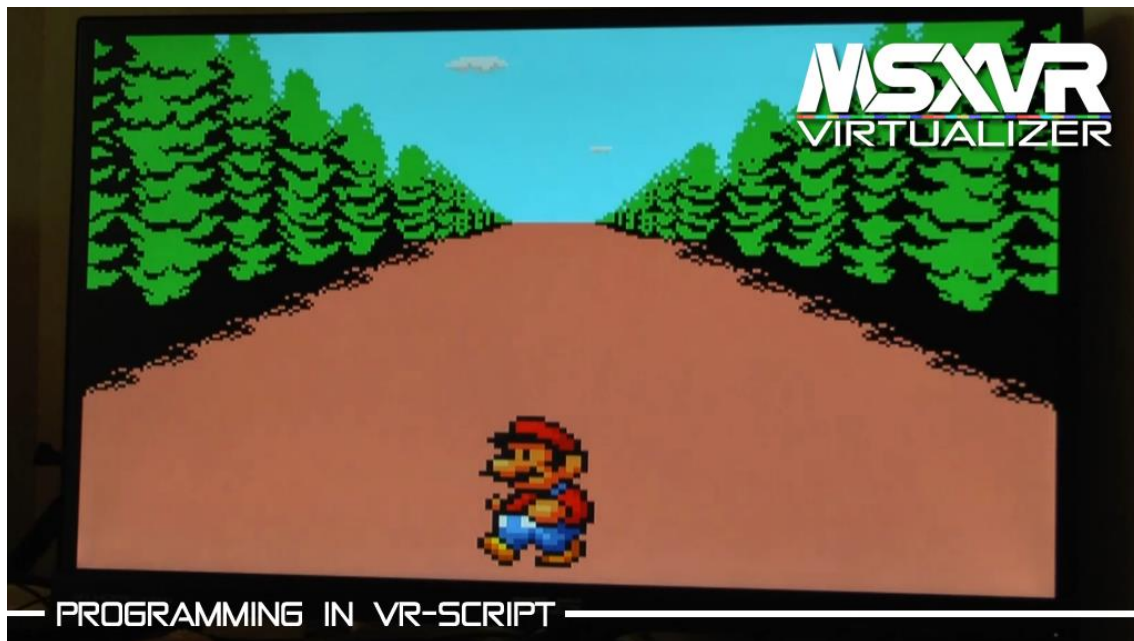


# TUTORIAL #1

## PENGUIN-MARIO (VR-SCRIPT)



DIFICULTAD:



REVISIÓN: 1

## ¿Qué aprenderemos?

- Crear un script
- Crear un "hello world"
- Primeros pasos en el uso de la librería GL
- Creación de sprites
- Control de sprites
- Animación de sprites

# ¡Comencemos!

Desde VR-DOS realizamos las siguientes acciones:

- Creamos una carpeta para nuestro proyecto

```
C:/>mkdir tutorial1
```

- Entramos dentro:

```
C:/>cd tutorial1
C:/tutorial1/>
```

- Editamos nuestro primer programa:

```
C:/tutorial1/>edit main.pi
```

Se abrirá el editor de textos y escribiremos el siguiente código:

```
1  class Main implements GLProgram
2  {
3      virtual Start()
4      {
5          GetConsole().PrintLn("Hola mundo!");
6      }
7  }
```

Lo guardamos: CTRL + S o bien a través del menú File -> Save

Salimos al VR-DOS: CTRL + Q

Y lo lanzamos:

```
C:/tutorial1/>main
Hola mundo!
```

Has hecho tu primer programa con VR-Script, ¡enhorabuena!

Para interrumpir la ejecución de nuestro programa y volver al VR-DOS, podemos pulsar la combinación de teclas: CTRL + C

Ahora vamos a hacer algo un poco más complicado. Vamos a mostrar un Sprite en pantalla. Para ello usaremos uno de los gráficos que tenemos como ejemplo: "supermario.png"

Puedes bajarte el código y recursos de este tutorial haciendo:

```
wget http://msxvr.es/resources/tutorial1.zip
```

