Manual de Choose Your Destiny

El programa es un compilador e intérprete para ejecutar historias de tipo "Escoge tu propia aventura" o aventuras por opciones y libro juegos, para el Spectrum 48, 128, +2 y +3.

Consiste una máquina virtual que va interpretando comandos que se encuentra durante el texto para realizar las distintas acciones interactivas y un compilador que se encarga de traducir la aventura desde un lenguaje similar a BASIC, con el que se escribe el guion de la aventura, a un fichero interpretable por el motor.

Además, también puede mostrar imágenes comprimidas y almacenadas en el mismo disco, así como efectos de sonido basados en BeepFX de Shiru, melodías tipo PT3 creadas con Vortex Tracker o melodías creadas con WyzTracker 2.0.

- WHILE (condexpression) ... WEND
- REPEAT ... UNTIL (condexpression)
- SET varID TO varexpression
- SET [varID] TO varexpression
- SET varID TO {varexpression1, varexpression2,...}
- SET [varID] TO {varexpression1, varexpression2,...}
- LET varID = varexpression
- LET [varID] = varexpression
- LET varID = {varexpression1, varexpression2,...}
- LET [varID] = {varexpression1, varexpression2,...}
- END
- CLEAR
- CENTER
- WAITKEY
- OPTION GOTO ID
- OPTION GOSUB ID
- OPTION VALUE(varexpression) GOTO ID
- OPTION VALUE(varexpression) GOSUB ID
- CHOOSE
- CHOOSE IF WAIT expression THEN GOTO ID
- CHOOSE IF CHANGED THEN GOSUB ID
- OPTIONSEL()
- NUMOPTIONS()
- OPTIONVAL()
- CLEAROPTIONS
- MENUCONFIG varexpression, varexpression, varexpression, varexpression
- MENUCONFIG varexpression, varexpression, varexpression
- MENUCONFIG varexpression, varexpression
- CHAR varexpression
- REPCHAR expression, expression
- TAB expression
- PRINT varexpression
- PAGEPAUSE expression
- INK varexpression
- PAPER varexpression
- BORDER varexpression
- BRIGHT varexpression
- FLASH varexpression
- NEWLINE expression
- NEWLINE
- BACKSPACE expression
- BACKSPACE
- SFX varexpression
- PICTURE varexpression
- DISPLAY varexpression

Requerimientos e Instalación

Estos son los requerimientos externos de la herramienta:

- Python 3.11 o superior
- SjAsmPlus 1.20.3 o superior
- Descompresor de ficheros ZIP

Si se actualiza una versión más antigua, es recomendable NO sobrescribirla. Es mejor renombrar el directorio de la versión antigua, descomprimir la nueva versión, copiar los archivos de tu aventura a la nueva versión y configurar de nuevo el guion `make_adv` para cada caso.

Estas son las instrucciones más detalladas para cada sistema operativo:

Instalación en Windows

La instalación es sencilla, simplemente descomprimir el fichero ZIP correspondiente descargado de la sección Releases del repositorio. En este caso, está todo incluido en el paquete. Se requiere Windows 10 o superior, versión de 64 bits (32 bits no soportados).

Instalación en Linux, BSDs y compatibles

Para estos sistemas, los requerimientos son:

- GNU/Linux / Unix / macOS / BSD con shell compatible con BASH.
- Python 3.11 o superior
- Todos los programas comunes del sistema (grep, cat, etc...).
- Todos los programas requeridos para compilar programas C en el sistema (libc, libstdc++, g++, GNU make, etc...).
- Autotools (autoconf, automake, aclocals...).
- wget
- git

Estos requerimientos son necesarios para compilar `SjAsmPlus`. No existe distribución en binario de esta herramienta para sistemas compatibles UNIX, con lo que se requiere compilarla directamente.

Debido a la heterodoxa naturaleza de las diferentes distribuciones, me resulta imposible dar instrucciones detalladas para instalar los requerimientos en cada caso en particular, con lo que se requieren conocimientos por parte del usuario para ello.

Lo primero que hay que hacer es clonar el repositorio del motor en cualquier lugar que creas conveniente y de forma recursiva para bajarse las dependencias:

```
git clone --recursive https://github.com/cronomantic/ChooseYourDestiny.git
cd ChooseYourDestiny
```

Ahora se puede proceder a compilar **SjASMPlus** con los siguientes comandos:

```
cd external/sjasmplus
make clean
make
cd ..
```

Y con esto ya tenemos preparadas las dependencias. Como último paso, debemos poner el script de compilación como ejecutable:

```
cd ../..
chmod a+x make_adv.sh
```

Y con esto ya podrías usar la herramienta en Linux.

CYDC (Compilador)

Este programa es el compilador que traduce el texto de la aventura a un fichero TAP o DSK. Además de compilar la aventura en un fichero interpretable por el motor, realiza una búsqueda de las mejores abreviaturas para reducir el tamaño del texto.

```
cydc_cli.py [-h] [-l MIN_LENGTH] [-L MAX_LENGTH] [-s SUPERSET_LIMIT]
              [-T EXPORT-TOKENS_FILE] [-t IMPORT-TOKENS-FILE] [-C EXPORT-CHARSET]
              [-c IMPORT-CHARSET] [-S] [-n NAME] [-img IMAGES_PATH] [-trk
TRACKS_PATH]
              [-sfx SFX_ASM_FILE] [-scr LOAD_SCR_FILE] [-v] [-V] [-trim]
              [-wyz] [-nl NUM_LINES] [-720]
              {48k,128k,plus3} input.txt SJASMPLUS_PATH OUTPUT_PATH
```

- **-h**: Muestra la ayuda
- **-l MIN_LENGTH**: La longitud mínima de las abreviaturas a buscar (por defecto, 3).
- **-L MAX_LENGTH**: La longitud máxima de las abreviaturas a buscar (por defecto, 30).
- **-s SUPERSET_LIMIT**: Límite para el superconjunto de la heurística de la búsqueda (por defecto, 100).
- **-T EXPORT-TOKENS_FILE**: Exportar al fichero JSON indicado por el parámetro las abreviaturas encontradas.
- **-t IMPORT-TOKENS-FILE**: Importar abreviaturas desde el fichero indicado y obviar la búsqueda de las mismas.

- **-C EXPORT-CHARSET**: Exporta el juego de caracteres 6x8 usado por defecto en formato JSON.
- **-c IMPORT-CHARSET**: Importa en formato JSON el juego de caracteres a emplear.
- **-S**: Si un fragmento de texto comprimido no cabe en un banco, se divide en dos entre el banco actual y el siguiente con esta opción activada. Si no, el fragmento pasa al banco siguiente.
- **-n NAME**: Nombre a usar para el fichero de salida (nombre para el fichero TAP o DSK), si no se define será el mismo que el fichero de entrada.
- **-scr LOAD_SCR_FILE**: Ruta hacia un fichero SCR con la pantalla de carga a usar.
- **-img IMAGES_PATH**: Ruta al directorio con las imágenes de la aventura.
- **-trk TRACKS_PATH**: Ruta al directorio con los ficheros PT3 de música AY o ficheros de WyzTracker.
- **-sfx SFX_ASM_FILE**: Ruta a un fichero ensamblador generado por BeepFx.
- **-v**: Modo verboso, da más información del proceso.
- **-V**: Indica la versión del programa.
- **-trim**: Elimina el código de aquellos comandos que no se usen en la aventura para reducir el tamaño del intérprete.
- **-pause**: Número de segundos de pausa después de finalizar el proceso de carga, se puede cancelar con cualquier pulsación de tecla.
- **-wyz**: Usar música de tipo WyzTracker, en lugar de Vortex Tracker.
- **-nl NUM_LINES**: Número de líneas que se emplearán en los ficheros de imagen (por defecto, 192).
- **-720**: En caso de usar el formato Plus3, se usará una imagen de disco de 720 Kb en lugar del tamaño estándar de 180 Kb.

-**{48k,128k,plus3}**: Modelo de Spectrum a emplear: -- **48k**: Versión para cinta en formato TAP, no incluye el reproductor de PT3 ni WyzTracker y se carga todo de una vez. Depende del tamaño de la memoria disponible. -- **128k**: Versión para cinta en formato TAP, se carga todo de una vez en los bancos de memoria y depende del tamaño de la memoria disponible. -- **plus3**: Esta versión generará un fichero DSK para ejecutarlo en Spectrum+3. Los recursos se cargan dinámicamente según se necesiten y depende del tamaño en disco.

- **input.txt**: Fichero de entrada con el guion de la aventura.
- **SJASMPLUS_PATH**: Ruta al ejecutable de SjASMPlus.
- **OUTPUT_PATH**: Ruta donde se depositarán los ficheros de salida.

El compilador es un programa escrito en Python, por lo que se requiere tener el entorno de Python instalado. Para mayor comodidad, se incluye en la distribución un Python embebido y un guion batch llamado `cydc.cmd` para lanzarlo desde la línea de comandos.

El compilador depende de un programa externo para funcionar, el ensamblador `SjASMPlus`, por lo que habrá que indicar en los parámetros de entrada la correspondiente ruta a este programa, el cual está incluido en la distribución en el directorio `tools`.

También hay que indicar un directorio de destino para dejar los ficheros resultantes de la compilación.

Opcionalmente, podemos indicar las rutas a directorios que contengan las imágenes comprimidas en formato `scr`, que se incluirán en la cinta o disco resultante. Lo mismo para los ficheros de tipo `PT3`. Ambos tipos de archivos deben estar nombrados con números de 3 dígitos, del 0 al 255, de tal forma que sean `000.scr`, `001.scr`, `002.PT3`, y así. Se indicarán los directorios que contienen ambos ficheros con los parámetros `-img` y `-trk` respectivamente.

Las imágenes serán comprimidas en un fichero de formato `CSC`. Las pantallas pueden ser completas, o se puede limitar el número de líneas horizontales para ahorrar memoria. Además detecta imágenes espejadas

(simétricas) por el eje vertical, con lo que sólo almacena la mitad de la misma, pudiéndose incluso forzar este comportamiento y descartar el lado derecho de la imagen para ahorrar espacio. Más detalles de éste funcionamiento en la sección Imágenes.

Para la música, podemos usar módulos de Vortex Tracker (`PT3`), o música creada con WyzTracker v2.0 y superiores. El comportamiento está también descrito en sus secciones correspondientes: Melodías Vortex Tracker y Melodías WyzTracker.

Se pueden añadir efectos de sonido generados con la aplicación BeepFx de Shiru. Para utilizarlos, hay que exportar usando la opción `File->Compile` del menú superior. En la ventana flotante que aparece, debemos asegurarnos de que tenemos seleccionado `Assembly` e `Include Player Code`, el resto de las opciones son indiferentes. Luego guardar el fichero en algún punto accesible para indicar la ruta desde la línea de comandos con la opción `-sfx`.

CYD Character Set Converter

Esta utilidad permite convertir juegos de caracteres en formato `.chr`, `.ch8`, `.ch6` y `.ch4` en un fichero utilizable por el compilador en formato JSON. Estos formatos son editables con ZxPaintbrush.

```
cyd_chr_conv.py [-h] [-w WITDH_LOW] [-W WITDH_UP] [-v] [-V] charset.chr
charset.json
```

Los parámetros que soporta:

- **-w, --width_low**: Ancho de los caracteres (1-8) del juego de caracteres inferior.
- **-W, --width_high**: Ancho de los caracteres (1-8) del juego de caracteres superior.
- **-h, --help**: Muestra la ayuda.
- **-v, --verbose**: Modo verboso.
- **-V, --version**: Versión del programa.
- **charset.chr**: Huego de caracteres de entrada.
- **charset.json**: Fichero con el juego de caracteres para el compilador.

