ANEXOS

ANEXO I. Sobre el MSX y el MSX-BASIC

La plataforma o sistema elegido para desarrollar el videojuego es el MSX, un sistema de 8Bits y el lenguaje de programación es el MSX-BASIC.

Sobre el MSX.

A inicios de los años 80, el mundo de la microinformática doméstica contaba con una gran cantidad de fabricantes tales como Commodore, Amstrand, Sinclair (Spectrum), Spectravideo, Atari, etc. Cada fabricante diseñaba su propio hardware y software que eran totalmente incompatibles entre ellos.

MSX fue un sistema concebido por Kazunhiko Nishi de ASCII Corp (empresa japonesa) con la vocación de convertirse en un estándar dentro de los microordenadores domésticos de 8 bits. ASCII Corp contó con la colaboración del joven Bill Gates fundador de Microsoft (EEUU) que dotó al estándar de un sistema operativo, el MSX-DOS y un interpretador para programación, el MSX-BASIC.

Gracias a asegurar la compatibilidad a nivel de software y hardware, a la nueva norma se unieron un gran número de empresas como Sony, Philips, Sanyo, Panasonic, Yamaha, Toshiba, y un largo etcétera, por contra no consiguió penetrar en el mercado Inglés, dominado por Sinclair ni el Norteamericano dominado por Commodore, por lo que no acabó de consolidarse a nivel mundial. Años más tarde IBM si que conseguiría unificar el mundo de los microordenadores gracias al PC con la colaboración, otra vez, de Microsoft.

Sobre el MSX-BASIC.

El MSX-BASIC es una versión ampliada del Microfoft Standard Basic versión 4.5, e incluye soportes para gráficos, música y diversos periféricos. (Sony España, (1987). *Aprenda a programar en Basic MSX*, ed. 3, Barcelona, Editorial Sony España).

Arquitectura del MSX.

La arquitectura del sistema MSX en su primera generación está formado un microprocesador de 8 bits, el Z80, como CPU, capaz de direccionar 65536 bytes (64 Kb). Al Z80 o CPU se conecta memoria y otros componentes de hardware según se muestra en la ilustración 1. La memoria del MSX es de dos tipos:

- Memoria ROM (Read Only Memory) de 32 Kb, donde 16 Kb están destinados a almacenar la BIOS y otros 16 Kb recoge el interprete del MSX-BASIC, programa que pasa a código máquina las instrucciones que entramos por consola.

- Memoria RAM (Random Access Memory) o memoria de trabajo, esta memoria es la que almacena los programas que se cargan. La RAM pierde los datos que posee al ser apagada la computadora. La norma establece un mínimo de 8Kb pero la mayoría de marcas dotaron a sus máquinas con 64 Kb.

El resto de componentes de hardware que gestina el Z80 o CPU son:

- Procesador de video o VDP (Video Display Processor): Chip TMS9918, encargado de procesar todo lo relativo a imagen. De este procesador cuelga la VRAM (memoria RAM de 16Kb destinada sólo a la parte gráfica). El Z80 no puede gestionar la VRAM directamente, lo hace a través del VDP.

- Generador programable de sonido o PSG (Programmable Sound Generator): Chip AY-3-8910A que se encarga de gestionar la parte sonora. Este chip también gestiona la comunicación con el mouse, con el joystick y la conexión de entrada de cassette.

- Interfaz programable para periféricos o PPI (Programmable Peripheral Interface): Chip Intel 8255 responsable de la lectura del teclado, salida de impresora así como otras conexiones como el modem vía RS-232C.

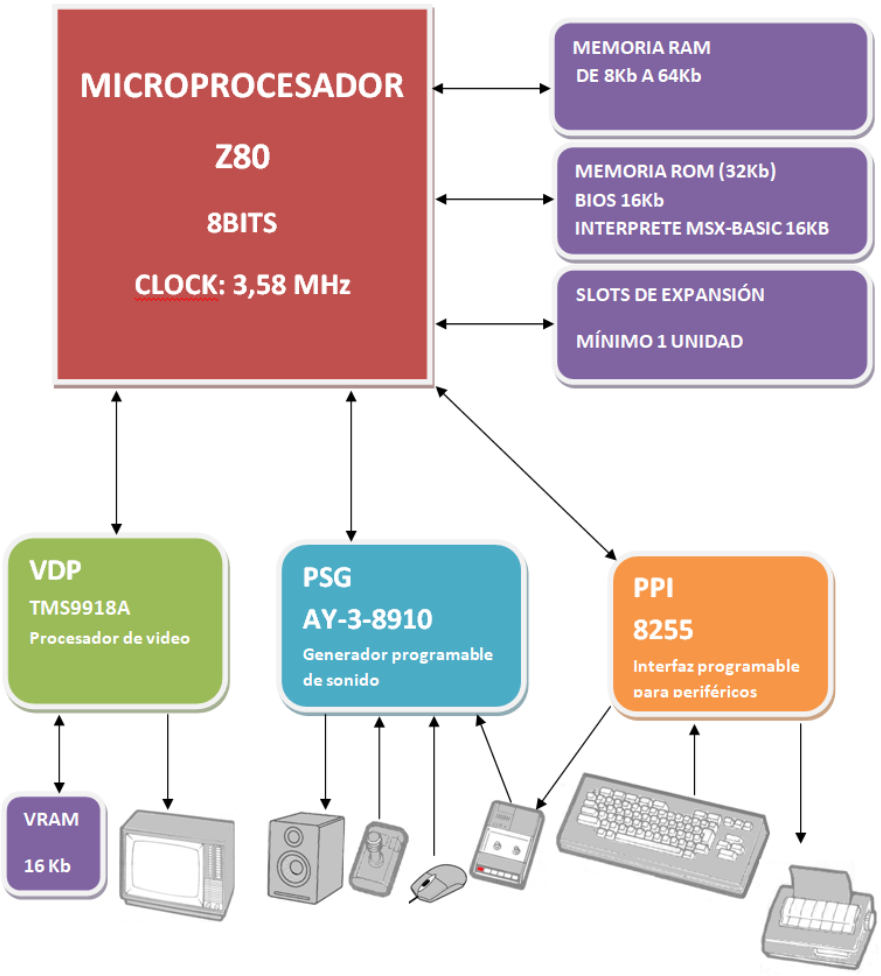


Ilustración 1. Arquitectura hardware MSX

ANEXO II. Preparación del entorno.

Para el desarrollo del videojuego no se utilizará, claro está, un ordenador MSX físico, por lo que se utilizará un programa emulador en un PC actual.

El emulador elegido es el BlueMSX. Para trabajar la parte gráfica se utilizará el programa Tinysprite para sprites o gráficos.

Todos estos programas son de código y distribución libre tanto para windows y linux o Mac.

	Programa Creador Licencia	Sistema Licencia	Descarga
Emulador	**BlueMSX** Daniel Vik	Descarga e instalación: Windows y linux licencia GPL Online: Windows, linux, mac	Descarga del programa http://bluemsx.msxblue.com/download.html Online https://www.bluemsx.com/
Gráficos Sprites (personajes)	**Tinysprite** Rafael Jannone	Programa online GNU LGPL v3 license.	Online http://msx.jannone.org/tinysprite/tinysprite.html

Tabla 1. Programario entorno trabajo

Instalación BlueMSX.

El emulador podemos instalarlo en nuestro PC o trabajar online si disponemos de una conexión a internet. Para instalar el BlueMsx primero hay que dirigirse a la dirección de descarga indicada en la tabla 14 y descargar en Download según se indica en la ilustración 5.

Ilustración 5. Descarga programa BLUEMSX

Escoger por defecto la versión completa, blueMSX 2.8.2. Full y descargar en Download. En la carpeta de descarga aparecerá el icono del BlueMSX, hacer doble click. Aparecerá el cargador del programa, ejecutar el cargador y confirmar la instalación. Según ilustración 6.

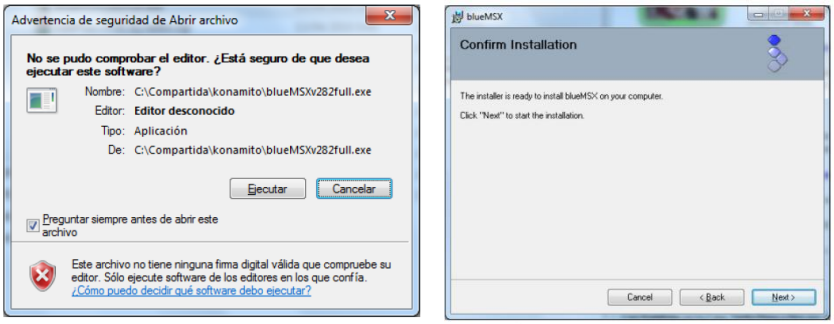


Ilustración 6. Primer paso instalación BLUEMSX

Seleccionar la carpeta donde instalar el programa, por defecto en C/archivos de programas. Presionar Next según ilustración 7.

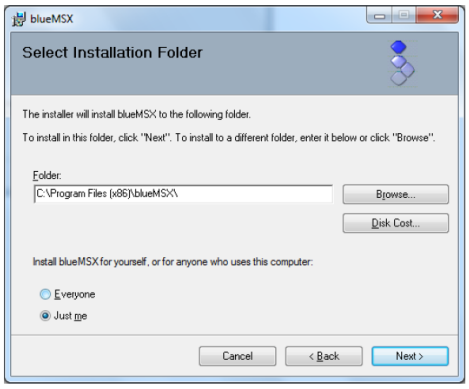


Ilustración 7. Segundo paso instalación BLUEMSX

Volvemos a pulsar Next para la instalación según ilustración 8.

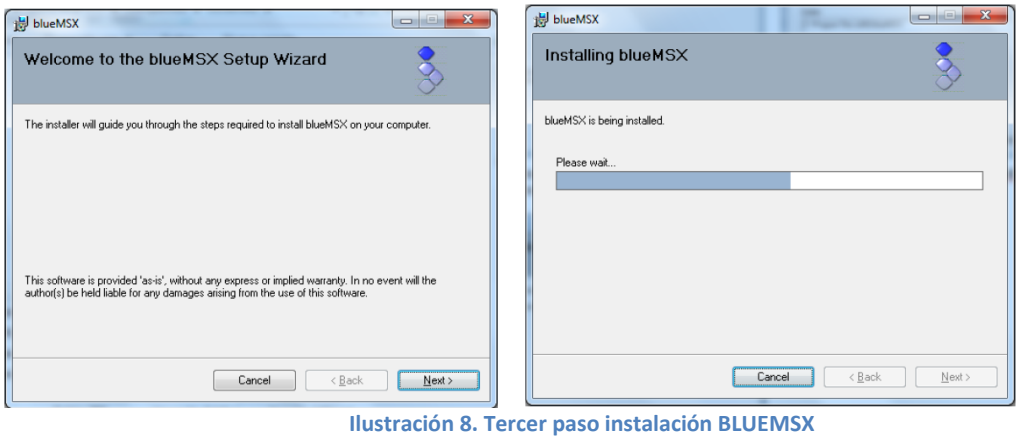


Ilustración 8. Tercer paso instalación BLUEMSX

Una vez finalizada la instalación se puede abrir el emulador según ilustración 9.

Ilustración 9. IDE interfaz BLUEMSX

Instalación TinySprite v 0.7.0

El programa TinySprite no hace falta descarga, se trabaja directamente en online en la dirección mostrada en la tabla 14. El ide del programa se muestra en la ilustración 10.

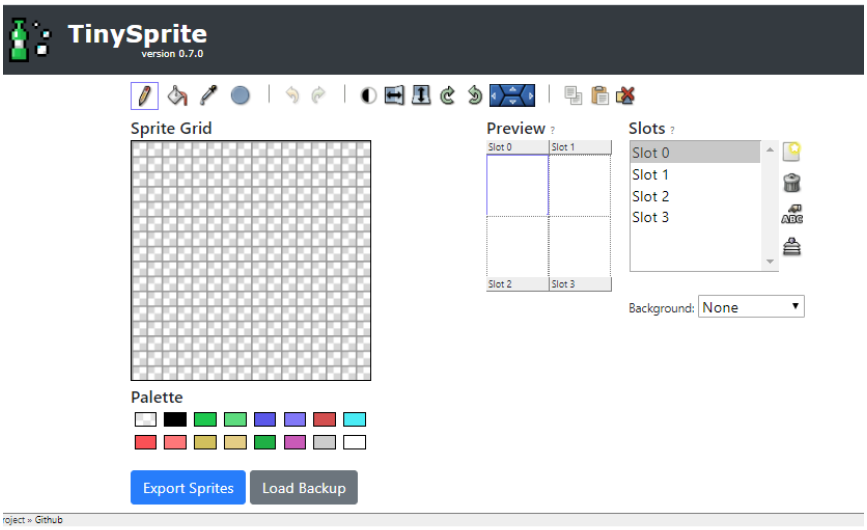


Ilustración 10. IDE o interfaz del TinySprite

ANEXO III. FIGURAS DE UN ALGORITMO.

