TUTORIAL DE INICIACION A LA PROGRAMACION EN LENGUAJE ENSAMBLADOR PARA MSX

3ª PARTE – EL MUNDO DE LOS GRAFICOS 1

En esta parte del tutorial vamos a crear un Hola Mundo Grafico utilizando un juego de caracteres de texto creado por nosotros y todo lo relacionado con el modo grafico Screen 2, así como las herramientas que necesitaremos para realizar todo este cometido.

Antes de empezar con el código del Hola Mundo Grafico, vamos con la parte que menos le gusta a la gente, la teoría… pero ojo que sin ella no se comprenden las cosas.

A diferencia del primer ejemplo donde solo seleccionábamos la fila columna y el texto que queríamos imprimir, y este salía impreso en la pantalla.

Recuerda que el modo de pantalla seleccionado era Screen 0 con el siguiente código.

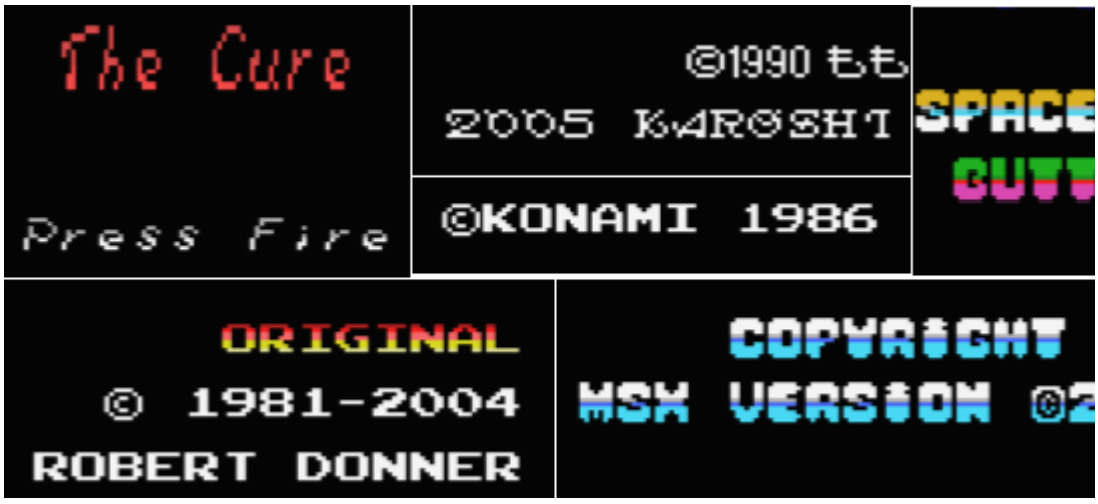
```
call    INITXT      ; BIOS set SCREEN 0
```

Este es el modo de pantalla que utiliza el Basic del MSX, el set de caracteres o letras que sale impreso en pantalla está integrado en la BIOS, y es el que se muestra cuando encendemos el ordenador.

Ahora que vamos a utilizar el Screen 2 este es un modo grafico, y podremos crear las letras como queramos dándoles un colorido especial, haciendo que nuestra ROM tenga un aspecto más profesional. (Tengo que aclarar que todo lo aquí explicado no solo sirve para crear letras, también sirve para crear los gráficos que utilizaremos en nuestras ROM´s, ya que el procedimiento es el mismo.)

MODOS EN MSX1

SCREEN 0: texto de 40 x 24 con 2 colores
SCREEN 1: texto de 32 x 24 con 16 colores
SCREEN 2: gráficos de 256 x 192 pixeles con 16 colores
SCREEN 3: gráficos de 64 x 48 pixeles con 16 colores



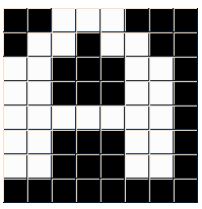
Aquí tienes muestras de diferentes sets de caracteres de varios juegos.

Letra A en Binario

```
00111000 - Byte 0
01101100 - Byte 1
11000110 - Byte 2
11000110 - Byte 3
11111110 - Byte 4
11000110 - Byte 5
11000110 - Byte 6
00000000 - Byte 7
```

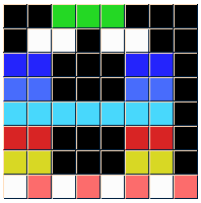
La pantalla en Screen 2 está compuesta de 256 Pixeles en horizontal por 192 pixeles en vertical pero si lo miramos en caracteres de 8x8 pixeles hablamos de 32 caracteres en horizontal por 24 en vertical.
Si multiplicamos 32x24x8 bytes que tiene cada carácter nos da un total de 6144 Bytes - 1800h son 6Kb.
Estos datos gráficos se almacenan en la Video Ram o VRAM desde la posición 0000h hasta la posición 17FFh en el mundo del MSX se le llama a esta zona Character Pattern Table.

Letra A Grafica



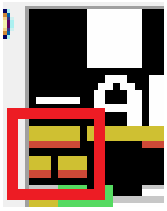
Para que lo entiendas mejor te pongo una imagen que vale más que mil palabras.
Esto es un carácter de 8x8 Pixeles son 8 bytes ya que un byte son 8 bits el ancho por 8 bytes de alto. Por eso multiplico los 32 caracteres de ancho por los 24 caracteres de alto por los 8 Bytes que tiene cada carácter. En total 6144 Bytes 1800h en Hexa.
(Puedes ver el carácter A en modo grafico y en modo binario 00111000 – Byte 0)

El color en los caracteres según el estándar del MSX1 se tiene que ajustar a un máximo de 2 colores por cada byte, un color para el fondo y otro para la tinta de un total de 16 colores disponibles. Nuevamente te pongo la misma imagen del carácter pero con una prueba de color, en el primer byte el fondo es negro y la tinta verde. El segundo byte el fondo es negro y la tinta blanca, pero si te fijas en el último byte del carácter el fondo es blanco y la tinta rosa.



Estos datos gráficos se almacenan en la VRAM desde la posición 2000h hasta la posición 37FFh en el mundo del MSX se le llama a esta zona Colour Table y ocupa lo mismo que la Character Pattern Table ósea 32x24x8 = 6144 Bytes - 1800h en Hexadecimal (Siempre procuro trabajar con esta nomenclatura)

Aclarado este tema que es bastante importante ya que en el MSX todo el tema grafico funciona con caracteres, abreviado CHR del Ingles CHaRacter. Míralo de esta manera si cada pantalla grafica ocupa 6 Kb. sin color y tenemos una ROM de 32 Kb pues nos da para 5 pantallas sin dejarnos espacio para el color, los sprites, la música, y la programación etc. Por eso todas las ROM´s o los juegos utilizan un juego de CHRs o Tiles (Azulejos) y con ellos se construye un mosaico a base de repetir Tiles por las diferentes zonas de las pantallas que iremos creando. (Vamos con las imágenes para que lo entiendas)



Como puedes ver en la pantalla del King´s Valley de Konami tenemos un carácter el de los ladrillos amarillos que se repite por diferentes zonas de la pantalla al igual que el de las escaleras, a esta técnica de utilizar caracteres repetidos para crear pantallas se le llama mapeado. La pantalla de la derecha es la Character Pattern Table junto con la Colour Table que son los gráficos que los dibujantes de Konami crearon, mientras que el mapeado o Tileado en MSX se conoce como Name Table. La imagen de la izquierda.

Estos datos se almacenan en la VRAM desde la posición 1800h hasta la posición 1AFFh esta zona ocupa 32x24 caracteres = 768 Bytes – 300h Hexa. Con esta técnica podemos crear muchas más pantallas y si comprimimos los datos de los gráficos y el mapeado pues mejor.

Ahora la parte más difícil de explicar. Si te fijas de nuevo en la pantalla de la derecha veras que los CHRs o Tiles están repetidos 3 veces en la Character pattern Table y en la Colour Table esto es debido a una limitación del MSX, sabemos que la pantalla está compuesta de 32 CHRs de ancho por 24 CHRs de alto en total 768 caracteres, pero lo que no sabemos es que a su vez está dividida en 3 tercios o bancos de 256 CHRs. Fíjate en la imagen de la derecha en su izquierda veras p0, p1, p2 estos son los 3 bancos de caracteres cada banco está compuesto de 32 CHRs de ancho x 8 CHRs de alto = 256 CHRs.

Pregunta: Entonces no puedo utilizar más de 256 CHRs porque tengo que copiarlos en los 3 bancos?

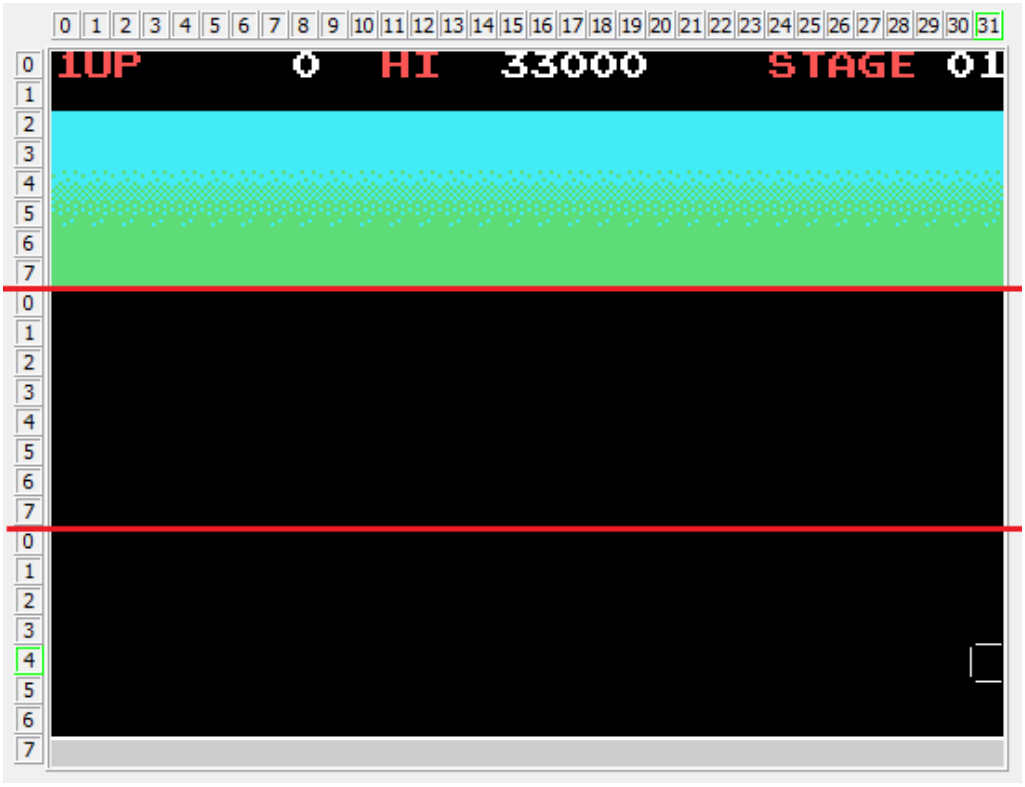
Respuesta: NO. Si un CHR se ha de emplear en toda la pantalla, si lo has de tener en los 3 bancos pero si un CHR solo lo usas en la zona del primer banco, pues solo lo tienes que colocar en ese banco, hay que ir colocando los CHRs en los bancos según la zona de la pantalla donde los tengamos que utilizar, además a medida que cambiamos de niveles podemos ir incorporando nuevos banco de CHRs o ir agregando nuevos CHRs a los bancos que ya tenemos en VRAM a medida que los necesitemos. Vamos con las imágenes que siempre nos revelan o nos aclaran más las cosas que con texto.