El ancho de los caracteres utilizado por defecto es 6 para el juego de caracteres inferior (desde primero al número 127) y 4 para el superior (desde el número 145 hasta el 256), pero se pueden cambiar estos valores con los parámetros `-w` y `-W` respectivamente. Indicar que los caracteres del 128 al 144 son especiales, ya que se usan para los cursores y siempre tendrán ancho 8, con lo que el tamaño de la fuente será ignorado en esos caracteres. Si se desea definir un ancho específico para cada carácter tendrás que editarlo en el fichero JSON de salida. Tienes más información en la sección Juego de caracteres.

Sintaxis Básica

Los comandos para el intérprete se delimitan dentro de dos pares de corchetes, abiertos y cerrados respectivamente. Todo texto que aparezca fuera de esto, se considera "texto imprimible", incluidos los espacios y saltos de línea, y se presentarán como tal por el intérprete. Los comandos se separan entre sí con saltos de línea o dos puntos si están en la misma línea.

Los comentarios dentro del código se delimitan con `/*` y `*/`, todo lo que haya en medio se considera un comentario.

Este es un ejemplo resumido y auto explicativo de la sintaxis:

```
Esto es texto [[ INK 6 ]] Esto es texto de nuevo pero amarillo
    Sigue siendo texto [[
         /* Esto es un comentario y lo siguiente son comandos */
        WAITKEY
        INK 7: PAPER 0
    ]]
    Esto vuelve a ser texto pero blanco, y ¡ojo con el salto de línea que lo
precede!
```

El intérprete recorre el texto desde el principio, imprimiéndolo en pantalla si es "texto imprimible". Cuando una palabra completa no cabe en lo que queda de la línea, la imprime en la línea siguiente. Y si no cabe en lo que queda de pantalla, se genera una espera y petición al usuario de que pulse la tecla de confirmación para borrar la sección de texto y seguir imprimiendo (este último comportamiento es opcional).

Cuando el intérprete detecta comandos, los ejecuta secuencialmente, a menos que encuentre saltos. Los comandos permiten introducir lógica programable dentro del texto para hacerlo dinámico y variado según ciertas condiciones. La más común y poderosa es la de solicitar escoger al jugador entre una serie de opciones (hasta un límite de 8 a la vez), y que puede elegir con las teclas `P` y `Q` o los cursores, y seleccionar con `SPACE`, `M` o `ENTER`. También soporta un joystick de tipo Kempston para desplazarse por el menú. De nuevo, éste es un ejemplo auto explicativo:

```
[[ /* Comandos que ponen colores de pantalla y la borra */
   PAPER 0   /* Color de fondo a cero (negro)*/
   INK   7   /* Color de la tinta (blanco) */
   BORDER 0  /* Color del borde (negro) */
   CLEAR     /* Borrar el área de texto (pantalla completa) */
   LABEL Localidad1]]Estás en la localidad 1. ¿Donde quieres ir?
[[OPTION GOTO Localidad2]]Ir a la localidad 2
[[OPTION GOTO Localidad3]]Ir a la localidad 3
[[CHOOSE]]
[[ LABEL Localidad2]]¡¡¡Lo lograste!!!
[[ GOTO Final]]
[[ LABEL Localidad3]]¡¡¡Estas muerto!!!
[[ GOTO Final]]
[[ LABEL Final : WAITKEY: END ]]
```

El comando `OPTION GOTO etiqueta` generará un punto de selección en el lugar en donde se haya llegado al comando. Cuando llegue al comando `CHOOSE`, el intérprete permitirá elegir al usuario entre uno de los puntos de opción que haya acumulados en pantalla hasta el momento. Se permiten un máximo de 32.

Al escoger una opción, el intérprete saltará a la sección del texto donde se encuentre la etiqueta correspondiente indicada en la opción. Las etiquetas se declaran con el pseudo-comando `LABEL`

`identificador` dentro del código, y cuando se indica un salto a la misma, el intérprete comenzará a procesar a partir del punto en donde hemos declarado la etiqueta. En el caso del ejemplo, si elegimos la opción 1, el intérprete saltará al punto indicado en `LABEL localidad2`, con lo que imprimirá el texto "*¡¡¡Lo lograste!!!*", y después pasa a `GOTO Final` que hará un salto incondicional a donde está definido `LABEL Final` e ignorando todo lo que haya entre medias.

Los identificadores de las etiquetas sólo soportan caracteres alfanuméricos (cifras y letras), deben empezar con una letra y son sensibles al caso (se distinguen mayúsculas y minúsculas), es decir `LABEL Etiqueta` no es lo mismo que `LABEL etiqueta`. Los comandos, por el contrario, no son sensibles al caso, pero por claridad, es recomendable ponerlos en mayúsculas.

También se permite una versión acortada de las etiquetas precediendo el caracter `#` al identificador de la etiqueta, de tal manera que `#MiEtiqueta` es lo mismo que `LABEL MiEtiqueta`. Con esto, podríamos reescribir el anterior código así:

```
[[ /* Comandos que ponen colores de pantalla y la borra */
   PAPER 0   /* Color de fondo a cero (negro)*/
   INK   7   /* Color de la tinta (blanco) */
   BORDER 0  /* Color del borde (negro) */
   CLEAR     /* Borrar el área de texto (pantalla completa) */
   LABEL Localidad1]]Estás en la localidad 1. ¿Donde quieres ir?
[[OPTION GOTO Localidad2]]Ir a la localidad 2
[[OPTION GOTO Localidad3]]Ir a la localidad 3
[[CHOOSE]]
[[ #Localidad2 ]]¡¡¡Lo lograste!!!
[[ GOTO Final ]]
[[ #Localidad3 ]]¡¡¡Estas muerto!!!
[[ GOTO Final]]
[[ #Final : WAITKEY: END ]]
```

Los comandos disponibles están descritos en su sección correspondiente.

Variables y expresiones numéricas

Los valores numéricos constantes se puede expresar en base 10 por defecto, en hexadecimal con el prefijo `0x` o en binario con el prefijo `0b`. Por ejemplo, `240` sería en decimal, `0xF0` en hexadecimal y `0b11110000` en binario.

Hay a disposición del programador 256 contenedores de un byte (de 0 a 255) para almacenar valores, realizar operaciones y comparaciones con ellos. Constituyen el estado del programa. A partir de este momento, pueden ser llamados variables, banderas o "variables" indistintamente a lo largo de este documento.

Para referirnos a una variable en una expresión numérica, el número debe estar precedido por el carácter `@`. Pero es posible dar un nombre significativo a las variables usando el comando `DECLARE` de la siguiente manera:

```
[[
DECLARE 7 AS ColorTinta
INK @ColorTinta
]]
```

Hay que indicar que no se puede declarar una variable dos veces, ni puede haber etiquetas y variables con los mismos nombres, pero se permiten sinónimos de la siguiente forma:

```
[[
DECLARE 10 AS UnNombre
DECLARE 10 AS OtroNombre
]]
```

Así, tanto *UnNombre* como *OtroNombre* servirán para identificar el variable 10. Ten en cuenta de a pesar de tener distinto nombre, son la misma variable.

Para asignar valores a un variable, usamos el comando `SET varId TO varexpression`, de la siguiente manera:

```
[[
  SET 0 TO 1                     /* Ponemos el variable cero a 1 */
  SET variable TO 2              /* Ponemos el variable llamado variable a 2 */
  SET variable2 TO @variable + 2  /* Ponemos el variable llamado variable2 al dos
sumado al valor del variable llamado variable */
  SET variable2 TO @variable2 - (@variable + 2)  /* Permite paréntesis */
]]
```

También tenemos ahora el siguiente sinónimo con `LET varId = varexpression`, de tal forma que los ejemplos anteriores se podrían reescribir así:

```
[[
  LET 0 = 1                     /* Ponemos el variable cero a 1 */
  LET variable = 2              /* Ponemos el variable llamado variable a 2 */
  LET variable2 = @variable + 2  /* Ponemos el variable llamado variable2 al dos
sumado al valor del variable llamado variable */
  LET variable2 = @variable2 - (@variable + 2)  /* Permite paréntesis */
]]
```

Como se puede ver, a un variable se le puede asignar el valor de una expresión matemática compuesta por números, funciones y variables. Como ya se ha indicado, las variables dentro de una expresión numérica, es decir, el lado derecho de una asignación **siempre deben estar precedidas del carácter `@`**.

Los operandos disponibles son:

- Suma: `SET variable TO @variable + 2`
- Resta: `SET variable TO @variable - 2`
- "AND" binario: `SET variable TO @variable & 2`
- "OR" binario: `SET variable TO @variable | 2`
- "NOT" binario o complemento de bits: `SET variable TO !@variable`
- Desplazamiento de bits a la izquierda: `SET variable TO @variable << 2`
- Desplazamiento de bits a la derecha: `SET variable TO @variable >> 2`

El resultado de una expresión numérica no pueden ser mayor de 255 (1 byte) ni menor que cero (no se soportan números negativos). Si al realizar las operaciones se rebasan ambos límites, el resultado se ajustará al límite correspondiente, es decir, si una suma supera 255, se ajustará a 255 y una resta que dé un resultado inferior a cero, se quedará en cero.

Una cosa a destacar es que los operadores binarios **no son los mismos que los operadores lógicos de las expresiones condicionales** descritos más abajo. Un **&** no es lo mismo que un **AND**. Los operadores binarios realizan las correspondientes operaciones sobre los bits del variable.

La mayor parte de los comandos aceptan expresiones numéricas en sus parámetros, por ejemplo:

```
[[
INK 7
INK @7
]]
```

El primer comando pondrá el color del texto en blanco (color 7), mientras que el segundo pondrá el color del texto con el valor contenido en la variable número 7. Combinando con todo lo anteriormente descrito, se podría hacer:

```
[[
INK 3+4
INK @Var-1
]]
```

Consulta la referencia de comandos para saber cuáles de ellos los aceptan.

Por último, hay una serie de comandos especiales llamados **funciones**. Estos comandos no pueden usarse por sí solos, sino que deben usarse dentro de una expresión numérica, ya que devuelven un valor a emplear dentro de ella.

- **RANDOM(expression)**: Devuelve un número aleatorio entre 0 y el valor indicado en **expression**.
- **RANDOM()** : Devuelve un número aleatorio entre 0 y 255. Es el equivalente a `RANDOM(255)`.
- **RANDOM(expression,expression)**: Devuelve un número aleatorio entre el valor indicado en el primer parámetro y el segundo, ambos inclusive.
- **INKEY()** Se espera hasta que se pulse una tecla y devuelve el código de la tecla pulsada (sólo teclado).
- **KEMPSTON()** Devuelve los botones pulsados en un joystick kempston conectado al Spectrum.

- **MIN(varexpression,varexpression)**: Acota el valor del primer parámetro al mínimo indicado por el segundo. Es decir, si el parámetro 1 es menor que el parámetro 2, se devuelve el parámetro 2, en otro caso se devuelve el 1.
- **MAX(varexpression,varexpression)**: Acota el valor del primer parámetro al máximo indicado por el segundo. Es decir, si el parámetro 1 es mayor que el parámetro 2, se devuelve el parámetro 2, en otro caso se devuelve el 1.
- **POSY()**: Devuelve la fila actual en la que se encuentra el cursor en coordenadas 8x8.
- **POSX()**: Devuelve la columna actual en la que se encuentra el cursor en coordenadas 8x8 (Debido a la naturaleza de la fuente de ancho variable, el valor devuelto será la columna 8x8 donde esté actualmente el cursor).
- **LASTPOS(ID)**: Devuelve el índice de la última posición del array dado.

Así que, por ejemplo `SET variable_no TO RANDOM(6)`, asigna a la variable *variable_no* un número aleatorio del 0 al 6. Y de la misma manera, podemos operar con ellos: `SET variable_no TO RANDOM(6) + @var + 1`

Control de flujo y expresiones condicionales

Para controlar el flujo de ejecución, tenemos diferentes comandos, que describiré ahora:

- **GOTO ID**

Salta a la etiqueta *ID*.

