AYUDA BLITZ BASIC 3D

ÍNDICE

SCANCODES

TECLA DEL TECLADO	SCANCODE	COMENTARIOS
ESCAPE	1	
1	2	
2	3	
3	4	
4	5	
5	6	
6	7	
7	8	
8	9	
9	10	
0	11	
Menos (-)	12	On Main Keyboard
Igual (=)	13	
Backspace	14	Backspace key
Tab	15	
Q	16	
W	17	
E	18	
R	19	
T	20	
Y	21	
U	22	
I	23	
O	24	
P	25	
Corchete Izq. ([)	26	
Corchete Der. (])	27	
Intro	28	Return/Enter on Main Keyboard
Control Izq	29	
A	30	
S	31	
D	32	
F	33	
G	34	
H	35	
J	36	
K	37	
L	38	
Punto y coma (;)	39	
Apóstrofe (')	40	
Grave	41	Accent Grave
Shift Izq	42	
Barra invertida (\)	43	
Z	44	
X	45	
C	46	
V	47	
B	48	
N	49	
M	50	
Coma (,)	51	
Punto (.)	52	On Main keyboard
Barra (/)	53	On Main Keyboard
Shift Der	54	
Asterisco (*)	55	On Numeric Keypad
Alt Izq/Menú	56	

TECLA DEL TECLADO	SCANCODE	COMENTARIOS
Espacio	57	
Capital	58	
F1	59	
F2	60	
F3	61	
F4	62	
F5	63	
F6	64	
F7	65	
F8	66	
F9	67	
F10	68	
NumLock	69	
Scroll Lock	70	
NumPad 7	71	
NumPad 8	72	
NumPad 9	73	
Menos (-)	74	On Numeric Keypad
NumPad 4	75	
NumPad 5	76	
NumPad 6	77	
Más (+)	78	On Numeric Keypad
NumPad 1	79	
NumPad 2	80	
NumPad 3	81	
NumPad 0	82	
Punto (.)	83	On Numeric Keypad
OEM_102	86	On UK/Germany Keyboards
F11	87	
F12	88	
F13	100	(NEC PC98)
F14	101	(NEC PC98)
F15	102	(NEC PC98)
Kana	112	Japanese Keyboard
ABNT_C1	115	/? on Portugese (Brazil) keyboards
Convert	121	Japanese Keyboard
NoConvert	123	Japanese Keyboard
Yen	125	Japanese Keyboard
ABNT_C2	126	Numpad on Portugese (Brazil) keyboards
Igual	141	= on numeric key pad (NEC PC98)
PrevTrack	144	Previous Track
AT	145	(NEC PC98)
Dos puntos (:)	146	(NEC PC98)
Underline	147	(NEC PC98)
Kanji	148	Japanese Keyboard

TECLA DEL TECLADO	SCANCODE	COMENTARIOS
Stop	149	(NEC PC98)
AX	150	Japan AX
Unlabeled	151	(J3100)
Next Track	153	Next Track
Intro	156	ENTER on Numeric Keypad
Control Der	157	
Mute	160	Mute
Calculator	161	Calculator
Play/Pause	162	Play/Pause
Media Stop	164	Media Stop
Volume Down	174	Volume -
Volume Up	176	Volume +
Web Home	178	Web Home
Comma (,)	179	On Numeric Keypad(NEX PC98)
Divide (/)	181	On Numeric Keypad
SysReq	183	
Right Alt/Menu	184	Right Alt
Pause	197	Pause
Home	199	Home on Arrow Pad
Up	200	Up Arrow on Arrow Keypad
Page Up/Prior	201	Page Up on Arrow Keypad
Left	203	Left Arrow on Arrow Keypad
Right	205	Right Arrow on Arrow Keypad
End	207	End Key on Arrow Keypad
Down	208	Down Key on Arrow Keypad
Next	209	Next Key on Arrow Keypad
Insert	210	Insert Key on Arrow Keypad
Delete	211	Delete Key on Arrow Keypad
Left Windows	219	Left Windows Key
Right Windows	220	Right Windows Key
Apps	221	Apps Menu Key
Power	222	System Power
Sleep	223	System Sleep
Wake	227	System Wake
Web Search	229	
Web Favorites	230	
Web Refresh	231	
Web Stop	232	
Web Forward	233	
Web Back	234	
My Computer	235	
Mail	236	
Media Select	237	

2D - Categorías

BASICO

If

Definición:
Comprueba una condición y ejecuta un código si la condición es verdadera.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Úsalo para comprobar el valor de una variable o para ver si una condición es verdadera o falsa. El código entre IF y END IF (o ENDIF) es ejecutado si la condición es verdadera. Usando NOT, puedes también actuar si la condición no es verdadera (o usar ELSE para usar un bloque de código distinto que si la condición es verdadera). Podrás usar múltiples IF's anidados usando ELSE IF (o ELSEIF) para hacer MUCHAS comparaciones. Si usas muchos IF's 'anidados', quizás deberías plantearte si te vensría bien usar una estructura SELECT.

Ejemplo:

```
; Ejemplo IF THEN
; Introduce el nombre del usuario
name$=Input$("¿Cómo te llamas? ")

If name$ = "Sergio" Then

Print "¡Te he reconocido, Sergio! ¡Bienvenido!"

Else

Print "¡No eres Sergio! ¡No importa, también eres bienvenido!"

End If
```

Then

Definición:
Parte de una estructura condicional IF.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Se usa en una estructura IF para indicar el fin de la comprobación. Mira el ejemplo para más información.

Ejemplo:
Ver Ejemplo IF THEN

Else

Definición:
Comienzo del código que será ejecutado si la condición del IF falla.

Descripción de los Parámetros*:*
Ninguno.

Descripción del Comando:
Hay veces que en un condicional IF ... THEN tu quieres que se ejecute un código si la condición NO es verdadera. El comando ELSE comienza este bloque de código, y termina con el comando END IF. Mira el ejemplo.

Ejemplo:

```
; Ejemplo ELSE

name$=Input$("¿Cómo te llamas?")
```

```
If name$="Sergio" then
Print "¡Hola Sergio!"
Else
Print "¡No tengo ni idea de quién eres!"
End If
```

ElseIf

Definición:
Añade una condición IF ... THEN en el interior de una estructura condicional.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
En una estructura condicional IF ... THEN, puedes querer comprobar otra condición si la original falla. Mira el ejemplo.

Ejemplo:

```
; Ejemplo ELSEIF

; Introduce el nombre del usuario
name$=Input$("¿Cómo te llamas? ")

; Se llama Sergio?
If name$ = "Sergio" Then

Print "¡Te he reconocido, Sergio! ¡Bienvenido!"

ElseIf name$="Xtreme" then
Print "¡Te he reconocido, Xtreme! ¡Bienvenido!"

Else
Print "¡No sé quién eres! ¡No importa, también eres bienvenido!"

; Fin de la comprobación de la condición
End If
```

Else If

Definición:
Añade una condición IF ... THEN en el interior de una estructura condicional.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
En una estructura condicional IF ... THEN, puedes querer comprobar otra condición si la original falla. Mira el ejemplo.

Ejemplo:

```
; Ejemplo ELSE IF

; Introduce el nombre del usuario
name$=Input$("¿Cómo te llamas? ")
; Se llama Sergio?
If name$ = "Sergio" Then

Print "¡Te he reconocido, Sergio! ¡Bienvenido!"

Else If name$="Xtreme" then
Print "¡Te he reconocido, Xtreme! ¡Bienvenido!"
Else
Print "¡No sé quién eres! ¡No importa, también eres bienvenido!"

; Fin de la comprobación de la condición
End If
```

EndIf

Definición:
Otra versión aceptable de END IF.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Concluye una estructura condicional IF ... THEN. Mira END IF para más información.

Ejemplo:

```
; Ejemplo IF THEN

; Introduce el nombre del usuario
name$=Input$("¿Cómo te llamas? ")

; ¿No es el nombre del usuario igual a SERGIO?
If name$ = "Sergio" Then

Print "¡Te he reconocido, Sergio! ¡Bienvenido!"

Else

Print "¡No eres Sergio! ¡No importa, también eres bienvenido!"

; Fin de la condición
EndIf
```

End If

Definición:
Cierra una condición IF/THEN.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
END IF señala el final de una condición IF THEN. Mira IF para más información. Puedes también usar ENDIF como un comando de una sola palabra (funciona igualmente).

Ejemplo:
Ver Ejemplo IF THEN

Select variable

Definición:
Ejecuta comandos dependiendo del valor de la variable especificada.

Descripción de los Parámetros:
variable - una variable válida

Descripción del Comando:
Este comando te permite crear una estructura que, dependiendo del valor de la variable suministrada, ejecutará un grupo de comandos en concreto en diferentes casos (CASE). Puedes también especificar un caso DEFAULT por si NINGUNO de los casos anteriores se dio. El final de esta estructura se marca con el comando END SELECT.
Elegir esta estructura elimina la necesidad de usar enormes condiciones IF/THEN anidadas. Mira el ejemplo para más información.

Ejemplo:

```
; Ejemplo SELECT/CASE/DEFAULT/END SELECT

; Signa un número aleatorio entre 1-10
mission=Rnd(1,10)
```

```
; Comienza el proceso de selección basado en el valor de 'mission'
Select mission

; es mission = 1?
Case 1
Print "Tu misión es coger el plutonio y ponerlo a salvo."

; es mission = 2?
Case 2
Print "Tu misión es destruir todos los enemigos."

; es mission = 3?
Case 3
Print "Tu misión es robarle al enemigo los plaos del edificio."

; ¿Qué hacer si ninguno de los casos se ha cumplido?
Default
Print "Las misiones -10 no están disponibles todavía."

; Fin del proceso de selección
End Select
```

Case valor

Definición:
Comienza un grupo de comandos de una estructura SELECT si el valor de la variable SELECT es igual que el valor especificado como parámetro.
Descripción de los Parámetros:
valor = algún valor válido de la variable SELECT

Descripción del Comando:
Cuando usas una estructura SELECT, el comando CASE define el punto de partida de la ejecución de comandos si el valor de la variable SELECT es igual que el valor especificado como argumento. Si el valor de SELECT no coincide con el valor de CASE, el comando seguirá ignorándolo hasta el siguiente CASE, DEFAULT, o END SELECT que sea encontrado. Mira SELECT y el ejemplo para un mejor entendimiento.

Ejemplo:
Ver Ejemplo SELECT

Default

Definición:
Especifica un juego de comandos que se ejecutarán en una estructura Select si ninguno de los CASE's ha sido reconocido.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
En una estructura SELECT, puedes ejecutar código si ninguno de los cases especificados son reconocidos. Todo el código entre DEFAULT y END SELECT será ejecutado si ningún CASE es reconocido. Mira SELECT, CASE, y el ejemplo para saber más.

Ejemplo:
Ver Ejemplo SELECT

End Select

Definición:
Cierra una estructura SELECT.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando termina la estructura SELECT. Mira SELECT, CASE, y DEFAULT para más información.

Ejemplo:
Ver Ejemplo SELECT

And

Definición:
Operador lógico comparativo para expresiones condicionales.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
AND es un operador lógico para hacer comprobaciones condicionales de múltiples valores y/o expresiones. Úsalo para asegurarte de que dos o más condiciones son verdaderas, normalmente en un condicional IF ... THEN. Mira el ejemplo y verás OR, NOT, y XOR.

Ejemplo:

```
; Ejemplo AND

name$=Input$("Introduce tu nombre:")
pw$=Input$("Password:")

if name$="Sergio" and pw$="www.blitzbasico.tk" then
print "¡Acceso concedido! ¡Bienvenido!"
else
print "Tu nombre o tu password no fue reconocido"
end if
```

Or

Definición:
Conjunción entre dos valores para derivar un valor booleano.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Una expresión lógica devuelve un valor booleano (TRUE o FALSE) de la comparación de expresiones o valores. Úsalo para comparar dos o más expresiones y actuar en consecuencia. Mira el ejemplo.

Ejemplo:

```
; Ejemplo OR

myNum1=Rnd(0,10)
myNum2=Rnd(0,10)

If myNum1 = 0 OR myNum2 = 0 then
print "Uno de mis números es un 0"
end if
```

Not

Definición:
Operador lógico para comprobar si una condición es falsa.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
El operador NOT se usa para determinar si una condición es FALSE en vez de TRUE. Se usa frecuentemente en bucles WHILE y estructuras IF para comprobar la NO-EXISTENCIA de un evento o valor. Usamos NOT frecuentemente en nuestros ejemplos, dentro de bucles WHILE para saber si la tecla ESC ha sido presionada.

Ejemplo:

```
; Ejemplo NOT

; Bucle hasta que se pulse ESC
```

```
While Not KeyHit(1) ; Mientras no se pulse ESC ...
Print "¡Pulsa ESC para salir!"
Wend
```

Repeat

Definición:
Primer comando de un bucle REPEAT ... UNTIL.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
El bucle REPEAT ... UNTIL te permite realizar una serie de comandos hasta que se cumpla una determinada condición. Esto permite que la condición aparezca después de que se ejecuten los comandos antes de la comprobación, no antes como en el bucle WHILE ... WEND. En general, usa REPEAT ... UNTIL si quieres que los comandos se ejecuten al menos una vez.

Ejemplo:

```
; Se repite hasta que el usuario pulse la tecla ESC

Repeat
print "¡Pulsa ESC para quitar esto!"
Until KeyHit(1)
```

Until condition

Definición:
El comando de cierre de un bucle REPEAT ... UNTIL.

Descripción de los Parámetros:
condition = alguna expresión válida (mira el ejemplo)

Descripción del Comando:
Esta porción del bucle REPEAT ... UNTIL dicta qué condición debe ser cumplida antes de terminar el bucle. Todos los comandos entre REPEAT y UNTIL serán ejecutados indefinidamente hasta que se cumpla la condición del UNTIL.

Ejemplo:
Ver Ejemplo REPEAT

Forever

Definición:
Se usa en un bucle REPEAT para crear un bucle infinito.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Remplaza el comando UNTIL en un bucle REPEAT ... UNTIL para crear un bucle infinito. Recuerda, la ejecución del programa continuará indefinidamente hasta que salgas del bucle por medio del código. Es normalmente conocido como un 'bucle infinito'. Usa EXIT (para marcharte del bucle) o END para acabar el programa.

Ejemplo:

```
; Ejemplo FOREVER

Repeat
If KeyHit(1) Then Exit
Print "¡Estás atrapado en un bucle infinito! ¡Pulsa ESC!"
Forever

Print "¡El bucle infinito ha acabado!"
```

While condition

Definición:
Comienza un bucle condicional WHILE/WEND.

Descripción de los Parámetros:
condition = una expresión válida

Descripción del Comando:
El bucle WHILE/WEND se usa cuando deseas ejecutar una serie de comandos varias veces deoendiendo de que una condición sea verdadera o no. Esto es similar al bucle REPEAT/UNTIL, exceptuando que la comprobación de la condició está al PRINCIPIO del bucle en vez de estar al final. Si necesitas ejecutar los comandos al menos una vez antes de comprobar la condición, usa REPEAT/UNTIL. Mira le ejemplo.

Ejemplo:

```
; Ejemplo While/Wend

; La condición del bucle está al PRINCIPIO del bucle
While Not KeyHit(1) ; Tanto tiempo como el usuario esté sin pulsar ESC...
Print "¡Pulsa ESC para acabar!" ; Print this
Wend ; Vuelve al inicio del bucle WHILE
```

Wend

Definición:
Comando de cierre de un bucle WHILE/WEND.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando es el que hace que la ejecución del programa se desvíe hacia el inicio del bucle WHILE/WEND donde esta el comando WHILE. Mira el comando WHILE para entender todo esto mejor.

Ejemplo:
Ver Ejemplo WHILE

For variable

Definición:
Primer comando de un bucle FOR ... NEXT.

Descripción de los Parámetros:
variable = una variable válida

Descripción del Comando:
El primer comando de un bucle FOR ... NEXT, este comando se usa para asignar a una variable una serie de números para ejecutar un bloque de código un número de veces. Usando el comando STEP podrás saltar un valor en concreto entre cada ciclo del bucle. Mira el ejemplo.
Nota: Al contrario que en muchos lenguajes BASIC, el comando NEXT NO usa la variable del comando FOR como un identificador.

Ejemplo:

```
; Imprime números del 10 al 1
For t = 1 To 10
Print t
Next

; Imprime los valores 1,3,5,7,9
For t = 1 To 10 Step 2
Print t
Next
```

To

Definición:
Dicta el rango de valores en un bucle FOR ... NEXT.

Descripción de los Parámetros:
Mira el ejemplo.

Descripción del Comando:
Usa el comando TO para decir en un bucle FOR ... NEXT entre que números se ejecutará el bucle. Si cuentas hacia abajo en vez de hacia arriba, debes usar un valor STEP negativo. Los valores deben ser números enteros. Mira el ejemplo.

Ejemplo:

```
; Imprime números del 10 al 1

For t = 10 to 1 Step -1
Print t
Next
```

Step

Definición:
Establece el incremento de un bucle FOR ... NEXT.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Úsalo para que un bucle FOR ... NEXT incremente un valor en concreto cada vez que se acaba un ciclo en vez de incrementar siempre 1. STEP 1 está asumido por lo que no necesita ser declarado.

Ejemplo:

```
; Print 1 through 100, by tens

For t = 1 To 100 Step 10
Print t
Next
```

Next

Definición:
El comando de cierre de un bucle FOR ... NEXT.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando cierra un bucle FOR ... NEXT, causando que la ejecución del programa vuelva al comando FOR si la condición no se ha cumplido. Comprueba el ejemplo para más información. Nora: NO uses la variable del comando FOR como un parámetro (p.e. NEXT t) como lo harías en la mayoría de los lenguajes BASIC. Blitz automáticamente usa la variable del FOR más cercano.

Ejemplo:

```
; Imprime números del 10 al 1
For t = 1 To 10
Print t
Next
; Imprime los valores 1,3,5,7,9
For t = 1 To 10 Step 2
Print t
Next
```

Exit

Definición:
Sale de un bucle inmediatamente.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando te permitirá marcharte de un bucle antes de llegar a su fin.

Ejemplo:

```
; Ejemplo EXIT

For t = 1 To 100
Print t
If t = 50 Then Exit
Next
```

Goto label

Definición:
Desvía la ejecución de un programa a la etiqueta especificada.

Descripción de los Parámetros:
label = etiqueta válida

Descripción del Comando:
Éste desvía el flujo del programa a la etiqueta especificada. Con el uso de funciones dentro de Blitz, no es muy práctico usar GOTO's, pero podrás encontrarlas útiles. Si tienes la necesidad de devolver la ejecución atrás, usa el comando GOSUB. Mira el ejemplo.

Ejemplo:

```
Print "El programa empieza..."
Goto label1
Print "Esta línea jamás será imprimida..."
End

.label1
Print "¡Hemos saltado aquí!"

; espera que pulses ESC para acabar
While Not KeyHit(1)
Wend
```

Gosub label

Definición:
Desvía la ejecución de un programa a la etiqueta especificada con la intención de volver a la llamada original.

Descripción de los Parámetros:
label = etiqueta válida

Descripción del Comando:
Éste desvía el flujo del programa a la etiqueta especificada con el entendimiento de que habrá un RETURN después que devolverá la ejecución donde el GOSUB fue llamado. Con el uso de funciones dentro de Blitz, no es muy práctico usar GOSUB's, pero podrás encontrarlas útiles. Si no tienes la necesidad de devolver la ejecución atrás, usa el comando GOTO. Mira el ejemplo.

Ejemplo:

```
Print "El programa comienza..."
Gosub label1
Print "El programa acaba..."
```

```
; espera que pulses ESC para acabar
While Not KeyHit(1)
Wend
End

.label1
Print "Podríamos hacer todo lo que quisieramos en esta parte del programa..."
Print "Pero volveremos donde fuimos llamados..."
Return
```

.label

Definición:
Establece una etiqueta a la que se puede llamar en el interior de tu programa.

Descripción de los Parámetros:
label = un nombre válido

Descripción del Comando:
Una etiqueta es una 'marca' a la que se puede llamar en el interior de tu programa. Básicamente quiere decir que es donde Blitz va a comenzar algo; en cada ejecución del programa (a través de Goto y Gosub) o donde comienza a leer Data (a través del comando Read). Mientras la mayoría de los programadores 'profesionales' critican y odian estos 'desvíos' con Goto/Gosub, puedes encontrar mucha utilidad en ellos. Usa Return para volver a la línea desde donde se llamo a la etiqueta mediante el comando Gosub (no puedes usar Return desde un comando Goto). Además, DEBES usar las etiquetas para leer valores Data usando el comando Read.

Ejemplo:

```
; The program starts here, but we'll branch to other labels

Gosub start
Gosub label3
Gosub label2

; Let's wait for ESC to be pressed before ending
While Not KeyHit(1)
Wend
End

.start
Print "We start here ..."
Return

.label2
Print "This is label 2"
Return

.label3
Print "This is label 3"
Return
```

Return value

Definición:
Sale inmediatamente de una función con un valor opcional o resume la ejecución de una subrutina llamada con Gosub.

Descripción de los Parámetros:
value = TRUE o FALSE

Descripción del Comando:
Cuando se llama dentro de una estructura FUNCTION, el comando RETURN inmediatamente vuelve a la línea desde que fue llamada esta función. Un valor opcional puede ser devuelto. Mira FUNCTION para más información. Recuerda, después de un Return, el código restante de la función no será ejecutado. Mira el ejemplo. También devuelve la ejecución de una subrutina llamada con el comando Gosub.

Ejemplo:

```
; RETURN example

; Set result to the return value of the function 'testme'
result=testme(Rnd(0,10));

; The program effectively ends here.

; The actual function
Function testme(test);

; If the random number passed = 0
If test=0 Then
Print "Value was 0"
Return False ; The Function ends immediately
Else
Print "The value was greater than 0"
Return True ; The Function ends immediately
End If
Print "This line never gets printed!"
End Function
```

Function name

Definición:
Comienza un fragmento de código independiente para llamar desde tu programa.

Descripción de los Parámetros:
name = un nombre válido que no sea una palabra clave de Blitz

Descripción del Comando:
Una función es una rutina de comandos que puedes elegir para llamar frecuentemente en el interior de tu programa.
Las funciones son independientes de tu 'código principal' y sólo se ejecutarán si son llamadas. Tienen su propio nombre, y las variables creadas fuera de la función NO están disponibles en el interior de la función (esto pasa también con las estructuras TYPE) a no ser que declares la variable o estructura Type como GLOBAL. Una vez se alcance el comando END FUNCTION, devolverá la ejecución del programa a la siguiente línea de la llamada de la función. Mira el ejemplo para más información.

Ejemplo:

```
; Function Example

; Get the user's name
name$=Input$("Enter Your Name:")

; Call a function to print how many letters the name has
numletters(name$);

; Let's get something BACK from the function
thefirst$=firstletter(name$)

; Now print results
Print "Was the first letter an 'S'? (1=True/0=False)" + thefirst$

;The program basically ends here, because functions don't run unless called.

; The actual function
Function numletters(passedname$)
Print "Your name has " + Len(passedname$) + " letters in it."
End Function

; Function to see if the first letter is S
Function firstletter(passedname$)

; If the first letter is an 'S' then return from the function a true value
If Left$(passedname$,1) = "S" Then
Return True
```

```
; Otherwise, return false
Else

Return False

End If
End Function
```

End Function

Definición:
Marca el final de una estructura Function.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Esta línea termina una estructura FUNCTION. Al llegar a esta línea, Blitz irá al comando siguiente de la llamada original de la función. Mira el comando FUNCTION para más información.

Ejemplo:

```
; End Function Example

; Get the user's name
name$=Input$("Enter Your Name:")

; Call a function to print how many letters the name has
numletters(name$);

;The program basically ends here, because functions don't run unless called.

; The actual function
Function numletters(passedname$)
Print "Your name has " + Len(passedname$) + " letters in it."
End Function
```

Const variablename

Definición:
Declara una variable como una constante y le asigna un valor.

Descripción de los Parámetros:
variablename = algún nombre válido de variable

Descripción del Comando:
Este comando declara una variable como una constante (una variable cuyo valor no podrá ser modificado) y le asigna un valor.

Ejemplo:

```
; Ejemplo CONST

Const scrWidth=640
Const scrHeight=480
Const alienscore=100

Graphics scrWidth,scrHeight

Print "Los puntos por disparar a un alien son siempre " + alienscore
```

Global variable

Definición:
Declara variables globales para usar en tu programa.

Descripción de los Parámetros:
variable = nombre válido de variable/TYPE

Descripción del Comando:
Hay dos tipos de variables en Blitz Basic; variables locales y variables globales. Las variables globales pueden ser usadas en cualquier parte de tu programa (p.e. el programa principal tiene funciones. Usa las variables globales cuando necesitas un valor en el programa entero). También puedes definir TYPEs como globales.

Ejemplo:

```
Global player1score ; Declare player 1's score as global
Global graph.Cursor ; Declare the Cursor TYPE as global
```

Local variable

Definición:
Declara una variable como local.

Descripción de los Parámetros:
variable = un nombre de variable válido

Descripción del Comando:
Este comando es probablemente una muestra de compatibilidad con otros llenguajes BASIC. LOCAL te permitirá especificar que la variable que estás definiendo está disponible SÓLO en el programa o Function donde fue definido. Si quieres que la variable sea accesible en todo tu programa, necesitas hacerla GLOBAL. Cuando creas una variable que no es Global, automáticamente se crea como una variable LOCAL. Si quieres puedes asignarle un valor a la variable en el momento de su declaración. Mira el ejemplo.

Ejemplo:

```
; Local example

; set lives to 5 for the main program loop
Local lives=5

; Call a function
while not keyhit(1)
showlives()
Wend

Function showlives()
; For this function, lives will be 10!
Local lives=10
Print lives
End Function
```

Dim variable(elements)

Definición:
Crea un array de variables para almacenar datos.

Descripción de los Parámetros:
variable = nombre de la variable
elements = número de elementos que crear. Puede ser multidimensional.

Descripción del Comando:
Prepara un array usando el tipo de variable especificado. Puedes usar tantas dimensiones como desees (el límite está en la memoria de tu ordenador). No hay forma de 'redimensionar' los arrays en Blitz, así que asegúrate de definirlos adecuadamente. Usa la notación correcta en la declaración de la variable para que sea una cadena de texto ($), número real (#) o número entero. Nota: Los arrays empiezan siempre con la referencia CERO (así variable(0) es el primer elemento del array). Para usar este comando de forma efectiva, debes entender los arrays. Piensa en un array como en una variable que tiene múltiples bloques con valores distintos todo dentro del mismo nombre de variable. Antes de la llegada de los TYPE's, éste era el mejor método de almacenar elementos repetidos porque puedes alterarlos a través de un array fácilmente. Por ejemplo, quieres poner 100 aliens en la pantalla. Tendrías que usar dos variables (alienx y alieny). Mediante estos arrays (alienx() y alieny(), con 100 elementos cada uno), podrías fácilmente establecer la localización del primer alien usando alienx(1), alieny(1). Las coordenadas del siguiente alien estarían en alienx(2), alienx(2) y así sucesivamente. Esto se hace fácilmente usando un bucle FOR ... NEXT o EACH pasando por todos los elementos. Los arrays también son muy útiles para notaciones 'multidimensionales'. En vez de almacenar las coordenadas de los 100 aliens con el método anterior, puedes usar una única variable: alien(100,1). El primer elemento indica que en la colección hay 100 aliens, el

segundo elemento es la localización X e Y de ese alien (el elemento 0 sería la coordenada X y el 1 la coordenada Y). Así, para establecer la posición del alien 57 a X=640, Y=480 se haría esto:

```
alien(57,0)=640
alien(57,1)=480
DrawImage imgAlien,alien(57,0),alien(57,1)
```

De acuerdo, los TYPE's son mucho mejor método para manejar estos múltiples valores con pocas rutinas.
Los arrays son muy buenos para alamacenar colecciones de objetos con un elemento común, pero a veces los TYPE's son más útiles.

Ejemplo:

```
; DIM example
; Create a collection of 100 random numbers

; Declare our array
Dim nums(100)

; Fill each element with a random number
For T = 1 to 100
nums(t) = Rnd(1,100)
Next
```

Type variable

Definición:
Define un objeto con una colección de variables.

Descripción de los Parámetros:
variable = un nombre de variable válido

Descripción del Comando:
Si conoces la programación en C, un TYPE es básicamente un STRUCT en Blitz Basic. Si no conoces C, ¡sigue leyendo!
TYPE es tu mejor amigo. Se usa para crear una 'colección' de objetos que comparten los mismos parámetros y necesitan ser utilizados rápida y fácilmente. Piensa en SPACE INVADERS. Hay muchos aliens en la pantalla a la vez. Cada uno de esos aliens tienen unas variables que todos necesitan: las coordenadas X e Y más una variable para controlar el gráfico que deben mostrar (piernas fuera o piernas dentro). Ahora, podríamos crear cientos de variables como invader1x, invader1y, invader2x, invader2y, etc. para controlar todos los aliens, ¿pero eso tendría sentido? Pordrías usar un array para almacenarlos; invader(número,x,y,gráfico), y el bucle FOR ... NEXT para manejarlos pero eso es mucho trabajo! La colección de variables TYPE fue creada para manejar precisamente este tipo de necesidades.
TYPE define una colección de objetos. Cada objeto en la colección tiene su propia copia de las variables definidas por el comando TYPE FIELD. Cada variable de cada objeto en la colección puede ser leída individualmente y fácilmente modificada. Usa el comando FIELD para asignar las variables que quieras entre los comandos TYPE y END TYPE. Si ayuda, piensa que una colección TYPE es una base de datos. Cada objeto es una entrada de la base de datos, y cada variable es un campo de la entrada. Usando comandos como BEFORE, AFTER, y FOR ... EACH, puedes mover el puntero de la 'base de datos' para apuntar a una entrada diferente y recuperar/modificar el valor del campo (variable).
¿No eres un gurú de las bases de datos? ¿Necesitas otro ejemplo? Vale. Diremos que estás preparando un auditorio para dar un dscurso o algún evento y está poniendo cientos de sillas para los espectadores. Las sillas tienen un lugar en concreto en el suelo, y algunas necesitarás reservarlas para gente importante (dignatarios, el alcalde, etc.). Compruebas el suelo, y te das cuenta de que no es más que una cuadrícula! Esto será muy fácil! Sólo necesitas numerar cada 'pieza' del suelo en un dibujo en papel y poner en la cuadrícula el 'nivel de importancia' de la persona que se sentará en ella. Así, tendrás para cada silla una fila y una columna en el dibujo que has hecho en papel (posicion X e Y) y un nivel al que ajustarla ('importancia' o 'altura'). Bien, ya estamos organizados. Ahora, aunque todo lo que tenemos está en papel, todavía tenemos que realizar el trabajo de colocar los asientos. Después de que esté todo hecho, diremos que tus jefes van hacia ti y te dicen 'No están centrados a la derecha... muévelos todos un asiento'. ¡No pasa nada! Tienes todo controlado, ya que es tan simple como mover todo una silla a la derecha (después de todo, si orden y 'altura' no cambiarán) - ¡todavía tienes que mover cada silla!
En Blitz, podrías tener un TYPE llamado SILLA, establecer los FIELD'S del TYPE como X, Y, y ALTURA. Puedes crear tantas sillas como necesites con el comando NEW (cada vez que usas NEW, crea una nueva silla, con sus PROPIAS variables X, Y y ALTURA) y las asigna la X, Y y ALTURA que creas conveniente. En el ejemplo siguiente, cuando el jefe te diga que debes mover las sillas una posición, probablemente te quejes. ¡Eso es mucho trabajo! En Blitz, podríamos usar cuatro líneas de código para ajustar todos nuestros objetos SILLA a su nueva posición (mediante comandos FOR ... EACH). ¿Todavía perdido? Está bien - los TYPE's son difíciles de entender. Mira el ejemplo e intentaremos mostrarte como trabajan los TYPE's en un entorno práctico. Te recomiendo mirar el código de otra gente también, para ayudarte a obtener un buen manejo de ellos. Una vez lo hagas, sabrás por qué la gente que programa en C está loca por los STRUCTs y por qué se usan tanto los TYPE's en Blitz.

Ejemplo:

```
; Define el Type CHAIR (SILLA)

Type CHAIR
Field X
Field Y
```

```
Field HEIGHT
End Type

; Crea 100 nuevas sillas usando un bucle FOR ... NEXT sobre una colección llamada ROOM (HABITACION)

For tempx = 1 to 10
For tempy = 1 to 10
room.chair = New Chair
room\x = tempx
room\y = tempy
room\height = Rnd(0,10) ; establece la altura aleatoriamente entre 0 t 10
Next
Next

; Las mueve todas una posición (como en la descripción del ejemplo)

For room.chair = Each chair
room\x = room\x + 1
Next
```

Field variable

Definición:
Asigna una variable para usar dentro de una estructura de objetos TYPE.

Descripción de los Parámetros:
variable = una variable válida

Descripción del Comando:
Si no has leído el comando TYPE, deberías hacerlo antes de continuar.
Cuando defines un Type personalizado, necesitas asignar algunas variables en su interior. Usando el comando Field dentro de los comandos Type .. End Type, estableces una variable que puede ser usada en CADA objeto creado con el comando NEW.

Ejemplo:
Ver Ejemplo TYPE

End Type

Definición:
El último comando de la creación de un objeto TYPE ... END TYPE.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Si no has leído el comando TYPE, deberías hacerlo antes de continuar.
Después de asignar todas las variables Field en tu objeto TYPE, úsalo para acabar la creación del Type.

Ejemplo:
Ver Ejemplo TYPE

New type_variable

Definición:
Crea un nuevo objeto TYPE.

Descripción de los Parámetros:
type_variable = el actual nombre Type

Descripción del Comando:
Si no estás familiarizado con el comando TYPE, por favor, échale un vistazo antes de continuar leyendo acerca de este comando.
Crea un NUEVO objeto en una colección Type. Cada llamada a este comando automáticamente inserta un nuevo objeto en la colección Type especificada. Comprueba el ejemplo y otros comandos Type para más información.

Ejemplo:
Ver Ejemplo TYPE

Each type_variable

Definición:
Usado para moverse a través de una colección de objetos TYPE.

Descripción de los Parámetros:
type_variable = Un objeto TYPE declarado anteriormente

Descripción del Comando:
Si no has leído el comando TYPE, deberías hacerlo antes de continuar.
El bucle For .. Each te permite ir a través de cada objeto en la colección TYPE. Esto es perfecto para actualizar un gran grupo de objetos (como un grupo de aliens invasores). Echa un vistazo al comando Type.

Ejemplo:
Ver Ejemplo TYPE

First type_variable

Definición:
Mueve el puntero de objetos Type al primer objeto de la colección.

Descripción de los Parámetros:
type_variable = el nombre actual del Type, no el nombre personalizado

Descripción del Comando:
Si no has leído el comando TYPE, deberías hacerlo antes de continuar.
Úsalo para asignar un objeto Type personalizado al primer objeto de la colección. Mira el ejemplo.

Ejemplo:
Ver Ejemplo TYPE

Last type_variable

Definición:
Mueve el el puntero de objeto al último objeto de la colección Type.

Descripción de los Parámetros:
type_variable = el nombre actual Type, no el nombre personalizado

Descripción del Comando:
Si no has leído el comando TYPE, deberías hacerlo antes de continuar.
Úsalo para asignar un objeto Type al último objeto de la colección. Mira el ejemplo.

Ejemplo:
Ver Ejemplo TYPE

Before custom_type_variable

Definición:
Mueve el puntero del obeto al anterior objeto en la colección Type.

Descripción de los Parámetros:
custom_type_variable = no el nombre del Type, pero el nombre del Type personalizado sí

Descripción del Comando:
Si no has leído el comando TYPE, hazlo si quieres hacer algo antes de continuar.
Úsalo para asignar un objeto Type personalizado al anterior objeto de la colección. Mira el ejemplo.

Ejemplo:
Ver Ejemplo TYPE

After custom_type_variable

Definición:
Mueve el puntero del obeto al siguiente objeto en la colección Type.

Descripción de los Parámetros:
custom_type_variable = no el nombre del Type, pero el nombre del Type personalizado sí

Descripción del Comando:
Si no has leído el comando TYPE, hazlo si quieres hacer algo antes de continuar.
Úsalo para asignar un objeto Type personalizado al siguiente objeto de la colección. Mira el ejemplo.

Ejemplo:
Ver Ejemplo TYPE

Delete custom_type_name

Definición:
Borra un objeto de una colección TYPE personalizada.

Descripción de los Parámetros:
custom_type_name = el nombre de un objeto existente

Descripción del Comando:
Si no has leído el comando TYPE, deberías hacerlo antes de continuar.
Usa el comando Delete para borrar un objeto de una colección TYPE. Usa los comandos FIRST, LAST, BEFORE, y NEXT para moverte al puntero del objeto que deseas borrar.
Éste es a menudo usado en un bucle FOR ... EACH cuando hay una colisión y deseas quitar el objeto de la colección.

Ejemplo:

```
; Las mueve una posición (como en la descripción del ejemplo en el comando TYPE)
; Si la silla no esta en pantalla, se elimina

For room.chair = Each chair
room\x = room\x + 1
if room\x > 640 then
Delete room
Next
```

True

Definición:
Una expresión booleana usada en comparaciones. Actualmente devuelve un valor de 1.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
TRUE es una palabra clave que indica un resultado positivo en una condición. A menudo, TRUE está implicado y no necesita ser referenciado directamente. TRUE puede usarse también como un valor RETURN de una FUNCIÓN. Mira el ejemplo.

Ejemplo:

```
; Ejemplo TRUE

; Asigna a test un número aleatorio (1 o 0)
test= Rnd(0,1)

; TRUE está implicado; Esta línea significa REALLMENTE: si test es igual a 1 de verdad (TRUE)
entonces continua
If test=1 Then
Print "Test es igual a 1"
End If

; Establezcamos test a TRUE
```

```
test=True

If test=True Then
Print "Test es true"
End If
```

False

Definición:
Una expresión booleana usada en las comparaciones. Actualmente devuelve un valor de 0.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
FALSE es una palabra clave que denota un resultado negativo en una condición. Muchas veces, FALSE está implicado y no necesita hacerle referencia (se puede usar el comando NOT en la comparación). FALSE puede también ser usado como un valor RETURN desde una FUNCTION. Mira también el comando TRUE. Mira el ejemplo.

Ejemplo:

```
; Ejemplo FALSE

; Asigna a test un numero aleatorio de 1 o 0
test= Rnd(0,1)

; FALSE esta implicado porque esta el NOT
If not test=1 Then
Print "Test es igual a 0"
End If

; Establezcamos a 'tst' como FALSE
test=False

If test=False Then
Print "Test es FALSE"
else
print "Test es TRUE"
End If
```

Null

Definición:
Indica un valor nulo o vacío.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Designa un valor nulo. Útil para borrar una variable o para probar si una variable tiene algún valor.

Ejemplo:

```
; Ejemplo Null

strHello$="Hola Mundo"

Print strHello$

strHello$=Null
```

Data list_of_values

Definición:
Crea una lista de valores constantes para ser usados en tu programa.

Descripción de los Parámetros:
list_of_values = una lista de los valores separados por comas (las cadenas de texto deben ir entre comillas)

Descripción del Comando:
Data se usa para crear ordenadamente una lista de valores constantes separados por comas que serán leídos (y probablemente reusados) durante la ejecución del programa. Puedes almacenar información del nivel (número de enemigos, estrellas, etc.) o lo primero que se te ocurra! Puedes mezclar fácilmente diferentes tipos de datos (cadenas de texto, enteros, reales), tan largo como sean leídos más tarde con el comando Read. Necesitarás usar el comando Restore para apuntar a la .Label desde donde comienza la declaración del Data. Mira los comandos mencionados para más información y ejemplos.

Ejemplo:

```
Print "¡Allá vamos!"

; Restaura al inicio las declaraciones Data
Restore startData

; Obtiene el primer numero el cual es el total de usuarios
Read Users

; Imrpimelos todos!
For T = 1 To Users
Read firstname$
Read age
Read accuracy#
Read lastname$
Print firstname$ + " " + lastname$ + " tiene " + age + " años con " + accuracy# + " accuracy!"
Next

While Not KeyHit(1)
Wend
End

.startData
Data 3
Data "Sergio", 14, 33.3333, "Padrino"
Data "Bob", 28, 12.25, "Smith"
Data "Roger", 54, 66.66, "Rabbit"
```

Read variable

Definición:
Obtiene el siguiente lote de valores de un estamento Data.

Descripción de los Parámetros:
variable = variable válida, debe coincidir el tipo de datos que estás leyendo (entero, texto, etc.)

Descripción del Comando:
Lee el siguiente valor en un estamento Data. Esto te permite almacenar grandes bloques de información constantes, y luego recuperar esta información fácilmente.
Al contrario que la mayoría de los lenguajes BASIC, los estamentos Data no tienen que ser linear y secuencial. Mira Data, Restore o .Label para más información.
Nota: puedes leer múltiples valores a la vez; Leer X,Y,Z por ejemplo.

Ejemplo:

```
; Ejemplo de read/restore/data/label

; Pongamos el puntero data al segundo data
Restore seconddata

; Imprimamos todos en la pantalla
For t = 1 To 10
Read num ; Obtiene el siguiente valor data
Print num
Next
```

```
; Ahora el primer data
Restore firstdata

; Imrpimamo todo en la pantalla
For t = 1 To 10
Read num ; Obtiene el siguiente valor del data
Print num
Next

; Este es el primer grupo data
.firstdata
Data 1,2,3,4,5,6,7,8,9,10

; Este es el segundo grupo data
.seconddata
Data 11,12,13,14,15,16,17,18,19,20
```

Restore label

Definición:
Mueve el puntero Data a la etiqueta seleccionada.

Descripción de los Parámetros:
label = etiqueta válida

Descripción del Comando:
Cuando uses Data para almacenar grandes bloques de constantes para usar con el comando Read, es necesario especificar el inicio del Data con una .etiqueta. El comando Restore mueve el puntero a la primera declaración Data tomando como referencia la etiqueta especificada. DEBES usar Restore etiqueta antes que Read. Este método te permite almacenar grupos de declaraciones Data no-secuenciales. Es diferente (si has usado otros lenguajes BASIC) pero lo encontrarás más flexible. Mira el ejemplo y otros comandos relacionados con el comando Data.

Ejemplo:
Ver Ejemplo READ

Include filename

Parámetros:
filename - nombre del archivo .bb para incluir

Descripción:
Incluye el contenido del .bb especificado al actual archivo de código.
Nota: los archivos .bb sólo pueden ser incluidos una vez.

Ejemplo:

```
; Ejemplo Include
; ---------------

Include "print.bb"
```

MATEMÁTICOS

Pi

Definición:
Devuelve el valor de Pi.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Devuelve seis dígitos del valor de Pi (3.141592), necesario para rutinas geométricas.

Ejemplo:

```
; Ejemplo Pi

Print "El valor de Pi es: " + Pi
```

Int (value)

Parámetros:
value - algún valor o variable válida

Descripción:
Este valor redondea al número entero más cercano y lo devuelve.

Ejemplo:

```
; Int Example
; -----------

Print Int(0.25) ; = 0
Print Int(0.5) ; = 0
Print Int(0.75) ; = 1
```

Float variable/value

Definición:
Convierte un valor entero en uno de coma flotante (real).

Descripción de los Parámetros:
variable/value = valor entero o variable

Descripción del Comando:
Usa este comando para transformar un valor entero en uno de coma flotante.

Ejemplo:

```
; Float example

a=100
b#=2.5
c#=Float a

Print b# + c#
```

Floor# (float)

Definición:
Redondea una variable decimal a la baja.

Descripción de los Parámetros:
float=número de coma flotante

Descripción del Comando:
Muchas veces necesitarás redondear a la alta o a la baja un número decimal. Este comando los redondea a la baja. Usa Ceil# para redondear a la alta.

Ejemplo:

```
; Ejemplo Floor#/Ceil#
; Creo una variable de coma flotante
myVal#=0.5

; La redondeo hacia arriba
Print Floor#(myval)

;La redondeo hacia abajo
Print Ceil#(myval)
```

Ceil# (float)

Definición:
Redondea un valor decimal al número más cercano a la alta.

Descripción de los Parámetros:
float = número real

Descripción del Comando:
Muchas veces, necesitarás redondear un número a la alta o a la baja. Este comando redondea un número a la alta al número más cercano. Usa Floor# para redondear el número a la baja.

Ejemplo:

```
; Ejemplo Floor#/Ceil#

; Creo una variable de coma flotante
myVal#=0.5

; La redondeo hacia arriba
Print Floor#(myval)

;La redondeo hacia abajo
Print Ceil#(myval)
```

Sgn (number)

Definición:
Devuelve el signo del número especificado.

Descripción de los Parámetros:
number=flotante o entero

Descripción del Comando:
Esta función se usa para saber si un valor o número es mayor, menor o igual a 0.

Ejemplo:

```
Print Sgn(10) ; prints 1
Print Sgn(5.5) ; prints 1.000000
Print Sgn(0) ; prints 0
Print Sgn(0.0) ; prints 0.000000
Print Sgn(-5.5) ; prints -1.000000
Print Sgn(-10) ; prints -1
```

Abs (número)

Definición:
Devuelve el valor absoluto (positivo) de un número.

Descripción de los Parámetros:
número = algún número válido o variable numérica

Descripción del Comando:
Usa este comando para devolver el valor absoluto de un número, es decir, su valor positivo. Un -3 se convertiría en un 3. Si lo que quieres es un número sin decimales (p. e. convertir 3.1415 en 3) usa el comando Int().

Ejemplo:

```
number=-3

Print "El valor absoluto de " + number + " es: " + Abs(number)

WaitKey()
```

Mod

Definición:
Devuelve el resto de una división.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Simplemente, devuelve el resto de una división. Por ejemplo de la división entre 10 y 3 (10 MOD 3) devolvería 1.

Ejemplo:

```
; Ejemplo MOD

Print 10 MOD 3
```

Sqr (float)

Definición:
Devuelve la raíz cuadrada del valor especificado.

Descripción de los Parámetros:
float = número de coma flotante (los enteros se convierten al vuelo)

Descripción del Comando:
Este comando devolverá la raíz cuadrada de un número específico. El valor devuelto será un úmero de coma flotante.

Ejemplo:

```
; sqr# example

value=25

print "La raíz cuadrada de nuestro valor es: " + sqr#(value)
waitkey()
```

Sin (Number)

Definición:
El comando Sin, o Sinus, es una función de trigonometría que devuelve un número entre -1 y 1. Este valor representa la coordenada "Y" de un punto.

Descripción de los Parámetros:
number=flotante o entero que representa un valor en grados

Descripción del Comando:
Este comando se usa para trasladar valores de ángulos a coordenadas, pero hay algunas cosas que deberías tener en cuenta cuando lo uses. Lo primero de todo el comando Sin() asume el punto que quieres es de radio 1 (pixel), después usa un círculo donde 0 grados es el punto más al ESTE e incrementa en el sentido contrario de las agujas del reloj, entonces tienes que tener en cuenta que el eje Y en una pantalla de ordenador es arriba-a-abajo comparado con un sistema de coordenadas matemático normal.
Mira también ASin, Cos, ACos, Tan, Atan, ATan2

Ejemplo:

```
Graphics 640,480
Origin 320,240 ; Mueve el punto de origen de todos los comandos de dibujo al centro de la pantalla

For degrees=0 To 359; Pasa a través de todos los grados del círculo
Delay(5)

; Luego calcula la coordenada Y del punto del círculo usando Sin y multiplicando por 100 (para
hacer mas grande el radio)
y=Sin(degrees)*100

y=-y ; Invierte la coordenada Y para representar el círculo correctamente en la pantalla
```

```
; La siguiente linea calcula la coordenada X usando el comando COS y multiplicando por 100
x=Cos(degrees)*100

Rect x,y,1,1 ; Dibuja el punto del circulo actual

Next ; Nos da otro angulo

MouseWait ; Espera que toques un boton del raton
End ; Termina la ejecución del programa
```

Cos (Number)

Definición:
El comando Cos es una función de trigonometría que devuelve un número entre -1 y 1. Este valor representa la coordenada X de un punto (x,y).

Descripción de los Parámetros:
number=real o entero que representa un valor en grados

Descripción del Comando:
Este comando se usa para trasladar ángulos en coordenadas, pero hay unas pocas cosas que tienes que tener en cuenta cuando lo utilices. Primero, todos los comandos Cos() suponen que el punto que quieres es de radio 1 (pixel), usa un círculo donde 0 grados representa la derecha y se incrementa en el sentido CONTRARIO de las agujas del reloj, entonces tienes que tener en cuenta que el eje Y en una pantalla de ordenador está al revés que en un sistema de coordenadas matemático.
Mira también ASin, Cos, ACos, Tan, Atan, ATan2

Ejemplo:
Ver Ejemplo Sin

Tan (number)

Definición:
Devuelve la tangente de un ángulo.

Descripción de los Parámetros:
number=flotante o entero representando un valor en grados

Descripción del Comando:
Tan coge un ángulo y devuelve la relación de dos lados de un triángulo rectángulo. La relación es el resultado de la longitud del lado opuesto al ángulo dividido por el lado contiguo (el que no es ni la hipotenusa ni el opuesto al ángulo).
Esto es útil para juegos que usen armas donde las balas serán disparadas hacia un barril en un cierto ángulo. Puedes usar el valor Tan en tu fórmula de potencia/ángulo para ayudar a determinar la trayectoria del disparo.

Ejemplo:

```
; Ejemplo Tan

angle1=25
angle2=45

print "La tangente del ángulo 1 es: " + tan(angle1)
print "La tangente del ángulo 2 es: " + tan(angle2)
waitkey()
```

ASin (número)

Definición:
Devuelve el arcoseno del parámetro especificado.

Descripción de los Parámetros:
número=flotante o entero representando un radio de la coordenada "Y" al desplazamiento tagencial.

Descripción del Comando:
Este comando se usa para trasladar los valores de las coordenadas X/Y a ángulos. Recuerda que la pantalla del ordenador usa un eje Y invertido (los valores son mayores cuanto más abajo estén).

Ejemplo:

```
Graphics 640,480
Repeat
Cls
; Determina las posiciones y la longitud de la hipotenusa
x# = MouseX() ; desplazamientto horizontal por la pantalla
y# = MouseY() ; desplazamientto vertical por la pantalla
r# = Sqr((x#*x#)+(y#*y#)) ; longitud de la hipotenusa
; Dibuja los ejes
Color 104,104,104
Line x#,0,x#,y# ; dibuja una linea perpendicular al cursor desde la parte superior de la pantalla
Line 0,y#,x#,y# ; dibuja una linea perpendicular al cursor desde la parte izquierda de la pantalla
Locate x#+10,y#-10 : Write "X=" : Print x#
Locate x#-10,y#+10 : Write "Y=" :Print y#

Origin 0,0

; dibuja la línea del ángulo
Color 255,255,255
Line 0,0,x#,y# ; dibuja una linea desde la esquina superior izquierda hasta el cursor
theta# = ASin(y#/r#) ; el angulo entre el eje x y el eje y
Locate 60,10 : Write "Ángulo:" : Print theta

; Dibuja un arco representando el ángulo
For degrees#=0 To theta#; pasa a través de todos los grados en el ángulo
; La siguiente linea calcula las coordenadas X e Y del punto del circulo usando los comandos Sin y
Cos
; y multiplicandolo por 5 el resultado (para obtener un radio mayor)
cy=Sin(degrees#)*50
cx=Cos(degrees#)*50
Plot cx,cy ; Dibuja el punto actual del círculo.
Next

Flip
; Pulsa ESC para salir
Until KeyDown(1)
End
```

ACos (número)

Definición:
Devuelve el arcocoseno del argumento especificado.

Descripción de los Parámetros:
número = flotante o entero representando un radio de la coordenada "X" al desplazamiento tagencial.

Descripción del Comando:
Este comando se usa para trasladar las coordenadas X/Y a ángulos. Recuerda que la pantalla del ordenador usa un eje Y invertido (los valores son mayores cuanto más abajo esté).

Ejemplo:
Ver Ejemplo Asin

ATan (flotante)

Definición:
Devuelve el ángulo desde el eje X hasta un punto (y,x).

Descripción de los Parámetros:
flotante = punto en valor flotante (grados)

Descripción del Comando:
Se usa en la trigonometría básica, este comando te permitirá derivar un ángulo basado en la velocidad X e Y de un objeto en movimiento. Mira también ATan2.

Ejemplo:

```
; Ejemplo Atan

Graphics 640,480,16,0
SetBuffer BackBuffer()

; Punto de inicio de nuestra caja movil
x=0
y=0

; velocidades aleatorias
xSpeed=Rand(5)
ySpeed=Rand(5)

; repite hasta que pulses ESC o la caja salga de la pantalla
While Not KeyHit(1) Or y > 480 Or x > 640
Cls
; Dibuja nuestra caja
Rect x,y,10,10,1
; incrementa la posicion de la caja dependiendo de las velocidades aleatorias
x=x+xSpeed
y=y+yspeed
; imprimo el ángulo de nuestra caja
Text 0,0,ATan(yspeed,xspeed)
Flip
Wend
```

ATan2 (valorx,valory)

Definición:
Devuelve el ángulo desde el eje X hasta un punto (X,Y).

Descripción de los Parámetros:
(valory,valorx)

Descripción del Comando:
Se usa en la trigonometría básica, este comando te permitirá derivar un ángulo basado en la velocidad X e Y de un objeto en movimiento.