Previamente al diseño del algoritmo es importante definir un diagrama de flujo del programa según las anotaciones de la fase creativa.

El diagrama de flujo consistirá en una sucesión de acciones ordenadas cronológicamente de inicio a fin. El diagrama de flujo contará con desviaciones condicionales, bucles con contador, salto direccionado a otra parte de código, salto condicionado a otra parte de código y llamadas a subrutinas.

Las formas a utilizar en el diagrama de flujo se recogen en la tabla 15.

Una vez establecido el diagrama de flujo se escribirá el algoritmo del programa, en la tabla 5 se muestra como traducir a algoritmo los diferentes componentes del diagrama de flujo.

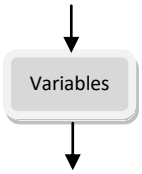
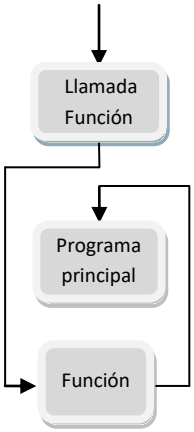
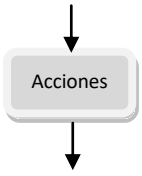
DIAGRAMA DE FLUJO	ALGORITMO	BASIC	DESCRIPCIÓN
	Variables: **Var** A,B,C: entero X,Y,Z: simple presición R,S,T: doble precisión VI,NU: alfanumerica A(X,Y,..,Z): matriz **Fvar**	A%,B%,C%: entero X!,Y!,Z!: simple precisión R#,S#,T#: doble precisión VI$,UN$: alfanumérica A(3,2..,255): matriz	Se pueden declarar variables de tipo entero (entre -32768 y 32767), simple precisión (6 dígitos de precisión), doble precisión (14 dígitos de precisión), alfanumérica (combinación de letras y números), matrices donde se indica entre paréntesis cuantos subindices tienen cada dimensión definida)
	Funciones: **Función** Nombre función Parámetros Expresión **FinFunción** Llamada de la función: **Nombre (parámetros)**	**DEFFN** A(X,Y)=(X*Y)+2 A:Nombre de la función X,Y: Parámetros entrada (X*Y)+2: Expresión devuelta Llamada: A=FN A(5,4) A es igual a (5*4)+2	Se puede declarar funciones para realizar una operación concreta, esta función consta de un nombre, parámetros de entrada y operación a realizar con los parámetros. Para llamarla hay que indicar el nombre de la función y los parámetros de entrada.
	Acción: **A=A+1**; **Imprimirpantalla** “hola”; **Colocarcursor** 10,10; …	**A=A+1** **PRINT** “HOLA” **LOCATE** 10,10	Realiza una acción por ejemplo dar valor a una variable, realizar una operación matemática, imprimir un texto, colocar el cursor, etc.

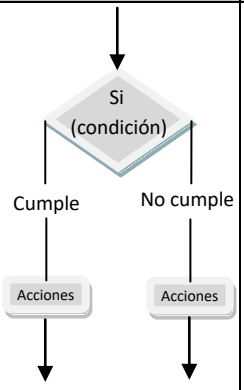
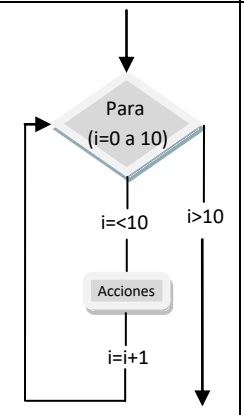
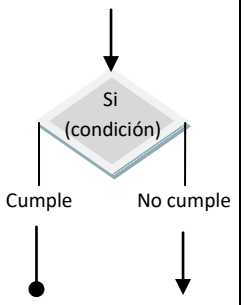
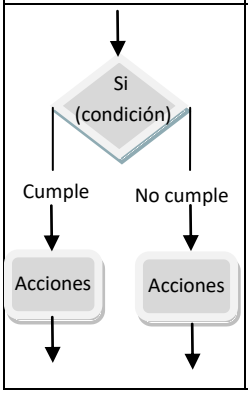
DIAGRAMA DE FLUJO	ALGORITMO	BASIC	DESCRIPCIÓN
	Decisión condicional: **Si** (condición) **entonces** acciones **Sino** acciones **FinSi**	… **IF** A=1 **THEN** PRINT “HOLA” **ELSE** PRINT “ADIOS” …	Ejecuta unas acciones si se cumple una determinada condición. Si no se cumple ejecuta otras acciones.
	Bucle con contador: **Para** (i=0 hasta 10) Acciones **i = i + 1** **FinPara**	… **FOR** I=0 **TO** 10 PRINT “HOLA” **NEXT I** …	Se repite unas acciones determinadas mientras un contador no llegue a un valor determinado. Por cada iteración el valor del contador aumenta.
	Salto direccionado: **Ira** 1000 Salto condicional: **Si** (condición) **Ira** Final **Sino Ira** Inicio **FinSi**	Salto direccionado … **GOTO** 1000 … Salto condicional … **IF** A>10 **GOTO** 1000 **ELSE GOTO** 10 …	Los saltos en MSX-Basic, al contrario que en C que se utilizan etiquetas, se hacen indicando la línea de programa donde se quiere llamar y no a una etiqueta. Para suplir esta mancanza se puede direccionar el salto a una linea de comentario (REM) que describe la función del código al que saltamos. Ej: Final, CambioFase, Inicio…
	Acción condicional: **Si** (condición) **Entonces** (Acciones) **Sino** (Acciones) **FinSi**	Acción condicional: … **IF** A>10 **THEN** (Acciones) **ELSE** (Acciones) …	Igual que los saltos condicionales se puede determinar acciones condicionales, si se cumple una condición se realizan unas condiciones si no se realizan otras diferentes.

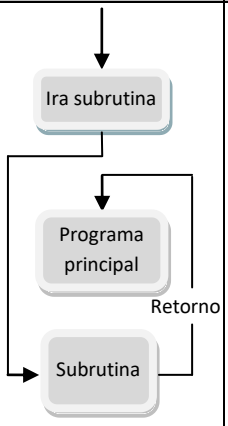
DIAGRAMA DE FLUJO	ALGORITMO	BASIC	DESCRIPCIÓN
	Llamadas a subrutina: **Ira** subrutina Subrutina: Acciones **Retorno**	Llamada a subrutina **GOSUB** 100 Ejemplo subrutina: 100 REM SUBRUTINA 110 A=A+B 120 IF A>10 THEN B=0 130 B=B+1 140 PRINT A 150 **RETURN**	Los saltos o llamadas a subrutinas se realizan igual que los saltos incondicionales. Una vez se salta a la subrutina se ejecutan las acciones pertinentes de la subrutina y se retorna al punto siguiente de la llamada. Igual que en los saltos la llamada a la subrutina se realiza a un número de línea de programa por lo que se puede utilizar una línea de comentario (REM) a modo de etiqueta
	Final de programa Finalalgoritmo	**END**	Fin de programa.

Tabla 15. Relación diagrama de flujo, algoritmo, orden Basic

ANEXO IV. MSX-BASIC comandos de programación

Una vez se tenga el algoritmo se deberá programar utilizando el MSX-BASIC, para ello se ejecutará el programa BLUEMSX, ilustración 11, bien en aplicación u online.

Al ejecutarlo nos aparecerá directamente el intérprete de BASIC, que es el programa que se utiliza para programar en MSXBASIC. Es posible que el emulador nos pida la fecha actual, si fuera el caso se introduce o se presiona al Return directamente.

Ilustración 11. IDE o interfaz BLUEMSX modo BASIC

El intérprete nos muestra un mensaje inicial indicando la versión del MSX BASIC, la memoria disponible, y la versión del Disk Basic o sistema operativo de disco y un mensaje final de OK donde indica que el sistema ha arrancado correctamente. Debajo del OK aparece el cursor, a partir de esta línea podremos introducir nuestro código.

Las línea inferior muestra las palabras **color auto goto list run,** estas palabras indican las funciones preestablecidas a las teclas de función F1 a F5 de muestro PC. No las utilizaremos, para borrarlas teclear **KEYOFF** y presionar Return.

Se puede introducir comandos en modo directo, por ejemplo **PRINT** 10+10 y al presionar Return aparecerá por pantalla 20. Una vez ejecutado un comando directo no queda almacenado en memoria, por lo que su uso principal es para utilizar el Basic a modo de calculadora.

Para programar se utiliza el modo indirecto donde debemos numerar líneas de programa y estas quedan almacenadas en memoria RAM por lo que hay que tener en cuenta que si no se salva el programa, al cerrar el emulador se perderá todo el programa.

Para salvar el programa se utiliza ARCHIVO – GRABAR ESTADO CPU en el IDE del BLUEMSX

Para recuperar el programa se utiliza ARCHIVO – ESTADO CARGA CPU.

Una vez introducido un programa se ejecutará mediante la orden **RUN**. Si hubiese un error en el código introducido nos aparecería un mensaje de error indicando la línea y el tipo de error, por ejemplo: **Syntax error in 10** (error de sintaxis en línea 10).

Para empezar a introducir un programa se puede utilizar la instrucción **AUTO** para numerar automáticamente las líneas de programa. Una línea queda introducida tras presionar RENTURN. Se puede introducir en cada línea más de un comando separándolos mediante ":"

Por ejemplo:

10 (comando) **:** (comando) **:** (comando)

Equivaldría a:

10 (comando)
20 (comando)
30 (comando)

Variables y operaciones.

TIPOS DE VARIABLES: Según tabla 16

TIPO	DECLARACIÓN	DESCRIPCIÓN
ENTERA	**DEFINT A** o **A%**	Número entero entre -32768 y 32767.
SIMPLE PRECISIÓN	**DEFSNG A** o **A!**	Número decimal con 6 dígitos de precisión.
DOBLE PRECISIÓN	**DEFDBL A** o **A#**	Número decimal con 14 dígitos de precisión.
ALFANUMÉRICA	**DEFSTR A** o **A$**	Combinación de letras y números
MATRIZ	**DIM A(X,Y,…,255)** **DIM A!(X,Y,…,255)** **DIM A#(X,Y,…,255)** **DIM A$(X,Y,…,255)**	Entre paréntesis se indica los subíndices de cada dimensión, se puede utilizar hasta 255 dimensiones y hay que indicar el tipo de dato que contiene la matriz, si no se indica se considera entera.

Tabla 16. Variables

OPERADORES ARITMÉTICOS: Según tabla 17.

OPERADOR	OPERACIÓN	EXPRESIÓN
+	SUMA	A + B
-	RESTA	A – B
*	MULTIPLICACIÓN	A * B

/	DIVISIÓN	A / B
–	NEGACIÓN	– A
^	EXPONENCIAL	A ^ B
\	DIVISIÓN ENTERA	A \ B
MOD	MÓDULO O RESTO	A MOD Y

Tabla 17. Operadores aritméticos

OPERADORES DE RELACIÓN: Según tabla 18.

OPERADOR	RELACIÓN	EXPRESIÓN
=	IGUAL	A = B
<>	DIFERENTE	A <> B
<	NENOR QUE	A < B
>	MAYOR QUE	A > B
<=	MENOR IGUAL QUE	A <= B
>=	MAYOR IGUAL QUE	A >= B

Tabla 18. Operadores relacionales

OPERADORES LÓGICOS: Según tabla 19.

OPERADOR	OPERACIÓN	EXPRESIÓN
NOT	Negación – NO	NOT A
AND	Producto – Y	A AND B
OR	Suma – O	A OR B
XOR	OR exclusiva	A XOR B
IMP	Implicación	A IMP Y
EQV	Equivalencia	A EQV B

Tabla 19. Operadores lógicos

OPERACIONES CON CADENAS ALFANUMÉRICAS (STRING).

