa.int/ESA_Multimedia/Copyright_Notice_Images

Licencia GNU GPL (https://www.gnu.org/licenses/)

Despedida

Agradezco a todo lector el interés en este proyecto y en el presente documento y espero que mis trabajos sean disfrutados siquiera una pequeña fracción de lo que yo he disfrutado creándolos.

Para mí, el microordenador *Amstrad CPC* siempre ha sido y me ha demostrado que es la mejor herramienta para el aprendizaje en la programación a todos los niveles, tanto por su excelente manual de usuario y referencia como por, a mi modo de ver, sus insuperables características de explotación en su programación.

Saludos.