Ejemplo:

```
; Ejemplo Atan2

Graphics 640,480,16,0
SetBuffer BackBuffer()

; Punto de inicio de nuestra caja movil
x=0
y=0

; velocidades aleatorias
xSpeed=Rand(5)
ySpeed=Rand(5)

; repite hasta que pulses ESC o la caja salga de la pantalla
While Not KeyHit(1) Or y > 480 Or x > 640
Cls
; Dibuja nuestra caja
Rect x,y,10,10,1
; incrementa la posicion de la caja dependiendo de las velocidades aleatorias
x=x+xSpeed
y=y+yspeed
; imprimo el ángulo de nuestra caja
Text 0,0,ATan2(yspeed,xspeed)
Flip
Wend
```

Exp (number)

Definición:
Devuelve el valor de un logarismo de 'e' elevado a una potencia.

Descripción de los Parámetros:
number=real o entero

Descripción del Comando:
Exp() es una función exponencial. La función Exp() y Log usan numero e a partir de una base (aprox. 2.718). Exp() es la inversa de la función Log. mirar en: Log Log10
Gracias a xuri por ayudarme con la traducción de este comando.

Ejemplo:

```
; Para encontrar el valor de e
print Exp(1) ; i.e. e^1, devuelve 2.718281
; Para encontrar el valor de e^5
print Exp(5) ; devuelve 148.413162
```

Log (number)

Definición:
Devuelve el logaritmo natural del argumento especificado.

Descripción de los Parámetros:
number=flotante o entero

Descripción del Comando:
Log() es la función logarítmica (o inversa exponencial). Los logaritmos de base 'e' son llamados Logaritmos naturales. Las funciones Exp() y Log() usan el número trascendental 'e' como una base (aprox. 2.718). Log() es la función inversa de Exp(). Log(e) = 1 Mira también: Exp Log10

Ejemplo:

```
; Para encontrar el valor de Log(e^5)
e# = Exp(1)
x# = 5
y# = e#^x#
Print y ; i.e. e^5 = e*e*e*e*e: devuelve 148.413131
Print Log(y) ; ie. Log(e^5) = Log(10000): devuelve 5.000000
```

Log10 (number)

Definición:
Devuelve el logaritmo común del argumento proporcionado (base 10).

Descripción de los Parámetros:
number=flotante o entero

Descripción del Comando:
Log10() es la función logarítmica (o inversa exponencial) de base 10. Los logaritmos de base 10 son llamados Logaritmos comunes. La función Log10() usa el 10 como base. Siguen series de 10^x. Esta función es útil para encontrar la potencia de un número de base 10. Mira también: Exp Log

Ejemplo:

```
; Para encontrar el valor de Log10(10^5)
x = 5
y = 10^x
Print y ; i.e. 10^5 = 10*10*10*10*10: devuelve 100000
Print Log10(y) ; ie. Log10(10^5) = Log10(10000): devuelve 5.000000
```

Xor

Definición:
Realiza un OR exclusivo a nivel de bit entre dos valores.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Se usa a menudo para encriptaciones ligeras (mejor dicho insignificantes ;), cogerá dos valores y realizará un OR exclusivo con cada bit siguiendo las reglas básicas del XOR. Las reglas básicas del Xor son:
```
0 Xor 0 = 0
1 Xor 0 = 1
0 Xor 1 = 1
1 Xor 1 = 0
```
Es decir, sólo se debe cumplir UNA de las condiciones. Mira el ejemplo para más información.

Ejemplo:

```
num=%11110000111100001111000011110000 ; Define un patrón de bits fácil de reconocer

bitmask=Rnd(-2147483648,2147483647) ; Define un patrón aleatorio de 32bit con el que realizar la
comparación (lo llamaremos 'bitmask')

; Esta línea imprime los valores binario y decimal antes de realizar el Xor
Print "El Número binario es: "+Bin$(num)+" ("+num+")"

; Esta línea imprime los valores binario y decimal del bitmask antes de realizar el Xor
Print "El bitmask del Xor es: "+Bin$(bitmask)+" ("+bitmask+")"

Print "------------------------------------------------------------------"

; Esta línea realiza la comparación xor del número con el bitmask
xres=num Xor bitmask

; Esta línea imprime el valor en binario y decimal después del Xor
Print "El resultado del Xor es: "+Bin$(xres)+" ("+xres+")"
Print "------------------------------------------------------------------"

; Esta línea realiza un Xor con el resultado anterior y el bitmask otra vez
xres=xres Xor bitmask
; Esta línea imprime el valor en binario y decimal después del segundo Xor. NOTA: Este número es
idéntico al número original.
Print "Resultado del segundo Xor es: "+Bin$(xres)+" ("+xres+")"

WaitMouse ; Espera a que pulses un botón del ratón antes de acabar
```

Shl repetitions

Definición:
Realiza un desplazamiento binario a la izquierda.

Descripción de los Parámetros:
repetitions = nº de desplazamientos a realizar

Descripción del Comando:
Realiza un desplazamiento binario a la izquierda en el valor el número especificado de veces. Esto básicamente es un método rápido de multiplicar un valor exponencialmente. Desplazando una vez a la derecha, estás multiplicando por 2 el valor. Desplazando dos veces, multiplicas por 4, etc.

Ejemplo:

```
; Ejemplos shl, shr, sar

value = 100

; multiplica por 2
Print "Mover 1 bit a la izquierda; Valor = " + value Shl 1
; multiplica por 4
Print "Mover 2 bits a la izquierda; Valor = " + value Shl 2
; multiplica por 16
Print "Mover 4 bits a la izquierda; Valor = " + value Shl 4
```

```
; divide entre 2
Print "Mover 1 bit a la derecha; Valor = " + value Shr 1
; divide entre 4
Print "Mover 2 bits a la derecha; Valor = " + value Shr 2
; divide entre entre 16
Print "Mover 4 bits a la derecha; Valor = " + value Shr 4

Print "Mover con SAR 4 veces = " + value Sar 4

WaitKey()
```

Shr repetitions

Definición:
Realiza un desplazamiento binario a la derecha.

Descripción de los Parámetros:
repetitions = nº de desplazamientos a realizar

Descripción del Comando:
Realiza un desplazamiento binario a la derecha en el valor el número especificado de veces. Esto básicamente es un método rápido de dividir un valor exponencialment. Desplazando una vez a la derecha, estás dividiendo entre 2 el valor. Desplazando dos veces, divides entre 4, etc.

Ejemplo:
Ver Ejemplo Shl

Sar repetitions

Definición:
Realiza un desplazamiento binario a la derecha.

Descripción de los Parámetros:
repetitions = nº de desplazamientos a realizar

Descripción del Comando:
Realiza un desplazamiento binario a la derecha en el valor el número especificado de veces. Esto básicamente es un método rápido de dividir un valor exponencialmente. Desplazando una vez a la derecha, estás dividiendo entre 2 el valor. Desplazando dos veces, divides entre 4, etc.
El comando Sar varía con respecto a Shr en que rellena los bit vacíos con copias del signo, 0 para positivos y 1 para negativos. Mira también Shl.

Ejemplo:
Ver Ejemplo Shl

Rnd (start#,end#)

Definición:
Devuelve un número aleatorio.

Descripción de los Parámetros:
start# = límite inferior
end# = límite superior

Descripción del Comando:
Devuelve un número real o entero generado aleatoriamente. El tipo de número dependerá de la variable a la que se lo asignes. Asegúrate de usar el comando SeedRnd para que los números aleatorios no sean iguales cada vez que ejecutes el programa.

Ejemplo:

```
y=Rnd(0,10) ; Establece a Y a un entero aleatorio entre 0 y 10
y#=Rnd(0,5) ; Establece a Y un valor de coma flotante entre 0.000000 y 10.000000
```

Rand ([low value],high value)

Definición:
Genera un entero aleatorio entre los valores especificados.

Descripción de los Parámetros:
low value = opcional - por defecto 1; valor más bajo
high value = valor más alto

Descripción del Comando:
Al contrario que el comando RND, este comando actualmente devuelve sólo valores enteros. El valor más bajo es por defecto 1 si no se especifica ninguno. El valor más alto es el límite del número generado. Si necesitas generar números flotantes, usa Rnd.

Ejemplo:

```
; Ejemplo Rand

; Establece la semilla de numeros aleatorios para que sean nuevos
SeedRnd (MilliSecs())

; Genera números aleatorios entre 1 y 100
For t = 1 To 20
Print Rand(1,100)
Next
```

SeedRnd seed

Definición:
Establece el conjunto de números aleatorios para el generador con un valor 'semilla'.

Descripción de los Parámetros:
seed = valor entero

Descripción del Comando:
Los generadores de números aleatorios en los ordenadores no son realmente aleatorios. Generan números basados en un valor 'semilla' (un número entero). Si no cambias esta 'semilla', siempre devolverá la misma serie de números. Usa este comando para asegurarte de que los números que obtienes son siempre distintos. Normalmente establecerás el valor 'semilla' al reloj del sistema para segurarte de que es distinto en cada ejecución. Mira el ejemplo para comprobar el uso más corriente de este comando.

Ejemplo:

```
SeedRnd Millisecs() ; Cambia la semilla con el tiempo de sistema actual en milésimas de segundo.
```

CADENAS

Str variable/value

Definición:
Convierte un valor numérico en una cadena de texto.

Descripción de los Parámetros:
variable/value = valor o variable numérica

Descripción del Comando:
Usa este comando para transformar un valor numérico en una cadena de texto . Blitz imprime valores numéricos sin problemas, pero podrías querer usar funciones como LEFT$, que sólo funcionan en cadenas de texto.

Ejemplo:

```
; Ejemplo STR

num#=2.5
mynum$=str num#

Print mynum$
```

Left$ (string$, length)

Definición:
Devuelve un cierto número de letras de una cadena de texto, empezando a contar por la izquierda.

Descripción de los Parámetros:
string$ = una cadena de texto válida
length = un número entero que indica la longitud de la cadena a extraer

Descripción del Comando:
Usa este comando para obtener un determinado número de letras de una cadena de texto, empezando a contar por la izquierda.

Ejemplo:

```
name$="Sergio Padrino"
Print "Las tres primeras letras de tu nombre son: " + Left$(name$,3)
```

Right$ (string$, length)

Definición:
Devuelve un número específico de carácteres empezando a contar por la derecha.

Descripción de los Parámetros:
string$ = cadena de texto
length = número de caracteres a devolver

Descripción del Comando:
Puedes obtener un cierto número de caracteres desde la derecha de una cadena de texto. Mira el ejemplo.

Ejemplo:

```
name$="Sergio Padrino"
Print "Las cuatro últimas letras de tu nombre son: " + Right$(name$,4)
```

Mid$ (string$, offset, characters)

Definición:
Devuelve un número en concreto de carácteres dentro de una cadena de texto, desde una posición en particular.

Descripción de los Parámetros:
string$ = una cadena válida
offset = posición dentro de la cadena para empezar a leer
characters = cuántos carácteres se leerán desde la posición de comienzo

Descripción del Comando:
Usa este comando para obtener una cantidad de carácteres del interior de una cadena de texto. Puedes elegir DONDE se empezará a leer dentro de la cadena , y cuántos carácteres recogerás. Probablemente usarás este comando para descodificar una cadena y obtener cada carácter para realizar una conversión o una validación. Mira el ejemplo.

Ejemplo:

```
name$="Sergio Padrino"
For T = 1 To Len(name$)
Print Mid$(name$,t,1)
Next
```

Replace$ (string$, find$, replace$)

Definición:
Remplaza las ocurrencias encontradas de una cadena en otra.

Descripción de los Parámetros:
string$ = cadena de texto válida
find$ = cadena de texto válida
replace$ = cadena de texto válida

Descripción del Comando:
Este comando te permitirá remplazar caracteres dentro de una cadena dentro de otra. Úsalo para convertir letras de tus cadenas de texto (como quitar espacios o convertirlos en subrayados '_').

Ejemplo:

```
name$="Sergio Padrino"
Print "Tu nombre antes del reemplazo: " + name$
Print "Tu nombre con la R cambiada por la L: " + Replace$(name$,"r","l")
```

Instr (string1$, string2$, offset)

Definición:
Encuentra la posición de un texto dentro de otro.

Descripción de los Parámetros:
string1$ = texto en el que quieres buscar
string2$ = texto que quieres buscar
offset = valor entero que indica la posición desde donde se comenzará a buscar (opcional)

Descripción del Comando:
Este comando te permitirá buscar un texto en otro. El comando devolverá la localización (número de letras desde la izquierda) de la cadena que estás buscando. El comando devuelve 0 si no encuentra nada.

Ejemplo:

```
name$="Sergio Padrino"
Print name$
location = Instr( name$,"P",1)
Print "Tu nombre tiene la 'P' en la posición número " + location + "!"
```

Upper$ (string$)

Definición:
Convierte una cadena de texto entera a mayúsculas.

Descripción de los Parámetros:
string$ = una cadena de texto válida

Descripción del Comando:
Este comando coge la cadena de texto dada y la convierte entera a mayúsculas.

Ejemplo:

```
name$="Sergio Padrino Recio"
print "Tu nombre entero en mayúsculas es: " + upper$(name$)
```

Lower$ (string$)

Definición:
Convierte una cadena de texto entera a minúsculas.

Descripción de los Parámetros:
string$ = una cadena de texto válida

Descripción del Comando:
Cogerá una cadena de texto y la convertirá entera a minúsculas.

Ejemplo:

```
name$="SeRgIo PaDrInO"
Print "El nombre original es: " + name$
Print "En minúsculas es: " + Lower$(name$)
```

Trim$ (string$)

Definición:
Quita los espacios iniciales y finales de una cadena de texto.

Descripción de los Parámetros:
string$ = cadena de texto

Descripción del Comando:
Usa este comando para quitar los espacios iniciales y finales de una cadena de texto.

Ejemplo:

```
name$=" Sergio Padrino Recio "
Print "Tu nombre antes del arreglo: '" + name$ "' ..."
Print "Tu nombre arreglado es: '" + Trim$(name$) + "' ..."
```

LSet$ (string$, length)

Definición:
Completa una cadena de texto con el número de espacios especificado, alineando a la izquierda el texto.

Descripción de los Parámetros:
string$ = cadena de texto válida
length = número de espacios

Descripción del Comando:
Si tienes una cadena de texto de, por ejemplo, 10 carácteres de largo, y quieres completarla hasta tener 25 letras, rellena el resto de huecos con espacios. Este comando lo hará por ti, dejando la cadena original alineada a la izquierda. Si ejecutas el ejemplo lo entenderás mejor.

Ejemplo:

```
name$="Sergio Padrino Recio"
Print "Nueva cadena de texto: '" + LSet$(name$,40) + "'"
```

RSet$ (string$, length)

Definición:
Completa una cadena de texto con el número de espacios especificado, alineando a la derecha el texto.

Descripción de los Parámetros:
string$ = cadena de texto válida
length = número de espacios

Descripción del Comando:
Si tienes una cadena de texto de, por ejemplo, 10 carácteres de largo, y quieres completarla hasta tener 25 letras, rellena el resto de huecos con espacios. Este comando lo hará por ti, dejando la cadena original alineada a la derecha. Si ejecutas el ejemplo lo entenderás mejor.

Ejemplo:

```
name$="Sergio Padrino"
Print "Nueva cadena de texto: '" + RSet$(name$,40) + "'"
```

Chr$ (integer)

Definición:
Convierte un código ASCII en su correspondiente carácter.

Descripción de los Parámetros:
integer = código ASCII válido

Descripción del Comando:
Usa este comando para convertir un código ASCII conocido (por ejemplo 65) en su carácter equivalente (en este caso la letra 'A').

Ejemplo:

```
Print " El carácter del valor ASCII 65 es: " + Chr$(65)
```

Asc (cadena$)

Definición:
Devuelve el valor ASCII de la primera letra de una cadena.

Descripción de los Parámetros:
cadena$ = alguna variable de cadena válida (sólo el valos ASCII del primer carácter será devuelto).

Descripción del Comando:
Éste devolverá el valor ASCII de la primera letra de una cadena.

Ejemplo:

```
a$=Input$("Introduce una letra:")
Print "El valor ASCII de esa letra es:" + Asc(a$)
```

Len (string$)

Definición:
Devuelve el número de carácteres de una cadena de texto.

Descripción de los Parámetros:
string$ = una cadena de texto válida

Descripción del Comando:
Este comando te permitirá saber la longitud (número de letras, espacios, carácteres, números, etc.) dentro de una cadena de texto. Puedes usar este comando para asegurarte de que el jugador introduce un número en concreto de letras (por ejemplo 3 letras para la tabla de récord's).

Ejemplo:

```
name$="Sergio Padrino"
Print "Hay " + Len(name$) + " letras en tu nombre."
```

Hex$ (integer)

Definición:
Convierte un valor entero en uno hexadecimal.

Descripción de los Parámetros:
integer = un valor entero

Descripción del Comando:
Convierte valores enteros en valores hexadecimales. Si no sabes que es un valor hexadecimal, no necesitarás este comando :)

Ejemplo:

```
intValue="64738"
Print "El valor hexadecimal de "+intValue+" es: " + hex$(intValue)
```

Bin$ (entero)

Definición:
Convierte un valor entero en un valor binario.

Descripción de los Parámetros:
entero = algún valor entero válido o una variable entera

Descripción del Comando:
Convierte valores enteros en valores binarios. Si no sabes lo que es binario, no necesitas conocer este comando :)

Ejemplo:

```
intValue="64738"
Print "El valor binario de "+intValue+" es: " + bin$(intValue)
```

String$ (string$, integer)

Definición:
Devuelve una cadena de texto un número de veces en concreto.

Descripción de los Parámetros:
string$ = cadena de texto
integer = número de veces a repetir el texto

Descripción del Comando:
Este comando crea una cadena de texto compuesta por el texto especificado repetido un número de veces en concreto. En otras palabras, puedes usar este comando para escribir una misma cadena de texto una y otra vez. Mira el ejemplo.

Ejemplo:

```
name$="Sergio"
' Escribe el nombre 10 veces
Print String$(name$,10)
```

TEXTO

Print [string$]

Parameter:
string$ (opcional) - cadena de texto o valor

Descripción:
Escribe una cadena de texto en el front buffer y comienza una nueva línea.
Si el parámetro opcional string es omitido, entonces el comando comenzará una nueva línea.
Mira también: Write.

Ejemplo:

```
; Ejemplo Print
; -------------

Print "Blitz "
Print "Basic"
```

Write string$

Parameter:
string$ - cadena de texto o valor numérico

Descripción:
Escribe una cadena de texto en el front buffer (es decir, la pantalla), pero no cambia de línea (al contrario que Print). Mira también: Print.

Ejemplo:

```
; Ejemplo Write
; -------------

Write "Blitz "
Write "Basic"
```

Locate x,y

Definición:
Posiciona los comandos de texto en la pantalla.

Descripción de los Parámetros:
x=coordenada x de la pantalla
y=coordenada y de la pantalla

Descripción del Comando:
Algunas veces quieres usar los comandos PRINT e Input$ en una posición específica de la pantalla. Este comando posiciona el 'cursor' en la localización especificada.

Ejemplo:

```
; Ejemplo Locate

strName$=Input$("¿Cómo te llamas?")

Locate 100,200

Print "Hola, " + strName$

While Not KeyHit(1)
Wend
```

Text x,y,string$,[center x],[center y]

Definición:
Escribe texto en la pantalla en las coordenadas especificadas.

Descripción de los Parámetros:
x = coordenada x para comenzar a escribir
y = coordenada y para comenzar a escribir
string$ = texto a escribir
center x = opcional; true = centra horizontalmente
center y = opcional; true = center verticalmente

Descripción del Comando:
Imprime un texto en las coordenadas de la pantalla especificadas. Puedes centrar el texto en las coordenadas estableciendo center x/center y en TRUE. El texto se dibujará en el color actual de dibujo.
Nota: Imprimiendo un espacio con texto NO se dibujará un bloque - un espacio es un valor vacío. Así imprimiendo " " no aparecerá una caja.

Ejemplo:

```
; Ejemplo Text

; Activo el modo gráfico
Graphics 800,600,16

; Espera que pulses la tecla ESC antes de acabar
While Not KeyHit(1)
;Imprime el texto, centrado horizontalmente en las coordenadas x=400, y=0
Text 400,0,"!Hola Mundo!",True,False
Wend
```

LoadFont (fontname$,height,bold,italic,underlined)

Parámetros:
fontname$ - nombre de la fuente a cargar, p.e. "arial"
height - altura en pixels de la fuente
bold - True para cargarla en negrita, False en caso contrario
italic - True para cargarla en cursiva, False en caso contrario
underlined - True para cargarla subrayada, False en caso contrario

Descripción:
Carga una fuente y devuelve su handle.
Puedes entonces usar el handle con comandos como SetFont y FreeFont.
Nota: Blitz no trabaja con fuentes de SÍMBOLOS, como Webdings y WingDings.

Ejemplo:

```
; Ejemplo LoadFont/SetFont/FreeFont
; -------------------------------
```

```
; Activa el modo Gráfico
Graphics 800,600,16

; Crea las variables globales para las fuentes
Global fntArial,fntArialB,fntArialI,fntArialU

; Carga las fuentes
fntArial=LoadFont("Arial",24,False,False,False)
fntArialB=LoadFont("Arial",18,True,False,False)
fntArialI=LoadFont("Arial",32,False,True,False)
fntArialU=LoadFont("Arial",14,False,False,True)

; Establece la fuente e imprime el texto
SetFont fntArial
Text 400,0,"Esta es la Arial 24 puntos",True,False

SetFont fntArialB
Text 400,30,"Esta es la Arial 18 puntos negrita",True,False

SetFont fntArialI
Text 400,60,"Esta es la Arial 32 puntos cursiva",True,False

SetFont fntArialU
Text 400,90,"Esta es la Arial 14 puntos subrayada",True,False

; Espera a que el usuario pulse ESC para salir
While Not KeyHit(1)
Wend

; ¡Borra todas las fuentes de la memoria!
FreeFont fntArial
FreeFont fntArialB
FreeFont fntArialI
FreeFont fntArialU
```

SetFont fonthandle

Definición:
Establece una fuente cargada previamente para operaciones de escritura.

Descripción de los Parámetros:
fontname = cadena de texto que contiene el nombre de la fuente
height = altura de la fuente
bold = TRUE para cargarla en NEGRITA
italic = TRUE para cargarla en CURSIVA
underlined = TRUE para cargarla en SUBRAYADA

Descripción del Comando:
Activa una fuente TrueType cargada previamente (a través del comando LoadFont) para futuras operaciones de escritura como Text.
Nota: Blitz no trabaja con fuentes de SÍMBOLOS, como Webdings y WingDings.
Asegúrate de liberar la memoria usada por la fuente cuando acabes de usarla mediante el comando FreeFont.

Ejemplo:
Ver Ejemplo LoadFont

SetFont fonthandle

Definición:
Establece una fuente cargada previamente para operaciones de escritura.

Descripción de los Parámetros:
fontname = cadena de texto que contiene el nombre de la fuente
height = altura de la fuente
bold = TRUE para cargarla en NEGRITA
italic = TRUE para cargarla en CURSIVA
underlined = TRUE para cargarla en SUBRAYADA

Descripción del Comando:
Activa una fuente TrueType cargada previamente (a través del comando LoadFont) para futuras operaciones de escritura como Text.
Nota: Blitz no trabaja con fuentes de SÍMBOLOS, como Webdings y WingDings.
Asegúrate de liberar la memoria usada por la fuente cuando acabes de usarla mediante el comando FreeFont.

Ejemplo:
Ver Ejemplo LoadFont

FontWidth()

Definición:
Devuelve el ancho de la fuente seleccionada actualmente.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Devuelve el ancho, en pixels, de la fuente seleccionada actualmente (usando SetFont - cargada previamente con LoadFont). Este comando devolverá el ancho del carácter MÁS ANCHO de la fuente actual.

Ejemplo:

```
; Ejemplo FontWidth()/FontHeight

; Activa el modo Gráfico
Graphics 800,600,16

; Crea las variables globales para las fuentes
Global fntArial

;Carga la fuente
fntArial=LoadFont("Arial",13,False,False,False)

; Establece la fuente e imprime los tamaños
SetFont fntArial
Text 400,0,"El ancho de la fuente es:"+ FontWidth(),True,False
Text 400,30,"El alto de esta fuente es:"+ FontHeight(),True,False

; Pulsa ESC para salir
While Not KeyHit(1)
Wend

; Borra todas las fuentes de la memoria!
FreeFont fntArial
```

FontHeight()

Definición:
Devuelve la altura de la fuente actualmente seleccionada.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Devuelve la altura, en pixels, de la fuente seleccionada actualmente (usando SetFont - cargada anteriormente con LoadFont).

Ejemplo:
Ver Ejemplo FontWidth

StringWidth (string)

Definición:
Devuelve la anchura (en pixels) de la cadena especificada.

Descripción de los Parámetros:
string = cadena de texto

Descripción del Comando:
Este comando devolverá la anchura, en pixels, de la cadena de texto especificada. Es útil para hacer scroll's con el texto, por ejemplo.

Ejemplo:

```
; Ejemplo StringWidth/Height

a$="¡Hola Sergio!"
Print "A$=" + a$
Print "Esta cadena tiene "+ StringWidth(a$) + " pixels de ancho y"
Print StringHeight(a$) + " de alto, según la fuente actual"
```

StringHeight (string)

Definición:
Devuelve la altura (en pixels) de la cadena especificada.

Descripción de los Parámetros:
string = cadena de texto

Descripción del Comando:
Este comando devolverá la altura, en pixels, de la cadena de texto especificada. Es útil para hacer scroll's con el texto, por ejemplo.

Ejemplo:
Ver Ejemplo StringWidth

ENTRADA

Input$ (prompt$)

Definición:
Obtiene una datos del usuario.

Descripción de los Parámetros:
prompt$ = una cadena válida (opcional)

Descripción del Comando:
El comando devolverá una cadena de texto del usuario con una línea de información opcional en la pantalla (si no esta en modo gráfico) o en el actual buffer de dibujo usado por el programa. Normalmente usarás estos datos más tarde.

Ejemplo:

```
; Obtiene el nombre del usuario y le saluda

name$=Input$("¿Cómo te llamas?")
Write "¡Hola, " + name$ + "!"
```

KeyDown (scancode)

Definición:
Devuelve TRUE si la tecla especificada del teclado está siendo pulsada.

Descripción de los Parámetros:
scancode = scancode correspondiente a la tecla

Descripción del Comando:
Este comando (similar a MouseDown y JoyDown) se usa para detectar si una tecla está siendo pulsada. Este comando devuelve un 0 si la tecla no está pulsada, 1 si lo está. Mira los ScanCodes.

Ejemplo:

```
; Ejemplo KeyDown()

Print "¡Mantén pulsada la tecla ENTER!"
Delay 3000
```

```
While Not KeyHit(1)
If KeyDown(28) Then
Print "¡¡Enter está siendo pulsado!!"
Else
Print
End If
Wend
```

KeyHit (scancode)

Definición:
Devuelve el número de veces que se ha pulsado una tecla determinada.

Descripción de los Parámetros:
scancode = scancode de la tecla a comprobar

Descripción del Comando:
Este comando devuelve el número de veces que una tecla en concreto ha sido pulsada la última vez que llamaste al comando KeyHit(). Mira los ScanCodes.

Ejemplo:

```
; Ejemplo KeyHit

; Crea un timer
current=MilliSecs()
Print "Pulsa ESC un montón de veces durante 5 segundos..."

; Espera 5 segundos
While MilliSecs() < current+5000
Wend

; Imrpime los resultados
Print "ESC fue pulsado " + KeyHit(1) + " veces."
```

GetKey()

Definición:
Comprueba si se pulsa una tecla y devuelve su valor ASCII.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando comprobará si una tecla ha sido pulsada y devuelve su valor ASCII. No todas las teclas tienen valores ASCII (si necesitas usar SHIFT, ALT u otra tecla sin valor ASCII usa KeyHit o KeyDown).

```
Ejemplo:

; Ejemplo GetKey

Print "Por favor, pulsa una tecla ASCII..."

While Not value
value=GetKey()
Wend

Print "Has presionado una tecla con un valor ASCII de:" + value
```

WaitKey()

Definición:
Para la ejecución del programa hasta que una tecla sea presionada y devuelve su código ASCII.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando para tu programa hasta que pulses una tecla y devuelve su código ASCII. Mira el ejemplo.

Ejemplo:

```
; Ejemplo WaitKey()

Print "Pulsa una tecla para continuar."

key=WaitKey()

Print "El código ASCII de la tecla presionada es: " + key
Print "Ahora pulsa una tecla para salir."

WaitKey()

End
```

FlushKeys

Definición:
Elimina todas las pulsaciones de teclas actuales en cola.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando 'resetea' o 'vacía' todas las pulsaciones de teclas en cola. Por ejemplo, si tu has pulsado por cualquier razón las teclas 'A', 'D', 'ESPACIO' y 'F', en el momento que tu hagas un Input$ (por ejemplo: variable$=Input$("Nombre: ")), si no has usado este comando, en la respuesta aparecerán las teclas pulsadas, es decir, aparecería en pantalla esto: "Nombre: AD F" y tendrías que borrar esa respuesta que no has escrito, y eso si te das cuenta... No puedo explicarlo más fácilmente que como lo he hecho.
Gracias a Xtreme por ayudarme con la traducción de este comando.

Ejemplo:

```
; borra todas las pulsaciones en cola
FlushKeys
```

MoveMouse x,y

Definición:
Mueve el puntero del ratón a las coordenadas especificadas de la pantalla.

Descripción de los Parámetros:
x = coordenada x de la pantalla
y = coordenada y de la pantalla

Descripción del Comando:
Aunque el ratón no sea visible en la pantalla, la posición de éste puedes saberla y asociarle un gráfico. Sin embargo, hay veces que quieres poner el puntero en una posición específica de la pantalla. Usa este comando para mover le ratón a la localización especificada.

Ejemplo:

```
; Ejemplo MoveMouse

Graphics 640,480
For t = 1 To 2
Print "Mueve el ratón y pulsa el botón izquierdo."
WaitMouse()
Print "El ratón está en la siguiente posición: " + MouseX() + "," + MouseY() + "."
Next
Print "Ahora moveré el ratón..."
Delay 2000
MoveMouse 320,320
Print "El ratón está AHORA en: " + MouseX() + "," + MouseY() + "."
Print "Haz click con el ratón para salir."
WaitMouse()
```

MouseDown (button)

Definición:
Devuelve TRUE si el botón del ratón especificado está siendo pulsado.

Descripción de los Parámetros:
button = 1: botón izquierdo, 2: botón derecho, 3: botón central

Descripción del Comando:
Este comando (al igual que KeyDown y JoyDown) se usa para detectar si un botón del ratón está siendo pulsado. Debes probar con cada botón independientemente (al contrario que KeyDown que devuelve la TECLA que es pulsada). Mira también MouseHit.

Ejemplo:

```
; Ejemplo MouseDown

; Hasta que el usuario pulse ESC, muestra el botón del rat´n presionado
While Not KeyHit(1)
button$="Ninguno"
If MouseDown(1) Then button$="Izquierdo"
If MouseDown(2) Then button$="Derecho"
If MouseDown(3) Then button$="Medio"

Print "¡Botón " + button$ + " presionado!"
Wend
```

MouseHit (button)

Definición:
Devuelve el número de veces que un botón en concreto del ratón es pulsado.

Descripción de los Parámetros:
button = botón (1=izquierdo, 2=derecho, 3=centro)

Descripción del Comando:
Este comando devuelve el número de veces que un botón específico del ratón ha sido pulsado. Mira también KeyHit y JoyHit.

Ejemplo:

```
; Ejemplo MouseHit

; Crea el timer
current=MilliSecs()
Print "Pulsa el botón izquierdo del ratón un montón de veces durante 5 segundos..."

; Espera 5 segundos
While MilliSecs() < current+5000
Wend

; Imprime los resultados
Print "El botón ha sido presionado " + MouseHit(1) + " veces."
```

GetMouse()

Definición:
Comprueba si un botón del ratón está siendo pulsado y devuelve el número del botón o 0 si no se está pulsando ninguno.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Al contrario que otros comandos similares (JoyDown y JoyHit), este comando no necesita saber qué botón estás intentando probar. Éste mira si se está pulsando algún botón y devuelve su número. Si estás pulsando más de uno a la vez sólo devolverá uno. Usa este comando junto con Select/Case para una máxima eficiencia.

Ejemplo:

```
; Ejemplo GetMouse

While Not KeyHit(1)
button=GetMouse()
If button <> 0 Then
Print "Has pulsado el botón del ratón #" + button
End If
Wend
```

WaitMouse()

Definición:
Para la ejecución del programa hasta que pulses un botón del ratón y devuelve su código.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando para la ejecución del programa hasta que pulses un botón del tatón y devuelve su código. Mira el ejemplo.

Ejemplo:

```
; Ejemplo WaitMouse()

Print "Pulsa un botón del ratón para continuar."

button=WaitMouse()

Print "El código del botón pulsado es: " + button
Print "Ahora pulsa un botón para salir."

WaitMouse()

End
```

MouseX()

Definición:
Devuelve la coordenada X del ratón.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando devuelve la coordenada X del ratón en la pantalla. Úsalo junto con el comando DrawImage para crearte tu propio puntero del ratón, o para controlar algo directamente enla pantalla con el ratón. Usa MouseY() para obtener la coordenada Y.

Ejemplo:

```
; Ejemplo LoadAnimImage/MaskImage MouseX()/MouseY()
; With animation timers

; Incluso cuando no hay funciones, debemos hacer variables globales
; Una variable tendrá el handle del gráfico, otra
; el frame actual que estamos mostrando y otra el timer para ajustar
; la velocidad de la animacion
Global gfxSparks, frmSparks, tmrSparks

Graphics 640,480,16
SetBuffer BackBuffer()

; Carga la tira de imagenes
gfxSparks=LoadAnimImage("c:\Program Files\BlitzBasic\samples\Graphics\spark.bmp",32,32,0,3)

; Hacemos que el color rosa sea transparente en esta imagen
```

```
MaskImage gfxSparks,255,0,255

; Bucle hasta que se pulse ESC
While Not KeyHit(1)
Cls

; En la siguiente linea se controla la velocidad de la animacion, Cambia
; el 100 por otros valores mayores o menores para comprobar la diderencia
If MilliSecs() > tmrSparks + 100 Then
tmrSparks=MilliSecs() ; 'reset' the timer
frmSparks=( frmSparks + 1 ) Mod 3 ; incrementa el frame
End If
DrawImage gfxSparks,MouseX(),MouseY(),frmSparks ; dibuja la imagen en las coordenadas del ratón
Flip ; muestra el buffer
Wend
```

MouseY()

Definición:
Devuelve la coordenada Y del ratón.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando devuelve la coordenada Y del ratón en la pantalla. Úsalo junto con el comando DrawImage para crearte tu propio puntero del ratón, o para controlar algo directamente enla pantalla con el ratón. Usa MouseX() para obtener la coordenada X.

Ejemplo:

```
; Ejemplo LoadAnimImage/MaskImage MouseX()/MouseY()
```

MouseZ()

Parámetros:
Ninguno.

Descripción:
MouseZ devuelve la posición actual de la rueda del ratón. Empieza en 0 cuando comienza el programa. El valor MouseZ disminuye cuando la giras hacia ti y aumenta cuando haces lo contrario.

Ejemplo:

```
Graphics 640, 480, 0, 2

SetBuffer BackBuffer ()

Repeat
Text 20, 20, "Posición de la rueda del ratón: " + MouseZ ()
Until KeyHit (1)

End
```

MouseXSpeed()

Definición:
Devuelve los cambios en la posición del ratón desde la última vez que llamaste al comando.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
A menudo querrás saber la diferencia entre donde ESTABA el ratón y donde está AHORA. Puedes usar este comando y MouseYSpeed() para saberlo. En el ejemplo, cuando lo ejecutes, mueve el ratón hasta un lugar y pulsa el botón izquierdo. Esto hará la primera llamada. Entonces, mueve el ratón y pulsa el botón derecho para realizar la segunda y saber la diferencia.

Ejemplo:

```
; Ejemplo MouseXSpeed()/MouseYSpeed()

Graphics 800,600,16
SetBuffer BackBuffer()

; hasta que se pulse un boton del raton...
Repeat
Cls
Rect MouseX(),MouseY(),2,2,1 ; dibujo un pequeño cuadrado donde esta el raton
If MouseHit(1) Then startx=MouseXSpeed():starty=MouseYSpeed()

; Cuando el usuario pulsa el boton derecho del raton, graba la diferencia entre la ultima llamada
; y esta llamada a los comandos mousex/yspeed.
If MouseHit(2) Then endx=MouseXSpeed():endy=MouseYSpeed()
Flip
Until endx

; Muestra los resultados
Text 0,0,"Cambios en las coordenadas del ratón: " + endx + "," + endy
Flip

; Espera que pulses ESC
While Not KeyHit(1)
Wend
```

MouseYSpeed()

Definición:
Devuelve los cambios en la posición del ratón desde la última vez que llamaste al comando.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
A menudo querrás saber la diferencia entre donde ESTABA el ratón y donde está AHORA. Puedes usar este comando y MouseXSpeed() para saberlo. En el ejemplo, cuando lo ejecutes, mueve el ratón hasta un lugar y pulsa el botón izquierdo. Esto hará la primera llamada. Entonces, mueve el ratón y pulsa el botón derecho para realizar la segunda y saber la diferencia.

Ejemplo:
Ver Ejemplo MouseXSpeed

MouseZ()

Parámetros:
Ninguno.

Descripción:
MouseZ devuelve la posición actual de la rueda del ratón. Empieza en 0 cuando comienza el programa. El valor MouseZ disminuye cuando la giras hacia ti y aumenta cuando haces lo contrario.

Ejemplo:

```
Graphics 640, 480, 0, 2

SetBuffer BackBuffer ()

Repeat
Text 20, 20, "Posición de la rueda del ratón: " + MouseZ ()
Until KeyHit (1)

End
```

FlushMouse

Definición:
Elimina todas las pulsaciones de los botones del ratón en cola.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Hay muchas veces que no estás interesado en las docenas de posibles botones del ratón presionados antes de comprobar una combinación de botones en particular. O quizás quieres parar el juego y esperar a que algún botón del ratón sea presionado, pero no quieres que un botón "en cola" interrumpa esta pausa. Usa este comando cuando quieras obtener una pulsación de un botón del ratón.
Gracias a Xtreme por ayudarme con la traducción de este comando.

Ejemplo:

```
; Ejemplo Flushmouse

FlushMouse

Print "Pulsa un botón del ratón para salir!"

WaitMouse()

End
```

oyType ([port])

Definición:
Devuelve el tipo de joystick que hay conectado al ordenador.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve el tipo de joystick que está conectado actualmente al ordenador. Devuelve 0 si no hay ninguno, 1 si es digital y 2 si es analógico.

Ejemplo:

```
; Ejemplo JoyType()

; Mira si el joystick está presente e imrpime el mensaje correspondiente
Select JoyType()
Case 0
Print "Lo siento, no existe ningún joystick actualmente!"
Case 1
Print "Encontrado joystick digital!"
Case 2
Print "Encontrado joystick analógico!"
End Select

; Espera que el usuario pulse ESC para salir
While Not KeyHit(1)
Wend
```

JoyDown (button,[port])

Definición:
Devuelve TRUE si el botón especificado del joystick está siendo pulsado.

Descripción de los Parámetros:
button = número del botón del joystick a comprobar
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando (al igual que KeyDown y MouseDown) se usa para detectar si un botón del joystick está siendo pulsado. Debes comprobar cada botón independientemente con su número correspondiente (al contrario que KeyDown que devuelve el número de la tecla que es pulsada). Mira también JoyHit.

Ejemplo:

```
; Ejemplo JoyDown

; Hasta que el usuario pulse ESC, muestra el botón del joystick presionado
While Not KeyHit(1)
button$="No"
For t = 1 To 5
If JoyDown(t) Then button$=Str(t)
Print "El botón del joystick " + button$ + " ha sido presionado!"
Next
Wend
```

JoyHit (button,[port])

Definición:
Devuelve el número de veces que se pulsa un botón determinado del joystick.

Descripción de los Parámetros:
button = número del botón del joystick a comprobar
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve el número de veces que un botón del joystick es pulsado la última vez que se usó el comando JoyHit(). Mira también KeyHit y MouseHit.

Ejemplo:

```
; Ejemplo JoyHit

; Crea un timer
current=MilliSecs()
Print "Pulsa un montón de veces el botón del joystick nº 1 durante 5 segundos..."

; Espera 5 segundos
While MilliSecs() < current+5000
Wend

; Imrpime los resultados
Print "El botón fue pulsado " + JoyHit(1) + " veces."
```

GetJoy ([port])

Definición:
Comprueba si un botón de un joystick ha sido presionado (devuelve el número del botón o 0 si ninguno ha sido presionado)

Descripción de los Parámetros:
port = puerto de joystick a leer (opcional)

Descripción del Comando:
Al contrario que otros comandos similares (JoyDown y JoyHit), este comando no necesita saber qué botón estás intentando probar. Éste mira si se está pulsando algún botón y devuelve su número. Si estás pulsando más de uno a la vez sólo devolverá uno. Usa este comando junto con Select/Case para una máxima eficiencia.

Ejemplo:

```
; Ejemplo GetJoy

While Not KeyHit(1)
button=GetJoy()
If button <> 0 Then
Print "Has presionado el botón del joystick nº #" + button
```

```
End If
Wend
```

WaitJoy ([port])

Definición:
Para la ejecución del programa hasta que se pulse un botón del joystick.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando para tu programa hasta que pulses un botón del joystick. Como cualquier comando de joystick, DEBES tener un joystick compatible con DirectX y configurado correctamente en Windows. Mira la documentación de tu joystick para más información.

Ejemplo:

```
; Ejemplo WaitJoy()

Print "Pulsa un botón del joystick para continuar."

button=WaitJoy()

Print "El código del botón pulsado es: " + button
Print "Ahora pulsa un botón para salir."

WaitJoy()

End
```

JoyX ([port])

Definición:
Devuelve la coordenada del eje X del joystick.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve el valor del eje X del joystick. El rango está entre -1 y 1 (todo a la izquierda y todo a la derecha). El valor devuelto es un número de coma flotante (real). Mira el ejemplo.

Ejemplo:

```
; Ejemplo JoyX()/JoyY()

While Not KeyHit(1)
Cls
Text 0,0,"Valor Joy X: " + JoyX() + " - Valor Joy Y:" + JoyY()
Wend
```

JoyY ([port])

Definición:
Devuelve la coordenada del eje Y del joystick.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve el valor del eje Y del joystick. El rango está entre -1 y 1 (todo arriba y todo abajo). El valor devuelto es un número de coma flotante (real). Mira el ejemplo.

Ejemplo:
Ver Ejemplo JoyX

JoyZ ([port])

Definición:
Devuelve la coordenada del eje Z del joystick.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve el valor del eje Z del joystick. El rango está entre -1 y 1. El valor devuelto es un número de coma flotante (real). Mira el ejemplo.

Ejemplo:

```
; Ejemplo JoyZ()

While Not KeyHit(1)
Cls
Text 0,0,"Valor Joy Z: " + JoyZ()
Wend
```

JoyXDir ([port])

Definición:
Devuelve la dirección del eje X del joystick.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve la dirección del eje X del joystick. El calor -1 (izquierda) o 1 (derecha). El valor devuelto es un número entero. Mira el ejemplo. Perfecto para joysticks digitales. Como en cualquier comando de joystick, DEBES tener un joystick compatible DirectX y configurado correctamente en Windows para trabajar con él. Mira la documentación de tu joystick para más información.

Ejemplo:

```
; Ejemplo JoyXDir()

While Not KeyHit(1)
Cls
Text 0,0,"Dirección X del Joystick: " + JoyXDir()
Wend
```

JoyYDir ([port])

Definición:
Devuelve la dirección del eje Y del joystick.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve la dirección del eje Y del joystick. El calor -1 (arriba) o 1 (abajo). El valor devuelto es un número entero. Mira el ejemplo. Perfecto para joysticks digitales. Como en cualquier comando de joystick, DEBES tener un joystick compatible DirectX y configurado correctamente en Windows para trabajar con él. Mira la documentación de tu joystick para más información.

Ejemplo:

```
; Ejemplo JoyYDir()

While Not KeyHit(1)
Cls
Text 0,0,"Dirección Y del Joystick: " + JoyYDir()
Wend
```

JoyZDir ([port])

Definición:
Devuelve la dirección del eje Z del joystick.

Descripción de los Parámetros:
port = puerto del joystick a comprobar (opcional)

Descripción del Comando:
Este comando devuelve la dirección del eje Z del joystick. El calor -1 (arriba) o 1 (abajo). El valor devuelto es un número entero. Mira el ejemplo. Perfecto para joysticks digitales. Como en cualquier comando de joystick, DEBES tener un joystick compatible DirectX y configurado correctamente en Windows para trabajar con él. Mira la documentación de tu joystick para más información.

Ejemplo:

```
; Ejemplo JoyZDir()

While Not KeyHit(1)
Cls
Text 0,0,"Dirección Z del Joystick: " + JoyZDir()
Wend
```

FlushJoy

Definición:
Elimina todos los botones del joystick presionados en cola.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Hay muchas veces que no estás interesado en las docenas de posibles botones del joystick presionados antes de comprobar una combinación de botones en particular. O quizás quieres parar el juego y esperar a que algún botón del joystick sea presionado, pero no quieres que un botón "en cola" interrumpa esta pausa. Usa este comando cuando quieras obtener una pulsación de un botón del joystick.
Gracias a Xtreme por ayudarme con la traducción de este comando.

Ejemplo:

```
; Ejemplo FlushJoy

FlushJoy

Print "Pulsa un botón del ratón para salir!"

WaitJoy()

End
```

BANCO

CreateBank (size)

Parameter:
size - tamaño del banco de memoria en bytes

Descripción:
Crea un banco de memoria y devuelve su handle. Los comandos de bancos te permiten realizar operaciones de datos rápidamente en un bloque de memoria. Esto es útil para programar tus propias rutinas de compresión/descompresión, pasando y recibiendo datos de una DLL, p. e.
Los tipos de datos disponibles para usar con un banco son:
Byte - coge un byte. Los valores están entre 0 y 255.
Short - coge 2 bytes. Los valores están entre 0 y 65535.
Int - coge 4 bytes. Los valores están entre -2147483647 y 2147483647.
Float - coge 4 bytes. Los valores están entre -3.4x10??? y 3.4x10??

Ejemplo:

```
; Ejemplo de Comandos de Banco
; --------------------

bnkTest=CreateBank(12)

PokeByte bnkTest,1,Rand(255)
PokeShort bnkTest,2,Rand(65535)
PokeInt bnkTest,4,Rand(-2147483648,2147483647)
PokeFloat bnkTest,8,0.5

Print PeekByte(bnkTest,1)
Print PeekShort(bnkTest,2)
Print PeekInt(bnkTest,4)
Print PeekFloat(bnkTest,8)

FreeBank bnkTest
```

FreeBank bank

Parameter:
bank - handle del banco

Descripción:
Libera un banco de memoria.

Ejemplo:

```
; Ejemplo de Comandos de Banco


BankSize (handle_de_banco)

Definición:

Devuelve el tamaño de un banco.
Descripción de los Parámetros:

handle_de_banco = handle asignado al banco cuando es creado.

Descripción del Comando:

Usa este comando para determinar el tamaño de un banco existente. Mira CreateBank, ResizeBank, y
CopyBank.

Ejemplo:

; Ejemplo BankSize, ResizeBank, CopyBank

; crea un banco
bnkTest=CreateBank(5000)

; lo rellena con enteros aleatorios
For t = 0 To 4999
PokeByte bnkTest,t,Rand(9)
Next

; Redimensiona el banco
ResizeBank bnkTest,10000

; Copia la primera mitad del banco en la segunda mitad
CopyBank bnkTest,0,bnkTest,5000,5000

; Imprime el tamaño final del banco
Print BankSize(bnkTest)
```

ResizeBank bankhandle,new_size

Definición:
Redimensiona un banco.

Descripción de los Parámetros:
bankhandle = handle del banco
new_size = nuevo tamaño del banco en bytes

Descripción del Comando:
Redimensiona un banco de memoria creado previamente. Los datos existentes en el banco no serán modificados, pero pueden ser movidos. Mira también CreateBank, CopyBank y BankSize.

Ejemplo:

```
; Ejemplo BankSize, ResizeBank, CopyBank
```

CopyBank src_bank,src_offset,dest_bank,dest_offset,count

Definición:
Copia los datos de un banco a otro.

Descripción de los Parámetros:
src_bank = handle del banco de memoria de origen
src_offset = localización offset desde donde se comenzará a copiar
dest_bank = handle del banco de memoria de destino
dest_offset = localización offset desde donde se comenzará a escribir
count = cuántos bytes se copiarán

Descripción del Comando:
Copia los datos desde un banco de memoria hasta otro. Si se copia se produce entre bancos iguales, los handles superpondrán sus rangos de memoria.

Ejemplo:

```
; Ejemplo BankSize, ResizeBank, CopyBank
```

```
PeekByte(bank,offset)
```

```
Parámetros:
```

```
bank - handle del banco
```

offset - offset en bytes, en que se comenzará

Descripción:
Lee un byte de un banco de memoria y devuelve el valor. Los valores pueden ser del rango 0 a 255.
Mira también: PeekShort, PeekInt, PeekFloat.

Ejemplo:
Ver Ejemplo de Comandos de Banco

PeekShort(bank,offset)

Parámetros:
bank - handle del banco
offset - offset en bytes, en que se comenzará

Descripción:
Lee un short de un banco de memoria y devuelve el valor. Un short coge 2 bytes de un banco de memoria. Los valores pueden estar en el rango 0 a 65535. Mira también: PeekByte, PeekInt, PeekFloat.

Ejemplo:
; Ejemplo de Comandos de Banco

PeekInt(bank,offset)

Parámetros:
bank - handle del banco
offset - offset en bytes, en que se comenzará

Descripción:
Lee un entero de un banco de memoria y devuelve el valor. Un entero coge 4 bytes del banco de memoria. Los valores pueden estar en el rango -2147483647 a 2147483647. Mira también: PeekByte, PeekShort, PeekFloat.

Ejemplo:
Ver Ejemplo de Comandos de Banco

PeekFloat(bank,offset)

Parámetros:
bank - handle del banco
offset - offset en bytes, en que se comenzará

Descripción:
Lee un flotante de un banco de memoria y devuelve su valor. Un flotante coge 4 bytes de un banco de memoria. Los valores pueden estar en el rango -3.4x10??? a 3.4x10??. Mira también: PeekByte, PeekShort, PeekInt.

Ejemplo:
Ver Ejemplo de Comandos de Banco

PokeByte bank,offset,value

Parámetros:
bank - handle del banco
offset - offset en bytes, desde donde se comenzará a escribir
value - valor que será escrito en el banco

Descripción:
Escribe un byte en un banco de memoria. Los valores pueden estar en el rango 0 a 255. Mira también: PokeShort, PokeInt, PokeFloat.

Ejemplo:
Ver Ejemplo de Comandos de Banco

PokeShort bank,offset,value

Parámetros:
bank - handle del banco
offset - offset en bytes, desde donde se comenzará a escribir
value - valor que será escrito en el banco

Descripción:
Escribe un short en un banco de memoria. Un short coge 2 bytes de un banco de memoria. Los valores pueden estar en el rango 0 a 65535. Mira también: PokeByte, PokeInt, PokeFloat.

Ejemplo:
Ver Ejemplo de Comandos de Banco

PokeInt bank,offset,value

Parámetros:
bank - handle del banco
offset - offset en bytes, desde donde se comenzará a escribir
value - valor que será escrito en el banco

Descripción:
Escribe un entero en un banco de memoria. Un entero coge 4 bytes de un banco de memoria. Los valores están en el rango -2147483647 a 2147483647. Mira también: PokeByte, PokeShort, PokeFloat.

Ejemplo:
Ver Ejemplo de Comandos de Banco

PokeFloat bank,offset,value

Parámetros:
bank - handle del banco
offset - offset en bytes, desde donde se comenzará a escribir
value - valor que será escrito en el banco

Descripción:
Escribe un flotante en un banco de memoria. Un flotante coge 4 bytes. Los valores pueden estar en el rango -3.4x10??? a 3.4x10??.
Mira también: PokeByte, PokeShort, PokeInt.

Ejemplo:
Ver Ejemplo de Comandos de Banco

FICHEROS

OpenFile (filename$)

Definición:
Abre un archivo del disco para leer y escribir sobre él.

Descripción de los Parámetros:
filename$ = ruta completa y nombre de archivo. El valor devuelto es el handle de archivo usado por otros comandos de ficheros.

Descripción del Comando:
Este comando abren el archivo especificado y lo prepara para ser actualizado. El archivo debe existir porque esta función no creará uno nuevo. Usando FilePos y SeekFile se puede determinar y cambiar la posición desde donde se leerá o escribirá en el archivo. Esto permite a un archivo ser leído y actualizado sin tener que hacer una copia del archivo para poder trabajar con él. Esto podría ser útil si has creado una base de datos y quieres encontrar y actualizar unas entradas dentro de él.