Las cadenas se pueden sumar mediante el signo +, por ejemplo B$=A$+C$

También se pueden comparar utilizando los operadores de relación.

FUNCIONES NUMÉRICAS. Según tabla 20.

FUNCIÓN	EXPRESIÓN
ABS(X)	Muestra el valor absoluto de un real X.
INT(X)	Muestra la parte entera de un real redondeando a la baja en números negativos.
FIX(X)	Muestra la parte entera de un real redondeando al alta en número negativos.
SGN(X)	Muestra el signo de un real X. Si es positivo SGN(X)=1 Si es negativo SNG(X)=-1

FUNCIÓN	EXPRESIÓN
	Si es 0 SNG(X)=0
CDBL(X)	Muestra un real X en doble precisión (14 dígitos)
CSNG(X)	Muestra un real X en simple precisión (6 dígitos)
CINT(X)	Muestra un real X como un entero (entre -32768 y 32767)
EXP(X)	Calcula el exponencial de un real X
LOG(X)	Calcula el logaritmo de un real X
SQR(X)	Calcula la raíz cuadrada de un real X
SIN(X)	Calcula el seno de un real X
COS(X)	Calcula el coseno de un real X
TAN(X)	Calcula la tangente de un real X
ATN(X)	Calcula el arcotangente de un real X
RND(X)	Da un número aleatorio entre 0 y 1

Tabla 20. Operadores artiméticos

Comandos de programación.

Un comando es una función ya diseñada por el MSXBASIC para realizar ciertas operaciones.

En este apartado enunciaremos los comandos más usuales para programar en MSXBASIC.

COMANDOS DE PANTALLA

CLS:

Sintaxis: **CLS**

Descripción: Borra la pantalla.

Ejemplo: **CLS**

COLOR:

Sintaxis: **COLOR** (X),(Y),(Z)

Descripción: Establece el color de fondo de la letra (X), color de fondo (Y) y del margen (Z) de la pantalla. La paleta de color de MSX es de 16 colores, cada uno tiene un código según tabla 21.

CÓDIGO	COLOR
0	Transparente
1	Negro
2	Verde
3	Verde claro
4	Azul oscuro
5	Azul claro
6	Rojo oscuro
7	Azul celeste
8	Rojo

9	Rojo claro
10	Amarillo oscuro
11	Amarillo claro
12	Verde oscuro
13	Magenta
14	Gris
15	Blanco

Tabla 21. Paleta de color

Ejemplo: **COLOR** 2,1,1

LOCATE:

Sintaxis: **LOCATE** (X),(Y)

Descripción: Coloca el cursor en la posición indicada. X columna Y fila.

Ejemplo: **LOCATE 10,10**

KEY ON/OFF:

Sintaxis: **KEY ON, KEY OFF**

Descripción: Borra los títulos de las letras de función de la parte inferior de la pantalla.

Ejemplo: **KEY ON**

COMANDOS DE CONSOLA

Este grupo de comando se utiliza para asistir al programador.

AUTO:

Sintaxis: **AUTO** (Número primera línea),(incremento)

Descripción: Se utiliza para enumerar las líneas automáticamente, si no entran parámetros enumerará a partir de la línea 10 con incrementos de 10 unidades por línea

Ejemplo: **AUTO**
AUTO 10,100

DELETE:

Sintaxis: **DELETE** (Número primera línea) – (Número última línea)

Descripción: Borra las líneas en el intervalo indicado del programa. Si no se indica el número de fila final sólo borra la línea indicada.

Ejemplo: **DELETE 100 – 220**
DELETE 30

LIST:

Sintaxis: **LIST** (Número primera línea) – (Número última línea)

Descripción: Muestra en pantalla las líneas de programa del intervalo indicado. Si sólo se entra un número de línea muestra todo el programa a partir de dicha línea. Si no se introducen números de línea lista todo el programa.

Ejemplo: **LIST 10 – 50**
LIST 100
LIST

NEW:

Sintaxis: **NEW**

Descripción: Borra toda la RAM

Ejemplo: **NEW**

RENUM:

Sintaxis: **RENUM** (Número línea nuevo),(Número línea anterior),(Incremento)

Descripción: Reenumera el número de línea del programa, si se omiten los parámetros de entrada reenumera el programa desde la línea 10 a intervalos de 10.

Ejemplo: **RENUM 1000,120,10**

RENUM

REM:

Sintaxis: **REM** (Comentarios)

Descripción: Inserta una línea de comentario, esta línea no es ejecutable por lo tanto es obviada por el Basic. También se puede introducir mediante el signo ‘.

Ejemplo: **REM** Programa inicial
REM Subrutina movimiento
‘ Subrutina marcadores

COMANDOS DE PROGRAMACIÓN

RUN:

Sintaxis: **RUN** (Número de línea)

Descripción: Ejecuta el programa desde el número de línea indicado, si no se indica ejecuta el programa desde el inicio.

Ejemplo: **RUN 100**

RUN

END:

Sintaxis: **END**

Descripción: Finaliza la ejecución del programa.

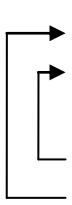
Ejemplo: **END**

FOR:

Sintaxis: **FOR** (Variable=valor inicial) **TO** (Variable=valor final) **STEP** (Incremento)
…
Acciones
…
NEXT (Variable)

Descripción: Se utiliza para realizar bucles con contador, utiliza una variable numérica a modo de contador, esta variable incrementa según el incremento indicado **STEP**, si no se indica, el incremento será de una unidad. El comando FOR va asociado siempre al comando **NEXT**, que se coloca al final de bucle e indica que la variable del contador se debe incrementar. **NEXT** cierra el bucle y retorna el programa al inicio del bucle donde se encuentra el **FOR**.

Ejemplo:
```
10 REM Rutina para imprimir los 100 primeros números
20 FOR I=0 TO 100
30 PRINT I
40 NEXT I
50 END
```
Es posible anidar bucles teniendo en cuenta la correcta colocación de los comando **NEXT**



```
10 REM Rutina para imprimir las tablas de multiplicar hasta el 10
20 FOR I=1 TO 10
30 FOR J=1 TO 10
40 PRINT I*J
50 NEXT J
60 NEXT I
70 END
```

GOTO:

Sintaxis: **GOTO** (Número de línea)

Descripción: Realiza un salto direccionado a la línea indicada.

Ejemplo:

```
10 REM Imprimir HOLA MUNDO con salto direccionado
20 GOTO 50
30 PRINT "MUNDO"
40 END
50 PRINT "HOLA "
60 GOTO 30
```

IF:

Sintaxis: **IF (**Condición**) THEN (**Acciones**) ELSE (**Acciones**)**

IF (Condición**) GOTO (**Acciones**) ELSE (**Acciones**)**

Descripción: Realiza un salto condiciona, si se cumple la condición se realizan las acciones que se indican tras el **THEN**, si no se cumple se realizan las acciones que indica el **ELSE**. Si se utiliza **GOTO** en vez del **THEN** se realizarán saltos condicionales

Ejemplo:

```
10 REM Programa para calcular la mayoría de edad
20 INPUT "Que edad tienes"; A
30 IF A<18 THEN PRINT "Eres mayor de edad" ELSE PRINT "Eres
   menor de edad"
40 END
```

Salto condicional

```
10 REM Calcula mayoría de edad con salto condicional
20 INPUT "Que edad tienes"; A
30 IF A>17 GOTO 40 ELSE 60
40 PRINT "Eres mayor de edad"
50 END
60 PRINT "Eres menor de edad"
70 END
```

GOSUB:

Sintaxis: **GOSUB** (número de línea)

Descripción: Llama a una subrutina. La subrutina debe ser concluida por el comando **RETURN** para volver al programa principal en el punto justo después a la llamada mediante un **GOSUB**.

Ejemplo:

```
10 REM Llamada a una subrutina para calcular el cuadrado
20 INPUT "Entra un número: "; A
30 GOSUB 1000
40 PRINT "El cuadrado de ";A; "es: ";B
50 END
1000 REM Subrutina para calcular cuadrados
```

1010 B=A
1020 B=B^2
1030 RETURN

COMANDOS DE DECLARACIÓN Y TRATAMIENTO DE VARIABLES

DIM:

Sintaxis: **DIN** Vabriable (Subíndice dim1, subíndice dim2,…,subíndice din255)

Descripción: Dimensiona una variable matriz, se indica los subíndices de cada dimensión definida, puede haber hasta 255 dimensiones.

Ejemplo: **DIN** A(2,4,8,1)

La matriz A tiene cuatro dimensiones y cada dimensión contiene 2,4,8 y 1 elemento respectivamente.

DEFINT:

Sintaxis: **DEFINT** (Variables)

Descripción: Define como enteros una lista de variables. Una variable entera también se puede declarar mediante el signo % o simplemente no declarar. Una variable no declarada se considerará entera.

Ejemplo: **DEFINT** A,B,C

A%=2 (Variable A declarada como entera con un valor de 2)

DEFSNG:

Sintaxis: **DEFSNG** (Variables)

Descripción: Define como simple precisión una lista de variables. Una variable de simple precisión cuenta con 6 cifras como máximo. También se puede declarar mediante el signo !.

Ejemplo: **DEFSNG** A,B,C

A!=4354,32 (Variable A declarada como simple precisión con un valor de 4354,32)

DEFDBL:

Sintaxis: **DEFDBL** (Variables)

Descripción: Define como doble precisión una lista de variables. Una variable de doble precisión cuenta con 14 cifras como máximo. También se puede declarar mediante el signo #.

Ejemplo: **DEFDBL** A,B,C

A#=56754298,67893432 (Variable A declarada como doble precisión con un valor de 56754298,67893432)

DEFSTR:

Sintaxis: **DEFSTR** (Variables)

Descripción: Define como cadena alfanumérica (string) una lista de variables. También se puede declarar mediante el signo $.

Ejemplo: **DEFSTRL** A,B,C

A$= "Hola" (Variable A declarada string o cadena alfanumérica con valor HOLA)

DEFFN:

Sintaxis: **DEFFN** (Nombre) (Parámetros de entrada de la función) = (Expresión de la función)

Descripción: Define una función, la expresión de la función será siempre matemática. La llamada de la función se realizará mediante la expresión **FN** (Nombre) (Parámetros de entrada).

Ejemplo:
```
10 REM Función cálculo área rectángulo.
20 DEFFN A(X,Y)=X*Y
30 INPUT "Indica longitud de la base: ";X
40 INPUT "Indica longitud de la altura: ";Y
50 PRINT FN A(X,Y)
60 END
```

TIME:

Sintaxis: **TIME**

Descripción: Variable del sistema que cuenta las interrupciones del VDP que se producen a 60 Hz (60 veces por segundo). Comienza a contar cuando se enciende el sistema pero se puede iniciar cuando interese con la expresión **TIME**=0.

Ejemplo:
```
10 REM Cronómetro
20 TIME=0
30 LOCATE 10,10
40 A=TIME/60
50 PRINT A
60 GOTO 30
```

COMANDOS PROCESOS DE DATOS

READ:

Sintaxis: **READ (**Variables)

Descripción: Asigna a una variable numérica o alfanumérica el valor recogido en un comando **DATA.** Se asignará el valor siguiente al último valor **DATA** leído. El programa empieza a leer la primera línea de **DATA** que encuentra, y continuará en futuras órdenes **READ** por el punto donde se ha quedado.

Ejemplo:

```
10 REM Rutina para sumar serie de datos
20 FOR I=1 TO 10
30 READ A
40 B=B+A
60 NEXT I
70 PRINT B
80 END
90 READ 3,4,5,4,9,7,1,2,8,0
```

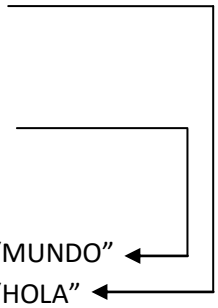
RESTORE:

Sintaxis: **RESTORE (**Número de línea)

Descripción: Indica un número de línea determinado donde se encuentra una instrucción **DATA.** Si no se indica el número de línea buscará la primera orden **DATA** del programa

Ejemplo:

```
10 REM Hola mundo
20 RESTORE 1010
30 READ A$
40 PRINT A$
50 RESTORE
60 READ A$
70 PRINT A$
80 END
1000 DATA "MUNDO"
1010 DATA "HOLA"
```



INPUT:

Sintaxis: **INPUT** ("Mensaje");(Variables)

Descripción: Asigna un valor entrado por teclado a una variable. Se puede introducir un mensaje que aparecerá en pantalla, tras el mensaje aparece el signo ? por defecto, si no se introduce un mensaje aparecerá el signo ? sólo en pantalla.