IMAGEN:

Banco de Caracteres.

Banco 0

Este es el primer banco de caracteres que hemos colocado en la Character Pattern Table he puesto números dentro de cada CHR para que veas como se enumeran los CHRs empieza por el 0 y termina en el 255 de arriba abajo y de izquierda a derecha de la misma manera que nosotros leemos los textos.



FILAS

COLUMNAS

1º - Tercio.

2º - Tercio.

3º - Tercio.

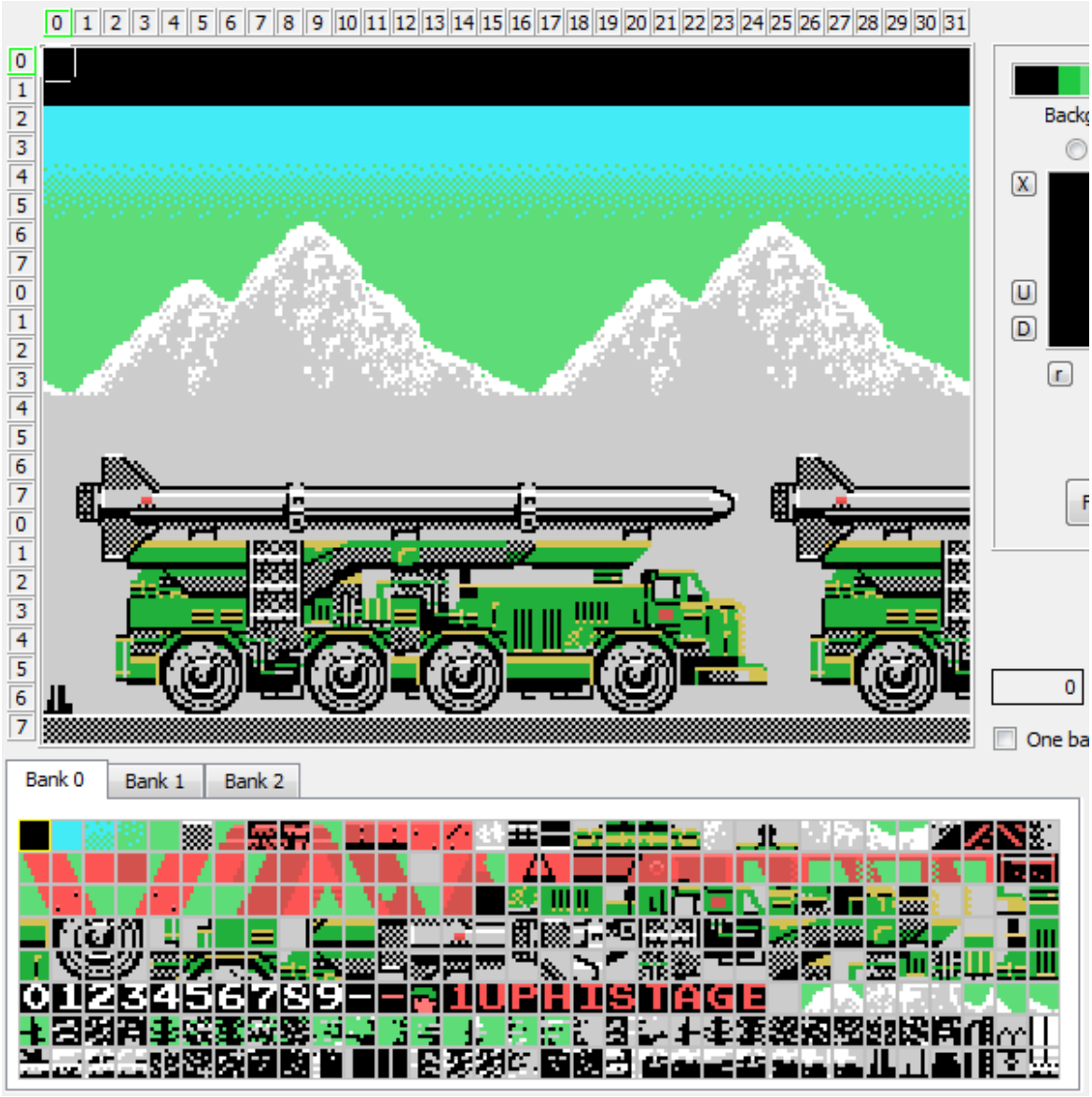
Esta es la Name Table donde creamos las pantallas con los CHRs creando nuestro mapeado, he puesto unas líneas rojas para separar los 3 tercios, como solo he colocado un banco de caracteres solo podemos construir la pantalla en el primer tercio, para poder utilizar los otros 2 tercios tendríamos que colocar CHRs en los otros 2 bancos restantes y a partir de ahí podemos construir la pantalla completa.

Ahora te enseño como se construye esta pantalla: Fíjate en la columna 0-fila 0 hay un numero 1 Fíjate en la imagen de arriba donde tenemos el banco de CHRs en qué posición tenemos ese número, es el CHR nº 6 pues en la Name Table en la Columna 0 - fila 0 escribiríamos un 6

La primera pantalla se construiría de la siguiente manera en la columna 0 – fila 0 colocamos el 6 comprueba tu mismo viendo esta tabla y las 2 imágenes de arriba para que lo comprendas mejor.

__	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
00	06	07	08	00	00	00	00	00	09	00	00	10	11	00	00	12	12	09	09	09	00	00	00	00	14	15	16	17	00	00	09	12
01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
02	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
03	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
04	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02	02
05	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03	03
06	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04
07	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04	04

Espero que todo lo hayas entendido a la perfección sino es así repásalo todas las veces que lo necesites ya que es muy importante que comprendas esta parte. Como puedes ver en las pantallas del mapeado los datos se repiten mucho y son fácilmente comprimibles ocupando muy poco espacio.



Aquí puedes ver una pantalla completa con los 3 bancos de CHRs en uso y como con pequeños CHRs se pueden construir unas pantallas con gráficos grandes. Imagen del nMSXtiles.

RESUMEN

Character Pattern Table: Ocupa 6Kb en la VRAM desde 0000h a 17FFh Abreviado CHRTBL
Aquí es donde situamos los CHRs que después usaremos para crear las pantallas.

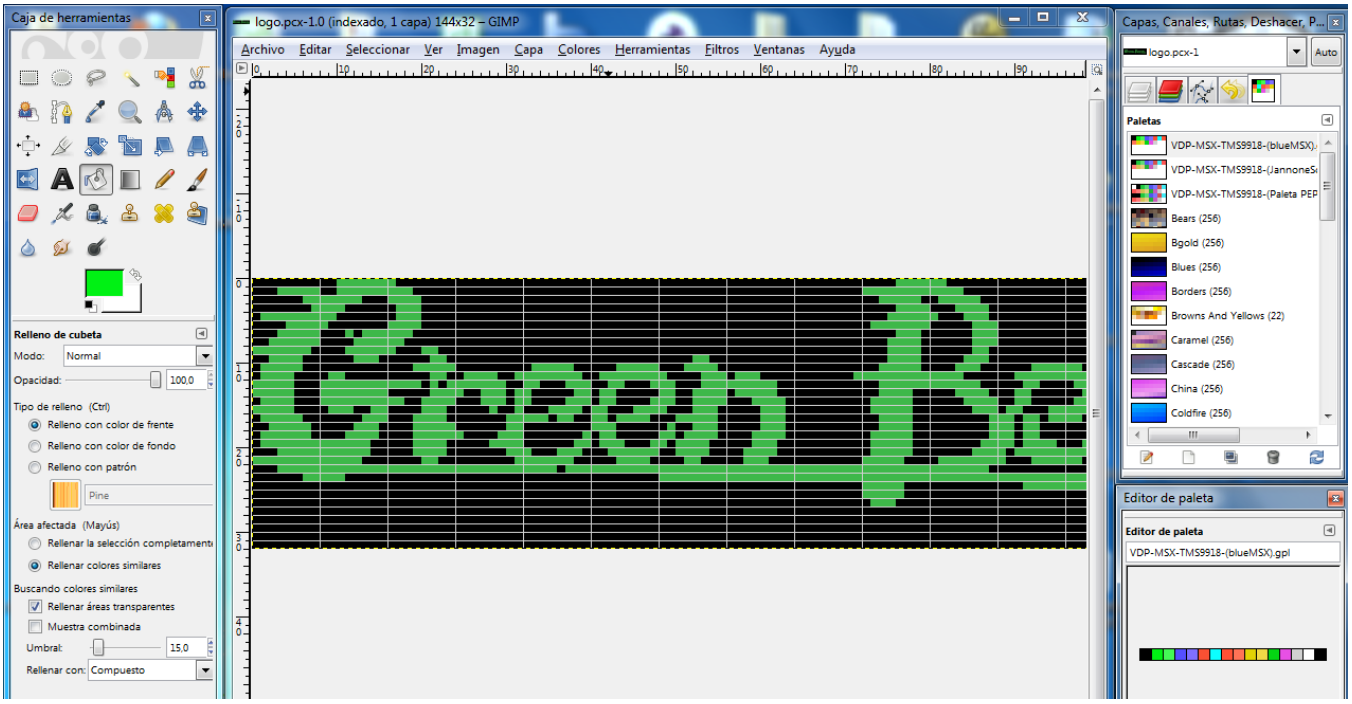
Colour Table: Ocupa 6Kb en la VRAM desde 2000h a 37FFh Abreviado CLRTBL
Aquí es donde colocamos los colores que tendrán los CHRs que tenemos en la CHRTBL

Name Table: Ocupa 768 Bytes en la VRAM desde 1800h a 1CFFh Abreviado NAMTBL
Aquí es donde situamos el mapeado con los números de CHRs que generan las pantallas.

Todos estos datos los usaremos después en el Código Ensamblador del Hola Mundo Grafico.