- **GOSUB ID**

Salto de subrutina, hace un salto a la etiqueta *ID*, pero devuelve la ejecución a la siguiente instrucción al `GOSUB` en cuanto encuentre un comando `RETURN`.

- **RETURN**

Retorna al punto posterior de la llamada de la subrutina, ver `GOSUB`

- **IF condexpression THEN ... ENDIF**

Si la expresión condicional *condexpression* resulta cierta, se imprime el texto y se ejecutan los comandos que haya desde `THEN` hasta `ENDIF`.

- **IF condexpression THEN ... ELSE ... ENDIF**

Si la expresión condicional *condexpression* resulta cierta, se imprime el texto y se ejecutan los comandos que haya desde `THEN` hasta `ELSE`. En caso contrario, se imprime el texto y se ejecutan los comandos que haya desde `ELSE` hasta `ENDIF`

- **IF condexpression THEN ... ELSEIF condexpression THEN [... ELSEIF condexpression THEN] ... ELSE ... ENDIF**

Si la expresión condicional *condexpression1* resulta cierta, se imprime el texto y se ejecutan los comandos que haya desde `THEN` hasta `ELSE`. En caso contrario, se verifica si la expresión condicional *condexpression2* se cumple, en cuyo caso imprime el texto y se ejecutan los comandos que haya desde `ELIF` hasta el `ELSE` u otro `ELSEIF`. Se pueden encadenar varios `ELSEIF` hasta que se llegue a una última cláusula `ELSE`, que se ejecuta si ninguna de las condiciones anteriores se ha cumplido.

- **WHILE (condexpression) ... WEND**

Se repiten repetidamente el texto y los comandos que haya hasta el `WEND` mientras la condición *condexpression* evalúe a cierto. La evaluación se realiza al principio de cada iteración.

- **REPEAT ... UNTIL (condexpression)**

Se repiten repetidamente el texto y los comandos que haya desde `REPEAT` hasta el `UNTIL` mientras la condición *condexpression* evalúe a falso. La evaluación se realiza al final de cada iteración.

Como se puede ver, muchos de estas funciones se ejecutan dependiendo de si se cumple una condición. Por ejemplo:

```
[[IF @var = 0 THEN GOTO salto ENDIF]]
```

Para que se ejecute el salto, tiene que cumplirse que la variable *var* sea igual a cero. Esto es lo que se llama una expresión condicional.

Una condición es siempre una comparación entre dos expresiones numéricas:

- Igual que: `IF @variable = 0 THEN GOTO salto ENDIF`
- Mayor que: `IF @variable > 0 THEN GOTO salto ENDIF`
- Menor que: `IF @variable2 < @variable THEN GOTO salto ENDIF`
- Distinto que: `IF @variable <> 0 THEN GOTO salto ENDIF`
- Mayor o igual que: `IF @variable <= 0 THEN GOTO salto ENDIF`
- Menor o igual que: `IF @variable >= 0 THEN GOTO salto ENDIF`

Además, las condiciones se pueden combinar formando expresiones condicionales mediante operaciones lógicas. Por ejemplo:

```
[[IF (@variable = 0 AND @variable2 = 1) AND NOT @variable3 = 1 THEN GOTO salto
ENDIF]]
```

- Cond1 **AND** Cond2: Cierto si se cumple tanto como Cond1 como Cond2.
- Cond1 **OR** Cond2: Cierto si se cumple Cond1 o Cond2.
- **NOT** Cond1: Cierto si la condición Cond1 es falsa.

Recuerdo de nuevo que los operadores lógicos de las expresiones condicionales **no son los mismos que los operadores binarios**.

Además, para aquellos que tengan nociones avanzadas de programación, las condiciones no son "cortocircuitantes", es decir, se evalúan todas las operaciones lógicas y comparaciones hasta el final.

Asignaciones e indirección

Las asignaciones y expresiones numéricas ya las hemos visto en su sección, pero vamos a explicar más detalladamente su funcionamiento con la indirección, lo cual merece una sección aparte.

La indirección se realiza poniendo entre corchetes simples una expresión numérica, por ejemplo `[@var+1]`. Lo que estamos queriendo decir es que **nos vamos a referir al variable o variable con el número calculado por la expresión numérica encerrada entre corchetes**. Vamos a verlo con ejemplos:

- `SET var to [2+2]`: indica que vamos a asignar a *var* el contenido del variable 2+2, es decir 4. Esto es equivalente a `SET var TO @4`, pero nos permite ir afianzando el concepto.
- `SET var TO [@indice]`: aquí es donde se muestra realmente la utilidad, ya que estamos indicando que guardamos en *var* el contenido de la variable cuyo número corresponde con el contenido de la variable *indice*. Es decir, podemos cambiar de forma dinámica la variable a la cual hacemos la asignación.
- `SET var TO [@indice+2]`: la indirección soporta cualquier expresión numérica, esto significa que guardamos en *var* el contenido de la variable corresponda con el contenido de *indice* mas dos.
- `SET [var] TO 0`: La indirección también puede usarse en el lado derecho, y en este caso, no se soportan expresiones numéricas, sólo el número o el identificador de la variable si se ha declarado. Esto significa "asigna cero a la variable cuyo índice corresponda con el contenido de la variable *var*.
- `INK [@var]`: La indirección también es soportada por aquellos comandos que admitan expresiones numéricas.
- `SET [@@var] TO 0`: Por completitud, ya que operamos con punteros, usando `@@` tenemos la operación complementaria a la indirección, es decir, devuelve el número de índice de una variable dada, con lo que en el ejemplo, la operación sería equivalente a `SET var TO 0`.

Si usamos los indicadores numéricos con las variables (p.ej. `@0`) no resulta muy útil. Su verdadera utilidad reside en usarlo con los identificadores de variables, tal que si tenemos:

```
[[
DECLARE 20 AS var : DECLARE 10 AS index: SET index AS @@var : SET [index] AS 0
]]
```

Vemos que esto nos permite inicializar variables que vayamos a usar para indirección con el identificador de otra variable.

La indirección constituye una herramienta poderosa que nos permite implementar arrays en los variables, pero ¡ojo, que sólo tenemos 256! Además, no hay que confundirlos con los arrays descritos en la siguiente sección, que son de una naturaleza diferente.

Por último, podemos realizar asignaciones múltiples mediante la siguiente construcción:

```
[[
DECLARE 10 AS var
SET var AS {0, 1, 2, 3}
]]
```

Esto sería equivalente a:

```
[[
SET 10 AS 0
SET 11 AS 1
SET 12 AS 2
SET 13 AS 3
]]
```

Es decir, podemos asignar de forma consecutiva una secuencia de valores a una secuencia de variables, comenzando por la variable indicada en el lado izquierdo. Lo cual permite asignar múltiples valores de forma consecutiva.

Constantes

A lo largo del desarrollo, puede que necesitemos tener valores con una denominación significativa. De la misma forma que con las variables, podemos darles nombre con `CONST nombre = valor`, y luego usar `nombre` en su lugar. Por ejemplo:

```
[[
  CONST maxBalas = 10      /* declaramos la constante maxBalas con valor 10 */
  PRINT maxBalas           /* imprimimos el valor 10 */
]]
```

Al contrario que las variables, podemos usar tantas constantes como queramos, ya que durante el proceso de compilación se sustituirán por su valor correspondiente.

El nombre de la constante no puede coincidir con el de otra variable, etiqueta o alguno de los comandos.

Arrays o "secuencias"

En la sección anterior hemos descrito el uso de la indirección para usar las variables como un array, pero `CYD` también dispone de una forma alternativa de crear arrays de datos mediante el comando `DIM`, de tal mnanera que con `DIM nombre(tamaño)` declaramos un array de nombre `nombre` y tamaño `tamaño`. El tamaño de un array no puede ser mayor de 256 ni ser cero. Accedemos a cada uno de los elementos del array con la nomenclatura `nombre(pos)`, donde `pos` es el número de posición del elemento a acceder dentro del array, empezando desde cero. Vamos a verlo con ejemplos:

```
[[
  DIM miArray(5)          /* Declaramos un array de 5 elementos del 0 al 4
*/
  LET miArray(3) = 1      /* Asignamos 1 a la posición 3 del array, que sería la
cuarta */
  PRINT miArray(3)         /* Imprimimos el valor almacenado en la posición 3
*/
]]
```

Si intentamos acceder a una posición fuera del rango, el motor fallará y dará un error. Para saber cual es la última posición accessible del array, tenemos la función `LASTPOS`:

```
[[
  DIM miArray(5)          /* Declaramos un array de 5 elementos  */
  PRINT LASTPOS(miArray)  /* Devolvería 4                        */
]]
```

Asignar valores iniciales a un array se puede hacer de la siguiente forma: `DIM nombre_array(tamaño) = {valor1, valor2, ...}`. Siguiendo el ejemplo anterior:

```
[[
  DIM miArray(5) = {1, 2, 3, 4, 5}   /* Asignamos valores al array */
  PRINT miArray(1)                   /* Esto imprimiría 2          */
]]
```

Las posiciones no inicializadas siempre tendrán valor 0 por defecto, con lo que podemos tener un array con más posiciones que valores. Lo contrario nos daría un error:

```
[[
  DIM miArray(5) = {1, 2, 3}         /* Esto se permite  */
  DIM miArray(3) = {1, 2, 3, 4, 5}   /* Esto daría error */
]]
```

Por último, y como conveniencia, podemos dejar vacío el número de posiciones del array al declararlo con valores. El compilador creará un array de tantas posiciones como elementos haya en la inicialización:

```
[[
  DIM miArray() = {1, 2, 3}  /* Esto se permite, miArray sería de 3 posiciones  */
  DIM miArray()              /* Esto daría error                                 */
]]
```

Este tipo arrays tienen una serie de limitaciones que hay que tener en cuenta a la hora de usarlos. La más evidente es que no pueden tener más de 256 elementos y son unidimensionales y no se pueden redimensionar.

Otra limitación es que no se pueden salvar ni recuperar los datos de estos arrays mediante `SAVE` y `LOAD` directamente. La única opción es mover los datos del array desde o hacia las variables y realizar las operaciones de grabar o recuperar en las mismas.

Por último, si se modifican los datos de un array, no se pueden recuperar los valores originales a menos que se haga una copia en otro array previamente.

Listado de comandos

Antes de describir los comandos, vamos a hablar resumidamente de la leyenda utilizada:

- **ID** es un identificador y es una cadena de cifras, letras o subrayado. No se admiten letras acentuadas ni la ñ.
- **expression** es una expresión numérica sin variables. Es un número en decimal o hexadecimal, pero también se admiten operaciones aritméticas simples que serán calculadas en tiempo de compilación.
- **varexpression**, expresión numérica tal y como es descrita en su sección correspondiente. Puede contener variables con `@` e indirecciones.
- **condexpression**, expresión condicional tal y como es descrita en su sección correspondiente. Consisten en una comparación o varias comparaciones con operaciones lógicas entre las mismas.
- **varID**, identificador de variable, puede ser una *expression* si usamos su índice numérico o un *ID* si lo hemos declarado así con `DECLARE`.

Este es listado completo de comandos:

LABEL ID

Declara la etiqueta labelId en este punto. Todos los saltos con referencia a esta etiqueta dirigirán la ejecución a este punto.

#ID

Versión resumida para declarar la etiqueta *ID*, de tal manera que `#LabelId` es lo mismo que `LABEL LabelId`.

DECLARE expression AS ID

Declara el identificador *ID* como un símbolo que representa al variable *expression* en su lugar.

CONST ID = expression

Declara la constante *ID* con el valor de *expression*.

DIM ID(expression)

Crea un array de nombre *ID* con tantos elementos como se indique en *expression*. El número de elementos no puede ser mayor de 256.

DIM ID(expression) = {expression, expression...}

Igual que DIM ID(expression), pero asignando valores separados por comas.

GOTO ID

Salta a la etiqueta *ID*.

GOSUB ID

Salto de subrutina, hace un salto a la etiqueta *ID*, pero vuelve al siguiente comando cuanto encuentra un comando `RETURN`.