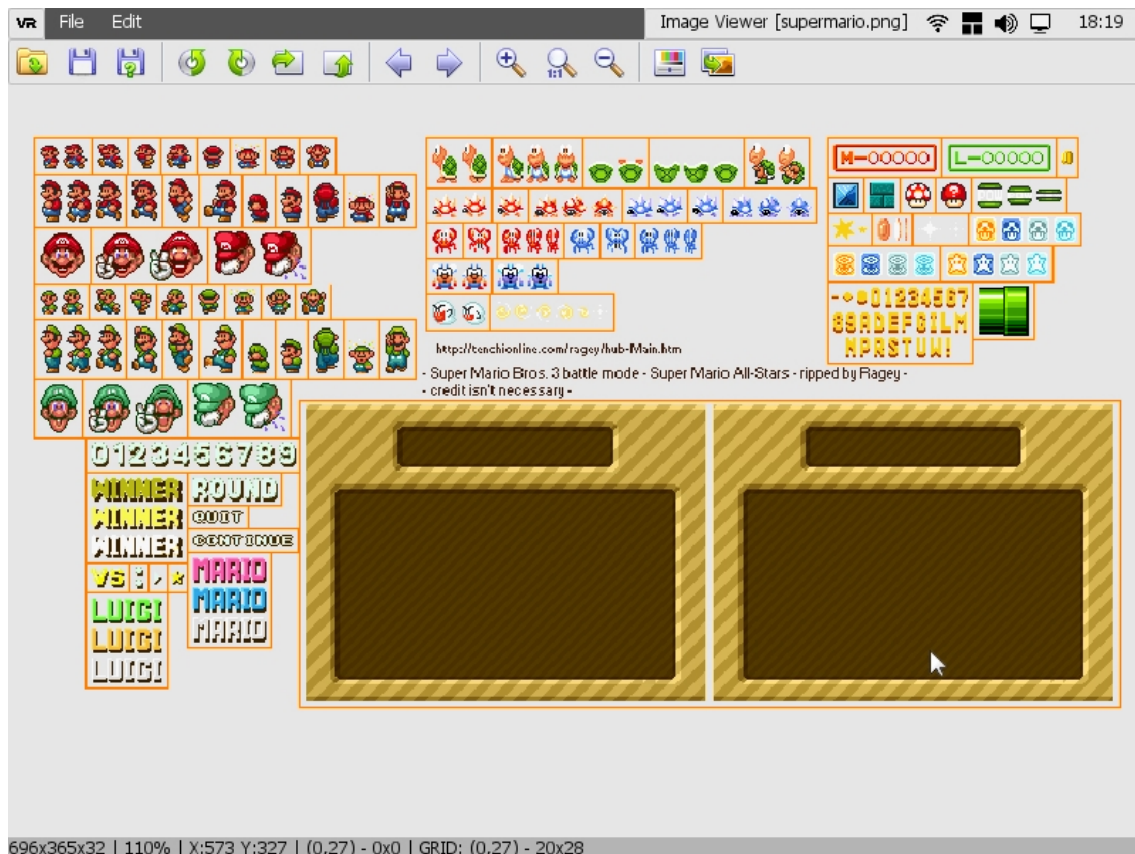
Y descomprimiendo el ZIP:

```
ziptool /E tutorial1.zip *.* tutorial1/
```

Si miramos el contenido de este archivo gráfico:

```
C:/tutorial1/>play supermario.png
```

Veremos lo siguiente:



Vamos a crear la animación de andar del Mario, en este caso, los gráficos están distribuidos de cualquier forma y con distintos tamaños. Podemos crear animaciones de muchas formas, pero ahora vamos a hacerlo del siguiente modo:

- Vamos a ajustar la rejilla para que englobe los frames que queremos en nuestra animación de andar. Abriremos el panel con Edit -> GRID y pondremos un valor de ancho y alto para la rejilla, pongamos Width=32 y Height=32. También podemos usar CONTROL + SHIFT + CURSORES para ajustar este tamaño.
- Activaremos la rejilla con CTRL + G



- Ajustamos el origen para que los rectángulos ajusten en cada frame de la animación. Para ello moveremos la rejilla con CTRL + CURSORES



- Ajustaremos el ancho y alto de la rejilla con CTRL + SHIFT + CURSORES hasta que tengamos la animación localizada entre el frame 0 y 2.
- Nos anotamos los valores que aparecen abajo