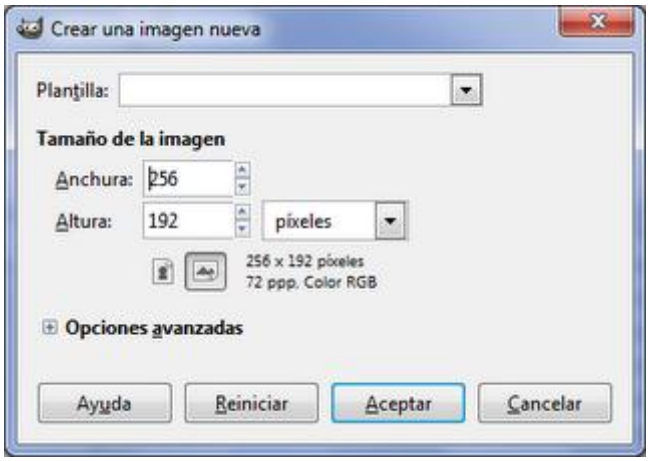
Queda la Sprite Pattern Table - SPRTBL y la Sprite Attribute Table - SPRATR que veremos más adelante en otra entrega del tutorial dedicada al mundo de los Sprites.

Lo primero que necesitamos para empezar es un programa de dibujo que nos permita realizar los gráficos que después incorporaremos a nuestro código. Hay muchas opciones donde escoger yo personalmente me quede con el GIMP ya que es un programa gratuito tanto para Windows como Linux, muy completo. Página oficial para descargar el programa. http://www.gimp.org Bájatelo e instálalo.



Para configurar el GIMP de manera correcta para crear gráficos para el MSX, tenemos que configurar la resolución, la paleta de colores, y un Grid apropiado y grabar el fichero de imagen en formato PCX para después utilizarlo con otras herramientas. Fantástico manual de aOrante que me permito reproducir.

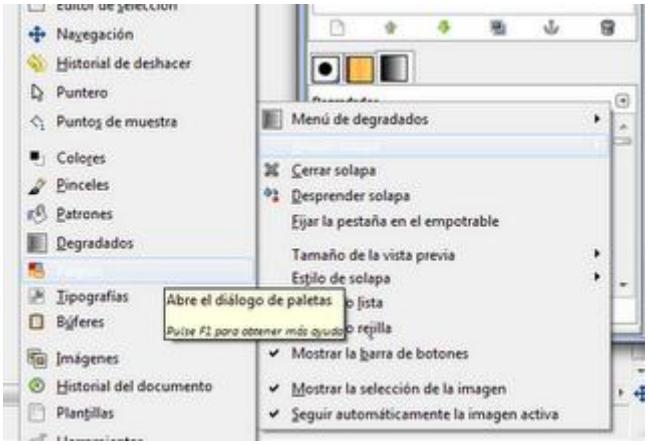
2ª Parte:
aOrante----------
Una vez que tengamos abierto el programa, lo primero que haremos es crear una nueva imagen y para ello nos iremos al menú "*Archivo*" de la ventana principal y accionaremos la opción "*Nuevo*". Podemos acceder más rápidamente pulsando [CTRL]+[N]. Se nos mostrará una ventana donde nos pedirá el tamaño de la imagen y que le daremos 256 de anchura y 192 de altura.



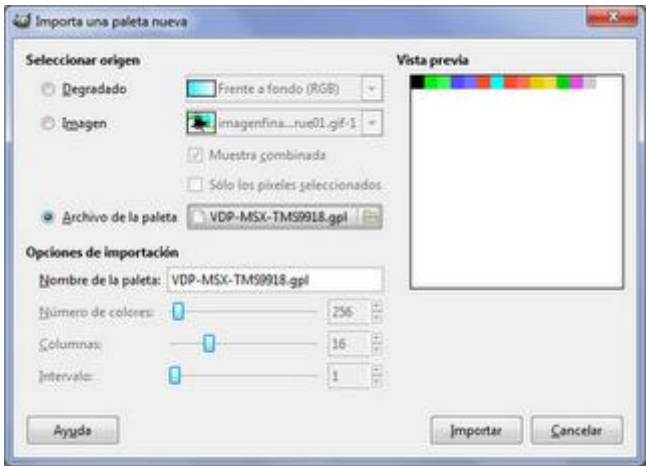
NOTA: Si vamos a utilizar a menudo Gimp para dibujar en este formato, podemos crear una plantilla desde "*Archivo>Guardar como plantilla*". Cuando empecemos una imagen, solo tendremos que indicar nuestra plantilla, desde el selector que se encuentra en la parte superior de la ventana para crear una nueva imagen.

El siguiente paso, será indicarle la cantidad de colores y la paleta que usaremos. Para nuestro caso necesitaremos que sea de tipo indexado de 16 colores con la paleta del VDP de los MSX. Para ello, necesitaremos disponer de la paleta. Podemos crearla nosotros desde "Ventanas>Diálogos empotrables>Paletas", pero para simplificaros la tarea he creado dos que podéis descargar desde este artículo. Una es la paleta que utilizan los emuladores blueMSX/OpenMSX y la otra es la del MSX Screen Conversor de Jannone, útil para cuando utilicemos esta aplicación. (Las paletas están al final para bajar)

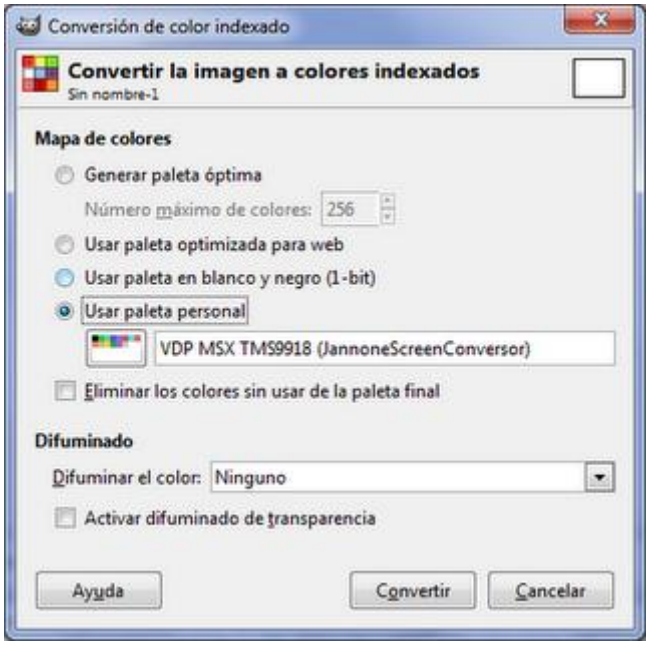
Para cargarla, antes recomiendo los siguientes pasos para cambiar la configuración de la ventana de diálogos empotrables (la que por defecto se muestra a la derecha). Primero añadiremos una nueva solapa con el gestor de paletas. Hay dos formas, abrirla desde el menú "*Ventanas>Diálogos empotrables>Paletas*" y arrastrándola directamente a la zona inferior de la ventana. La segunda opción será pulsando ese pequeño botón con una flecha apuntando a la izquierda, situado en la parte superior derecha donde se encuentran las solapas. Se nos mostrará un menú y seleccionaremos la opción añadir solapa, donde encontraremos una llamada "*Paleta*".



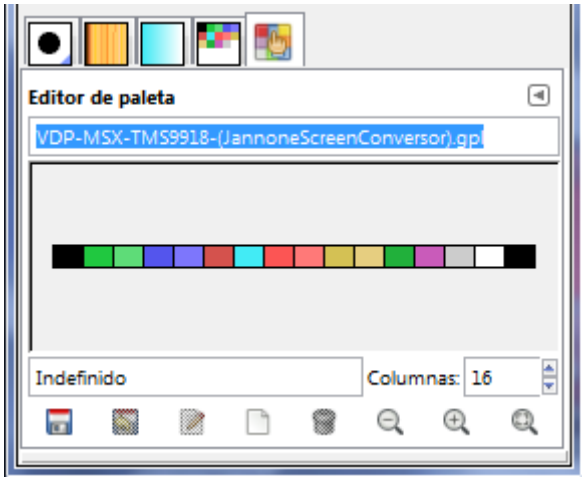
Una vez la tengamos insertada, pulsaremos de nuevo el pequeño botón, para acceder a la primera opción "*Menú de paletas*" y desde este seleccionaremos la opción "*Importar paleta...*". Luego utilizaremos la opción "Archivo de la paleta", para cargar la paleta.



Ahora que ya tenemos la paleta, ya podemos indicar el tipo de imagen. Seleccionaremos "*Imagen>Modo>Indexado*". Se nos mostrará una ventana, Donde activaremos la opción "Usar paleta personal" y pulsaremos en el botón que hay en la línea inmediata, para seleccionar la paleta.
Importante: Inhabilitaremos el control que indica "*Eliminar los colores sin usar de la paleta final*" ya que de lo contrario, nos será imposible pintar en la pantalla.



El siguiente paso será poder acceder a nuestra paleta de colores y para ello, tendremos que añadir una solapa con el editor de paleta. Esto se consigue situándonos con el cursor encima de nuestra paleta y con el botón derecho abriremos el menú contextual donde pulsaremos sobre la opción de "*Editar paleta*". Se nos abrirá una ventana, con nuestros colores, que aconsejo incrustar junto a la solapa del selector de paletas.



NOTA: Las modificaciones de la interfaz de usuario de Gimp, se mantienen al iniciar una nueva sesión, por lo que no deberemos repetir estos pasos de configuración, a excepción de la paleta, que se cambia por la "default".

Para ayudarnos a respetar las características del modo de pantalla del Screen2 (2 colores cada 8 pixeles), nos ayudaremos utilizando la rejilla. Antes de activarla, modificaremos su configuración mediante la opción "*Imagen>Configurar la rejilla*". Se nos mostrará una ventana donde cambiaremos los valores de "*Espaciado*", primeramente pulsando el icono de cadena, para inhabilitar la proporción, para luego indicar en la "*Anchura*" 8 y en "*Altura*" 1. Ahora toca activarla. Nos iremos al menú "*Ver*" y activaremos la opción "*Mostrar Rejilla*".

Para poder ver correctamente la rejilla, tendremos que trabajar visualizando la imagen con un mínimo de un 400%. Esto se ajusta desde "*Ver>Ampliación*", o en el marco inferior de la ventana principal. Otra forma de trabajar será utilizando una rejilla de 8x8.

Ahora ya podemos dibujar. Una vez terminado y guardado el fichero, lo siguiente es tratar esos datos y eso lo explicaremos más adelante.

> Paleta para Gimp (basada en la del BlueMSX modificada por PEPE)
> Esta paleta la tenéis en el fichero grafico.rar de este tutorial.
> Fichero VDP-MSX-TMS9918-(Paleta PEPE).gpl

aOrante----------

En el manual de aOrante habla de crear una pantalla de presentación de 256x192 pero podemos crear el tamaño que queramos dependiendo del grafico o gráficos que vallamos a crear sin superar los 256x192, como puedes ver en mi primera captura para crear el logo del Green Beret especifico una resolución de 144 x 32 ya que solo quiero crear ese grafico para incorporarlo al código.