Ejemplo:

```
; Ejemplo OpenFile, WriteFile, ReadFile, SeekFile, WriteInt

; Abro/creo un archivo para escribir
fileout = WriteFile("mydata.dat")

; Escribo información en el archivo
WriteInt( fileout, 1 )
WriteInt( fileout, 2 )
WriteInt( fileout, 3 )
WriteInt( fileout, 4 )
WriteInt( fileout, 5 )

; Cierro el archivo
CloseFile( fileout )

DisplayFile( "The file as originally written", mydata.dat" )
; Abro el archivo y cambio el tercer entero

file = OpenFile("mydata.dat")
SeekFile( file, 8 ) ; Se mueve al tercer entero del archivo
WriteInt( file, 9999 ) ; Remplazar el valor original por 9999
CloseFile( file )

DisplayFile( "The file after being midified", "mydata.dat" )
WaitKey()
; **** Definición de las Funciones ****

; Lee el archivo y lo imrpime
Function DisplayFile( Tittle$, Filename$ )
Print tittle$
```

```
filein = ReadFile( Filename$ )
While Not Eof( filein )
Number = ReadInt( filein )
Print Number
Wend
CloseFile( filein )
Print
End Function
```

ReadFile (filename$)

Definición:
Abre un archivo de un disco para ser leído.

Descripción de los Parámetros:
filename$ = ruta y nombre de archivo.

Descripción del Comando:
Este comando abre el archivo especificado y lo prepara para ser leído. Úsalo para leer tu propio archivo de configuración, información del juego guardada, etc. El handle de archivo devuelto lo necesitarás para otras funciones que operen con archivos. Si no pudiste abrir el archivo, por ejemplo, porque no existe, el handle sería igual a 0.

Ejemplo:
Ver Ejemplo OpenFile

WriteFile (filename$)

Definición:
Abre un archivo del disco duro para realizar operaciones de escritura.

Descripción de los Parámetros:
filename$ = ruta completa y nombre de archivo

Descripción del Comando:
Este comando abre el archivo especificado y lo prepara para se modificado. Úsalo para crear tu propio archivo de configuración, guardar datos del juego, etc. también es útil para salvar types en archivos. El handle del archivo devuelto es un valor entero que el sistema operativo utiliza para identificar qué archivo será modificado y debes pasarlo a funciones como WriteInt(). Si el archivo no pudo ser abierto, el handle será igual a 0.

Ejemplo:
Ver Ejemplo OpenFile

CloseFile filehandle

Definición:
Cierra un archivo previamente abierto con un comando de operaciones de archivos.

Descripción de los Parámetros:
filehandle = variable definida con con el comando WriteFile o OpenFile

Descripción del Comando:
Usa este comando para cerrar un archivo previamente abierto. Deberías cerrar siempre un archivo tan pronto como hayas acabado de leerlo o escribir sobre él.

Ejemplo:
Ver Ejemplo OpenFile

FilePos (filehandle)

Definición:
Devuelve la posición actual dentro de un archivo que está siendo leído, escrito o modificado.

Descripción de los Parámetros:
filehandle = la variable devuelta por el comando Readfile WriteFile o OpenFile cuando el archivo fue abierto. El valor devuelto es el offset desde el inicio del archivo. (0 = Inicio del archivo)

Descripción del Comando:
Este comando devuelve la posición actual dentro de un archivo que es procesado. El entero devuelto es el offset en bytes desde el inicio del archivo hasta la posición actual de lectura/escritura. Nota, éste es 0 cuando apunta al primer byte del archivo. Usando FilePos y SeekFile la posición dentro del archivo puede ser determinada y cambiada. Esto permite a un archivo ser leído y actualizado sin tener que hacer una nueva copia del archivo. Esto sería útil si has creado una base de datos y quieres encontrar y actualizar algunas entradas. Es posible crear un archivo de índice que contenga punteros donde comienza cada entrada en en la base de datos.

Ejemplo:
Ver Ejemplo OpenFile

SeekFile (filehandle, offset)

Definición:
Mueve el puntero actual dentro de un archivo abierto a una nueva posición.

Descripción de los Parámetros:
filehandle = handle del archivo obtenido al abrirlo con ReadFile, WriteFile u OpenFile.

Descripción del Comando:
*Este comando permite que la posición en un archivo sea cambiada. Esto permite un acceso aleatorio a los datos en el interior de los archivos y puede usarse con archivos abiertos con ReadFile, WriteFile y OpenFile. Nota: el offset es el número de bytes desde el inicio del archivo, donde el offset del primer byte es 0. Es importante tener en cuenta el tamaño de los elementos de tu archivo. Usando FilePos y SeekFile la posición dentro del archivo puede ser determinada y cambiada. Esto permite a un archivo ser leído y actualizado sin tener que hacer una nueva copia del archivo. Esto sería útil si has creado una base de datos y quieres encontrar y actualizar algunas entradas. Es posible crear un archivo de índice que contenga punteros donde comienza cada entrada en en la base de datos. Para calcular el offset que necesitas saber la longitud de cada elemento es; Offset = Elemento_Deseado * tamaño_del_elemento - tamaño_del_elemento. Por ejemplo un archivo de enteros los cuáles son 4 bytes de largo se calculan así:*
*El 7º entero está en el offset 7 * 4 - 4, es decir, 24*
Nota, mucho cuidado cuando actualices archivos que contienen cadenas de texto, ya que éstas no tienen un tamaño fijo.

Ejemplo:
Ver Ejemplo OpenFile

ReadDir (directory)

Definición:
Abre un directorio de una unidad para leerlo.

Descripción de los Parámetros:
directory = ruta completa y nombre del directorio para abrir

Descripción del Comando:
En operaciones de ficheros, a menudo necesitarás moverte a través de un directorio para recuperar nombres de archivo desconocidos y otras carpetas. Este comando abre el directorio especificado para comenzar estas operaciones. El comando devuelve un handle de archivo el cuál se usa en otros comandos para realizar otras operaciones. Usarás el comando NextFile$ para moverte por el directorio (usa FILETYPE para comprobar si es un archivo o una carpeta). Recuerda, una vez completado, un buen programador diría que debes cerrar el directorio (usando el comando CloseDir). El ejemplo debería ayudarte bastante.

Ejemplo:

```
; Ejemplo ReadDir/NextFile$/CloseDir

; Define el directorio de inicio...
folder$="C:\"

; Abre el directorio y le asigna un handle
myDir=ReadDir(folder$)

Repeat
file$=NextFile$(myDir)
If file$="" Then Exit
; Usa FileType para determinar si es un directorio (valor 2) o un archivo e imprime los resultados
If FileType(folder$+"\"+file$) = 2 Then
Print "Directorio:" + file$
Else
Print "Archivo:" + file$
End If
```

```
Forever

; Cierra el directorio
CloseDir myDir

Print "Hecho!"
```

CloseDir filehandle

Definición:
Cierra un directorio previamente abierto con el comando ReadDir.

Descripción de los Parámetros:
filehandle = filehandle válido asignado con el comando ReadDir

Descripción del Comando:
Una vez has acabado con NextFile$ en el directorio previamente abierto para leer con el comando ReadDir, usa este comando para cerrar el directorio.

Ejemplo:
Ver Ejemplo ReadDir

NextFile$ (filehandle)

Definición:
Recupera el siguiente archivo/directorio de un directorio abierto con el comando ReadDir.

Descripción de los Parámetros:
filehandle = handle del archivo asignado con el comando ReadDir

Descripción del Comando:
Este comando devolveráel SIGUIENTE archivo o carpeta del directorio abierto actualmente (usa ReadDir para abrir un directorio y leerlo). Éste devolverá una cadena de texto que contiene el nombre de la carpeta o archivo más la extensión. Usa FILETYPE para determinar si es un directorio.

Ejemplo:
Ver Ejemplo ReadDir

CurrentDir$()

Definición:
Devuelve una cadena de texto que contiene el directorio seleccionado actualmente.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando devolverá el directorio seleccionado actualmente para realizar operaciones de disco, útil para operaciones avanzadas con ficheros. Usa CHANGEDIR para cambiar el directorio actual. El valor devuelto no tendrá la barra al final ('\').

Ejemplo:

```
; Ejemplo CurrentDir$()

; Imprime el directorio actual hasta que pulses ESC
While Not KeyHit(1)
Print CurrentDir$()
Wend
```

ChangeDir directory/path

Definición:
Cambia al directorio seleccionado para manejar ficheros.

Descripción de los Parámetros:
directory/path = ruta completa al directorio

Descripción del Comando:
Este comando cambiará al directorio seleccionado para manejar ficheros. Usa CURRENTDIR$() para ver cuál es el directorio actual. Usa un directorio/ruta ".." para subir un nivel (ir al directorio "padre").

Ejemplo:

```
; Ejemplo ChangeDir

ChangeDir "c:\windows\system32"
Print "El directorio ha sido cambiado a: " + currentdir$()
```

CreateDir path/name

Definición:
Crea un directorio en una unidad de almacenamiento.

Descripción de los Parámetros:
path/name = ruta completa y nombre del nuevo directorio

Descripción del Comando:
Crea un directorio (carpeta de archivos) en el destino especificado. No uses una barra al final del parámetro path/name. No podrás asegurarte de que el directorio fue creado con este comando, así que necesitarás verificar su existencia por ti mismo (usa el comando FILETYPE).

Ejemplo:

```
; Ejemplo CREATEDIR

fldr$="c:\mydirectorio"
createDir fldr$
Print "¡Directorio creado!"
```

DeleteDir directory/path

Definición:
Borra el directorio especificado.

Descripción de los Parámetros:
directory/path = ruta completa/nombre del directorio

Descripción del Comando:
Borra la carpeta especificada de una unidad. Nota: Este comando sólo trabaja con directorios VACÍOS, no puedes borrar un directorio con otros subdirectorios o archivos en su interior. No hay que poner al final la barra ('\').

Ejemplo:

```
; DeleteDir example

DeleteDir "C:\test"
```

FileType (filename$)

Definición:
Comprueba si el archivo especificado existe o es un directorio.

Descripción de los Parámetros:
filename$ = una variable válida con la ruta/nombre_del_archivo

Descripción del Comando:
Este comando comprueba que el nombre de archivo que has pasado existe o es un directorio. Aquí están los valores que devuelve:
1 = El archivo existe
0 = El archivo no existe
2 = No es un archivo, es un directorio
Úsalo para asegurarte de que un archivo existe antes de hacer algo con él.

Ejemplo:

```
; Los usuarios de Windows NT necesitarán cambiar la localización de calc.exe

filename$="c:\windows\calc.exe"

if fileType(filename$)=1 then Print "¡El archivo existe!"
if fileType(filename$)=0 then Print "¡Archivo no encontrado!"
if fileType(filename$)=2 then Print "¡Es un directorio!"

Print "Pulsa una tecla para salir."

WaitKey()
```

FileSize (filename$)

Definición:
Devuelve el tamaño del archivo especificado en bytes.

Descripción de los Parámetros:
filename$ = una variable válida con la ruta/nombre_del_archivo

Descripción del Comando:
A menudo será útil saber el tamaño de un archivo. El tamaño de un archivo es importante para copiar, leer y otras operaciones con ficheros.

Ejemplo:

```
; Los usuarios de Windows NT necesitarán cambiar la localización de calc.exe

filename$="c:\windows\calc.exe"

Print "El tamaño del archivo en bytes es: " + FileSize(filename$)

Print "Pulsa una tecla para salir."

WaitKey()
```

CopyFile from$, to$

Definición:
Copia un archivo del disco duro en una nueva localización.

Descripción de los Parámetros:
from$ = ruta/nombre de archivo válido del archivo que va a ser copiado
to$ = ruta/nombre de archivo donde se va a copiar

Descripción del Comando:
Usa este comando para copiar un archivo desde una ubicación hasta otra. Quizás programarás tu propio instalador y necesitarás copiar archivos desde el directorio de instalación al directorio donde se instalará el programa. Asegúrate de que los archivos que serán copiados existen antes de ejecutar este comando.

Ejemplo:

```
file$="c:\autoexec.bat"
destination$="a:\autoexec.bat"

Print "Pulsa una tecla para copiar tu Autoexec.bat en un diskette"

WaitKey()

CopyFile file$,destination$
```

DeleteFile path/filename

Definición:
Borra el archivo especificado.

Descripción de los Parámetros:
path/filename = ruta completa\nombre del archivo que se va a borrar

Descripción del Comando:
Elimina el archivo especificado de la unidad. Necesitarás saber si el archivo existe antes de ejecutar este comando y asegurarte de que ha sido borrado DESPUÉS de su ejecución. Usa FILETYPE para saberlo.

Ejemplo:

```
; DELETEFILE example

DeleteFile "C:\test\myfile.bb"
```

ExecFile (filename$)

Definición:
Ejecuta un ejecutable externo desde el interior de un programa Blitz.

Descripción de los Parámetros:
filename$ = una variable válida o ruta/nombre_de_archivo del ejecutable

Descripción del Comando:
Usa este comando para pausar la ejecución de tu programa y ejecutar uno externo.
Nota: Este comando usa ShellExecute para permitirte abrir un archivo (como un .doc o .txt) con su programa asociado por defecto.

Ejemplo:

```
; Ejemplo ExecFile - EJECÚTALO EN VENTANA!
; Los usuarios de Windows NT necesitarán cambiar la localización de calc.exe

filename$="c:\windows\calc.exe"

Print "¡Pulsa una tecla para ejecutar CALC.EXE!"

WaitKey()

ExecFile(filename$)

Print "Pulsa una tecla para salir."

WaitKey()
```

FICHEROS/FLUJOS

Eof (filehandle/stream)

Definición:
Comprueba si se ha llegado al End Of File (Final Del Archivo).

Descripción de los Parámetros:
filehandle/stream = una variable válida establecida con el comando OpenFile, ReadFile o OpenTCPStream (versión 1.52 o superior).

Descripción del Comando:
Comprueba que se ha llegado al Final Del Archivo de un fichero o flujo abierto. Úsalo para determinar si deberías sacar más datos de un archivo/flujo o no. Úsalo para leer texto de un archibo de longitud desconocida y muéstralo. Mira el ejemplo. Eof devuelve 1 si se ha llegado al eof o, en el caso de un flujo TCP, el flujo ha sido cerrado. Eof devuelve -1 si hubo algo mal durante el procesamiento del flujo. Los flujos sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Ejemplo Eof

file$="c:\autoexec.bat"

filein = ReadFile(file$)
```

```
Print "Aquí está tu archivo Autoexec.bat..."

; Se repite hasta que llega al final del archivo
While Not Eof(filein)
Print ReadLine$(filein)
Wend
```

ReadAvail (filehandle/streamhandle)

Definición:
Devuelve el número de bytes que serán leídos con éxito desde un flujo.

Descripción de los Parámetros:
filehandle/streamhandle = handle asignado al archivo o flujo cuando fue abierto/creado

Descripción del Comando:
En el caso de los archivos, refleja su tamaño. En caso de flujos TCP, indica cuánta información ha 'llegado'.
Gracias a Depredador por ayudarme con la traducción de este comando.

Ejemplo:

```
; Ejemplo OpenTCPStream/CloseTCPStream/ReadAvail

Print "Conectando..."
tcp=OpenTCPStream( "www.blitzbasement.com",80 )

If Not tcp Print "Fallo.":WaitKey:End

Print "¡Conectado! Enviando petición..."

WriteLine tcp,"GET http://www.blitzbasement.com HTTP/1.0"
WriteLine tcp,Chr$(10)

If Eof(tcp) Print "Fallo.":WaitKey:End

Print "Petición enviada! Esperando respuesta..."

While Not Eof(tcp)
Print ReadLine$( tcp )
Print "Bytes disponibles:" + ReadAvail(tcp)
Wend

If Eof(tcp)=1 Then Print "Completado!" Else Print "Error!"

CloseTCPStream tcp

WaitKey
```

ReadByte (filehandle/stream)

Definición:
Lee un byte de datos de un archivo o flujo abierto y devuelve su valor como un valor entero entre 0 y 255.

Descripción de los Parámetros:
filehandle/stream = una variable válida establecida con OpenFile, ReadFile o OpenTCPStream (v1.52 o superior)

Descripción del Comando:
Una vez que has abierto un archivo (o flujo) para leerlo, usa este comando para leer un byte. Nota, un byte es un valor entero entre 0 y 255 y que ocupa 8 bits de espacio. Leer más allá del final del archivo no resultará ningún error, pero el valor devuelto será 0.
Los flujos o 'streams' sólo pueden ser usados en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Ejemplo ReadByte y WriteByte

; Inicializa algunas variables
```

```
Byte1% = 10
Byte2% = 100
Byte3% = 255
Byte4% = 256
Byte5% = 257
Byte6% = -1
Byte7% = -2
Byte8% = Asc("A")
; Abre un archivo para modificarlo
fileout = WriteFile("mydata.dat ")

; Escribe informacion en el archivo
WriteByte( fileout, Byte1 )
WriteByte( fileout, Byte2 )
WriteByte( fileout, Byte3 )
WriteByte( fileout, Byte4 )
WriteByte( fileout, Byte5 )
WriteByte( fileout, Byte6 )
WriteByte( fileout, Byte7 )
WriteByte( fileout, Byte8 )

; Cierra el archivo
CloseFile( fileout )

; Abre el archivo para leerlo
filein = ReadFile("mydata.dat")

Read1 = ReadByte( filein )
Read2 = ReadByte( filein )
Read3 = ReadByte( filein )
Read4 = ReadByte( filein )
Read5 = ReadByte( filein )
Read6 = ReadByte( filein )
Read7 = ReadByte( filein )
Read8 = ReadByte( filein )

; Cierra el archivo
CloseFile( filein )

Print "Escrito - Leído"
Write Byte1 + " - " : Print Read1
Write Byte2 + " - " : Print Read2
Write Byte3 + " - " : Print Read3
Write Byte4 + " - " : Print Read4
Write Byte5 + " - " : Print Read5
Write Byte6 + " - " : Print Read6
Write Byte7 + " - " : Print Read7
Write Byte8 + " - " : Print Chr$( Read8 )

WaitKey()
```

ReadShort (filehandle/stream)

Definición:
Lee un valor entero short (16 bits) de un archivo o flujo abierto y lo devuelve como un valor entero.

Descripción de los Parámetros:
filehandle/stream = handle del archivo o flujo
El valor devuelto es un entero entre 0 y 65535.

Descripción del Comando:
Una vez has abierto un archivo o flujo para leerlo, usa este comando para leer un valor entero short (16 bit). Nota: cada valor escrito ocupa 2 bytes de espacio. Leer más allá del final del archivo no resultará un error pero el valor que devolverá será igual a 0. Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Ejemplo ReadShort y WriteShort

; Inicializa algunas variables para el ejemplo
Int1% = 10
Int2% = 365
Int3% = 32767
Int4% = -32768

fileout = WriteFile("mydata.dat")

; Escribe algo en el archivo
WriteShort( fileout, Int1 )
WriteShort( fileout, Int2 )
WriteShort( fileout, Int3 )
WriteShort( fileout, Int4 )

CloseFile( fileout )

filein = ReadFile("mydata.dat")

Read1 = ReadShort( filein )
Read2 = ReadShort( filein )
Read3 = ReadShort( filein )
Read4 = ReadShort( filein )

CloseFile( filein )

Print "Datos Short Integer Data Leídos Del Archivo - mydata.dat "
Print Read1
Print Read2
Print Read3
Print Read4

WaitKey()
```

ReadInt (filehandle/stream)

Definición:
Lee un valor entero de 32bit de un archivo abierto (o flujo) y lo devuelve como un valor entero.

Descripción de los Parámetros:
filehandle/stream = handle del archivo o flujo
El valor devuelto es un entero entre -2147483648 y 2147483647

Descripción del Comando:
Una vez has abierto un archivo o flujo para leerlo, usa este comando para leer un valor entero. Nota: cada valor escrito ocupa 4 bytes de espacio. Leer más allá del final del archivo no resultará un error pero el valor que devolverá será igual a 0.
Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Ejemplo ReadInt y WriteInt

; Inicializa algunas variables para el ejemplo
Int1% = 10
Int2% = 365
Int3% = 2147483647
Int4% = - 2147483648

fileout = WriteFile("mydata.dat")

; Escribe algunos datos en el archivo
WriteInt( fileout, Int1 )
WriteInt( fileout, Int2 )
WriteInt( fileout, Int3 )
```

```
WriteInt( fileout, Int4 )

CloseFile( fileout )

filein = ReadFile("mydata.dat")

Read1 = ReadInt( filein )
Read2 = ReadInt( filein )
Read3 = ReadInt( filein )
Read4 = ReadInt( filein )

CloseFile( filein )

Print "Datos enteros leídos de - mydata.dat "
Print Read1
Print Read2
Print Read3
Print Read4

WaitKey()
```

ReadFloat (filehandle/stream)

Definición:
Lee un valor de coma flotante de un archivo o flujo abierto.

Descripción de los Parámetros:
filehandle/stream = handle del archivo o flujo obtenido con OpenFile, ReadFile o OpenTCPStream (v1.52 o superior)

Descripción del Comando:
Una vez has abierto un archivo del disco (o flujo) para leerlo, usa este comando para leer un valor de coma flotante (real). Nota: cada valor escrito usa 4 bytes de espacio. Leer más allá del final del archivo no resultará un error, pero devolverá un 0.
Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Ejemplo ReadFloat y WriteFloat

; Inicializa algunas variables para el ejemplo
Num1# = 10.5
Num2# = 365.25
Num3# = 32767.123
Num4# = -32768.123

fileout = WriteFile("mydata.dat")

; Escribe algunos datos en el archivo
WriteFloat( fileout, Num1 )
WriteFloat( fileout, Num2 )
WriteFloat( fileout, Num3 )
WriteFloat( fileout, Num4 )

CloseFile( fileout )

filein = ReadFile("mydata.dat")

Read1# = ReadFloat( filein )
Read2# = ReadFloat( filein )
Read3# = ReadFloat( filein )
Read4# = ReadFloat( filein )

CloseFile( filein )

Print "Datos de coma flotante leídos de - mydata.dat "
Print Read1
Print Read2
Print Read3
```

```
Print Read4

WaitKey()
```

ReadString$ (filehandle/stream)

Definición:
Lee una cadena de texto de un archivo o flujo abierto.

Descripción de los Parámetros:

filehandle/stream = handle del archivo o flujo. El valor devuelto es una cadena de texto.

Descripción del Comando:
Una vez has abierto un archivo o flujo para leerlo, usa este comando para leer una cadena de texto.
Cada cadena está almacenada en el archivo como un entero de 4 bytes (32bit) seguido de los caracteres que forman la cadena. El entero contiene el número de caracteres de la cadena, es decir, su longitud. Nota: las cadenas de texto no están limitadas a 255 caracteres como en algunos lenguajes. Leer más allá del final del archivo no resultará ningún error, pero la cadena devuelta tendrá una longitud de 0 caracteres (""). Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Ejemplo ReadString$ y WriteString

; Inicializa algunas variables para el ejemplo
String1$ = "Una cadena corta"
String2$ = "Una cadena más larga que las demás que hay declaradas"
String3$ = "Esta es la string3 "
String4$ = "unida a la string4"

fileout = WriteFile("mydata.dat")

; Escribe algunos datos en el archivo
WriteString( fileout, String1 )
WriteString( fileout, String2 )
WriteString( fileout, String3 + String4)
WriteString( fileout, "No es necesario usar variables" )

CloseFile( fileout )
filein = ReadFile("mydata.dat")

Read1$ = ReadString$( filein )
Read2$ = ReadString$( filein )
Read3$ = ReadString$( filein )
Read4$ = ReadString$( filein )

CloseFile( filein )

Print "Variables String leídas de - mydata.dat "
Print
Print Read1
Print Read2
Print Read3
Print Read4

WaitKey()
```

ReadLine$ (filehandle/stream)

Definición:
Lee una línea de texto de un archivo o flujo abierto.

Descripción de los Parámetros:
filehandle/stream = handle del archivo o flujo

Descripción del Comando:

Una vez has abierto un archivo o flujo para leerlo, usa este comando para leer una línea de texto de un archivo de texto o flujo. Cada línea es devuelta como una cadena de texto. Esta función puede ser usada para leer archivos de texto plano.
Los caracteres son leídos hasta el 'end-of-line' (fin de línea). Leer más allá del final del archivo no resultará un error, pero la cadena devuelta tendrá una longitud de 0 caracteres (""). ReadLine$ devuelve todos los caracteres excepto chr$(13)/chr$(10).
Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Reading and writing a file using ReadLine$ and WriteLine functions

; Initialise some variables for the example
String1$ = "Line 1 is short"
String2$ = "Line 2 is a longer line but they can be much longer"
String3$ = "Line 3 is made up "
String4$ = "of two parts joined together."

; Open a file to write to
fileout = WriteFile("mydata.txt")

; Write the information to the file
WriteLine( fileout, String1 )
WriteLine( fileout, String2 )
WriteLine( fileout, String3 + String4)
WriteLine( fileout, "Just to show you don't have to use variables" )

; Close the file
CloseFile( fileout )

; Open the file to Read
filein = ReadFile("mydata.txt")

Read1$ = ReadLine( filein )
Read2$ = ReadLine$( filein )
Read3$ = ReadLine$( filein )
Read4$ = ReadLine$( filein )

; Close the file once reading is finished
CloseFile( filein )

Print "Lines of text read from file - mydata.txt "
Print
Print Read1
Print Read2
Print Read3
Print Read4

WaitKey()
```

ReadBytes bank,file/stream,offset,count

Definición:

Lee datos de un archivo (o flujo) en un banco de memoria.

Descripción de los Parámetros:

bank = handle del banco
file/stream = handle de un archivo o flujo abierto
offset = offset en bytes para empezar a leer
count = cuántos bytes se leerán a desde el 'offset'

Descripción del Comando:

Puedes leer el contenido de un archivo (o flujo) en un banco de memoria a través de este comando.
Nota: el handle de archivo debe ser abierto con OpenFile o OpenTCPStream y cerrado con CloseFile o CloseTCPStream tras acabar con las operaciones de lectura. Devuelve cuántos bytes se pudieron leer de un flujo.
Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:

```
; Read/WriteBytes Commands Example

; Create a 50 byte memory bank
bnkTest=CreateBank(500)

; Let's fill the bank with crap
For t = 1 To 50
PokeByte bnkTest,t,Rnd(255)
PokeInt bnkTest,t+1,Rnd(10000)
PokeShort bnkTest,t+2,Rnd(10000)
PokeFloat bnkTest,t+3,Rnd(-.999,.999)
Next

; Open a file to write to
fileBank=WriteFile("test.bnk")
; Write the bank to the file
WriteBytes bnkTest,fileBank,0,50
; Close it
CloseFile fileBank

; Free the bank
FreeBank bnkTest

; Make a new one
bnkTest=CreateBank(500)

; Open the file to read from
fileBank=OpenFile("test.bnk")
; Write the bank to the file
ReadBytes bnkTest,fileBank,0,50
; Close it
CloseFile fileBank

; Write back the results!
For t = 1 To 50
Print PeekByte (bnkTest,t)
Print PeekInt (bnkTest,t+1)
Print PeekShort (bnkTest,t+2)
Print PeekFloat (bnkTest,t+3)
Next
```

WriteByte (filehandle/stream, mybyte)

Definición:
Escribe un byte en un archivo o flujo abierto.

Descripción de los Parámetros:
filehandle/stream = una variable válida establecida con OpenFile, WriteFile o OpenTCPStream (v1.52 o superior)
mybyte = puede ser un entero o flotante, pero sólo si está entre 0 y 255 será almacenado correctamente.

Descripción del Comando:
Una vez que has abierto un archivo (o flujo) para modificarlo, usa este comando para escribir un byte. Nota, un byte es un valor entero entre 0 y 255 y que ocupa 8 bits de espacio. Los flujos sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:
Ver Ejemplo ReadByte

WriteShort (filehandle/stream, myinteger)

Definición:
Escribe un valor entero short (16 bits) en un archivo o flujo abierto.
Descripción de los Parámetros:
filehandle/stream = a valid variable set with the OpenFile, WriteFile command, or OpenTCPStream (v1.52+)
myinteger = valor entero

Descripción del Comando:
Una vez has abierto un archivo o flujo para ser modificado, usa este comando para escribir un entero short (16 bit) en él. Nota, cada valor escrito usa 2 bytes. El rango de cada entero está entre 0 y 65535 Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:
Ver Ejemplo ReadShort

WriteInt (filehandle/stream, myinteger)

Definición:
Escribe un valor entero de 32bit en un archivo o flujo abierto para ser modificado.

Descripción de los Parámetros:
filehandle/stream = una variable válida establecida con OpenFile, WriteFile o OpenTCPStream (v1.52 o superior)
myinteger = valor entero

Descripción del Comando:
Una vez has abierto un archivo o flujo para modificarlo, usa este comando para escribir un valor entero. Nota: cada valor escrito ocupa 4 bytes de espacio. El rango del valor está entre -2147483648 y 2147483647 Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:
Ver Ejemplo ReadInt

WriteFloat (filehandle/stream, myFloat)

Definición:
Escribe un valor de coma flotante en un archivo o flujo abierto.

Descripción de los Parámetros:
filehandle/stream = una variable válida establecida con OpenFile, WriteFile o OpenTCPStream (v1.52 o superior)
myFloat = un número de coma flotante

Descripción del Comando:
Una vez has abierto un archivo de disco (o flujo) para modificarlo, usa este comando para escribir un valor de coma flotante. Nota, cada valor escrito usa 4 bytes. Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:
Ver Ejemplo ReadFloat

WriteString$ (filehandle/stream, mystring)

Definición:
Escribe una cadena de texto en un archivo o flujo abierto.

Descripción de los Parámetros:
filehandle/stream = a valid variable set with the OpenFile, WriteFile command, or OpenTCPStream (v1.52+)
mystring = cadena de texto entre comillas

Descripción del Comando:
Una vez has abierto un archivo o flujo para ser modificado, usa este comando para escribir una cadena de texto en él. Cada cadena está almacenada en el archivo como un entero de 4 bytes (32bit) seguido de los caracteres que forman la cadena. El entero contiene el número de caracteres de la cadena, es decir, su longitud. Nota: las cadenas de texto no están limitadas a 255 caracteres como en algunos lenguajes. Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:
Ver Ejemplo ReadString$

WriteLine$ (filehandle/stream, string$)

Definición:
Escribe una línea de texto en un archivo o flujo abierto.

Descripción de los Parámetros:
una variable válida establecida con OpenFile, WriteFile o OpenTCPStream (v1.52 o superior)
string$ = cadena de texto válida

Descripción del Comando:
Una vez has abierto un archivo para ser modificado, usa este comando para escribir una línea de texto en él. Cada línea es automáticamente terminada con una marca "end-of-line" (fin de línea). Esta función puede usarse para crear archivos de texto planos. Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:
Ver Ejemplo ReadLine$

WriteBytes bank,filehandle/stream,offset,count

Definición:
Escribe datos de un banco de memoria en un archivo (o flujo) abierto.

Descripción de los Parámetros:
bank = handle del banco
filehandle/stream = handle del fljujo o archivo
offset = offset en bytes para escribir el valor
count = nº de bytes a escribir

Descripción del Comando:
Puedes escribir el contenido de un banco de memoria en un archivo del disco (o flujo) usando este comando.
Devuelve cuántos bytes han sido escritos sin problemas. Los flujos o 'streams' sólo pueden usarse en Blitz Basic v1.52 o superior.

Ejemplo:
Ver Read/WriteBytes

NETWORK

CountHostIPs(host_name$)

Parámetros:
host_name$ - nombre del host

Descripción:
Busca hosts con el nombre especificado y devuelve el número de hosts encontrados.

HostIP(host_index)

Parámetros:
host_index - número de índice del host

Descripción:
Devuelve una dirección IP del host especificado. El parámetro host_index debe estar entre 1...CountHostIPs().

DottedIP$(IP)

Parámetros:
IP - dirección IP (entero)

Descripción:
Convierte una dirección IP en notación de internet.

CopyStream src_stream,dest_stream[,buffer_size]

Parámetros:
src_stream - flujo de origen
dest_stream - flujo de destino
buffer_size (opcional) - tamaño del buffer del flujo

Descripción:
Copia el flujo de origene n el flujo de destino.

TCP (NETWORK)

OpenTCPStream (ip$,port)

Definición:
Abre un cliente TCP al servidor especificado.

Descripción de los Parámetros:
i*p$=dirección del flujo*
port=puerto TCP/IP

Descripción del Comando:
Usa este comando para abrir un flujo TCP/IP al puerto y servidor designado. Si el comando se completa correctamente, devolverá el handle del flujo. De lo contrario devolverá 0. Puedes usar este comando para multitud de opciones en internet. Obviamente para contactar con un host TCP/IP fuera de tu propia red, deberás estar conectado a Internet. La dirección IP puede ser de la forma 1.2.3.4 o "www.dominio.com".

Ejemplo:

```
; Ejemplo OpenTCPStream/CloseTCPStream/ReadAvail

Print "Conectando..."
tcp=OpenTCPStream( "www.blitzbasement.com",80 )

If Not tcp Print "Fallo.":WaitKey:End

Print "¡Conectado! Enviando petición..."

WriteLine tcp,"GET http://www.blitzbasement.com HTTP/1.0"
WriteLine tcp,Chr$(10)

If Eof(tcp) Print "Fallo.":WaitKey:End

Print "Petición enviada! Esperando respuesta..."

While Not Eof(tcp)
Print ReadLine$( tcp )
Print "Bytes disponibles:" + ReadAvail(tcp)
Wend

If Eof(tcp)=1 Then Print "Completado!" Else Print "Error!"

CloseTCPStream tcp

WaitKey
```

CloseTCPStream streamhandle

Definición:
Cierra el flujo TCP especificado.

Descripción de los Parámetros:
streamhandle = handle asignado cuando el flujo fue creado.

Descripción del Comando:
Una vez has completado el uso de tu flujo TCP/IP, cierra la conexión que abriste con OpenTCPStream con este comando.

Ejemplo:
Ver Ejemplo OpenTCPStream

CreateTCPServer (port)

Definición:
Crea un servidor TCP en el puerto especificado.

Descripción de los Parámetros:
port = puerto a usar cuando se cree el servidor

Descripción del Comando:
Crea un servidor TCP/IP en el puerto especificado. Úsalo para comunicaciones entre otros clientes y el ordenador local. Mira OpenTCPStream, CloseTCPServer, y CloseTCPStream para más información. Devuelve el handle del servidor TCP/IP si la operación fue completada o un 0 en caso contrario.

Ejemplo:

```
; CreateTCPServer, CloseTCPServer, AcceptTCPStream Example
; This code is in two parts, and needs to be run seperately on the same machine

; --- Start first code set ---
; Create a server and listen for push

svrGame=CreateTCPServer(8080)

If svrGame<>0 Then
Print "Server started successfully."
Else
Print "Server failed to start."
End
End If

While Not KeyHit(1)
strStream=AcceptTCPStream(svrGame)
If strStream Then
Print ReadString$(strStream)
Delay 2000
End
Else
Print "No word from Apollo X yet ..."
Delay 1000
End If
Wend

End

; --- End first code set ---


; --- Start second code set ---
; Copy this code to another instance of Blitz Basic
; Run the above code first, then run this ... they will 'talk'

; Create a Client and push data

strmGame=OpenTCPStream("127.0.0.1",8080)

If strmGame<>0 Then
Print "Client Connected successfully."
Else
Print "Server failed to connect."
WaitKey
End
End If

; write stream to server
WriteString strmGame,"Mission Control, this is Apollo X ..."
Print "Completed sending message to Mission control..."

; --- End second code set ---
```

AcceptTCPStream (handle_del_servidor)

Definición:
Acepta una entrada de flujo TCP/IP.

Descripción de los Parámetros:
handle_del_servidor = el handle del servidor es asignado cuando el servidor es creado.

Descripción del Comando:
Acepta una entrada de flujo TCP/IP y devuelve 1 si esta disponible, 0 si no lo está. Mira CreateTCPServer y CloseTCPServer.

Ejemplo:
Ver CreateTCPServer

CloseTCPServer serverhandle

Definición:
Cierra un servidor TCP.

Descripción de los Parámetros:
serverhandle = handle asignado cuando el servidor es creado.

Descripción del Comando:
Cierra un servidor TCP/IP previamente creado con el comando CreateTCPServer.

Ejemplo:
Ver CreateTCPServer

TCPStreamIP(tcp_stream)

Parámetros:
tcp_stream - handle del flujo TCP

Descripción:
Devuelve la dirección IP del flujo TCP especificado. La dirección devuelta es siempre la del ordenador cliente.

TCPStreamPort(tcp_stream)

Parámetros:
tcp_stream - handle del flujo TCP

Descripción:
Devuelve el número de puerto del flujo TCP especificado. El número de puerto devuelto es siempre el del ordenador cliente.

TCPTimeouts read_millis,accept_millis

Parámetros:
read_millis - valor en milésimas de segundo
accept_millis - valor en milésimas de segundo

Descripción:
read_millis te permite controlar cuántos datos se pueden leer en un flujo TCP antes de causar un error. Por defecto, esta establecido a 10,000 (10 segundos). Esto significa que si tarda en leer los datos más de 10 segundos, ocurrirá un error y el flujo no podrá volver a usarse.
accept_millis te permite controlar cuánto esperará el comando AcceptTCPStream() para una nueva conexión. Por defecto, este valor es 0, así AcceptTCPStream() volverá automáticamente si no hay una nueva conexión disponible.

UDP (NETWORK)

CreateUDPStream([port])

Parámetros:
port (opcional) - número de puerto

Descripción:
Crea un flujo UDP en el puerto especificado y devuelve un handle de flujo UDP. Si no se especifica ningún puerto, se utilizará uno que esté libre y podrás entonces usar UDPStreamPort() para encontrar el puerto que se está utilizando.

CloseUDPStream udp_stream

Parámetros:
udp_stream - handle del flujo UDP

Descripción:
Cierra un flujo UDP.

SendUDPMsg udp_stream,dest_ip[,dest_port]

Parámetros:
udp_stream - handle de un flujo UDP
dest_ip - dirección IP de destino
dest_port (opcional) - número de puerto de destino

Descripción:
Transmite todos los datos escritos en el flujo UDP a la dirección IP y puerto especificados. La información está escrita con los comandos estándar de flujos. Si no se especifica el puerto, se utilizará el que se usó para crear el Flujo UDP.

RecvUDPMsg(udp_stream)

Parámetros:
udp_stream - handle del flujo UDP

Descripción:
Recibe un mensaje UDP del flujo UDP especificado. Comandos de flujos estándar pueden usarse para examinar el mensaje.
El valor devuelto es un entero con la dirección IP de origen del mensaje, o 0 si el mensaje no está disponible. Puedes usar UDPMsgPort() para encontrar el número de puerto de origen del mensaje.

UDPStreamIP(udp_stream)

Parámetros:
udp_stream - handle del flujo UDP

Descripción:
Devuelve la dirección IP del flujo UDP especificado. Actualmente, siempre devuelve 0.

UDPStreamPort(udp_stream)

Parámetros:
udp_stream - handle del flujo UDP

Descripción:
Devuelve el número de puerto del flujo UDP especificado. Esto puede ser útil si has creado el flujo UDP sin especificar el puerto.

UDPMsgIP(udp_stream)

Parámetros:
udp_stream - handle del flujo UDP

Descripción:
Devuelve la dirección IP del remitente del último mensaje UDP recibido. Este valor también es devuelto por RecvUDPMsg().

UDPMsgPort(udp_stream)

Parámetros:
udp_stream - handle del flujo UDP

Descripción:
Devuelve el puerto del remitente del último mensaje UDP recibido.

UDPTimeouts recv_millis

Parámetros:
recv_millis - valor en milésimas de segundo

Descripción:
recv_millis te permite controlar cuanto puede esperar la función RecvUDPMsg() para recibir un nuevo mensaje. Por defecto, está establecido en 0, lo que significa que RecvUDPMsg() retornará inmediatamente si no se ha recibido ningún mensaje.

DIRECTPLAY

StartNetGame()

Definición:
Comienza un juego en red.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Muestra un cuadro de diálogo de Windows con opción de unirse o empezar un nuevo juego multijugador en red, vía módem, conexión serial o TCP/IP (Internet). Nota: este comando debe usarse antes que cualquier otro comando de red, de otra manera los otros comandos fallarán. Un valor de retorno igual a 0 indica que falló, 1 significa que se unió al juego y 2 que se creó un nuevo juego (el host sería el ordenador local).

Ejemplo:

```
newGame = StartNetGame()
; Check the status of the new game.
If newGame = 0 Then
print "Could not start or join net game."
ElseIf newGame = 1
print "Successfully joined the network game."
ElseIf newGame = 2
print "A new network game was started."
EndIf
```

HostNetGame (gamename$)

Definición:
Crea un juego en red como HOST.

Descripción de los Parámetros:
gamename$ = valor string (cadena) en el que se asigna el nombre del juego

Descripción del Comando:
Este se usa para desviar el 'estándard' en juegos en red (normalmente usando StartNetGame) y empezar el juego como host directamente. Si devuelve el valor de 2, significa que el Host empezó el juego correctamente.

Ejemplo:

```
; HostNetGame example

joinResults=HostNetGame("ShaneGame")

Select joinResults
Case 2
Print "Successfully started host game!"
Default
Print "Game was unable to start!"
End Select
waitkey()
```

JoinNetGame (gamename$,serverIP$)

Definición:
Se une a un juego en red en curso.

Descripción de los Parámetros:
gamename$ = nombre del juego a unirse
serverIP$ = dirección IP del ordenador host del juego

Descripción del Comando:
Usa este comando para unirte a un juego en red, evitando el cuadro de diálogo que aparece al usar el comando StartNetGame. Devuelve 0 si el comando falla o 1 si se unió al juego correctamente.

Ejemplo:

```
; JoinNetGame example
; Note; run the HostNetGame example code on the other computer
; you wish to join with

gamename$="ShaneGame"
; Change this to match the other computer's IP!
serverIP$="0.0.0.0"

; Make the join attempt
joinResults=JoinNetGame(gamename$,serverIP$)

Select joinResults
Case 1
Print "Joined the game successfully!"
Default
Print "Joining the game was unsuccessful."
End Select
WaitKey()
```

StopNetGame

Definición:
Para un juego en red en curso.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Usa este comando para acabar un juego en red actualmente en curso (empezado con el comando StartNetGame()). Si es posible, el host será transferido a otro ordenador conectado al juego.

Ejemplo:

```
; Ejemplo StopNetGame()

newGame = StartNetGame()
; Comprueba el estado del nuevo juego.
If newGame = 0 Then
print "No puedes unirte o crear un juego en red."
ElseIf newGame = 1
print "Unido con éxito al juego en red."
ElseIf newGame = 2
print "Se creó un juego en red."
EndIf
waitkey()
StopNetGame()
print "Se paró el juego en red."
```

CreateNetPlayer (name$)

Definición:
Crea un nuevo jugador local en un juego en red.

Descripción de los Parámetros:
name$ = alguna cadena de texto válida para el nombre del jugador

Descripción del Comando:
Crea un nuevo jugador local. Esto también causa un mensaje especial que será enviado a todos los ordenadores remotos (mira NetMsgType). Éste devuelve un número de jugador (número entero) que será usado en los mensajes enviados/recibidos. Recuerda que debes crear al menos un jugador antes de poder enviar y recibir mensajes.

Ejemplo:

```
; CreateNetPlayer example

newGame = StartNetGame()
; Check the status of the new game.
If newGame = 0 Then
Print "Could not start or join net game."
End

ElseIf newGame = 1
Print "Successfully joined the network game"
ElseIf newGame = 2
Print "A new network game was started!"
EndIf

; Create a random player name
name$="Player" + Rand(100)

; Get a unique player id number for the player
; and create the player
playerID=CreateNetPlayer(name$)

If playerID = 0 Then
Print "Player could not be created!"
Else
Print "Player " + name$ + " was created and given ID#" + playerID
End If
WaitKey()
```

DeleteNetPlayer playerID

Definición:
Elimina un jugador local de un juego en red.

Descripción de los Parámetros:
playerID = valor asignado cuando el jugador es creado con CreateNetPlayer

Descripción del Comando:
Usando el playerID generado con la función CreateNetPlayer, este comando eliminará al jugador especificado de la partida en red. Esto también causa un mensaje especial que será enviado a todos los ordenadores remotos (mira NetMsgType).

Ejemplo:

```
; DeleteNetPlayer example

newGame = StartNetGame()
; Check the status of the new game. If newGame = 0 Then
Print "Could not start or join net game."
End

ElseIf newGame = 1
Print "Successfully joined the network game"
ElseIf newGame = 2
```

```
Print "A new network game was started!"
EndIf

; Create a random player name
name$="Player" + Rand(100)

; Get a unique player id number for the player
; and create the player
playerID=CreateNetPlayer(name$)

If playerID = 0 Then
Print "Player could not be created!"
Else
Print "Player " + name$ + " was created and given ID#" + playerID
WaitKey()
; delete the player!
DeleteNetPlayer playerID
Print "The local player was deleted!"
End If
waitkey()
```

NetPlayerName$ (playerID)

Definición:

Devuelve el nombre actual de un jugador de un juego en red.

Descripción de los Parámetros:

playerID = número ID del jugador (se obtiene con el comando NetMsgFrom())

Descripción del Comando:

Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Usa este comando junto con NetMsgFrom() (para obtener el ID del jugador) para saber el nombre actual del jugador. Este comando devuelve una cadena de texto. Usarás NetMsgType(), NetMsgFrom() y NetMsgTo() para obtener otra información importante del mensaje y actuar sobre esta. El ejemplo requiere ser ejecutado también en un ordenador remoto mientras el ordenador local usa el comando SendNetMsg.

Ejemplo:

```
; NetPlayerName$() example
; --------------------
; Run this program on the REMOTE computer to 'watch'
; the activity of the SendNetMsg example. Run that
; example on local machine.
;
; This program will tell you when a player involved in
; the game hits a wall ...

; We'll use this instead of JoinHostGame - make it easier
StartNetGame()

; Create a player - a player must be created to
; receive mesages!
playerID=CreateNetPlayer("Shane")

; Loop and get status
While Not KeyHit(1)

; Check to see if we've received a message
If RecvNetMsg() Then

; if we did, let's figure out what type it is
; we know it will be a user message, though
msgType=NetMsgType()

; 1-99 means a user message
If msgType>0 And msgType<100 Then
```

```
; Let's see who the message was from
msgFrom=NetMsgFrom()

; Let's get the message!
msgData$=NetMsgData$()

; Print the message
Print msgData$
Print "(Message was from:"+ NetPlayerName$(NetMsgFrom()) + ")"
End If
End If
Wend
```

NetPlayerLocal (playerID)

Definición:
Determina si el jugador esta en el ordenador local.

Descripción de los Parámetros:
playerID = ID del jugador (obtenido con el comando NetMsgFrom())

Descripción del Comando:
Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Usa este comando junto con NetMsgFrom() (para obtener el ID del jugador) para comprobar si el jugador en cuestión está en el ordenador local (o de lo contrario en uno remoto). Usarás NetMsgType(), NetMsgFrom() y NetMsgTo() para obtener otra información importante del mensaje y actuar sobre esta. El ejemplo requiere ser ejecutado también en un ordenador remoto mientras el ordenador local usa el comando SendNetMsg.

Ejemplo:

```
; NetPlayerLocal example
; --------------------
; Run this program on the REMOTE computer to 'watch'
; the activity of the SendNetMsg example. Run that
; example on local machine.
;
; This program will tell you when a player involved in
; the game hits a wall ...

; We'll use this instead of JoinHostGame - make it easier
StartNetGame()

; Create a player - a player must be created to
; receive mesages!
playerID=CreateNetPlayer("Shane")

; Loop and get status
While Not KeyHit(1)

; Check to see if we've received a message
If RecvNetMsg() Then

; if we did, let's figure out what type it is
; we know it will be a user message, though
msgType=NetMsgType()

; 1-99 means a user message
If msgType>0 And msgType<100 Then

; Let's see who the message was from
msgFrom=NetMsgFrom()

; Let's get the message!
msgData$=NetMsgData$()

; Print the message
Print msgData$
```

```
if NetPlayerLocal(NetMsgFrom()) then
print "(This was sent from a local player)"
end if
End If
End If
Wend
```

RecvNetMsg()

Definición:
Valores booleanos indican si un mensaje network ha sido recibido.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Devuelve un valor TRUE si el mensaje ue recibido, o FALSE en caso contrarioThis returns a TRUE value if a message was received, FALSE if none has been received. This will typically go inside a function that is constantly being checked for message and decode and handle them. You will use NetMsgType, NetMsgFrom, NetMsgTo, and NetMsgData$ to get the important information from the message and act on it. The example requires that you run it on a remote machine while the local computer runs the example in the SendNetMsg command.

Ejemplo:

```
; RecvNetMsg() example
; --------------------
; Run this program on the REMOTE computer to 'watch'
; the activity of the SendNetMsg example. Run that
; example on local machine.
;
; This program will tell you when a player involved in
; the game hits a wall ...

; We'll use this instead of JoinHostGame - make it easier
StartNetGame()

; Create a player - a player must be created to
; receive mesages!
playerID=CreateNetPlayer("Shane")

; Loop and get status
While Not KeyHit(1)

; Check to see if we've received a message
If RecvNetMsg() Then

; if we did, let's figure out what type it is
; we know it will be a user message, though
msgType=NetMsgType()

; 1-99 means a user message
If msgType>0 And msgType<100 Then

; Let's see who the message was from
msgFrom=NetMsgFrom()

; Let's get the message!
msgData$=NetMsgData$()

; Print the message
Print msgData$
End If
End If
Wend
```

NetMsgType()

Definición:
Devuelve el tipo de mensaje recibido durante un juego en red.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Debes haber recibido el mensaje otra vez, determinado por el comando RecvNetMsg(). El valor devuelto por este comando denota el mensage; 1-99 indica que es un mensaje de usuario, 100 significa que un nuevo jugador se ha unido al juego, 101 significa que un jugador ha sido eliminado del juego (NetMsgFrom() devuelve el jugador eliminado), 102 que el host original se marchó del juego y ESTA máquina es el nuevo host. Si recives un 200, significa que un error fatal ha ocurrido y necesitas salir del juego. Usarás NetMsgFrom, NetMsgTo, y NetMsgData$ para obtener información importante del mensaje y actuar sobre él. El ejemplo requiere ser ejecutado también en un ordenador remoto mientras el ordenador local usa el comando SendNetMsg.

Ejemplo:

```
; NetMsgType() example
; --------------------
; Run this program on the REMOTE computer to 'watch'
; the activity of the SendNetMsg example. Run that
; example on local machine.
;
; This program will tell you when a player involved in
; the game hits a wall ...

; We'll use this instead of JoinHostGame - make it easier
StartNetGame()

; Create a player - a player must be created to
; receive mesages!
playerID=CreateNetPlayer("Shane")

; Loop and get status
While Not KeyHit(1)

; Check to see if we've received a message
If RecvNetMsg() Then

; if we did, let's figure out what type it is
; we know it will be a user message, though
msgType=NetMsgType()

; 1-99 means a user message
If msgType>0 And msgType<100 Then

; Let's see who the message was from
msgFrom=NetMsgFrom()

; Let's get the message!
msgData$=NetMsgData$()

; Print the message
Print msgData$
End If
End If
Wend
```

NetMsgFrom()

Definición:
Devuelve el ID del emisor de un mensaje network.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:

Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Debes haber recibido el mensaje otra vez, determinado por el comando RecvNetMsg() - y probablemente el tipo de mensaje sea determinado con NetMsgType(). El valor devuelto por este comando es el número ID del emisor del mensaje que le fue asignado cuando fue creado con el comando CreateNetPlayer. Úsalo para realizar acciones sobre el jugador desde el ordenador local. Usarás NetMsgType(), NetMsgTo() y NetMsgData$() para obtener otra información importante del mensaje y actuar sobre él. El ejemplo requiere ser ejecutado también en un ordenador remoto mientras el ordenador local usa el comando SendNetMsg.

Ejemplo:

```
; NetMsgFrom() example
; --------------------
; Run this program on the REMOTE computer to 'watch'
; the activity of the SendNetMsg example. Run that
; example on local machine.
;
; This program will tell you when a player involved in
; the game hits a wall ...

; We'll use this instead of JoinHostGame - make it easier
StartNetGame()

; Create a player - a player must be created to
; receive mesages!
playerID=CreateNetPlayer("Shane")

; Loop and get status
While Not KeyHit(1)

; Check to see if we've received a message
If RecvNetMsg() Then

; if we did, let's figure out what type it is
; we know it will be a user message, though
msgType=NetMsgType()

; 1-99 means a user message
If msgType>0 And msgType<100 Then

; Let's see who the message was from
msgFrom=NetMsgFrom()

; Let's get the message!
msgData$=NetMsgData$()

; Print the message
Print msgData$
End If
End If
Wend
```

NetMsgTo()

Definición:

Devuelve el ID del recipiente de un mensaje network.

Descripción de los Parámetros:

Ninguno.

Descripción del Comando:

Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Debes haber recibido el mensaje otra vez, determinado por el comando RecvNetMsg() - y probablemente el tipo de mensaje sea determinado con NetMsgType(). El valor devuelto por este comando es el ID del recipiente del mensaje que le fue asignado cuando fue creado con CreateNetPlayer. Usarás NetMsgType(), NetMsgFrom() y NetMsgData$() para obtener otra información importante sobre el mensaje y actuar sobre él. El ejemplo requiere ser ejecutado también en un ordenador remoto mientras el ordenador local usa el comando SendNetMsg.

Ejemplo:

```
; NetMsgTo() example
; -------------------
; Run this program on the REMOTE computer to 'watch'
; the activity of the SendNetMsg example. Run that
; example on local machine.
;
; This program will tell you when a player involved in
; the game hits a wall ...

