

CÓMO SE HIZO

Programa: Rompetechos
Lenguaje: 100% ZX Spectrum Basic
Autor: IvanBasic
Fecha: Diciembre 2016

Estructura del programa

Se explican los diferentes bloques siguiendo el orden de número de línea del programa, no en el orden en que aparecen las secuencias al jugar. Se indica el intervalo de líneas y el nombre del bloque.

0001 RUN

Se incluye un RUN 9000, que es donde se inicializa el programa, para que en caso de BREAK o 'bug', se pueda empezar de nuevo simplemente tecleando RUN.

0002 A 0038 MOTOR DE MOVIMIENTO DE ROMPETECHOS

Hay 4 tipos de 'movimiento'

- Quieto (0002 a 0005)
- Pie tapeando (0006 a 0019)
- Movimiento a derecha (0020 a 0028)
- Movimiento a izquierda (0030 a 0038)

El giro de la cabeza es aleatorio, y se hace en los 4 estados de movimiento.

En todos los movimientos se hace una llamada a la subrutina de los elementos particulares de cada pantalla (**GO SUB e**), que pueden ser enemigos (agua, semáforos, personajes enfadados, Rayos X), u otras condiciones (límites de pantallas). Al retorno de cada subrutina se detecta si ha habido bofetada o chapuzón (**IF f THEN GO TO u**).

Para ganar velocidad, la transición entre los distintos tipos de movimiento se hace en función de si hay o no una tecla pulsada, y cuál es ésta (**PEEK 23560**).

El movimiento a derecha (o izquierda) tiene dos sprites, que se animan repitiendo las mismas sentencias, pero solo cambiando el gráfico de las piernas. Esto consume mucha memoria, pero permite ganar velocidad. Lo habitual en juegos BASIC es combinar ambas direcciones y animación en una sola línea, usando variables y decisiones lógicas, esto ahorra líneas y memoria, pero resta mucha velocidad.

0040 A 0066 SELECTOR DE ACCIONES CUANDO SE PULSA 'ESPACIO'

Se hace un **IF f\$(x)="código"** para todos los posibles valores de la situación que hay en la posición **x** de Rompetechos cuando se pulsa Espacio. Cada situación tiene un código ("**s**" es cambio de pantalla hacia arriba, "**b**" hacia abajo, "**1**" son escaleras que no se pueden subir, "**f**" es fruta,...), siendo **f\$** la variable que almacena todos los elementos de una pantalla.

Esta parte del programa es optimizable, pues si se usan códigos con símbolos correlativos (letras minúsculas de la "**a**" a la "**z**"), se puede dirigir fácilmente a cada acción calculando la línea en función del código del símbolo (por ejemplo **GO TO 137-CODE "a"** nos llevaría a la línea 40 para un elemento definido por el símbolo "**a**"). No se hizo así porque se fueron añadiendo acciones y situaciones al ir 'improvisando' el juego al ver su potencial. Quizás para una versión siguiente se pueda ahorrar memoria con esta optimización y así añadir más situaciones, complejidad, o personajes...

En este bloque de acciones, tienen importancia particular las líneas siguientes:

- Diferentes resultados cuando encontramos un buzón (0060 a 0062)
- Respuesta del policía cuando se le habla (0063 a 0066)

0076 A 0083 GLOBOS DE DIÁLOGOS

Se imprimen los globos y el texto dentro de ellos, controlado por la variable **g\$**.

La línea 0076 escribe el globo de diálogo de Rompetechos.

La línea 0080 escribe el globo del otro personaje con el que está hablando.

0085 A 0089 BOFETADA/CHAPUZÓN

La variable **u**, que se define en cada tipo de 'enemigo', es el número de línea según el golpe sea una bofetada (**u=85**) o un chapuzón (**u=86**), eligiendo en cada caso el color del globo y el texto dentro.

Si el bofetómetro llega a 8, se pasa al bloque de la línea 90, final del juego.

0090 A 0099 FIN DE LA PARTIDA

El final es el mismo en todas las situaciones, solo cambia el texto según cómo haya finalizado:

- Fin de la aventura por acumulación de bofetadas
- Misión completada al entregar la carta en el buzón, con sello, y antes de la hora límite
- Carta entregada en buzón, pero sin sello
- Carta entregada con sello, pero tarde. Deja abierta una posibilidad futura de una segunda parte...
- Carta entregada sin sello, y además tarde

0100 A 0182 ENEMIGOS Y OTRAS SUBRUTINAS

En estas líneas se controlan:

- Semáforos
- Agua (río, mar, piscinas, cloacas)
- Rayos X y pasillo prohibido, en el Hospital.
- Andén del metro
- Borrado de la pizarra en la clase
- Interacción con personajes (frutería, tienda de abalorios)
- Límites de pantallas que no se pueden pasar (paredes)

Las subrutinas se llaman con **GO SUB e**, siendo "e" la línea correspondiente a cada estado del 'enemigo'. Se ha programado como subrutina individual cada estado 'unitario' del enemigo, ya que con esto se gana velocidad al no tener que hacer animaciones dentro de una misma subrutina, que exigirían el control con **IF** y operadores lógicos de variables numéricas y alfanuméricas.