Vamos con la creación del set de caracteres que vamos a utilizar para el hola mundo grafico.

	000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	` 96	p 112
0001 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0010 2	STX 2	DC2 18	“ 34	2 50	B 66	R 82	b 98	r 114
0011 3	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0100 4	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
0101 5	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0110 6	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0111 7	BEL 7	ETB 23	’ 39	7 55	G 71	W 87	g 103	w 119
1000 8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1001 9	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1010 A	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1011 B	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1100 C	FF 12	FS 28	´ 44	< 60	L 76	\ 92	l 108	| 124
1101 D	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1110 E	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
1111 F	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	DEL 127

Tabla del código ASCII 7 bits

Según la tabla ASCII tienes que crear el set de caracteres o letras ciñéndote a su normativa.

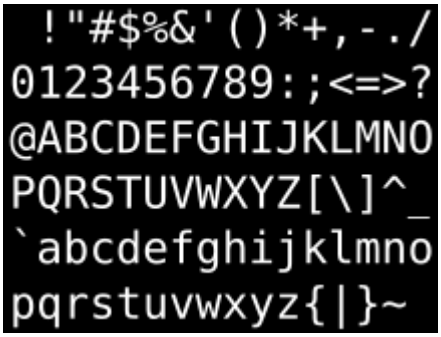
Hay que empezar con el carácter 32 te lo remarco en color verde que es el espacio en blanco, y terminarlo en el carácter 127. Como veras en la imagen del set de caracteres que he dibujado a mano con el Gimp el orden de cada carácter esta creado según la normativa de la tabla ASCII.

32–Espacio, 33–Admiración, 34–Comillas, 35–Hash, 36–Dólar, 37–Porciento, Etc. Etc. hasta el 127.

Como puedes ver aquí cada letra se corresponde con un número de Carácter. De tal manera que cuando escribimos una letra “A” Mayúsculas remarcada en verde nuestro ordenador coloca un 65.

La explicación de crearlo de esta manera es porque después en nuestro código en ensamblador, pondremos los textos que vamos a usar en el tutorial en ASCII.

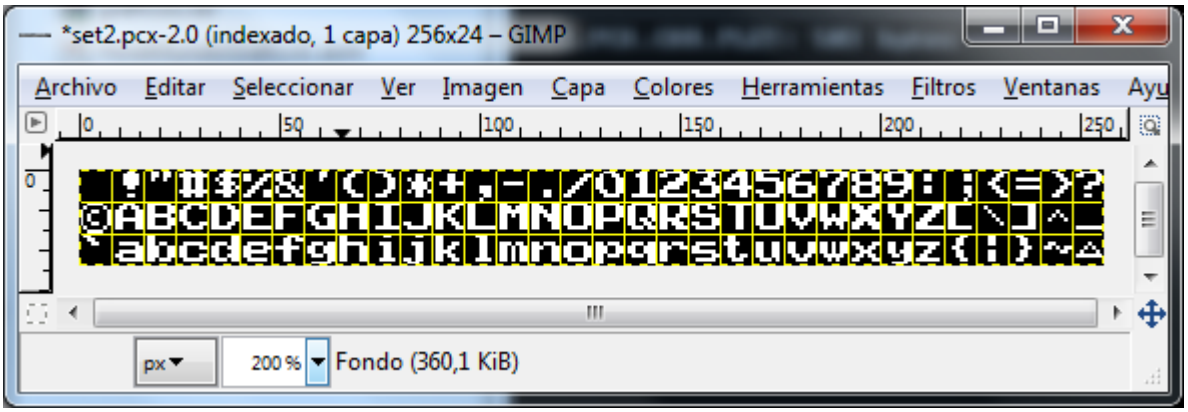
A la hora de situar los CHRs de nuestro set de letras en la VRAM empezaremos colocando el primer CHR en la posición 32 de la CHRTBL, para que coincida con la numeración que tienen en la tabla ASCII, de esta manera colocando en la NAMTBL un texto en ASCII coincidirá con el mismo número de carácter que tiene en la Character Pattern Table - CHRTBL.
Como nuestro primer grafico es el eSPacio en el set de caracteres que hemos creado y su ASCII es el 32 es en este número donde tenemos que empezar a colocar los CHRs en la CHRTBL en la VRAM.



Estos son los 95 caracteres imprimibles ASCII en su orden correcto empieza con el carácter 20h o 32 que es el espacio (SP) seguido de símbolos con los números el alfabeto en Mayúsculas y el alfabeto en Minúsculas con algún que otro símbolo entremezclado.

Aquí tienes el fichero Tutorial3.rar con los ejemplos para este tutorial, incluidos los gráficos código fuente y demás programas.

http://www.dimensionzgames.com/wp-content/uploads/downloads/2012/01/Tutorial3.rar

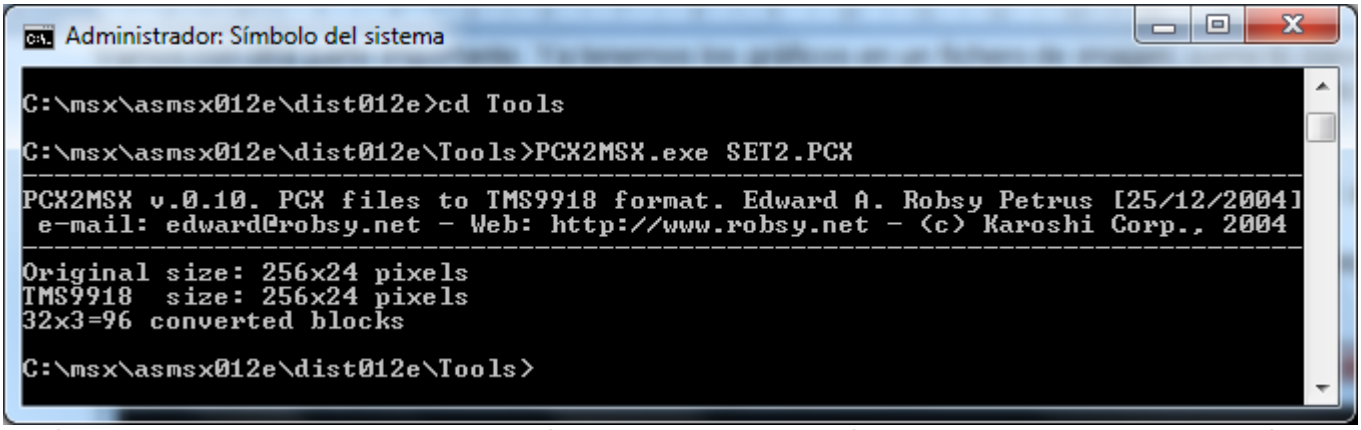


Aquí puedes ver como he dibujado el set de caracteres con el GIMP, puedes crearte el tuyo propio o abrir el fichero SET2.PCX que yo he realizado para el ejemplo. (Aunque es mejor que vayas haciendo algo con el Gimp para que te vayas familiarizando.) He creado una nueva imagen con el tamaño de 256 que es el ancho total de la pantalla en screen2 por 24 de alto que son las 3 líneas de caracteres que necesito, y el Grid de 8x8 pixeles. (El resultado el que ves en la foto. Nota: no empleo color. Ya que este se puede poner por código mas tarde.) Fíjate como he creado el set de caracteres por si te animas a crear el tuyo propio, no has de usar los 8x8 pixeles en total deja una línea libre debajo y otra a la izquierda, para que cuando pintes letras seguidas estas no se toquen unas con otras y se mezclen.

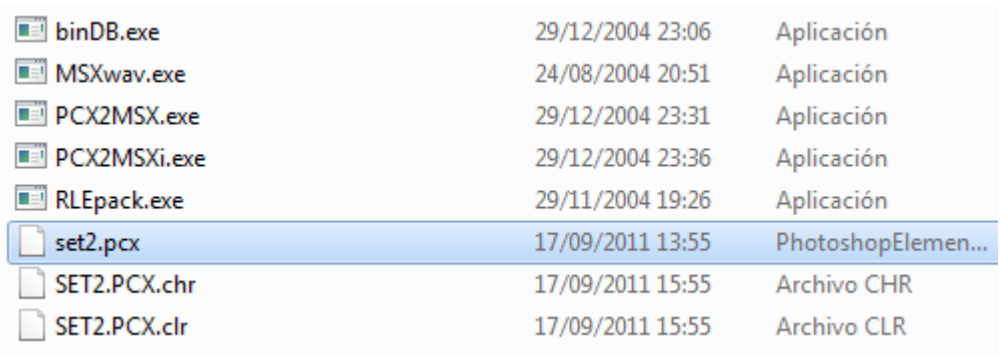
Vamos con otra parte muy importante. Ya tenemos los gráficos en un fichero de imagen, como los incorporo a nuestro código para después compilar con el ensamblador asMSX. Todas estas funciones las vamos a realizar desde el MS-DOS o Símbolo del sistema. Copia el fichero SET2.PCX a la ruta donde vayas a crear el código en mi caso en C:\msx\asmsx012e\dist012e\Tools\
En el pack-MSX.rar de la primera parte del tutorial descomprimimos el asMSX 0.12e en C:\MSX abre desde Windows botón Inicio – Todos los programas – Accesorios – Símbolo del sistema
Ahora teclea, cd msx intro, cd asmsx012e intro, cd dist012e intro, cd Tools intro. Fíjate en la imagen que te pongo debajo de este texto.

Ahora vamos a convertir la imagen PCX a fichero BINARIO, para posteriormente comprimir los datos con el compresor. Y con el programa binDB.exe convertir el fichero BINARIO a TEXTO en formato DB´s para nuestro ensamblador. (La necesidad de comprimir los datos…, el espacio en ROM puedo parecer muy grande pero los gráficos se llevan el 60% del tamaño de nuestra ROM) Estas 3 herramientas nos las proporciona nuestro querido maestro E. Robsy, pero en mi caso comprimo los datos con PLETTER del grupo XL2S Entertainment ya que la compresión que realiza es muchísimo más intensa que la que realiza el RLE. Teclea PCX2MSX.exe SET2.PCX. Pulsamos intro.