; We'll use this instead of JoinHostGame - make it easier
StartNetGame()

; Create a player - a player must be created to
; receive mesages!
playerID=CreateNetPlayer("Shane")

; Loop and get status
While Not KeyHit(1)

; Check to see if we've received a message
If RecvNetMsg() Then

; if we did, let's figure out what type it is
; we know it will be a user message, though
msgType=NetMsgType()

; 1-99 means a user message
If msgType>0 And msgType<100 Then

; Let's see who the message was from
msgFrom=NetMsgFrom()

; Let's get the message!
msgData$=NetMsgData$()

; Print the message
Print msgData$
Print "(Message was to:"+ NetMsgTo() + ")"
End If
End If
Wend
```

NetMsgData$()

Definición:
Devuelve el mensaje actual de un mensaje network.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Debes haber recibido el mensaje otra vez, determinado por el comando RecvNetMsg() - y probablemente el tipo de mensaje sea determinado con NetMsgType(). La cadena de texto devuelta por este comando es el actual mensaje de texto que fue enviado. Usarás NetMsgType(), NetMsgFrom() y NetMsgTo() para obtener otra información importante del mensaje y actuar sobre esta. El ejemplo requiere ser ejecutado también en un ordenador remoto mientras el ordenador local usa el comando SendNetMsg.

Ejemplo:

```
; NetMsgData$() example
; --------------------
; Run this program on the REMOTE computer to 'watch'
; the activity of the SendNetMsg example. Run that
; example on local machine.
;
```

```
; This program will tell you when a player involved in
; the game hits a wall ...

; We'll use this instead of JoinHostGame - make it easier
StartNetGame()

; Create a player - a player must be created to
; receive mesages!
playerID=CreateNetPlayer("Shane")

; Loop and get status
While Not KeyHit(1)

; Check to see if we've received a message
If RecvNetMsg() Then

; if we did, let's figure out what type it is
; we know it will be a user message, though
msgType=NetMsgType()

; 1-99 means a user message
If msgType>0 And msgType<100 Then

; Let's see who the message was from
msgFrom=NetMsgFrom()

; Let's get the message!
msgData$=NetMsgData$()

; Print the message
Print msgData$
Print "(Message was to:"+ NetMsgTo() + ")"
End If
End If
Wend
```

SendNetMsg type,data$,from,to,reliable

Definición:

Sends a message during a network game.

Descripción de los Parámetros:

type = valor 1-99
data$ = cadena que contiene el mensaje a enviar
from = ID del jugador remitente
to = ID del jugador de destino (0=broadcast)
reliable = flag para realizar un envío seguro

Descripción del Comando:

Esto SÓLO funciona cuando te has unido a un juego en red vía StartNetGame o JoinNetGame y has creado un jugador mediante la función CreateNetPlayer (debes crear un jugador primero). Éste es probablemente el comando de red más complicado. Sirve àra enviar un mensaje a uno o todos los jugadores del juego en red. Los otros jugadores usarán RecvNetMsg() para interceptar tu mensaje. El parámetro TYPE es un número entre 1 y 99. Estos valores son conocidos como 'mensajes del usuario'. El parámetro Data$ es una cadena de texto que contiene el mensaje a enviar. Es útil saber como introducir el mayor número de datos posible en un solo mensaje para mejorar la velocidad. Por ejemplo, podrías enviar X, Y, y FRAME en una sola cadena como "200,100,4". FROM es el ID del jugador que envía el mensaje. Es el valor devuelto por el comando CreateNetPlayer(). TO es el ID del jugador a quien quieres enviar el mensaje. El valor 0 mandará el mensaje a todos los jugadores. El flag RELIABLE asegurará que el mensaje no se pierda. Sin embargo, es al menos 3 veces más lento que un mensaje normal. El ejemplo requiere ser ejecutado en el ordenadro local y en uno remotole ejecutar el del comando RecvNetMsg().

Ejemplo:

```
; SendNetMsg example
; ------------------
; Run this example on the local computer
; run the example for RecvNetMsg() on a remote computer

; Graphics mode with double buffering
```

```
Graphics 640,480,16
SetBuffer BackBuffer()

; Create a network game with NO requester
joinStatus=HostNetGame("ShaneGame")

; A type to hold all the player's information
Type multi
Field x
Field y
Field id
Field name$
Field xspeed
Field boxColor
End Type

; make sure the game started ok...
If joinStatus=2 Then
Print "Hosted game started... "
Else
Print "Hosted game could not be started!"
End
End If

; Create 5 local players using TYPEs
For t = 1 To 5
; New type instance
player.multi = New Multi
; Assign the ID field with the created player ID and name him
player\ID=CreateNetPlayer("Player" + t)

; if the player was created ok ... assign some random parameters
If player\ID <> 0 Then
player\name$="Player" + t
player\x = Rand(640)
player\y = Rand(480)
player\boxColor = Rand(255)
player\xspeed = Rand(1,5)
; Print some text results
Print "Player " + t + " has joined the game with ID=" + player\ID
Else
Print "The player couldn't join! Aborting!"
End If
Next

; We've got them all! Wait for a key
Print "All local players are joined! Press a key ..."
WaitKey()

; Loop this routine
While Not KeyHit(1)
Cls
; for each of the players, update their locations on the screen
For player = Each multi
Color player\boxColor,player\boxColor,player\boxColor
Rect player\x,player\y,10,10,1
Text player\x-10,player\y-15,player\name$
player\x = player\x + player\xspeed
If player\x > 640 Or player\x < 0 Then
player\xspeed=-player\xspeed
message$="Player ID #" + player\ID + " hit a wall!"
; Send a broadcast message if a player rebounds off the wall
; this message will show up on the remote machine
SendNetMsg Rand(1,99),message$,player\id,0
End If
Next
Flip
Wend
End
```

DIRECTSHOW

OpenMovie(file$)

Parámetros:
file$ - nombre de archivo de la película

Descripción:
Abre un vídeo y devuelve su handle.

CloseMovie movie

Parámetros:
movie - handle a la película

Descripción:
Cierra una película.

DrawMovie movie,x,y,[,width,height]

Parámetros:
movie - handle del vídeo
x - posición x de la esquina superior izquierda del vídeo
y - posición y de la esquina superior izquierda del vídeo
width (opcional) - ancho del vídeo. Por defecto el ancho original del vídeo.
height (opcional) - altura del vídeo. Por defecto la altura original del vídeo.

Descripción:
Dibuja un vídeo en el buffer actual. Devuelve True si el vídeo todavía se está reproduciendo. El soporte de vídeos depende de DirectShow, así que deberás asegurarte de que tienes los drivers correctamente instalados.
Los vídeos se moverán normalmente más rápidos con su tamaño original.

MovieWidth(movie)

Parámetros:
movie - handle del vídeo

Descripción:
Devuelve el ancho de un vídeo.

MovieHeight(movie)

Parámetros:
movie - handle del vídeo

Descripción:
Devuelve la altura de un vídeo.

MoviePlaying(movie)

Parámetros:
movie - handle del vídeo

Descripción:
Devuelve 1 (true) si la película especificada está siendo reproducida.

SONIDO/MÚSICA

LoadSound (filename$)

Definición:
Carga un sonido en memoria.

Descripción de los Parámetros:
filename$ - nombre del archivo de sonido. Formatos soportados: raw/wav/mp3/ogg

Descripción del Comando:
Este comando carga un archivo de sonido en memoria. Devuelve un número si se completó la operación correctamente, o 0 si hubo un problema durante el proceso. Debes asignar el valor devuelto a una variable (preferiblemente una variable Global) para poder ser reproducido con la función PlaySound. Echa un vistazo al ejemplo.

Ejemplo:

```
; Assign a global variable for the sound
Global sndPlayerDie

; Load the sound file into memory

sndPlayerDie=LoadSound("sounds/die.wav")

; Play the sound

PlaySound sndPlayerDie
```

FreeSound sound_variable

Definición:
Elimina el sonido y libera la memoria que estaba usando.

Descripción de los Parámetros:
sound_variable - handle del sonido

Descripción del Comando:
Cuando has acabado de usar un efecto de sonido, deberías liberar la memoria que estuviera usando y borrar el sonido. Este comando eliminará un sonido cargado con el comando LoadSound. ¿Por qué iba yo a querer hacer esto? Quizás tengas diferentes grupos de sonidos para cada nivel del juego. Quizás el bucle de música cambia cada nivel. Quieres hacer las cosas bien y manejar tus propios recursos. Nota: Cuando cierras un programa, todos los recursos cargados se liberan automáticamente.

Ejemplo:

```
; Load a sound into memory
sndOneUp=LoadSound("audio\1up.wav")

; Free the memory up and delete the sound
FreeSound sndOneUp
```

LoopSound sound_variable

Definición:
Establece que se reproduzca un bucle infinito de un sonido cargado previamente.

Descripción de los Parámetros:
sound_variable = handle del sonido

Descripción del Comando:
Este comando establece un archivo de sonido (.WAV o .MP3) para que se reproduzca en un bucle sin fin (como la música de fondo). Debes cargar antes un sonido usando el comando LoadSound. Usa una variable Global para asegurarte de que el sonido puede ser reproducido desde cualquier parte de tu programa. Nota: Este comando NO REPRODUCE el bucle, sólo hace que la próxima vez que uses PlaySound se reproducirá indefinidamente.

Ejemplo:

```
; Assign a global variable for the sound loop
Global sndMusicLoop

; Load the sound loop file into memory

sndMusicLoop=LoadSound("sounds/loop1.wav")

; Set the sound loop

LoopSound sndMusicLoop

PlaySound sndMusicLoop
```

SoundPitch sound_variable, hertz

Definición:
Cambia la frecuencia de un sonido.

Descripción de los Parámetros:
sound_variable = handle del sonido obtenido al cargarlo con el comando LoadSound
hertz = velocidad en hertzios (más de 44000 Hz).

Descripción del Comando:
Cambia la frecuencia (y con ello la velocidad) de un sonido cargado anteriormente con el comando LoadSound. Cambiando la frecuencia, puedes reusar sonidos para diferentes usos o simular un continuo 'arriba-abajo'. Para crear un sonido con mucha frecuencia, aumenta los hertzios. Y al contrario disminúyelos para obtener uno con poca frecuencia. Nota: ésta es en relación con el número original de hertzios del sonido.

Ejemplo:

```
; Load the sound (11,000 hertz)

snd1Up = LoadSound("audio\oneup.wav")

; Play the sound normally
PlaySound snd1Up

; Change the pitch UP and play it
SoundPitch snd1Up, 22000
PlaySound snd1Up

; Change the pitch down and play it
SoundPitch snd1Up, 8000
PlaySound snd1Up
```

SoundVolume sound_variable,volume#

Definición:
Cambia el volumen de un sonido.

Descripción de los Parámetros:
sound_variable = handle del sonido obtenido cuando fue cargado con el comando LoadSound
volume# = número de coma flotante desde 0 (silencio) a 1 (máximo volumen)

Descripción del Comando:
Cambia el volumen de un efecto de sonido usando este comando. Este comando usa números de coma flotante de 0 a 1 para controlar el volumen. Por favor, mira ChannelVolume para más opciones.

Ejemplo:

```
; Load sound sample
sndDeath=LoadSound("audio\death.wav")

; Change volume level to half
SoundVolume sndDeath,.5
```

```
; Play sound
PlaySound sndDeath


SoundPan sound_variable,pan#

Definición:

Cambia el balance del efecto de sonido especificado.
Descripción de los Parámetros:

sound_variable = handle del sonido obtenido al cargarlo con el comando LoadSound.
pan# = número de coma flotante desde -1 (izquierda) a 0 (centro) a 1 (derecha)

Descripción del Comando:

Usa este comando para cambiar el balance de tu sonido entre los altavoces izquierdo y derecho (o
restaurarlo al centro).

Ejemplo:

; Load sound sample
sndDeath=LoadSound("audio\death.wav")

; Pan sound effect half to the left
SoundPan sndDeath,-.5

; Play sound
PlaySound sndDeath
```

PlaySound sound_variable

Definición:
Reproduce un sonido cargado previamente en memoria.

Descripción de los Parámetros:
sound_variable = handle del sonido obtenido al usar el comando

Descripción del Comando:
Reproduce un sonido cargado previamente. Mira el ejemplo. Necesitarás asignar un handle de canal al sonido cuando lo reproduzcas. Todas las subsecuentes operaciones con el sonido necesitan usar el handle del CANAL, no del sonido - como StopChannel, PauseChannel, ResumeChannel, ChannelPitch, ChannelVolume, ChannelPan y ChannelPlaying.

Ejemplo:

```
; Assign a global variable for the sound
Global sndPlayerDie

; Load the sound file into memory

sndPlayerDie=LoadSound("sounds/die.wav")

; Play the sound

chnDie=PlaySound sndPlayerDie
```

PlayMusic (filename$)

Definición:
Carga y reproduce una pieza de música.

Descripción de los Parámetros:
filename$ - nombre del archivo de música. Formatos soportados: raw/mod/s3m/xm/it/mid/rmi/wav/mp2/mp3/ogg/wma/asf

Descripción del Comando:
Este comando cargará y reproducirá un archivo de música. DEBES usar una variable de canal para parar o ajustar la reproducción. Puedes usar StopChannel, PauseChannel, ResumeChannel, etc. con este comando.

Cada vez que llames al comando PlayMusic, el archivo será cargado de nuevo y reproducido. Esto significa que si usas el comando mientras algunos gráficos se están moviendo por la pantalla, se puede producir una pequeña ralentización. Para solucionar este problema, a lo mejor te convendría usar los comandos PlaySound/LoopSound en vez de este.

Ejemplo:

```
; Load and play the background music

chnBackground=PlayMusic("music\background.wav")
```

PlayCDTrack track,[mode]

Definición:
Reproduce una pista de un CD de audio.

Descripción de los Parámetros:
track = nº de pista a reproducir
mode = 1; sólo se reproduce una vez, 2; bucle infinito, 3; hasta el final del CD

Descripción del Comando:
Puedes reproducir un CD de audio a través de Blitz Basic con este comando. El parámetro opcional MODE permite variaciones en la reproducción. Recuerda, esta reproducción sólo se puede llevar acabo si dentro de tu Pc tienes unida la unidad de CD-Rom con tu tarjeta de sonido. Muchos Pc's (por alguna razón), no tienen este cable conectado correctamente. En este caso, NO oirás el sonido del CD aunque oirás los demás sonidos y músicas. Necesitarás asignar el comando a una variable porque este comando devuelve un handle del canal.

Ejemplo:

```
; PlayCDTrack example

; Get a track to play from user
track=Input$("Enter a CD track number to play:")

; Play the track, assign a channel - just play once
chnCD=PlayCDTrack(track,1)

; Figure out what time it is now
oldTime=MilliSecs()
; Play until the channel is over or ESC
While ChannelPlaying(chnCD) And (Not KeyHit(1))
; clear and print the time elapsed
Cls
Locate 0,0
Print "Time Elapsed (sec):" + ((MilliSecs()-oldTime)/1000)
Wend

; Stop the channel
StopChannel chnCD
```

StopChannel channel_handle

Definición:
Para un canal de sonido en reproducción.

Descripción de los Parámetros:
channel_handle = handle del canal obtenido cuando el sonido fue reproducido

Descripción del Comando:
Este comando remplaza a StopSound en las últimas versiones de Blitz Basic. Una vez tengas a un sonido reproduciendo y un canal asociado a éste, usa este comando para parar el sonido. Trabaja con todo tipo de canales de sonido, incluyendo MP3, WAV, MIDI y CD.

Ejemplo:

```
; Ejemplo Channel

Print "Cargando sonido ..."
; Carga el ejemplo - necesitarás tener este sonido en tu ordenador
; para mejores resultados, créalo alrededor de 5-10 segundos...
sndWave=LoadSound("level1.wav")
; Prepara el sonido para reproducirlo infinitas veces
LoopSound sndWave

chnWave=PlaySound(sndWave)

Print "Reproduciendo el sonido durante 2 segundos..."
Delay 2000

Print "Parando el sonido durante 2 segundos..."
PauseChannel chnWave
Delay 2000

Print "Volviendo a reproducir el sonido..."
ResumeChannel chnWave
Delay 2000

Print "Cambiando la frecuencia del sonido..."
;StopChannel chnWave
ChannelPitch chnWave, 22000
Delay 2000

Print "Reproduciendo el sonido con la nueva frecuencia..."
Delay 2000

Print "Sólo el altavoz izquierdo"
ChannelPan chnWave,-1
Delay 2000

Print "Sólo el altavoz derecho"
ChannelPan chnWave,1
Delay 2000

Print "¡Todo hecho!"
StopChannel chnWave
```

PauseChannel channel_handle

Definición:
Para la reproducción de un canal de sonido.

Descripción de los Parámetros:
channel_handle = handle del canal

Descripción del Comando:
Cuando estás reproduciendo un canal de sonido, puede llegar un momento en que desees pausar el sonido por alguna razón (p.e. para reproducir otro efecto de sonido). Este comando lo hace - y el canal puede ser resumido con el comando ResumeChannel. Puedes usar StopChannel para parar el sonido.

Ejemplo:
Ver Ejemplo Channel

ResumeChannel channel

Parámetros:
channel - un canal de música o sonido asignado previamente con LoadSound, PlayMusic, etc.

Descripción:
ResumeChannel se usa para continuar la reproducción de un sonido o pista de música del canal dado después de que pararás temporalmente su reproducción con el comando PauseChannel.

Ejemplo:

```
Graphics 640, 480, 0, 2

musicchannel = PlayMusic ("oohyeahbaby.mp3") ; Remplaza este archivo por uno de tu disco duro!!

Repeat

    Print "Pulsa una tecla para pausar la música..."
    WaitKey

    PauseChannel musicchannel

    Print "Pulsa una tecla para continuar la música..."
    WaitKey

    ResumeChannel musicchannel

Until KeyHit (1)

End
```

ChannelPitch channel_handle, hertz

Definición:
Altera los hertzios (frecuencia) de un canal de sonido.

Descripción de los Parámetros:
channel_handle = variable asignada al canal cuando es reproducido
hertz = frecuencia que se va a aplicar al canal (8000-44000)

Descripción del Comando:
Puedes alterar la frecuencia de un canal de sonido que se esté reproduciendo (o en uso, sólo pausado). ¡Estoy seguro de que puedes pensar numerosos usos para este comando! Aunque se parece a SoundPitch, este comando te permite cambiar la frecuencia de cada canal en uso INDIVIDUALMENTE.

Ejemplo:
Ver Ejemplo Channel

ChannelVolume channel_handle, volume#

Definición:
Ajusta el nivel de volumen de un canal de sonido.

Descripción de los Parámetros:
channel_handle = variable asignada al canal cuando es reproducido
volume# = nivel de volumen (número real entre 0 y 1)

Descripción del Comando:
Mientras SoundVolume cambia el nivel de volumen del programa entero, este comando te permitirá ajustar el volumen de cada canal. Extremadamente útil. El valor de volumen es un número real entre 0 y 1 (0 = silencio, 0.5 = volumen medio, 1 = volumen completo). ¡Puedes usar ChannelPitch y ChannelPan también!

Ejemplo:
Ver Ejemplo Channel

ChannelPan channel_handle, pan#

Definición:
Balance de sonido entre los canales izquierdo y derecho.

Descripción de los Parámetros:
channel_handle = variable asignada al canal cuando es reproducido
pan# = balance (valor real) que denota el canal en que se reproducirás el sonido

Descripción del Comando:
Cuando quieras hacer cambios de balance realistas, deberás usar este comando. Éste te permite cambiar el balance para hacer que suene en el altavoz izquierdo o derecho. El balance debe estar entre -1 y 1, siendo 1 el centro, -1 a la izquierda del todo y 1 a la derecha del todo.

Ejemplo:
Ver Ejemplo Channel

ChannelPlaying (channel_handle)

Definición:
Comprueba si un canal de sonido está todavía reproduciéndose.

Descripción de los Parámetros:
channel_handle = variable asignada al canal cuando es reproducido

Descripción del Comando:
A menudo necesitarás saber si un canal de sonido está siendo reproducido o no. Este comando devolverá 1 si el sonido está siendo reproducido todavía ó 0 si ha acabado. Úsalo para recomenzar tu música de fondo o algún otro sonido que puede haber sido parado inintencionadamente. Nota: Este comando actualmente no funciona con un canal asignado a una pista de CD.

Ejemplo:
Ver Ejemplo Channel

GRÁFICOS

Graphics width, height, color depth,[mode]

Definición:
Establece el modo de vídeo.

Descripción de los Parámetros:
width = ancho de la pantalla en pixels (640, 800, etc)
height = alto de la pantalla en pixels (480, 600, etc)
color depth = profundidad de color en bits (16, 24, o 32 bit - usa 16 si es posible!)
mode = Modo de vídeo (mira la descripción); Opcional

Descripción del Comando:
Este comando establece a Blitz en el modo gráfico con la altura, anchura y profundidad de color especificadas. Este comando debe ser ejecutado antes de usar algún comando de dibujo. La profundidad de color es OPCIONAL. Usa el comando GfxModeExists para asegurarte de que un modo de vídeo es soportado antes de usarlo. Las resoluciones más comunes y seguras son 640x480 y 800x600. Recuerda, cada paso hacia arriba en resolución y color marca un gran salto en requerimientos de sistema y puede ser inversamente proporcional a su rendimiento. Si usas la resolución más baja posible mucha más gente podrá jugar y disfrutar tu juego. La mayoría de los ordenadores soporta una resolución de 640x480. Un parámetro extra se usa al final del comando después de la profundidad de color. Aquí están los parámetros:
0 : auto - en ventana en modo debug, pantalla completa en modo no debug (este está por defecto)
1 : pantalla completa
2 : modo ventana
3 : modo ventana (escalada)

Ejemplo:

```
;GRAPHICS Example

; Set The Graphic Mode
Graphics 800,600

; Now print something on the graphic screen
Text 0,0, "This is some text printed on the graphic screen (and a white box)! Press ESC ..."

; Now for a box
Rect 100,100,200,200,1

While Not KeyHit(1)
Wend
```

SetBuffer buffer

Definición:
Se usa para establecer el buffer de dibujo actual.

Descripción de los Parámetros:
buffer - puede ser FrontBuffer(), BackBuffer() o un ImageBuffer() El buffer por defecto es el FrontBuffer

Descripción del Comando:
Usa este comando para establecer el buffer de dibujo actual. SetBuffer también restablece el origen a 0,0 y el Viewport al ancho y alto del buffer.

Ejemplo:

```
SetBuffer FrontBuffer() ;Sets FrontBuffer as the current drawing buffer
```

FrontBuffer()

Definición:
Designa el FrontBuffer como buffer de dibujo.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Es importante entender los buffers cuando programas un juego. Lo que el jugador puede ver normalmente en su pantalla es el FrontBuffer. Todo lo que dibujes en este buffer sera INMEDIATAMENTE visible para el jugador.

Ejemplo:

```
; FrontBuffer()/Rect Example

; Engage graphics mode
Graphics 640,480,16

; Set the drawing buffer to front - instant drawing ops!
SetBuffer FrontBuffer()

; Repeat this until user presses ESC
While Not KeyHit(1)
; Set a random color
Color Rnd(255),Rnd(255),Rnd(255)
; Draw a rectangle at a random location, with a random width and height
; plus randomly choose if the rectangle is solid or just an outline
Rect Rnd(640),Rnd(480),Rnd(50),Rnd(50),Rnd(0,1)
; Blitz is so dang fast, we need a delay so you can watch it draw!
Delay 10
Wend
```

BackBuffer()

Definición:
Indica que el BackBuffer se dibuje en el buffer.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Éste es un valor usado normalmente con SETBUFFER para denotar el buffer secundario no visible llamado Back Buffer sea dibujado. En la MAYORÍA de las situaciones de un juego, querrás usar el comando BackBuffer() para dibujar operaciones y usando Flip para traer ese buffer al FrontBuffer() donde puede ser visto. Hay otros usos del comando, pero éste es el principal. Mira SETBUFFER para más información y observa detenidamente el ejemplo. Si estableces todas las operaciones de dibujo en el BackBuffer() NO verás nada hasta que no hagas un FLIP.

Ejemplo:

```
; Flip/Backbuffer()/Rect Example

; Set Graphics Mode
Graphics 640,480

; Go double buffering
SetBuffer BackBuffer()

; Setup initial locations for the box
box_x = -20 ; negative so it will start OFF screen
box_y = 100

While Not KeyHit(1)
Cls ; Always clear screen first
Rect box_x,box_y,20,20,1 ; Draw the box in the current x,y location
Flip ; Flip it into view
box_x = box_x + 1 ; Move the box over one pixel
If box_x = 640 Then box_x=-20 ; If it leaves the Right edge, reset its x location
Wend
```

LoadBuffer (buffer, filename$)

Definición:
Carga una imagen directamente en un buffer de Blitz.

Descripción de los Parámetros:
buffer = buffer de destino
filename$ = ruta completa y nombre del archivo

Descripción del Comando:
Hay mil y un usos para este comando, pero probablemente el más usado podría ser mostrar una pantalla de título o alguna otra imagen que sólo sea una vez vista (como en nuestro ejemplo).
También puedes cargarlas en el image buffer o back buffer. La imagen se escala para que coincida con el tamaño del buffer. Este comando devuelve 1 si no hubo ningún error, 0 en caso contrario.

Ejemplo:

```
; LoadBuffer example

; Set graphics mode
Graphics 800,600,16

; Load an image directly to the front buffer (your location may be different)
LoadBuffer (FrontBuffer(),"C:\Program Files\Blitz Basic\samples\blitzanoid\gfx\title.bmp")

; wait for ESC so user gets to see the screen
While Not KeyHit(1)
Wend
```

SaveBuffer (buffer,filename$)

Definición:
Guarda el buffer de vídeo seleccionado en un archivo bitmap.

Descripción de los Parámetros:
buffer = buffer a guardar; FrontBuffer() o BackBuffer()
filename$ = ruta completa/nombre de archivo

Descripción del Comando:
Normalmente se usa para obtener una captura de pantalla. Este comando guardará el buffer que hayas especificado en un archivo .bmp. Recuerda, usa el nombre del buffer que desees guardar; FrontBuffer() para lo que hay actualmente visible en pantalla, y BackBuffer() para lo que hay detrás.

Ejemplo:

```
; Save the screen when player pushes F10

If KeyHit(10) Then
SaveBuffer(FrontBuffer(),"screenshot.bmp")
End If
```

LockBuffer buffer

Definición:
Bloquea un buffer para aumentar la velocidad de las operaciones con pixels.

Descripción de los Parámetros:
buffer = buffer a bloquear (opcional)

Descripción del Comando:

Después de aplicar LockBuffer sobre un buffer, los únicos comandos gráficos que puedes utilizar son ReadPixel, WritePixel, ReadPixelFast, WritePixelFast, CopyPixelFast, y CopyPixel. Debes usar UnlockBuffer antes de usar otros comandos gráficos. El parámetro 'buffer' no es indispensable. Si se omite, el buffer por defecto será el establecido con la función SetBuffer. Mira los otros comandos para más información.

Ejemplo:

```
; Comandos Gráficos de Alta Velocidad

Graphics 640,480,16

; Dibujo un montón de mierda en la pantalla
For t= 1 To 1000
Color Rnd(255),Rnd(255),Rnd(255)
Rect Rnd(640),Rnd(480),Rnd(150),Rnd(150),Rnd(1)
Next

Delay 3000

; Copio la mitad superior de la pantalla encima de la inferior
; usando los 'fast pixels' y buffer's bloqueados
For x = 1 To 640
For y = 1 To 240
LockBuffer FrontBuffer()
WritePixelFast x,y+241,ReadPixelFast(x,y)
UnlockBuffer FrontBuffer()
Next
Next

Delay 3000

; Dibujo la mitad izquierda de la pantalla sobre la derecha
; usando los 'slow pixels' y buffer's desbloqueados
For x = 1 To 320
For y = 1 To 480
WritePixel x+320,y,ReadPixel(x,y)
Next
Next
```

UnlockBuffer buffer

Definición:
Desbloquea el buffer previamente bloqueado para realizar operaciones de pixels rápidas.

Descripción de los Parámetros:
buffer = un buffer de pantalla/imagen válido (opcional)

Descripción del Comando:
Después de usar LockBuffer sobre un buffer, los únicos comandos gráficos que puedes usar son ReadPixel, WritePixel, ReadPixelFast y WritePixelFast. Debes usar este comando antes de utilizar otros comandos gráficos.

El parámetro buffer no es necesario. Si se omite, el buffer establecido con el comando SetBuffer será usado.
Mira los otros comandos para más información.

ReadPixel (x,y,[buffer])

Parámetros:
x - coordenada x
y - coordenada y
buffer (opcional) - nombre del buffer donde se leerá el pixel, p.e. BackBuffer()

Descripción:
Lee un valor de color del buffer actual o del especificado, y lo devuelve. El valor devuelto está en la forma de un entero que contiene los valores alfa, rojo, verde y azul del pixel. Puedes usar este comando en un buffer bloqueado para aligerar la velocidad. Mira LockBuffer. Mira también: GetColor, ReadPixelFast.

Ejemplo:

```
; ReadPixel/WritePixel Example
; ---------------------------

Graphics 640,480,16

Print "Press a key to read color values (this may take a few seconds)"
WaitKey()

; Load and draw an image on to the screen - can be anything
pic=LoadImage("media/blitz_pic.bmp")
DrawImage pic,0,0

; Initialise an array big enough to fit all the color information of the screen
Dim pix(GraphicsWidth(),GraphicsHeight())

; Use ReadPixel to get all the color information of the screen
For y=0 To GraphicsHeight()
For x=0 To GraphicsWidth()
pix(x,y)=ReadPixel(x,y)
Next
Next

Cls
Locate 0,0
Print "Press a key to write pixels (this may takes a few seconds)"
Print "Once this has finished, you can then press a key to end the program"

WaitKey()

; Use WritePixel to redraw the screen using the color information we got earlier
For y=0 To GraphicsHeight()
For x=0 To GraphicsWidth()
WritePixel x,y,pix(x,GraphicsHeight()-y) ; get y array value in backwards order, to flip screen
Next
Next

WaitKey()
```

WritePixel x,y,argb,[buffer]

Parámetros:
x - coordenada x
y - coordenada y
argb - valor de color argb del pixel (alpha (alfa), red (rojo), green (verde), blue(azul))
buffer (opcional) - nombre del buffer de destino, por ejemplo, BackBuffer()

Descripción:
Escribe un valor de color en el buffer especificado. Puedes usar este comando en un buffer bloqueado para mejorar la velocidad.
Mira LockBuffer. Mira también: Plot, WritePixelFast.

Ejemplo:

```
; ReadPixel

ReadPixelFast (x,y,[buffer])

Parámetros:

x - coordenada x
y - coordenada y
```

buffer (opcional) - nombre del buffer donde se leerá el pixel, p.e. BackBuffer()

Descripción:

Lee un valor de color del buffer actual o del especificado, y lo devuelve. El valor devuelto está en la forma de un entero que contiene los valores alfa, rojo, verde y azul del pixel. ***IMPORTANTE****: DEBES usar este comando en un buffer bloqueado, de lo contrario el comando fallará. Mira LockBuffer. También debes asegurarte de que las coordenadas que estás leyendo son válidas.*
CUIDADO*: Si no sigues el anterior aviso, puedes cargarte tu Pc. Mira también: GetColor, ReadPixel.*

Ejemplo:

```
; ReadPixelFast/WritePixeFast Example
Graphics 640,480,16

Print "Press a key to read color values"
WaitKey()

; Load and draw an image on to the screen - can be anything
pic=LoadImage("media/blitz_pic.bmp")
DrawImage pic,0,0

; Initialise an array big enough to fit all the color information of the screen
Dim pix(GraphicsWidth(),GraphicsHeight())

; Lock buffer before using ReadPixelFast
LockBuffer

; Use ReadPixel to get all the color information of the screen
For y=0 To GraphicsHeight()
For x=0 To GraphicsWidth()
pix(x,y)=ReadPixelFast(x,y)
Next
Next

; Lock buffer after using ReadPixelFast
UnlockBuffer

Cls
Locate 0,0
Print "Press a key to write pixels"
Print "Once this has finished, you can then press a key to end the program"

WaitKey()

; Lock buffer before using WritePixelFast
LockBuffer

; Use WritePixel to redraw the screen using the color information we got earlier
For y=0 To GraphicsHeight()
For x=0 To GraphicsWidth()
WritePixelFast x,y,pix(x,GraphicsHeight()-y) ; get y array value in backwards order, to flip screen
Next
Next

; Unlock buffer after using WritePixelFast
UnlockBuffer

WaitKey()
```

WritePixelFast x,y,argb,[buffer]

Parámetros:
x - coordenada x
y - coordenada y
argb - valor de color argb del pixel (alpha (alfa), red (rojo), green (verde), blue(azul))
buffer (opcional) - nombre del buffer de destino, por ejemplo, BackBuffer()

Descripción:
*Escribe un valor de color en el buffer especificado. **IMPORTANTE**: DEBES usar este comando sobre un buffer bloqueado, sino el comando fallará. Mira LockBuffer. También debes asegurarte de que las coordenadas especificadas son válidas, sino acabarás sobreescribiendo otras áreas de memoria. **CUIDADO**: Si no sigues la advertencia anterior, puedes bloquear tu ordenador. Mira también: Plot, WritePixel.*

Ejemplo:

```
; ReadPixelFast
```

CopyPixel src_x,src_y,src_buffer,dest_x,dest_y,[dest_buffer]

Parámetros:
***src_x** - coordenada x del pixel que se copiará. **src_y** - coordenada y del pixel que se copiará*
***src_buffer** - nombre del buffer desde donde se copiará, p. e. ImageBuffer(). **dest_x** - coordenada x del pixel donde se copiará*
***dest_y** - coordenada y del pixel donde se copiará. **dest_buffer** (optional) - nombre del buffer donde se copiará, p. e. BackBuffer()*

Descripción:
Copia un pixel en otra localización. Puedes usar este comando en un buffer bloqueado para aumentar velocidad. Mira LockBuffer. Mira también: CopyPixelFast.

Ejemplo:

```
; CopyPixel/CopyPixelFast Example

Graphics 640,480,16

Print "Press a key to use CopyPixel to copy the top half of an image to the frontbuffer"
WaitKey()

; Load an image - can be anything
pic=LoadImage("media/blitz_pic.bmp")

; Use CopyPixel to copy the top half of the image to the frontbuffer
For y=0 To ImageHeight(pic)/2
For x=0 To ImageWidth(pic)
CopyPixel x,y,ImageBuffer(pic),x,y
Next
Next

Locate 0,GraphicsHeight()/2
Print "Press a key to use CopyPixelFast to copy the bottom half of the image"
Print "Once this has finished, you can then press a key to end the program"

WaitKey()

; Lock buffer before using CopyPixelFast
LockBuffer

; Use CopyPixelFast to copy the bottom half of the image to the frontbuffer
For y=0 To (ImageHeight(pic)/2)+ImageHeight(pic)
For x=0 To ImageWidth(pic)
CopyPixelFast x,y,ImageBuffer(pic),x,y
Next
Next

; Unlock buffer after using CopyPixelFast
UnlockBuffer

WaitKey()
```

CopyPixelFast src_x,src_y,src_buffer,dest_x,dest_y,[dest_buffer]

Parámetros:
src_x *- coordenada x del pixel que se copiará.* ***src_y*** *- coordenada y del pixel que se copiará*
src_buffer *- nombre del buffer desde donde se copiará, p. e. ImageBuffer().* ***dest_x*** *- coordenada x del pixel donde se copiará*
dest_y *- coordenada y del pixel donde se copiará.* ***dest_buffer*** *(optional) - nombre del buffer donde se copiará, p. e. BackBuffer()*

Descripción:
*Copia un pixel en otra localización. IMPORTANTE: *DEBES* usar este comando en un buffer BLOQUEADO, sino fallará. Mira LockBuffer. También debes asegurarte de que las coordenadas especificadas son válidas, sino acabará sobreescribiendo otras áreas de memoria. ATENCIÓN: Si no haces caso de la anterior advertencia, puedes hacer que tu ordenador se bloquee. Mira también: CopyPixel.*

Ejemplo:
Ver CopyPixel

CopyRect src_x,src_y,src_width,src_height,dest_x,dest_y,[src_buffer],[dest_buffer]

Definición:
Copia un rectángulo de gráficos desde un buffer a otro.

Descripción de los Parámetros:
src_x *= coordenada x desde donde se empezará a copiar.* ***src_y*** *= coordenada y desde donde se empezará a copiar*
src_width *= anchura del área a copiar.* ***src_height*** *= altura del área a copiar*
dest_x *= coordenada x de destino.* ***dest_y*** *= coordenada y de destino*
src_buffer *= handle del buffer de imagen de origen (opcional).* ***dest_buffer*** *= handle del buffer de imagen de destino (opcional)*

Descripción del Comando:
Copia un rectángulo de gráficos desde un buffer a otro. Si el buffer es omitido, se usará el buffer actual.

Ejemplo:

```
; CopyRect Example

; Turn on graphics mode
Graphics 800,600,16

; create a blank image
gfxBlank=CreateImage (300,300)

; Fill the screen with random boxes in random colors
For t = 1 To 1000
Rect Rand(800),Rand(600),Rand(100),Rand(100),Rand(0,1)
Color Rand(255),Rand(255),Rand(255)
Next

; Wait a couple of seconds so the user can see it
Delay 2000

; Copy graphics randomly from the front buffer to the blank image
CopyRect Rand(800),Rand(600),300,300,0,0,FrontBuffer(),ImageBuffer(gfxBlank)

; Clear the screen, draw the copied to image, wait for user to hit a key
Cls
DrawImage gfxBlank,0,0
WaitKey
```

Viewport x, y, width, height

Definición:
Restringe los comandos de dibujo a un área específica de la pantalla.

Descripción de los Parámetros:
x = coordenada x de la esquina superior izquierda del nuevo área
y = coordenada x de la esquina superior izquierda del nuevo área
width = ancho del área (en pixels)
height = alto del área (en pixels)

Descripción del Comando:
Hay MUCHAS MUCHAS veces que quieres dibujar gráficos (aliens, barcos, etc) SÓLO en una región de la pantalla mientras dejas las demás solas. Hay millones de usos para esto; poner un radar/mapa en la pantalla, mostrar ventanas del estilo de Ultima, scroll's, etc. Es un poco más compleja que la mayoría de los comandos gráficos, así que asegúrate de entenderlo bien antes de intentar usarlo. El mayor consejo que puedeo darte acerca de este comando es: ¡RECUERDA BORRAR EL VIEWPORT CUANDO HAYAS ACABADO! Haz esto estableciendo el viewport de forma que ocupe toda la pantalla (es decir, Viewport 0,0,640,480 si tu juego está en 640x480). Mira cuidadosamente el ejemplo. Recuerda, los dos últimos parámetros no indican la localización FINAL del viewport, sino el TAMAÑO empezando por las primeras coordenadas.

Ejemplo:

```
; Ejemplo ViewPort

; Activo el Modo Gráfico
Graphics 800,600

; Activo el Double Buffering
SetBuffer BackBuffer()

; Creo un viewport empezando en 100,100 con un tamaño de 200,200 pixels
Viewport 100,100,200,200

; Dibujo infinitos rectángulos aleatoriamente con colores también aleatorios
While Not KeyHit(1)
Cls ; Limpio la pantalla
For t = 1 To 100 ; Hago 100 rectángulos cada vez
Color Rnd(255),Rnd(255),Rnd(255) ; Color aleatorio
Rect Rnd(800),Rnd(600),Rnd(300),Rnd(300),Rnd(0,1) ; Tamaño y posición aleatoria, algunos rellenos
Next ; Repite el bucle de dibujo
Flip
Wend
```

Origin x,y

Definición:
Te permite elegir un punto de origen para todos tus comandos de dibujo.

Descripción de los Parámetros:
x = coordenada x
y = coordenada y

Descripción del Comando:
Este comando establece un punto de origen para todos los subsecuentes comandos de dibujo. Éste puede ser positivo o negativo.

Ejemplo:

```
; Origin example
Graphics 800,600,16

; Offset drawing options with origin command -200 in each direction
Origin -200,-200

; Wait for ESC to hit
While Not KeyHit(1)

; Draw an oval - SHOULD be at the exact center, but it isn't!
Oval 400,300,50,50,1
Wend
```

Flip [vwait]

Definición:
En un entorno con doble buffering, invierte el BackBuffer a la vista.

Descripción de los Parámetros:
vwait = establece a TRUE para esperar a la sincronización vertical para acabar

Descripción del Comando:
Cuando estés en doble buffering (o haciendo operaciones de dibujo en el BackBuffer(), y hayas dibujado todos tus elementos) necesitarás 'voltear' el Backbuffer para que sea visible en el FrontBuffer(). Este comando realiza esta función. Éste te permite dibujar muchos gráficos rápidamente y voltearlos a la vista. Mira BackBuffer() para más información.

Ejemplo:

```
; Flip/Backbuffer()/Rect Example

; Set Graphics Mode
Graphics 640,480

; Go double buffering
SetBuffer BackBuffer()

; Setup initial locations for the box
box_x = -20 ; negative so it will start OFF screen
box_y = 100

While Not KeyHit(1)
Cls ; Always clear screen first
Rect box_x,box_y,20,20,1 ; Draw the box in the current x,y location
Flip ; Flip it into view
box_x = box_x + 1 ; Move the box over one pixel
If box_x = 640 Then box_x=-20 ; If it leaves the Right edge, reset its x location
Wend
```

VWait [frames]

Definición:
Espera a que acaben uno o más "barridos" o ciclos verticales de la pantalla antes de continuar.
Descripción de los Parámetros: [frames] = número opcional de frames a esperar

Descripción del Comando:
Hay veces que quieres dibujar muy rápido, y tus operaciones de dibujo lo son tanto que ocurren cosas indeseables. Este comando fuerza a Blitz a esperar hasta que la 'línea' de dibujo alcance el tope inferior de la pantalla antes de continuar. Prueba el ejemplo con y sin el comando VWAIT.

Ejemplo:

```
; Ejemplo Vwait
Graphics 800,600,16

; Espera que pulses ESC para acabar
While Not KeyHit(1)
; Establece un color aleatorio
Color Rnd(255),Rnd(255),Rnd(255)
; Dibuja una línea aleatoria
Line Rnd(800),Rnd(600),Rnd(800),Rnd(600)
; Espera que acabe un ciclo vertical para continuar con el bucle
VWait
Wend
```

ScanLine()

Definición:
Devuelve la localización del scanline de la operación de dibujo.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Si por alguna razón necesitas saber la localización del scanline actual del sistema de dibujo, aquí tienes cómo obtenerla.

Ejemplo:

```
; ScanLine() Example

While Not KeyHit(1)
Print ScanLine()
Wend
```

GraphicsHeight()

Definición:
Devuelve la altura, en pixels, del actual modo gráfico.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando te dirá la altura, en pixels, del modo gráfico actual.

Ejemplo:

```
; GraphicsWidth(), GraphicsHeight(), GraphicsDepth(), GraphicsBuffer() example

; Set a graphics mode and buffer
Graphics 640,480,16
SetBuffer FrontBuffer()

; Print the details
Print "Screen width is: " + GraphicsWidth()
Print "Screen height is: " + GraphicsHeight()
Print "Screen color depth is: " + GraphicsDepth()
Print "Current buffer handle is: " + GraphicsBuffer()

; Wait for ESC before exiting
While Not KeyHit(1)
Wend
```

GraphicsWidth()

Definición:
Devuelve el ancho, en pixels, del actual modo gráfico.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando te dirá el ancho, en pixels, del modo gráfico actual.

Ejemplo:
Ver GraphicsHeight

GraphicsDepth()

Definición:
Devuelve la profundidad de color del modo gráfico actual.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando te dirá la profundidad de color del modo gráfico actual.

Ejemplo:
Ver GraphicsHeight

GraphicsBuffer()

Definición:
Devuelve el handle del buffer de gráficos actual.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Usa este comando para obtener el buffer que Blitz está usando actualmente.

Ejemplo:
Ver GraphicsHeight

Color rojo,verde,azul

Definición:
Establece el color que se usará en todas las operaciones de dibujo.

Descripción de los Parámetros:
rojo = valor de componente rojo (0-255)
verde = valor de componente verde (0-255)
azul = valor de componente azul (0-255)

Descripción del Comando:
Este comando establece el color de dibujo (usando un valor RGB) para todos los comandos de dibujo que se llamen después (Line, Rect, Text, etc.). Debes estar en modo Graphics para ejecutar este comando.

Ejemplo:

```
; Color, ColorRed(), ColorBlue(), ColorGreen() Example

; Gotta be in graphics mode
Graphics 640,480

; Change the random seed
SeedRnd MilliSecs()

; Let's set the color to something random
Color Rnd(0,255),Rnd(0,255),Rnd(0,255)

; Now let's see what they are!
While Not KeyHit(1)
Text 0,0,"This Text is printed in Red=" + ColorRed() + " Green=" + ColorGreen() + " Blue=" +
ColorBlue() + "!"
Wend
```

ClsColor rojo,verde,azul

Definición:
Selecciona el color para el comando Cls.

Descripción de los Parámetros:
rojo, verde y azul = número entre 0 y 255

Descripción del Comando:
Éste cambia el color para las siguientes llamadas a un CLS. Usa este comando cuando necesitas CLS para 'borrar' la pantalla con algún otro color que no sea negro.

Ejemplo:

```
;set ClsColor to red
ClsColor 255,0,0

;set current drawing buffer to the color set by the ClsColor command
Cls
```

```
Cls

Definición:

Borra el buffer de dibujo actual.
Descripción de los Parámetros:

Ninguno.

Descripción del Comando:

Este comando vaciará el buffer de dibujo actual de todos los gráficos y textos presentes y reinicia
el buffer de dibujo al color definido con el comando ClsColor.

Ejemplo:

;set ClsColor to red
ClsColor 255,0,0

;set current drawing buffer to the color set by the ClsColor command
Cls
```

Plot x,y

Definición:
Pone un pixel en la pantalla del color actual de dibujo.

Descripción de los Parámetros:
x= un nº desde 0 hasta el ancho del actual modo gráfico
y= un nº desde 0 hasta el alto del actual modo gráfico

Descripción del Comando:
Se usa para poner un pixel en las coordenadas especificadas del color de dibujo actual definido con el comando Color.

Ejemplo:

```
;Set the color to green
Color 0,255,0

;Draw a dot at location 100,200 with the color green
plot 100,200
```

Line x,y,x1,y1

Definición:
Dibuja una línea del color actual desde x,y hasta x1,y1.

Descripción de los Parámetros:
x=localización x de inicio de la línea
y=localización y de inicio de la línea
x1=localización x final de la línea
y1=localización y final de la línea

Descripción del Comando:
Este comando dibuja una línea, del color de dibujo actual, desde un punto de la pantalla hasta otro (desde x,y hasta x1,y1). Mira el ejemplo.

Ejemplo:

```
; Line example
Graphics 800,600,16
; Wait for ESC to hit
While Not KeyHit(1)
; Set a random color
Color Rnd(255),Rnd(255),Rnd(255)
; Draw a random line
Line Rnd(800),Rnd(600),Rnd(800),Rnd(600)
Wend
```

Rect x, y, width, height, solid

Definición:
Dibuja un rectángulo en la pantalla.

Descripción de los Parámetros:
x = coordenada x para empezar a dibujar
y = coordenada y para empezar a dibujar
width = ancho en pixels del rectángulo
height = alto en pixels del rectángulo
solid = False para hacerlo hueco o True para hacerlo sólido

Descripción del Comando:
Este comando dibujará un rectángulo del Color seleccionado actualmente. El último determina si el rectángulo esta relleno o hueco.

Ejemplo:

```
; Flip/Backbuffer()/Rect Example

; Set Graphics Mode
Graphics 640,480

; Go double buffering
SetBuffer BackBuffer()

; Setup initial locations for the box
box_x = -20 ; negative so it will start OFF screen
box_y = 100

While Not KeyHit(1)
Cls ; Always clear screen first
Rect box_x,box_y,20,20,1 ; Draw the box in the current x,y location
Flip ; Flip it into view
box_x = box_x + 1 ; Move the box over one pixel
If box_x = 640 Then box_x=-20 ; If it leaves the Right edge, reset its x location
Wend
```

Oval x,y,width,height[,solid]

Definición:
Dibuja un elipse en la posición especificada de la pantalla.

Descripción de los Parámetros:
x = coordenada x
y = coordenada y
width = ancho del elipse
height = alto del elipse
[solid] = 1 para hacer el elipse sólido

Descripción del Comando:
Úsalo para dibujar un elipse en las coordenadas de la pantalla que has elegido. Puedes hacer el elipse hueco o sólido.

Ejemplo:

```
; Oval example
Graphics 800,600,16

; Wait for ESC to hit
While Not KeyHit(1)
; Set a random color
Color Rnd(255),Rnd(255),Rnd(255)
; Draw a random oval
Oval Rnd(800),Rnd(600),Rnd(100),Rnd(100),Rnd(0,1)
Wend
```

GetColor x, y

Definición:
Establece el color de dibujo actual al color del pixel designado.

Descripción de los Parámetros:
x = coordenada x del pixel
y = coordenada y del pixel

Descripción del Comando:
Este comando trabaja como un capturador de color.

Ejemplo:

```
; GetColor Example

Graphics 320,200
SetBuffer BackBuffer()

For t = 1 To 1000
Color Rnd(255), Rnd(255), Rnd(255)
Rect Rnd(320),Rnd(200),10,10,1
Next

GetColor 100,100
Print "Box at 100,100 is RGB:" + ColorRed() + "," + ColorGreen() + "," + ColorBlue() + "!"
Flip
While Not KeyHit(1)
Wend
```

ColorRed()

Definición:
Devuelve el componente rojo del color de dibujo actual.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Usa este comando para devolver el componente verde del color RGB del color de dibujo actual. Usa ColorBlue() y ColorGreen() para los otros dos componentes de color.

Ejemplo:

```
; Color



ColorGreen()

Definición:

Devuelve el componente verde del color de dibujo actual.
Descripción de los Parámetros:

Ninguno.

Descripción del Comando:

Usa este comando para devolver el componente verde del color RGB del color de dibujo actual. Usa
ColorRed() y ColorBlue() para los otros dos componentes de color.