Ejemplo:

```
10 REM Saludo
20 INPUT "Como te llamas";A$
30 PRINT "Hola ";A$
```

40 END

PRINT:

Sintaxis: **PRINT** (“Expresión”);(Variables)

Descripción: Imprime por pantalla el valor de una variable, una operación matemática o bien un mensaje entrecomillado.

Ejemplo:
```
10 REM Impresión por pantalla
20 INPUT “Entra tu número de la suerte: ”;B
30 A$=“Buen número!!!!”
40 PRINT “Tu número de la suerte es: ”;B;A$
50 END
```

SWAP:

Sintaxis: **SWAP** (Variable),(Variable)

Descripción: Intercambia el valor de dos variables del mismo tipo.

Ejemplo:
```
10 REM Cambio
20 A=10
30 B=20
40 PRINT A,B
50 SWAP A,B
60 PRINT A,B
70 END
```

INKEY$:

Sintaxis: **INKEY$**

Descripción: Lee un carácter introducido por el teclado. Después de un tiempo, si no se pulsa una tecla interpreta que se ha leído un carácter nulo. El tiempo de espera es corto pero mediante un bucle, esta función es útil para pausar el programa hasta que se presione una tecla.

Ejemplo:
```
10 REM Espera a pulsar una tecla
20 PRINT “A LA ESPERA”
30 A$=INKEY$
40 IF A$=”” GOTO 30
50 PRINT “SE TERMINÓ LA ESPERA”
60 END
```

POKE:

Sintaxis: **POKE** (Dirección),(Dato)

Descripción: Introduce un dato a una posición de memoria determinada. **VPOKE** introduce el dato en la memoria de video. Los datos deben ser un Byte.

Ejemplo: **POKE 1200,45**

PEEK:

Sintaxis: **PEEK** (Dirección)

Descripción: Recupera un dato (Byte) de una posición de memoria determinada. **VPEEK** lee el dato de la memoria de video.

Ejemplo: **PRINT PEEK**(234)

STICK:

Sintaxis: **STICK** (X)

Descripción: Indica el movimiento realizado por el Joystick, El valor de X depende del dispositivo a analizar, según tabla 22.

DISPOSITIVO	VALOR DE X
0	CURSORES
1	JOYSTICK 1
2	JOYSTICK 2

Tabla 22. Valor dispositivo STICK

El valor indicado según el movimiento va de 0 a 8 según tabla 23.

MOVIMIENTO	VALOR STICK
Centro	0
Arriba	1
Arriba-Derecha	2
Derecha	3
Derecha-Abajo	4
Abajo	5
Abajo-Izquierda	6
Izquierda	7
Izquierda-Arriba	8

Tabla 23. Valor dirección STICK

Ejemplo:
```
10 REM Mueva los cursores
20 LOCATE 10,10
30 PRINT STICK(0)
40 GOTO 20
```

STRING:

Sintaxis: **STRING** (X)

Descripción: Indica si está pulsado el botón de disparo o la barra espaciadora según tabla 24.

DISPOSITIVO	VALOR DE X
0	ESPACIO
1	JOYSTICK 1

2	JOYSTICK 2

Tabla 24. Valor dispositivo STRING

El valor de STRING es -1 si se pulsa el botón de disparo o barra espaciadora, si no se pulsa es 0

Ejemplo:

```
10 REM Presione espacio
20 LOCATE 10,10
30 PRINT STRING(0)
40 GOTO 20
```

COMANDOS DE INTERRUPCIÓN

ON STRIG GOSUB:

Sintaxis: **ON STRIG GOSUB** (Número de línea)

Descripción: Interrumpe el programa cuando el botón de disparo o barra espaciadora es pulsada. Tras la interrupción el programa deriva a una función establecida. Para poder utilizar este comando previamente hay que activarlo mediante el comando **STRIG(X) ON**, una vez activado se puede desactivar la interrupción mediante el comando **STRIG(X) OFF**. Donde X indica el dispositivo según tabla 25.

DISPOSITIVO	VALOR DE X
0	ESPACIO
1-3	JOYSTICK 1
2-4	JOYSTICK 2

Tabla 25. Valor dispositivo ON STRIG GOSUB

Ejemplos:

```
10 REM Interrupción de contador con barra espaciador
20 STRIG(0) ON
30 A=A+1
40 LOCATE 10,10
50 PRINT A
60 ON STRIG GOSUB 1000
70 GOTO 30
1000 REM Fin de contador
1010 STRIG(0) OFF
1020 END
```

ON SPRITE GOSUB:

Sintaxis: **ON SPRITE GOSUB** (Número de línea)

Descripción: Interrumpe el programa cuando dos **Sprites** colisionan. Tras la interrupción el programa deriva a una función establecida. Para poder utilizar este comando hay que activar el comando **SPRITE ON**, una vez

activado se puede desactivar la interrupción mediante el comando **SPRITE OFF**.

Ejemplos: ***NOTA****: este ejemplo es una modificación de una rutina mostrada en el Manual de Referencia del MSX BASIC, 1986, ed. TOSHIBA CORPORATION, la rutina muestra comandos que se verán en la parte de diseño gráfico).*

```
10 SCREEN Interrupción al colisionar dos Sprites
20 SPRITE ON
30 ON SPRITE GOSUB 100
40 SPRITE$(0)=STRING$(8,255)
50 FOR K=0 TO 300
60 PUT SPRITE 0,(K,200-K),1,0
70 PUT SPRITE 1,(200-K,K),8,0
80 NEXT K
90 END
100 SCREEN 0
110 LOCATE 10,10
120 PRINT "COLISIÓN!!!!"
130 END
```

ON INTERVAL GOSUB:

Sintaxis: **ON INTERVAL =** (Tiempo) **GOSUB** (Número de línea)

Descripción: Interrumpe el programa tras un determinado tiempo. El comando utiliza como contador las interrupciones del VDP por lo que incrementa cada 1/60 segundos. Para poder utilizar este comando previamente hay que activarlo mediante el comando **INTERVAL ON**, una vez activado se puede desactivar la interrupción mediante el comando **INTERVAL OFF**.

Ejemplos:

```
10 REM Interrupción tras tiempo de espera
20 INTERVAL ON
30 ON INTERVAL=300 GOSUB 1000
40 LOCATE 10,10
50 PRINT "TIEMPO DE ESPERA"
60 GOTO 40
1000 REM Subrutina Fin de espera
1010 INTERVAL OFF
1020 PRINT "FIN DE LA ESPERA"
1030 END
```

ANEXO V. PROGRAMACIÓN GRÁFICA.

La parte gráfica de un MSX está controlada por el **VDP** o procesador de video (Video Display Processor) que está conformado por el micro **TMS9918A**.

La pantalla se conforma con la superposición de varias capas, en concreto 34.

La capa de fondo determina el borde, donde sólo se puede cambiar el color y sobresale del resto de capas o pantallas. Encima de la capa de borde se encuentra la capa de caracteres, donde se representan los caracteres de textos o gráficos (tiles). Las 32 capas siguientes contienen los **Sprites** o gráficos. Estos **Sprites** se utilizarán para crear y mover los personajes por pantalla. La composicón de las capas se ve en la ilustración 12.

El orden de prioridad de capas establecido es en primer lugar las 32 de **Sprites** (de la 0 a la 31) después la de caracteres y por último la de borde.

Ilustración 12. Imagen de las capas desplegadas y superpuestas.

El VDP se puede configurar en cuatro modos de pantalla llamados SCREEN, cada modo tiene una resolución determinada tal como se muestra en la tabla 26.

SCREEN	CARACTERES	RESOLUCIÓN	SPRITES
SCREEN 0	40x24	---	NO
SCREEN 1	32x24	---	32
SCREEN 2	---	256x192	32
SCREEN 3	---	64x48	32

Tabla 26. Tipo SCREEN

SCREEN 0.

Pantalla para solo para textos, muestra 40 caracteres de ancho por 24 de alto. Se puede establecer mediante la instrucción **COLOR** el color del texto, de la pantalla y del borde. No permite la utilización de **SPRITES**.

En este modo no se puede utilizar comandos de modo gráfico como **LINE**, **CIRCLE**…

SCREEN 1

Pantalla para texto 32x24 caracteres. Se puede establecer mediante la instrucción **COLOR** el color del texto, pantalla y borde y está permitido el uso de **SPRITES.**

Al igual que en **SCREEN 0** no está permitido utilizar comandos de modo gráfico.

SCREEN 2

Pantalla gráfica de alta resolución (256x192 pixeles). En este modo se puede utilizar comandos de modo gráfico pero no de texto como **LOCATE, INPUT, PRINT…**

Se pueden utilizar tiles de 8x8 pixeles definiendo sólo dos colores por línea para crear gráficos de fondo. Se pueden utilizar **SPRITES**.

Es el modo de pantalla mayoritariamente utilizado para la creación de videojuegos.

SCREEN 3

Modo multicolor donde se permite dibujar gráficos por bloques de 4x4 pixeles con un máximo de 64x48 bloques en pantalla. En cada bloque de 4x4 se puede utilizar 16 colores.

Dibujo mediante comandos gráficos.

El MSX posee unos comandos para trazar figuras simples tales como líneas, puntos, círculos, elipses,

SCREEN:

Sintaxis: **SCREEN** (Modo),(Tamaño Sprites)

Descripción: Selecciona el modo de pantalla. El sistema arranca en SCREEN 0 por defecto. El modo va de 0 a 3 según el Screen deseado. El tamaño de los Sprites va de 0 a 3 según tabla 27.

TAMAÑO	RESOLUCIÓN	TIPO
0	8X8	NORMAL
1	8X8	AMPLIADO
2	16X16	NORMAL
3	16X16	AMPLIADO

Tabla 27. Tipo Sprite

Ejemplo: **SCREEN 2,2**

PSET:

Sintaxis: **PSET** (X,Y),(Color)

Descripción: Dibuja un punto en pantalla, X e Y son las coordenadas en pantalla. El color va 0 a 15 según la paleta de color del **VDP.** Este comando sólo es válido para **SCREEN** 2 y **SCREEN** 3. Para borrar el punto se utiliza **PRESET** (X,Y).

Ejemplo:

```
10 REM Punto
20 SCREEN 2
30 PSET (100,100),6
40 FOR I=1 TO 2000
50 NEXT I
60 PRESET (100,100)
70 END
```

LINE:

Sintaxis: **LINE** (X_i,Y_i) – (X_f,Y_f),(Color),(B o BF)

Descripción: Dibuja una entre dos coordenadas. X_i,Y_i coordenadas de inicio X_f,Y_f coordenadas del final. El color va 0 a 15 según la paleta de color del **VDP.** Este comando sólo es válido para **SCREEN** 2 y **SCREEN** 3. Se puede dibujar un rectángulo especificando B donde los puntos iniciales y finales conformarán sus vértices opuestos. Si se especifica BF el área del rectángulo quedará pintada.

Ejemplo: **LINE** (10,10) – (150,150),(8)

CIRCLE:

Sintaxis: **CIRCLE** (X,Y),(Radio),(Color),(Ángulo inicial),(Ángulo final),(Excentricidad)

Descripción: Dibuja círculos, elipses y arcos. X,Y son las coordenadas del centro, El ángulo inicial y final se utiliza para trazar arcos, estos ángulos deben expresarse en radianes (0 - 2π). La excentricidad es la relación que hay entre el eje horizontal y el vertical de la elipse. . El color va 0 a 15 según la paleta de color del **VDP.** Este comando sólo es válido para **SCREEN** 2 y **SCREEN** 3.

Ejemplo: **CIRCLE** (100,100),60,3

CIRCLE (100,100),80,4,,,6

PAINT:

Sintaxis: **PAINT** (X,Y),(Color 1)

Descripción: Pinta de un mismo color un área determinada cerrada. Las coordenadas deben hacer referencia a un punto interior al área a dibujar. Color 1 será el color con el que se pintará el área interior, este color debe ser igual que el borde de la figura a pintar. El color va 0 a 15 según la paleta de color del **VDP.** Este comando sólo es válido para **SCREEN** 2 y **SCREEN** 3.