RETURN

Retorna al punto posterior de la llamada de la subrutina, ver `GOSUB`

IF condexpression THEN ... ENDIF

Si la expresión condicional *condexpression* resulta cierta, se imprime el texto y se ejecutan los comandos que haya desde `THEN` hasta `ENDIF`.

IF condexpression THEN ... ELSE ... ENDIF

Si la expresión condicional *condexpression* resulta cierta, se imprime el texto y se ejecutan los comandos que haya desde `THEN` hasta `ELSE`. En caso contrario, se imprime el texto y se ejecutan los comandos que haya desde `ELSE` hasta `ENDIF`

IF condexpression1 THEN ... ELIF condexpression2 THEN ... ELSE ... ENDIF

Si la expresión condicional *condexpression1* resulta cierta, se imprime el texto y se ejecutan los comandos que haya desde `THEN` hasta `ELSE`. En caso contrario, se verifica si la expresión condicional *condexpression2* se cumple, en cuyo caso imprime el texto y se ejecutan los comandos que haya desde `ELIF` hasta el `ELSE` u otro `ELSEIF`. Se pueden encadenar varios `ELSEIF` hasta que se llegue a una última cláusula `ELSE`, que se ejecuta si ninguna de las condiciones anteriores se ha cumplido.

WHILE (condexpression) ... WEND

Se repiten repetidamente el texto y los comandos que haya hasta el `WEND` mientras la condición *condexpression* evalúe a cierto. La evaluación se realiza al principio de cada iteración.

REPEAT ... UNTIL (condexpression)

Se repiten repetidamente el texto y los comandos que haya desde `REPEAT` hasta el `UNTIL` mientras la condición *condexpression* evalúe a falso. La evaluación se realiza al final de cada iteración.

SET varID TO varexpression

Asigna el valor de *varexpression* a la variable *varID*.

SET [varID] TO varexpression

Asigna el valor de *varexpression* a la variable cuyo índice corresponde con el contenido de *varID*.

SET varID TO {varexpression1, varexpression2,...}

Asigna el valor de *varexpression1* a la variable *varID*, *varexpression2* a la variable *varID*+1, y así.

SET [varID] TO {varexpression1, varexpression2,...}

Asigna el valor de *varexpression1* a la variable cuyo índice corresponde con el contenido de *varID*, *varexpression2* a la variable cuyo índice corresponde con el contenido de *varID*+1, y así.

LET varID = varexpression

Asigna el valor de *varexpression* a la variable *varID*.

LET [varID] = varexpression

Asigna el valor de *varexpression* a la variable cuyo índice corresponde con el contenido de *varID*.

LET varID = {varexpression1, varexpression2,...}

Asigna el valor de *varexpression1* a la variable *varID*, *varexpression2* a la variable *varID*+1, y así.

LET [varID] = {varexpression1, varexpression2,...}

Asigna el valor de *varexpression1* a la variable cuyo índice corresponde con el contenido de *varID*, *varexpression2* a la variable cuyo índice corresponde con el contenido de *varID*+1, y así.

END

Finaliza la aventura y reinicia el ordenador.

CLEAR

Borra el área de texto definida.

CENTER

Pone el cursor de impresión en el centro de la línea.

WAITKEY

Espera la pulsación de la tecla de aceptación para continuar, presentando un icono animado de espera. Ideal para separar párrafos o pantallas.

OPTION GOTO ID

Crea un punto de opción que el usuario puede seleccionar (ver `CHOOSE`). Si confirma esta opción, salta a la etiqueta *ID*. Si se borra la pantalla, el punto de opción se elimina y sólo se permiten 32 como máximo en una pantalla. Los puntos de opción se van acumulando en orden de declaración en una lista que será recorrida de acuerdo a la pulsación de las teclas de manejo, y en el mismo orden en el que se declararon los puntos de opción. Si resulta la opción seleccionada en el menú, el valor devuelto por `OPTIONVAL()` es su posición dentro de la lista de opciones del menú.

OPTION GOSUB ID

Crea un punto de opción que el usuario puede seleccionar (ver `CHOOSE`). Si confirma esta opción, hace un salto de subrutina a etiqueta *ID*, volviendo después del `CHOOSE` cuando encuentra un `RETURN`. Si se borra la pantalla, el punto de opción se elimina y sólo se permiten 32 como máximo en una pantalla. Los puntos de

opción se van acumulando en orden de declaración en una lista que será recorrida de acuerdo a la pulsación de las teclas de manejo, y en el mismo orden en el que se declararon los puntos de opción.Si resulta la opción seleccionada en el menú, el valor devuelto por `OPTIONVAL()` es su posición dentro de la lista de opciones del menú.

OPTION VALUE(varexpression) GOTO ID

Funciona igual que `OPTION GOTO ID`, pero le asignamos un valor específico que será lo que devuelva `OPTIONVAL()` si resulta la opción elegida en el menú.

OPTION VALUE(varexpression) GOSUB ID

Funciona igual que `OPTION GOSUB ID`, pero le asignamos un valor específico que será lo que devuelva `OPTIONVAL()` si resulta la opción elegida en el menú.

CHOOSE

Detiene la ejecución y permite al jugador seleccionar una de las opciones que haya en este momento en pantalla. Realizará el salto a la etiqueta indicada en la opción correspondiente. La selección se realiza con las teclas **O** y **P** para "desplazamiento horizontal" y **Q** y **A** para desplazamiento vertical. También se pueden usar las direcciones correspondientes del joystick Kempston y las teclas del cursor. Cuando se pulsan las teclas, un puntero se desplaza sobre la lista de opciones realizando incrementos o decrementos de acuerdo a la configuración actual del mismo (ver MENUCONFIG). Por defecto, sólo tiene configurado un desplazamiento vertical con las teclas **Q** y **A** ó **arriba** y **abajo** en el joystick Kempston o teclas de cursor. Es responsabilidad del usuario colocar los puntos de opción en pantalla de una forma coherente al movimiento del menú configurado.

Recuerda que si se borra la pantalla antes de este comando, perderás las opciones y dará un error de sistema.

CHOOSE IF WAIT expression THEN GOTO ID

Funciona exactamente igual que `CHOOSE`, pero con la salvedad de que se declara un timeout, que si se agota sin seleccionar ninguna opción, salta a la etiqueta *ID*. El timeout tiene como máximo 65535 (16 bits).

CHOOSE IF CHANGED THEN GOSUB ID

Funciona exactamente igual que `CHOOSE`, pero se hace un salto de subrutina a la etiqueta *ID* cada vez que se cambia de opción. Por tanto, es recomendable que se haga un `RETURN` lo antes posible para evitar lentitud. Para saber la opción elegida y el número de opciones total de nuestro menú, podemos usar las funciones `OPTIONSEL()` y `NUMOPTIONS` respectivamente.

OPTIONSEL()

Función que devuelve la opción actualmente seleccionada en el menú actualmente activo. Si no tenemos un menú activo, el resultado de esta función está indefinido.

NUMOPTIONS()

Función que devuelve el número de opciones del menú actualmente activo. Si no tenemos un menú activo, el resultado de esta función está indefinido.

OPTIONVAL()

Función que devuelve el valor asignado a la opción seleccionada y aceptada en el menú, sólo se actualiza cuando una opción es elegida y se sale del menú.

CLEAROPTIONS

Elimina las opciones almacenadas en el menú.

MENUCONFIG varexpression, varexpression, varexpression, varexpression

Configura el menú de opciones. Los parámetros que se pueden configurar son los siguientes:

- El primer parámetro determina el incremento o decremento del número de opción seleccionado cuando pulsamos **P** y **O** respectivamente (o las teclas de izquierda y derecha del cursor o del joystick).
- El segundo parámetro determina el incremento o decremento del número de opción seleccionado cuando pulsamos **A** y **Q** respectivamente (o las teclas de arriba y abajo del cursor o del joystick).
- El tercer parámetro es la opción que se seleccionará al principio cuando se inicie el menú con `CHOOSE`. El valor por defecto es cero (la primera opción registrada).
- El cuarto parámetro, si es cero no muestra el icono de selección y si es distinto de cero, lo muestra.

El comportamiento al iniciarse el intérprete es como si se hubiese ejecutado `MENUCONFIG 0,1,0,1`. **Si se cambia la configuración para un menú, es recomendable hacerlo lo primero, antes de situar las opciones.**

MENUCONFIG varexpression, varexpression, varexpression

Si se omite el cuarto parámetro, `MENUCONFIG x,y,d` equivale a `MENUCONFIG x,y,d,1`.

MENUCONFIG varexpression, varexpression

Si se omiten el tercer parámetro y cuarto parámetros, `MENUCONFIG x,y` equivale a `MENUCONFIG x,y,0,1`.

CHAR varexpression

Imprime el carácter indicando con su número correspondiente.

REPCHAR expression, expression

Imprime el carácter indicado en el primer parámetro tantas veces como número se indique en el segundo parámetro. Ambos valores tienen un tamaño de 1 byte, es decir, val del 0 al 255. Además, si el número de veces es cero, el carácter se repetirá 256 veces en lugar de ninguna.

TAB expression

Imprime tantos espacios como los indicados en el parámetro. Es el equivalente a `REPCHAR 32, expression`, pero ocupa menos memoria.

PRINT varexpression

Imprime el valor numérico indicado.

PAGEPAUSE expression

Controla si al rellenar por completo el área de texto actual, debe solicitar continuar al jugador, presentando un icono animado de espera (parámetro <> 0) ó hace un desplazamiento hacia arriba del texto y sigue imprimiendo (parámetro = 0).

INK varexpression

Define el valor del color de los caracteres (tinta). Valores de 0-7, correspondientes a los colores del Spectrum.

PAPER varexpression

Define el valor del color del fondo (papel). Valores de 0-7, correspondientes a los colores del Spectrum.

BORDER varexpression

Define el color del borde, valores 0-7.

BRIGHT varexpression

Activa o desactiva el brillo (0 desactivado, 1 activado).

FLASH varexpression

Activa o desactiva el parpadeo (0 desactivado, 1 activado).

NEWLINE expression

Imprime tantos saltos de línea como el número indicado en el parámetro (0 es 256)

NEWLINE

Equivalente a `NEWLINE 1`.

BACKSPACE expression

Retrasa el cursor de impresión tantas posiciones como las indicadas en el parámetro (0 es 256), eliminado el carácter en la nueva posición y sustituyéndolo por un espacio. El tamaño de carácter que se usará es el del espacio (número 32). Hay que tener en cuenta que si se usan carácteres de distintos tamaños al del espacio, esta operación no funcionará bien.

BACKSPACE

Equivalente a `BACKSPACE 1`.

SFX varexpression

Si se ha cargado un fichero de efectos de sonido, reproduce el efecto indicado. Si no se ha cargado dicho fichero, el comando es ignorado.

PICTURE varexpression

Carga en el buffer la imagen indicada como parámetro. Por ejemplo, si se indica 3, cargará el fichero `003.CSC`. La imagen no se muestra, lo que permite controlar cuándo se realiza la carga del fichero.

DISPLAY varexpression

Muestra el contenido actual del buffer en pantalla. El parámetro indica si se muestra o no la imagen, con un 0 no se muestra, y con un valor distinto de cero, sí. Se muestran tantas líneas como se hayan definido en la imagen correspondiente y el contenido de la pantalla será sobrescrito.

BLIT varexpression, varexpression, varexpression, varexpression AT varexpression, varexpression

Copia una parte de la imagen cargada en el buffer a la pantalla. Definimos con los parámetros un rectángulo dentro del buffer que se copiará en la pantalla a partir de la posición indicada.

Los parámetros, por órden, son:

- Columna origen del rectángulo a copiar desde el buffer.
- Fila origen del rectángulo a copiar desde el buffer.
- Ancho del rectángulo a copiar desde el buffer.
- Alto del rectángulo a copiar desde el buffer.
- Columna destino en pantalla.
- Fila destino en pantalla.