(Tú puedes utilizar el método de compresión que prefieras, como BITBUSTER, MSX-o.-Mizer etc. Lo que realmente hay que valorar es la velocidad, que la compresión sea buena y rutina con pocos bytes)

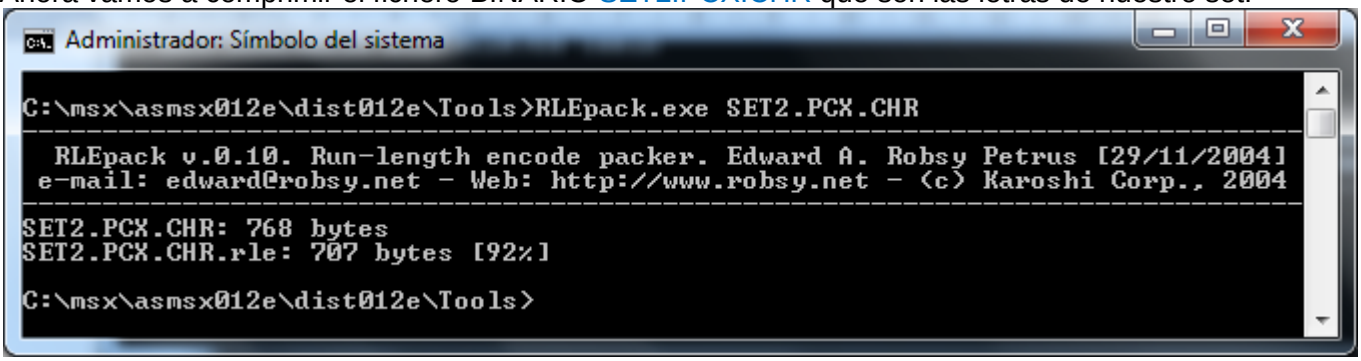


Aquí tienes el resultado de que la operación se ha realizado con éxito, cuando trabajemos con gráficos o CHRs que tengan color presta atención al siguiente ERROR Color collision at line (X,Y) Esto quiere decir que hemos usado más de 2 colores por byte, Indicándote en que Línea y Columna esta el error.



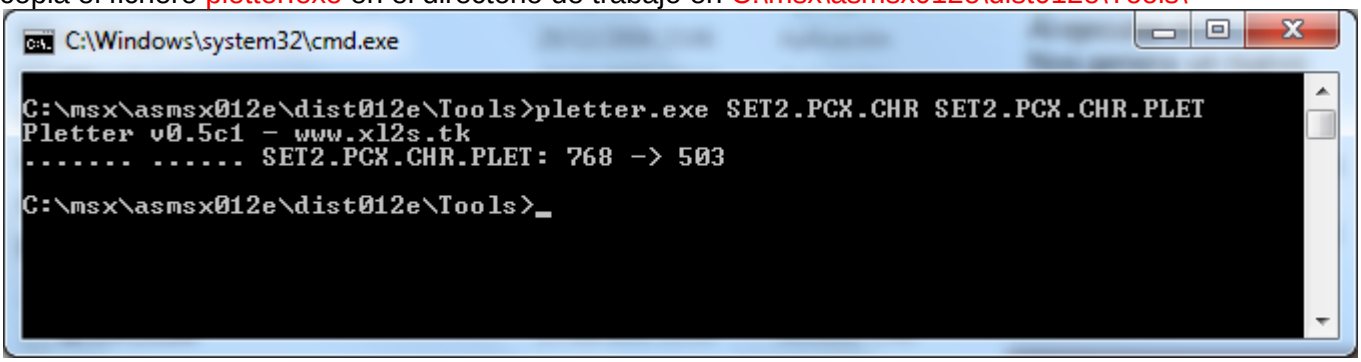
Aquí puedes ver cómo ha generado los dos ficheros binarios.
SET2.PCX.chr que contiene los caracteres y SET2.PCX.clr que contiene los colores de los caracteres que no vamos a utilizar.

Ahora vamos a comprimir el fichero BINARIO SET2.PCX.CHR que son las letras de nuestro set.

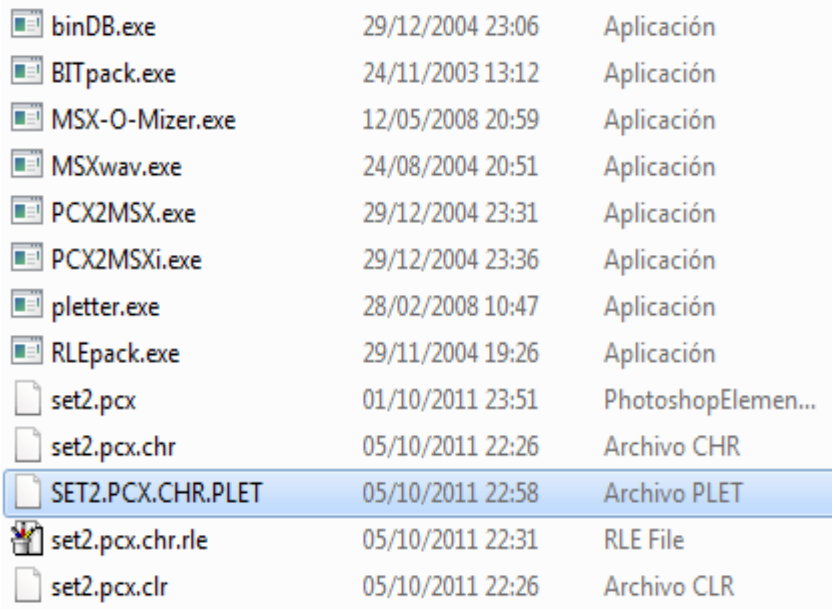


Aquí puedes observar que solo ha reducido su tamaño un 8% ya que el algoritmo de compresión del RLE es muy pobre y comprime muy poco los datos. Esto solo te lo muestro para que lo veas.

Ahora para que veas la diferencia voy a comprimir con PLETTER Ver. 5c1- usando pletter.exe de XL2S En los ejemplos de este tutorial en el Tutorial3.rar dentro esta el fichero pletter5c1.rar descomprímelo y copia el fichero pletter.exe en el directorio de trabajo en C:\msx\asmsx012e\dist012e\Tools\



Teclea pletter.exe SET2.PCX.CHR SET2.PCX.CHR.PLET. Aquí puedes ver la diferencia en RLE los 768 bytes se quedan en 707 bytes mientras que con PLETTER los 768 bytes se quedan en 503 Bytes.



Hemos Ganado 265 Bytes en la compresión. Solo en esto.

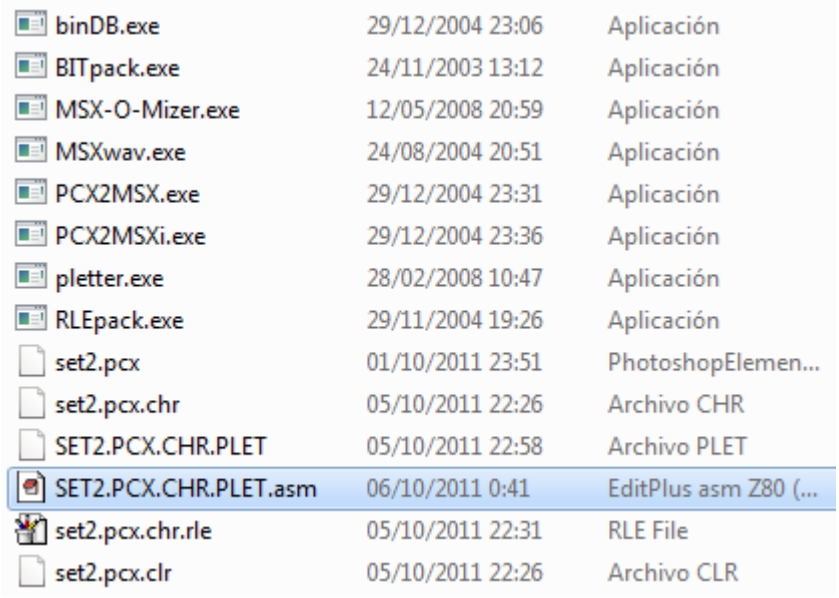
Al ejecutar el RLEpack.exe
Nos genera un nuevo fichero acabado en .rle

Mientras que en pletter le decimos el nombre del fichero a comprimir SET2.PCX.CHR y el nombre que queremos darle al fichero comprimido yo le doy el mismo nombre pero le añado la extensión .PLET generándonos un nuevo fichero SET2.PCX.CHR.PLET

Vamos con el .PLET que es el de nuestro tutorial.

Ahora vamos a convertir el fichero SET2.PCX.CHR.PLET que ya es un binario comprimido a texto en formato DB´s para incorporarlo a nuestro nuevo proyecto. Teclea binDB.exe SET2.PCX.CHR.PLET

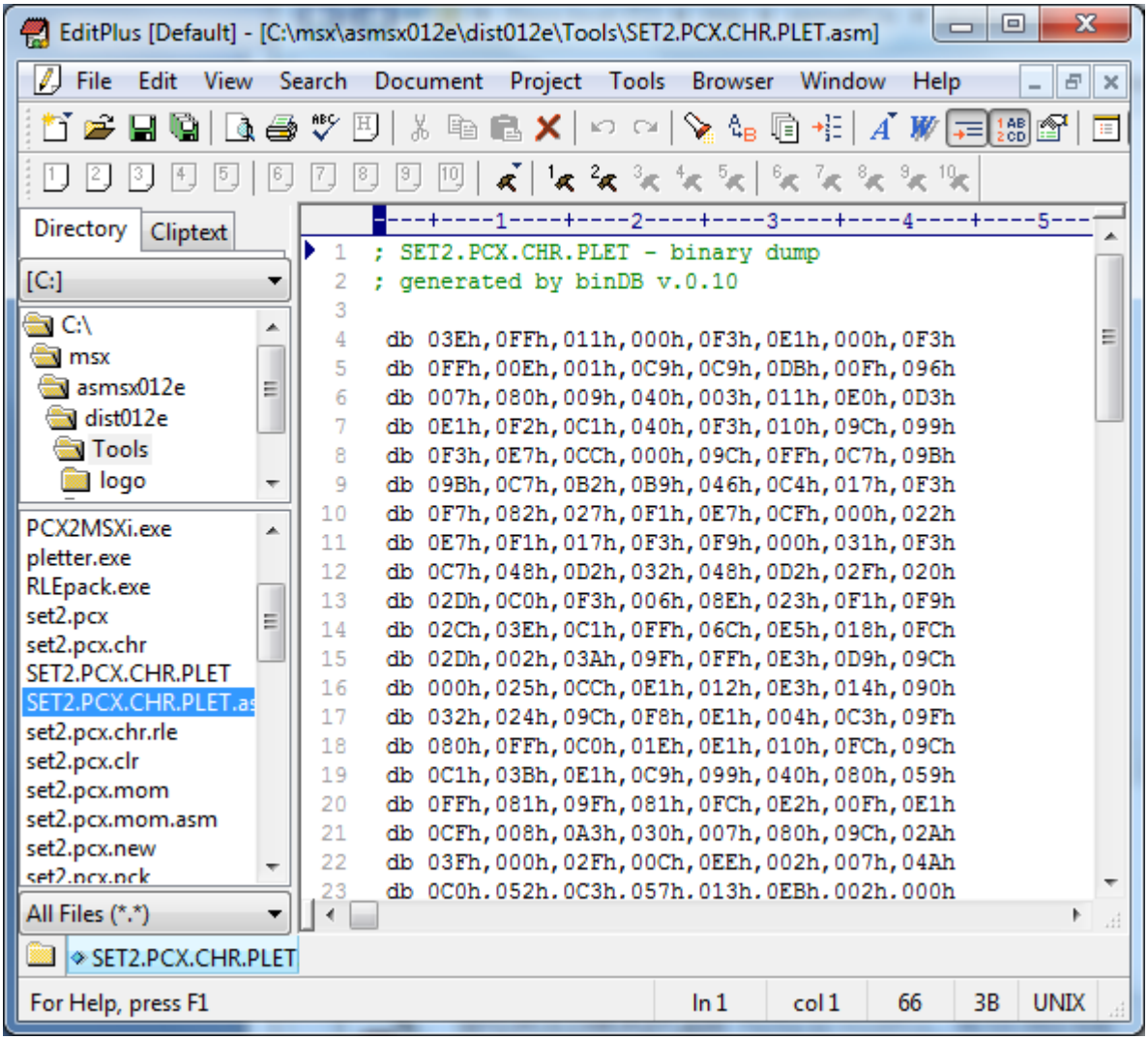
Este es el resultado final ahora tenemos un nuevo fichero llamado SET2.PCX.CHR.PLET.asm



Aquí puedes ver el fichero acaba en .asm y está asociado para que se abra con el EditPlus.