Ejemplo: **PAINT** (100,100),6

DRAW:

Sintaxis: **DRAW** (expresión)

Descripción: Dibuja trazos en pantalla según la dirección deseada. Para ello hay que introducir en forma de cadena de caracteres o **String** una serie de instrucciones indicando la dirección y un valor numérico indicando la longitud del trazo en píxels. Estas instrucciones o parámetros se detallan el la tabla 28.

CARÁCTER	OPERACIÓN
Mx,y	Desplaza el puntero a unas coordenadas absolutas en pantalla (resolución: 256x192), si las coordenadas x e y van seguidas de un signo + o – el desplazamiento es relativo con dirección según el signo.
Bn	Desplazamiento sin trazar en pantalla de n píxels.
Nn	Realiza el trazo de n píxels y vuelve a la coordenada de partida
Un	Desplazamiento hacia arriba de n píxels.
Dn	Desplazamiento hacia abajo de n píxels.
Rn	Desplazamiento hacia la izquierda de n píxels.
Ln	Desplazamiento hacia la derecha de n píxels.
En	Desplazamiento diagonal derecha-arriba de n píxels.
Fn	Desplazamiento diagonal derecha-abajo de n píxels.
Gn	Desplazamiento diagonal izquierda-abajo de n píxels.
Hn	Desplazamiento diagonal izquierda-arriba de n píxels.
An	Gira la figura dibujada en incrementos de 90 grados (volteo horizontal y vertical). El ángulo n se indica del 0 al 3 siendo $0=0^o$,$1=90^o$,$2=180^o$,$3=270^o$
Cn	Cambia el color a utilizar. Valor de n 0-15 acorde con paleta de color
Sn	Factor de escala donde n va de 1 a 255. El factor de escala es igual a n/4, por defecto si no se indica n=4 o sea factor 1 (4/4).

X(variable)	Se puede introducir un **String** recogido en una variable alfanumérica

Tabla 28. Parámetros Comando DRAW

Ejemplo:

```
10 REM Dibujo de un barco
20 SCREEN 2
30 DRAW "BM80,130R100E20L140F20BM+20,-20R10D10R20D10"
40 GOTO 40
```

Resultado se ve en la ilustración 13.

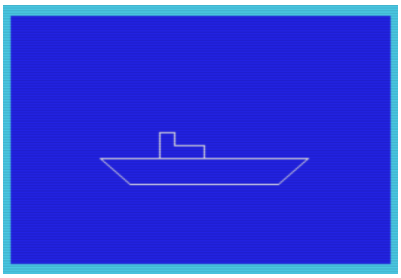


Ilustración 13. Resultado comando DRAW

Generación de gráficos mediante TinySprite.

Un **Sprite** es un gráfico de 8x8 o 16x16 píxels que procesa el **VDP** en capas diferentes, hasta 32 como hemos visto, esto permite el poder superponer gráficos uno por encima de otro dando creando efecto de profundidad.

Los **Sprites** son fáciles de dotar de movimiento, una vez definidos hay que indicar la dirección y velocidad y el **VDP** realiza el trabajo gráfico independientemente de la **CPU**.

Como se indica en los comandos de programación, mediante la interrupción **ON SPRITE GOSUB** nos permite crear rutinas cuando existe una colisión entre ellos por lo que no es necesario crear rutinas de comprobación de colisiones.

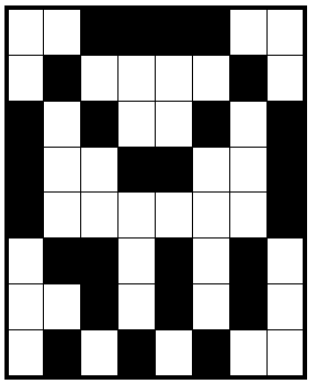
COMANDOS PARA SPRITES

SPRITE$(n):

Sintaxis: **SPRITE$** (n)= (Definición)

Descripción: Define la forma del **Sprite** donde **n** es el número de gráfico definido donde irá insertado**.** Como ya se ha comentado hay 32 capas (de 0 a 31) donde la capa 0 tiene mayor preferencia y la capa 31 la menor a la hora de superponerse. El tamaño del **Sprite** es de 8x8 o 16x16, esto se indica mediante la instrucción **SCREEN** ya comentada.

La definición del **Sprite** se realiza mediante la introducción de cada línea lo forma por separado en forma de carácter **CHR$,** cada carácter contendrá un número binario (con el prefijo **&B**) que define a la línea. En la ilustración 18 se ve el gráfico y su codificación binaria.

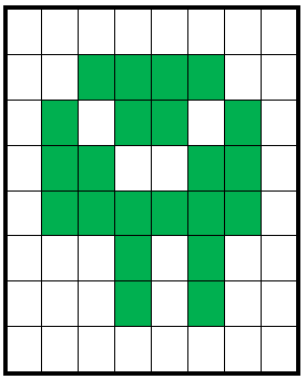


0	0	1	1	1	1	0	0	&B00111100
0	1	0	0	0	0	1	0	&B01000010
1	0	1	0	0	1	0	1	&B10100101
1	0	0	1	1	0	0	1	&B10011001
1	0	0	0	0	0	0	1	&B10000001
0	1	1	0	1	0	1	0	&B01101010
0	0	1	0	1	0	1	0	&B00101010
0	1	0	1	0	1	0	0	&B01010100

Ilustración 14. Generación SPRITE I

Ejemplo: **SPRITE$**(0) = CHR$(&B00111100)+ CHR$ (&B01000010)+ CHR$ (&B10100101)+ CHR$ (&B10011001)+ CHR$ (&B10000001)+ CHR$ (&B01101010)+ CHR$ (&B00101010)+ CHR$ (&B01010100)

Se puede utilizar varios **Sprites** para dar colorido a los personajes. Si definimos un segundo **Sprite**. En la ilustración 15 se ve el resultado en binario de este segundo **Sprite.**



0	0	0	0	0	0	0	0	&B00000000
0	0	1	1	1	1	0	0	&B00111100
0	1	0	1	1	0	1	0	&B01011010
0	1	1	0	0	1	1	0	&B01100110
0	1	1	1	1	1	1	0	&B01111110
0	0	0	1	0	1	0	0	&B00010100
0	0	0	1	0	1	0	0	&B00010100
0	0	0	0	0	0	0	0	&B00000000

Ilustración 15. Generación SPRITE II

**SPRITE$**(1) = CHR$(&B00000000)+ CHR$ (&B00111100)+ CHR$ (&B01011010)+ CHR$ (&B01100110)+ CHR$ (&B01111110)+ CHR$ (&B00010100)+ CHR$ (&B00010100)+ CHR$ (&B00000000)

Al superponer la capa 0 y la 1 obtenemos un resultado más vistos, como se aprecia en la ilustración 16.

Ilustración 16. Superposición SPRITES

PUT SPRITE:

Sintaxis: **PUT SPRITE** (n),(X,Y),(Color),(nº Sprite)

Descripción: Esta sentencia se utiliza para colocar una imagen o **Sprite** en pantalla y definir sus atributos.

En **n** se indica el número de capa donde colocamos el **Sprite.**

Las coordenadas (X,Y) indica la posición del **Sprite**, para dar movimiento basta con ir variando el valor de **X** e **Y** pero se puede sustituir por **STEP (x,y)** donde en **x** se indica la variación que hará **X** e **y** la variación que hara **Y** en cada ciclo.

El Color se indica con un valor de 0 a 15 según la paleta.

El último parámetro pertenece al número de **Sprite** definido.

Ejemplo:

```
10 SCREEN 2
20 X=100:Y=100
30 REM Definir sprites
40 SPRITE$(0) = CHR$(&B00111100)+ CHR$ (&B01000010)+ CHR$
(&B10100101)+ CHR$ (&B10011001)+ CHR$ (&B10000001)+ CHR$
(&B01101010)+ CHR$ (&B00101010)+ CHR$ (&B01010100)
50 SPRITE$(1) = CHR$(&B00000000)+ CHR$ (&B00111100)+ CHR$
(&B01011010)+ CHR$ (&B01100110)+ CHR$ (&B01111110)+ CHR$
(&B00010100)+ CHR$ (&B00010100)+ CHR$ (&B00000000)
60 REM Colocar Sprites
70 PUTSPRITE 0,(X,Y),1,0
80 PUTSPRITE 1,(X,Y),2,1
90 REM Mover Sprites
100 X=X+1
110 REM Retardo
120 FOR I=1 TO 50
130 NEXT I
140 GOTO 60
```

Generar binarios utilizando TinySprite.

Para facilitar la generación de binarios para los **Sprites** se puede utilizar un programa para tal fin. Un programa muy útil es el **TinySprite**.

El programa está concebido para trabajar con **Sprites** de 8x8 pero es igualmente útil para gráficos de 8x8

Su uso es muy sencillo, primeramente se dibujará la fomra deseada en la rejilla o **Sprite Grid** utilizando el color que se desee de la paleta de color inferior, como se ve en la ilustración 17.

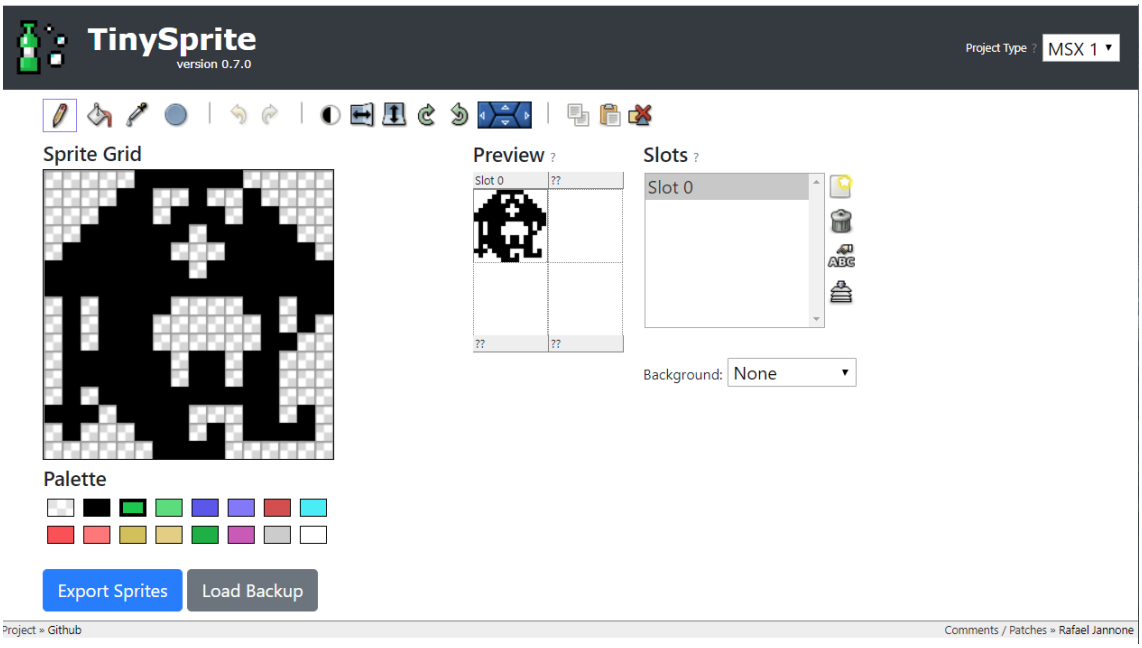


Ilustración 17. SPRITE con TINYSPRITE

Una vez dibujado el gráfico obtendremos los gráficos se obtiene un archivo con el cógido necesario para generarlo presionando en **Export Sprites.** Aparacerá una ventana donde podemos indicar como queremos obtener los datos en el desplegable **Export As..,** como es muestra en la ilustración 18.