Así, por ejemplo, las olas del río, formadas por dos posiciones de las olas, no tienen una única subrutina con las dos secuencias de la animación, sino que cada secuencia es en sí misma un único enemigo. La línea 0101 a 0103 es una secuencia del río, y la línea 0104 a 0106 es el mismo río pero con cambio de ola. Ambas subrutinas hacen exactamente lo mismo, pero son mucho más rápidas que si se programa una única subrutina que controle el tipo de ola con una variable, una cadena de caracteres, y un **IF** más operadores lógicos. La desventaja es que consumen más memoria.

0200 A 0208 ESCRITURA EN LA PIZARRA DE LA CLASE

Rompetechos puede escribir letras y números en la pizarra. Puede finalizar la escritura con ENTER en cualquier momento. Lo escrito permanece en la pizarra hasta que se borre, lo cual se consigue simplemente caminando a lo largo de la misma.

1010 A 1047 DIBUJADO DE PANTALLA

En esta rutina se dibujan las pantallas del mapa. Cada pantalla está formada por bloques estándar que se combinan. Cada bloque tiene su código de identificación y un número que sirve para definir la posición o el color del bloque. Hay una relación entre el código del carácter que define cada bloque, y la línea del programa en la que se dibuja ese bloque (esta es la optimización explicada anteriormente).

La pantalla se dibuja en tres partes:

- La zona superior, con cielo y fondo de edificios (líneas 1020 y 1025)
- La zona inferior, que tiene la base o suelo de calles, avenida con semáforos, muelle, o río. Se codifica en cada pantalla.
- La zona media, por la que se mueve Rompetechos, que se dibuja con los diferentes bloques codificados (casas, buzón, semáforos, setos, vallas, farolas, estanterías, etc.)

La línea 1044 dibuja la carta si entramos en el lugar donde estaba perdida.

La línea 1045 decide, aleatoriamente, si aparece el guardia en esa pantalla, excluyendo únicamente los apartamentos.

1051 A 1304 DATOS DE CODIFICACIÓN DE PANTALLAS

Estas líneas son los **DATA** de los códigos de bloques de cada pantalla. La variable “**p**” controla el número de pantalla, que coincide con el número de línea que tiene su codificación. Así, con **RESTORE p: READ d\$** leemos rápidamente el código de la pantalla número **p** en la que está Rompetechos.

Si se consigue optimizar el programa y ahorrar más memoria, se pueden añadir más pantallas y así ampliar el mapa actual.

1651 A 2169 SUBRUTINAS DE BLOQUES DE PANTALLAS

Cada grupo de líneas limitadas por **RETURN** dibujan un bloque o un escenario.

9000 A 9205 INICIALIZACIÓN Y PRESENTACIÓN

Se inicializan todas las variables que se van a usar, se fijan los parámetros de atributos, y se imprime la presentación.

9900 CARGAR BYTES DE LOS GRÁFICOS

El programa debe salvarse como **LINE 9900**. En esta línea se cargan los 1024KB de los gráficos, que corresponden a un juego completo de caracteres, más 21 UDG.

9999 RESTAURAR EL CONTROL NORMAL

Si interrumpimos el juego y queremos manejar el listado, tecleando **GO TO 9999** se restaura el modo normal. Es difícil teclear esa instrucción pues el juego de caracteres está cambiado, y además se ha pukeado la velocidad de respuesta del teclado, con lo que hace falta cierta habilidad para poder hacer esta instrucción.

Técnicas de programación

Se explica seguidamente qué técnicas se han elegido para la programación. El criterio, ordenado por prioridad, ha sido:

- 1/ Respuesta inmediata al teclado. **INKEY\$** es muy limitado; **IN puerto teclado** no almacena ni memoriza las pulsaciones; así que la mejor solución es **POKE/PEEK 23560**. Se eligen como teclas de movimiento dos letras con código consecutivo (O-P, pero podrían ser J-K, K-L, G-H), que permiten acelerar ciertas funciones.
- 2/ Máxima velocidad posible para el manejo de Rompetechos, teniendo en cuenta que está formado por 3x2 caracteres, y que tiene que 'redibujar' el fondo por el que va pasando. De ahí la decisión de movimiento solo Der-IZq.
- 3/ Sencillez en 'enemigos'. Esto hace que no se hayan programado personajes móviles, para tener la mejor velocidad posible.