696x365x32 | 389% | X:88 Y:-27 | (0,27) - 0x0 | GRID: (2,27) - 18x28

Ahora vamos a crear el siguiente programa:

```

1  class Main implements GL_Program
2  {
3      virtual Start()
4      {
5          _spr = NewSprite("Mario");
6          _ani = _spr.AddAnimationWithSheet("walk", "supermario.png", 0, 2, 2, 27, 18, 28);
7          _spr.SetAnimation(_ani);
8      }
9  }
```

Los parámetros del AddAnimationWithSheet son:

- El nombre que le ponemos a la animación
- El nombre del archivo donde vamos a sacar los gráficos
- El frame inicial que será el 0
- El frame final que será el 2



- El origen de la rejilla
- El tamaño de la rejilla, que los obtenemos de aquí:

| GRID: (2,27) - 18x28

Si ejecutamos nuestro programa veremos a Mario en la pantalla de arriba de la pantalla andando. Como se ve pequeño, vamos a aplicar escala al sprite (SetScale) y ya de paso lo pondremos en otro lugar de la pantalla (SetPos):

```
1  class Main implements GL_Program
2  {
3      virtual Start()
4      {
5          _spr = NewSprite("Mario");
6          _ani = _spr.AddAnimationWithSheet("walk", "supermario.png", 0, 2, 2, 27, 18, 28);
7          _spr.SetAnimation(_ani);
8          _spr.SetScale(5);
9          _spr.SetPos(200, 200);
10     }
11 }
```

## Pongamos un fondo

Ahora mismo, nuestro sprite aparece sobre el VR-DOS y claro, nos gustaría ubicarlo en un escenario un tanto más apropiado ¿cierto?

Vamos a poner un color sólido como fondo, usaremos la función SetBgColor a la cual se le pasa un color RGB (rojo, verde, azul) o ARGB(opacidad, rojo, verde azul)

```
1  class Main implements GL_Program
2  {
3      virtual Start()
4      {
5          SetBgColor(RGB(0,0,0));
6
7          _spr = NewSprite("Mario");
8          _ani = _spr.AddAnimationWithSheet("walk", "supermario.png", 0, 2, 2, 27, 18, 28);
9          _spr.SetAnimation(_ani);
10         _spr.SetScale(5);
11         _spr.SetPos(200, 200);
12     }
13 }
```

En el ejemplo anterior mostramos un fondo totalmente negro. Si quisiéramos no tener fondo, podemos usar el color 0 o cualquier color con opacidad cero.

Ahora vamos a poner como fondo una imagen. Usaremos "background1.jpg".

Cambiaremos la sentencia SetBgColor por estas dos:

```
5         SetBgColor( RGB(255, 255, 255));
6         SetBgImage("background1.jpg");
```

Hay que indicar que el color es el blanco, porque el color también se usa para dar tonalidad a la imagen que queramos usar. Si usamos un RGB(255,255,255) o lo que es lo mismo, color blanco, la imagen saldrá con sus colores originales.

## ¡Muévete Mario!

Hay muchas formas de aplicar movimientos a nuestros sprites, pero vamos a aprender una de las más sencillas. Usaremos un controlador. Los controladores permiten asociar comportamientos a nuestros objetos, en este caso, a nuestro sprite de Mario.

```
1  class Main implements GL_Program
2  {
3      virtual Start()
4      {
5          SetBgColor( RGB(255, 255, 255));
6          SetBgImage("background1.jpg");
7
8          _spr = NewSprite("Mario");
9          _ani = _spr.AddAnimationWithSheet("walk", "supermario.png", 0, 2, 2, 27, 18, 28);
10         _spr.SetAnimation(_ani);
11         _spr.SetScale(6);
12         _spr.SetPos((GetResX()-_spr.GetHF())/2, (GetResY()-_spr.GetHF()-10));
13
14         _pad = NewPad("pad");
15         _pad.SetPreset("CURSORS");
16         _pad.Start();
17         _controller = _spr.CreateController();
18         _controller.SetPad(_pad);
19         _controller.SetSpeed(4);
20
21         _controller.SetUpdateCallback(this, "OnUpdate");
22     }
```

Como se puede ver, al final de nuestro programa, hemos creado un controlador en el sprite de Mario y a dicho controlador le hemos asociado un PAD al que le hemos asignado el modo CURSORES. El SetSpeed permite indicar a que velocidad se moverá el personaje por pantalla.