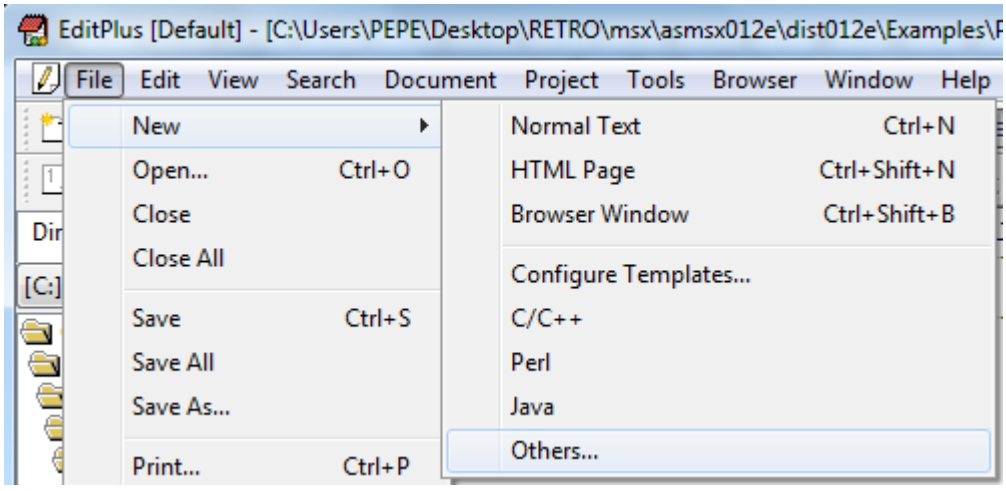
Pulsa doble clic sobre el fichero y veras que se abre en nuestro editor listo para copiarlo y pegarlo a nuestro nuevo proyecto.

El tema de tener tantas extensiones es para saber de qué fichero de imagen parte todo el proceso. SET2.PCX es la IMAGEN .CHR que son los caracteres, .PLET es el fichero ya comprimido con pletter y .asm es el fichero en formato DB´s listo para nuestro ensamblador.
Pulsa doble clic sobre este fichero para que se abra con el EditPlus 3.



Aquí tienes el set de caracteres en formato DB´s para el ensamblador, esto son los bytes de los CHRs comprimidos que lo forman, y que veremos más adelante en este tutorial como tratarlos.

Tercera parte:
El código del Hola Mundo Grafico. Te atreves a empezar tu mismo la primera parte del código en ensamblador? Ya sabes una ROM, etc etc… como vimos en el primer y segundo manual.



Pulsa en el menú File – New – Others… -

asm Z80 y botón OK

Después en File – Save As… y lo guardas en el directorio que quieras y con el nombre que quieras. Yo le pongo HolaMundoGrafico1.asm

Seguimos teniendo el grafico en otra pestaña.

Vamos con el código del Hola Mundo Grafico
Fichero. “HolaMundoGrafico1.asm” está dentro del fichero Tutorial3.rar

```
;--------------------------------------------------------
; Nombre de nuestro programa
; Hola Mundo Grafico - 24/09/2011
; Versión 1
;--------------------------------------------------------

;--------------------------------------------------------
; CONTANTES
;--------------------------------------------------------
; No definimos ninguna constante

;--------------------------------------------------------
; VARIABLES DEL SISTEMA
;--------------------------------------------------------
; Aquí definimos con nombres las direcciones en VRAM como te he explicado en la teoría de la primera parte del tutorial.
; Direcciones de la VRAM
       CHRTBL         equ     0000h  ; Tabla de caracteres
       NAMTBL         equ     1800h  ; Tabla de Nombres
       CLRTBL         equ     2000h  ; Tabla del color de los caracteres
       SPRATR         equ     1B00h  ; Tabla de los atributos de los sprites
       SPRTBL         equ     3800h  ; Tabla de Sprites
; Variables del Sistema MSX
       CLIKSW         equ     $F3DB  ; Keyboard click sound
       FORCLR         equ     $F3E9  ; Foreground colour

;--------------------------------------------------------
; DIRECTIVAS PARA EL ENSAMBLADOR ( asMSX )
;--------------------------------------------------------
       .bios          ; Definir Nombres de las llamadas a la BIOS
       .page 2        ; Definir la dirección del código irá en 8000h
       .rom           ; esto es para indicar que crearemos una ROM
       .start INICIO  ; Inicio del Código de nuestro Programa
; Seguir la norma del Standard MSX para una ROM
        dw 0,0,0,0,0,0; 12 Ceros

;--------------------------------------------------------
; INICIO DEL PROGRAMA
;--------------------------------------------------------
INICIO:
       call    INIT_MODE_SCx  ; iniciar el modo de pantalla x
       call    INIT_GRAFICOS  ; colocar los gráficos en VRAM
FIN:
       jp      FIN            ; esto es como 100 goto 100 para que se quede en un bucle sin fin.

;--------------------------------------------------------

; Todo esto lo hemos visto en el tutorial anterior así que no necesita muchas mas explicaciones
; La única diferencia es que aquí usamos el SCREEN 2, por eso activo la opción - call  INIGRP
; Esta rutina os sirve para activar cualquiera de estos modos descritos en los textos a modo de comentario.

;--------------------------------------------------------
; INICIALIZA EL MODO DE PANTALLA Y COLORES
;--------------------------------------------------------
; BASIC: COLOR 15,1,1
; Establecer los colores

INIT_MODE_SCx:
       ld      hl,FORCLR      ; Variable del Sistema
       ld      [hl],15        ; Color del primer plano 15=blanco
       inc     hl             ; FORCLR+1
       ld      [hl],1         ; Color de fondo 1=negro
       inc     hl             ; FORCLR+2
       ld      [hl],1         ; Color del borde 1=negro
; call INITXT          ; BIOS set SCREEN 0
; call INIT32          ; BIOS set SCREEN 1
       call    INIGRP         ; BIOS set SCREEN 2
; call INIMLT          ; BIOS set SCREEN 3
;
; SCREEN 0 : texto de 40 x 24 con 2 colores
; SCREEN 1 : texto de 32 x 24 con 16 colores
; SCREEN 2 : gráficos de 256 x 192 pixeles con 16 colores
; SCREEN 3 : gráficos de 64 x 48 pixeles con 16 colores

       ret
;--------------------------------------------------------
```

```
;-------------------------------------------------------
; COLOCA LOS GRAFICOS EN LA VRAM
;-------------------------------------------------------
INIT_GRAFICOS:
; Esto lo realizamos para que no se vea nada en la pantalla
; Mientras colocamos los CHRs y los Colores en la VRAM
        call    DISSCR          ; BIOS deshabilitar la pantalla

; Esto es para quitar el sonido que emite el msx cuando se pulsa una tecla
        xor     a               ; ld a,0
        ld      [CLIKSW],a      ; Variable BIOS desactivar sonido teclas

; Borrar los 768 Bytes de la Name Table - NAMTBL en VRAM
; Originalmente la BIOS rellena la NAMTBL con  los valores del 0 al 255 en cada tercio por eso lo relleno
todo con 0
        ld      hl,NAMTBL       ; Dirección origen en VRAM
        ld      bc,768          ; nº de CHRs a rellenar
        xor      a              ; a=0 - Valor a rellenar
        call    FILVRM          ; BIOS –Fill block of VRAM with data byte ; Te explico más abajo el
funcionamiento de FILVRM
```

Aquí entra en acción algo que todavía no te he explicado en anteriores tutoriales, esta parte es la encargada de colocar las letras que hemos creado, y que tenemos comprimidas en formato DB´s en la VRAM, te pongo un fragmento debajo de este texto para que lo entiendas.

```
        ;-----------------------------------
        ; SET2.PCX.CHR.PLET - binary dump                   ; Este es el fichero generado por el programa binDB.exe
        ; generated by binDB v.0.10                         ; obtenido de la imagen PCX y comprimido con PLETTER
        ;-----------------------------------                ; debes copiar el texto y pegarlo en tu código al final del todo
        SET2_PLET:                                         ; Ahora tienes que colocar una etiqueta para que apunte al
         db 03Eh,000h,011h,000h,018h,03Ch,000h,018h        ; principio de los datos de nuestro set de caracteres.
         db 000h,00Eh,001h,06Ch,06Ch,048h,00Fh,096h        ; Yo escribo.   SET2_PLET:
         db 007h,0FEh,009h,000h,003h,000h,030h,07Ch        ; Justo debajo de los comentarios del binDB. Para después
         db 0B0h,078h,034h,0F8h,040h,030h,010h,0C6h        ; por código saber donde empiezan estos datos.
                ……..
```

Si no quieres que tu código en ensamblador sea tan grande como cuando creas un videojuego, puedes omitir crear el fichero que ves encima de este texto con el bindDB.exe y colocar el fichero binario que has comprimido con el compresor directamente, de la siguiente manera.

```
        ;-----------------------------------------------
        ; Set de caracteres BINario comprimido con PLETTER
        SET2_PLET:
                .INCBIN         "SET2.PCX.CHR.PLET"
```

Este es el código que irá colocando los bytes ya descomprimidos de nuestros set de caracteres en la VRAM en la Character Pattern Table .

```
; Colocar los gráficos de las letras en VRAM en la CHRTBL
        ld      hl,SET2_PLET            ; set de CHRs de las letras - Origen
        ld      de,CHRTBL+(32*8)        ; Empezar en el CHR 32 - Destino
        call    DEPLET                  ; Descomprimir en VRAM
```

La rutina DEPLET está situada en nuestro código y tiene unos parámetros de entrada, que son los que te describo a continuación:
El registro HL tiene que apuntar a la dirección de memoria donde empiezan los bytes de nuestros datos comprimidos.
En el ejemplo que nos ocupa apunta a la dirección SET2_PLET que es donde están los bytes comprimidos del Set de Caracteres.

```
        ld      hl,SET2_PLET            ; set de CHRs de las letras
```