En MI opinión particular, los juegos BASIC aunque sean lentos, son jugables si la respuesta al teclado es inmediata. Después se debe buscar la máxima velocidad posible del protagonista y enemigos/obstáculos.

MAPEADO

La variable "**p**" controla el número de pantalla. El mapa primario (calles de la ciudad) está formado por 48 pantallas (8 filas x 6 columnas). El número de pantalla coincide con el número de línea del programa que tiene su codificación para dibujarla mediante bloques.

Al bajar una calle, el número de pantalla aumenta en 6, y disminuye en 6 al subir. Avanzar a la derecha suma 1, y a la izquierda lo resta. Al haber 'enemigos' en las pantallas extremos (ríos al Oeste y mar al Este), no es posible avanzar en esas direcciones.

El mapa secundario o "paralelo" está formado por el interior de los locales. Cuando se 'entra' en un sitio, el número de pantalla aumenta en 100, y al salir disminuye en esta cantidad. Dentro de un sitio, si nos movemos arriba/abajo/izquierda/derecha en la pantalla, sigue el mismo criterio para el cálculo del nuevo número de pantalla que el mapa primario.

VARIABLES

Para acelerar el manejo de variables, todas las que se van a utilizar se definen al inicio, para que durante el juego no se gaste tiempo moviendo toda la pila de variables cada vez que se crea una nueva. Las líneas 9005 a 9007 definen todas las variables que se van a utilizar.

Todas las variables numéricas son de una sola letra, para ahorrar un poquito de memoria, y para que su manejo sea rápido. No se usan matrices (arrays) por la misma razón.

CONTROL DE TECLADO

PEEK 23560 almacena el **CODE** de la última tecla pulsada. Solo detecta pulsaciones de teclas individuales. De este modo, sea cual sea el momento en que se pulsa una tecla, ésta será detectada.

Se lee **PEEK 23560** en cada etapa del movimiento de Rompetechos, y una vez movido, se hace **POKE 23560,0** para que el movimiento no se mantenga cuando se suelta la tecla. Para que el movimiento sea rápido, se debe poner al mínimo la variable del sistema que controla el tiempo que se debe pulsar una tecla para que haya repetición (**POKE 23561,0**). Esto dificulta la edición del programa si se pulsa **BREAK**, por eso se restaura en la línea 9999.

Además, haciendo **POKE 23560, CODE "tecla"** se puede manejar Rompetechos sin que el jugador pulse teclas. Esto es lo que sucede cuando tras taping el pie, se desplaza un paso o realiza una acción si hay algo cerca. Esto es útil si en un juego se quiere programar una 'ruta' o comportamiento del personaje: se crea un listado de código de tecla y se va leyendo, con lo que el personaje adquiere 'autonomía' respecto al jugador. Quizás para posteriores versiones...

CONTROL DE ENEMIGOS Y DETECCIÓN DE TORTAZO

Para simplificar la detección del contacto con un enemigo o un obstáculo, se han programado dichos contactos dependiendo solamente de la posición “x” de Rompetechos en la pantalla. En la subrutina de cada ‘enemigo’ u obstáculo se comprueba si esa posición supone recibir una bofetada o un chapuzón, o bien si es una pared o límite de pantalla.

Por ejemplo, la línea 0101 pilota el río. Si Rompetechos está en la columna menor de 15, es que se ha caído al agua (**IF x<15 THEN LET f=1**), y al hacer **RETURN** y volver al motor de movimiento, se llegará a **IF f THEN GO TO u** (es decir, si hay daño, ir a la rutina en línea **u=85** –tortazo- o **u=86** –chapuzón-)

Por tanto es el ‘enemigo’ o el obstáculo quien comprueba si Rompetechos está en tal situación, y no el motor de Rompetechos el que detecta si hay contacto, lo que haría más lento el movimiento al tener que programar todas las formas posibles de contacto.

Usar la posición como condición de contacto hace mucho más rápida la detección de contactos cuando el juego es unidimensional (Rompetechos solo se mueve a izquierda y derecha, siempre en la misma línea), frente al tradicional uso de **ATTR** o **SCREEN\$**. En otro tipo de juegos (bidireccionales) sí que **ATTR** puede ser más versátil y rápido que detectar contacto por posición.

Se ha explicado que cada enemigo se divide en varias subrutinas para cada secuencia de su movimiento, con lo que el resultado es más rápido, aunque consume más memoria.