Un controlador permite diferentes modos, pudiendo asociar joysticks u otros dispositivos si lo hacemos a través de un objeto PAD. También permite asociar otro tipo de movimientos como una ruta o a otro sprite u objeto de una escena.

Ahora vamos a mejorar un poco el comportamiento de nuestro Mario. Vamos a hacer que si nos movemos a la IZQUIERDA o DERECHA, el personaje se oriente en esa dirección.

Para ello, vamos a engancharnos a nuestro controlador de manera que podamos tener cierto control sobre lo que se va haciendo fotograma a fotograma por el mismo. Nuestro controlador se encarga de detectar las pulsaciones de los cursores para aplicar movimiento al sprite, una vez hace eso, si nosotros se lo pedimos, llamará a una función definida por nosotros. Aquí, en este "callback", es donde miraremos si estamos avanzando a la izquierda o a la derecha o arriba o abajo, lo que viene a ser la dirección de movimiento.




Echa un vistazo al siguiente programa:

```

1  class Main implements GL_Program
2  {
3      virtual Start()
4      {
5          SetBgColor(RGB(255,255,255));
6          SetBgImage("background1.jpg");
7
8          _spr = NewSprite("Mario");
9          _ani = _spr.AddAnimationWithSheet("walk", "supermario.png", 0, 2, 2, 27, 18, 28);
10         _spr.SetAnimation(_ani);
11         _spr.SetScale(6);
12         _spr.SetPos((GetResX()-_spr.GetWF())/2, (GetResY()-_spr.GetHF()-10));
13
14         _pad = NewPad("pad");
15         _pad.SetPreset("CURSORS");
16         _pad.Start();
17         _controller = _spr.CreateController();
18         _controller.SetPad(_pad);
19         _controller.SetSpeed(4);
20
21         _controller.SetUpdateCallback(this, "OnUpdate");
22     }
23
24     function OnUpdate(_controller)
25     {
26         _spr = _controller.target;
27         if (_controller.dirX < 0) _spr.SetHFlip(false);
28         else if (_controller.dirX > 0) _spr.SetHFlip(true);
29
30         if (_controller.dirX != _controller.dirY)
31             _spr.SetAnimationSpeedFactor(1.0f);
32         else
33         {
34             _spr.SetAnimationSpeedFactor(0.0f);
35             _spr.ResetAnimation();
36         }
37     }

```

La función “OnUpdate” se invoca constantemente y en ella podemos saber la dirección que tiene nuestro controlador y con la que mueve a nuestro sprite de Mario. Las propiedades “dirX” y “dirY” nos dan esta información. Si el valor de “dirX” es positivo o negativo, nos permite saber si va a la izquierda o derecha. Sabiendo esto, vamos a aplicar un efecto de Volteo Horizontal (Horizontal Flip), de manera que si nuestro sprite, por defecto, mira a la izquierda, volteándolo, mirará a la derecha.

NORMAL	HORIZONTAL FLIP	VERTICAL FLIP
		

Viendo cómo se mueve nuestro Mario y la capacidad de tener de gestionar el movimiento y comportamiento de nuestro personaje, ¿no os apetece hacer que se haga grande o pequeño a medida que subimos y bajamos por la pantalla? Sería una forma muy interesante de aprender algo nuevo a la par que hacemos que nuestro personaje se integre mejor en ese fondo. ¡Vamos allá!



Necesitaremos saber si subimos o bajamos, usaremos “dirY” al igual que hemos hecho con “dirX”. Para hacer a nuestro personaje más grande o más pequeño usaremos la función SetScale, función que ya hemos usado inicialmente. Haremos que si subimos la escala va reduciéndose gradualmente y si bajamos la incrementaremos.

```

20
21     function OnUpdate(_controller)
22     {
23         _spr = _controller.target;
24         if (_controller.dirX < 0) _spr.SetHFlip(false);
25         else if (_controller.dirX > 0) _spr.SetHFlip(true);
26
27         _scale = _spr.GetScale();
28         if (_controller.dirY < 0) _scale -= 0.001f;
29         else if (_controller.dirY > 0) _scale += 0.001f;
30         _scale = clamp(_scale, 0.1, 6);
31         _spr.SetScale(_scale);
32     }
33 }

```

Esto se va complicando por momentos, ¿eh? Claro, a poco que queramos ir puliendo y aplicando más detalles a nuestros programas, más código hará falta e inevitablemente irá creciendo. ¡Pero no pasa nada! Esto hay que asumirlo y todo esto no es un problema cuando el código se entiende y en eso estamos. Vamos a ir viendo lo nuevo de este programa:

- Por un lado hemos centrado a nuestro personaje en la parte inferior de la pantalla

```

12         _spr.SetPos((GetResX()-_spr.GetWf())/2, (GetResY()-_spr.GetHF()-10));

```

- Y luego hemos agregado algo más de código en el “OnUpdate” asociado al controllador. En este caso, la gestión de la escala para simular que nuestro personaje se aleja o se acerca.

```

27         _scale = _spr.GetScale();
28         if (_controller.dirY < 0) _scale -= 0.001f;
29         else if (_controller.dirY > 0) _scale += 0.001f;
30         _scale = clamp(_scale, 0.1, 6);
31         _spr.SetScale(_scale);

```

En este código, obtenemos la escala del personaje que inicialmente hemos considerado como 6, la más alta y que correspondería con la posición inferior de la pantalla. A medida que subimos, decrementamos la escala con un paso constante de 0.001f para que nuestro personaje simule que se aleja. Podéis jugar con todos estos valores para ver que ocurre.

Vamos a rematar el trabajo un poquito, vamos a aplicar una serie de consideraciones que hemos tenido en cuenta en el código que verás más abajo, son estas:

- Vamos a impedir que el personaje se pueda salir de la pantalla.
- Vamos a hacer que su recorrido en X se ajuste a la perspectiva del camino.
- Vamos a hacer que su velocidad en X y en Y se ajuste a la perspectiva también.
- Vamos a hacer que su animación pare si no nos movemos.

```

21     function OnUpdate(_controller)
22     {
23         _spr = _controller.target;
24         if (_controller.dirX < 0) _spr.SetHFlip(false);
25         else if (_controller.dirX > 0) _spr.SetHFlip(true);
26
27         if (_controller.dirX || _controller.dirY)
28             _spr.SetAnimationSpeedFactor(1.0f);
29         else
30         {
31             _spr.SetAnimationSpeedFactor(0.0f);
32             _spr.ResetAnimation();
33         }
34
35         _my = 180;
36
37         _scale = _spr.GetScale();
38         _scale = 2 + 4*(_spr.GetY() - _my) / (GetResY() - _spr.GetHF() - _my);
39         _scale = clamp(_scale, 2, 6);
40         _spr.SetScale(_scale);
41
42         _controller.SetSpeed(_scale, _scale/2);
43
44         _mx = 80*(6-_scale);
45         _mw = GetResX() - _spr.GetWF() - _mx*2;
46         _mh = GetResY() - _spr.GetHF() - _my;
47         _controller.EnableLimits(_mx, _my, _mw, _mh);
48     }
49 }

```

La escala del personaje se ajusta entre 2 y 6. Asumimos que el personaje no podrá subir más allá de una posición Y de 180 y que en esa posición su escala será la mínima, o sea 2.



Y=180

Para que un controlador tenga limitaciones de movimiento usaremos su función “EnableLimits”. A esta función se le pasa un rectángulo. En nuestro caso, modificaremos los límites en función de nuestra posición vertical, o sea, en función de la escala que vayamos aplicando al personaje.

`_spr.GetWF()` nos devolverá el tamaño en pantalla del personaje, o sea, tu tamaño original con la escala aplicada. Lo mismo ocurre con `_spr.GetHF()` pero para su altura.

En este trozo de código es donde se controla que el personaje pare y se vuelva a mostrar el frame 0 de la animación de andar. Si “dirX” o “dirY” tienen valor, es que el personaje se ha movido y no ha llegado a ningún límite.

```

27         if (_controller.dirX || _controller.dirY)
28             _spr.SetAnimationSpeedFactor(1.0f);
29         else
30         {
31             _spr.SetAnimationSpeedFactor(0.0f);
32             _spr.ResetAnimation();
33         }

```

Para hacer que el personaje se mueva más rápido o más lento, intentando representar su posición dentro del 3D falso de la pantalla, usamos este código:

```

42         _controller.SetSpeed(_scale, _scale/2);

```

Si usamos un solo parámetro, aplicamos la misma velocidad tanto a X como Y. Si le pasamos dos parámetros, podemos aplicar diferentes velocidades, lo cual nos interesa para simular esa falsa perspectiva.