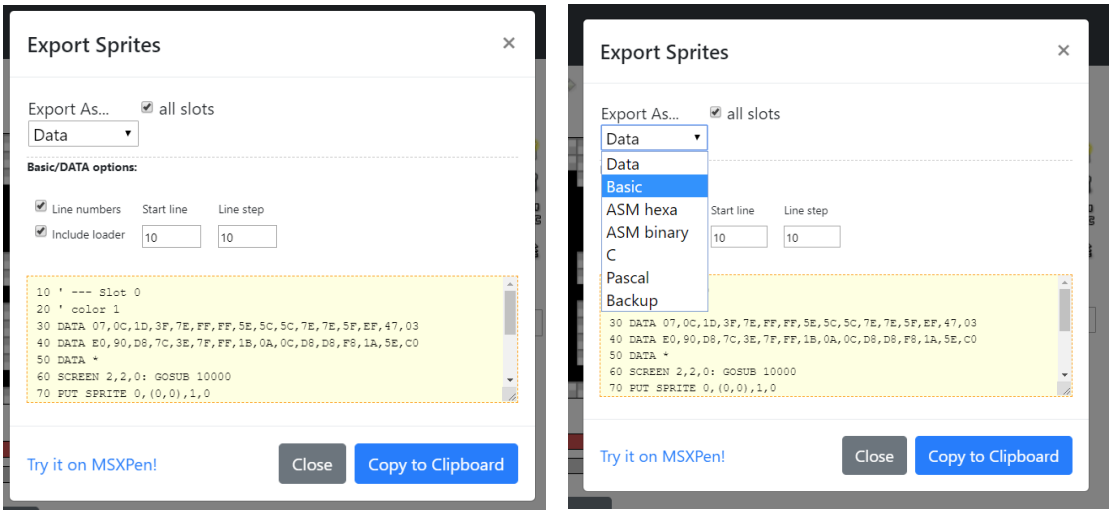


Ilustración 18. Obtención de código TinySprite

El programa proporciona los datos que forman el gráfico y un programa cargador con el código a introducir para definir los **Sprites.**

Los datos los podemos obtener en varios tipos.

DATA: en hexadecimal, el cargador carga los datos en el **VDP** directamente mediante instrucciones VPOKE

BASIC: en hexadecimal, los datos se cargan directos a memoria de video igual que en **DATA.**

ASM binary: se obtienen los binarios para ensamblador (sistema de programación de bajo nivel). Este archivo es útil para obtener los binarios y definirlos mediante la instrucción **SPRITE$.**

C: para programas codificados en lenguaje C.

PASCAL: para programas codificados en lenguaje Pascal.

BACKUP: nos genera un archivo que nos permitirá recuperar el trabajo en otra sesión.

Al presionar Copy to Clipboard queda seleccionado y pegado todo el texto listo para pegar.

En este caso el archivo sería:

```
10 ' --- Slot 0
20 ' color 1
30 DATA &H07,&H0C,&H1D,&H3F,&H7E,&HFF,&HFF,&H5E
40 DATA &H5C,&H5C,&H7E,&H7E,&H5F,&HEF,&H47,&H03
50 DATA &HE0,&H90,&HD8,&H7C,&H3E,&H7F,&HFF,&H1B
60 DATA &H0A,&H0C,&HD8,&HD8,&HF8,&H1A,&H5E,&HC0
70 DATA *
80 SCREEN 2,2,0: GOSUB 10000
90 PUT SPRITE 0,(0,0),1,0
100 GOTO 100
10000 REM -- LOAD SPRITES
10010 S=BASE(9)
10020 READ R$: IF R$="*" THEN RETURN ELSE VPOKE S,VAL(R$):S=S+1:GOTO 10020
```

Las líenas **10** y **20** equivalen a un **REM.** De la líena **30** a la **70** recoge en hexadecimal la definición del gráfico. De la **80** a la **100** coloca el gráfico en pantalla, esta parte de código no la necesitarios ya que no se ajustará a lo que necesitamos. De la línea **10000** a la **10020** se recoge el cargador en memoria del gráfico, equivale a **SPRITE$** y sería el **SPRITE** número 0.

ANEXO VI. Desarrollo musical.

El sonido, tanto música como efectos, se genera mediante el **PSG** o Programmable Sound Generator controlado por el chip **AY-3-8910.** En Basic se controla mediante los comandos **PLAY** (música) y **SOUND** (efectos).

Este generador posee tres canales A,B,C de tonos que pueden sonar simultáneamente y un cuarto canal de "ruido" para producir efectos sonoros.

PLAY:

Sintaxis: **Play** (expresión canal A),(expresión canal B),(expresión canal C)

Descripción: Por cada canal se indicará las notas, duración, pausas, ritmo, intensidad y envolvente.

NOTAS: Se expresan según notación anglosajona A-G según ilustración 19.

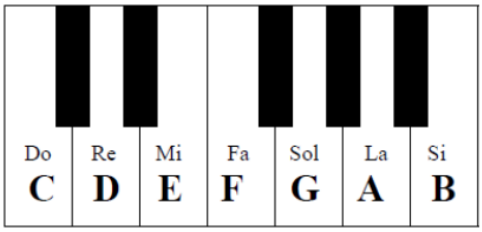


Ilustración 19. Escala musical

Se puede indicar si es bemol mediante el signo – o sostenido mediante los signos # o + indiferentemente. Ejemplo **PLAY "**A#" (La sostenido) **PLAY** "E-" (Mi bemol). Al contrario que la notación musical de una partitura, no se permite indicar bemoles o sostenidos en notas si en medio no hay semitonos, por lo que no sería válido B#, C-,E#, F- (que se podrían expresar mediante las expresiones C, B, F, E respectivamente).

OCTAVAS: Se dispone de 8 octavas, si no se indica se interpreta en la cuarta octava. Se indica mediante **O**n donde es el número de octava (1-8). Ejemplo: **PLAY "**O5A" (octava 5 La).

DURACIÓN: Se representa mediante **L**n donde n varía de 1 a 64 donde 1 es la nota completa, 2 es media nota, 3 es un tercio de nota… Al indicar la duración afecta a todas las notas siguientes hasta que se vuelva a indicar otra duración. Ejemplo: **PLAY "L4**C+EFE**L1**D+EFE". Si se quiere indicar la duración a una única nota no hace falta introducir

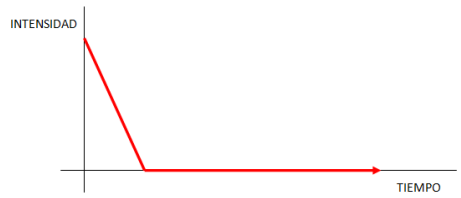
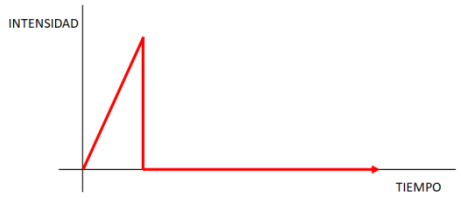
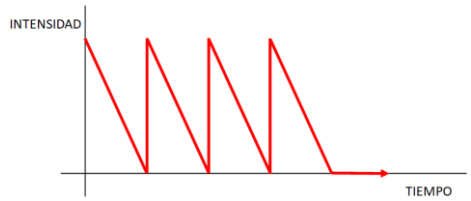
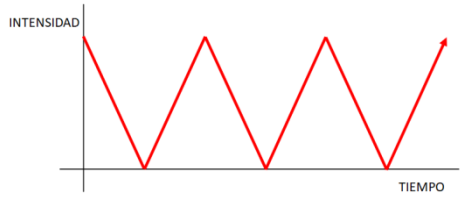
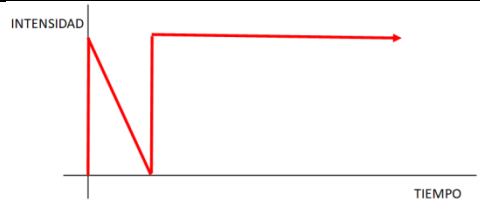
L, indicando la duración tras la nota es suficiente, ejemplo: **PLAY "**BA4C", sólo el LA (A) durará un cuarto de nota.

PAUSA: se representa mediante **R**n donde **n** indica la duración de la pausa de 1 a 64. Ejemplo: **PLAY "**ABCR1ABC"

RITMO: se representa mediante **T**n donde **n** es un entero entre 32 y 255 e indica las corcheas o cuartos de notas a tocar por minuto. El valor de **T** por defecto es 120. Ejemplo **PLAY** "T200ABCT100ABC"

VOLUMEN: se especifica mediante **V**n donde **n** tiene un valor de 0 a 15. Si no se especifica el volumen por defecto es 8. Ejemplo: **PLAY** "V15ABCV3ABC".

ENVOLVENTE O TIMBRE: la envolvente o timbre de la nota se expresa mediante dos parámetros **S**n y **M**n, donde **S** indica el tipo de envolvente siendo **n** de 0 a 15 según tabla 29, y **M** es el valor de la modulación o período de onda, **n** va de 1 a 65635,

Vn (TIPO)	FORMA ENVOLVENTE EN EL TIEMPO
0 – 1 – 2 – 3 – 9	
4 – 5 – 6 – 7 – 15	
8	
10	
11	

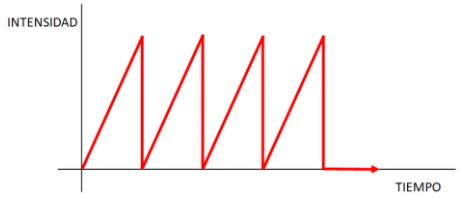
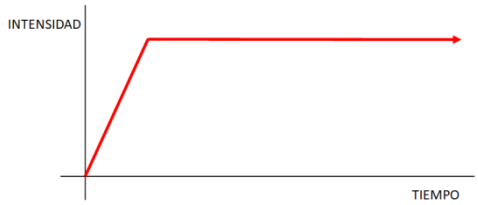
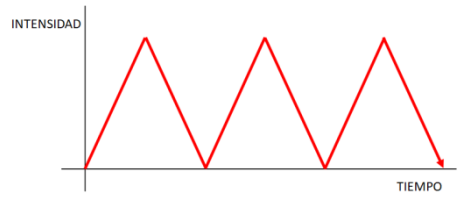
Vn (TIPO)	FORMA ENVOLVENTE EN EL TIEMPO
12	
13	
14	

Tabla 29. Tipo de envolvente

USO DE VARIABLE X: se puede utilizar una variable tipo **String** expresada como **X** que sustituya a un código. Ejemplo:

10 A$=”ABCABCEFGEFG”
20 PLAY “XA$”

Ejemplo de melodía con dos canales:

10 PLAY “T180S0M15000”,”T180S1M400”
20 PLAY “01CEGB-“,”O5C8O4B-8R8G8B-8O5C8O5C8O4B”
30 PLAY “02C01B-GE”, “O5C8O4B-4G8O5C8”
40 PLAY “O1CEGB-”
50 PLAY “O2C01B-GE”
60 PLAY “O1FAO2CE-“,“O5F8E-C8”
70 PLAY “FE-CO1A”,“F8E-4C8F8”
80 PLAY “CEGB-”
90 PLAY “O2CO1B-GE”
100 PLAY “O2GBDB”, “G8B8D8B8”

En relación a los efectos de sonido el **PSG** tiene un canal llamado de “ruido” este canal se controla mediante el comando **SOUND.**

SOUND:

Sintaxis: **SOUND** (Registro PSG),(Valor)

Descripción: El **PSG** cuenta con 14 registros de escritura (0-13) cada registro realiza una función determinada, como se muestra en la ilustración 30.

REGISTRO	VALOR	FUNCIÓN
0	0-255	Frecuencia canal A
1	0-15	
2	0-255	Frecuencia canal B
3	0-15	
4	0-255	Frecuencia canal C
5	0-15	
6	0-31	Frecuencia de ruido
7	0-63	Selecciona un canal para generación de tonos y ruido
8	0-15	Volumen canal A
9	0-15	Volumen canal B
10	0-15	Volumen canal C
11	0-255	Frecuencia del patrón de variación de volumen
12		
13	0-14	Selección del patrón de variación de volumen

Tabla 30. Registros comando SOUND

Ejemplo:

```
10 FOR I=1 TO 50
20 SOUND 6,0
30 SOUND 7,55
40 SOUND 8,16
50 SOUND 11,23
60 SOUND 12,2
70 SOUND 13,9
80 NEXT I
```

ANEXO VI. EJEMPLO DE DESARROLLO.

Ponemos un ejemplo de un videojuego programado en MSX-BASIX.

Historia:

Título: **PENGUIN SLALOM**

Autor: Juan C. Jurado (2019).

Xixo es un pequeño pingüino decido a esquiar en la prueba de eslalon para la próxima olimpiada que se celebrará en Ziloglandia del Norte.

Para ello debe clasificarse alcanzado el nivel 10 en la prueba. Cada cinco veces que pase entre las banderas subirá un nivel, pero cuidado, no se puede cometer más de 10 faltas durante la prueba o será descalificado.

Si pasa fuera de las banderas se considerará una penalización, también se penaliza si se toca una bandera o a otro esquiador.