El registro DE tiene que apuntar a la dirección en VRAM donde queremos descomprimir los bytes.
Quiero empezar a colocar bytes en el CHR 32 de la CHRTBL en VRAM por eso le sumo a CHRTBL, 32 x 8 bytes que tiene cada carácter
La explicación de colocar el set de caracteres a partir del CHR 32 la explico con más detalle en la página 9 de este tutorial.

```
        ld      de,CHRTBL+(32*8)        ; Empezar en el CHR 32
```

Esta es la llamada a la rutina que toma los bytes comprimidos y los descomprime en VRAM según los valores pasados en HL y DE
Empieza a tomar bytes desde SET2_PLET los descomprime y los coloca en la Character Pattern Table - CHRTBL a partir del CHR 32.

```
        call    DEPLET                  ; Descomprimir en VRAM
```

```
; Colocar el color de las letras en VRAM en la CLRTBL
        ld      hl,CLRTBL+(32*8)        ; Empezar en el CHR 32 de la CLRTBL
        ld      bc,(32*24)              ; numero de CHRs
        ld      a,0Fh                   ; Valor a rellenar
        call    FILVRM                  ; BIOS –Fill block of VRAM with data byte
```

La rutina FILVRM está situada en la BIOS y tiene unos parámetros de entrada, que son los que te describo a continuación:
El registro HL tiene que apuntar a la dirección de memoria de la VRAM donde queremos empezar a colocar los bytes
En el ejemplo que nos ocupa queremos empezar a colocar bytes en el CHR 32 de la Colour Table – CLRTBL

```
        ld      hl,CLRTBL+(32*8)        ; Empezar en el CHR 32 de la CLRTBL
```

En el registro BC le decimos cuantos son los bytes que hay que rellenar.
El set de caracteres ocupa 32 CHRs de ancho por 3 CHRs de alto, pero recuerda que cada CHR tiene de alto 8 bytes, entonces hay que multiplicar 3 x 8=24 bytes de altura, por eso en el registro BC pongo 32 x 24

```
        ld      bc,(32*24)              ; numero de CHRs
```

En el registro A le decimos el byte que tiene que usar para rellenar toda la zona seleccionada.
En el ejemplo que nos ocupa el Color de la tinta es blanco F y el fondo negro 0, o lo que es lo mismo 0Fh en hexa. 16 Colores de fondo y 16 colores de tinta, recuerda que los colores empiezan con el 0 y terminan en el 15 si miras la paleta del MSX veras que el blanco es el 15 y el negro el 0 - 15 en hexadecimal es F y el negro 0, resultado 0Fh si quieres cambiar colores ya sabes cómo tienes que hacerlo.


```
        ld      a,0Fh                   ; Valor a rellenar
```

Esta es la llamada a la rutina que empezara escribiendo el byte 0Fh en VRAM empezando en CLRTBL+256 y no parara hasta que se rellenen 32x24=768 Bytes que son las 3 filas de 32 CHR que ocupa el set de caracteres en la Color Table - CLRTBL

```
        call    FILVRM                  ; BIOS -Fill block of VRAM with data byte
```

```
; Colocar la cadena de texto en la pantalla
; LOCATE 6,2: PRINT "Hola Mundo Grafico"
        ld      hl,TXT_HOLA             ; Dirección donde tenemos  el texto
        ld      de,NAMTBL+6+(2*32)      ; LOCATE 6,2 : Destino la NAMBTL en VRAM
        ld      bc,18                   ; Numero de CHR que tiene el texto
        call    LDIRVM                  ; BIOS - Copy block to VRAM, from memory
```

La rutina LDIRVM está situada en la BIOS y tiene unos parámetros de entrada, que son los que te describo a continuación:
El registro HL tiene que apuntar a la dirección de memoria RAM-ROM donde tenemos los bytes que queremos llevar a la VRAM
En el ejemplo que nos ocupa donde tenemos la cadena de texto en ASCII que queremos llevar a la VRAM

```
        ld      hl,TXT_HOLA             ; Dirección donde tenemos el texto
```

El registro DE tiene que apuntar a la dirección en VRAM donde queremos colocar los bytes que están en RAM-ROM
Quiero empezar a colocar bytes en la Name Table – NAMTBL en la Fila 2 – Columna 6 de la pantalla
Por eso le sumo a la dirección. NAMTBL la fila nº 2 x32 CHR que tiene cada fila y le añado los 6 de la Columna

```
        ld      de,NAMTBL+6+(2*32)      ; LOCATE 6,2
```

En el registro BC le decimos el número de bytes que transferimos de RAM-ROM a VRAM
En el ejemplo que nos ocupa 18 bytes que son los CHR que ocupa el texto “Hola Mundo Grafico”=18 letras.

```
        ld      bc,18                   ; Numero de CHR que tiene el texto
```

Esta es la llamada a la rutina que empezara a trasferir 18 bytes desde la RAM-ROM hacia la VRAM concretamente en la NAMTBL+70
En el ejemplo que nos ocupa el valor de los códigos ASCII del texto, los situara en la Name Table - NAMTBL

```
        call    LDIRVM                  ; BIOS - Copy block to VRAM, from memory
```

```
; Esto lo realizamos para que se muestre de nuevo la pantalla.
        call    ENASCR                  ; BIOS habilitar la pantalla

; Salir de la rutina INIT GRAFICOS
        ret
;-----------------------------------------------------------
```

Vamos con el texto para nosotros es más fácil poner db “HOLA MUNDO Grafico” que tener que ver la tabla ASCII de la pagina 9 y mirar los valores que corresponden a cada letra, y tener que poner db 72,79,76,65,32,77,85,78,68,79,32,71,114,97,102,105,99,111 pero esta labor la hace por nosotros el asMSX a la hora de compilar, ahora eso si en la NAMTBL los valores que se escribirán serán los valores ASCII, repasa la pagina 9 de este tutorial que explica este apartado con más detalle.

```
; Cadena de Texto a visualizar en la pantalla
TXT_HOLA:
        db      "HOLA MUNDO Grafico"
```

Esta es la rutina encargada de la descompresión de bytes en VRAM en mi caso yo he escogido el sistema de compresión PLETTER, pero el mercado hay un gran elenco de compresores-descompresores como RLE, BITBUSTER,MSX-o-Mizer, Exomizer, etc. Tu puedes seleccionar el sistema de compresión que más se adapte a tus necesidades cambiando esta rutina por el código del sistema de compresión que tu hayas escogido, y comprimir el fichero binario de tu imagen con el compresor que te suministren.

Aquí tienes la rutina del descompresor Pletter v0.5b de RAM-ROM a VRAM directamente. Si quieres puedes estudiar su código para saber cómo funciona o simplemente utilizarla, ya te he explicado anteriormente que parámetros de entrada debes de proporcionarle a la rutina.

```
;-----------------------------------------------------------
; Pletter v0.5b VRAM Depacker v1.1 - 16 Kb version
; HL = RAM/ROM source
; DE = VRAM destination
;-----------------------------------------------------------
DEPLET:
        di

; VRAM address setup
        ld      a,e
        out     (099h),a
```

```
        ld      a,d
        or      040h
        out     (099h),a
; Initialization
        ld      a,[hl]
        inc     hl
        exx
        ld      de,0
        add     a,a
        inc     a
        rl      e
        add     a,a
        rl      e
        add     a,a
        rl      e
        rl      e
        ld      hl,modes
        add     hl,de
        ld      e,[hl]
        ld      ixl,e
        inc     hl
        ld      e,[hl]
        ld      ixh,e
        ld      e,1
        exx
        ld      iy,loop
; Main depack loop
literal:
        ld      c,098h
        outi
        inc     de
loop:
        add     a,a
        call    z,getbit
        jr      nc,literal
; Compressed data
        exx
        ld      h,d
        ld      l,e
getlen:
        add     a,a
        call    z,getbitexx
        jr      nc,lenok
lus:
        add     a,a
        call    z,getbitexx
        adc     hl,hl
        ret     c
        add     a,a
        call    z,getbitexx
        jr      nc,lenok
        add     a,a
        call    z,getbitexx
        adc     hl,hl
        jp      c,Depack_out
        add     a,a
        call    z,getbitexx
        jp      c,lus
lenok:
        inc     hl
        exx
        ld      c,[hl]
        inc     hl
        ld      b,0
        bit     7,c
        jp      z,offsok
        jp      ix

mode7:
        add     a,a
        call    z,getbit
        rl      b
mode6:
        add     a,a
        call    z,getbit
        rl      b
mode5:
        add     a,a
        call    z,getbit
        rl      b
```

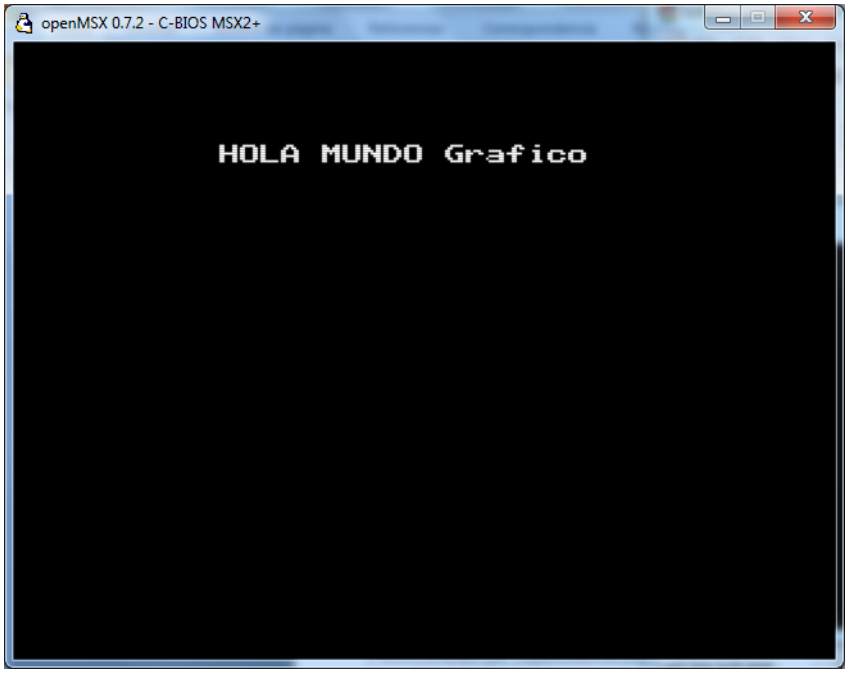
```
mode4:
        add     a,a
        call    z,getbit
        rl      b
mode3:
        add     a,a
        call    z,getbit
        rl      b
mode2:
        add     a,a
        call    z,getbit
        rl      b
        add     a,a
        call    z,getbit
        jr      nc,offsok
        or      a
        inc     b
        res     7,c
offsok:
        inc     bc
        push    hl
        exx
        push    hl
        exx
        ld      l,e
        ld      h,d
        sbc     hl,bc
        pop     bc
        push    af
@@loop:
        ld      a,l
        out     (099h),a
        ld      a,h
        nop                                     ; VDP timing
        out     (099h),a
        nop                                     ; VDP timing
        in      a,(098h)
        ex      af,af'
        ld      a,e
        nop                     ; VDP timing
        out     (099h),a
        ld      a,d
        or      040h
        out     (099h),a
        ex      af,af'
        nop                     ; VDP timing
        out     (098h),a
        inc     de
        cpi
        jp      pe,@@loop
        pop     af
        pop     hl
        jp      iy
getbit:
        ld      a,[hl]
        inc     hl
        rla
        ret
getbitexx:
        exx
        ld      a,[hl]
        inc     hl
        exx
        rla
        ret
; Depacker exit
Depack_out:
        ei
        ret
modes:
        dw      offsok
        dw      mode2
        dw      mode3
        dw      mode4
        dw      mode5
        dw      mode6
        dw      mode7
```

Aquí en esta parte del código es donde tienes que pegar el código en formato DB´s del set de caracteres que previamente has comprimido y convertido a formato DB´s con el binDB.exe y que te he explicado anteriormente.