La unidad de medida empleada en todos los parámetros es el carácter 8x8 y sólo los dos últimos aceptan variables.

WAIT expression

Realiza una pausa. El parámetro es el número de "fotogramas" (50 por segundo) a esperar, y es un número de 16 bits.

PAUSE expression

Igual que `WAIT`, pero con la salvedad de que el jugador puede abortar la pausa con la pulsación de la tecla de confirmación.

TYPERATE expression

Indica la pausa que debe haber entre la impresión de cada carácter. Mínimo 0, máximo 65535.

MARGINS expression, expression, expression, expression

Define el área de pantalla donde se escribirá el texto. Los parámetros, por órden, son:

- Columna inicial.
- Fila inicial.
- Ancho (en carácteres).
- Alto (en carácteres).

Los tamaños y posiciones siempre se definen como si fuesen caracteres 8x8.

FADEOUT expression, expression, expression, expression

Hace un fundido en negro en el área de pantalla definida por los parámetros que, por órden, son:

- Columna inicial.
- Fila inicial.
- Ancho (en carácteres).
- Alto (en carácteres).

Los tamaños y posiciones siempre se definen como si fuesen caracteres 8x8.

AT varexpression, varexpression

Sitúa el cursor en una posición dada, relativa al área definida por el comando `MARGINS`. Los parámetros, por orden, son:

- Columna relativa al origen del área de texto.
- Fila relativa al origen del área de texto.

Las posiciones se asumen en tamaño de carácter 8x8.

FILLATTR varexpression, varexpression, varexpression, varexpression, varexpression

Rellenamos en pantalla un rectángulo con un valor de atributos determinado. Los píxeles no se alteran. Los parámetros, por órden, son:

- Columna origen del rectángulo a rellenar.
- Fila origen del rectángulo a rellenar.
- Ancho del rectángulo a rellenar.
- Alto del rectángulo a rellenar.
- Valor de atributos en formato de pantalla de Spectrum, es decir, un byte con bits en formato FBPPPIII (F = Flash, B = Brillo, P = Papel, I = Tinta).

PUTATTR varexpression, varexpression AT varexpression, varexpression

Ponemos los atributos de un carácter 8x8 en pantalla con unos valores determinados. Los píxeles no se alteran. Los parámetros, por órden, son:

- Valor de atributos en formato de pantalla de Spectrum, es decir, un byte con bits en formato FBPPPIII (F = Flash, B = Brillo, P = Papel, I = Tinta).
- Máscara a aplicar sobre los atributos ya existentes en pantalla. Si el bit es cero, conservamos el de la pantalla, y si es uno usamos el nuevo valor.
- Columna del carácter 8x8.
- Fila del carácter 8x8.

PUTATTR varexpression AT varexpression, varexpression

Equivalente a `PUTATTR varexpression, 0xFF AT varexpression, varexpression`

GETATTR (varexpression, varexpression)

Función que devuelve el valor de atributo de un carácter 8x8 en pantalla, en formato de pantalla de Spectrum, es decir, un byte con bits en formato FBPPPIII (F = Flash, B = Brillo, P = Papel, I = Tinta).

Los parámetros, por órden, son:

- Columna del carácter 8x8.
- Fila del carácter 8x8.

ATTRVAL (expression COMMA expression COMMA expression COMMA expression)

Función que devuelve el valor de atributos en formato de pantalla de Spectrum, es decir, un byte con bits en formato FBPPPIII (F = Flash, B = Brillo, P = Papel, I = Tinta), utilizado en los comandos `PUTATTR` y `FILLATTR`.

Los parámetros, por órden, son:

- Color de tinta (0 a 7).
- Color de fondo (0 a 7).
- Uso de brillo (0 ó 1).
- Uso de parpadeo (0 ó 1).

ATTRMASK (expression COMMA expression COMMA expression COMMA expression)

Función que devuelve un valor de máscara, utilizado en los comandos `PUTATTR` y `FILLATTR`.

Los parámetros, por órden, son:

- Máscara de tinta. Si el valor es cero, conservamos el valor de la pantalla, y si es uno usamos el nuevo valor.
- Máscara de fondo. Si el valor es cero, conservamos el valor de la pantalla, y si es uno usamos el nuevo valor.
- Máscara de brillo. Si el valor es cero, conservamos el valor de la pantalla, y si es uno usamos el nuevo valor.
- Máscara de parpadeo. Si el valor es cero, conservamos el valor de la pantalla, y si es uno usamos el nuevo valor.

RANDOM(expression)

Función que devuelve un número aleatorio entre 0 y el valor indicado en **expression**. El máximo es 255.

RANDOM()

Función que devuelve un número aleatorio entre 0 y 255. Es el equivalente a `RANDOM(255)`.

RANDOM(expression, expression)

Función que devuelve un número aleatorio entre el valor indicado en el primer parámetro y el segundo, ambos inclusive.

INKEY(expression)

Función que devuelve el código de la tecla pulsada. Si el parámetro es cero, se espera hasta que se pulse una tecla. Si es distinto de cero, devuelve el código de la tecla pulsada en ese momento, y cero si no hay ninguna

tecla pulsada. **Sólo responde a las pulsaciones del teclado, no del joystick**

INKEY()

Función que espera hasta que se pulse una tecla y devuelve el código de la tecla pulsada. Es equivalente a `INKEY(0)`

KEMPSTON()

Función que devuelve las teclas pulsadas en un joystick kempston conectado. Los bits siguientes estarán a uno si el botón correspondiente está pulsado:

- Bit 0 = Derecha.
- Bit 1 = Izquierda.
- Bit 2 = Abajo.
- Bit 3 = Arriba.
- Bit 4 = Disparo.

MIN(varexpression,varexpression)

Función que acota el valor del primer parámetro al mínimo indicado por el segundo. Es decir, si el parámetro 1 es menor que el parámetro 2, se devuelve el parámetro 2, en otro caso se devuelve el 1.

MAX(varexpression,varexpression)

Función que acota el valor del primer parámetro al máximo indicado por el segundo. Es decir, si el parámetro 1 es mayor que el parámetro 2, se devuelve el parámetro 2, en otro caso se devuelve el 1.

YPOS()

Función que devuelve la fila actual en la que se encuentra el cursor en coordenadas 8x8.

XPOS()

Función que devuelve la columna actual en la que se encuentra el cursor en coordenadas 8x8 (Debido a la naturaleza de la fuente de ancho variable, el valor devuelto será la columna 8x8 donde esté actualmente el cursor).

WINDOW expression

Se cambia a la "ventana" indicada por el parámetro y se soporta hasta 8 ventanas (desde 0 a 7). Por defecto, la ventana inicial es la 0. Las "ventanas" de CYD son un área de pantalla definida por su posición de origen, su ancho, su alto, la posición del cursor y los atributos actuales. Esto permite tener diferentes áreas de texto independientes en pantalla simultáneamente sin tener que estar cambiando márgenes y la posición del cursor continuamente para pasar de un área a otra. Ten en cuenta que si se solapan ventanas el contenido de una ventana no se conserva si se sobreescribe con en otra ventana.

CHARSET expression

Cambia el juego de carácteres empleado al imprimir textos. Con el parámetro a cero, se usa el juego inferior (caracteres del 0 al 127), y con un valor distinto de cero, se usa el juego superior (caracteres del 128 al 255).

Recuerda que los caracteres del 0 al 32 y del 127 al 143 no son imprimibles. Esta instrucción no afecta a REPCHAR y CHAR.

RANDOMIZE

Inicializa el generador de números aleatorios. La generación de números aleatorios no es realmente "aleatoria" y esto puede ocasionar que el generador devuelva siempre los mismos resultados si se usa en un emulador, por lo que se necesita alguna fuente de aleatoriedad o entropía. Lo que hace este comando es inicializar el generador usando el número de "frames" o "fotogramas" transcurridos, con lo que si se ejecuta en respuesta a algún evento arbitrario, como la pulsación de una tecla, garantizamos la aleatoriedad.

TRACK varexpression

Carga en memoria el fichero de Vortex Tracker como parámetro. Por ejemplo, si se indica 3, cargará la pista de música del fichero `003.PT3` (o `003.mus` en caso de usar WyzTracker). Si existiese una pista cargada previamente, la sobrescribirá.

PLAY varexpression

Si el parámetro es distinto de cero y la música está desactivada, reproduce la pista musical cargada. Si está en ese momento reproduciendo, y se pasa 0 como parámetro, para la reproducción.

LOOP varexpression

Establece si al acabar la pista musical cargada en ese momento, se repite de nuevo o no. Un valor 0 significa falso y distinto de cero, verdadero. Este comando no funciona con WyzTracker.

RAMSAVE varID, expression

Copia desde la variable indicada en el primer parámetro, tantas variables como las indicadas en el segundo parámetro (si es cero, se considera como 256) a un almacenamiento temporal. Antes de realizar la copia, el almacén temporal se rellena con ceros, con lo que las variables no copiadas tendrán este valor en el almacén temporal.

RAMSAVE varID

Es el equivalente a `RAMSAVE varId, 0`, es decir, copia todas las variables desde el primer parámetro hasta el final al almacén temporal.

RAMSAVE

Es el equivalente a `RAMSAVE 0, 0`, es decir, copia todas las variables al almacén temporal.

RAMLOAD varID, expression

Es la operación inversa a `RAMSAVE`. Copia desde el almacenamiento temporal a las variables, desde la variable indicada en el primer parámetro y tantas variables como las indicadas en el segundo (si es cero, se considera como 256).

RAMLOAD varID

Es el equivalente a `RAMLOAD varId, 0`, es decir, copia todas las variables desde el primer parámetro hasta el final desde el almacén temporal.

RAMLOAD

Es el equivalente a `RAMLOAD 0, 0`, es decir, copia todas las variables desde el almacén temporal.

SAVE varexpression, varId, expression

Salva una serie de variables en cinta o disco. El primer parámetro indica el número de fichero a guardar, el segundo parámetro es la variable inicial del rango de variables que se guardará el fichero, y el tercero en número de variables a guardar en ese rango (si es cero, se considera como 256). Cualquier cosa que haya en el almacén temporal que se haya cargado con `RAMSAVE` será eliminado, ya que se usará para almacenar el fichero de forma temporal.

- En el caso del disco, se guardará el fichero con el número indicado de tres dígitos y extensión `.SAV`. P.ej. el fichero a cargar con 16 sería el `016.SAV`. Si el fichero ya existiese, se sobreescribirá.
- En el caso de la cinta, se guardará en ésta. El sistema empezará inmediatamente a guardar, con lo que el autor es el responsable de indicar al usuario que se va a grabar y que tiene que poner la unidad de casette (o equivalente) a grabar. El proceso puede interrumpirse con la tecla **BREAK**.

El resultado de la operación se puede consultar con la función `SAVERESULT()`.

SAVE varexpression, varId

Es el equivalente a `SAVE varexpression, varId, 0`, es decir, guarda todas las variables desde el segundo parámetro hasta el final.

SAVE varexpression

Es el equivalente a `SAVE varexpression, 0, 0`, es decir, guarda todas las variables

LOAD varexpression

Carga una serie de variables desde cinta o disco. El parámetro indica el número de fichero a cargar, y se cargará el rango de variables que se haya indicado al salvar el fichero. Cualquier cosa que haya en el almacén temporal que se haya cargado con `RAMSAVE` será eliminado, ya que se usará para almacenar el fichero de forma temporal.

- En el caso del disco, se cargará el fichero con el número indicado de tres dígitos y extensión `.SAV`. P.ej. el fichero a cargar con 16 sería el `016.SAV`.
- En el caso de la cinta, se cargará el primer bloque sin cabecera que encuentre, con lo que es necesario que el usuario avance la cinta a la posición correcta. El sistema empezará inmediatamente a cargar, con lo que el autor es el responsable de indicar al usuario que se va a cargar y que tiene que poner la unidad de casette (o equivalente) a reproducir. El proceso puede interrumpirse con la tecla **BREAK**.