Ejemplo:

; Color
```

ColorBlue()

Definición:
Devuelve el componente azul del color de dibujo actual.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Usa este comando para devolver el componente azul del color RGB del color de dibujo actual. Usa ColorRed() y ColorGreen() para los otros dos componentes de color.

Ejemplo:
Ver Color

CountGFXModes()

Definición:
Devuelve el número de modos gráficos soportados por la tarjeta de vídeo actual.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Usa este comando para devolver el número de modos gráficos soportados por la tarjeta de vídeo actual. Usa GFXModeWidth, GFXModeHeight, y GFXModeDepth con cada número de modo de vídeo para determinar la anchura, altura y profundidad de color de cada modo. Mira el ejemplo.

Ejemplo:

```
; CountGFXModes()/GfxModeWidth/GfxModeHeight/GfxModeDepth example

intModes=CountGfxModes()

Print "There are " + intModes + "graphic modes available:"

; Display all modes including width, height, and color depth
For t = 1 To intModes
Print "Mode " + t + ":
Print "Width=" + GfxModeWidth(t)
Print "Height=" + GfxModeHeight(t)
Print "Height=" + GfxModeDepth(t)
Next
```

CountGfxDrivers()

Definición:
Devuelve el número de tarjetas gráficas de tu sistema.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Algunos ordenadores pueden tener más de una tarjeta de vídeo. Una vez que saves cuántos drivers hay instalados, puedes conseguir sus nombres con el comando GfxDriverName$ y mostrarlos para elegir cuál usar. Una vez que el usuario lo ha elegido, puedes seleccionar la tarjeta gráfica que vas a utilizar con SetGfxDriver. Normalmente, esto no será necesario en programación 2D.

Ejemplo:

```
; GfxDriver Examples

; Count how many drivers there are
totalDrivers=CountGfxDrivers()
Print "Choose a driver to use:"

; Go through them all and print their names (most people will have only 1)
```

```
For t = 1 To totalDrivers
Print t+") " + GfxDriverName$(t)
Next

; Let the user choose one
driver=Input("Enter Selection:")

; Set the driver!
SetGfxDriver driver
Print "Your driver has been selected!"
```

CountGfxDrivers()

Definición:
Devuelve el número de tarjetas gráficas de tu sistema.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Algunos ordenadores pueden tener más de una tarjeta de vídeo. Una vez que saves cuántos drivers hay instalados, puedes conseguir sus nombres con el comando GfxDriverName$ y mostrarlos para elegir cuál usar. Una vez que el usuario lo ha elegido, puedes seleccionar la tarjeta gráfica que vas a utilizar con SetGfxDriver. Normalmente, esto no será necesario en programación 2D.

Ejemplo:
Ver GfxDriver Examples

SetGfxDriver index

Definición:
Selecciona el controlador gráfico a utilizar para los comandos de dibujo.

Descripción de los Parámetros:
index = número obtenido con el comando CountGfxDrivers

Descripción del Comando:
Algunos ordenadores tienen más de una tarjeta de vídeo/controlador gráfico instalado (un buen ejemplo es un ordenador con una tarjeta de vídeo primaria y una Voodoo2 u otra tarjeta gráfica).
Una vez sepas cuántos controladores hay usando el comando CountGfxDrivers(), puedes mostrarlos usando GfxDriverName$ para que el usuario elija el que quiera. Una vez el usuario haya elegido (o tú), puedes establecerlo con este comando. Normalmente esto no será necesario en programas 2D.

Ejemplo:
Ver GfxDriver Examples

GFXModeWidth (mode)

Definición:
Devuelve el ancho del modo de vídeo actual.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Una vez sepas los modos de vídeo disponibles usando CountGFXModes(), puedes cambiarlos determinando el ancho, alto y la profundidad de color. Usa este comando para obtener el ancho del modo. Usa los comandos GFXModeHeight y GFXModeDepth para obtener los parámetros restantes.

Ejemplo:
Ver CountGFXModes

GFXModeHeight (mode)

Definición:
Devuelve la altura del modo de vídeo actual.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Una vez sepas los modos de vídeo disponibles usando CountGFXModes(), puedes cambiarlos determinando el ancho, alto y la profundidad de color. Usa este comando para obtener la altura del modo. Usa los comandos GFXModeWidth y GFXModeDepth para obtener los parámetros restantes.

Ejemplo:
Ver CountGFXModes

GFXModeDepth (mode)

Definición:
Devuelve la profundidad de color del modo de vídeo seleccionado.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Una vez sepas los modos de vídeo disponibles usando CountGFXModes(), puedes cambiarlos determinando el ancho, alto y la profundidad de color. Usa este comando para obtener la profundidad de color del modo. Usa los comandos GFXModeWidth y GFXModeHeight para obtener los parámetros restantes.

Ejemplo:
Ver CountGFXModes

GfxModeExists (width,height,depth)

Definición:
Comprueba que el modo gráfico especificado existe.

Descripción de los Parámetros:
width = ancho, en pixels (i.e. 640)
height = altura, en pixels (i.e. 480)
depth = profundidad de color (i.e. 16, 24, 32)

Descripción del Comando:
Usa este comando para verificar que la tarjeta gráfica del usuario soporta un modo en concreto. Devuelve TRUE si el modo existe, FALSE si no. Si quieres saber que número de modo es, usa FindGFXMode.

Ejemplo:

```
; GFXModeExists example

; If there is a mode, tell user
mode=GfxModeExists(800,800,16)

If mode=1 Then
Print "The mode you requested exists!"
Else
Print "Sorry, that mode doesn't exist."
End If

; Wait for ESC press from user
While Not KeyHit(1)
Wend
```

TotalVidMem()

Parámetros:
Ninguno.

Descripción:
TotalVidMem () simplemente devuelve el total de memoria disponible en la tarjeta gráfica, en bytes. Fíjate que para saber la memoria disponible actualmente en bytes, deberías usar AvailVidMem ().

Ejemplo:

```
Print "Total de memoria gráfica disponible: " + TotalVidMem () + " bytes."
; NOTA: Para saber la memoria gráfica *disponible*, ¡usa AvailVidMem ()!
```

AvailVidMem()

Definición:
Devuelve la memoria de video disponible.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando devolverá el número total de bytes disponibles de la memoria de video. Úsalo para saber el estado de tus recursos.

Ejemplo:

```
Print "Your available video memory is: " + AvailVidMem()
```

IMAGEN

LoadImage (Filename)

Definición:
Carga una imagen en memoria y le asigna un handle.

Descripción de los Parámetros:
filename = ruta completa y nombre del archivo

Descripción del Comando:
Este comando carga una imagen del disco y le asigna un handle de archivo. Tendrás que usar el comando DrawImage para mostrar el gráfico más tarde. La versión demo de Blitz Basic soporta archivos BMP; la versión completa de Blitz Basic soporta también los formatos JPG y PNG. Muchos comandos multimedia para fuentes, gráficos y sonidos necesitan usar HANDLES DE ARCHIVO. Necesitarás entender bien estos handles si quieres llevarte bien con Blitz Basic :D
Un handle de archivo es una variable (normalmente GLOBAL) que contiene un identificador único del recurso cargado (fuente, imagen, sonido, música, etc.). Este identificador se usa después en las subsecuentes operaciones designadas al recurso cargado. Encontrarás handles de archivo en todo Blitz. Mira el ejemplo para entender mejor todo esto.

Ejemplo:

```
; LoadImage and DrawImage example

; Declare a variable to hold the graphic file handle
Global gfxPlayer

; Set a graphics mode
Graphics 640,480,16

; Set drawing operations for double buffering
SetBuffer BackBuffer()

; Load the image and assign its file handle to the variable
; - This assumes you have a graphic called player.bmp in the
; same folder as this source code
gfxPlayer=LoadImage("player.bmp")

; Let's do a loop where the graphic is drawn wherever the
; mouse is pointing. ESC will exit.
While Not KeyHit(1)
Cls ; clear the screen
DrawImage gfxPlayer,MouseX(),MouseY() ; Draw the image!
Flip ; flip the image into view and clear the back buffer
Wend
```

CopyImage (handle)

Definición:
Duplica una imagen cargada en un nuevo handle.

Descripción de los Parámetros:
handle=la variable que contiene el handle de la imagen cargada

Descripción del Comando:
En vez de cargar un gráfico dos veces en dos handles diferentes, puedes cargarlo UNA SOLA VEZ y usar el comando CopyImage para hacer en memoria tantas copias como quieras.
¿Por qué ibas a querer hacer esto? Así todavía tendrás una copia de la imagen original en caso de que quieras alterar la copia u otra cosa.

Ejemplo:

```
; CopyImage Example

; Load an image and give its handle to gfxOld variable
gfxOld=LoadImage("mypicture.bmp")

; Duplicate the gfxOld image to a new handle variable
gfxNew=CopyImage(gfxOld)
```

CreateImage (width,height[,frames])

Definición:
Crea una nueva imagen en memoria y mete el handle en una variable.

Descripción de los Parámetros:
width=ancho de la nueva imagen (o sus frames)
height=altura de la nueva imagen
frames= numero de frames opcional (se omite si sólo tiene un frame)

Descripción del Comando:
Algunas veces quieres crear una imagen nueva al vuelo (usando otras imágenes, con comandos de dibujo, etc.) en vez de cargar una. Este comando te permitirá crear una imagen nueva con uno o varios frames para una animación. Especificas la anchura, altura y el número de frames opcional. El ejemplo puede ser interesante.

Ejemplo:

```
; CreateImage/TileImage/ImageBuffer example

; Again, we'll use globals even tho we don't need them here
; One variable for the graphic we'll create, one for a timer
Global gfxStarfield, tmrScreen

; Declare graphic mode
Graphics 640,480,16

; Create a blank image that is 320 pixels wide and 32 high with 10 frames of 32x32
gfxStarfield=CreateImage(32,32,10)

; loop through each frame of the graphic we just made
For t = 0 To 9
; Set the drawing buffer to the graphic frame so we can write on it
SetBuffer ImageBuffer(gfxStarfield,t)
; put 50 stars in the frame at random locations
For y = 1 To 50
Plot Rnd(32),Rnd(32)
Next
Next

; Double buffer mode for smooth screen drawing
SetBuffer BackBuffer()

; Loop until ESC is pressed
While Not KeyHit(1)
```

```
; Only update the screen every 300 milliseconds. Change 300 for faster or
; slower screen updates
If MilliSecs() > tmrScreen+300 Then
Cls ; clear the screen

; Tile the screen with a random frame from our new graphic starting at
; x=0 and y=0 location.
TileImage gfxStarfield,0,0,Rnd(9)
Flip ; Flip the screen into view
tmrScreen=MilliSecs() ; reset the time
End If
Wend
```

LoadAnimImage (filename,width,height,first,count)

Definición:

Carga una 'tira de imágenes' para usarla como una imagen animada que será dibujada después y devuelve su handle.

Descripción de los Parámetros:

filename *= ruta completa y nombre del archivo*
width*=ancho en pixels de cada frame de la imagen.* ***height****=alto en pixels de cada frame de la imagen*
first*=frame de inicio (normalmente 0).* ***count****=número de frames*

Descripción del Comando:

Al contrario que LoadImage, el comando LoadAnimImage carga una imagen que está hecha de 'frames', de imágenes separadas. Como en el comando LoadImage, este comando devuelve un handle de archivo. Usa una variable (normalmente GLOBAL) para contener este handle, porque lo necesitarás para DIBUJARLO con el comando DrawImage. Mira el comando LoadImage para más detalles. La tira de imágenes en sí misma consiste en 2 o más frames, alineados horizontalmente en una sola imagen. No hay espacios entre los frames, y cada frame debe tener el mismo ancho y alto. Por ejemplo, mira el archivo kaboom.bmp o sparks.bmp incluidos en el directorio C:\Archivos de Programa\Blitz3D\samples\graphics de tu ordenador. Hay algunas utilidades gratis que pueden ayudarte para hacer esto. Cuando dibujes en la pantalla con el comando DrawImage, especifica qué frame dibujar en el parámetro correspondiente. Actualmente, para hacer una imagen animada, necesitarás hacer un ciclo con las imágenes (como una película) rápidamente para crear la ilusión del movimiento. Nuestro ejemplo te mostrará cómo usar uno de los ejemplos de tiras de imágenes y crear una imagen animada. Por favor, mira el ejemplo para entender mejor el comando. Nota: Puedes necesitar cambiar la localización del archivo según su ubicación en tu ordenador.

Ejemplo:

```
; Ejemplo LoadAnimImage/MaskImage MouseX()/MouseY()
; With animation timers

; Incluso cuando no hay funciones, debemos hacer variables globales
; Una variable tendrá el handle del gráfico, otra
; el frame actual que estamos mostrando y otra el timer para ajustar
; la velocidad de la animacion
Global gfxSparks, frmSparks, tmrSparks

Graphics 640,480,16
SetBuffer BackBuffer()

; Carga la tira de imagenes
gfxSparks=LoadAnimImage("c:\Program Files\BlitzBasic\samples\Graphics\spark.bmp",32,32,0,3)

; Hacemos que el color rosa sea transparente en esta imagen
MaskImage gfxSparks,255,0,255

; Bucle hasta que se pulse ESC
While Not KeyHit(1)
Cls

; En la siguiente linea se controla la velocidad de la animacion, Cambia
; el 100 por otros valores mayores o menores para comprobar la diderencia
If MilliSecs() > tmrSparks + 100 Then
tmrSparks=MilliSecs() ; 'reset' the timer
frmSparks=( frmSparks + 1 ) Mod 3 ; incrementa el frame
End If
DrawImage gfxSparks,MouseX(),MouseY(),frmSparks ; dibuja la imagen en las coordenadas del ratón
Flip ; muestra el buffer
Wend
```

FreeImage handle

Definición:
Elimina la imagen especificada de la memoria.

Descripción de los Parámetros:
handle=handle de la imagen

Descripción del Comando:
Cuando ya has usado una imagen y ya no te es necesaria, usa este comando para borrar la imagen de la memoria y así liberar esa memoria para otro uso.

Ejemplo:

```
; FreeImage command

; Global, as always, for graphics
Global gfxPlayer

; Enter graphics mode and start double buffering
Graphics 640,480,16
SetBuffer BackBuffer()

; Load the image-assign the handle to gfxPlayer
gfxPlayer=LoadImage("player.bmp")

; draw the image at random until ESC pressed
While Not KeyHit(1)
Cls
DrawImage gfxPlayer,Rnd(640),Rnd(480)
Flip
Wend

; Erase the image from memory!
FreeImage gfxPlayer
```

SaveImage (image,bmpfile$,[frame])

Definición:
Guarda una imagen en el disco duro como un archivo .bmp.

Descripción de los Parámetros:
image = handle de la imagen
bmpfile$ = ruta completa y nombre del archivo
frame = opcional; frame de la imagen a guardar

Descripción del Comando:
Guarda una imagen o uno de sus frames en el disco duro. Necesitarás una imagen existente para poder guardarla. Devuelve 1 si el guardado se completó con éxito, o 0 en caso contrario.

Ejemplo:

```
; SaveImage example

; Set Graphics Mode
Graphics 800,600,16

; Load an image to tile (your location might vary)
gfxBall=LoadImage("C:\Program Files\Blitz Basic\samples\ball.bmp")

; Save the image to the c: drive ...
ok=SaveImage (gfxBall,"c:\newball.bmp")

; Print results
If ok=1 Then
Print "Save successful!"
Else
```

```
Print "There was an error saving!"
End If

; Wait for ESC to hit
While Not KeyHit(1)
Wend
```

GrabImage image,x,y,[frame]

Definición:
Coge una porción del actual buffer de dibujo y la guarda en una imagen.

Descripción de los Parámetros:
image = handle de la imagen
x = localización x de inicio para capturar la imagen
y = localización y de inicio para capturar la imagen
frame = frame para meter la imagen (opcional)

Descripción del Comando:
Seguramente uno de los más útiles y confusos comandos en Blitz Basic sea GrabImage.
Este comando te permite grabar una porción del actual buffer de dibujo (esto puede ser una imagen también si has establecido el buffer como ImageBuffer) y lo mete en una imagen. Hay mil y un usos para este comando, sólo necesitas imaginación. Primero debes usar el comando CreateImage para crear una nueva imagen vacía en la que grabarás. Su tamaño debe ser igual o mayor al de la imagen a capturar. Por ejemplo, creas una nueva imagen con un tamaño de 50 pixels por 50 pixels. Cuando llamas a GrabImage, eliges x=100, y=100. El área que grabarás y pondrás en tu nueva imagen será de 100,100 a 150,150. Si intentas usar GrabImage en una imagen que no existe, recibirás un error. Nota: Puedes REMPLAZAR una imagen existente. Mira el ejemplo para más información.

Ejemplo:

```
; GrabImage example
; Turn on graphics
Graphics 640,480,16

; We'll be drawing right to the front buffer
SetBuffer FrontBuffer()

; You must create an empty image to grab from first
gfxGrab=CreateImage(100,100)

; Draw random rectangles on the screen until the
; user presses ESC
While Not KeyHit(1)
; random color
Color Rnd(255),Rnd(255),Rnd(255)
; super random rectangles
Rect Rnd(640),Rnd(480),Rnd(100),Rnd(100),Rnd(1)
Delay 50
Wend

; Now, grab an image, starting at 100,100 and put it in gfxGrab
GrabImage gfxGrab,100,100

; Clear screen and show user the grabbed image
Cls
Text 0,0, "Here is your grabbed image! Press a mouse key ..."
DrawImage gfxgrab,50,50

; Wait for a mouse press
WaitMouse()
```

ImageBuffer (handle[,frame])

Definición:
Se usa para especificar una imagen y un frame para realizar operaciones de dibujo directamente.
Descripción de los Parámetros:
handle=handle de la imagen
frame=número de frame (opcional)

Descripción del Comando:
Hay 1000 usos para este comando. Puedes dibujar en una imagen que hayas cargado (LoadImage o LoadAnimImage) o creado (CreateImage). Podrías, por ejemplo, tener un gráfico de un muro y quieres añadirle un 'graffiti' basado en la acción del usuario. En vez de intentar dibujar una docena de imágenes encima del muro, sólo usa SetBuffer para especificar el gráfico del muro como buffer 'objetivo', ¡y dibuja encima! La próxima vez que dibujes este gráfico (DrawImage), verás los cambios. ¡Es un comando muy poderoso!

Ejemplo:

```
; CreateImage/TileImage/ImageBuffer example

; Again, we'll use globals even tho we don't need them here
; One variable for the graphic we'll create, one for a timer
Global gfxStarfield, tmrScreen

; Declare graphic mode
Graphics 640,480,16

; Create a blank image that is 320 pixels wide and 32 high with 10 frames of 32x32
gfxStarfield=CreateImage(32,32,10)

; loop through each frame of the graphic we just made
For t = 0 To 9
; Set the drawing buffer to the graphic frame so we can write on it
SetBuffer ImageBuffer(gfxStarfield,t)
; put 50 stars in the frame at random locations
For y = 1 To 50
Plot Rnd(32),Rnd(32)
Next
Next

; Double buffer mode for smooth screen drawing
SetBuffer BackBuffer()

; Loop until ESC is pressed
While Not KeyHit(1)

; Only update the screen every 300 milliseconds. Change 300 for faster or
; slower screen updates
If MilliSecs() > tmrScreen+300 Then
Cls ; clear the screen

; Tile the screen with a random frame from our new graphic starting at
; x=0 and y=0 location.
TileImage gfxStarfield,0,0,Rnd(9)
Flip ; Flip the screen into view
tmrScreen=MilliSecs() ; reset the time
End If
Wend
```

DrawImage handle,x,y,[frame]

Definición:
Dibuja una imagen o frame de una imagen animada en la pantalla.

Descripción de los Parámetros:
handle *= handle de la imagen.* ***x*** *= localización x de la pantalla donde se dibujará el gráfico*
y *= localización y de la pantalla donde se dibujará el gráfico.* ***frame*** *= número del frame a mostrar (opcional - 0 por defecto)*

Descripción del Comando:
Este comando dibuja una imagen previamente cargada. Este comando muestra tanto imágenes simples (cargadas con el comando LoadImage) como imágenes animadas (cargadas con el comando LoadAnimImage). Tú especificas en qué coordenadas de la pantalla deseas que aparezca la imagen. Puedes dibujarla también fuera de la pantalla especificando valores negativos o positivos (mayores que el tamaño de la pantalla). Finalmente, si estás usando una imagen animada (cargada con el comando LoadAnimImage), puedes especificar qué frame quieres que se muestre con el comando DrawImage. Uno de los problemas más comunes con los nuevos programadores Blitz ocurre cuando usando los comandos de dibujo ven a que el gráfico no aparece en la pantalla, ¡y nunca lo verán! Recuerda, Blitz Basic es rápido... más rápido que el ojo humano. Tendrás que redibujar la imagen encima una y otra vez (parecido a los dibujos animados de la TV), limpiando la pantalla cada vez que haces un cambio (esto es llamado doble buffering); o necesitarás retrasar la ejecución de Blitz durante mucho "tiempo humano" para poder VER la imagen.

Ejemplo:

```
; LoadImage and DrawImage example

; Declare a variable to hold the graphic file handle
Global gfxPlayer

; Set a graphics mode
Graphics 640,480,16

; Set drawing operations for double buffering
SetBuffer BackBuffer()

; Load the image and assign its file handle to the variable
; - This assumes you have a graphic called player.bmp in the
; same folder as this source code
gfxPlayer=LoadImage("player.bmp")

; Let's do a loop where the graphic is drawn wherever the
; mouse is pointing. ESC will exit.
While Not KeyHit(1)
Cls ; clear the screen
DrawImage gfxPlayer,MouseX(),MouseY() ; Draw the image!
Flip ; flip the image into view and clear the back buffer
Wend
```

DrawImageRect image,x,y,rect_x,rect_y,rect_width,rect_height,[frame]

Definición:
Dibuja un área rectangular de una imagen en la localización especificada.

Descripción de los Parámetros:
image = handle de la imagen
x = localización x de la pantalla para dibujar la imagen
y = localización y de la pantalla para dibujar la imagen
rect_x = localización x del interior de la imagen para empezar a dibujar
rect_y = localización y del interior de la imagen para empezar a dibujar
rect_width = ancho del área a dibujar
rect_height = altura del área a dibujar
frame = número de frame de la imagen a dibujar (opcional)

Descripción del Comando:
Este comando te permitirá dibujar una PORCIÓN rectangular de una imagen en la posición especificada de la pantalla. Las partes transparentes de la imagen original no se dibujarán. Esto es útil si estás haciendo algo parecido a revelar parte de la pantalla cuando el jugador descubra algo. Si quieres dibujar la porción de la imagen sin transparencias, usa el comando DrawBlockRect.

Ejemplo:

```
; DrawImageRect Example

; Turn on graphics mode
Graphics 640,480,16

; Create new empty graphic to store our circle in
gfxCircle=CreateImage(50,50)

; Draw the circle image
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxCircle)
Color 255,0,0
; Note the extra space between the circle and the edge of the graphic
Oval 10,10,30,30,1
SetBuffer FrontBuffer()


; Let's draw portions of the circle randomly
While Not KeyHit(1)
```

```
; We take random sized portions of the circle and put them
; at a random location ... wash, rinse, and repeat
DrawImageRect gfxCircle,Rnd(640),Rnd(480),0,0,Rnd(50),Rnd(50),0
Delay 100
Wend
```

DrawBlockRect image,x,y,rect_x,rect_y,rect_width,rect_height,[frame]

Definición:
Dibuja un área rectangular de una imagen en la localización especificada sin transparencia.

Descripción de los Parámetros:
image = handle de la imagen
x = localización x para dibujar la imagen
y = localización x para dibujar la imagen
rect_x = localización x de inicio dentro de la imagen a dibujar
rect_y = localización y de inicio dentro de la imagen a dibujar
rect_width = altura del área a dibujar
rect_height = ancho del área a dibujar
frame = frame de la imagen a dibujar (opcional)

Descripción del Comando:
Este comando te permitirá dibujar una PORCIÓN rectangular de una imagen en la localización especificada de la pantalla. Las porciones transparentes de la imagen original serán ignoradas y dibujadas con la imagen.
Si quieres dibujar una porción de imagen con transparencia, usa el comando DrawImageRect.

Ejemplo:

```
; DrawBlockRect Example

; Turn on graphics mode
Graphics 640,480,16

; Create new empty graphic to store our circle in
gfxCircle=CreateImage(50,50)

; Draw the circle image
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxCircle)
Color 255,0,0
; Note the extra space between the circle and the edge of the graphic
Oval 10,10,30,30,1
SetBuffer FrontBuffer()

; Set the screen to white so you can see the transparent areas
ClsColor 255,255,255
Cls

; Let's draw portions of the circle randomly
While Not KeyHit(1)
; We take random sized portions of the circle and put them
; at a random location ... wash, rinse, and repeat
DrawBlockRect gfxCircle,Rnd(640),Rnd(480),0,0,Rnd(50),Rnd(50),0
Delay 100
Wend
```

DrawBlock image,x,y,[frame]

Definición:
Dibuja una imagen en una localización sin respetar la transparencia.

Descripción de los Parámetros:
image = handle de la imagen
x = localización x para dibujar la imagen
y = localización x para dibujar la imagen
frame = frame de la imagen a dibujar (opcional - por defecto 0)

Descripción del Comando:
Éste es similar al comando DrawImage excepto que la transparencia o el MaskImage es ignorado y se dibuja la imagen entera. El frame es opcional.

Ejemplo:

```
; DrawBlock Example

; Turn on graphics mode
Graphics 640,480,16

; Create new empty graphic to store our circle in
gfxCircle=CreateImage(50,50)

; Draw the circle image
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxCircle)
Color 255,0,0
; Note the extra space between the circle and the edge of the graphic
Oval 10,10,30,30,1

; Let's not forget to put the drawing buffer back!
SetBuffer BackBuffer()
; Set the CLS color to white
ClsColor 255,255,255

; Let the user move the circle graphic around a white screen
; putting the graphic at the MouseX,Y coordinates
While Not KeyHit(1)
Cls
DrawBlock gfxCircle,MouseX(),MouseY()
Flip
Wend
```

TileImage handle,[x],[y],[frames]

Definición:
Crea un mosaico en la pantalla con una imagen.

Descripción de los Parámetros:
image = handel de la imagen
x = coordenada x (opcional)
y = coordenada y (opcional)
frame = frame de la imagen (opcional)

Descripción del Comando:
Si quieres crear un campo de estrellas o algún otro tipo de fondo hecho con mosaicos, este es TU comando. Todo lo que tienes que hacer es especificar el handle de la imagen (una imagen cargada con el comando LoadImage o LoadAnimImage). De forma opcional, puedes especificar una localización de comienzo así como el frame de la imagen.

Ejemplo:
Ver ImageBuffer

TileBlock image [,x,y,frame]

Definición:
Crea un mosaico en la pantalla con una imagen ignorando las transparencias.

Descripción de los Parámetros:
image = handel de la imagen
x = coordenada x (opcional)
y = coordenada y (opcional)
frame = frame de la imagen (opcional)

Descripción del Comando:
Similar a TileImage pero ignora las transparencias. Úsalo para crear un mosaico en una parte o toda la pantalla usando una imagen.

Ejemplo:

```
; Ejemplo TileBlock
Graphics 800,600,16

; Carga una imagen (su localización puede cambiar en tu ordenador, en el mío ni siquiera existe,
jejeje)
gfxBall=LoadImage("C:\Archivos de Programa\Blitz3D\samples\ball.bmp")

; Crea el mosaico sin transparencia
TileBlock gfxBall

; Espera que pulses Esc
While Not KeyHit(1)
Wend
```

MaskImage handle,red,green,blue

Definición:
Establece una máscara de imagen o color transparente.

Descripción de los Parámetros:
handle=handle de la imagen
red=valor rojo (0-255)
green=valor verde (0-255)
blue=valor azul (0-255)

Descripción del Comando:
Blitz Basic asume que cuando cargas una imagen (usando LoadImage o LoadAnimImage) para dibujarla (usando DrawImage), quieres que el color negro (RGB 0,0,0) de tu imagen sea transparente. Llegará un momento en que quieras que algún otro color sea transparente. Este comando te permite hacerlo usando un valor RGB (yo uso Paint Shop Pro para saber los valores de rojo, verde y azul). Mira el ejemplo para comprender mejor este comandos.

Ejemplo:
Ver Ejemplo LoadAnimImage

HandleImage image,x,y

Definición:
Establece una nueva localización de handle a una imagen existente.

Descripción de los Parámetros:
image = handle de la imagen
x = localización x del nuevo handle de imagen
y = localización y del nuevo handle de imagen

Descripción del Comando:
Cuando una imagen es cargada con LoadImage, el handle de la imagen (la localización en el interior de la imagen desde donde será dibujada) está siempre por defecto en la esquina superior izquierda (coordenadas 0,0). Esto significa que si tú dibujas una imagen de 50x50 pixels en las coordenadas 200,200 de la pantalla, la imagen comenzará a ser dibujada en 200,200 hasta acabar en 250,250. Este comando mueve el handle de la imagen de 0,0 a la coordenada especificada. Puedes recuperar las coordenadas del handle actual con los comandos ImageXHandle y ImageYHandle. Finalmente, puedes hacer que todas las imágenes se carguen automáticamente con el handle en el punto medio usando el comando AutoMidHandle. Nota acerca del término 'handle'. Hay dos tipos de 'handles' que usaremos en estos documentos. Uno es la localización en el interior de una imagen (al que se hace referencia en este comando). El otro es un 'handle de archivo', una variable usada para mantener una imagen, sonido o fuente cargada con un comando. Mira LoadImage para más información acerca de los handles de archivo. Mira también: MidHandle

Ejemplo:

```
;HandleImage Example
Graphics 800,600,16

gfxPlayer=LoadImage("player.bmp")
HandleImage gfxPlayer,20,20
DrawImage gfxPlayer,0,0
WaitKey
```

MidHandle image

Definición:
Establece el handle de imagen de la imagen especificada en el centro de la misma.

Descripción de los Parámetros:
image = handle de archivo de la imagen

Descripción del Comando:
Cuando una imagen es cargada con LoadImage, su handle (la localización dentro de la imagen desde donde la imagen es dibujada) siempre está por defecto en la esquina superior izquierda (coordenadas 0,0). Esto significa que si tu dibujas una imagen de 50x50 pixels en las coordenadas 200,200, la imagen comenzará a ser dibujada en las coordenadas 200,200 y se extenderá hasta 250,250. Este comando mueve le handle de imagen al punto medio exactamente de la imagen. Puedes establecer manualmente la localización del handle usando el comando HandleImage. Puedes obtener la posición del handle usando los comandos ImageXHandle y ImageYHandle. Finalmente, puedes hacer que todas las imágenes carguen automáticamente con el handle en el centro usando el comando AutoMidHandle. Nota acerca del término 'handle'. Hay dos tipos de 'handles' que veremos en estos documentos. Uno está localizado en el interior de una imagen (como vemos en este momento). El otro es un 'handle de fichero', una variable usada para manejar una imagen, un sonido o una fuente cargada con un comando. Mira LoadImage para más información acerca de los 'handles de ficheros'.

Ejemplo:

```
; MidHandle/ImageXHandle()/ImageYHandle()/AutoMidHandle

; Initiate Graphics Mode
Graphics 640,480,16

; Set up the image file handle as a global
Global gfxBall

; Load the image - you may need to change the location of the file
gfxBall=LoadImage ("C:\Program Files\Blitz Basic\samples\ball.bmp")

; Until the user presses ESC key ...
While Not KeyHit(1)
Text 0,0,"Default Image Handle for gfxBall... Press ESC ..."
Text 0,14,"X handle-" + ImageXHandle(gfxBall) ; Print the location of the image handle x location
Text 0,28,"Y handle-" + ImageYHandle(gfxBall) ; Print the location of the image handle y location
DrawImage gfxBall,200,200,0 ; draw the image at 200,200
Wend

; Clear the screen
Cls

; Set the ball's handle to the center of the image
MidHandle gfxBall

; Until the user presses ESC key ... show the new information
While Not KeyHit(1)
Text 0,0,"New Image Handle for gfxBall... Press ESC ..."
Text 0,14,"X handle-" + ImageXHandle(gfxBall)
Text 0,28,"Y handle-" + ImageYHandle(gfxBall)
DrawImage gfxBall,200,200,0
Wend

; Makes all images load up with their handles in the center of the image
AutoMidHandle True
Cls

; Load the image again
gfxBall=LoadImage ("C:\Program Files\Blitz Basic\samples\ball.bmp")

; Until the user presses ESC key ... show the new information
While Not KeyHit(1)
Text 0,0,"Automatic image handle of gfxBall... Press ESC ..."
Text 0,14,"X handle-" + ImageXHandle(gfxBall)
Text 0,28,"Y handle-" + ImageYHandle(gfxBall)
DrawImage gfxBall,200,200,0
Wend
```

AutoMidHandle true/false

Definición:
Automáticamente establece un handle de imagen en el centro de cada imagen cargada.
Descripción de los Parámetros:
true = imágenes cargadas con el midhandle establecido por defecto
false = imágenes cargan con el handle por defecto en 0,0

Descripción del Comando:
Cuando una imagen es cargada con LoadImage, su handle (la localización dentro de la imagen desde donde la imagen es dibujada) siempre está por defecto en la esquina superior izquierda (coordenadas 0,0). Esto significa que si tu dibujas una imagen de 50x50 pixels en las coordenadas 200,200, la imagen comenzará a ser dibujada en las coordenadas 200,200 y se extenderá hasta 250,250. El comando MidHandle mueve el handle de la imagen al punto medio de la imagen. Mira este comando para más información acerca del handle de imagen.
Este comando elimina la necesidad de cambiar la posición del handle en cada nueva imagen cargada, cambiándolo automáticamente cada vez que son cargadas. Nota acerca del término 'handle'. Hay dos tipos de 'handles' que veremos en estos documentos. Uno está localizado en el interior de una imagen (como vemos en este momento). El otro es un 'handle de fichero', una variable usada para manejar una imagen, un sonido o una fuente cargada con un comando. Mira LoadImage para más información acerca de los 'handles de ficheros'.

Ejemplo:
Ver MidHandle

ScaleImage image,xscale#,yscale#

Definición:
Escala una imagen usando porcentajes.

Descripción de los Parámetros:
image = handle de la imagen
xscale# = porcentaje de escala de la anchura
yscale# = porcentaje de escala de la altura

Descripción del Comando:
Usa este comando para reescalar una imagen a un nuevo tamaño mediante porcentajes de coma flotante (1.0 = 100%, 2.0 = 200%, etc). Usando un valor negativo invertirás la imagen. Debes haber cargado la imagen con LoadImage o LoadAnimImage. ***¡¡NO sirve para escalar las imágenes en TIEMPO REAL!!*** *Debes calcular el tamaño de las imágenes antes de ejecutar tu programa o notarás graves ralentizaciones.*

Ejemplo:

```
; ScaleImage example

; Set Graphics Mode
Graphics 800,600,16

; Randomize the random seed
SeedRnd MilliSecs()

; Load an image to tile (your location might vary)
gfxBall=LoadImage("C:\Program Files\Blitz Basic\samples\ball.bmp")

; Scale it randomly from 50% to 150% both x and y
ScaleImage gfxBall,Rnd(-2.0,2.0),Rnd(-2.0,2.0)

; Wait for ESC to hit
While Not KeyHit(1)
DrawImage gfxball,Rnd(800),Rnd(600)
VWait
Wend
```

ResizeImage image,width#,height#

Definición:
Redimensiona una imagen.

Descripción de los Parámetros:
image = handle de la imagen
width# = nuevo ancho en pixels
height# = nueva altura en pixels

Descripción del Comando:
Similar a ScaleImage, pero usa valores en pixels en vez de porcentajes. Usa este comando para redimensionar una imagen cargada previamente con LoadImage o LoadAnimImage. No se recomienda para escalar las imágenes en TIEMPO REAL, precalcula el tamaño de tus imágenes antes de continuar la ejecución de tu programa, o notarás graves ralentizaciones.

Ejemplo:

```
; ResizeImage example

; Set Graphics Mode
Graphics 800,600,16

; Randomize the random seed
SeedRnd MilliSecs()

; Load an image to tile (your location might vary)
gfxBall=LoadImage("C:\Program Files\Blitz Basic\samples\ball.bmp")

; Size it randomly from 300 to -300 both x and y
ResizeImage gfxBall,Rnd(-300,300),Rnd(-300,300)

; Wait for ESC to hit
While Not KeyHit(1)
DrawImage gfxball,Rnd(800),Rnd(600)
VWait
Wend
```

RotateImage image,value#

Definición:
Rota la imagen en el sentido contrario de las agujas del reloj.

Descripción de los Parámetros:
image = handle del archivo
value# = número flotante entre 0 y 360 grados

Descripción del Comando:
Este comando no es lo suficientemente rápido como para ser usado en rotaciones en tiempo real. Entonces, el propósito de este comando es rotar una imagen especificada un número concreto de grados. Este comando automáticamente alisa los bordes de la imagen rotada, y las transparencias serían ignoradas. Para evitar esto, usa el comando TFormFilter. Éste renderizará las imágenes rotadas con un filtro bilinear.

Ejemplo:

```
; RotateImage/TFormFilter Example

; Turn on graphics mode
Graphics 640,480,16

; Change the 0 to a 1 to see the difference
; between filter on and off.
TFormFilter 0

; Create new empty graphic to store our circle in
gfxBox=CreateImage(50,50)

; Draw the box image
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxBox)
Color 255,0,0
; Note the extra space between the box and the edge of the graphic
Rect 10,10,30,30,1
SetBuffer FrontBuffer()

While Not KeyHit(1)
; Make a copy of the image so we are always using a fresh one each time
; we rotate it.
gfxTemp=CopyImage(gfxBox)
```

```
; Rotate it a random value and draw it at a random location
RotateImage gfxTemp,Rnd(360)
DrawImage gfxTemp,Rnd(640),Rnd(480)
Wend
```

TFormImage image,a#,b#,c#,d#

Parámetros:
image - handle de la imagen
a# - elemento 1,1 de una matriz de 2x2
b# - elemento 2,1 de una matriz de 2x2
c# - elemento 1,2 de una matriz de 2x2
d# - elemento 2,2 de una matriz de 2x2

Descripción:
Transforma una imagen basada en una matriz de 2x2. El único uso real para este comando que no está disponible vía ScaleImage o RotateImage es que puedes usarlo para cortar una imagen.

TFormFilter enable

Definición:
Activa el filtro bilinear en imágenes retorcidas.

Descripción de los Parámetros:
enable = 0 para desactivar el filtro; 1 para activarlo

Descripción del Comando:
Este comando activa/desactiva el filtro bilinear en imágenes que son retorcidas (alteradas) por comandos como TFormImage y RotateImage. Este filtro permite a este tipo de gráficos tener suavizado, las aristas más alisadas. Esto también ralentiza las operaciones. El filtro bilinearpuede también crear aristas no transparentes. Experimenta para obtener mejores resultados. Prueba cambiando el ejemplo para ver la diferencia.

Ejemplo:
Ver Ejemplo RotateImage

ImageWidth (image handle)

Definición:
Devuelve el ancho de la imagen especificada, en pixels.

Descripción de los Parámetros:
image handle = handle de la imagen

Descripción del Comando:
Usa este comando con ImageHeight para saber el tamaño de la imagen (usando el handle obtenido cuando la imagen fue cargada con LoadImage) en pixels.

Ejemplo:

```
; ImageHeight/ImageWidth Example
; Global, as always, for graphics
Global gfxPlayer

; Enter graphics mode and start double buffering
Graphics 640,480,16
SetBuffer BackBuffer()

; Load the image-assign the handle to gfxPlayer
gfxPlayer=LoadImage("player.bmp")

; Print the image dimensions
Print "The image height is: " + ImageHeight(gfxPlayer)
Print "The image width is: " + ImageWidth(gfxPlayer)

; Wait until ESC is pressed so you can see the output
While Not KeyHit(1)
Wend
```

ImageHeight (image handle)

Definición:
Devuelve la altura de la imagen especificada, en pixels.

Descripción de los Parámetros:
image handle = handle de la imagen

Descripción del Comando:
Usa este comando con ImageWidth para saber el tamaño de la imagen (usando el handle obtenido cuando la imagen fue cargada con LoadImage) en pixels.

Ejemplo:
Ver ImageWidth

ImageXHandle image

Definición:
Devuelve la localización X del handle de la imagen especificada.

Descripción de los Parámetros:
image = handle de la imagen

Descripción del Comando:
A veces es útil saber la localización del handle de una imagen. Este comando devuelve la coordenada X. Usa ImageYHandle para obtener la coordenada Y. Por favor, mira MidHandle para más información sobre los handles de imagen.
Nota acerca del término 'handle'. Hay dos tipos de 'handles' que usaremos en estos documentos. Uno es la localización en el interior de una imagen (al que se hace referencia en este comando). El otro es un 'handle de archivo', una variable usada para mantener una imagen, sonido o fuente cargada con un comando. Mira LoadImage para más información acerca de los handles de archivo.

Ejemplo:
Ver MidHandle

ImageYHandle image

Definición:
Devuelve la localización Y del handle de la imagen especificada.

Descripción de los Parámetros:
image = handle de la imagen

Descripción del Comando:
A veces es útil saber la localización del handle de una imagen. Este comando devuelve la coordenada Y. Usa ImageXHandle para obtener la coordenada X. Por favor, mira MidHandle para más información sobre los handles de imagen. Nota acerca del término 'handle'. Hay dos tipos de 'handles' que usaremos en estos documentos. Uno es la localización en el interior de una imagen (al que se hace referencia en este comando). El otro es un 'handle de archivo', una variable usada para mantener una imagen, sonido o fuente cargada con un comando. Mira LoadImage para más información acerca de los handles de archivo.

Ejemplo:
Ver MidHandle

ImagesOverlap (image1,x1,y1,image2,x2,y2)

Definición:
Comprueba si dos gráficos se han superpuesto.

Descripción de los Parámetros:
image1 = handle de la primera imagen
x1 = localización x de la primera imagen
y1 = localización y de la primera imagen
image2 = handle de la segunda imagen
x2 = localización x de la segunda imagen
y2 = localización y de la segunda imagen

Descripción del Comando:
Éste es muy rápido, te permitirá saber si una de las dos imágenes está superpuesta a la otra. No se tienen en cuenta los pixels transparentes (mira ImagesCollide). Como todo sistema de detección de colisión en Blitz, necesitarás los handles de las dos imágenes y sus coordenadas X e Y en el momento en que ocurrirá la colisión. Usa este comando con imágenes con formas muy regulares, no con formas extrañas.
El ejemplo usa gráficos que son mucho más pequeños que su contenedor para mostrarte la imprecisión de este comando (si no se usa adecuadamente). El ejemplo de ImagesCollide es idéntico a este (y muestra como trabaja la colisión por pixel perfecta).

Ejemplo:

```
; ImagesOverlap Example

; Turn on graphics mode
Graphics 640,480,16

; Create two new empty graphics to store our circle and box in
gfxBox=CreateImage(50,50)
gfxCircle=CreateImage(50,50)

; Draw the box image first
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxBox)
; Change drawing color to blue
Color 0,0,255
;Draw our box (note that it has a 10 pixel space around it)
Rect 10,10,30,30,1

; Repeat for the circle graphic
SetBuffer ImageBuffer(gfxCircle)
Color 255,0,0
; Note the extra space between the circle and the edge of the graphic
Oval 10,10,30,30,1

; Let's not forget to put the drawing buffer back!
SetBuffer BackBuffer()

; Locate our box to a random, visible screen location
boxX=Rnd(50,610)
boxY=Rnd(50,430)

; Repeat the loop until we've had a collision
Repeat
; Attach our mouse to the circle to move it
circleX=MouseX()
circleY=MouseY()
; Standard double buffer technique; clear screen first
Cls
; Draw our objects at the designated location
DrawImage gfxBox,boxX,boxY
DrawImage gfxCircle,circleX,circleY
; Standard double buffer technique; flip after all drawing is done
Flip
; We test the locations of our box and circle to see if they have overlapped
Until ImagesOverlap (gfxBox,boxX,boxY,gfxCircle,circleX,circleY)

; Loop is over, we must've collided!
Text 0,0, "WE'VE HAD A COLLISION! PRESS A MOUSE BUTTON"
; Can't see the text until we flip ..
Flip
; Wait for a mouse click
WaitMouse()
; End our graphics mode
EndGraphics
```

ImagesCollide (image1,x1,y1,frame1,image2,x2,y2,frame2)

Definición:
Comprueba si dos imágenes han colisionado.

Descripción de los Parámetros:

image1 = handle de la primera imagen
x1 = localización x de la primera imagen
y1 = localización y de la primera imagen
frame1 = frame de la primera imagen (opcional)
image2 = handle de la segunda imagen
x2 = localización x de la segunda imagen
y2 = localización y de la segunda imagen
frame2 = frame de la segunda imagen (opcional)

Descripción del Comando:

Este es EL COMANDO para obtener colisiones por pixel perfectas entre imágenes. No tendrá en cuenta los pixels transparentes durante la comprobación (básicamente, sólo la 'chicha' de la imagen producirá la colisión). Esto lo hace perfecto para la mayoría de las situaciones donde tengas figuras raras. El comando ImagesOverlap es MUY MUY rápido, sin embargo, sólo puedes determinar si LGUNA de las dos imágenes se ha superpuesto (esto incluye los pixels transparentes). Este método trabaja si tú tienes gráficos que están cubiertos completamente y no necesitas tanta precisión. Como todo sistema de detección de colisión en Blitz, necesitarás los handles de las dos imágenes y sus coordenadas X e Y en el momento en que ocurrirá la colisión. El ejemplo usa gráficos que son mucho más pequeños que su contenedor para mostrarte la precisión de este comando. El ejemplo de ImagesOverlap es idéntico a este (y muestra la imprecisión del método del 'overlapping').

Ejemplo:

```
; ImagesCollide Example

; Turn on graphics mode
Graphics 640,480,16

; Create two new empty graphics to store our circle and box in
gfxBox=CreateImage(50,50)
gfxCircle=CreateImage(50,50)

; Draw the box image first
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxBox)
; Change drawing color to blue
Color 0,0,255
;Draw our box (note that it has a 10 pixel space around it)
Rect 10,10,30,30,1

; Repeat for the circle graphic
SetBuffer ImageBuffer(gfxCircle)
Color 255,0,0
; Note the extra space between the circle and the edge of the graphic
Oval 10,10,30,30,1

; Let's not forget to put the drawing buffer back!
SetBuffer BackBuffer()

; Locate our box to a random, visible screen location
boxX=Rnd(50,610)
boxY=Rnd(50,430)

; Repeat the loop until we've had a collision
Repeat
; Attach our mouse to the circle to move it
circleX=MouseX()
circleY=MouseY()
; Standard double buffer technique; clear screen first
Cls
; Draw our objects at the designated location
DrawImage gfxBox,boxX,boxY
DrawImage gfxCircle,circleX,circleY
; Standard double buffer technique; flip after all drawing is done
Flip
; We test the locations of our box and circle to see if they have pixel collided
Until ImagesCollide (gfxBox,boxX,boxY,0,gfxCircle,circleX,circleY,0)

; Loop is over, we must've collided!
```

```
Text 0,0, "WE'VE HAD A COLLISION! PRESS A MOUSE BUTTON"
; Can't see the text until we flip ..
Flip
; Wait for a mouse click
WaitMouse()
; End our graphics mode
EndGraphics
```

RectsOverlap (rect1 X,rect1 Y,rect1 Width,rect1 Height,rect2 X,rect2 Y,rect2 Width,rect2 Height)

Definición:

Comprueba si dos áreas rectangulares se están superponiendo.

Descripción de los Parámetros:

rect1 X = posición x del rectángulo 1
rect1 Y = posición y del rectángulo 1
rect1 Width = ancho del rectángulo 1
rect1 Height = alto del rectángulo 1
rect2 X = posición x del rectángulo 2
rect2 Y = posición y del rectángulo 2
rect2 Width = ancho del rectángulo 2
rect2 Height = alto del rectángulo 2

Descripción del Comando:

Este comando cogerá dos áreas rectangulares de la pantalla y verá si se están superponiendo. Necesitarás conocer sus coordenadas, sus anchos y altos para hacerlo. Todavía estoy intentando encontrar un buen uso lógico de este comando con otros comandos de colisión disponibles ImagesOverlap, ImagesCollide, ImageRectOverlap y ImageRectCollide. Al contrario que otros comandos de colisión, no hay imagen con la que detectar la colisión, simplemente un área rectangular superpuesta a otra. Podrías probablemente usar este comando en vez de ImageRectOverlap, como básicamente hacen lo mismo (y creo que éste es más rápido).
Podría ser muy útil para hacer juegos del estilo de 'Monkey Island'.

Ejemplo:

```
; RectsOverlap Example
; Flashing graphics warning! Gets hypnotic ...