CONTROLES:

Xixo se mueve a izquierda y derecha utilizando los cursores o bien el joystick en el puerto 1.

Storyboard:

Un extracto de un boceto realizado a mano podría ser según la ilustración 20.

Ilustración 20. Storyboard.

Digitalización del boceto podría quedar según la ilustración 21 procesada con GIMP.

Ilustración 21. Boceto digitalizado

Diseño gráfico:

Creación de los personajes: Con el personaje del boceto se elabora el gráfico del personaje como se ve en la ilustración 22.

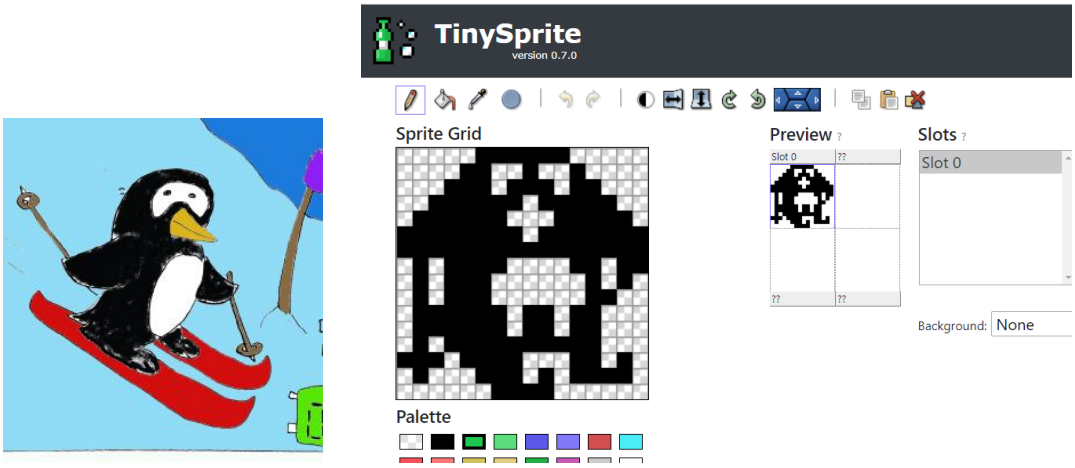


Ilustración 22. Paso del boceto a gráfico.

Generación de fondos mediante Gimp según ilustración 23 y 24.

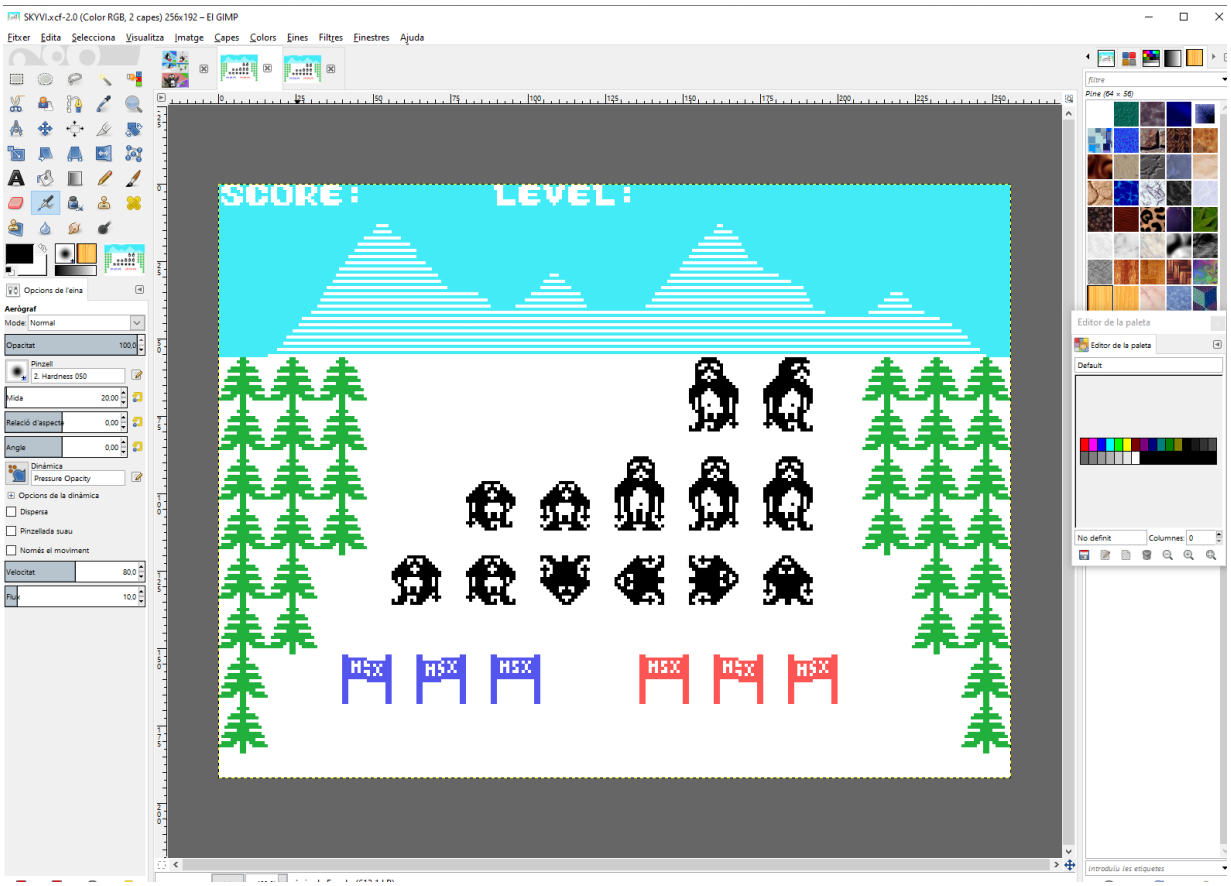


Ilustración 23. Diseño de fondo

Ilustración 24. Diseño de pantallas

Diseño musical:

Melodía de dos voces, bajos y agudos. El código en Basic sería según ilustración 25.

```
PLAY "T180S0M15000","T180S1m4000"
PLAY "O1CEGB-","O5C8O4B-8r8g8b-8o5c8o4b-8"
PLAY "O2CO1B-GE","O5C8O4B-46805C8"
PLAY "O1CEGB-"
PLAY "O2CO1B-GE"
PLAY "O1FAO2CE-","O5F8E-C8"
PLAY "FE-CO1A","F8E-4C8F8"
PLAY "CEGB-"
PLAY "O2CO1B-GE"
PLAY "O2GBDB","G8B8D8B8"
PLAY "O1FO2ACA","F8A8C8A8C8A8"
PLAY "O1CEGB-","o5c8o4b-8r8g8b-8b8o5c8"
PLAY "O2C8R8C8R8C8R8C8","C8R8C8R8C8R8C8"
```

Ilustración 25. Código melodía.