El resultado de la carga se puede consultar con la función `SAVERESULT()`.

SAVERESULT()

Función que devuelve el resultado de la última operación de `SAVE` o `LOAD`. Los valores posibles son los siguientes:

0. Resultado correcto.
1. Error al cargar/guardar en cinta/disco. En caso de cinta, operación interrumpida con **BREAK**.
2. La partida grabada no corresponde al juego actual.
3. El slot seleccionado de la partida grabada no corresponde al que se ha solicitado.
4. Error de chequeo del fichero de partida grabada.

Con cualquier otro valor, el resultado está indefinido.

ISDISK()

Función que devuelve 1 si el intérprete es la versión de Plus3 y 0 en caso contrario.

LASTPOS(ID)

Función que devuelve la última posición accessible del array dado. A todos los efectos, el tamaño del array menos 1.

Imágenes

El motor soporta un máximo de 256 imágenes, aparte de lo que quepa en el disco o la memoria, y deben estar nombradas con un número de 3 dígitos, que corresponderá al número de imagen que se invocará desde el programa. Por ejemplo, la imagen 0 debería llamarse `000.scr`, la imagen número 1 `001.scr`, y así hasta 255. Las imágenes deben estar en el formato de pantalla de Spectrum.

Para mostrar imágenes, el compilador comprime los ficheros en formato SCR de la pantalla de Spectrum en ficheros comprimidos con extensión `csc`. El proceso de compresión puede ser largo, así que el compilador detecta si por cada fichero `scr`, existe un fichero equivalente con extensión `csc` más reciente en el directorio de imágenes. Si es el caso, entonces se salta este fichero, y si no, lo vuelve a comprimir, generando de nuevo el fichero `csc`.

Se puede configurar el número de líneas horizontales de la imagen a mostrar usando el parámetro correspondiente con el compilador para reducir aún más el tamaño. Además, si detecta que la mitad derecha de la misma está espejada con la izquierda, descarta ésta para reducir aún mas el tamaño, aunque podemos forzar este comportamiento con cualquier imagen. Para ello hay que crear un fichero llamado `images.json` dentro del directorio de imágenes.

La estructura del fichero `images.json` es la siguiente:

```
[
  {
    "id": 0,
    "num_lines": 192,
    "force_mirror": false
  },
  {
    "id": 1,
```

```
    "num_lines": 64,
    "force_mirror": true
  }
]
```

Es un listado de registros con los siguientes campos:

- `id` : Es el número de imágen correspondiente a esta entrada. En el ejemplo, `"id":0` corresponde a la imagen `000.scr` e `"id":1` corresponde a la imagen `001.scr`.
- `num_lines` : Indica el número de líneas a incluir en el fichero comprimido. El mínimo es 1 y el máximo (pantalla completa) es 192.
- `force_mirror`: Con valor `true`, forzamos a que la imagen se considere simétrica. Esto hará que se descarte la mitad derecha de la imagen y que cuando se descomprima em memoria, se dibuje en su lugar la mitad izquierda reflejada. En otro caso, este campo debe tener el valor `false`.

Debemos indicar en el JSON tantos registros como imágenes queramos definir el comportamiento. Con aquellas imágenes que no estén definidas en el fichero `images.json`, se tratarán con el comportamiento normal, el cual es usar el número de líneas definidas con el parámetro `-nl` o 192 por defecto, y usar el espejado sólo en aquellas imágenes detectadas como simétricas.

Hay dos comandos necesarios para mostrar una imagen, el comando `PICTURE n` cargará en un 'buffer' la imagen número n. Es decir, si hacemos `PICTURE 1`, cargará el fichero `001.CSC` en el 'buffer'. Cualquier operación con una imagen requiere su carga previa en dicho 'buffer'. Esto es útil para controlar cuándo se debe cargar la imagen, ya que supondrá espera para la carga desde el disco , además de cierto tiempo para realizar la descompresión de la imagen tanto en cinta como en disco. Por eso es recomendable realizar este comando en un momento adecuado, por ejemplo hacerlo al iniciar un capítulo. Si intentamos cargar una imagen que no exista, recibiremos un error 1, y si se intenta cargar una imagen cuyo fichero no existe en disco, se generará el error de disco 23.

Para mostrar una imagen cargada en el buffer, usamos `DISPLAY n`, donde n tiene que ser cero para ejecutarse. La imagen que se mostrará será la última cargada en el buffer (si existe). La imagen comienza a pintarse desde la esquina superior izquierda de la pantalla y se dibujan tantas líneas como las indicadas al comprimir el fichero y se sobrescribe todo lo que hubiese en pantalla hasta el momento.

También existe un método alternativo para mostrar imágenes, que es el comando `BLIT`, el cual nos permite copiar un fragmento en lugar de una imagen completa en pantalla, además de poder indicar su posición en la misma. El formato del comando es `BLIT xo, yo, ancho, alto AT xd, yd`, y tienes más detalles sobre ella en su sección correspondiente. `BLIT` es más versátil que `DISPLAY`, pero también es más lento, y también requiere que haya una imagen cargada en el 'buffer' de imagen con `PICTURE` previamente y sobreescribirá cualquier cosa que hubiese en pantalla en la posición donde se dibuje.

Efectos de sonido

Añadir efectos de sonido con el beeper es muy sencillo. Para ello tenemos que crear un banco de efectos con la utilidad BeepFx. Debemos exportar el fichero de efectos como un fichero en ensamblador usando la opción `File->Compile` del menú superior. En la ventana flotante que aparece, debemos asegurarnos de que tenemos seleccionado `Assembly` e `Include Player Code`, el resto de opciones son irrelevantes, ya que el compilador modificará el fichero fuente para incrustarlo en el intérprete.

Para invocar un efecto, usamos el comando `SFX n`, siendo n el número del efecto a reproducir. Si se llama a este comando sin que exista un fichero de efectos cargado, será ignorado y seguirá la ejecución.

No está contemplado llamar a un número de efecto que no exista en el fichero incluido, así que es responsabilidad del autor tener cuidado de esta circunstancia, ya que el comportamiento en ese caso es impredecible.

Melodías Vortex Tracker

El motor `CYD` permite reproducir módulos de música creados con Vortex Tracker en formato `PT3`. Su funcionamiento replica el mecanismo de carga de imágenes, es decir, los módulos deben nombrarse con tres dígitos que representan el número de pista que el intérprete cargará con el comando `TRACK`. Por ejemplo, si el intérprete encuentra el comando `TRACK 3`, entonces buscará la pista del fichero `003.PT3` y cargará en memoria el módulo para su reproducción. Y de la misma manera, el máximo número de módulos que se pueden cargar son 256 (de 0 a 255) y no se permite que el módulo sea de más de 16 Kilobytes para cinta y 8 Kilobytes en caso de usar Plus3.

Una vez cargado un módulo, se podrá reproducir con el comando `PLAY`. Como se indica en la referencia de comandos, si el parámetro es distinto de cero, se reproducirá el módulo; y si es igual a cero, se detendrá la reproducción.

Cuando se llegue al final del módulo, la reproducción se detendrá automáticamente, pero podemos cambiar este comportamiento con el comando `LOOP`. De nuevo, según se indica en la referencia, si el parámetro es igual a cero, al llegar al final se detendrá la reproducción, como se ha indicado antes. Pero si el parámetro es distinto de cero, el módulo volverá a reproducirse desde el principio.

Melodías WyzTracker

`CYD` también permite reproducir módulos de música creados con WyzTracker v2.0. El funcionamiento es similar al mecanismo de Melodías de Vortex Tracker y las imágenes; los nombres de los ficheros de los módulos tienen que ser números de 0 a 255 y con tres dígitos, de tal manera que con `TRACK 3`, cargaríamos el módulo `003.mus` correspondiente.

Una vez cargado un módulo, se podrá reproducir con el comando `PLAY`. Como se indica en la referencia de comandos, si el parámetro es distinto de cero, se reproducirá el módulo; y si es igual a cero, se detendrá la reproducción.

A diferencia de con los módulos de Vortex Tracker, no se soporta el comando `LOOP` de forma completa, ya que son los propios módulos los que definen si éstos se reproducirán en bucle o no.

Debido a las peculiaridades del formato de WyzTracker, hay que tener en cuenta una serie de consideraciones que afectarán al uso del motor con este reproductor de música:

- Todos los módulos deben compartir los mismos instrumentos. El fichero de instrumentos se debe llamar `instruments.asm` y debe depositarse junto a los ficheros de módulos en el directorio `TRACKS`.
- *Se reservará la totalidad del banco 1_ de la memoria del Spectrum, en el cual se almacenará el código del reproductor, los instrumentos y las melodías. Esto supone que se dispondrán de 16 Kb menos para la versión de cinta, y 8 Kb menos para la versión de Plus3.

- Las diferentes melodías serán comprimidas para ahorrar espacio y cada vez que se cargue una de ellas, será descomprimida en el espacio restante del banco 1. El compilador generará un error si alguna de las melodías incluidas no cabe en ese espacio restante.
- No se soportan efectos de sonido.

Cómo generar una aventura

Lo primero es generar un guion de la aventura mediante cualquier editor de textos empleando la sintaxis arriba descrita. Es MUY recomendable hacer el guion de la misma antes de ponerse a programar la lógica ya que conviene tener el texto perfilado antes para tener una compresión adecuada (más detalles más adelante).

Es importante que la codificación del fichero sea UTF-8, pero hay que tener en cuenta que caractéres por encima de 128 no se imprimirán bien y sólo se admiten los caracteres propios del castellano, indicados en la sección Juego de Caracteres, que serán convertidos a los códigos allí indicados.

Una vez tenemos la aventura, usamos el compilador `CYDC` para generar un fichero DSK o TAP. EL compilador busca las mejores abreviaturas para comprimir el texto lo máximo posible. El proceso puede ser muy largo dependiendo del tamaño de la aventura. Por eso es importante tener la aventura perfilada antes, para realizar este proceso al principio. La compilación la realizaremos con el parámetro `-T` de tal manera que con `-T abreviaturas.json`, por ejemplo, exportaremos las abreviaturas encontradas al fichero *abreviaturas.json*.

A partir de este momento, si ejecutamos el compilador con el parámetro `-t abreviaturas.json`, éste no realizará la búsqueda de abreviaturas y usará las que ya habíamos encontrado antes, con lo que la compilación será casi instantánea. Cuando ya consideremos que la aventura está terminada, podremos volver a realizar una nueva búsqueda de abreviaturas para intentar conseguir algo más de compresión.

Para añadir efectos de sonido, imágenes y melodías, consulta las secciones correspondientes.

El proceso es bastante simple, pero tiene algunos pasos dependientes, con lo que se recomienda usar ficheros BAT (Windows) o guiones de shell (Linux, Unix) o la utilidad Make (o similar) para acelerar el desarrollo.

Para facilitar las cosas, se incluye un programa llamado `make_adventure.py` que realizará todas estas tareas por nosotros, además de los correspondientes scripts `make_adv.cmd` para Windows y `make_adv.sh` para UNIX para ejecutarlo.

El programa `make_adventure.py` buscará y comprimirá automáticamente los ficheros SCR que se atengan al formato de nombre establecido (número de 0 a 255 con 3 dígitos) dentro del directorio `.\IMAGES`. Lo mismo hará con los módulos que haya dentro del directorio `.\TRACKS` que cumplan el formato de nombre. Luego compilará el fichero `test.txt` y generará el fichero `tokens.json` con las abreviaturas, si no existiese previamente. Si se desea que se vuelva a generar el fichero de abreviaturas (es recomendable hacerlo cuando se nos esté agotando la memoria o estemos finalizando la aventura), simplemente borrándolo hará que el script indique al compilador lo genere de nuevo. Además buscará de forma automática si existe un fichero de efectos de sonido llamado `SFX.ASM` que debe generarse con **BeepFX**, y si existiese un fichero JSON con el juego de caracteres llamado `charset.json`, también lo utilizará.