; Turn on graphics mode
Graphics 640,480,16

; Double buffering, and randomize the randomizer
SetBuffer BackBuffer()
SeedRnd MilliSecs()

; Repeat the loop until ESC pressed
While Not KeyHit(1)

; Generate a random rectangle
rect1X=Rnd(50,610)
rect1Y=Rnd(50,430)
rect1W=20
rect1H=20

; And another
rect2X=Rnd(50,610)
rect2Y=Rnd(50,430)
rect2W=20
rect2H=20
; Clear the screen standard double buffering
Cls
; Draw our rectangle2 in random colors
Color Rnd(255),Rnd(255),Rnd(255)
Rect rect1X,rect1Y,rect1W,rect1H,0
Color Rnd(255),Rnd(255),Rnd(255)
Rect rect2X,rect2Y,rect2W,rect2H,0

; Did they collide? If so, print a message and exit the loop!
If RectsOverlap (rect1X,rect1Y,rect1W,rect1H,rect2X,rect2Y,rect2W,rect2H) Then
```

```
Text 0,0, "Our boxes finally collided! Press a mouse button..."
; We do a flip here to ensure the text message gets seen too!
Flip
Exit ; exit the While/Wend loop
End If
; Flip the rects into view, wait 1/10th of a sec, repeat
Flip
Delay 100
Wend
; Wait for a mouse click
WaitMouse()
; End our graphics mode
EndGraphics
```

ImageRectOverlap (image,x,y,rect x,rect y,rect width,rect height)

Definición:

Comprueba que una imagen y un área rectangular de la pantalla están superpuestos.

Descripción de los Parámetros:

image = handle de la imagen
x = localización x de la imagen
y = localización y de la imagen
rect x = localización x de comienzo del rectángulo a comprobar
rect y = localización y de comienzo del rectángulo a comprobar
rect width = ancho del rectángulo
rect height = altura del rectángulo

Descripción del Comando:

Hay muchas veces que necesitas saber si una imagen ha colisionado (o está tocando) un área específica de la pantalla. Este comando realiza una detección de colisión entre la imagen que has elegido y un rectángulo especificado de la pantalla. Los pixels transparentes son ignorados durante la comprobación, haciendo este comando muy impreciso con formas extrañas. Mira ImageRectCollide para colisiones por pixel perfectas entre una imagen y un área rectangular de la pantalla. La mayor utilidad de este comando viene cuando quieres hacer un juego del estilo Monkey Island (cuando tienes de fondo una habitación con objetos con los que el jugador puede interactuar con el uso del puntero del ratón). En algunos casos, los objetos de la pantalla actuarán por separado (a menudo animados o se mueven). En esta situación, podrías usar mejor ImagesCollide o ImagesOverlap para detectar la colisión entre el puntero y la imagen. Sin embargo, tu programa sólo necesita detectar un gráfico (como el puntero del ratón) sobre una localización/región de la pantalla en particular (a menudo llamada 'hot spot'). Como en toda colisión en Blitz, necesitarás saber la localización PRECISA del gráfico que desesas saber si ha colisionado, es decir, la x, y, ancho y alto del área de la pantalla que comprobarás. Este ejemplo usa gráficos que son mucho más pequeños que su contenedor para mostrarte como es de impreciso este comando (si no lo usas adecuadamente). El ejemplo de ImageRectCollide es idéntico a este (y muestra la precisión del método por pixel perfecto).

Ejemplo:

```
; ImageRectOverlap Example

; Turn on graphics mode
Graphics 640,480,16

; Create new empty graphic to store our circle in
gfxCircle=CreateImage(50,50)

; Draw the circle image
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxCircle)
Color 255,0,0
; Note the extra space between the circle and the edge of the graphic
Oval 10,10,30,30,1

; Let's not forget to put the drawing buffer back!
SetBuffer BackBuffer()
Color 0,0,255

; Locate our box to a random, visible screen location
hotX=Rnd(50,610)
hotY=Rnd(50,430)
hotW=Rnd(20,100)
hotH=Rnd(20,100)
```

```
; Repeat the loop until we've had a collision
Repeat
; Attach our mouse to the circle to move it
circleX=MouseX()
circleY=MouseY()
; Standard double buffer technique; clear screen first
Cls
; Draw our rectangle
Rect hotX,hotY,hotW,hotH,0
DrawImage gfxCircle,circleX,circleY
; Standard double buffer technique; flip after all drawing is done
Flip
; We test the locations of our rectangle area and circle to see if they have overlapped
Until ImageRectOverlap (gfxCircle,circleX,circleY,hotX,hotY,hotW,hotH)

; Loop is over, we must've collided!
Text 0,0, "WE'VE HAD A COLLISION! PRESS A MOUSE BUTTON"
; Can't see the text until we flip ..
Flip
; Wait for a mouse click
WaitMouse()
; End our graphics mode
EndGraphics
```

ImageRectCollide (image,x,y,frame,rect x,rect y,rect width,rect height)

Definición:

Comprueba si la imagen y el área rectangular de la pantalla designada han colisionado.

Descripción de los Parámetros:

image = handle de la imagen
x = localización x de la imagen
y = localización y de la imagen
frame = frame de la imagen
rect x = localización x de comienzo del rectángulo a comprobar
rect y = localización y de comienzo del rectángulo a comprobar
rect width = ancho del rectángulo
rect height = altura del rectángulo

Descripción del Comando:

Hay muchas veces que necesitas saber si una imagen ha colisionado (o está tocando) un área específica de la pantalla. Este comando realiza una detección de colisión por pixel perfecta entre la imagen que has elegido y el rectángulo especificado en la pantalla. La mayor utilidad de este comando viene cuando quieres hacer un juego del estilo Monkey Island (cuando tienes de fondo una habitación con objetos con los que el jugador puede interactuar con el uso del puntero del ratón). En algunos casos, los objetos de la pantalla actuarán por separado (a menudo animados o se mueven). En esta situación, podrías usar mejor ImagesCollide o ImagesOverlap para detectar la colisión entre el puntero y la imagen. Sin embargo, tu programa sólo necesita detectar un gráfico (como el puntero del ratón) sobre una localización/región de la pantalla en particular (a menudo llamada 'hot spot'). Como en toda colisión en Blitz, necesitarás saber la localización PRECISA del gráfico que desesas saber si ha colisionado, es decir, la x, y, ancho y alto del área de la pantalla que comprobarás. El ejemplo usa gráficos que son mucho más pequeños que su contenedor para mostrarte la precisión de este comando. El ejemplo de ImageRectOverlap es idéntico a este (y muestra como es de impreciso el método de 'overlapping').

Ejemplo:

```
; ImageRectCollide Example

; Turn on graphics mode
Graphics 640,480,16

; Create new empty graphic to store our circle in
gfxCircle=CreateImage(50,50)

; Draw the circle image
; Set drawing operations to point to our new empty graphic
SetBuffer ImageBuffer(gfxCircle)
Color 255,0,0
; Note the extra space between the circle and the edge of the graphic
Oval 10,10,30,30,1
```

```
; Let's not forget to put the drawing buffer back!
SetBuffer BackBuffer()
Color 0,0,255

; Locate our box to a random, visible screen location
hotX=Rnd(50,610)
hotY=Rnd(50,430)
hotW=Rnd(20,100)
hotH=Rnd(20,100)

; Repeat the loop until we've had a collision
Repeat
; Attach our mouse to the circle to move it
circleX=MouseX()
circleY=MouseY()
; Standard double buffer technique; clear screen first
Cls
; Draw our rectangle
Rect hotX,hotY,hotW,hotH,0
DrawImage gfxCircle,circleX,circleY
; Standard double buffer technique; flip after all drawing is done
Flip
; We test the locations of our rectangle area and circle to see if they have pixel collided
Until ImageRectCollide (gfxCircle,circleX,circleY,0,hotX,hotY,hotW,hotH)

; Loop is over, we must've collided!
Text 0,0, "WE'VE HAD A COLLISION! PRESS A MOUSE BUTTON"
; Can't see the text until we flip ..
Flip
; Wait for a mouse click
WaitMouse()
; End our graphics mode
EndGraphics
```

TIEMPO

Millisecs()

Definición:
Devuelve el tiempo del sistema en milésimas de segundo.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Este comando devuelve el tiempo del sistema en milésimas de segundo. Es increíblemente útil para eventos que ocurren en momentos precisos. Un uso frecuente de este comando es crear una semilla de números aleatorios con el comando SeedRnd.

Ejemplo:

```
; This prints STILL WAITING! for three seconds then ends.
oldTime=MilliSecs()
While MilliSecs() < oldTime + 3000
Print "Still waiting!"
Wend
```

Delay milliseconds

Definición:
Para la ejecución del programa durante un cierto tiempo.

Descripción de los Parámetros:
milliseconds = la cantidad de milésimas de segundo que se parará el programa. 1000=1 segundo

Descripción del Comando:
Este comando para toda la ejecución del programa durante el tiempo especificado. TODA ejecución parará.

Ejemplo:

```
; Delay for 5 seconds

Delay 5000
```

CurrentDate$()

Definición:
Devuelve la actual fecha del sistema.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Devuelve la actual fecha del sistema en el formato: DD MM AAAA (p. e. 07 DIC 1987 - mi fecha de nacimiento X-P).

Ejemplo:

```
; Print the current date to the screen

Print "The date is:" + CurrentDate$()
```

CurrentTime$()

Definición:
Devuelve la hora actual del sistema.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Devuelve la hora actual del sistema en el formato: HH:MM:SS (p.e. 14:31:57).

Ejemplo:

```
; Print the current time to the screen

Print "The Time is:" + CurrentTime$()
```

CreateTimer (frequency)

Definición:
Crea un cronómetro para seguir una frecuencia.

Descripción de los Parámetros:
frequency = normalmente un framerate de 50 o 60

Descripción del Comando:
Usa este comando en conjunción con el comando WaitTimer para controlar la velocidad de un programa en ejecución (fps).

Ejemplo:

```
; Create the timer to track speed
frameTimer=CreateTimer(60)

; Your main screen draw loop
While Not KeyHit(1)
WaitTimer(frameTimer) ; Pause until the timer reaches 60
Cls
; Draw your screen stuff
Flip
Wend
```

FreeTimer (timer_variable)

Definición:
Destruye el timer especificado y libera su memoria.

Descripción de los Parámetros:
timer = handle del timer

Descripción del Comando:
Este comando destruirá un timer creado con el comando CreateTimer y libera la memoria que estuviera usando.

Ejemplo:

```
; Create the timer to track speed
frameTimer=CreateTimer(60)

; Your main screen draw loop
While Not KeyHit(1)
WaitTimer(frameTimer) ; Pause until the timer reaches 60
Cls
; Draw your screen stuff
Flip
Wend

; Kill the timer
FreeTimer(frameTimer)
```

WaitTimer (timer_variable)

Definición:
Para la ejecución del programa hasta que el tiempo especificado haya transcurrido.

Descripción de los Parámetros:
timer = handle del timeer obtrenido cuando fue creado con el comando CreateTimer

Descripción del Comando:
Úsalo junto con el comando CreateTimer. Este comando parará la ejecución del programa hasta que el tiemer alcance su valor. Esto es útil para controlar la velocidad de tus programas. Echa un vistazo al comando CreateTime para entender esto mejor.

Ejemplo:

```
; Crea el timer para controlar la velocidad
frameTimer=CreateTimer(60)

; Bucle principal de dibujo
While Not KeyHit(1)
WaitTimer(frameTimer) ; Para hasta que el timer alcance 60
Cls
Flip
Wend
```

WINDOWS/DEBUG

CallDLL(nombre_dll$, nombre_proc$[,en_banco,fuera_banco])

Parámetros:
nombre_dl$ - nombre de la dll
nombre_proc$ - nombre del procedimiento
en_banco (opcional) - handle del banco que ha sido creado en Blitz para un procedimiento de DLL
fuera_banco (opcional) - handle del banco que ha sido creado en un procedimiento de DLL para Blitz

Descripción:
Llama a un procedimiento específico de la DLL especificada y devuelve el valor entero devuelto por el procedimiento de la DLL.
La DLL es llamada a través de punteros y tamaños de bancos de memoria. El prototipo de una función de DLL debería parecerse algo a este ejemplo (Visual C++):

extern "C"{
*_declspec(dllexport) int _cdecl my_dll_func(const void *in,int in_size,void *out,int out_sz);*
}

El bit 'extern "C"' previene el C++ 'destroza-nombres' y el bit _cdecl previene nombre decoración. Podrías llamar a esta función usando algo parecido a esto:

in_bank=CreateBank(...)
out_bank=CreateBank(...)

;poke input parameters into in_bank
result=CallDLL("mydll","my_dll_func",bank1,bank2)
;peek output results from out_bank

SystemProperty (property$)

Parámetros:
property$ - propiedad del sistema que quieras conocer

Descripción:
SystemProperty () devuelve la localización del directorio de sistema, que puede ser diferentes en cada ordenador. Nota: no es una buena idea jugar en el directorio Windows o System de un usuario.

Parámetros válidos son:
"tempdir" - Directorio de temporales
"systemdir" - Directorio System
"windowsdir" - Directorio Windows
"appdir" - Directorio de Archivos de Programa

Ejemplo:

```
Print "Localización del directorio System: " + SystemProperty ("systemdir")
Print "Localización del directorio Windows: " + SystemProperty ("windowsdir")
Print "Localización del directorio de temporales: " + SystemProperty ("tempdir")
Print "Localización del directorio de Archivos de Programa: " + SystemProperty ("appdir")
```

CommandLine$()

Definición:
Lee los parámetros pasados en la línea de comandos en tiempo de ejecución.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Si estás escribiendo una aplicación o un juego que permita empezar con parámetros especiales en la línea de comandos, puedes usar este comando para recuperar esos parámetros. Por ejemplo, quieres comenzar el programa con una variable de debug establecida para poder depurar durante la ejecución. Así, puedes ofrecer la posibilidad de lanzar el ejecutable con una opción /debug. Si se ejecuta con el parámetro puedes activar un flag dentro del juego para saberlo. Para simular la línea de comandos en el editor, selecciona PROGRAMA->LÍNEA DE COMANDOS DEL PROGRAMA desde el menú desplegable e introduce los parámetros que quieras. Mira el ejemplo.

Ejemplo:

```
; CommandLine$() Example
; Be sure to use PROGRAM->PROGRAM COMMAND LINE from the
; pull down and put /debug in there to test with.

a$=CommandLine$()

If a$="/debug" Then
Print "Debug mode is on!"
debug=1
Else
Print "No debugging activated."
debug=0
End If
```

ShowPointer

Parámetros:
Ninguno.

Descripción:
ShowPointer se usa en modos de ventana, y simplemente muestra el puntero del ratón tras haber sido escondido (con HidePointer). No tiene ningún efecto en modos a pantalla completa.

Ejemplo:

```
Graphics 640, 480, 0, 2

HidePointer

Print "Move pointer over window and press a key..."
WaitKey

ShowPointer

Delay 1000
```

HidePointer

Parámetros:
Ninguno.

Descripción:
HidePointer se usa cuando se está en modo ventana y simplemente oculta el puntero de Windows cuando se mueve por la ventana del juego. Puedes volver a mostrarlo con ShowPointer. No tiene efecto en modos a pantalla completa.

Example (using ShowPointer):

```
Graphics 640, 480, 0, 2

HidePointer

Print "Move pointer over window and press a key..."
WaitKey

ShowPointer

Delay 1000
```

AppTitle título$[,mensaje_final$]

Definición:
Establece el título de la barra del programa.

Descripción de los Parámetros:
título$ - el texto que aparecerá en la barra de título de la ventana del programa
mensaje_final$ (opcional) - el texto que será mostrado en un cuadro de diálogo con las opciones OK/CANCEL cuando un usuario hace click en el botón de cerrar. Si no se especifica nada, el cuadro de diálogo no será mostrado y el programa se cerrará inmediatamente.

Descripción del Comando:
Te permite establecer el texto de la barra de título del programa, y el cuadro de diálogo "¿está seguro de querer cerrar el programa?".

Ejemplo:

```
; Set the title bar

AppTitle "Super Invaders V1.0"
```

RuntimeError message$

Definición:
Aparece un cuadro de diálogo de error con un mensaje específico.

Descripción de los Parámetros:
message$ = cadena de texto

Descripción del Comando:
Cuando pones tus propias "trampas" de errores, usa este comando para hacer aparecer un error y cerrar el programa. Puedes especificar el mensaje de error a mostrar.

Ejemplo:

```
;There was a problem - raise an error and quit

RuntimeError "Installation corrupted. Please reinstall."
```

DebugLog message

Definición:
Escribe un mensaje en un informe debug para depurar las aplicaciones.

Descripción de los Parámetros:
message = mensaje de texto

Descripción del Comando:
¡Ahora tienes tu propio informe debug para escribir!
Para los que no estén familiarizados con este tipo de cosas, piensa que el Informe Debug es tu 'notepad' privado. Úsalo para escribir mensajes durante la ejecución del programa. Por ejemplo, podrías escribir los modos de vídeo que el usuario tiene en su sistema, o sólo pequeñas alertas que te permitirán saber por qué parte del código pasa el programa para evitar posibles errores sin necesidad de interrumpir su ejecución.
Estoy seguro de que encontrarás un montón de utilidades a esta función. Mira el ejemplo si todavía estás perdido.

Ejemplo:

```
; DebugLog Example
; Let's start graphics mode
Graphics 640,480,16

; Now, let's load an image that doesn't exist!
gfxPlayer=LoadImage("noimagefound.jpg")
If gfxPlayer=0 Then
DebugLog "Player's Graphics failed to load!"
End If

; This is supposed to generate an error. Press F9 to see the log!
While Not KeyHit(1)
DrawImage gfxPlayer,100,100
Wend
```

Stop

Definición:
Para el la ejecución del programa durante la depuración.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Si estás ejecutando el programa en modo debug (depuración), este comando te permite parar la ejecución del programa y devolverte al editor donde puedes avanzar línea a línea por tu código, ver las vaiables, etc.

Ejemplo:

```
; Halt the program and go to the editor/debugger

Stop
```

End

Definición:
Para la ejecución del programa y sale de él.

Descripción de los Parámetros:
Ninguno.

Descripción del Comando:
Usa este comando para parar la ejecución de tu programa. Serás devuelto al editor si lo estabas ejecutando desde allí o el programa saldrá completamente si estás usando un EXE compilado.

Ejemplo:

```
; If the game is over, then quit

if gameOver=1 then End
```

3D - Categorías

GLOBAL

Graphics3D (ancho,alto[,profundidad,modo])

Parámetros:
ancho – ancho de la resolución de pantalla
alto – alto de la resolución de pantalla
profundidad (opcional) – profundidad de color de la pantalla. Por defecto es el color de máxima profundidad disponible.
modo (opcional) – modo de pantalla. Por defecto es 0.
0: en ventana (si es posible) en el modo de depuración, pantalla completa en modo de no depuración
1: pantalla completa siempre
2. en ventana siempre
3: en ventana/escalada siempre

Descripción:
Establece el modo de gráficos 3D. Este comando debe ejecutarse antes de cualquier otro comando 3D, de otro modo los programas podrían dar error. El ancho y el alto establecen la resolución de la pantalla y los valores más comunes son 640,480 y 800,600. La resolución debe ser compatible con la tarjeta 3D y con el monitor que se están usando. La profundidad establece el modo de color de la pantalla. Si se omite este valor o se establece a 0, se usará la profundidad de color más alta que haya disponible. Otros valores disponibles normalmente son 16, 24 y 32. El modo de color 16-bit muestra el número mínimo de colores, 65536. Los modos de color 24-bit y 32-bit muestran más de 16 millones de colores y suele dar como resultado una mejor calidad de la imagen, aunque puede dar como resultado programas más lentos que en 16-bit.

Ejemplo:

```
; Graphics3D Example
; ------------------

; Sets 3D graphics mode
Graphics3D 640,480,16,0
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,5

While Not KeyDown( 1 )
RenderWorld
Flip
Wend

End
```

Dither enable

Parámetros:
enable - true para activar el dithering, false para desactivarlo. Por defecto es true.

Descripción:
Activa o desactiva el dithering del hardware. El dithering del hardware se usa cuando ejecutamos juegos en modo de color de 16-bit. Debido al hecho de que el modo 16-bit ofrece menos colores de los que el ojo humano distingue, se pueden notar bandas separadas de colores en objetos sombreados. Sin embargo, el dithering del hardware tendrá efecto sobre toda la pantalla para dar la impresión de que se están usando más colores de los que realmente se usan, y generalmente da mejor impresión. Debido al hecho de que los modos 24-bit y 32-bit ofrecen más colores de los que el ojo humano puede distinguir, el dithering del hardware se hace bastante más redundante en estos modos.

Ejemplo:

```
; Dither Example
; --------------

Graphics3D 640,480,16
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

; Rotate light so that it creates maximum shading effect on sphere
RotateEntity light,90,0,0

sphere=CreateSphere( 32 )
PositionEntity sphere,0,0,2

While Not KeyDown( 1 )

; Toggle dither enable value between true and false when spacebar is pressed
If KeyHit( 57 )=True Then enable=1-enable

; Enable/disable hardware dithering
Dither enable

RenderWorld

Text 0,0,"Press spacebar to toggle between Dither True/False"
If enable=False Then Text 0,20,"Dither False" Else Text 0,20,"Dither True"

Flip

Wend

End
```

WBuffer enable

Parámetros:
enable - True para activar el render w-buffering, False para desactivarlo. Por defecto es True para modos de color de 16-bit, y False para 24-bit y 32-bit.

Descripción:
Activa o desactiva el w-buffering. W-buffering es una técnica usada para dibujar objetos 3D según su profundidad – por ejemplo, primero los que estén más lejos de la cámara, después los que estén más cerca de la cámara, etc. Normalmente, el z-buffering se usa como estrategia, pero el z-buffer puede ser ligeramente impreciso en modos de 16-bit, por lo que el nivel de precisión es menos que en modos de color de 24-bit o 32-bit. Esto significa que en algunas situaciones, los objetos aparecerán superponiéndose unos a otros cuando no debería ocurrir. Para compensar esto, puedes usar el w-buffering. Éste es ligeramente más preciso que el z-buffering, aunque en algunas ocasiones puede ser menos compatible que el z-buffering.

Antialias enable

Parámetros:
enable - true para activar el antialiasing a pantalla completa, false para desactivarlo. Por defecto es false.

Descripción:
Activa o desactiva el antialiasing a pantalla completa. Antialiasing a pantalla completa es una técnica usada para suavizar la pantalla completa, para que las líneas angulosas sean menos notables.
Algunas tarjetas 3D tienen soporte incorporado para el antialiasing a pantalla completa, que permitiría activar el efecto sin restar velocidad. Sin embargo, para las tarjetas que no soportan el antialiasing a pantalla completa, activar el efecto podría causar una gran lentitud.

Ejemplo:
```
; AntiAlias Example
; -----------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

sphere=CreateSphere()
PositionEntity sphere,0,0,2

While Not KeyDown( 1 )

; Toggle antialias enable value between true and false when spacebar is pressed
If KeyHit( 57 )=True Then enable=1-enable

; Enable/disable antialiasing
AntiAlias enable

RenderWorld

Text 0,0,"Press spacebar to toggle between AntiAlias True/False"
If enable=False Then Text 0,20,"AntiAlias False" Else Text 0,20,"AntiAlias True"

Flip

Wend

End
```

Wireframe enable

Parámetros:
enable - True para activar el render en wireframe, False para desactivarlo. Por defecto es False.

Descripción:
Activa o desactiva el render wireframe. Esto mostrará las aristas de cada polígono de la pantalla, con las áreas no sombreadas.
Este modo suele usarse sólo para depurar, porque el soporte del controlador es desigual. Por la misma razón no se ofrece soporte para el render wireframe para los entes de polígono individual.

Ejemplo:
```
; Ejemplo Wireframe
; -----------------

Graphics3D 640,480,16
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

sphere=CreateSphere( 32 )
PositionEntity sphere,0,0,2

While Not KeyDown( 1 )
; Toggle wireframe enable value between true and false when spacebar is pressed
```

```
If KeyHit( 57 )=True Then enable=1-enable

; Enable/disable wireframe rendering
WireFrame enable

RenderWorld

Text 0,0,"Press spacebar to toggle between Wireframe True/False"
If enable=True Then Text 0,20,"Wireframe True" Else Text 0,20,"Wireframe False"

Flip

Wend

End
```

HWMultiTex activa

Parámetros:
activa - true para activar la multitextura del hardware, false para desactivarlo. Por defecto es true.

Descripción:
Activa o desactiva la multitextura del hardware. La multitextura es una técnica usada para mostrar más de una textura a la vez en un objeto. A veces, el hardware 3D tiene soporte para esto, por lo que usar dos o más texturas por objeto no causará más lentitud que usar sólo una. Aunque, algunas tarjetas dan problemas con la multitextura del hardware, para estas situaciones tienes la opción de deshabilitarlo. Cuando no se está usando la textura de hardware, Blitz3D usará su propia técnica de software, con lo que involucra objetos duplicados que tienen una textura cada uno.

AmbientLight rojo#,verde#,azul#

Parámetros:
rojo# - valor rojo de la luz ambiental
verde# - valor verde de la luz ambiental
azul# - valor azul de la luz ambiental

Descripción:
Establece el color de la luz ambiental. Los valores verde, rojo y azul deberían estar en el rango 0-255. El color de la luz ambiental por defecto es 255,255,255. La luz ambiental es el origen de una luz que afecta a todos los puntos de un objeto 3D por igual. Por tanto, solo con una luz ambiental, todos los objetos 3D aparecerán absolutamente, no habrá sombras. La luz ambiental se usa para proporcionar un nivel de luz, antes de añadir otras luces que proporcionen un efecto de luz realista. Un nivel de luz ambiental de 0,0,0 dará como resultado un ambiente sin luz.

Ejemplo:

```
; AmbientLight Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

sphere=CreateSphere( 32 )
PositionEntity sphere,-2,0,5

cone=CreateCone( 32 )
PositionEntity cone,2,0,5

; Set initial ambient light colour values
red#=127
green#=127
blue#=127

While Not KeyDown( 1 )

; Change red, green, blue values depending on key pressed
If KeyDown( 2 )=True And red#>0 Then red#=red#-1
```

```
If KeyDown( 3 )=True And red#<255 Then red#=red#+1
If KeyDown( 4 )=True And green#>0 Then green#=green#-1
If KeyDown( 5 )=True And green#<255 Then green#=green#+1
If KeyDown( 6 )=True And blue#>0 Then blue#=blue#-1
If KeyDown( 7 )=True And blue#<255 Then blue#=blue#+1

; Set ambient light using red, green, blue values
AmbientLight red#,green#,blue#

RenderWorld

Text 0,0,"Press keys 1-6 to change AmbientLight red#,green#,blue# values
Text 0,20,"Ambient Red: "+red#
Text 0,40,"Ambient Green: "+green#
Text 0,60,"Ambient Blue: "+blue#

Flip

Wend

End
```

ClearCollisions

Parámetros:
Ninguno.

Descripción:
Limpia la lista de información de colisiones. Siempre que se utiliza el comando Collisions para activar las colisiones entre dos tipos de ente diferentes, la información se añade a la lista de colisiones. Este comando limpia esa lista, por tanto no se detectará ninguna colisión hasta que se use de nuevo el comando Collisions. El comando no limpiará la información de colisión del ente. Por ejemplo, el radio del ente, tipo, etc.

Ejemplo:

```
; ClearCollisions Example
; ----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

sphere=CreateSphere( 32 )
PositionEntity sphere,-2,0,5

cone=CreateCone( 32 )
EntityType cone,type_cone
PositionEntity cone,2,0,5

; Set collision type values
type_sphere=1
type_cone=2

; Set up sphere collision data
EntityRadius sphere,1
EntityType sphere,type_sphere

; Set up cone collision data
EntityType cone,type_cone

; Enable collisions between type_sphere and type_cone, with sphere->polygon method and slide
response
Collisions type_sphere,type_cone,2,2

While Not KeyDown( 1 )
x#=0
y#=0
z#=0
```

```
If KeyDown( 203 )=True Then x#=-0.1
If KeyDown( 205 )=True Then x#=0.1
If KeyDown( 208 )=True Then y#=-0.1
If KeyDown( 200 )=True Then y#=0.1
If KeyDown( 44 )=True Then z#=-0.1
If KeyDown( 30 )=True Then z#=0.1

MoveEntity sphere,x#,y#,z#

; If spacebar pressed then clear collisions
If KeyHit( 57 )=True Then ClearCollisions

; Perform collision checking
UpdateWorld

RenderWorld

Text 0,0,"Use cursor/A/Z keys to move sphere"
Text 0,20,"Press spacebar to use ClearCollisions command"

Flip

Wend

End
```

Collisions orig_tipo,dest_tipo,método,respuesta

Parámetros:

orig_tipo – tipo de ente para comprobar si colisiona.
dest_tipo – tipo de ente con el que colisiona.
método – método de detección de colisión.
1: colisión de esfera a esfera
2: colisión de esfera a polígono
3: colisión de esfera a caja
respuesta – lo que hace el ente de orig_tipo cuando ocurre la colisión.
1: parar
2: deslizarse1 – deslizamiento completo en la colisión
3: deslizarse2 – desliza al ente hacia abajo en una cuesta

Descripción:

Activa las colisiones entre dos tipos distintos de entes. Los tipos de ente son simplemente números que se asignan a un ente usando EntityType. Blitz utiliza los tipos de entes para comprobar las colisiones entre todos los entes que tienen esos tipos de ente.
Blitz tiene varias formas de comprobar las colisiones, como demuestra el parámetro método. Sin embargo, la colisión comprobada es siempre la de una esfera contra algo. Para que Blitz pueda saber el tamaño del ente de origen, se debe asignar primero un "radio del ente" al ente de origen utilizando EntityRadius. En el caso de detectar una colisión teniendo seleccionado el método 1 (esfera a esfera), los entes de destino implicados en la colisión deberán tener también asignado un EntityRadius. En el caso de estar seleccionado el método 3 (esfera a caja), los entes de destino necesitarán tener asignado un EntityBox. El método 2 (esfera a polígono) no requiere que el ente de destino tenga nada asignado. Blitz no solo comprueba las colisiones, sino que también actúa sobre ellas cuando las detecta, como demuestra el parámetro respuesta. Hay tres opciones cuando se da esta situación. Se puede escoger entre hacer al ente detenerse, deslizarse o simplemente hacer que se deslice cuesta abajo. Todas las colisiones y sus respuestas se ejecutan cuando se llama a UpdateWorld. Finalmente, cada vez que se utiliza el comando Collision, la información de la colisión se añade a una lista de información de colisiones. Esta puede borrarse utilizando el comando ClearCollisions.

Ejemplo:

```
; Collisions Example
; ------------------

Graphics3D 640,480
SetBuffer BackBuffer()

; Set collision type values
type_character=1
type_scenery=2

camera=CreateCamera()
RotateEntity camera,45,0,0
```

```
PositionEntity camera,0,15,-10

light=CreateLight()
RotateEntity light,45,0,0

; Create sphere 'character'
sphere=CreateSphere( 32 )
EntityRadius sphere,1
EntityType sphere,type_character
PositionEntity sphere,0,7,0

; Create cube 'scenery'
cube=CreateCube()
EntityRadius cube,1
EntityType cube,type_scenery
PositionEntity cube,0,-5,0
EntityColor cube,127,0,0
ScaleEntity cube,10,10,10

; Create cylinder 'scenery'
cylinder=CreateCylinder( 32 )
ScaleEntity cylinder,2,2,2
EntityColor cylinder,255,0,0
EntityType cylinder,type_scenery
PositionEntity cylinder,-4,7,-4

; Create cone 'scenery'
cone=CreateCone( 32 )
ScaleEntity cone,2,2,2
EntityColor cone,255,0,0
EntityType cone,type_scenery
PositionEntity cone,4,7,-4

; Create prism 'scenery'
prism=CreateCylinder( 3 )
ScaleEntity prism,2,2,2
EntityColor prism,255,0,0
EntityType prism,type_scenery
PositionEntity prism,-4,7,4
RotateEntity prism,0,180,0

; Create pyramid 'scenery'
pyramid=CreateCone( 4 )
ScaleEntity pyramid,2,2,2
EntityColor pyramid,255,0,0
EntityType pyramid,type_scenery
RotateEntity pyramid,0,45,0
PositionEntity pyramid,4,7,4

; Set collision method and response values
method=2
response=2

method_info$="sphere-to-polygon"
response_info$="slide1"

While Not KeyDown( 1 )

x#=0
y#=0
z#=0

If KeyDown( 203 )=True Then x#=-0.1
If KeyDown( 205 )=True Then x#=0.1
If KeyDown( 208 )=True Then z#=-0.1
If KeyDown( 200 )=True Then z#=0.1

MoveEntity sphere,x#,y#,z#
MoveEntity sphere,0,-0.02,0
```

```
; Change collision method
If KeyHit(50)=True
method=method+1
If method=4 Then method=1
If method=1 Then method_info$="sphere-to-sphere"
If method=2 Then method_info$="sphere-to-polygon"
If method=3 Then method_info$="sphere-to-box"
EndIf

; Change collision response
If KeyHit(19)=True
response=response+1
If response=4 Then response=1
If response=1 Then response_info$="stop"
If response=2 Then response_info$="slide1"
If response=3 Then response_info$="slide2"
EndIf

; Enable Collions between type_character and type_scenery
Collisions type_character,type_scenery,method,response

; Perform collision checking
UpdateWorld

RenderWorld

Text 0,0,"Use cursor keys to move sphere"
Text 0,20,"Press M to change collision Method (currently: "+method_info$+")"
Text 0,40,"Press R to change collision Response (currently: "+response_info$+")"
Text 0,60,"Collisions type_character,type_scenery,"+method+","+response

Flip

Wend

End
```

UpdateWorld [veloc_anim#]

Parameters:
veloc_anim# (opcional) - un control general de la velocidad de las animaciones. Por defecto vale 1.

Descripción:
Anima todos los entes del mundo, y realiza la comprobación de colisiones.
El parámetro opcional veloc_anim# permite afectar a la velocidad de las animaciones de todos los entes a la vez. Un valor de 1 animará los entes a su velocidad de animación normal, un valor de 2 animará los entes al doble de su velocidad normal de animación, y así sucesivamente.
Para mejores resultados usa este comando una vez en el bucle principal, justo antes de RenderWorld.

CaptureWorld

Parámetros:
Ninguno.

Descripción:
Hace una captura de la posición y orientación de cada uno de los entes del mundo.
Esta captura puede usarse luego con RenderWorld para renderizar los entes en el punto entre sus posiciones capturadas y sus posiciones actuales. Ver RenderWorld para saber más sobre cómo y porqué se hace.

RenderWorld [tween#]

Parámetros:
tween# - por defecto es 1.

Descripción:
Renderiza todos los entes en un mundo. El parámetro opcional tween puede usarse para renderizar todos los entes en un punto cualquiera entre sus posiciones capturadas y su posición actual. Un valor tween de 0 renderizará los entes a su posición capturada. Un valor de 1 renderizará los entes a su posicioón actual. Pueden usarse otros valores para la interpolación. Por defecto es 1. Tweening es una técnica usada para permitir actualizar la lógica los juegos un número fijo de veces, por ejemplo, 30 mientras interpolamos con Blitz3D en estos juegos los actualiza tantas frames por Segundo como puede, por ejemplo 60+, con cada frame diferente al final. Esto resulta en un juego en el que solamente tiene que tener su lógica de juego actualizada a la mitad de sus renderizados por Segundo, liberando tiempo de CPU, mientras al mismo tiempo el juego se ejecuta de la mejor manera posible en la máquina de cada uno. Renderizar con tweening es una técnica bastante avanzada, y no es necesaria usarla, por lo tanto no te preocupes si no lo entiendes demasiado. Mira la demo del castillo incluida en la carpeta mak (nickname de Mark Sibly, autor de Blitz3D) en la sección de ejemplos de Blitz3D para una demostración del renderizado con tweening.

ClearWorld [entes][,pinceles][,texturas]

Parámetros:
Entes – true para borrar los entes, false para no hacerlo
Pinceles – true para borrar los pinceles, false para no hacerlo
Texturas – true para borrar las texturas, false para no hacerlo

Descripción:
Borra todos los entes, pinceles y/o texturas de un mundo. Se usa cuando el nivel de un juego se ha terminado y se desea borrar todo para prepararse para cargar nuevos entes/pinceles/texturas sin tener que borrar cada ente/pincel/textura individualmente.
Ver también: FreeEntity, FreeBrush, FreeTexture.

LoaderMatrix extensión_archivo$,xx#,xy#,xz#,yx#,yy#,yz#,zx#,zy#,zz#

Parámetros:
Extension_archivo$ - extensión del archivo 3d, por ejemplo ".x",".3ds"
xx# - elemento 1,1 de una matriz de 3x3
xy# - elemento 2,1 de una matriz de 3x3
xz# - elemento 3,1 de una matriz de 3x3
yx# - elemento 1,2 de una matriz de 3x3
yy# - elemento 2,2 de una matriz de 3x3
yz# - elemento 3,2 de una matriz de 3x3
zx# - elemento 1,3 de una matriz de 3x3
zy# - elemento 2,3 de una matriz de 3x3
zz# - elemento 3,3 de una matriz de 3x3

Descripción:
Establece una matriz de archivos 3d cargados con la extensión de archivo especificada. Puede usarse para cambiar el sistema de coordenadas cuando se cargan. Por defecto, se usa el siguiente cargador de matrices:
LoaderMatrix "x",1,0,0,0,1,0,0,0,1 ; sin cambios en el sistema de coordenadas
LoaderMatrix "3ds",1,0,0,0,0,1,0,1,0 ; intercambia los ejes y/z
Puedes usar LoaderMatrix para invertir mallas/animaciones si es necesario, por ejemplo:
LoaderMatrix "x",-1,0,0,0,1,0,0,0,1 ; invierte las coordenadas x para los archivos ".x"
LoaderMatrix "3ds",-1,0,0,0,0,-1,0,1,0 ; intercambia y/z, anula x/z para archivos ".3ds"

TrisRendered()

Parámetros:
Ninguno.

Descripción:
Devuelve el número de triángulos renderizados durante el último RenderWorld.
Útil para depurar - para ver si estás mostrando muchos o pocos polígonos.

Ejemplo:
```
; TrisRendered Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,0,0,-2

light=CreateLight()
RotateEntity light,90,0,0
```

```
segs=Rand( 2,16 )
sphere=CreateSphere(segs)

While Not KeyDown( 1 )

If KeyHit( 57 )=True

FreeEntity sphere
segs=Rand( 2,16 )
sphere=CreateSphere( segs )

EndIf

RenderWorld

Text 0,0,"Press space to create a sphere with a random segments value"

; Display triangles rendered
Text 0,20,"Triangles Rendered: "+TrisRendered()

Flip

Wend

End
```

TEXTURA

CreateTexture (ancho,alto[,flags][,frames])

Parámetros:
ancho – ancho de la textura
alto – alto de la textura
flags (opcional) – flag de la textura
1: Color
2: Alfa
4: Oculta
8: Mipmapped (mapa de bajo detalle)
16: Clamp U
32: Clamp V
64: mapa de reflexión esférica
frames (opcional) – número de frames que tendrá la textura

Descripción:
Crea una textura y devuelve su puntero. El ancho y el alto son el tamaño de la textura. Nota que el tamaño actual de la textura puede ser diferente del ancho y alto requeridos, ya que diferentes tipos de hardware 3D soportan diferentes tamaños de texturas. El parámetro opcional flags permite aplicar ciertos efectos a la textura. Los pueden añadirse para combinar dos o más efectos, por ejemplo 3 (1+2) = textura con color y mapas alfa. Aquí tienes algunas descripciones de los flags:
1: Color – mapa de color, lo que ves es lo que obtienes.
2: Alfa – mapa de alfa. Si una imagen contiene un mapa de alfa, este se usará para hacer ciertas áreas de la textura transparente. A parte de esto, el mapa de color podrá usarse como un mapa alfa. Con los mapas alfa, las áreas oscuras siempre equivalen a transparencias altas, las áreas iluminadas equivalen a baja transparencia.
4: Oculta – todas las áreas de una textura coloreada 0,0,0 no se dibujarán en la pantalla.
8: Mipmapped – versión de bajo detalle de la textura que se usará en grandes distancias. Da como resultado una apariencia borrosa.
16: Clamp u – coordenada u de la textura anclada. Evita que el plegado de la textura.
32: Clamp v – coordenada v de la textura anclada. Evita que el plegado de la textura.
64: mapa de reflexión esférica – mapa de ambiente, para una apariencia brillante.
Una vez que se haya creado una textura, se usará SetBuffer TextureBuffer para dibujarla. Aunque, para mostrar gráficos 2D en una textura, normalmente es más rápido dibujarlos a una imagen y luego copiarlos al buffer de la textura (texturebuffer), y para mostrar gráficos 3D en una textura, la única opción es copiar desde el backbuffer al buffer de la textura. Ver también: LoadTexture, LoadAnimTexture.

Ejemplo:

```
; CreateTexture Example
; ---------------------

Graphics3D 640,480
SetBuffer BackBuffer()
```

```
camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Create texture of size 256x256
tex=CreateTexture(256,256)

; Set buffer - texture buffer
SetBuffer TextureBuffer(tex)

; Clear texture buffer with background white color
ClsColor 255,255,255
Cls

; Draw text on texture
font=LoadFont("arial",24)
SetFont font
Color 0,0,0
Text 0,0,"This texture"
Text 0,40,"was created using" : Color 0,0,255
Text 0,80,"CreateTexture()" : Color 0,0,0
Text 0,120,"and drawn to using" : Color 0,0,255
Text 0,160,"SetBuffer TextureBuffer()"

; Texture cube with texture
EntityTexture cube,tex

; Set buffer - backbuffer
SetBuffer BackBuffer()

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Flip

Wend

End
```

LoadTexture (archivo$[,flags])

Parámetros:
archivo$ - nombre del archive de imagen que se usará como textura
flags (opcional) – flag de la textura:
1: Color
2: Alfa
4: Oculto
8: Mipmapped
16: Clamp U
32: Clamp V
64: mapa de reflexión esférica

Descripción:

Carga una textura desde un archive de imagen y devuelve el puntero de la textura. El parámetro opcional flags permite aplicar ciertos efectos a la textura. Los flags se pueden añadir para combinar dos o más efectos, por ejemplo 3 (1+2)= textura con color y mapas alfa. Pequeña descripción de los flags:
1: Color – color del mapa, lo que ves es lo que obtienes.
2: Alfa – mapa alfa. Si una imagen contiene un mapa alfa, este se usará para hacer transparentes ciertas áreas de la textura. A parte de esto, el mapa de color se usará como un mapa alfa. En los mapas alfa, las áreas oscuras equivalen a una alta transparencia, las áreas iluminadas equivalen a una baja transparencia.
4: Oculta – todas las áreas de una textura coloreadas 0,0,0 no se dibujarán en la pantalla.
8: Mipmapped – version de bajo detalle de la textura, se usará para distancias altas. El resultado es una apariencia suavizada y borrosa
16: Clamp u – anclar textura en coordenada u. Evita el plegado de la textura.
32: Clamp v – anclar textura en coordenada v. Evita el plegado de la textura.
64: mapa de reflexión esférica – mapa de ambiente, para una apariencia más brillante
Algo a tener en cuenta cuando aplicamos flags de texturas al cargar una textura es que la textura puede tener ya ciertos flags aplicados mediante el comando TextureFilter. El valor por defecto del comando TextureFilter es 9 (1+8), que es una textura coloreada y mipmapped. Esto no se puede controlar mediante el parámetro flags del comando LoadTexture – si se desea eliminar los filtros será necesario usar el comando ClearTextureFilters. Ver también: CreateTexture, LoadAnimTexture.

Ejemplo:

```
; LoadTexture Example
; -------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture("../media/b3dlogo.jpg")

; Texture cube with texture
EntityTexture cube,tex

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Flip

Wend

End
```

LoadAnimTexture (archivo$,flags,ancho_frame,alto_frame,primer_frame,contar_frame)

Parámetros:

archivo$ - nombre del archive con los frames de la animación situados de izquierda a derecha y de arriba a abajo flags – flag de la textura:
1: Color
2: Alfa
4: Oculta
8: Mipmapped

16: Clamp U
32: Clamp V
64: Mapa de reflexión esférica
ancho_frame – ancho de cada frame de animación
alto_frame – alto de cada frame de animación
primer_frame – el primer frame que se usará como frame de animación.
contar_frame – el número de frames que se usará

Descripción:
Carga una secuencia de frames de animación en una textura. Ver también: CreateTexture, LoadTexture.

FreeTexture textura

Parámetros:
textura – puntero de la textura

Descripción:
Libera una textura. Esto permitirá que la memoria que antes estaba ocupada que de libre para usarse en otros propósitos. Liberar una textura significa que no podrás volver a usarla de nuevo; aunque los entes que ya están texturizados no perderán la textura.

Ejemplo:
```
; FreeTexture Example
; -------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture("../media/b3dlogo.jpg")

; Texture cube with texture
EntityTexture cube,tex

While Not KeyDown( 1 )

; If spacebar pressed then free texture
If KeyHit( 57 )=True Then FreeTexture tex

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Text 0,0,"Press spacebar to free texture"
Text 0,20,"As you can see this will not effect already textured entities"
Flip

Wend

End
```

TextureBlend textura,mezcla

Parámetros:
textura – nombre de la textura
mezcla – modo de mezcla de la textura
0: textura desactivada
1: alfa
2: múltiple (por defecto)
3: añadir

Descripción:
Establece el modo de mezcla para una textura. Se usa con multitextura para controlar como la textura se combina con otras texturas.

TextureCoords textura,coordenadas

Parámetros:
textura – nombre de la textura
coordenadas -
0: las coordenadas UV (ultravioletas) están establecidas de primeras en los vértices (por defecto)
1: las coordenadas UV (ultravioletas) están establecidas de segundas en los vértices.

Descripción:
Establece el modo de coordinar las texturas para una textura. Esto determina donde se usan los valores UV para buscar la procedencia de una textura.

ScaleTexture textura,u_escala#,v_escala#

Parámetros:
textura – nombre de la textura
u_escala# - escala u
v_escala# - escala v

Descripción:
Escala una textura por un número absoluto. Tendrá efecto inmediato sobre todos los objetos que estén usando la textura.

Ejemplo:
```
; ScaleTexture Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture( "media/b3dlogo.jpg" )

; Texture cube
EntityTexture cube,tex

; Set initial uv scale values
u_scale#=1
v_scale#=1

While Not KeyDown( 1 )

; Change uv scale values depending on key pressed
If KeyDown( 208 )=True Then u_scale#=u_scale#-0.01
If KeyDown( 200 )=True Then u_scale#=u_scale#+0.01
If KeyDown( 203 )=True Then v_scale#=v_scale#-0.01
```

```
If KeyDown( 205 )=True Then v_scale#=v_scale#+0.01

; Scale texture
ScaleTexture tex,u_scale#,v_scale#

TurnEntity cube,0.1,0.1,0.1

RenderWorld

Text 0,0,"Use cursor keys to change uv scale values"
Text 0,20,"u_scale#="+u_scale#
Text 0,40,"v_scale#="+v_scale#

Flip

Wend

End
```

PositionTexture textura,posición_u#,posición_v#

Parámetros:
textura – puntero de la textura
posición_u# - posición u de la textura
posición_v# - posición v de la textura

Descripción:
Posiciona una textura en una posición absoluta. Tendrá un efecto inmediato en todos los objetos que usen la textura. Posicionar una textura se usa para realizar un scroll de textura, como por ejemplo agua, etc.

Ejemplo:

```
; PositionTexture Example
; -----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture( "media/b3dlogo.jpg" )

; Texture cube
EntityTexture cube,tex

; Set initial uv position values
u_position#=1
v_position#=1

While Not KeyDown( 1 )

; Change uv position values depending on key pressed
If KeyDown( 208 )=True Then u_position#=u_position#-0.01
If KeyDown( 200 )=True Then u_position#=u_position#+0.01
If KeyDown( 203 )=True Then v_position#=v_position#-0.01
If KeyDown( 205 )=True Then v_position#=v_position#+0.01

; Position texture
PositionTexture tex,u_position#,v_position#
```

```
TurnEntity cube,0.1,0.1,0.1

RenderWorld

Text 0,0,"Use cursor keys to change uv position values"
Text 0,20,"u_position#="+u_position#
Text 0,40,"v_position#="+v_position#

Flip

Wend

End
```

RotateTexture textura,ángulo#

Parámetros:
textura – puntero de la textura
ángulo# - ángulo absoluto de rotación de la textura

Descripción:
Rota una textura. Tendrá efecto inmediato sobre todos los objetos en los que se usa la textura. Rotar una textura se usa mayormente para realizar el efecto remolino en texturas, como el humo, etc.

Ejemplo:

```
; RotateTexture Example

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture( "media/b3dlogo.jpg" )

; Texture cube
EntityTexture cube,tex

; Set initial texture angle value
angle#=1

While Not KeyDown( 1 )

; Change texture angle value depending on key pressed
If KeyDown( 205 )=True Then angle#=angle#-1
If KeyDown( 203 )=True Then angle#=angle#+1

; Rotate texture
RotateTexture tex,angle#

TurnEntity cube,0.1,0.1,0.1

RenderWorld

Text 0,0,"Use left and right cursor keys to change texture angle value"
Text 0,20,"angle#="+angle#

Flip

Wend
End
```

TextureWidth (textura)

Parámetros:
textura – puntero de la textura

Descripción:
Devuelve el ancho de una textura.

TextureHeight (textura)

Parámetros:
textura – puntero de la textura

Descripción:
Devuelve el alto de una textura.

TextureBuffer (textura[,frame])

Parámetros:
textura – puntero de la textura
frame (opcional) – frame (escena) de la textura

Descripción:
Devuelve el puntero del buffer dibujado de una textura. Se puede usar con SetBuffer para realizar operaciones de dibujo 2D a la textura, aunque normalmente es más rápido dibujar una imagen, y copiar el buffer de la imagen a través del buffer de la textura usando CopyRect. No se puede renderizar 3D a un buffer de una textura, el 3D solo se puede renderizar al Back Buffer. Para mostrar gráficos 3D en una textura, usa CopyRect para copiar el contenido del back buffer a un buffer de una textura.

Ejemplo:
```
; TextureBuffer Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Create texture of size 256x256
tex=CreateTexture(256,256)

; Set buffer - texture buffer
SetBuffer TextureBuffer(tex)

; Clear texture buffer with background white color
ClsColor 255,255,255
Cls

; Draw text on texture
font=LoadFont("arial",24)
SetFont font
Color 0,0,0
Text 0,0,"This texture"
Text 0,40,"was created using" : Color 0,0,255
Text 0,80,"CreateTexture()" : Color 0,0,0
Text 0,120,"and drawn to using" : Color 0,0,255
Text 0,160,"SetBuffer TextureBuffer()"

; Texture cube with texture
EntityTexture cube,tex
```

```
; Set buffer - backbuffer
SetBuffer BackBuffer()

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Flip

Wend

End
```

ClearTextureFilters

Parámetros:
Ninguno.

Descripción:
Borra la lista actual de filtros de textura.

TextureFilter match_text$,flags

Parámetros:
match_text$ - texto que, si se encuentra en el nombre del fichero de la textura, activará ciertos filtros.
flags – filtro del flags

Descripción:
Añade un filtro de textura. A cualquier textura cargada que contenga el texto especificado en match_text$ se le añadirá el flags suministrado. Esto se utiliza mayormente cuando se carga una malla. Por defecto se usa el siguiente filtro de la textura: TextureFilter "",1+8. Esto significa que todas las texturas cargadas tendrán el color y el mapeado por defecto.

PINCEL

CreateBrush ([rojo#][,verde#][,azul#])

Parámetros:
rojo# (opcional) – valor rojo del pincel
verde# (opcional) – valor verde del pincel
azul# (opcional) – valor azul del pincel

Descripción:
Crea un pincel y devuelve el puntero de un pincel. Los valores verde, rojo y azul permiten establecer el color del pincel. Los valores deberían estar en el rango 0-255. Si se omiten los valores, por defecto son 255. Un pincel es una colección de propiedades como Color, Alfa, Brillo, Textura, etc que se almacenan como parte de un pincel. Entonces todas estas propiedades pueden aplicarse a un ente, malla o superficie a la vez utilizando PaintEntity, PaintMesho PaintSurface. Cuando creamos nuestra propia malla, si lo deseamos podemos hacer que una determinada superficie parezca diferente a otra, es entonces cuando necesitaremos usar los pinceles para pintar superficies individuales. Usando comandos como EntityColor, EntityAlpha aplicaremos el efecto a todas las superficies a la vez, algo que tal vez no deseemos hacer. Ver también: LoadBrush.

Ejemplo:

```
; CreateBrush Example
; -------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture("../media/b3dlogo.jpg")

; Create brush
brush=CreateBrush()

; Apply texture to brush
BrushTexture brush,tex

; And some shininess
BrushShininess brush,1

; Paint mesh with brush
PaintMesh cube,brush

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Flip

Wend

End
```

LoadBrush (archivo$[,flags][,escala_u][,escala_v]

Parámetros:

archivo$ - nombre del archive
flags – flags del pincel
flags (opcional) – se pueden añadir flags para combinar efectos:
1: Color
2: Alfa
4: Oculto
8: Mipmapped
16: Clamp U
32: Clamp V
64: mapa de reflexión esférica
escala_u – escala u del pincel
escala_v – escala v del pincel

Descripción:
Crea un pincel, le asigna una textura y devuelve el puntero del pincel.

FreeBrush pincel

Parámetros:
pincel – puntero del pincel

Descripción:
Libera un pincel.

BrushColor brush,rojo#,verde#,azul#

Parámetros:
brush – puntero del pincel
rojo# - valor rojo del pincel
verde# - valor verde del pincel
azul# - valor azul del pincel

Descripción:
Establece el color de un pincel. Los valores verde, rojo y azul deberían estar en el rango 0-255. El color por defecto del pincel es 255,255,255. Por favor, nota que si se está usando EntityFX o BrushFX flag 2 el color del pincel no tendrá efecto y se usarán en su lugar los colores de los vértices.

Ejemplo:

```
; BrushColor Example
Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Create brush
brush=CreateBrush()

; Set brush color
BrushColor brush,0,0,255

; Paint mesh with brush
PaintMesh cube,brush

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Flip

Wend

End
```

BrushAlpha brush,alpha#

Parámetros:
brush – puntero del pincel
alpha# - nivel de alfa para el pincel

Descripción:
Establece el nivel de alfa para el pincel. El valor de alpha# debería estar en el rango 0-1. El valor por defecto del alfa del pincel es 1. El nivel de alfa es como es de transparente un ente. Un valor de 1 significará que el ente no es transparente, es decir, opaco. Un valor de 0 significará que el ente es completamente transparente, es decir, invisible. Los valores entre 0 y 1 causarán variaciones en la transparencia según el número, se usa para imitar la apariencia de algunos objetos como el cristal y el hielo. Un valor del alfa del pincel de 0 es especialmente práctico, porque Blitz3D no renderizará entes con dicho valor, pero los incluirá en las colisiones de los entes. Es diferente a HideEntity, que no incluye los entes en las colisiones.

BrushShininess brush,brillo#

Parámetros:
brush – puntero del pincel
brillo# - brillo del pincel

Descripción:
Establece el reflejo del brillo de un pincel. El valor brillo# debería estar en el rango 0-1. El valor por defecto del brillo es 0. El brillo es como brillan ciertas áreas de un objeto cuando la luz les ilumina directamente. Establecer el valor del brillo en 1 para la mitad superior de una esfera, combinado con la creación de una luz iluminando en su dirección, dará la apariencia de una bola de billar brillante.

BrushTexture brush,textura[,frame][,índice]

Parámetros:
brush – puntero del pincel
textura – textura a tratar
frame (opcional) – frame de la textura. Por defecto es 0.
índice (opcional) – índice de la textura. Por defecto es 0.

Descripción:
Asigna una textura a un pincel. El parámetro opcional frame especifica que frame de la animación, si existe alguno, debería asignarse al pincel. El parámetro opcional índice especifica la capa de la textura a la que debería asignarse la textura. Los pinceles tienen cuatro capas de textura, 0-3 inclusives.

Ejemplo:
Ver Ejemplo CreateBrush

BrushBlend brush,blend

Parámetros:
brush – puntero del pincel
blend – mezcla:
1: alfa
2: multiplicar (por defecto)
3: añadir

Descripción:
Establece el modo de mezcla de un pincel.

BrushFX brush,fx

Parámetros:
brush – puntero del pincel
fx -
1: brillo completo
2: usa los colores de los vértices en lugar del color del pincel
4: flatshaded
8: desactivar niebla

Descripción:
Establece los efectos miscelánea de un pincel. Pueden añadirse flags para combinar dos o más efectos, especificando un flag de 3(1+2) resultará un brillo completo y un pincel con el color de los vértices.

Ejemplo:
```
; BrushFX Example
; ---------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone()
PositionEntity cone,0,0,5

While Not KeyDown( 1 )

    RenderWorld
    Flip

Wend

End
```

MALLA

CreateMesh ([padre])

Parámetros:
padre (opcional) – ente padre de la malla

Descripción:
Crea una malla y devuelve el puntero de la malla.

LoadMesh (archivo$[,padre])

Parámetros:
archivo$ - nombre del archive de la malla
padre (opcional) – ente padre de la malla

Descripción:
Carga una malla desde un archivo .x o .3ds y devuelve el puntero de la malla. Cualquier información de la animación y de su jerarquía que esté en el archivo será ignorada. Usa LoadAnimMesh para conservar la información de la animación y de su jerarquía. El parámetro opcional padre permite especificar un ente padre para la malla, de forma que cuando se mueve el padre, la malla hija se mueve con él. Aunque esta relación sólo se da en un sentido, aplicar comandos de movimiento a un hijo no afectará al padre. Especificar un ente padre creará la malla en la posición 0,0,0 en lugar de en la posición del ente padre. Ver también: LoadAnimMesh.

Ejemplo:
```
; LoadMesh Example
Graphics3D 640,480
SetBuffer BackBuffer()
camera=CreateCamera()
light=CreateLight()
RotateEntity light,90,0,0
drum=LoadMesh("media/oil-drum/oildrum.3ds")
PositionEntity drum,0,0,MeshDepth(drum)*2
While Not KeyDown( 1 )
RenderWorld
Flip
Wend
End
```

LoadAnimMesh (archivo$[,padre])

Parámetros:
archivo$ - nombre del archive de la malla
padre – ente padre de la malla

Descripción:
Carga una malla desde un archivo .x o .3ds y devuelve el puntero de la malla. Cualquier información de la animación y su jerarquía del archivo será retenida (si está presente en el archivo!).

CreateCube([padre])

Parámetros:
padre (opcional) – ente padre del cubo

Descripción:
Crea una malla/ente cubo y devuelve su puntero. Un cubo se extenderá de -1,-1,-1 to +1,+1,+1. El parámetro opcional padre permite especificar un ente padre para el cubo para que cuando el padre sea movido, el cubo hijo se mueva con él. Aunque esta relación solo se da en un sentido, aplicar un comando de movimiento a un hijo no afectará al padre. Especificar un ente padre dará como resultado que el cubo se cree en la posición 0,0,0 en lugar de en la posición del ente padre. Ver también: CreateSphere, CreateCylinder, CreateCone.

Ejemplo:
```
; CreateCube Example
; ------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Create cube
cube=CreateCube()

PositionEntity cube,0,0,5

While Not KeyDown( 1 )
RenderWorld
Flip
Wend

End
```

CreateSphere ([segmentos][,padre])

Parámetros:
segmentos (opcional) – detalle de la esfera. Por defecto es 8.
padre (opcional) – ente padre de la esfera

Descripción:
Crea una malla/ente esfera y devuelve su puntero. La esfera se centrará en 0,0,0 y tendrá un radio de 1. El valor de los segmentos debe estar en el rango 2-100 inclusive, aunque esto sólo se comprueba en el modo de depuración. Un error muy común es dejar el modo de depuración apagado y especificar un parámetro padre (normalmente un número de ocho cifras en la dirección de la memoria) en lugar del valor de los segmentos. Como el número de polígonos usado para crear la esfera es exponencialmente proporcional al valor de los segmentos, esto dará como resultado que Blitz intente crear una esfera con un número inimaginable de polígonos! Dependiendo de lo afortunado que seas, tu computador se colgará.
Ejemplo de valores de segmentos:
8: 224 polígonos – número mínimo de polígonos para una esfera
16: 960 polígonos – esfera suavizada en distancias medias-altas
32: 3968 polígonos – esfera suavizada en distancias cortas
El parámetro opcional padre permite especificar un ente padre para la esfera, de forma que cuando el ente padre se mueve, la esfera hija se moverá con él. Aunque esta relación sólo se da en un sentido, aplicar un comando de movimiento al hijo no tendrá efecto sobre el padre. Especificar un ente padre hará que la esfera se cree en la posición 0,0,0 en lugar de en la posición de ente padre. Ver también: CreateCube, CreateCylinder, CreateCone.

Ejemplo:
```
; CreateSphere Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Create sphere
sphere=CreateSphere()