Código programa.

```
10 SCREEN 1,2:KEYOFF:COLOR 1,15,15
20 LOCATE 3,12:PRINT"LOADING PENGUIN SLALOM"
30 ' CARGAR SPRITES ------------------------------------------------------------
40 GOSUB 2640
50 REM INICIAR TABLA DE COLORES ------------------------------------------------
60 FOR K=3 TO 19                ' COLOR TILES BLANCOS Y AZUL MONTANYA
70 VPOKE(BASE(6))+K,247
80 NEXT K
90 REM RELLENAR TABLA DE PATRONES ----------------------------------------------
100 FOR K=0 TO 7
110 VPOKE(BASE(7))+(34*8+K),255
120 NEXT K
130 FOR K=0 TO 7
140 VPOKE(BASE(7))+(32*8+K),0
150 NEXT K
160 RESTORE
170 FOR K=0 TO 31                ' CARGA TILES ARBOL I
180 READ A
190 VPOKE(BASE(7))+K,A
200 NEXT K
210 RESTORE
220 FOR K=64 TO 95               ' CARGA TILES ARBOL LINEA II
230 READ A
240 VPOKE(BASE(7))+K,A
250 NEXT K
260 FOR K=0 TO 7
270 VPOKE(BASE(7))+(44*8)+K,0
280 NEXT K
290 FOR K=0 TO 27*8            ' CARGA TILES MONTANYA
300 READ A
310 VPOKE(BASE(7))+(128*8)+K,A
320 NEXT K
330 RESTORE 1830
340 FOR K=0 TO 10*8         ' RELLENAR NUMEROS
350 READ A
360 VPOKE(BASE(7))+(48*8)+K,A
370 NEXT K
380 RESTORE 1940
390 FOR K=0 TO 26*8        ' RELLENAR LETRAS
400 READ A
410 VPOKE(BASE(7))+(65*8)+K,A
420 NEXT K
430 REM PANTALLA PRESENTACION --------------------------------------------------
440 CLS
450 LOCATE 0,2:PRINT"------ PENGUIN SLALOM ------"
460 PUTSPRITE 4,(20,46),1,8
470 LOCATE 4,6:PRINT"CURSOR OR JOY RIGHT"
480 PUTSPRITE 5,(20,75),1,9
490 LOCATE 4,10:PRINT"CURSOR OR JOY LEFT"
500 LOCATE 0,14:PRINT"KEEP FAR AWAY FROM OTHERS    PENGUINS AND CROSS THROUGH   THE
FLAGS"
510 LOCATE 0,19:PRINT"TRY TO FINISH THE SLALOM WITHLESS OF 9 FAULTS          "
520 GOSUB 3050
530 LOCATE 9,22:PRINT"PRESS SPACE"
540 W=STRIG(0)
550 IF W=-1 THEN GOTO 570 ELSE GOTO 540
560 REM RELLENAR TABLA DE NOMBRES ----------------------------------------------
570 CLS:PUTSPRITE 0,(100,208),1,10
580 B=32*7
590 FOR J=0 TO 3
600 VPOKE(BASE(5))+B,2:VPOKE(BASE(5))+B+2,10:VPOKE(BASE(5))+B+4,2
610 VPOKE(BASE(5))+B+1,3:VPOKE(BASE(5))+B+3,11:VPOKE(BASE(5))+B+5,3
620 VPOKE(BASE(5))+B+26+1,3:VPOKE(BASE(5))+B+26+3,11:VPOKE(BASE(5))+B+26+5,3
630 VPOKE(BASE(5))+B+26,2:VPOKE(BASE(5))+B+26+2,10:VPOKE(BASE(5))+B+26+4,2
640 C=B+32
```

```
720 FOR J=0 TO 3
730 VPOKE(BASE(5))+B,10:VPOKE(BASE(5))+B+2,2:VPOKE(BASE(5))+B+4,10
740 VPOKE(BASE(5))+B+1,11:VPOKE(BASE(5))+B+3,3:VPOKE(BASE(5))+B+5,11
750 VPOKE(BASE(5))+B+26,10:VPOKE(BASE(5))+B+26+2,2:VPOKE(BASE(5))+B+26+4,10
760 VPOKE(BASE(5))+B+26+1,11:VPOKE(BASE(5))+B+26+3,3:VPOKE(BASE(5))+B+26+5,11
770 C=B+32
780 B=B+128
790 VPOKE(BASE(5))+C,8:VPOKE(BASE(5))+C+2,0:VPOKE(BASE(5))+C+4,8
800 VPOKE(BASE(5))+C+1,9:VPOKE(BASE(5))+C+3,1:VPOKE(BASE(5))+C+5,9
810 VPOKE(BASE(5))+C+26,8:VPOKE(BASE(5))+C+26+2,0:VPOKE(BASE(5))+C+26+4,8
820 VPOKE(BASE(5))+C+26+1,9:VPOKE(BASE(5))+C+26+3,1:VPOKE(BASE(5))+C+26+5,9
830 NEXT J
840 RESTORE 1750
850 FOR J=0 TO (32*7)-1
860 READ A
870 VPOKE((BASE(5))+J),A
880 NEXT J
890 FOR K=7 TO 22
900 FOR J=0 TO 19
910 VPOKE(BASE(5))+((K*32)+6)+J,34
920 NEXT J
930 NEXT K
940 FOR K=0 TO 31
950 VPOKE(BASE(5))+(23*32)+K,34
960 NEXT K
970 X=127:Y=80:CS=1 ' ----------------- INICIAR VARIABLES --------------------
980 N=255:M=63:X1=127:X2=127:Y1=192:Y4=192:IN=6:LE=80:SF=4:LV=1:FL=0:Y5=192:T=0:X4=1
50:X5=50:FA=0:SP=7
990 ' TABLA ATRIBUTOS SPRITES ------------------------------------------------
1000 ON SPRITE GOSUB 2910
1010 SPRITE ON
1020 IF SF>6 THEN SF=4
1030 IF TI>0 THEN SPRITE OFF
1040 PUTSPRITE 0,(X,Y),1,SP
1050 PUTSPRITE 2,(X1,Y1),8,SF
1060 PUTSPRITE 3,(X1+64,Y1),4,SF
1070 PUTSPRITE 4,(X4,Y4),13,SF+3
1080 PUTSPRITE 5,(X5,Y5),10,SF+3
1090 SF=SF+1
1100 REM CAMBIAR TABLA DE COLORES SCREEN1 ------------------------------------
1110 IF O=2 THEN SWAP M,N:O=0
1120 VPOKE(BASE(6)),M
1130 VPOKE(BASE(6))+1,N
1140 O=O+1
1150 ' MOVIMIENTO DEL PINGUINO -----------------------------------------------
1160 W=STICK(0):V=STICK(1)
1170 IF TI>0 THEN 1220
1180 IF W=3 OR V=3 THEN X=X+4:SP=8:IF X>192 THEN X=192
1190 IF W=7 OR V=7 THEN X=X-4:SP=9:IF X<48 THEN X=48
1200 IF W=0 AND V=0 THEN SP=7
1210 ' MOVIMIENTO BANDERAS ---------------------------------------------------
1220 IF Y1<193 THEN Y1=Y1-IN
1230 IF Y1<64 THEN Y1=192:GOSUB 2690:X1=XR
1240 IF X1<48 THEN X1=48
1250 IF X1>99 THEN X1=99
1260 ' MOVIMIENTO DE LOS PINGUINOS PEQUENYOS
1270 IF Y1<150 OR Y4<192 THEN Y4=Y4-IN
1280 IF Y1<110 OR Y5<192 THEN Y5=Y5-IN
1290 IF Y4<64 THEN Y4=192:GOSUB 2700:X4=XR
1300 IF X4>191 THEN CS=(-1)
1310 IF X4<48 THEN CS=(1)
1320 X4=X4+(T*CS)
1330 IF Y5<64 THEN Y5=192:GOSUB 2700:X5=XR
1340 IF X5>191 THEN CS=(-1)
1350 IF X5<48 THEN CS=(1)
```

```
1420 IF LV>10 THEN 2720 ' FINAL DE PARTIDA ----------------------------------
1430 IF TI>0 THEN SPRITE OFF:TI=TI+1:SP=SP+1:IF SP>3 THEN SP=0
1440 IF TI>6 THEN SPRITE ON:TI=0
1450 GOTO 1020
1460 REM DATOS ---------------------------------------------------------------
1470 DATA 3,63,127,15,125,249,1,1,192,252,254,240,190,159,128,128,1,3,1,15,31,1,31,6
3,128,192,128,240,248,128,240,252
1480 DATA 0,3,0,7,0,31,0,255
1490 DATA 0,3,0,31,0,63,0,255
1500 DATA 0,31,0,63,0,127,0,255
1510 DATA 0,0,0,1,0,3,0,7
1520 DATA 0,1,0,7,0,31,0,63
1530 DATA 0,0,0,0,0,30,0,63
1540 DATA 0,224,0,252,0,255,0,255
1550 DATA 0,224,0,248,0,254,0,255
1560 DATA 0,192,0,255,0,255,0,255
1570 DATA 0,0,0,240,0,254,0,255
1580 DATA 0,0,0,7,0,31,0,255
1590 DATA 0,0,0,0,0,28,0,127
1600 DATA 0,192,0,248,0,252,0,254
1610 DATA 0,1,0,7,0,15,0,63
1620 DATA 0,7,0,31,0,63,0,127
1630 DATA 0,7,0,31,0,127,0,255
1640 DATA 0,0,0,0,0,112,0,252
1650 DATA 0,0,0,128,0,192,0,224
1660 DATA 0,252,0,255,0,255,0,255
1670 DATA 0,0,0,0,0,248,0,255
1680 DATA 0,248,0,252,0,254,0,255
1690 DATA 0,0,0,0,0,1,0,7
1700 DATA 0,0,0,120,0,254,0,255
1710 DATA 0,240,0,255,0,255,0,255
1720 DATA 0,0,0,128,0,248,0,255
1730 DATA 0,192,0,224,0,252,0,255
1740 DATA 0,255,0,255,0,255,0,255
1750 DATA 83,67,79,82,69,58,44,44,44,44,44,44,44,76,69,86,69,76,58,44,44,44,44,70,65,85
,76,84,58,44,44,44,44
1760 DATA 44,44,44,44,44,44,44,133,44,44,44,44,44,44,44,44,44,44,44,44,44,44,44,144,44,44,44,
44,44,44,44,44,44,44,44,44
1770 DATA 44,44,44,44,44,44,132,154,134,44,44,44,44,44,44,44,44,44,44,44,44,44,143,154,145,44
,44,44,44,44,44,44,44,44,44
1780 DATA 44,44,44,44,44,131,154,154,154,135,44,44,44,44,139,44,44,44,44,142,154,154,14
6,147,44,44,44,44,44,44,44,44,44
1790 DATA 44,44,44,44,44,130,154,154,154,154,136,137,44,138,154,140,44,44,141,154,154,1
54,154,154,148,44,44,149,150,44,44,44,44
1800 DATA 44,44,44,44,129,154,154,154,154,154,154,154,154,154,154,154,154,154,154,154,154,1
54,154,154,154,154,154,154,154,154,151,152,44,44
1810 DATA 44,128,154,154,154,154,154,154,154,154,154,154,154,154,154,154,154,154,154,154,154
,154,154,154,154,154,154,154,154,154,154,153,44
1820 ' DATA NUMEROS
1830 DATA 0,62,99,69,73,81,99,62
1840 DATA 0,12,30,6,6,6,6,31
1850 DATA 0,60,102,6,60,96,96,126
1860 DATA 0,126,71,3,31,7,71,126
1870 DATA 0,54,102,102,63,6,6,63
1880 DATA 0,127,115,112,126,7,103,126
1890 DATA 0,127,115,112,126,103,103,126
1900 DATA 0,127,103,14,28,56,112,96
1910 DATA 0,62,115,99,62,99,103,62
1920 DATA 0,62,115,99,63,14,28,120
1930 ' DATA LETRAS
1940 DATA 0,62,119,99,99,127,99,99
1950 DATA 0,126,103,103,126,103,103,126
1960 DATA 0,62,119,115,112,115,119,62
1970 DATA 0,124,126,99,97,99,126,124
1980 DATA 0,62,127,112,126,112,127,62
```

```
2060 DATA 0,99,119,127,107,99,99,99
2070 DATA 0,99,113,125,111,99,99,115
2080 DATA 0,62,103,99,99,99,115,62
2090 DATA 0,62,115,113,127,126,112,112
2100 DATA 0,62,115,97,97,109,102,57
2110 DATA 0,126,115,115,118,124,118,115
2120 DATA 0,62,115,113,28,71,103,62
2130 DATA 0,127,127,93,28,28,28,28
2140 DATA 0,97,115,115,115,115,127,62
2150 DATA 0,99,119,54,54,62,28,8
2160 DATA 0,99,119,54,62,62,54,20
2170 DATA 0,115,58,30,28,60,46,103
2180 DATA 0,99,119,54,62,28,28,62
2190 DATA 0,127,79,14,28,56,121,127
2200 ' PINGUINO CAIDA
2210 DATA &Hf1,&Hb9,&H1f,&H4f,&Hef,&H4f,&H5f,&Hff
2220 DATA &Hff,&H7f,&H3c,&H1d,&H0F,&H06,&H03,&H01
2230 DATA &H8f,&H9d,&Hf8,&Hf2,&Hf7,&Hf2,&Hfa,&Hfe
2240 DATA &Hff,&Hff,&H9e,&Hdc,&Hf8,&H30,&H60,&Hc0
2250 DATA &H01,&H03,&H07,&H0f,&H1f,&H3f,&H73,&Hdb
2260 DATA &H9f,&Hdb,&H73,&H3f,&H1f,&H0f,&H07,&H03
2270 DATA &H93,&Hf9,&H93,&Hc7,&Hfe,&Hfc,&Hfc,&Hff
2280 DATA &Hff,&Hfc,&Hfc,&Hfe,&Hc7,&H93,&Hf9,&H13
2290 DATA &H03,&H06,&H0c,&H1f,&H3b,&H79,&Hff,&Hff
2300 DATA &H7f,&H5f,&H4f,&Hef,&H4f,&H1f,&Hb9,&Hf1
2310 DATA &H80,&Hc0,&H60,&Hf0,&Hb8,&H3c,&Hfe,&Hff
2320 DATA &Hff,&Hfa,&Hf2,&Hf7,&Hf2,&Hf8,&H9d,&H8f
2330 DATA &Hc8,&H9f,&Hc9,&He3,&H7f,&H3f,&H3f,&Hff
2340 DATA &Hff,&H3f,&H3f,&H7f,&He3,&Hc9,&H9f,&Hc9
2350 DATA &Hc0,&He0,&Hf0,&Hf8,&Hfc,&Hce,&Hdb,&Hf9
2360 DATA &Hdb,&Hce,&Hfc,&Hf8,&HF0,&He0,&Hc0,&H80
2370 ' SPRITE FLAG
2380 DATA &HC0,&HC0,&HFF,&HEA,&HE2,&HEB,&HEA,&HFF
2390 DATA &HFF,&HC0,&HC0,&HC0,&HC0,&HC0,&HC0,&HC0
2400 DATA &H00,&HFF,&H57,&HEF,&H6F,&H57,&HFF,&HFF
2410 DATA &H00,&H00,&H00,&H00,&H00,&H00,&H00,&H00
2420 DATA &HC0,&HFF,&HEA,&HE2,&HEB,&HEA,&HFF,&HC0
2430 DATA &HC0,&HC0,&HC0,&HC0,&HC0,&HC0,&HC0,&HC0
2440 DATA &H00,&HFF,&H57,&HEF,&H6F,&H57,&HFF,&H00
2450 DATA &h00,&h00,&h00,&h00,&h00,&h00,&h00,&h00
2460 DATA &HC0,&HFF,&HEA,&HE2,&HEB,&HEA,&HFF,&HFF
2470 DATA &HC0,&HC0,&HC0,&HC0,&HC0,&HC0,&HC0,&HC0
2480 DATA &H00,&H00,&HFF,&H57,&HEF,&H6F,&H57,&HFF
2490 DATA &HFF,&H00,&H00,&H00,&H00,&H00,&H00,&H00
2500 ' SPRITE PINGUINO PEQUENYO
2510 DATA &H03,&H04,&H0D,&H1F,&H3E,&H7F,&HFF,&HFC
2520 DATA &HD8,&H58,&H4C,&HED,&H4D,&H1F,&HBC,&H7E
2530 DATA &HE0,&H90,&HD8,&H78,&H3C,&H7E,&HFF,&H3F
2540 DATA &H1B,&H1A,&H32,&HB7,&HB2,&HF8,&H3D,&H7E
2550 DATA &H07,&H0C,&H1D,&H3F,&H7E,&HFF,&HFF,&H5E
2560 DATA &H5C,&H5C,&H7E,&H7E,&H5F,&HEF,&H47,&H03
2570 DATA &HE0,&H90,&HD8,&H7C,&H3E,&H7F,&HFF,&H1B
2580 DATA &H0A,&H0C,&HD8,&HD8,&HF8,&H1A,&H5E,&HDC
2590 DATA &H07,&H09,&H1B,&H3E,&H7C,&HFE,&HFF,&HD8
2600 DATA &H50,&H30,&H1B,&H1B,&H1F,&H58,&H7A,&H3B
2610 DATA &HE0,&H30,&HB8,&HFC,&H7E,&HFF,&HFF,&H7A
2620 DATA &H3A,&H3A,&H7E,&H7E,&HFA,&HF7,&HE2,&HC0
2630 DATA %
2640 ' SUBRUTINA PARA CARGAR SPRITES ----------------------------------------
2650 S=BASE(9):RESTORE 2210
2660 READ R$:IF R$="%" THEN RETURN ELSE VPOKE S,VAL(R$)
2670 S=S+1
2680 GOTO 2660
```