Este programa necesita los directorios `dist` y `tools` con su contenido para realizar el proceso. Ahora se detallan las peculiaridades en cada sistema operativo:

Windows

Como ejemplo se ha incluido el fichero `make_adv.cmd` en la raíz del repositorio, que compilará la aventura de muestra incluida en el fichero `test.cyd`.

Puedes usarlo como base para crear tu propia aventura de forma sencilla. Se puede personalizar el comportamiento modificando en la cabecera del script algunas variables:

```
REM ---- Configuration variables ----------

REM Name of the game
SET GAME=test
REM This name will be used as:
REM   - The file to compile will be test.cyd with this example
REM   - The name of the TAP file or +3 disk image

REM Target for the compiler (48k, 128k for TAP, plus3 for DSK)
SET TARGET=48k

REM Number of lines used on SCR files at compressing
SET IMGLINES=192

REM Loading screen
SET LOAD_SCR="LOAD.scr"

REM Parameters for compiler
SET CYDC_EXTRA_PARAMS=

REM Run emulator after compilation (none/internal/default)
REM     -None: Do nothing
REM     -internal: run the compiled file with Zesarux (must be on the directory
.\tools\zesarux\)
REM     -default: run the compiled file with the program set on Windows to run its
extension
SET RUN_EMULATOR=none

REM Backup CYD file after compilation (yes/no) on ./BACKUP directory.
SET BACKUP_CYD=no

REM ---------------------------------------
```

- La variable `GAME` será el nombre del fichero txt que se compilará y el nombre del fichero DSK o TAP resultante.
- La variable `TARGET` es el sistema y formato de salida, con estas posibles opciones: -- 48k: Genera un fichero TAP para Spectrum 48K, sin soporte de música AY. -- 128k: Genera un fichero TAP para Spectrum 128K. -- plus3: Genera un fichero DSK para Spectrum +3, con mayor capacidad y carga dinámica de recursos.
- La variable `IMGLINES` es el número de líneas horizontales de los ficheros de imagen que se comprimirán. Por defecto es 192 (la pantalla completa del Spectrum)
- La variable `LOAD_SCR` es la ruta a un fichero de tipo SCR (pantalla de Spectrum) con la pantalla que se usará durante la carga.
- La variable `CYDC_EXTRA_PARAMS` se usa para añadir parámetros extra en la llamada al compilador cydc.

- La variable `RUN_EMULATOR` indica si queremos que se ejecute el programa compilado bajo un emulador con los siguientes valores posibles: -- none: Si no queremos que haga esto. -- internal: Ejecuta el fichero compilado bajo Zesarux que debe estar dentro del directorio `.\tools\zesarux\`. -- default: Si la extensión del fichero está asociada bajo Windows con otro emulador, lo ejecutará con éste.
- La variable `BACKUP_CYD` con el valor `yes` hace una copia de seguridad del fichero actual dentro del directorio `.\BACKUP`. Cada copia añade al nombre del fichero la fecha en la cual se creó.

El guión producirá un fichero DSK o TAP (dependiendo del formato seleccionado en `TARGET`) que podrás ejecutar con tu emulador favorito. Pero si deseas acelerar más el trabajo, si te descargas Zesarux y lo instalas en de la carpeta `.\tools\zesarux`, tras la compilación se ejecutará automáticamente con las opciones adecuadas.

Linux, BSDs

Como ejemplo se ha incluido el fichero `make_adv.sh` en la raíz del repositorio, que compilará la aventura de muestra incluida en el fichero `test.cyd`.

Puedes usarlo como base para crear tu propia aventura de forma sencilla. Se puede personalizar el comportamiento modificando en la cabecera del script algunas variables:

```
# ---- Configuration variables ----------
# Name of the game
GAME="test"
# This name will be used as:
#   - The file to compile will be test.cyd with this example
#   - The name of the TAP file or +3 disk image
#
# Target for the compiler (48k, 128k for TAP, plus3 for DSK)
TARGET="48k"
#
# Number of lines used on SCR files at compressing
IMGLINES="192"
#
# Loading screen
LOAD_SCR="./IMAGES/LOAD.scr"
#
# Parameters for compiler
CYDC_EXTRA_PARAMS=
# ---------------------------------------
```

- La variable `GAME` será el nombre del fichero txt que se compilará y el nombre del fichero DSK o TAP resultante.
- La variable `TARGET` es el sistema y formato de salida, con estas posibles opciones: -- 48k: Genera un fichero TAP para Spectrum 48K, sin soporte de música AY. -- 128k: Genera un fichero TAP para Spectrum 128K. -- plus3: Genera un fichero DSK para Spectrum +3, con mayor capacidad y carga dinámica de recursos.
- La variable `IMGLINES` es el número de líneas horizontales de los ficheros de imagen que se comprimirán. Por defecto es 192 (la pantalla completa del Spectrum)

- La variable `LOAD_SCR` es la ruta a un fichero de tipo SCR (pantalla de Spectrum) con la pantalla que se usará durante la carga.

Ejemplos

Dispones de una serie de ejemplos para comprobar las capacidades del motor y aprender de ellos:

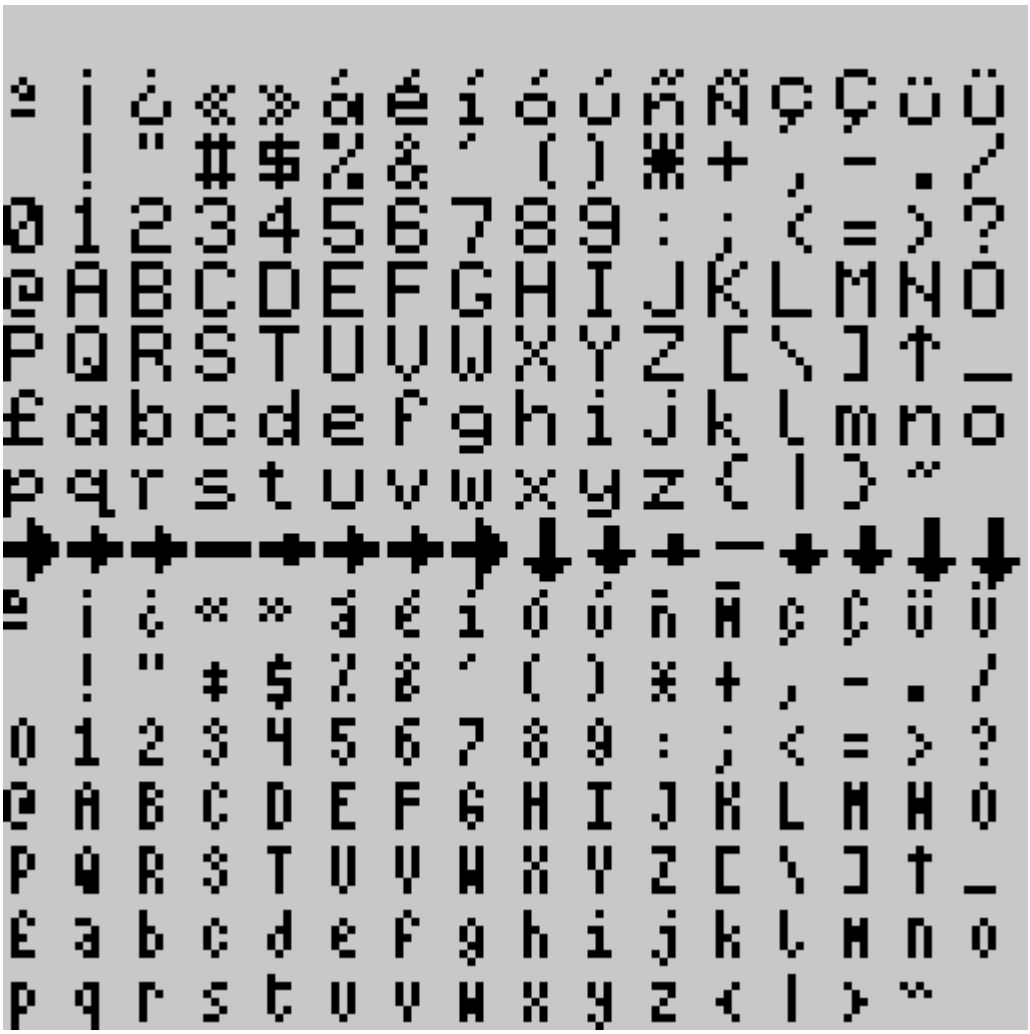
- En la carpeta `examples\test` hay una muestra ampliada del ejemplo incluido en la sección de Sintaxis. Se incluye imágenes de prueba en el directorio `examples\test\IMAGES`, y una canción de prueba dentro de su directorio `examples\test\TRACKS`. Solo funciona en los ordenadores de 128K.
- En la carpeta `examples\ETPA_ejemplo` tienes un ejemplo del comienzo de un libro tipo "Elije Tu Propia Aventura" simple, que resulta algo más avanzado. Se incluyen imágenes de prueba en `examples\ETPA_ejemplo\IMAGES`
- En la carpeta `examples\guess_the_number` hay un ejemplo de uso para menús de múltiples columnas.
- En la carpeta `examples\input_test` se muestra cómo leer desde el teclado y el uso de la indirección para simular arrays.
- En la carpeta `examples\blit` se muestra el uso del comando `BLIT`, con ejemplos progresivamente más avanzados en `examples\blit_island`, `examples\Rocky_Horror_Show` y `examples\CYD_presents`. En la subcarpeta `IMAGES` se incluyen las imágenes de prueba.
- En la carpeta `examples\Golden_Axe_select_character` podrás aprender sobre los comandos `FILLATTR` y `CHOOSE IF CHANGED...`. En la subcarpeta `IMAGES` se incluyen las imágenes de prueba.
- En la carpeta `examples\windows` tienes un ejemplo sencillo que ilustra el uso del comando `WINDOW`.
- En la carpeta `examples\SCUMM_16` hay un ejemplo de un menú tipo SCUMM para hacer una aventura de este tipo. Y en la carpeta `examples\delerict` hay un esqueleto de otra aventura más completa, que incluye lógica de manejo de objetos y localidades.

Se incluye en cada directorio un fichero TAP con el resultado compilado para que poder probarlos en vivo en un emulador.

Para compilarlos por uno mismo, simplemente habría que copiar los ficheros y directorios del ejemplo que quieras probar en el directorio principal de la distribución y ejecutar `make_adv.cmd`. Recuerda borrar (y guardar si fuese el caso) los ficheros que ya existiesen antes.