PANTALLA GRÁFICA Y PANTALLA LÓGICA

Las tres líneas en las que siempre se mueve Rompetechos se almacenan en las tres variables **a\$, b\$, c\$**. Al desplazarse, se imprime en la última posición el valor de **a\$, b\$, c\$**. Al crear cada pantalla, con cada bloque se almacenan en estas tres variables, en la posición adecuada, los gráficos de las tres líneas en las que está Rompetechos.

Los distintos objetos y situaciones en una pantalla (puertas, escaleras, guardia, estanterías, carta, etc.) se detectan al pulsar **ESPACIO**, para lo cual se almacenan en la variable **d\$**, con los códigos ya explicados. Así, al pulsar **ESPACIO**, se lee el carácter en **d\$(x)**, siendo **x** la posición de Rompetechos. De este modo, no es necesario incluir la detección de objetos o elementos del escenario en el motor de movimiento, que lo haría más lento.

MOVIMIENTO

Ya se ha explicado que todas las secuencias de movimiento de Rompetechos se han programado de una en una, repitiendo muchas líneas e instrucciones, lo cual consume mucha memoria, pero el resultado es mucho más rápido que programando un solo bucle para todos los movimientos, controlado con **IF+Operadores Lógicos+Variables alfanuméricas y numéricas**.

Solo el movimiento de cabeza es aleatorio, restando algo de velocidad pero dotando al personaje de un poco más de personalidad propia.

PANTALLAS FORMADAS POR BLOQUES ESTÁNDAR

La forma más económica de tener un mapeado extenso es codificar las pantallas, las cuales se componen de combinaciones de bloques estándar. Pero la ejecución es lenta, como en todas las soluciones que economizan memoria.

La codificación incluye un carácter como símbolo del bloque, y un número que puede ser su posición en la pantalla (así por ejemplo podemos poner las farolas en cualquier lugar), o bien su color, incluso ambos. Por ejemplo, la primera pantalla (número 1051) tiene el código “X00M07A08A16016nn”, donde:

- **X00** es el terreno alrededor del río
- **M07** es el seto elevado, en posición columna 7
- **A08** es un árbol, en posición 8
- **A16** es otro árbol, en posición 16

- **016** indica la posición del guardia (y de la carta) en esa pantalla, en caso de que aparezcan; sería la columna 16
- **nn** indica que en el extremo izquierdo de la pantalla no hay flechas para cambio de pantalla, y tampoco en el derecho

Se lee la cadena alfanumérica con los códigos, en tramos de 3 en 3 caracteres, hasta llegar al código de dos letras formado por “n” o “s” que indica fin de los bloques de esa pantalla. Así las pantallas pueden tener distinto número de bloques.

Los bloques se superponen en el orden en que se indican en la cadena de códigos. En el ejemplo anterior, el árbol se superpone al seto elevado, con lo que se tiene una variedad de árbol más ancho y con dos tonos de verde.

COLOR TRANSPARENTE

La forma más rápida de mover el personaje por la pantalla es con color transparente de papel (**PAPER 8, BRIGHT 8**), siendo todos los elementos y los personajes dibujados en tinta negra.

Además, el color transparente hace que se oculte el sprite cuando **PAPER=INK**, por ejemplo en los extremos de las pantallas, en las paredes internas de alguna sala, en la máquina de RX, y entre vagones del metro. Con ello se consigue cierto efecto de profundidad relativa entre el personaje y el escenario en esos puntos.

SONIDOS

Es muy difícil hacer buenos efectos de sonido en BASIC, y además detienen el movimiento durante la duración de la nota. Por eso se deben elegir sonidos con duración muy corta, y tono muy alto (semáforos, RX) o muy bajo (pasos, tapeo).

GRÁFICOS

La estrategia de diseño de los gráficos va encaminada a facilitar la velocidad manteniendo definición. Para ello:

- Los bloques de las pantallas se dibujan en negro en sus bordes (**INK 0**), coloreando el interior con el color de **PAPER**, y haciendo variaciones con **BRIGHT**. Se intenta simular así la técnica de los tebeos, con borde negro y relleno de color (con las limitaciones del Spectrum sumadas a las del BASIC)
- Los caracteres unitarios (aristas + esquinas) son combinables de forma múltiple para hacer bordes internos y externos, rectos y curvos.
- Los caracteres además son multipropósito: los caracteres de las olas se usan para la fila inferior de adoquines en las calles; las rejas se hacen con el mismo gráfico que la base de los semáforos; las corcheras en la piscina sirven para unir vagones del metro; las asas de las rejas de las tiendas se usan en los gráficos de vigas; el gráfico para las rejas de las tiendas se usa para las escaleras que bajan a las alcantarillas o bajan de los barcos, etc.
- Lo único que no se consiguió hacer re-aprovechable son los gráficos de los personajes, es difícil recombinarlos para conseguir otros personajes. Ese es un reto para futuros programas.