PositionEntity sphere,0,0,5

While Not KeyDown( 1 )
RenderWorld
Flip
Wend

End
```

CreateCylinder ([segmentos][,padre][,sólido])

Parámetros:
segmentos (opcional) – detalle del cilindro. Por defecto es 8.
padre (opcional) – ente padre del cilindro
sólido (opcional) - true para un cilindro, false para un tubo. Por defecto es true.

Descripción:
Crea una malla/ente cilindro y devuelve su puntero. El cilindro se centrará en 0,0,0 y tendrá un radio de 1. El valor del segmento debe estar en el rango 3-100 inclusive, aunque esto sólo se comprueba en el modo de depuración. Un error muy común es dejar el modo de depuración apagado y especificar un parámetro padre (normalmente una número de ocho cifras en la dirección de memoria) en lugar del valor de los segmentos. Como el número de polígonos que se usa para crear un cilindro es exponencialmente proporcional al valor de segmentos, esto causará que el Blitz trate de crear un cilindro con un número inimaginable de polígonos! Dependiendo de lo afortunado que seas, tu computadora se colgará. Ejemplo de valores de segmentos (sólido=true):
3: 8 polígonos – un prima
8: 28 polígonos – número mínimo de polígonos para un cilindro
16: 60 polígonos – cilindro suavizado a una distancia media-alta
32: 124 polígonos – cilindro suavizado a una distancia corta
El parámetro opcional padre permite especificar un ente padre para el cilindro de forma que cuando el padre se mueve, el cilindro hijo se mueva con él. Aunque esto solo se da en un sentido, aplicar comandos de movimiento al hijo no afectará al padre. Especificar un ente padre hará que el cilindro se cree en la posición 0,0,0 en lugar de en la posición del ente padre. Ver también: CreateCube, CreateSphere, CreateCone.

Ejemplo:
```
; CreateCylinder Example
; ----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Create cylinder
cylinder=CreateCylinder()

PositionEntity cylinder,0,0,5

While Not KeyDown( 1 )
RenderWorld
Flip
Wend
End
```

CreateCone ([segmentos][,padre][,sólido])

Parámetros:
segmentos (opcional) – detalles del cono. Por defecto es 8.
padre (opcional) – ente padre del cono
sólido (opcional) – true para un cono con base, false para un cono sin base. Por defecto es true.

Descripción:
Crea una malla/ente cono y devuelve su puntero. El cono se centrará en 0,0,0 y la base del cono tendrá un radio de 1. El valor segmento debe estar en el rango 3-100 inclusive, aunque esto solo se comprueba en el modo de depuración. Un error muy común es dejar el modo de depuración apagado y especificar el parámetro padre (normalmente un dígito de ocho cifras en la dirección de memoria) en lugar del valor de segmentos. Como el número de polígonos utilizados para crear un cono es exponencialmente proporcional al valor de los segmentos, esto tendrá como resultado en Blitz intentar crear un cono con un número inimaginable de polígonos! Dependiendo de lo afortunado que seas, tu máquina se colgará. Ejemplo de valores de segmentos (sólido=true):
4: 6 polígonos - una pirámide
8: 14 polígonos – el número mínimo de polígonos para un cono
16: 30 polígonos – cono suavizado a distancias medias-altas
32: 62 polígonos – cono suavizado a distancias cortas
El parámetro opcional padre permite especificar el ente padre del cono de forma que cuando el padre se mueve, el cono hijo se moverá con el . Aunque esta relación sólo se da en un sentido, aplicar un movimiento de comando al hijo no afectará al padre. Especificar un ente padre dará como resultado que el cono se cree en la posición 0,0,0 en lugar de en la posición del ente padre. Ver también: CreateCube, CreateSphere, CreateCylinder.

Ejemplo:
```
; CreateCone Example
; -----------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Create cone
cone=CreateCone()

PositionEntity cone,0,0,5

While Not KeyDown( 1 )
RenderWorld
Flip
Wend

End
```

AddMesh orig_malla,dest_mesh

Parámetros:
orig_malla – puntero de la malla de origen
dest_malla – puntero de la malla de destino

Descripción:
Añade la malla de origen a la malla de destino.

FlipMesh malla

Parámetros:
malla – puntero de la malla

Descripción:
Voltea todos los triángulos en una malla. Se usa por un par de razones. Primeramente, es importante entender ligeramente la teoría sobre los gráficos 3D. Un triángulo 3D se representa mediante tres puntos, sólo cuando esos tres puntos están configurados en el sentido de las agujas del reloj, el triángulo es visible. Por lo tanto, realmente, los triángulos sólo tienen un lado. Normalmente, por ejemplo en el caso de una esfera, un modelo tiene triángulos que están dentro del modelo, no te preocupes si no puedes verlos. Aunque, que ocurriría si quisiéramos usar la

esfera como un enorme cielo para el mundo, es decir, si solamente necesitásemos ver el interior? En este caso, simplemente usaríamos FlipMesh. Otro uso de FlipMesh es hacer objetos de bilaterales, así podremos verlos desde dentro y desde fuera si no podemos aún. En este caso, copia la malla original usando CopyEntity, especificando la malla original como padre, y voltéala usando FlipMesh. Ahora tendrás dos mallas ocupando el mismo espacio- esto lo hará bilateral, pero cuidado, también dobla el número de polígonos! Las técnicas de arriba son muy valiosas cuando se ha exportado un modelo de un programa externo de modelado y parece haber perdido algunos de sus triángulos.

Ejemplo:

```
; FlipMesh Example
; ----------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

; Create sphere
sphere=CreateSphere()

; Scale sphere
ScaleEntity sphere,100,100,100

; Texture sphere with sky texture
sky_tex=LoadTexture("../media/sky.bmp")
EntityTexture sphere,sky_tex

; Flip mesh so we can see the inside of it
FlipMesh sphere

Color 0,0,0

While Not KeyDown( 1 )
RenderWorld
Text 0,0,"You are viewing a flipped sphere mesh - makes a great sky!"
Flip
Wend

End
```

PaintMesh malla,pincel

Parámetros:

malla – puntero de la malla
pincel – puntero del pincel

Descripción:

Pinta una malla con un pincel. Tiene el efecto de alterar instantáneamente la apariencia visible de una malla, asumiendo que las propiedades del pincel sean distintas de las aplicadas antes a la superficie. La razón de usar PaintMesh para aplicar propiedades específicas a una malla usando un pincel en lugar de usar EntityTexture, EntityColor, EntityShininess etc, es que se puede predefinir un pincel y luego usarlo para pintar mallas una y otra vez con un solo comando en lugar de un montón de comandos por separado. Ver también: PaintEntity, PaintSurface.

Ejemplo:

```
; PaintMesh Example
; -----------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture("../media/b3dlogo.jpg")
```

```
; Create brush
brush=CreateBrush()

; Apply texture to brush
BrushTexture brush,tex

; And some other effects
BrushColor brush,0,0,255
BrushShininess brush,1

; Paint mesh with brush
PaintMesh cube,brush

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Flip

Wend

End
```

LightMesh malla,rojo#,verde#,azul#[,rango#][,luz_x#][,luz_y#][,luz_z#]

Parámetros:
malla – puntero de la malla
rojo# - valor rojo de la malla
verde# - valor verde de la malla
azul# - valor azul de la malla
rango# (opcional) – rango de la luz
luz_x# (opcional) – posición x de la luz
luz_y# (opcional) – posición y de la luz
luz_z# (opcional) – posición z de la luz

Descripción:
Realiza una "falsa" operación de iluminación sobre una malla.

FitMesh malla,x#,y#,z#,ancho#,alto#,profundidad#[,uniforme]

Parámetros:
malla – puntero de la malla
x# - posición x de la malla
y# - posición y de la malla
z# - posición z de la malla
ancho# - ancho de la malla
alto# - alto de la malla
profundidad# - profundidad de la malla
uniforme (opcional) – si es true, la malla se escalará con el mismo número en x, y y z, y no se deformará. Por defecto es false.

Descripción:
Escala y traslada todos los vertices de una malla de forma que la malla ocupe la caja especificada.

ScaleMesh malla,escala_x#,escala_y#,escala_z#

Parámetros:
malla – puntero de la malla
escala_x# - escala de la malla en x
escala_y# - escala de la malla en y
escala_z# - escala de la malla en z

Descripción:
Escala todos los vértices de una malla según los factores de escala especificados.

RotateMesh malla,pitch#,yaw#,roll#

Parámetros:
malla – puntero de la malla
pitch# - pitch de la malla
yaw# - yaw de la malla
roll# - roll de la malla

Descripción:
Rota todos los vértices de una malla según la rotación especificada. Pitch es lo mismo que el ángulo x de una malla, y es equivalente a inclinar alante/atrás. Yaw es lo mismo que el ángulo y de una malla, y es equivalente a girar izquierda/derecha. Roll es lo mismo que el ángulo z de una malla, y es equivalente a inclinar izquierda/derecha.

PositionMesh malla,x#,y#,z#

Parámetros:
malla – puntero de la malla
x# - posición x de la malla
y# - posición y de la malla
z# - posición z de la malla

Descripción:
Mueve todos los vértices de una malla.

UpdateNormals malla

Parámetros:
malla – puntero de la malla

Descripción:
Recalcula todas las normales en una malla. Esto es necesario para corregir la iluminación si no has establecido las normales de la superficie usando los comandos 'VertexNormals'.

MeshesIntersect (malla_a,malla_b)

Parámetros:
Malla_a – puntero de la malla_a
malla_b – puntero de la malla_b

Descripción:
Devuelve true si las mallas especificadas se están cruzando actualmente. Esta es una rutina bastante lenta – úsala con discreción... Este comando es actualmente la única rutina de comprobación de colisiones polígono->polígono disponible en Blitz3D.

Ejemplo:
```
; MeshesIntersect Example
; -----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
```

```
light=CreateLight()
RotateEntity light,90,0,0

drum=LoadMesh("media/oil-drum/oildrum.3ds")
PositionEntity drum,-20,0,100

crate=LoadMesh("media/wood-crate/wcrate1.3ds")
PositionEntity crate,20,0,100

While Not KeyDown( 1 )

TurnEntity drum,1,1,1
TurnEntity crate,-1,-1,-1

RenderWorld

; Test to see if drum and crate meshes are intersecting; if so then display message to confirm this
If MeshesIntersect(drum,crate)=True Then Text 0,0,"Meshes are intersecting!"

Flip

Wend

End
```

MeshWidth# (malla)

Parámetros:
malla – puntero de la malla

Descripción:
Devuelve el ancho de una malla. Ver también: MeshHeight, MeshDepth.

MeshHeight# (malla)

Parámetros:
malla – puntero de la malla

Descripción:
Devuelve la altura de una malla. Ver también: MeshWidth, MeshDepth.

MeshDepth# (malla)

Parámetros:
malla – puntero de la malla

Descripción:
Devuelve la profundidad de una malla. Ver también: MeshWidth, MeshHeight.

CountSurfaces (malla)

Parámetros:
malla – puntero de la malla

Descripción:
Devuelve el número de superficies (caras) en una malla. Las superficies son secciones de una malla. Una malla puede contener una sección o varias. Ver también: GetSurface.

GetSurface (malla, índice)

Parámetros:
malla – puntero de la malla
índice – índice de la superficie

Descripción:
Devuelve el puntero de una superficie unida a la malla especificada y con el número de índice especificado. El índice debería estar en el rango 1...CountSurfaces(malla), inclusive. Necesitarás obtener una superficie, es decir, averiguar su puntero, para poder usar esa superficie en particular con otros comandos. Ver también: CountSurfaces.

SUPERFICIE

CreateSurface (malla[,pincel])

Parámetros:
malla – puntero de la malla
pincel (opcional) – puntero del pincel

Descripción:
Crea una superficie unida a una malla y devuelve el puntero de la superficie. Las superficies son secciones de malla que se usan para unir triángulos. Debe haber al menos una superficie por malla para crear una malla visible, aunque se pueden usar tantas como se quiera. Dividir una malla en varias secciones permite afectar a esas secciones individualmente, lo que es más útil que si todas las superficies están combinadas en una sola.

PaintSurface superficie,pincel

Parámetros:
superficie – puntero de la superficie
pincel – puntero del pincel

Descripción:
Pinta una superficie con un pincel. Tiene el efecto de alterar instantáneamente la apariencia visible de una superficie particular, por ejemplo, una sección de malla, suponiendo que las propiedades del pincel sean diferentes de las aplicadas antes a la superficie. Ver también: PaintEntity, PaintMesh.

ClearSurface superficie,[limpiar_vértices][,limpiar_triángulos]

Parámetros:
superficie – puntero de la superficie
limpiar_vértices (opcional) - true para borrar todos los vértices de la superficie especificada, false para no hacerlo. Por defecto es true.
limpiar_triángulos (opcional) - true para borrar todos los triángulos de la superficie especificada, false para no hacerlo. Por defecto es true.

Descripción:
Borra todos los vértices y/o triángulos de una superficie. Se usa para borrar secciones de una malla. El resultado será visible instantáneamente. Después de borrar una superficie, puede quererse crearla de nuevo pero con una ligera diferencia en el número de polígonos para obtener un nivel dinámico de detalle (LOD).

FindSurface (malla,pincel)

Parámetros:
malla – puntero de la malla
pincel – puntero del pincel

Descripción:
Intenta encontrar una superficie unida a la malla y creada con el pincel especificado. Devuelve el puntero de la superficie si la encuentra o 0 si no la encuentra. Ver también: CountSurfaces, GetSurface.

AddVertex (superficie,x#,y#,z#[,u#][,v#][,w#])

Parámetros:
superficie – puntero de la superficie
x# - coordenada x del vértice
y# - coordenada y del vértice
z# - coordenada z del vértice
u# (opcional) – coordenada u de la textura del vértice
v# (opcional) – coordenada v de la textura del vértice
w# (opcional) – coordenada w de la textura del vértice – sin utilizar, incluida para una futura expansión

Descripción:
Añade un vértice a una superficie especificada y devuelve el número de índice del vértice, comenzando desde 0. x,y,z son las coordenadas geométricas del vértice, y u,v,w son las coordenadas del mapa de la textura. Un vértice es un punto en el espacio 3D que se usa para conectar los límites de un triángulo. Sin vértices, no se pueden hacer triángulos. Se necesitan tres vértices para crear un triángulo; uno por cada esquina. Los parámetros opcionales u,v permiten especificar las coordenadas de textura de un vértice, que determinarán como será el mapa de la textura de triángulo creado usando estos vértices. Funciona en base a lo siguiente: el punto izquierdo más bajo de una imagen tiene la coordenada uv 0,0. el punto derecho más bajo tiene las coordenadas 0,1, arriba a la izquierda 0,1 y arriba a la derecha 1,1. Ahora, dependiendo de que coordenada uv estés asignando al vértice, representará un punto en una imagen. Por ejemplo, una coordenada 0.5, 0.5 representaría el centro de la imagen. Ahora imagina que tienes un triángulo equilátero normal. Asignando al vértice abajo a la izquierda una coordenada uv de 0,0, luego abajo a la derecha la coordenada de 1,0 y arriba al centro 0.5,1, esto texturizará el triángulo con una imagen que se ajustará a él.

AddTriangle (superficie,v0,v1,v2)

Parámetros:
superficie – superficie a tratar
v0 - número de índice del primer vértice del triángulo
v1 – número de índice del segundo vértice del triángulo
v2 – número de índice del tercer vértice del triángulo

Descripción:
Añade un triángulo a una superficie y devuelve el número de índice del triángulo, comenzando desde 0. Los parámetros v0, v1 y v2 son los números de índice de los vértices creados usando AddVertex. Dependiendo de como estén organizados los vértices, el triángulo será visible solo desde cierto lado. Imaginemos que diseñamos los vértices de un triángulo punto a punto, numerados como v0, v1, v2. Si estos puntos van de v0 a v2 en el sentido de las agujas del reloj, entonces el triángulo será visible. Si estos puntos van en sentido anti-horario, entonces el triángulo no será visible. La razón de tener triángulos de un solo lado es que reduce el número de triángulos necesarios para renderizar cuando un lado está orientado hacia un lado de un objeto que no se verá (como la cara interior de una pelota de billar). Sin embargo, si quieres que un triángulo tenga dos caras, puedes crear dos triángulos usando la misma configuración de números en los vértices de ambos pero asignándolos en orden opuesto, o puedes usar CopyEntity y FlipMesh juntos.

VertexCoords superficie,índice,x#,y#,z#

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice
x# - posición x del vértice
y# - posición y del vértice
z# - posición z del vértice

Descripción:
Establece las coordenadas geométricas de un vértice existente. Este es el comando que se usa para realizar lo que comúnmente se conoce como 'deformación dinámica de malla'. Reposicionará un vértice de modo que las aristas conectadas a este también se moverán. Esto dará el efecto de que partes de la malla se deforman de repente.

VertexNormal superficie,índice,nx#,ny#,nz#

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice
nx# - normal x del vértice
ny# - normal y del vértice
nz# - normal z del vértice

Descripción:
Establece la normal de un vértice existente.

VertexColor superficie,índice,rojo#,verde#,azul#

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice
rojo# - valor rojo del vértice
verde# - valor verde del vértice
azul# - valor azul del vértice

Descripción:
Establece el color de un vértice existente.

VertexTexCoords superficie,índice,u#,v#[,w#][,coord_set]

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice
u# - coordenada u# del vértice
v# - coordenada v# del vértice
w# (opcional) - coordenada w# del vértice
coord_set (opcional) – coordenada establecida. Debería ser establecido a 0 o 1.

Descripción:
Establece las coordenadas de la textura de un vértice existente.

CountVertices (superficie)

Parámetros:
superficie – puntero de la superficie

Descripción:
Devuelve el número de vértices en una superficie.

CountTriangles (surperficie)

Parámetros:
superficie – puntero de la superficie

Descripción:
Devuelve el número de triángulos en una superficie.

VertexX# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve la coordenada x de un vértice.

VertexY# (superficie, índice)

Parámetros:
superficie – puntero de la superficie.
índice - índice del vértice

Descripción:
Devuelve la coordenada y de un vértice.

VertexZ (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve la coordenada z de un vértice.

VertexNX# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve el componente x de la normal de un vértice.

VertexNY# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve el componente y de la normal de un vértice.

VertexNZ# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve el componente z de la normal de un vértice.

VertexRed# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve el componente rojo del color de un vértice.

VertexGreen# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice – índice del vértice

Descripción:
Devuelve el componente verde del color de un vértice.

VertexBlue# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve el componente azul del color de un vértice.

VertexU# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve la coordenada de textura u de un vértice.

VertexV# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve la coordenada de textura v de un vértice.

VertexW# (superficie, índice)

Parámetros:
superficie – puntero de la superficie
índice - índice del vértice

Descripción:
Devuelve la coordenada de la textura w de un vértice.

TriangleVertex (superficie,índice_triángulo,esquina)

Parámetros:
superficie – puntero de la superficie
índice_triángulo - índice del triángulo
esquina - esquina del triángulo. Debería ser 0, 1 o 2.

Descripción:
Devuelve el vértice de una esquina del triángulo.

CÁMARA

CreateCamera ([padre])

Parámetros:
padre (opcional) – ente padre de la cámara

Descripción:
Crea un ente cámara y devuelve su puntero. Sin al menos una cámara, no seríamos capaces de ver nada en nuestro mundo 3D. Con más de una cámara, podremos realizar efectos como el modo de pantalla dividida y espejos. Una cámara solamente puede renderizar el backbuffer. Si deseamos mostrar gráficos 3D en una imagen o una textura, copiaremos el contenido del backbuffer al buffer apropiado. El parámetro opcional padre, permite especificar el ente padre de forma que cuando se mueve el padre, la cámara hijo se moverá con él. Aunque está relación solo se da en un sentido, aplicar comandos de movimiento al hijo no tendrá efecto sobre el padre. Especificando un ente padre, la cámara resultante será creada en la posición 0,0,0 en lugar de en la posición del ente padre.

Ejemplo:
```
; CreateCamera Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

; Create camera
camera=CreateCamera()

light=CreateLight()

cone=CreateCone()
PositionEntity cone,0,0,5

While Not KeyDown( 1 )
RenderWorld
Flip
Wend

End
```

CameraFogMode cámara,modo

Parámetros:
cámara – puntero de la cámara
modo – modo de niebla
0: sin niebla
1: niebla lineal

Descripción:
Establece el modo de niebla de la cámara. Esto activa/desactiva la niebla, una técnica usada para hacer desaparecer gradualmente los gráficos más alejados de la cámara. Se puede utilizar para evitar que los objetos 3D que van apareciendo en el horizonte, lo hagan de repente. El color por defecto de la niebla es negro y el rango por defecto de la niebla es 1-1000, aunque esto puede cambiarse utilizando CameraFogColor y CameraFogRange respectivamente. Cada cámara puede tener su propio modo de niebla, para múltiples efectos de niebla en la pantalla.

Ejemplo:
```
; CameraFogMode Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,0,1,0
CameraFogRange camera,1,10

light=CreateLight()
RotateEntity light,90,0,0

plane=CreatePlane()
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture plane,grass_tex

While Not KeyDown( 1 )

; Toggle camera fog mode between 0 and 1 when spacebar is pressed
If KeyHit( 57 )=True Then fog_mode=1-fog_mode : CameraFogMode camera,fog_mode

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.05
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.05

RenderWorld

Text 0,0,"Use cursor keys to move about the infinite plane"
Text 0,20,"Press spacebar to toggle between CameraFogMode 0/1"
If fog_mode=False Then Text 0,40,"CameraFogMode 0" Else Text 0,40,"CameraFogMode 1"

Flip

Wend

End
```

CameraFogRange cámara,cerca#,lejos#

Parámetros:
cámara – puntero de la cámara
cerca# - distancia enfrente de la cámara a la que empieza la niebla
lejos# - distancia enfrente de la cámara a la que acaba la niebla

Descripción:
Establece el rango de niebla de la cámara. El parámetro cerca especifica a que distancia enfrente de la cámara empezará el efecto de niebla, todos los objetos 3D antes de este punto no se desvanecerán. El parámetro lejos especifica a que distancia enfrente de la cámara terminará el efecto niebla, todos los objetos 3D situados más allá de este punto desaparecerán.

Ejemplo:

```
; CameraFogRange Example
; ----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,0,1,0

light=CreateLight()
RotateEntity light,90,0,0

plane=CreatePlane()
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture plane,grass_tex

; Set camera fog to 1 (linear fog)
CameraFogMode camera,1

; Set intial fog range value
fog_range=10

While Not KeyDown( 1 )

; If square brackets keys pressed then change fog range value
If KeyDown( 26 )=True Then fog_range=fog_range-1
If KeyDown( 27 )=True Then fog_range=fog_range+1

; Set camera fog range
CameraFogRange camera,1,fog_range

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.05
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.05

RenderWorld

Text 0,0,"Use cursor keys to move about the infinite plane"
Text 0,20,"Press [ or ] to change CameraFogRange value"
Text 0,40,"CameraFogRange camera,1,"+fog_range

Flip

Wend

End
```

CameraFogColor cámara,rojod#,verde#,azul#

Parámetros:

cámara – puntero de la cámara
rojo# - valor rojo de la niebla
verde# - valor verde de la niebla
azul# - valor azul de la niebla

Descripción:

Establece el color de la niebla de la cámara.

Ejemplo:

```
plane=CreatePlane()
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture plane,grass_tex

; Set camera fog to 1 (linear fog)
CameraFogMode camera,1
```

```
; Set camera fog range
CameraFogRange camera,1,10

; Set initial fog colour values
red#=0
green#=0
blue#=0

While Not KeyDown( 1 )

; Change red, green, blue values depending on key pressed
If KeyDown( 2 )=True And red#>0 Then red#=red#-1
If KeyDown( 3 )=True And red#<255 Then red#=red#+1
If KeyDown( 4 )=True And green#>0 Then green#=green#-1
If KeyDown( 5 )=True And green#<255 Then green#=green#+1
If KeyDown( 6 )=True And blue#>0 Then blue#=blue#-1
If KeyDown( 7 )=True And blue#<255 Then blue#=blue#+1

; Set camera fog color using red, green, blue values
CameraFogColor camera,red#,green#,blue#

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.05
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.05

RenderWorld

Text 0,0,"Use cursor keys to move about the infinite plane"
Text 0,20,"Press keys 1-6 to change CameraFogColor red#,green#,blue# values
Text 0,40,"Fog Red: "+red#
Text 0,60,"Fog Green: "+green#
Text 0,80,"Fog Blue: "+blue#

Flip

Wend

End
```

CameraViewport cámara,x,y,ancho,alto

Parámetros:

cámara – puntero de la cámara
x – coordenada x de la esquina superior izquierda del campo de visión
y – coordenada y de la esquina superior izquierda del campo de visión
ancho – ancho del campo de visión
alto – alto del campo de visión

Descripción:

Establece el tamaño y posición del campo de visión de la cámara. El campo de visión de la cámara es el área de la pantalla 2D en el que se muestran los gráficos 3D como los ve la cámara. Establecer el campo de visión de la cámara permite conseguir efectos de espejo.

Ejemplo:

```
; CameraViewport Example
; ----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

; Create first camera
cam1=CreateCamera()

; Set the first camera's viewport so that it fills the top half of the camera
CameraViewport cam1,0,0,GraphicsWidth(),GraphicsHeight()/2

; Create second camera
```

```
cam2=CreateCamera()

; Set the second camera's viewport so that it fills the bottom half of the camera
CameraViewport cam2,0,GraphicsHeight()/2,GraphicsWidth(),GraphicsHeight()/2

light=CreateLight()
RotateEntity light,90,0,0

plane=CreatePlane()
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture plane,grass_tex
PositionEntity plane,0,-1,0

While Not KeyDown( 1 )

If KeyDown( 205 )=True Then TurnEntity cam1,0,-1,0
If KeyDown( 203 )=True Then TurnEntity cam1,0,1,0
If KeyDown( 208 )=True Then MoveEntity cam1,0,0,-0.05
If KeyDown( 200 )=True Then MoveEntity cam1,0,0,0.05

RenderWorld

Text 0,0,"Use cursor keys to move the first camera about the infinite plane"

Flip

Wend

End
```

CameraClsMode cámara,cls_color,cls_zbuffer

Parámetros:
cámara – puntero de la cámara
cls_color - true para limpiar el color del buffer, false para no hacerlo
cls_zbuffer - true para limpiar el color del z-buffer, false para no hacerlo

Descripción:
Establece el modo de limpiar la cámara.

CameraClsColor cámara,rojo#,verde#,azul#

Parámetros:
cámara – puntero de la cámara
rojo# - valor rojo del color de fondo de la cámara
verde# - valor verde del color de fondo de la cámara
azul# - valor azul del color de fondo de la cámara

Descripción:
Establece el color de fondo de la cámara. Por defecto es 0,0,0.

CameraRange cámara,cerca#,lejos#

Parámetros:
cámara – puntero de la cámara
cerca – distancia enfrente de la cámara en que los objetos 3D empiezan a dibujarse
lejos - distancia enfrente de la cámara en que los objetos 3D dejan de dibujarse

Descripción:
Establece el rango de la cámara. Intenta hacer y mantener el radio de lejos/cerca lo más pequeño posible para optimizar el rendimiento del z-buffer. Por defecto es 1,1000.

CameraZoom cámara,zoom#

Parámetros:
cámara – puntero de la cámara
zoom# - factor de zoom de la cámara

Descripción:
Establece el factor de zoom de la cámara. Por defecto es 1.

CameraPick (cámara,viewport_x#,viewport_y#)

Parámetros:
cámara – puntero de la cámara
viewport_x# - coordenada x del campo de visión 2D
viewport_z# - coordenada z del campo de visión 2D

Descripción:
Escoge el ente posicionado en las coordenadas del campo de visión especificadas. Devuelve el ente escogido, o 0 si no hay ninguno allí. Un ente debe tener su EntityPickMode establecido a un valor distinto de 0 para poder ser "escogido".

PickedX# ( )

Parámetros:
Ninguno.

Descripción:
Devuelve la coordenada x del mundo del comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick. La coordenada representa el punto exacto donde algo fue "seleccionado". Ver también: PickedY, PickedZ.

PickedY# ( )

Parámetros:
Ninguno.

Descripción:
Devuelve la coordenada y del mundo del comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick. La coordenada representa el punto exacto en donde algo ha sido "seleccionado". Ver también: PickedX, PickedZ.

PickedZ# ( )

Parámetros:
Ninguno.

Descripción:
Devuelve la coordenada z del mundo del comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick. La coordenada representa el punto exacto donde algo ha sido "seleccionado". Ver también: PickedX, PickedY.

PickedNX ( )

Parámetros:
Ninguno.

Descripción:
Devuelve el componente x de la normal del comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick.

PickedNY ( )

Parámetros:
Ninguno.

Descripción:
Devuelve el componente y de la normal del comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick.

PickedNZ ( )

Parámetros:
Ninguno.

Descripción:
Devuelve el componente z de la normal del comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick.

PickedTime ( )

Parámetros:
Ninguno.

Descripción:
Devuelve el tiempo que llevó calcular el comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick.

PickedEntity ( )

Parámetros:
Ninguno.

Descripción:
Devuelve el ente "seleccionado" por el comando Pick ejecutado más recientemente. Puede haber sido CameraPick, EntityPick o LinePick. Devuelve 0 si no se ha seleccionado ningún ente.

PickedSurface ( )

Parámetros:
Ninguno.

Descripción:
Devuelve el número de índice de una superficie que fue "seleccionada" con el comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick.

PickedTriangle ( )

Parámetros:
Ninguno.

Descripción:
Devuelve el número de índice del triángulo que fue "seleccionado" con el comando Pick más recientemente ejecutado. Puede haber sido CameraPick, EntityPick o LinePick.

CameraProject cámara,x#,y#,z#

Parámetros:
cámara – puntero de la cámara
x# - coordenada x del mundo
y# - coordenada y del mundo
z# - coordenada z del mundo

Descripción:
Proyecta las coordenadas z,y,z del mundo en la pantalla 2D.

ProjectedX# ( )

Parámetros:
Ninguno.

Descripción:
Devuelve la coordenada x del punto de vista del comando CameraProject más recientemente ejecutado.

ProjectedY# ( )

Parámetros:
Ninguno.

Descripción:
Devuelve la coordenada y del punto de vista del comando CameraProject más recientemente ejecutado.

ProjectedZ# ( )

Parámetros:
Ninguno.

Descripción:
Devuelve la coordenada z del punto de vista del comando CameraProject más recientemente ejecutado.

EntityInView (ente,cámara)

Parámetros:
ente – puntero del ente
cámara – puntero de la cámara

Descripción:
Devuelve true si el ente especificado es visible a la cámara especificada. Si el ente es una malla, se comprobará su caja limitada en la visibilidad. Para todos los demás tipos de ente, sólo se comprobará su centro de posición.

LUZ

CreateLight ([tipo][,padre])

Parámetros:
tipo (opcional) – tipo de luz
1: direccional
2: punto
3: spot
padre (opcional) – ente padre de la luz

Descripción:
Crea una luz. La luz funciona afectando al color de los vértices dentro del rango de la luz. Es necesario crear al menos una luz si se desea usar gráficos 3D, de otro modo todo parecerá soso. El parámetro opcional tipo permite especificar el tipo de luz que se desea crear. Un valor 1 crea una luz direccional. Esto funciona de forma similar a un sol brillando sobre una casa. Todos los muros situados en una cierta posición se iluminarán igual. La intensidad de la iluminación dependerá del ángulo en que la luz incida sobre ellos. Un valor 2 crea un punto de luz. Esto funciona como una bombilla iluminando dentro de una casa, comenzando en un punto central y desvaneciéndose gradualmente hacia el exterior. Un valor 3 crea una luz spot. Esto es un cono de luz. Funciona de forma similar a una antorcha alumbrando en una casa. Comienza con un ángulo interior de luz y después se extiende hacia un ángulo exterior de luz. El parámetro opcional padre permite especificar un ente padre para la luz de forma que si el padre se mueve, el hijo se moverá con él. Aunque esto solo funciona en un sentido, utilizar un comando de movimiento con el hijo no tendrá efecto sobre el padre. Especificar un ente padre situará la luz creada en la posición 0,0,0 en lugar de en la posición del ente padre.

LightRange luz,rango#

Parámetros:
luz – puntero de la luz
rango – rango de la luz

Descripción:
Establece el rango de una luz. El rango de una luz es el alcance. Cualquier cosa fuera del rango de la luz no se verá afectada por ella.
El valor es aproximativo, y debería experimentarse con él para obtener los mejores resultados.

LightColor luz,rojo#,verde#,azul#

Parámetros:
luz – puntero de la luz
rojo# - valor rojo de la luz
verde# - valor verde de la luz
azul# - valor azul de la luz

Descripción:
Establece el color de una luz. Un valor r,g,b de 255,255,255 encenderá todos los brillos de la luz. Un valor r,g,b de 0,0,0 no afectará a ningún brillo. Un valor r,g,b de -255,-255,-255 oscurecerá todos los brillos. Se puede entender como una "luz negativa", y es útil para hacer efectos de sombra.

LightConeAngles luz,ángulo_interior#,ángulo_exterior#

Parámetros:
luz – puntero de la luz
ángulo_interior# - ángulo interior del cono
ángulo_exterior# - ángulo exterior del cono

Descripción:
Establece el ángulo del "cono" de un foco de luz. El ángulo del cono de luz por defecto está establecido en 0,90.

PIVOTE

CreatePivot ([padre])

Parámetros:
padre (opcional) – ente padre del pivote

Descripción:
Crea un ente pivote. Un ente pivote es un punto invisible en el espacio 3D que se usa principalmente para actuar como un ente padre de otros entes. El pivote puede usarse entonces para controlar varios entes a la vez, o actuar como un nuevo centro de rotación de otros entes. Para forzar esta relación, se usa EntityParent o se hace uso del parámetro opcional padre del ente disponible en todos los comandos de creación/carga de todos los entes. Realmente, este parámetro está también disponible con el comando CreatePivot si se desea que el pivote tenga un ente padre propio.

SPRITE

CreateSprite ([padre])

Parámetros:
padre (opcional) – ente padre del sprite

Descripción:
Crea un ente sprite y devuelve su puntero. El sprite se posicionará en 0,0,0 y se extenderá de 1,-1 a +1,+1.
Un ente sprite es un flat, objeto 3D cuadrado (que puede convertirse en rectángulo escalándolo). Los sprites tienen dos ventajas. La primera es que consisten en sólo dos polígonos, lo que significa que se pueden usar varios al mismo tiempo. Esto los hace ideales para efectos de partículas y usar juegos 2D y 3D donde se quieran usar varias sprites en la pantalla a la vez. La segunda, se puede asignar un modo de vista usando SpriteViewMode. Por defecto este modo de vista está establecido a 1, lo que significa que el sprite siempre afrontará la cámara. Por lo tanto no te preocupes de que la orientación de la cámara sea relativa al sprite, nunca notarás que son flat; dándoles una textura especial puedes hacer que parezcan una esfera normal. El parámetro opcional padre permite especificar un ente padre para el sprite, con lo cual cuando el padre se mueva el sprite hijo se moverá con él. Aunque esta relación solo se da en un sentido, aplicar un comando de movimiento a un hijo no tendrá efecto sobre el padre. Especificar un ente padre hará que el sprite se cree en la posición 0,0,0 en lugar de en la posición del ente padre. Ver también: LoadSprite.

LoadSprite (archivo_textura$[,flag_textura][,padre])

Parámetros:
archive_ textura$ - nombre del archive de imagen que se usará como sprite

flag_textura (opcional) – flag de la textura:
1: Color
2: Alfa
4: Oculto
8: Mipmapped
16: Clamp U
32: Clamp V
64: mapa de reflexión esférica
padre – padre del ente

Descripción:
Carga un ente sprite, y le asigna una textura. Ver también: CreateSprite, SpriteViewMode.

RotateSprite sprite,ángulo#

Parámetros:
sprite – puntero del sprite
ángulo# - ángulo absoluto de rotación de la sprite

Descripción:
Rota una sprite.

ScaleSprite sprite,escala_x#,escala_y#

Parámetros:
sprite – puntero del sprite
escala_x# - escala de la sprite en x
escala_y# - escala de la sprite en y

Descripción:
Escala una sprite.

HandleSprite sprite,puntero_x#,puntero_y#

Parámetros:
sprite – puntero del sprite. No confundir con HandleSprite – es decir, el puntero usado para la posición de la sprite, en lugar del puntero actual del sprite

Descripción:
Establece el puntero de un sprite. Por defecto es 0,0,0. Un sprite se extiende de 1,-1 a +1,+1.

SpriteViewMode sprite,modo_vista

Parámetros:
sprite – puntero del sprite
modo_vista – modo de vista del sprite
1: fijo (el sprite siempre encara la cámara)
2: libre (el sprite es independiente de la cámara)
3: plano vertical (el sprite siempre encara la cámara, pero se inclina con la cámara)

Descripción:
Establece el modo de vista de un sprite.

MD2

LoadMD2 (archivo_md2$[,padre])

Parámetros:
archivo_md2$ - nombre del archive del md2
padre (opcional) – ente padre del md2

Descripción:
Carga un ente md2 y devuelve su puntero. Una textura md2 tiene que cargarse y aplicarse al md2 de forma separada, de otro modo el md2 aparecerá sin textura. Los Md2 tienen sus propios comandos de configuración, y no funcionan con los comandos normales de animación. El parámetro opcional padre permite especificar un ente padre para el md2, de forma que cuando se mueve el padre el hijo md2 se mueve con él. Aunque esta relación sólo se da en un sentido, aplicar comandos de movimiento a un hijo no afectará al padre. Especificar un ente padre creará el md2 en la posición 0,0,0 en lugar de en la posición del ente padre.

Ejemplo:
```
; LoadMD2 Example
; ---------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Load md2
gargoyle=LoadMD2( "media/gargoyle/gargoyle.md2" )

; Load md2 texture
garg_tex=LoadTexture( "media/gargoyle/gargoyle.bmp" )

; Apply md2 texture to md2
EntityTexture gargoyle,garg_tex

PositionEntity gargoyle,0,-45,100
RotateEntity gargoyle,0,180,0

While Not KeyDown( 1 )
RenderWorld
Flip
Wend

End
```

AnimateMD2 md2[,modo][,velocidad#][,primer_frame][,último_frame]

Parámetros:
md2 – puntero del md2
modo (opcional) – modo de animación
0: parar la animación
1: loop (bucle) de la animación (por defecto)
2: ping-pong de la animación
3: one-shot (una fotografía) de la animación
velocidad# (opcional) – velocidad de la animación. Por defecto es 1.
primer_frame (opcional) - primer frame de la animación. Por defecto es 1.
último_frame# (opcional) - último frame de la animación. Por defecto es el último frame de todas las animaciones md2.

Descripción:
Inicia la animación de un ente md2. El md2 se moverá de un frame al siguiente cuando se llame a UpdateWorld, normalmente una vez en el bucle principal.

Ejemplo:
```
; AnimateMD2 Example
; ------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0
```

```
; Load md2
gargoyle=LoadMD2( "media/gargoyle/gargoyle.md2" )

; Load md2 texture
garg_tex=LoadTexture( "media/gargoyle/gargoyle.bmp" )

; Apply md2 texture to md2
EntityTexture gargoyle,garg_tex

; Animate md2
AnimateMD2 gargoyle,1,0.1,32,46

PositionEntity gargoyle,0,-45,100
RotateEntity gargoyle,0,180,0

While Not KeyDown( 1 )
UpdateWorld
RenderWorld
Flip
Wend

End
```

MD2AnimTime (md2)

Parámetros:
md2 – puntero del md2

Descripción:
Devuelve el tiempo de animación de un modelo md2. El tiempo de animación es el momento exacto en que está el md2 en sus frames de animación. Por ejemplo, si el md2 en un determinado momento se está moviendo entre las frames tercera y cuarta, el MD2AnimTime devolverá un número en la región 3-4.

Ejemplo:

```
Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Load md2
gargoyle=LoadMD2( "media/gargoyle/gargoyle.md2")

; Load md2 texture
garg_tex=LoadTexture( "media/gargoyle/gargoyle.bmp" )

; Apply md2 texture to md2
EntityTexture gargoyle,garg_tex

; Animate md2
AnimateMD2 gargoyle,1,0.1,32,46

PositionEntity gargoyle,0,-45,100
RotateEntity gargoyle,0,180,0

While Not KeyDown( 1 )

UpdateWorld
RenderWorld

; Output current animation frame to screen
Text 0,0,"MD2AnimTime: "+MD2AnimTime( gargoyle )

Flip
Wend
End
```

MD2AnimLength (md2)

Parámetros:
md2 – puntero del md2

Descripción:
Devuelve la longitud de la animación de un modelo md2. La longitud de la animación es el número total de frames de animación en que consiste el archivo md2.

Ejemplo:
```
; MD2AnimLength Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Load md2
gargoyle=LoadMD2( "media/gargoyle/gargoyle.md2")

; Load md2 texture
garg_tex=LoadTexture( "media/gargoyle/gargoyle.bmp" )

; Apply md2 texture to md2
EntityTexture gargoyle,garg_tex

PositionEntity gargoyle,0,-45,100
RotateEntity gargoyle,0,180,0

While Not KeyDown( 1 )

RenderWorld

; Output animation length to screen
Text 0,0,"MD2AnimLength: "+MD2AnimLength( gargoyle )

Flip

Wend

End
```

MD2Animating (md2)

Parámetros:
md2 – puntero del md2

Descripción:
Devuelve 1 (true) si el md2 se está ejecutando actualmente, 0 (false) en caso contrario.

Ejemplo:
```
; MD2Animating Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

; Load md2
```

```
gargoyle=LoadMD2( "media/gargoyle/gargoyle.md2")

; Load md2 texture
garg_tex=LoadTexture( "media/gargoyle/gargoyle.bmp" )

; Apply md2 texture to md2
EntityTexture gargoyle,garg_tex

PositionEntity gargoyle,0,-45,100
RotateEntity gargoyle,0,180,0

While Not KeyDown( 1 )

; Toggle animation stop/start when spacebar pressed
If KeyHit( 57 )=True start=1-start : AnimateMD2 gargoyle,start,0.1,32,46

UpdateWorld
RenderWorld

Text 0,0,"Press spacebar to stop/start md2 animation"

; Output current md2 animation status to screen
Text 0,20,"MD2Animating: "+MD2Animating( gargoyle )

Flip

Wend

End
```

BSP

LoadBSP (archivo$[,ajuste_gama#][,padre])

Parámetros:
archivo$ - nombre de archive del modelo BSP
ajuste_gama# (opcional) – intensidad del mada de luces BSP. Los valores deberían estar en el rando 0-1. Por defecto es 0
padre# (opcional) – ente padre del BSP

Descripción:
Carga un modelo BSP y devuelve su puntero. Algunas puntualizaciones sobre el soporte BSP del Blitz3D:
** Los sombreados no son soportados!*
** Los BSP no se pueden pintar, texturizar, colorear, ocultar, etc. en Blitz.*
** Un modelo BSP es simplemente un ente. Usa los comandos estándar del ente para escalarlo, rotarlo y posicionarlo, y los comandos estándar de colisiones para realizar colisiones con el BSP.*
**Los modelos BSP no se iluminan con cualquier luz ambiental normal, o mediante una iluminación direccional. Esto permite ejecutar la iluminación de los modelos de un juego sin que afecten a la iluminación de los BSP. Aunque, los modelos BSP iluminados por puntos o por luces disponibles.*
**Las texturas para los modelos BSP deben estar en el mismo directorio que el propio archivo BSP.*
El parámetro opcional padre permite especificar un ente padre para el BSP de modo que cuando el padre se mueve el hijo BSP se moverá con el. Aunque esta relación se da en un solo sentido; aplicar comandos de movimiento a un hijo no afectará al padre.
Especificar un ente padre hará que el BSP se cree en la posición 0,0,0, en lugar de en la posición del ente padre.
Ver también: BSPAmbientLight, BSPLighting.

Ejemplo:

```
Graphics3D 640,480

campiv = CreatePivot()
cam = CreateCamera(campiv)
CameraRange cam, 0.1,2000

level=LoadBSP( "nyk3dm1\nyk3dm1.bsp",.8 ) ; load a 'legal' quake3 bsp map
BSPAmbientLight level, 0,255,0 ; make the ambient light green
;BSPLighting level, False ; uncomment this line to turn lightmap off

While Not KeyDown(1) ; if ESCAPE pressed then exit
```

```
RenderWorld:Flip

mys = MouseYSpeed()
If Abs(EntityPitch(cam)+mys) < 75 ; restrict pitch of camera
TurnEntity cam, mys,0,0
EndIf

TurnEntity campiv,0,-MouseXSpeed(),0

If MouseDown(1) Then ; press mousebutton to move forward
TFormVector 0,0,3,cam,campiv
MoveEntity campiv,TFormedX(),TFormedY(),TFormedZ()
EndIf

MoveMouse 320,240 ; centre mouse cursor
Wend
End
```

BSPAmbientLight bsp, rojo#,verde#azul#

Parámetros:
bsp – puntero del BSP
rojo# - valor rojo de la luz ambiental BSP
verde# - valor verde de la luz ambiental BSP
azul# - valor azul de la luz ambiental BSP
Los valores rojo, verde y azul deberían estar en el rango 0-255. El color de la luz ambiental BSP por defecto es 0,0,0.

Descripción:
Establece el color de la luz ambiental de un modelo BSP. Nota que los modelos BSP no usan la configuración normal de la AmbientLight. Puede usarse también para incrementar el brillo de un modelo BSP, pero el efecto no es tan "bueno" como cuando se usa el parámetro ajustar_gama del comando LoadBSP. Ver también: LoadBSP, BSPLighting.

PLANO

CreatePlane ([sub_divs][,padre])

Parámetros:
sub_divs - sub divisiones del plano
padre (opcional) – ente padre del plano

Descripción:
Crea un ente plano y devuelve su puntero. Un ente plano es básicamente un , "suelo" infinito. Se usa en juegos al aire libre donde no se quiere que el jugador vea/alcance los límites del mundo del juego. El parámetro opcional sub_divs especifica las subdivisiones de polígonos que tendrá el plano. Aunque un plano es flat y por lo tanto añadir más polígonos no lo hará más suavizado, añadir más polígonos permitirá más vértices para un tener mayor detalle en los efectos de luz. El parámetro opcional padre permite especificar un ente padre para el plano, que hará que cuando se mueva el padre el plano hijo se moverá con él. Aunque esta relación sólo se da en un sentido, aplicar comandos de movimiento al hijo no tendrá efectos sobre el padre. Especificar un ente padre hará que el plano creado se sitúe en la posición 0,0,0 en lugar de en la posición del ente padre. Ver también: CreateMirror.

Ejemplo:

```
; CreatePlane Example
; ------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,0,1,0

light=CreateLight()
RotateEntity light,90,0,0

; Create plane
plane=CreatePlane()
```

```
grass_tex=LoadTexture( "media/mossyground.bmp" )

EntityTexture plane,grass_tex

While Not KeyDown( 1 )

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.05
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.05

RenderWorld

Text 0,0,"Use cursor keys to move about the infinite plane"

Flip

Wend

End
```

ESPEJO

CreateMirror ([padre])

Parámetros:
padre – ente padre del espejo

Descripción:
Crea un ente espejo y devuelve su puntero. Un ente espejo es básicamente un flat, "suelo" infinito. Este suelo es invisible, excepto que refeja todo lo que hay por encima y por debajo de él. Se usa en juegos en los que se quiere tener un efecto de suelo brillante mostrando un reflejo. Para un efecto de suelo brillante realista intenta combinar un ente espejo con un ente plano texturizado que tenga su nivel alfa a 0.5.
El parámetro opcional padre permite especificar un ente padre del espejo que hace que cuando el padre se mueva el espejo hijo se moverá con el. Aunque esta relación es solo en un sentido, aplicar un comando de movimiento a un hijo no afectará al padre.
Especificar un ente padre situará el espejo a crear en la posición 0,0,0 en lugar de en la posición del ente padre. Ver también: CreatePlane.

Ejemplo:

```
Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,0,1,-5

light=CreateLight()
RotateEntity light,90,0,0

; Create cone
cone=CreateCone(32)
PositionEntity cone,0,2,0

; Create plane
plane=CreatePlane()
grass_tex=LoadTexture( "media/chorme-2.bmp" )
EntityTexture plane,grass_tex
EntityAlpha plane,0.5

mirror=CreateMirror()

While Not KeyDown( 1 )

If KeyDown( 203 )=True Then MoveEntity cone,-0.1,0,0
If KeyDown( 205 )=True Then MoveEntity cone,0.1,0,0
If KeyDown( 208 )=True Then MoveEntity cone,0,-0.1,0
If KeyDown( 200 )=True Then MoveEntity cone,0,0.1,0
If KeyDown( 44 )=True Then MoveEntity cone,0,0,-0.1
If KeyDown( 30 )=True Then MoveEntity cone,0,0,0.1
```

```
RenderWorld

Text 0,0,"Use cursor/A/Z keys to move cone above infinite mirror"

Flip

Wend

End
```

TERRENO

CreateTerrain (tamaño_rejilla[,padre])

Parámetros:
tamaño_rejilla – número de cuadrados de rejilla por cada lado del terreno, y debe ser potencia de 2
padre (opcional) – ente padre del terreno

Descripción:
Crea un ente terreno y devuelve su puntero. El terreno se extiende desde 0,0,0 a tamaño_rejilla,1,tamaño_rejilla. Un terreno es un tipo especial de objeto poligonal que usa un nivel de detalle en tiempo real (LOD) para mostrar paisajes que deberían consistir teóricamente en cerca de un millón de polígonos con solo unos pocos cientos. El modo en que se consigue esto es reorganizando un número determinado de polígonos para mostrar altos niveles de detalle cerca de la vista y bajos niveles lejos. Estas constantes reorganizaciones de polígonos son notables aunque afecta a todos los paisajes LOD. Este efecto se puede reducir de varias maneras como se explicará en otros archivos de ayuda de terrenos. El parámetro opcional padre permite especificar un ente padre para el terreno de forma que cuando el padre se mueva, el hijo también se moverá con él. Aunque esta relación se da sólo en un sentido, aplicar comandos de movimiento al hijo no tendrá efectos sobre el padre. Especificar un ente padre situará el terreno creado en la posición 0,0,0 en lugar de en la posición del ente padre.
Ver también: LoadTerrain.

Ejemplo:

```
; CreateTerrain Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,0,1,0

light=CreateLight()
RotateEntity light,90,0,0

; Create terrain
terrain=CreateTerrain(128)

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex

While Not KeyDown( 1 )

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.05
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.05

RenderWorld

Text 0,0,"Use cursor keys to move about the terrain"

Flip

Wend

End
```

LoadTerrain (archivo$[,padre])

Parámetros:
archivo$ - nombre del archive de imagen que se usará como mapa de altura
padre (opcional) – ente padre del terreno

Descripción:
Carga un terreno desde un archive de imagen y devuelve el puntero del terreno. El canal rojo de la imagen se usa para determinar las alturas. El terreno tiene inicialmente el mismo ancho y profundidad que la imagen, y 1 de alto. Sugerencias para generar buenos terrenos:
** Suavizar o nublar el mapa de altura*
** Reducir la escala en y del terreno*
** Incrementar la escala en x/z del terreno*
** Reducir el rango de la cámara*
Cuando se texturiza un ente, la textura con una escala de 1,1,1 (por defecto) tendrá el mismo tamaño que uno de los cuadrados de la rejilla del terreno. Una textura escalada con el mismo tamaño que el tamaño del mapa de bits (bitmap) usado para cargarlo o el número de cuadrados de rejilla usados para crearlo, tendrá el mismo tamaño que el terreno. El parámetro opcional padre permite especificar un ente padre para el terreno de forma que cuando el padre se mueve, el terreno hijo se moverá con él. Aunque esta relación sólo se da en un sentido, aplicar comandos de movimiento a un hijo no afectará al padre. Especificar un ente padre hará que el terreno se cree en la posición 0,0,0 en lugar de hacerlo en la posición del ente padre. Ver también: CreateTerrain.

Ejemplo:

```
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,1,1,1

light=CreateLight()
RotateEntity light,90,0,0

; Load terrain
terrain=LoadTerrain( "media/height_map.bmp" )

; Set terrain detail, enable vertex morphing
TerrainDetail terrain,4000,True

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex,0,1

While Not KeyDown( 1 )

If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.1
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.1

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

terra_y#=TerrainY(terrain,x#,y#,z#)+5

PositionEntity camera,x#,terra_y#,z#
RenderWorld
Text 0,0,"Use cursor keys to move about the terrain"

Flip
Wend
End
```

TerrainSize (terreno)

Parámetros:
terreno – puntero del terreno

Descripción:
Devuelve el tamaño de la rejilla utilizado para crear el terreno.

Ejemplo:
```
; TerrainSize Example
; -------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,1,1,1

light=CreateLight()
RotateEntity light,90,0,0

; Load terrain
terrain=LoadTerrain( "media/height_map.bmp" )

; Set terrain detail, enable vertex morphing
TerrainDetail terrain,4000,True

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex,0,1

While Not KeyDown( 1 )

If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.1
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.1