Juego de caracteres

El motor soporta un juego de 256 caracteres, con 8 píxeles de altura y tamaño variable de ancho de 1 a 8 píxeles. El juego de caracteres por defecto incluido tiene un tamaño 6x8, junto con un juego alternativo de tamaño 4x8 a partir del carácter 144. Éste es el juego de caracteres por defecto, ordenados de izquierda a derecha y de arriba a abajo:



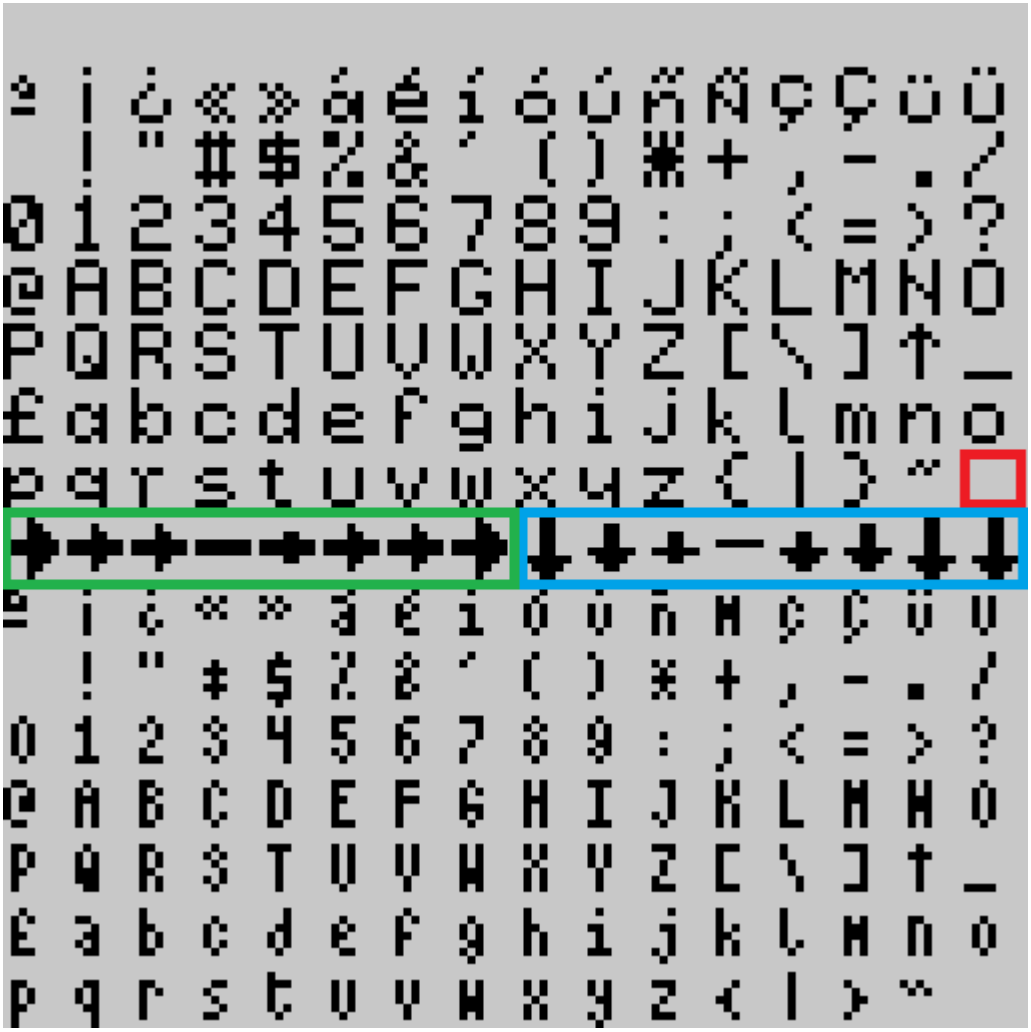
Los carácteres corresponden con el ASCII estándar, excepto los carácteres propios del castellano, que corresponden a las siguientes posiciones para los dos juegos de caracteres:

Carácter	Posición 6x8	Posición 4x8
'ª'	16	144
'¡'	17	145
'¿'	18	146
'«'	19	147
'»'	20	148
'á'	21	149
'é'	22	150
'í'	23	151
'ó'	24	152
'ú'	25	153
'ñ'	26	154
'Ñ'	27	155

Carácter	Posición 6x8	Posición 4x8
'ç'	28	156
'Ç'	29	157
'ü'	30	158
'Ü'	31	159

Los caracteres por encima del valor 127 (empezando desde cero) hasta el 143 (ambos incluidos) son especiales. Son utilizados como iconos en las opciones, es decir, en donde aparece una opción cuando se procesa el comando `OPTION`, y como indicadores de espera con un `WAITKEY` o al cambiar de página si el comando `PAGEPAUSE` está activo.

- El carácter 127 es el carácter usado cuando una opción no está seleccionada en un menú. (En rojo en la captura inferior)
- Los caracteres del 128 al 135 forman el ciclo de animación de una opción seleccionada en un menú. (En verde en la captura inferior)
- Los caracteres del 135 al 143 forman el ciclo de animación del indicador de espera. (En azul en la captura inferior)



El compilador dispone de dos parámetros, `-c` para importar un juego de caracteres nuevo, y `-C` para exportar el juego de caracteres actualmente empleado, por si puede servir de plantilla o realizar personalizaciones.

Este es el formato de importación/exportación del juego de caracteres:

```
[{"Id": 0, "Character": [255, 128, ...], "Width":8}, {"Id": 1, "Character": [0, 1,
...], "Width":6}, ...]
```

Es un JSON con un array de registros de tres campos:

- *Id*: número del carácter de 0 a 255, que no se puede repetir.
- *Character*: un array de 8 números que corresponde con el valor de los bytes del caracter y, por tanto, no puede haber valores mayores de 255. Cada byte corresponde con los pixels de cada línea del carácter.
- *Width*: un array con el ancho en pixels del carácter (los valores no pueden ser menores que 1 ni mayores que 8). Dado que el tamaño de cada línea del carácter del campo anterior es 8, los píxeles que sobren por la derecha serán descartados.

Para facilitar la tarea de creación de un juego de caracteres alternativo, se ha incluido la herramienta CYD Character Set Converter o cyd_chr_conv. Esta herramienta permite convertir fuentes en formato `.chr`, `.ch8`, `.ch6` y `.ch4` creadas con ZxPaintbrush en el formato JSON anteriormente mencionado. Además, en el directorio `assest` de la distribución podrás encontrar el fichero `default_charset.chr` con la fuente por defecto para que puedas editarla y personalizarla con dicho programa.

Códigos de error

La aplicación puede generar errores en tiempo de ejecución. Los errores son de dos tipos, de disco y del motor.

Los errores de motor son, como su nombre indica, los errores propios del motor cuando detecta una situación anómala. Son los siguientes:

- Error 1: El trozo accedido no existe. (Se intenta acceder a un fragmento no existente en el índice)
- Error 2: Se han creado demasiadas opciones, se ha superado el límite de opciones posibles.
- Error 3: No hay opciones disponibles, se ha lanzado un comando `CHOOSE` sin tener antes ninguna `OPTION` declarada.
- Error 4: El fichero con el módulo de música a cargar es demasiado grande, tiene que ser menor que 16Kib.
- Error 5: No hay un módulo de música cargado para reproducir.
- Error 6: Código de instrucción inválido.
- Error 7: Acceso a posición del array fuera del rango.
- Error 8: Opción perdida. Al hacer scroll, las opciones declaradas se desplazan hacia arriba, si una de ellas sale por el límite superior de los márgenes, se genera este error.

Los errores de disco son los errores que pudiesen ocasionarse cuando el motor del juego accede al disco, y corresponden con los errores de +3DOS:

- Error 0: Drive not ready
- Error 1: Disk is write protected
- Error 2: Seek fail

- Error 3: CRC data error
- Error 4: No data
- Error 5: Missing address mark
- Error 6: Unrecognised disk format
- Error 7: Unknown disk error
- Error 8: Disk changed whilst +3DOS was using it
- Error 9: Unsuitable media for drive
- Error 20: Bad filename
- Error 21: Bad parameter
- Error 22: Drive not found
- Error 23: File not found
- Error 24: File already exists
- Error 25: End of file
- Error 26: Disk full
- Error 27: Directory full
- Error 28: Read-only file
- Error 29: File number not open (or open with wrong access)
- Error 30: Access denied (file is in use already)
- Error 31: Cannot rename between drives
- Error 32: Extent missing (which should be there)
- Error 33: Uncached (software error)
- Error 34: File too big (trying to read or write past 8 megabytes)
- Error 35: Disk not bootable (boot sector is not acceptable to DOS BOOT)
- Error 36: Drive in use (trying to re-map or remove a drive with files open)

La aparición de estos errores ocurre cuando se accede al disco, al buscar más trozos de texto, imágenes, etc. Si aparece el error 23 (File not found), suele ser que se haya olvidado de incluir algún fichero necesario en el disco. Otros errores ya suponen algún error de la unidad de disco o del propio disco.

Referencias y agradecimientos

- David Beazley por PLY
- Einar Saukas por el compresor ZX0.
- Mokona por su versión del compresor ZX0 para Python.
- Sylvain Glaize por la versión del descompresor ZX0 para Python.
- DjMorgul por el buscador de abreviaturas, adaptado de Daad Reborn Tokenizer
- Shiru por BeepFx.
- Seasip por mkp3fs de Taptools.
- Sergey.V.Bulba por el reproductor de Vortex Tracker.
- Augusto Ruiz por el reproductor de WyzTracker.
- Al equipo responsable del ensamblador sjasmplus.
- Tranqui69 por el logotipo.
- XimoKom, Javier Fopiani y Fran Kapilla por su inestimable ayuda en las pruebas del motor.
- Pablo Martínez Merino por la ayuda con el testeo en Linux y ejemplos.
- Sergio ThePoPe por meterme el gusanillo del Plus3.
- El_Mesías, Arnau Jess y Javi Ortiz por la difusión.

- Al Club de Aventuras AD CAAD por el apoyo.
- A todos los miembros del grupo de Telegram de `Choose your Destiny`.

Licencias

Este paquete de software está sometido a diferentes licencias dependiendo de sus componentes:

- El compilador está bajo licencia GNU Affero General Public License v3.0.
- El intérprete (la parte ejecutable en la máquina objetivo) se encuentra bajo la siguiente licencia:

```
Choose Your Destiny

Copyright (c) 2025 Sergio Chico (Cronomantic)

Por la presente se concede permiso, libre de cargos, a cualquier persona que
obtenga una copia de este software y de los archivos de documentación asociados
(el "Software"), a utilizar el Software sin restricción, incluyendo sin limitación
los derechos a usar, copiar, modificar, fusionar, publicar, distribuir, y/o vender
copias del Software, y a permitir a las personas a las que se les proporcione el
Software a hacer lo mismo, sujeto a las siguientes condiciones:

- El aviso de copyright anterior y este aviso de permiso se incluirán en todas las
copias o partes sustanciales del Software.

- El aviso del copyright anterior y/o uno de los logotipos del proyecto deben
estar indicados de forma visible tanto en la pantalla de carga y/o dentro de los
propios programas que incluyan este Software, así como en la web de descarga en
caso de copia digital y/o en la portada en caso de copia física.

- EL SOFTWARE SE PROPORCIONA "COMO ESTÁ", SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O
IMPLÍCITA, INCLUYENDO PERO NO LIMITADO A GARANTÍAS DE COMERCIALIZACIÓN, IDONEIDAD
PARA UN PROPÓSITO PARTICULAR E INCUMPLIMIENTO. EN NINGÚN CASO LOS AUTORES O
PROPIETARIOS DE LOS DERECHOS DE AUTOR SERÁN RESPONSABLES DE NINGUNA RECLAMACIÓN,
DAÑOS U OTRAS RESPONSABILIDADES, YA SEA EN UNA ACCIÓN DE CONTRATO, AGRAVIO O
CUALQUIER OTRO MOTIVO, DERIVADAS DE, FUERA DE O EN CONEXIÓN CON EL SOFTWARE O SU
USO U OTRO TIPO DE ACCIONES EN EL SOFTWARE.
```

- PyZX0, está sometido a licencia BSD-3.
- El reproductor de WyzTracker está bajo licencia MIT
- PLY y el reproductor de WyzTracker no tienen licencias específicas, se asume una licencia similar a BSD o MIT.
- BeepFx está bajo licencia WTFPL v.2.

En términos sencillos, significa que si usas este motor para hacer un juego, siempre tienes que indicar en la pantalla de carga o dentro del propio juego que se ha realizado con `CYD` o usar alguno de los logotipos de proyecto (incluidos en el directorio `assets`). También debes hacerlo en la web de descarga si distribuyes el

juego online, y en la caja o recipiente del medio si lo distribuyes de forma física. Aparte de la anterior condición, el juego realizado **SIEMPRE** será de tu propiedad y autoría; puedes venderlo o distribuirlo sin necesitad de publicar el código fuente ni recursos artísticos. Sin embargo, la herramienta está bajo licencia AGPL, lo que significa que si la modificas o haces una versión propia, tienes obligación de publicar el código e indicar que está basado en este proyecto.