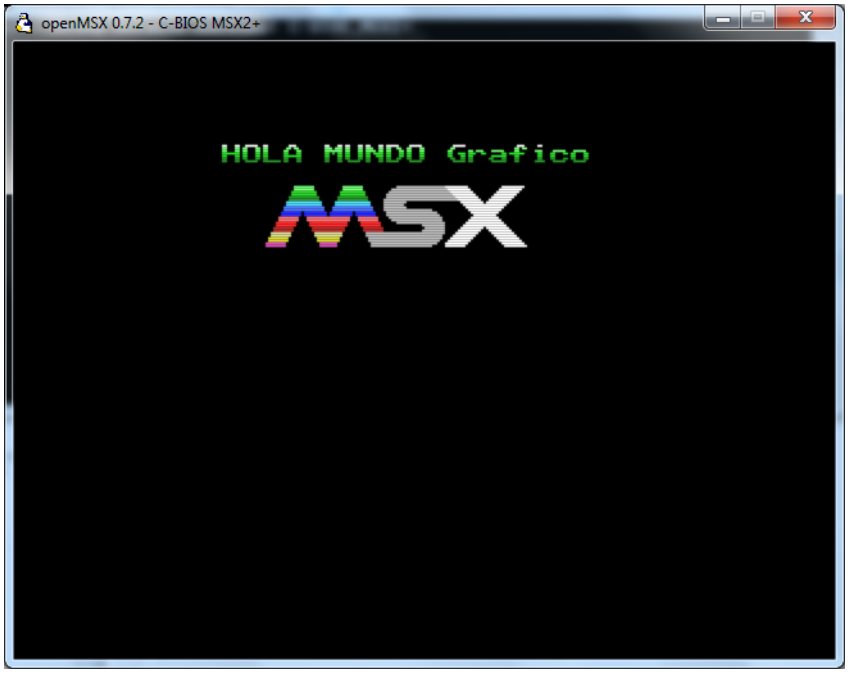
```
;-------------------------------------------------------
; SET2.PCX.CHR.PLET - binary dump
; generated by binDB v.0.10
;-------------------------------------------------------
SET2_PLET:
 db 03Eh,0FFh,011h,000h,0F3h,0E1h,000h,0F3h
 db 0FFh,00Eh,001h,0C9h,0C9h,0DBh,00Fh,096h
 db 007h,080h,009h,040h,003h,011h,0E0h,0D3h
 db 0E1h,0F2h,0C1h,040h,0F3h,010h,09Ch,099h
 db 0F3h,0E7h,0CCh,000h,09Ch,0FFh,0C7h,09Bh
 db 09Bh,0C7h,0B2h,0B9h,046h,0C4h,017h,0F3h
 db 0F7h,082h,027h,0F1h,0E7h,0CFh,000h,022h
 db 0E7h,0F1h,017h,0F3h,0F9h,000h,031h,0F3h
 db 0C7h,048h,0D2h,032h,048h,0D2h,02Fh,020h
 db 02Dh,0C0h,0F3h,006h,08Eh,023h,0F1h,0F9h
 db 02Ch,03Eh,0C1h,0FFh,06Ch,0E5h,018h,0FCh
 db 02Dh,002h,03Ah,09Fh,0FFh,0E3h,0D9h,09Ch
 db 000h,025h,0CCh,0E1h,012h,0E3h,014h,090h
 db 032h,024h,09Ch,0F8h,0E1h,004h,0C3h,09Fh
 db 080h,0FFh,0C0h,01Eh,0E1h,010h,0FCh,09Ch
 db 0C1h,03Bh,0E1h,0C9h,099h,040h,080h,059h
 db 0FFh,081h,09Fh,081h,0FCh,0E2h,00Fh,0E1h
 db 0CFh,008h,0A3h,030h,007h,080h,09Ch,02Ah
 db 03Fh,000h,02Fh,00Ch,0EEh,002h,007h,04Ah
 db 0C0h,052h,0C3h,057h,013h,0EBh,002h,000h
 db 023h,033h,002h,0FDh,0FBh,0A1h,066h,09Ah
 db 0FCh,095h,010h,05Bh,001h,075h,01Eh,00Ch
 db 012h,0C5h,037h,0FCh,0F1h,029h,000h,001h
 db 0C3h,0BDh,066h,05Eh,05Eh,066h,0BDh,015h
 db 0C3h,0E3h,0C9h,040h,053h,019h,0DFh,05Ch
 db 0D1h,002h,067h,0CCh,09Fh,060h,000h,097h
 db 083h,050h,099h,00Dh,0FAh,083h,0A4h,06Fh
 db 00Dh,083h,002h,080h,0F2h,007h,0B7h,010h
 db 0E0h,087h,098h,084h,0B7h,0E0h,096h,0E9h
 db 036h,046h,037h,0ECh,0C3h,0BFh,0FCh,0B0h
 db 000h,08Fh,082h,033h,093h,087h,093h,099h
 db 017h,075h,09Fh,000h,037h,00Eh,00Fh,088h
 db 080h,094h,027h,00Ah,09Ch,08Ch,084h,090h
 db 038h,007h,0B9h,07Fh,000h,0C1h,0D6h,06Fh
 db 06Dh,070h,04Fh,00Fh,092h,099h,0C4h,0E0h
 db 00Fh,098h,083h,091h,098h,081h,097h,099h
 db 09Fh,0C1h,0DBh,047h,02Fh,057h,0F3h,067h
 db 038h,02Fh,005h,0C9h,0E3h,072h,0F7h,007h
 db 094h,09Eh,098h,0DDh,057h,0C1h,012h,0E3h
 db 0C1h,088h,04Fh,0CCh,000h,070h,0E1h,027h
 db 080h,0F8h,0F1h,015h,0E3h,0C7h,08Fh,06Fh
 db 053h,0C8h,000h,081h,0F0h,002h,0DFh,0EFh
 db 0F7h,0FBh,0FDh,0FFh,00Fh,075h,0F9h,000h
 db 00Fh,017h,000h,0F7h,0EBh,035h,0D2h,000h
 db 027h,09Fh,0BCh,0BCh,00Bh,093h,06Eh,0C0h
 db 0C2h,081h,0AFh,093h,08Ch,038h,000h,091h
 db 00Fh,09Ch,09Fh,0C8h,06Fh,0CCh,021h,0C4h
 db 098h,091h,0C4h,0A5h,00Fh,053h,090h,08Fh
 db 0E4h,0E7h,081h,0E6h,0F7h,000h,0A3h,017h
 db 016h,0FCh,0C1h,038h,063h,092h,0CFh,0B9h
 db 0A7h,0C0h,0E1h,085h,007h,093h,0C7h,017h
 db 0A3h,080h,0A7h,00Dh,046h,017h,000h,089h
 db 094h,061h,000h,007h,091h,08Ch,0DDh,02Fh
 db 04Fh,0C7h,0CFh,057h,00Fh,06Eh,02Dh,08Eh
 db 04Fh,0FCh,00Fh,0E7h,0DFh,032h,01Fh,0F1h
 db 0ADh,01Fh,0DCh,087h,046h,002h,0F0h,031h
 db 00Fh,099h,000h,0C5h,099h,007h,036h,0A6h
 db 00Fh,01Dh,04Dh,0C1h,0EBh,00Fh,03Ah,00Dh
 db 0C9h,04Fh,080h,046h,0C8h,0E4h,0FCh,033h
 db 068h,081h,0D7h,08Ah,0E7h,004h,005h,0A3h
 db 002h,08Dh,08Eh,0ABh,003h,0ECh,00Ch,012h
 db 0A2h,027h,009h,094h,0F9h,005h,0D9h,087h
 db 05Fh,080h,005h,0FFh,0FFh,0FFh,0F8h
;-----------------------------------------------
```

O bien puedes quitar todos estos DB´s e incluir el fichero BINARIO directamente en el código de esta manera
El fichero SET2.PCX.PLET Tiene que estar en el directorio donde tengas el código en ensamblador.

```
;-----------------------------------------------
; SET2.PCX.CHR.PLET
; Set de caracteres en BINario comprimido con PLETTER
;SET2_PET:
;       .INCBIN        "SET2.PCX.CHR.PLET"
;-----------------------------------------------
; FINAL DE NUESTRO CODIGO EN ENSAMBLADOR.
;-----------------------------------------------
```



Este es el resultado Final que veras si todo lo has realizado de manera correcta.



Como queda un poco soso, sin colorido y demás en la segunda parte del mundo de los gráficos vamos a ver todo este proceso.

La imagen de la izquierda veremos cómo hacerla en la 2ª entrega.

Os animáis a crearlo vosotros mismos, con todo lo que habéis aprendido en este tutorial, debería bastaros para que vosotros mismos podáis realizarlo.

Espero ver vuestros comentarios y vuestras capturas de pantallas, en los BLOG´s o FORO´s del tutorial.

Espero que haya sido de vuestro total agrado y nos vemos en el próximo tutorial. Donde explicaré como dar un colorido especial a nuestras letras desde código, como incorporar gráficos a nuestro Hola Mundo Grafico y todo lo relacionado con la creación de pantallas o lo que es lo mismo trabajar con la NAMTBL creando mapeados usando el nMSXtiles.

José Vila Cuadrillero

"ES DETESTABLE ESA AVARICIA ESPIRITUAL QUE TIENEN, LOS QUE SABIENDO ALGO, NO PROCURAN LA TRANSMISION DE ESOS CONOCIMIENTOS."
Miguel de Unamuno
Escritor y Filósofo.
(Bilbao 1864 - Salamanca 1936)