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

terra_y#=TerrainY(terrain,x#,y#,z#)+5

PositionEntity camera,x#,terra_y#,z#

RenderWorld

Text 0,0,"Use cursor keys to move about the terrain"

; Output terrain size to screen
Text 0,20,"Terrain Size: "+TerrainSize(terrain)

Flip

Wend

End
```

TerrainDetail terreno,nivel_detalle[,vértice_morph]

Parámetros:
terreno – puntero del terreno
nivel_detalle – nivel de detalle del terreno
vértice_morph (opcional) - true para activar el morphing (una imagen se convierte gradualmente en otra) del vértice del terreno. Por defecto es false.

Descripción:
Establece el nivel de detalle del terreno. Es el número de triángulos usados para representar el terreno. El valor típico es 2000.
El parámetro opcional vertex_morph especifica si se activa el morphing del vértice. Se recomienda establecerlo a true, reducirá la visibilidad de LOD 'pop-in' ("echar un vistazo").

Ejemplo:

```
Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,1,1,1

light=CreateLight()
RotateEntity light,90,0,0

; Load terrain
terrain=LoadTerrain( "media/height_map.bmp" )

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex,0,1

; Set terrain detail value
terra_detail=4000

; Set vertex morph value
vertex_morph=True

While Not KeyDown( 1 )

; Change terrain detail value depending on key pressed
If KeyDown(26) Then terra_detail=terra_detail-10
If KeyDown(27) Then terra_detail=terra_detail+10

; Toggle vertex morphing on/off when spacebar is pressed
If KeyHit(57)=True Then vertex_morph=1-vertex_morph

; Set terrain detail, vertex morphing
TerrainDetail terrain,terra_detail,vertex_morph

If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.1
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.1

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

terra_y#=TerrainY(terrain,x#,y#,z#)+5
```

```
PositionEntity camera,x#,terra_y#,z#

RenderWorld

Text 0,0,"Use cursor keys to move about the terrain"
Text 0,20,"Use [ and ] keys to change terrain detail level"
Text 0,40,"Press spacebar to enable/disable vertex morphing"
Text 0,60,"Terrain Detail: "+terra_detail
If vertex_morph=True Then Text 0,80,"Vertex Morphing: True" Else Text 0,80,"Vertex Morphing: False"

Flip

Wend

End
```

TerrainShading terreno,activo

Parámetros:
terreno – puntero del terreno
activo – true para activar el sombreado del terreno, false para desactivarlo.

Descripción:
Activa o desactiva el sombreado del terreno. Los terrenos sombreados son un poco más lentos que los terrenos no sombreados, y en algunos casos puede incrementar la visibilidad de LOD 'pop-in' ("echar un vistazo"). Aunque, la opción está ahí para tener terrenos sombreados si deseas hacerlo así. El sombreado del terreno está desactivo por defecto.

Ejemplo:

```
; TerrainShading Example
; ----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,1,1,1

light=CreateLight()
RotateEntity light,45,0,0

; Load terrain
terrain=LoadTerrain( "media/height_map.bmp" )

; Set terrain detail, enable vertex morphing
TerrainDetail terrain,4000,True

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex,0,1

While Not KeyDown( 1 )

; Toggle terrain shading value between 0 and 1 when spacebar is pressed
If KeyHit(57)=True Then terra_shade=1-terra_shade

; Enable/disable terrain shading
TerrainShading terrain,terra_shade

If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1
```

```
If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.1
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.1

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

terra_y#=TerrainY(terrain,x#,y#,z#)+5

PositionEntity camera,x#,terra_y#,z#

RenderWorld

Text 0,0,"Use cursor keys to move about the terrain"
Text 0,20,"Press spacebar to toggle between TerrainShading True/False"
If terra_shade=True Then Text 0,40,"TerrainShading: True" Else Text 0,40,"TerrainShading: False"

Flip

Wend

End
```

TerrainHeight# (terreno,regilla_x,regilla_z)

Parámetros:
terreno – puntero del terreno
regilla_x – coordenada x de la rejilla del terreno
regilla_z – coordenada z de la rejilla del terreno

Descripción:
Devuelve la altura del terreno en las coordenadas x,z de la rejilla del terreno. El valor devuelto está entre el rango 0 y 1. Ver también: TerrainY.

ModifyTerrain terreno,rejilla_x,rejilla_z,altot#[,tiemporeal]

Parámetros:
terreno – puntero del terreno
rejilla_x – coordenada x de la rejilla del terreno
rejilla_y – coordenada y de la rejilla del terreno
altot# - alto del punto del terreno. Debería estar en el rango 0-1.
tiemporeal (opcional) - true para modificar el terreno inmediatamente. False para modificar el terreno la próxima vez que se llame a RenderWorld. Por defecto es false.

Descripción:
Establece el alto de un punto sobre el terreno.

TerrainX# (terreno,x#,y#,z#)

Parámetros:
terreno – puntero del terreno
x# - coordenada x del mundo
y# - coordenada y del mundo
z# - coordenada z del mundo

Descripción:
Devuelve la coordenada x interpolada (insertada) en el terreno. Ver también: TerrainY, TerrainZ.

Ejemplo:

```
Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,1,1,1
```

```
light=CreateLight()
RotateEntity light,90,0,0

; Load terrain
terrain=LoadTerrain( "media/height_map.bmp" )

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex,0,1

; Set terrain detail, enable vertex morphing
TerrainDetail terrain,4000,True

While Not KeyDown( 1 )

If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.1
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.1

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

terra_y#=TerrainY(terrain,x#,y#,z#)+5

PositionEntity camera,x#,terra_y#,z#

RenderWorld

Text 0,0,"Use cursor keys to move about the terrain"

; Output TerrainX value to screen
Text 0,20,"TerrainX: "+TerrainX(terrain,x#,terra_y#,z#)

Flip

Wend

End
```

TerrainY# (terreno,x#,y#,z#)

Parameters:
terreno – puntero del terreno
x# - coordenada x del mundo
y# - coordenada y del mundo
z# - coordenada z del mundo

Descripción:
Devuelve la coordenada y interpolada (intercalada) en el terreno. Básicamente, obtiene la altura del terreno. Ver también: TerrainX, TerrainZ, TerrainHeight.

Ejemplo:

```
Graphics3D 640,480
SetBuffer BackBuffer()
```

```
camera=CreateCamera()
PositionEntity camera,1,1,1

light=CreateLight()
RotateEntity light,90,0,0

; Load terrain
terrain=LoadTerrain( "media/height_map.bmp" )

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex,0,1

; Set terrain detail, enable vertex morphing
TerrainDetail terrain,4000,True

While Not KeyDown( 1 )

If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.1
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.1

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

terra_y#=TerrainY(terrain,x#,y#,z#)

PositionEntity camera,x#,terra_y#+5,z#

RenderWorld

Text 0,0,"Use cursor keys to move about the terrain"

; Output TerrainY value to screen
Text 0,20,"TerrainY: "+terra_y#

Flip

Wend

End
```

TerrainZ# (terreno,x#,y#,z#)

Parámetros:
terreno – puntero del terreno
x# - coordenada x del mundo
y# - coordenada y del mundo
z# - coordenada z del mundo

Descripción:
Devuelve la coordenada z interpolada (insertada) en el terreno. Ver también: TerrainX, TerrainY.

Ejemplo:

```
; TerrainZ Example
; ----------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
PositionEntity camera,1,1,1

light=CreateLight()
RotateEntity light,90,0,0

; Load terrain
terrain=LoadTerrain( "media/height_map.bmp" )

; Scale terrain
ScaleEntity terrain,1,50,1

; Texture terrain
grass_tex=LoadTexture( "media/mossyground.bmp" )
EntityTexture terrain,grass_tex,0,1

; Set terrain detail, enable vertex morphing
TerrainDetail terrain,4000,True

While Not KeyDown( 1 )

If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1

If KeyDown( 205 )=True Then TurnEntity camera,0,-1,0
If KeyDown( 203 )=True Then TurnEntity camera,0,1,0
If KeyDown( 208 )=True Then MoveEntity camera,0,0,-0.1
If KeyDown( 200 )=True Then MoveEntity camera,0,0,0.1

x#=EntityX(camera)
y#=EntityY(camera)
z#=EntityZ(camera)

terra_y#=TerrainY(terrain,x#,y#,z#)+5

PositionEntity camera,x#,terra_y#,z#

RenderWorld

Text 0,0,"Use cursor keys to move about the terrain"

; Output TerrainZ value to screen
Text 0,20,"TerrainZ: "+TerrainZ(terrain,x#,terra_y#,z#)

Flip

Wend

End
```

Oyente/Sonido

CreateListener (padre[,rolloff_factor#][,doppler_scale#][,distance_scale#])

Parámetros:
padre – ente padre del oyente. Debe especificarse un ente padre, típicamente una cámara, para "transportar" al oyente.
rolloff_factor# (opcional) – el porcentaje al cual el volumen disminuye con la distancia. Por defecto es 1.
doppler_scale# (opcional) – la gravedad del efecto doppler (cambio de onda). Por defecto es 1.
distance_scale# (opcional) – distancia escalas artificialmente. Por defecto es 1.

Descripción:
Crea un ente oyente y devuelve su puntero. Normalmente solo se soporta un solo oyente.

Load3DSound (archivo$)

Parámetros:
archivo$ - nombre del archive de sonido que se cargará y que se usará como sonido 3D

Descripción:
Carga un sonido y devuelve su puntero para usarlo con EmitSound.

EmitSound sonido,ente

Parámetros:
sonido – puntero del sonido
ente – puntero del ente

Descripción:
Emite un sonido unido al ente especificado y devuelve el canal del sonido. El sonido debe haberse cargado usando Load3DSound para efectos 3D.

MOVIMIENTO DE ENTIDADES

ScaleEntity ente,escala_x#,escala_y#,escala_z#,[,global]

Parámetros:
Ente – nombre del ente a escalar
escala_x# - tamaño x del ente
escala_y# - tamaño y del ente
escala_z# - tamaño z del ente
global (opcional) -

Descripción:
Escala un ente para que esté en un tamaño absoluto. Los valores de la escala 1,1,1 son el tamaño por defecto cuando creamos/cargamos entes. Los valores de escala 2,2,2 doblarán el tamaño de un ente. Los valores de escala 0,0,0 harán desaparecer un ente. Los valores de escala de menos de 0,0,0 invertirán un ente y lo harán más grande.

Ejemplo:
```
; ScaleEntity Example
; -------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,5

; Set scale values so that cone is default size to begin with
x_scale#=1
y_scale#=1
z_scale#=1
```

```
While Not KeyDown( 1 )

; Change scale values depending on the key pressed
If KeyDown( 203 )=True Then x_scale#=x_scale#-0.1
If KeyDown( 205 )=True Then x_scale#=x_scale#+0.1
If KeyDown( 208 )=True Then y_scale#=y_scale#-0.1
If KeyDown( 200 )=True Then y_scale#=y_scale#+0.1
If KeyDown( 44 )=True Then z_scale#=z_scale#-0.1
If KeyDown( 30 )=True Then z_scale#=z_scale#+0.1

; Scale cone using scale values
ScaleEntity cone,x_scale#,y_scale#,z_scale#

RenderWorld

Text 0,0,"Use cursor/A/Z keys to scale cone"
Text 0,20,"X Scale: "+x_scale#
Text 0,40,"Y Scale: "+y_scale#
Text 0,60,"Z Scale: "+z_scale#

Flip

Wend

End
```

PositionEntity ente,x#,y#,z#,[,global]

Parámetros:

ente – nombre del ente a posicionar
x# - coordenada x donde el ente será posicionado
y# - coordenada y donde el ente será posicionado
z# - coordenada z donde el ente será posicionado
global (opcional) - true si la posición debería ser relativa a 0,0,0 en lugar de a la posición del ente padre. False por defecto.

Descripción:

Posiciona un ente en una posición absoluta en el espacio 3D. El ente se posiciona usando un sistema de coordenadas x,y,z. X,y y z tienen sus propios ejes, y cada eje tiene su propio valor. Especificando un valor para cada eje, se puede posicionar un ente en cualquier parte del espacio 3D. 0,0,0 es el centro del espacio 3D, y si una cámara está dirigida hacia una posición z positiva por defecto, situando un ente con un valor z superior a 0 lo hará aparecer enfrente de la cámara, mientras que un valor negativo de z lo haría desaparecer detrás de la cámara. Cambiar el valor x lo movería lateralmente, y cambiar el valor y lo movería arriba/abajo. Por supuesto, la dirección en la que aparece un ente para moverse es relativa a la posición y orientación de la cámara. Ver también: MoveEntity, TranslateEntity.

Ejemplo:

```
; PositionEntity Example
; ----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )

; Set position values so that cone is positioned in front of camera, so we can see it to begin with
x#=0
y#=0
z#=10

While Not KeyDown( 1 )

; Change position values depending on key pressed
If KeyDown( 203 )=True Then x#=x#-0.1
If KeyDown( 205 )=True Then x#=x#+0.1
If KeyDown( 208 )=True Then y#=y#-0.1
If KeyDown( 200 )=True Then y#=y#+0.1
```

```
If KeyDown( 44 )=True Then z#=z#-0.1
If KeyDown( 30 )=True Then z#=z#+0.1

; Position cone using position values
PositionEntity cone,x#,y#,z#

RenderWorld

Text 0,0,"Use cursor/A/Z keys to change cone position"
Text 0,20,"X Position: "+x#
Text 0,40,"Y Position: "+y#
Text 0,60,"Z Position: "+z#

Flip

Wend

End
```

MoveEntity ente,x#,y#,z#

Parámetros:

ente – nombre del ente a mover
x# - número en x en que se moverá el ente
y# - número en y en que se moverá el ente
z# - número en z en que se moverá el ente

Descripción:

Mueve un ente en relación a su posición y orientación actuales. Lo que significa esto es que el ente se moverá en cualquier dirección. Así por ejemplo, si tienes un personaje del juego que está derecho la primera vez que cargas Blitz3D y quiere moverlo y que permanezca así (por ejemplo torcer solamente a derecha o izquierda) , entonces moverlo en z le hará siempre avanzar o retroceder, moverlo en y le hará siempre subir y bajar, y moverlo en x le hará siempre desplazarse a derecha e izquierda. Ver también: TranslateEntity, PositionEntity.

Ejemplo:

```
; MoveEntity Example
; ------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )

; Move cone in front of camera, so we can see it to begin with
MoveEntity cone,0,0,10

While Not KeyDown( 1 )

; Reset movement values - otherwise, the cone will not stop!
x#=0
y#=0
z#=0

; Change rotation values depending on the key pressed
If KeyDown( 203 )=True Then x#=-0.1
If KeyDown( 205 )=True Then x#=0.1
If KeyDown( 208 )=True Then y#=-0.1
If KeyDown( 200 )=True Then y#=0.1
If KeyDown( 44 )=True Then z#=-0.1
If KeyDown( 30 )=True Then z#=0.1

; Move cone using movement values
MoveEntity cone,x#,y#,z#

; If spacebar pressed then rotate cone by random amount
```

```
If KeyHit( 57 )=True Then RotateEntity cone,Rnd( 0,360 ),Rnd( 0,360 ),Rnd( 0,360 )

RenderWorld

Text 0,0,"Use cursor/A/Z keys to move cone, spacebar to rotate cone by random amount"
Text 0,20,"X Movement: "+x#
Text 0,40,"Y Movement: "+y#
Text 0,60,"Z Movement: "+z#

Flip

Wend

End
```

TranslateEntity ente,x#,y#,z#,[,global]

Parámetros:
ente - nombre del ente a trasladar
x# - cantidad a trasladar en el eje x
y# - cantidad a trasladar en el eje y
z# - cantidad a trasladar en el eje z
global (opcional) -

Descripción:
Traslada un ente en relación a su posición actual y no a su orientación. Lo que significa que un ente se moverá en una cierta dirección a pesar de su orientación. Imagina que tienes un personaje en el juego que quieres que salte en el aire al mismo tiempo que hace un triple salto mortal. Trasladar al personaje un número positivo en "y" significará que el personaje siempre irá directamente hacia arriba en el aire, a pesar de donde estuviera orientado, debido a la acción del salto. Ver también: MoveEntity, PositionEntity.

Ejemplo:

```
; TranslateEntity Example
; -----------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )

; Rotate cone by random amount to demonstrate that TranslateEntity is independent of entity
orientation
RotateEntity cone,Rnd( 0,360 ),Rnd( 0,360 ),Rnd( 0,360 )

; Translate cone in front of camera, so we can see it to begin with
TranslateEntity cone,0,0,10

While Not KeyDown( 1 )

; Reset translation values - otherwise, the cone will not stop!
x#=0
y#=0
z#=0

; Change translation values depending on the key pressed
If KeyDown( 203 )=True Then x#=-0.1
If KeyDown( 205 )=True Then x#=0.1
If KeyDown( 208 )=True Then y#=-0.1
If KeyDown( 200 )=True Then y#=0.1
If KeyDown( 44 )=True Then z#=-0.1
If KeyDown( 30 )=True Then z#=0.1

; Translate sphere using translation values
TranslateEntity cone,x#,y#,z#
```

```
; If spacebar pressed then rotate cone by random amount
If KeyHit( 57 )=True Then RotateEntity cone,Rnd( 0,360 ),Rnd( 0,360 ),Rnd( 0,360 )

RenderWorld

Text 0,0,"Use cursor/A/Z keys to translate cone, spacebar to rotate cone by random amount"
Text 0,20,"X Translation: "+x#
Text 0,40,"Y Translation: "+y#
Text 0,60,"Z Translation: "+z#

Flip

Wend

End
```

RotateEntity ente,pitch#,yaw#,roll#,[,global]

Parámetros:

Ente – nombre de la entidad a rotar
pitch# - ángulo inclinación de la rotación en grados
yaw# - ángulo de giro de la rotación en grados
roll# - ángulo de rotación en grados
global (opcional) – true si el ángulo rotado debería ser relativo a 0,0,0 antes que la orientación del ente padre.. False por defecto.

Descripción:

Rota un ente a fin de situarlo en su orientación absoluta. Pitch es lo mismo que el ángulo x de un ente, y es equivalente a inclinar alante/atrás. Yaw es lo mismo que el ángulo y de un ente, y es equivalente a girar izquierda/derecha. Roll es lo mismo que el ángulo z de un ente, y es equivalente a inclinar izquierda/derecha. Ver también: TurnEntity.

Ejemplo:

```
; RotateEntity Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,5

While Not KeyDown( 1 )

; Change rotation values depending on the key pressed
If KeyDown( 208 )=True Then pitch#=pitch#-1
If KeyDown( 200 )=True Then pitch#=pitch#+1
If KeyDown( 203 )=True Then yaw#=yaw#-1
If KeyDown( 205 )=True Then yaw#=yaw#+1
If KeyDown( 45 )=True Then roll#=roll#-1
If KeyDown( 44 )=True Then roll#=roll#+1

; Rotate cone using rotation values
RotateEntity cone,pitch#,yaw#,roll#

RenderWorld

Text 0,0,"Use cursor/Z/X keys to change cone rotation"
Text 0,20,"Pitch: "+pitch#
Text 0,40,"Yaw : "+yaw#
Text 0,60,"Roll : "+roll#

Flip
Wend
End
```

TurnEntity ente, pitch#,yaw#,roll#,[,global]

Parámetros:
ente - nombre del ente a rotar
pitch# - ángulo x
yaw# - ángulo y
roll# - ángulo z
global (opcional) -

Descripción:
Gira un ente en relación a su orientación actual. Pitch es lo mismo que el ángulo x de un ente, y es equivalente a inclinar alante/atrás. Yaw es lo mismo que el ángulo y de un ente, y es equivalente a girar izquierda/derecha. Roll es lo mismo que el ángulo z de un ente, y es equivalente a inclinar izquierda/derecha. Ver también: RotateEntity.

Ejemplo:
```
; TurnEntity Example
; ------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,5

While Not KeyDown( 1 )

; Reset turn values - otherwise, the cone will not stop turning!
pitch#=0
yaw#=0
roll#=0

; Change movement values depending on the key pressed
If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

; Move sphere using movement values
TurnEntity cone,pitch#,yaw#,roll#

RenderWorld

Text 0,0,"Use cursor/Z/X keys to turn cone"
Text 0,20,"Pitch: "+pitch#
Text 0,40,"Yaw: "+yaw#
Text 0,60,"Roll: "+roll#

Flip
Wend
End
```

PointEntity ente,objetivo[,roll#]

Parámetros:
ente – puntero del ente
objetivo – puntero del ente objetivo
roll# (opcional) – ángulo de rotación del ente

Descripción:
Dirige un ente hacia otro. El parámetro opcional roll permite especificar el ángulo de rotación así como dirigir un ente estableciendo sólo sus ángulos de pitch (inclinación) y yaw. Si se desea que un ente se dirija hacia una posición determinada en lugar de hacia otro ente, simplemente se ha de crear un ente pivote en la posición deseada, apuntando el ente hacia él y liberando después el pívot.

AlignToVector ente,vector_x#,vector_y#,vector_z#,eje

Parámetros:
ente – puntero del ente
vector_x# - vector x
vector_y# - vector y
vector_z# - vector z
eje – eje del ente que será alineado al vector
1: eje x
2: eje y
3: eje z

Descripción:
Alinea el eje de un ente a un vector.

Animación de Entidades

LoadAnimSeq (ente,archivo$)

Parámetros:
ente – puntero del ente
archivo$ - nombre del archivo del objeto 3D animado

Descripción:
Añade una secuencia de animación desde un archive a un ente. Devuelve el número de la secuencia de la animación añadida.

SetAnimKey ente,frame[,pos_key][,rot_key][,scale_key]

Parámetros:
Ente – puntero del ente
frame - frame de la animación que se usa como clave de la animación
pos_key (opcional) - true para incluir la información de la posición del ente cuando establecemos la clave. Por defecto es true.
rot_key (opcional) - true para incluir la información de la rotación del ente cuando establecemos la clave. Por defecto es true.
scale_key (opcional) - true para incluir la información de la escala del ente cuando establecemos la clave. Por defecto es true.

Descripción:
Establece la clave de una animación para el ente especificado en el frame especificado.

AddAnimSeq (ente,longitud)

Parámetros:
ente – puntero del ente
longitud -

Descripción:
Crea una secuencia de animación para un ente. Debe hacerse antes de establecer cualquier tecla de animación mediante SetAnimKey puede usarse en la animación actual. Retorna el número añadido de la secuencia de la animación.

ExtractAnimSeq(ente,primer_frame,último_frame[,sec_animación])

Parámetros:
ente – puntero del ente
primer_frame – primer frame de la secuencia de animación a extraer
último_frame – último frame de la secuencia de animación a extraer
sec_animación (opcional) – secuencia de la animación desde la que extraer. Normalmente es 0, y el valor por defecto es 0.

Descripción:
Este comando permite convertir una animación del estilo de secuencia de animación de series de MD2 en secuencias de animación puras del Blitz, y ejecutarlas usando Animate.

Ejemplo::

```
mesh=LoadAnimMesh( base_mesh$ ) ;anim sequence 0.
 ExtractAnimSeq( mesh,0,30 ) ;anim sequence 1: frames 0...30 are 'run'
 ExtractAnimSeq( mesh,31,40 ) ;anim sequence 2: frames 31...40 are 'jump' etc  etc...

Animate mesh,3,1,2 ;play one-shot jump anim
```

Animate ente,[,modo][,velocidad#][,secuencia][,transición#]

Parámetros:
ente – puntero del ente
modo (opcional) – modo de animación.
0: parar la animación
1: loop (bucle) de la animación (por defecto)
2: ping-pong de la animación
3: one-shot (una fotografía) de la animación
velocidad# (opcional) – velocidad de animación. Por defecto es 1.
secuencia (opcional) – especifica que secuencia de la animación se ejecuta. Por defecto es 0.
transición# (opcional) – usado para el tween entre la posición de rotación actual de los entes y el primer frame de la animación. Por defecto es 0.

Descripción:
Anima un ente. Más información acerca de los parámetros opcionales:
velocidad# - una velocidad negativa ejecutará la animación al revés.
secuencia – Inicialmente, un ente cargado con LoadAnimMesh tendrá una animación de secuencia simple. Se pueden añadir más secuencias usando LoadAnimSeq o AddAnimSeq. Las secuencias de animación están numeradas 0,1,2...etc.
transición# - Un valor de 0 causará un "salto" instantáneo a la primera frame, mientras que valores mayores que 0 causarán una transición más suave.

AnimSeq (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve la secuencia de animación actual del ente especificado.

AnimLength (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve el largo de una secuencia de animación actual del ente especificado.

AnimTime# (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve el tiempo de animación actual de un ente.

Animating (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve true si el ente especificado está siendo animado actualmente.

CONTROL DE ENTIDADES

FreeEntity ente

Parámetros:
ente – puntero del ente

Descripción:
Libera un ente.

CopyEntity (ente[,padre])

Parámetros:
ente – puntero del ente
padre (opcional) – el ente padre del ente copiado

Descripción:
Crea una copia de un ente y devuelve el puntero del ente copiado. Si se especifica un ente padre, el ente copiado se creará en la posición del ente padre. De lo contrario se creará en 0,0,0.

EntityColor ente,rojo#,verde#,azul#

Parámetros:
ente – puntero del ente
rojo# - valor rojo del ente
verde# - valor verde del ente
azul# - valor azul del ente

Descripción:
Establece el color de un ente. Los valores verde, rojo y azul deberían estar en el rango 0-255. El color por defecto del ente es 255,255,255.

Ejemplo:
```
; EntityColor Example
; -------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Set initial entity color values
red#=255
green#=255
blue#=255

While Not KeyDown( 1 )

; Change red, green, blue values depending on key pressed
If KeyDown( 2 )=True And red#>0 Then red#=red#-1
If KeyDown( 3 )=True And red#<255 Then red#=red#+1
If KeyDown( 4 )=True And green#>0 Then green#=green#-1
If KeyDown( 5 )=True And green#<255 Then green#=green#+1
If KeyDown( 6 )=True And blue#>0 Then blue#=blue#-1
If KeyDown( 7 )=True And blue#<255 Then blue#=blue#+1

; Set entity color using red, green, blue values
EntityColor cube,red#,green#,blue#
```

```
TurnEntity cube,0.1,0.1,0.1

RenderWorld

Text 0,0,"Press keys 1-6 to change EntityColor red#,green#,blue# values
Text 0,20,"Entity Red: "+red#
Text 0,40,"Entity Green: "+green#
Text 0,60,"Entity Blue: "+blue#

Flip

Wend

End
```

EntityAlpha ente,alfa#

Parámetros:
ente – puntero del ente
alfa# - nivel de alfa del ente

Descripción:
Establece el nivel de alfa de un ente. El valor alfa# debería estar en el rango 0-1. El valor por defecto del alfa del ente está establecido a 1. El nivel de alfa es cómo de transparente es un ente. Un valor de 1 significará que el ente no es transparente, es decir, opaco. Un valor de 0 significará que el ente es completamente transparente, es decir, invisible. Los valores entre 0 y 1 causarán una variación en la transparencia de acuerdo con el número, se usa para imitar la apariencia de objetos como el cristal y el hielo. Un valor de alfa de un ente de 0 se usa especialmente en Blitz3D, pues no se renderizarán los entes con dicho valor, pero si se involucrarán los entes en las colisiones. No es lo mismo que HideEntity, que no involucra los entes en las colisiones.

EntityShininess ente,brillo#

Parámetros:
ente – puntero del ente
brillo# - brillo del ente

Descripción:
Establece el brillo de reflejo de un ente. El valor de brillo# debería estar en el rango 0-1. El brillo por defecto está establecido en 0. El brillo es cúanto brillan ciertas áreas de un objeto cuando una luz incide directamente sobre ellas. Establecer un valor de brillo de 1 para la mitad superior de una esfera, combinado con la creación de una luz brillando en su dirección, dará la apariencia de una blola de billar brillante.

EntityTexture ente,textura[,frame][,índice]

Parámetros:
ente – puntero del ente
textura – puntero de la textura
frame (opcional) - frame de la textura
index (opcional) – índice de la capa a ser texturizada

Descripción:
Establece la textura de un ente. El parámetro opcional frame especifica qué frame de la animación de la textura, si existe, debería usarse como textura. Por defecto es 0. El parámetro opcional índice especifica qué capa de la textura debería usarse. Hay cuatro capas de textura disponibles, 0-3 inclusive. Por defecto es 0.

Ejemplo:

```
; EntityTexture Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()

light=CreateLight()
```

```
RotateEntity light,90,0,0

cube=CreateCube()
PositionEntity cube,0,0,5

; Load texture
tex=LoadTexture( "media/b3dlogo.jpg" )

; Texture entity
EntityTexture cube,tex

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cube,pitch#,yaw#,roll#

RenderWorld
Flip

Wend

End
```

EntityBlend ente,matiz

Parámetros:
ente – puntero del ente
matiz – modo de matiz del ente
0: desactivar textura
1: alfa
2: multiplicar (por defecto)
3: añadir

Descripción:
Establece el modo de matiz de un ente.

EntityFX ente,fx

Parámetros:
ente – puntero del ente
fx -
1: brillo completo
2: usar el color de los vértices en lugar del color del pincel
4: flatshaded
8: desactivar niebla

Descripción:
Establece los efectos de miscelánea de un ente. Se pueden añadir flags para combinar dos o más efectos. Por ejemplo, especificando un flag de 3 (1+2) se obtendrá un brillo completo y un pincel con los colores de los vértices.

EntityAutoFade ente,cerca#,lejos#

Parámetros:
ente – puntero del ente
cerca# - distancia enfrente de la cámara a la cual en ente comenzará a desvanecerse
lejos# - distancia enfrente de la cámara a la cual el ente dejará de desvanecerse (y se hará invisible)

Descripción:
Activa el auto desvanecimiento de un ente. Esto causará que el nivel de alfa del ente se ajuste a las distancias entre cerca y lejos para crear el efecto de desvanecimiento.

PaintEntity ente,pincel

Parámetros:
ente – puntero del ente
pincel – puntero del pincel

Descripción:
Pinta un ente con un pincel. La razón de usar PaintEntity para aplicar propiedades específicas a un ente usando un pincel en lugar de utilizar EntityTexture, EntityColor, EntityShininess etc, es que se puede predefinir un pincel, y pintar entes una y otra vez de nuevo usando un solo comando en lugar de un montón por separado.

EntityOrder ente,orden

Parámetros:
ente – puntero del ente
orden – orden en el que se dibujará un ente

Descripción:
Establece el orden de dibujo de un ente. Un orden de 0 significará que un ente se dibujará normalmente. Un valor mayor de 0 significará que un ente se dibujará primero, antes que cualquier otro. Un valor menor de 0 significará que el ente se dibujará más tarde, después que cualquier otro. Establecer un orden de ente a un número distinto de 0 tambien desactiva el z-buffering del ente, por lo tanto sólo debería usarse para entes simples convexos, como cajas de cielo, sorites, etc. EntityOrder afecta al ente especificado, pero no afecta a ninguno de sus hijos, si existen.

ShowEntity ente

Parámetros:
ente – puntero del ente

Descripción:
Muestra un ente. Lo opuesto totalmente a HideEntity. Una vez que se ha ocultado un ente usando HideEntity, se usa ShowEntity para hacerlo visible e implicarlo en las colisiones de nuevo. Los entes se muestran por defecto después de crearlos/cargarlos, por lo tanto solo se necesitaría usar ShowEntity después de haber usado HideEntity. ShowEntity afecta al ente especificado y a todos sus hijos (entes heredados), si es que existen.

HideEntity ente

Parámetros:
ente – puntero el ente

Descripción:
Oculta un ente, por tanto no será visible, y no se verá involucrado en las colisiones. El propósito principal de ocultar un ente es permitir crear entes al principio del programa, ocultarlos, copiarlos y mostrarlos cuando sean necesarios en el juego principal. Esto es más eficaz que crear entes a mitad del juego. Si se desea ocultar un ente y que no sea visible, pero que si esté involucrado en las colisiones, debe usarse EntityAlpha 0 en lugar de este commando. Esto hará que un ente sea completamente transparente. HideEntity afecta al ente especificado y a todos sus entes hijos, si existen.

NameEntity ente,nombre$

Parámetros:
ente – puntero del ente
nombre$ - nombre del ente

Descripción:
Establece un nombre de ente.

EntityParent ente,padre[,global]

Parámetros:
ente – puntero del ente
padre – puntero del ente padre
global (opcional) - true para que el ente hijo retenga su posición global y orientación. Por defecto es true.

Descripción:
Une un ente a un padre. El padre puede ser 0, en cuyo caso en ente no tendrá padre.

GetParent (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve el padre de un ente.

ESTADO DE ENTIDADES

EntityX# (ente[,global])

Parámetros:
ente – nombre del ente del que se devolverá la coordenada x
global (opcional) - true si la coordenada x devuelta debería ser relativa a 0,0,0 antes que a la posición del ente padre. Por defecto es false.

Descripción:
Devuelve la coordenada x de un ente.

Example:
```
Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,10

While Not KeyDown( 1 )

x#=0
y#=0
z#=0

If KeyDown(203)=True Then x#=-0.1
If KeyDown(205)=True Then x#=0.1
If KeyDown(208)=True Then y#=-0.1
If KeyDown(200)=True Then y#=0.1
If KeyDown(44)=True Then z#=-0.1
If KeyDown(30)=True Then z#=0.1

MoveEntity cone,x#,y#,z#

RenderWorld

Text 0,0,"Use cursor/A/Z keys to move cone"

; Return entity x position of cone
Text 0,20,"X Position: "+EntityX#( cone )

Flip
Wend
End
```

EntityY# (ente[,global])

Parámetros:
ente – nombre del ente del que se devolverá la coordenada y
global (opcional) - true si la coordenada y devuelta debería ser relativa a 0,0,0 antes que a la posición del ente padre. Por defecto es false

Descripción:
Devuelve la coordenada y de un ente.

Ejemplo:
```
; EntityY Example
; ---------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,10

While Not KeyDown( 1 )

x#=0
y#=0
z#=0

If KeyDown(203)=True Then x#=-0.1
If KeyDown(205)=True Then x#=0.1
If KeyDown(208)=True Then y#=-0.1
If KeyDown(200)=True Then y#=0.1
If KeyDown(44)=True Then z#=-0.1
If KeyDown(30)=True Then z#=0.1

MoveEntity cone,x#,y#,z#

RenderWorld

Text 0,0,"Use cursor/A/Z keys to move cone"

; Return entity y position of cone
Text 0,20,"Y Position: "+EntityY#( cone )

Flip

Wend

End
```

EntityZ# (ente[,global])

Parámetros:
ente – nombre del ente del que se devolverá la coordenada z
global (opcional) - true si la coordenada z devuelta debería ser relativa a 0,0,0 antes que a la posición del ente padre. False por defecto

Descripción:
Devuelve la coordenada z de un ente.

Ejemplo:
```
; EntityZ Example
; ---------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
```

```
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,10

While Not KeyDown( 1 )

x#=0
y#=0
z#=0

If KeyDown(203)=True Then x#=-0.1
If KeyDown(205)=True Then x#=0.1
If KeyDown(208)=True Then y#=-0.1
If KeyDown(200)=True Then y#=0.1
If KeyDown(44)=True Then z#=-0.1
If KeyDown(30)=True Then z#=0.1

MoveEntity cone,x#,y#,z#

RenderWorld

Text 0,0,"Use cursor/A/Z keys to move cone"

; Return entity z position of cone
Text 0,20,"Z Position: "+EntityZ#( cone )

Flip

Wend

End
```

EntityRoll# (ente[,global])

Parámetros:
ente – nombre del ente del que se quiere saber el ángulo de rotación
global (opcional) - true si el ángulo de rotación del ente debería ser relativo a 0 antes que al ángulo de rotación del ente padre. False por defecto.

Descripción:
Devuelve el ángulo de rotación de un ente. El ángulo de rotación es también el ángulo z de un ente.

Ejemplo:

```
; EntityRoll Example
; ------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,5

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
```

```
If KeyDown( 44 )=True Then roll#=1

TurnEntity cone,pitch#,yaw#,roll#

RenderWorld

Text 0,0,"Use cursor/Z/X keys to turn cone"

; Return entity roll angle of cone
Text 0,20,"Roll: "+EntityRoll#( cone )

Flip

Wend

End
```

EntityYaw# (ente[,global])

Parámetros:
ente – nombre del ente del que se devolverá el ángulo de yaw
global (opcional) - true si el ángulo de yaw devuelto debería ser relativo a 0 antes que al ángulo yaw del ente padre. Por defecto es false

Descripción:
Devuelve el ángulo de yaw de un ente. El ángulo de yaw es también el ángulo y de un ente.

Ejemplo:

```
; EntityYaw Example
; -----------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,5

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cone,pitch#,yaw#,roll#

RenderWorld

Text 0,0,"Use cursor/Z/X keys to turn cone"

; Return entity yaw angle of cone
Text 0,20,"Yaw: "+EntityYaw#( cone )

Flip

Wend

End
```

EntityPitch# (ente[,global])

Parámetros:
ente – nombre del ente del que se quiere saber su ángulo de inclinación
global (opcional) - true si el ángulo de inclinación devuelto debería ser relativo a 0, antes que al ángulo de inclinación del ente padre. False por defecto.

Descripción:
Devuelve el ángulo de inclinación de un ente. El ángulo de inclinación es también el ángulo x de un ente.

Ejemplo:
```
; EntityPitch Example
; -------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

cone=CreateCone( 32 )
PositionEntity cone,0,0,5

While Not KeyDown( 1 )

pitch#=0
yaw#=0
roll#=0

If KeyDown( 208 )=True Then pitch#=-1
If KeyDown( 200 )=True Then pitch#=1
If KeyDown( 203 )=True Then yaw#=-1
If KeyDown( 205 )=True Then yaw#=1
If KeyDown( 45 )=True Then roll#=-1
If KeyDown( 44 )=True Then roll#=1

TurnEntity cone,pitch#,yaw#,roll#

RenderWorld

Text 0,0,"Use cursor/Z/X keys to turn cone"

; Return entity pitch angle of cone
Text 0,20,"Pitch: "+EntityPitch#( cone )

Flip

Wend

End
```

EntityName$ (ente)

Parameters:
ente – puntero del ente

Descripción:
Devuelve el nombre de un ente. El nombre de un ente puede establecerse en un programa de modelado, o manualmente usando NameEntity.

CountChildren (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve el número de hijos de un ente.

GetChild (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice del ente hijo. Debería estar en el rango 1...CountChildren(ente) inclusive.

Descripción:
Devuelve el hijo de un ente.

FindChild (ente,nombre_hijo$)

Parámetros:
ente – puntero del ente
nombre_hijo$ - nombre del hijo a encontrar dentro del ente

Descripción:
Devuelve el primer hijo del ente especificado que coincide con el nombre nombre_hijo$.

EntityPick (ente,rango#)

Parámetros:
ente – puntero del ente
rango# - rango para elegir el área alrededor del ente

Descripción:
Devuelve el ente más cercano delante del ente especificado. Un ente debe tener un EntityPickMode distinto de 0 para poder ser elegido.

LinePick (x#,y#,z#,dx#,dy#,dz#[,radio#])

Parámetros:
x# - coordenada x para empezar la línea de selección
y# - coordenada y para empezar la línea de selección
z# - coordenada z para empezar la línea de selección
dx# - distancia en x de la línea de selección
dy# - distancia en y de la línea de selección
dz# - distancia en z de la línea de selección
radio (opcional) – radio de la línea de selección

Descripción:
Devuelve el primer ente entre x,y,z y x+dx,y+dy,z+dz.

EntityVisible (orig_ente,dest_ente)

Parámetros:
orig_ente – puntero del ente de origen
dest_ente – puntero del ente de destino

Descripción:
Devuelve true si el orig_ente y el dest_ente pueden "verse" el uno al otro.

EntityDistance# (orig_ente,dest_ente)

Parámetros:
orig_ente – puntero del ente de origen
dest_ente – puntero del ente de destino

Descripción:
Devuelve la distancia ente el ente de origen y el ente de destino.

TFormPoint x#,y#,z#,orig_ente,dest_ente

Parameters:
x# - coordenada x del punto en el que se realizará la transformación
y# - coordenada y del punto en el que se realizará la transformación
z# - coordenada z del punto en el que se realizará la transformación
orig_ente – espacio local del ente de origen desde el que transformar el punto
dest_ente – espacio local del ente de destino al que transformar el punto

Descripcion:
Transforma un punto desde el espacio local de orig_ente al espacio local de dest_ente. Si orig_ente o dest_ente es 0, entonces se usa el espacio global. El resultado de la transformación puede ser recuperado usando los comandos TFormedX, TFormedY y TFormedZ.

TFormVector x#,y#,z#,orig_ente,dest_ente

Parámetros:
x# - componente x del vector en el que se realizará la transformación
y# - componente y del vector en el que se realizará la transformación
z# - componente z del vector en el que se realizará la transformación
orig_ente – espacio local del ente de origen desde el que transformar el vector
dest_ente – espacio local del ente de destino al que transformar el vector

Descripción:
Transforma un vector del espacio local del orig_ente al espacio local del dest_ente. Si orig_ente o dest_ente es 0, entonces se usa el espacio global. Los resultados de la transformación pueden ser recuperados usando los comandos TFormedX, TFormedY y TFormedZ.

TFormedX

Parámetros:
Ninguno.

Descripción:
Devuelve el componente X de la última operación TFormPoint, TFormVector o TFormNormal.

TFormedY

Parámetros:
Ninguno.

Descripción:
Devuelve el componente Y de la última operación TFormPoint, TFormVector o TFormNormal.

TFormedZ

Parámetros:
Ninguno.

Descripción:
Devuelve el componente Z de la última operación TFormPoint, TFormVector o TFormNormal.

EntityAnimating (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve true si el ente está siendo animado actualmente.

EntityAnimTime# (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve el tiempo de animación actual de un ente.

COLISIÓN DE ENTIDADES

ResetEntity ente

Parámetros:
ente – puntero del ente

Descripción:
Resetea (pone a cero) el estado de colisión de un ente.

EntityRadius ente,radio#

Parámetros:
ente – puntero del ente
radio# - radio de la esfera de colisión del ente

Descripción:
Establece el radio de la esfera de colisión de un ente. Se debe establecer el radio del ente para todos los entes involucrados en colisiones esféricas, que son todos los entes de origen (ya que las colisiones son siempre de esfera a algo), y cualquier ente de destino que esté involucrado en un colisión esfera a esfera (método de colisión 1).

Ejemplo:
```
; EntityRadius Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

sphere=CreateSphere( 32 )
PositionEntity sphere,-2,0,5

cone=CreateCone( 32 )
EntityType cone,type_cone
PositionEntity cone,2,0,5

; Set collision type values
type_sphere=1
type_cone=2

; Set sphere radius value
sphere_radius#=1

; Set sphere and cone entity types
EntityType sphere,type_sphere
EntityType cone,type_cone

; Enable collisions between type_sphere and type_cone, with sphere->polygon method and slide
response
Collisions type_sphere,type_cone,2,2

While Not KeyDown( 1 )

x#=0
y#=0
z#=0
```

```
If KeyDown( 203 )=True Then x#=-0.1
If KeyDown( 205 )=True Then x#=0.1
If KeyDown( 208 )=True Then y#=-0.1
If KeyDown( 200 )=True Then y#=0.1
If KeyDown( 44 )=True Then z#=-0.1
If KeyDown( 30 )=True Then z#=0.1

MoveEntity sphere,x#,y#,z#

; If square brackets keys pressed then change sphere radius value
If KeyDown( 26 )=True Then sphere_radius#=sphere_radius#-0.1
If KeyDown( 27 )=True Then sphere_radius#=sphere_radius#+0.1

; Set entity radius of sphere
EntityRadius sphere,sphere_radius#

; Perform collision checking
UpdateWorld

RenderWorld

Text 0,0,"Use cursor/A/Z keys to move sphere"
Text 0,20,"Press [ or ] to change EntityRadius value"
Text 0,40,"EntityRadius sphere,"+sphere_radius

Flip

Wend

End
```

EntityRadius ente,radio#

Parámetros:
ente – puntero del ente
radio# - radio de la esfera de colisión del ente

Descripción:
Establece el radio de la esfera de colisión de un ente. Se debe establecer el radio del ente para todos los entes involucrados en colisiones esféricas, que son todos los entes de origen (ya que las colisiones son siempre de esfera a algo), y cualquier ente de destino que esté involucrado en un colisión esfera a esfera (método de colisión 1).

Ejemplo:

```
; EntityRadius Example
; --------------------

Graphics3D 640,480
SetBuffer BackBuffer()

camera=CreateCamera()
light=CreateLight()

sphere=CreateSphere( 32 )
PositionEntity sphere,-2,0,5

cone=CreateCone( 32 )
EntityType cone,type_cone
PositionEntity cone,2,0,5

; Set collision type values
type_sphere=1
type_cone=2

; Set sphere radius value
sphere_radius#=1

; Set sphere and cone entity types
```

```
EntityType sphere,type_sphere
EntityType cone,type_cone

; Enable collisions between type_sphere and type_cone, with sphere->polygon method and slide
response
Collisions type_sphere,type_cone,2,2

While Not KeyDown( 1 )

x#=0
y#=0
z#=0

If KeyDown( 203 )=True Then x#=-0.1
If KeyDown( 205 )=True Then x#=0.1
If KeyDown( 208 )=True Then y#=-0.1
If KeyDown( 200 )=True Then y#=0.1
If KeyDown( 44 )=True Then z#=-0.1
If KeyDown( 30 )=True Then z#=0.1

MoveEntity sphere,x#,y#,z#

; If square brackets keys pressed then change sphere radius value
If KeyDown( 26 )=True Then sphere_radius#=sphere_radius#-0.1
If KeyDown( 27 )=True Then sphere_radius#=sphere_radius#+0.1

; Set entity radius of sphere
EntityRadius sphere,sphere_radius#

; Perform collision checking
UpdateWorld

RenderWorld

Text 0,0,"Use cursor/A/Z keys to move sphere"
Text 0,20,"Press [ or ] to change EntityRadius value"
Text 0,40,"EntityRadius sphere,"+sphere_radius

Flip

Wend

End
```

EntityBox ente,x#,y#,z#,ancho#,alto#,profundidad#

Parámetros:
ente – puntero del ente
x# - posición x de la caja de colisión del ente
y# - posición y de la caja de colisión del ente
z# - posición z de la caja de colisión del ente
ancho# - ancho de la caja de colisión del ente
alto# - alto de la caja de colisión del ente
profundidad# - profundidad de la caja de colisión del ente

Descripción:
Establece las dimensiones de la caja de colisión de un ente.

EntityType ente,tipo_colisión[,repetitivo]

Parámetros:
ente – puntero del ente
tipo_colisión – tipo de colisión del ente
repetitivo (opcional) - true para aplicar el tipo de colisión a los hijos del ente. Por defecto es false.

Descripción:
Establece el tipo de colisión para un ente.

EntityPickMode ente,geometría_elección,esconder

Parámetros:
ente – puntero del ente
geometría_elección – tipo de figura geométrica usada para la elección:
0: Inelegible (por defecto)
1: Esfera (se usa EntityRadius)
2: Polígono
3: Caja (se usa EntityBox)
esconder - true para determinar que el ente "esconde" otros entes durante una llamada a EntityVisible.

Descripción:
Establece el modo de elección de un ente.

EntityCollided (ente,tipo)

Parámetros:
ente – puntero del ente
tipo – tipo de ente

Descripción:
Devuelve true si un ente colisiona con cualquier otro ente del tipo especificado.

CountCollisions (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve en cuantas colisiones se ha visto involucrado un ente durante la última actualización del mundo (UpdateWorld).

CollisionX# (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve la coordenada x del mundo de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente) inclusive.
Ver también: CollisionY, CollisionZ.

CollisionY# (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve la coordenada y del mundo de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente) inclusive.
Ver también: CollisionX, CollisionZ.

CollisionZ# (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve la coordenada z del mundo de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente) inclusive.
Ver también: CollisionX, CollisionY.

CollisionNX# (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve el componente x de la normal de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente) inclusive.
Ver también: CollisionNY, CollisionNZ.

CollisionNY# (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve el componente y de la normal de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente) inclusive.
Ver también: CollisionNX, CollisionNZ.

CollisionNZ# (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve el componente z de la normal de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente) inclusive.
Ver también: CollisionNX, CollisionNY.

CollisionTime (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve el tiempo que lleva calcular una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente) inclusive.

CollisionEntity (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve el otro ente involucrado en una colisión particular. El índice debería estar en el rango entre 1...CountCollisions(ente), inclusive.

CollisionSurface (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve el número de índice perteneciente a la superficie del ente especificado que está más cerca del punto de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente), inclusive.

CollisionTriangle (ente,índice)

Parámetros:
ente – puntero del ente
índice – índice de colisión

Descripción:
Devuelve el número de índice del triángulo perteneciente al ente especificado que está más cerca del punto de una colisión en particular. El índice debería estar en el rango 1...CountCollisions(ente), inclusive.

GetEntityType (ente)

Parámetros:
ente – puntero del ente

Descripción:
Devuelve el tipo de colisión de un ente.

MODO GRÁFICO

GfxModeExists (ancho,alto,profundiad[,gfx3d])

Parámetros:
ancho – el ancho del modo de pantalla que se quiere comprobar
alto – el alto del modo de pantalla que se quiere comprobar
profundidad – la profundidad del modo de pantalla que se quiere comprobar
gfx3d – (opcional) parámetro para comprobar si el modo es 3D-compatible (por defecto es 0)

Descripción:
GfxModeExists es tristemente un comando poco usado que te permitirá comprobar si el modo de pantalla dado existe en la tarjeta gráfica antes de llamar a los comandos Graphics o Graphics3D. Nota que omitiendo el parámetro opcional gfx3d también se comprobará si el modo especificado es 3D-compatible. GfxModeExists devolverá True si el modo de pantalla dado está disponible, y False si el modo especificado no está disponible. Si devuelve False, un juego usar un modo diferente o salir de forma correcta (por ejemplo: Error de Ejecución: he sido demasiado perezoso y he creado mi juego para usarse con un modo de pantalla en particular que tu sistema no tiene. Realmente debería haber usado CountGfxModes y GfxMode alto/ancho/profundidad para encontrar los modos que tiene tu sistema. Oh, bien, pero me he ahorrado 10 minutos de mi vida!"….espero que hayas notado el sutil sarcasmo!).

Ejemplo:
```
If GfxModeExists (2500, 2500, 64)
Graphics 2500, 2500, 64
Else
RuntimeError "PC display hardware isn't yet capable of silly resolutions like 2500 x 2500 x 64-bit!"
EndIf
```

GfxDriver3D (controlador)

Parámetros:
controlador – muestra el número del controlador a comprobar, de 1 a CountGfxDrivers ()

Descripción:
GfxDriver3D devuelve True si el controlador de gráficos especificado es capaz de ejecutar 3D. GfxDriver3D se usa generalmente después de obtener una lista de los controladores disponibles del sistema del usuario mediante CountGfxDrivers (). Aplica esto a cada controlador a su vez (o al controlador seleccionado) para ver si es compatible con 3D. Si devuelve false, el controlador no puede ejecutar operaciones 3D. En sistemas con más de un controlador de pantalla, puedes usar esto para comprobar la capacidad 3D de cada uno antes de escoger uno mediante el comando SetGfxDriver.

Ejemplo:
```
For a = 1 To CountGfxDrivers ()
If GfxDriver3D (a)
Print GfxDriverName (a) + " is 3D-capable"
Else
Print GfxDriverName (a) + " is NOT 3D-capable"
EndIf
Delay 100
Next
```

GfxMode3D (modo)

Parámetros:
Modo – modo de gráfico, numerado de 1 - CountGfxModes ()

Descripción:
GfxMode3D devuelve True si el modo gráfico especificado es compatible 3D. GfxMode3D se usa generalmente después de obtener una lista de los modos de pantalla disponibles del sistema del usuario mendiante CountGfxModes (). Aplica esto a cada uno de los modos (o al modo seleccionado) para ver si puedes ejecutar modo 3D. Si devuelve False, llamar a Graphics3D () con este modo de detalles fallará!
Si no deseas ejecutar esta comprobación, el único llamamiento a Graphics3D que puedes hacer con garantía de que funcione en un 99% en gráficos 3D, es 'Graphics3D 640, 480'. De todos modos, mira GfxModeExists' para ver otros métodos de ejecutar esta comprobación.

Ejemplo:
```
For a = 1 To CountGfxModes ()
If GfxMode3D (a)
Print "Mode " + a + " is 3D-capable"
Else
Print "Mode " + a + " is NOT 3D-capable"
EndIf
Delay 100
Next
```

Windowed3D

Parámetros:
Ninguno.

Descripción:
Windowed3D () devuelve True si la tarjeta gráfica es capaz de renderizar gráficos 3D en una ventana en el escritorio.
Ciertas tarjetas gráficas sólo permitirán una profundidad de color 3D determinada (por ej., Voodoo3 soporta sólo modos 3D de 16-bit y Matrox sólo soporta modos 3D de 16 y 32-bit pero permite establecer modos 2D de 24-bit). Si el escritorio del usuario está establecido a una profundidad de color no soportada en 3D, Windowed3D devolverá False.

Ejemplo:
```
If Windowed3D ()
Graphics3D 640, 480, 0, 2
Print "Windowed mode!"
Else
Graphics3D 640, 480, 0, 1
Print "Full screen modes only!"
EndIf

MouseWait
End
```