· DirectX8

En cualquier aplicación de DirectX que programemos debemos incluir la librería dxguid.lib.
En aplicaciones con DirectInput: dinput8.lib y dinput.h
En aplicaciones con DirectMusic: dmusicc.h y dmusici.h

· DirectInput

· Usar DirectInput

Para usar DI necesitamos seguir los siguientes pasos (Con * son opcionales):
Lo primero será inicializar DirectInput y luego añadir dispositivos de la siguiente forma:

1. Enumerar los dispositivos *
2. Crear dispositivos
3. Verificar las capacidades de los dispositivos *
4. Enumerar objetos *
5. Adquirir el formato de los datos
6. Entrar en nivel coperativo de windows
7. Modificar las propiedades de los dispositivos *
8. Adquirir dispositivo

Las variables que usaremos serán:

```
HRESULT                 result;
LPDIRECTINPUT8          pDirectInput;
LPDIRECTINPUTDEVICE8    lpDIDEVICE8;
UCHAR                   buffer[256];
```

Inicializar DI

```
if (FAILED(result = DirectInput8Create(hInstance, DIRECTINPUT_VERSION, IID_IDirectInput8, (void **) &pDirectInput, NULL)))
        {
                MessageBox(hwnd, "Error creando DI", NULL, NULL);
        }
```

*result = DirectInput8Create(hInstance, DIRECTINPUT_VERSION, IID_IDirectInput8, (void **) &pDirectInput, NULL)*
hInstance es la instancia del programa
DIRECTINPUT_VERSION es una constante dentro del dinput.h que indica la versión de DirectInput usada
pDirectInput es un puntero necesario a las funciones de DirectInput.

Crear dispositivos
result = pDirectInput->CreateDevice(GUID_SysKeyboard, &lpDIDEVICE8, NULL);
De esta forma creamos los dispositivos. El primer parámetro puede ser GUID_SysKeyboard o GUID_SysMouse. Y le pasamos el dispositivo donde lo crearemos.

Adquirir el formato de datos
result = lpDIDEVICE8->SetDataFormat(&c_dfDIKeyboard);

c_dfDIKeyboard	Array de caracteres de 256 que representa a cada tecla.
c_dfDIMouse	Una estructura con tres longs para las coordenadas del ratón (x,y,z) y un array de 4 bytes que indica el botón clicado.
c_dfDIJoystick	Otra estructura para los joysticks.

Entrar en el nivel cooperativo
result = lpDIDEVICE8->SetCooperativeLevel(hwnd, DISCL_BACKGROUND);
Indica de qué forma usaremos el dispositivo, necesitamos pasarle el HWND de la ventana y las otras opciones serán: DISCL_BACKGROUND, DISCL_EXECUTIVE, DISCL_FOREGROUND, DISCL_NONEXCLUSIVE y DISCL_NOWINKEY (que desactiva la tecla de windows).

Adquirir dispositivo
if (FAILED(result = lpDIDEVICE8->Acquire())) MessageBox(hwnd, "Error adquiriendo dispositivo DI", NULL, NULL);

lpDIDEVICE8->Acquire()	Esta función adquiere el dispositivo.

Coger la tecla apretada
Podemos hacer un define de la siguiente forma antes de proseguir:
#define KEYDOWN(name, key) (name[key] & 0x80)
De esta forma cuando llamemos a la siguiente función (dentro del bucle principal del programa) la sintaxis será más sencilla:
result = lpDIDEVICE8->GetDeviceState(sizeof(buffer),(LPVOID) &buffer);
Con GetDeviceState lo que hacemos es coger el estado del dispositivo creado, para meterlo en un array de x posiciones (con el tamaño indicado en el primer parámetro).
Luego podremos hacer esto:
if (KEYDOWN(buffer,DIK_RIGHT)) xpos+=0.1;

* A veces puede ser que perdamos el control del dispositivo y entonces el programa fallaría, para comprovar si aún lo tenemos y arreglar el error si no, haremos:
if (FAILED(lpDIDEVICE8->GetDeviceState(sizeof(buffer),(LPVOI) &buffer)) lpDIDEVICE8->Acquire();

· Mapa de caracteres en DirectInput

Constant	Hex	Description
DIK_ESCAPE	0x01	Escape
DIK_1	0x02	1
DIK_2	0x03	2
DIK_3	0x04	3
DIK_4	0x05	4
DIK_5	0x06	5
DIK_6	0x07	6
DIK_7	0x08	7
DIK_8	0x09	8
DIK_9	0x0A	9
DIK_0	0x0B	0
DIK_MINUS	0x0C	- on main keyboard
DIK_EQUALS	0x0D	=
DIK_BACK0x0E	Backspace	
DIK_TAB	0x0F	Tab
DIK_Q	0x10	Q

DIK_W	0x11	W
DIK_E	0x12	E
DIK_R	0x13	R
DIK_T	0x14	T
DIK_Y	0x15	Y
DIK_U	0x16	U
DIK_I	0x17	I
DIK_O	0x18	O
DIK_P	0x19	P
DIK_LBRACKET	0x1A	[
DIK_RBRACKET	0x1B	]
DIK_RETURN	0x1C	Enter on main keyboard
DIK_LCONTROL	0x1D	Left Ctrl
DIK_A	0x1E	A
DIK_S	0x1F	S
DIK_D	0x20	D
DIK_F	0x21	F
DIK_G	0x22	G
DIK_H	0x23	H
DIK_J	0x24	J
DIK_K	0x25	K
DIK_L	0x26	L
DIK_SEMICOLON	0x27	;
DIK_APOSTROPHE	0x28	'
DIK_GRAVE	0x29	Accent grave
DIK_LSHIFT	0x2A	Left shift
DIK_BACKSLASH	0x2B	\
DIK_Z	0x2C	A
DIK_X	0x2D	X
DIK_C	0x2E	C
DIK_V	0x2F	V
DIK_B	0x30	B
DIK_N	0x31	N
DIK_M	0x32	M
DIK_COMMA	0x33	,
DIK_PERIOD	0x34	. on main keyboard
DIK_SLASH	0x35	/ on main keyboard
DIK_RSHIFT	0x36	Right shift
DIK_MULTIPLY	0x37	* on numeric keypad
DIK_LMENU	0x38	Left Alt
DIK_SPACE	0x39	Space bar
DIK_CAPITAL	0x3A	Caps Lock
DIK_F1	0x3B	F1
DIK_F2	0x3C	F2
DIK_F3	0x3D	F3
DIK_F4	0x3E	F4
DIK_F5	0x3F	F5
DIK_F6	0x40	F6
DIK_F7	0x41	F7
DIK_F8	0x42	F8
DIK_F9	0x43	F9
DIK_F10	0x44	F10
DIK_NUMLOCK	0x45	Num Lock
DIK_SCROLL	0x46	Scroll Lock
DIK_NUMPAD7	0x47	7 on numeric keypad
DIK_NUMPAD8	0x48	8 on numeric keypad
DIK_NUMPAD9	0x49	9 on numeric keypad
DIK_SUBTRACT	0x4A	- on numeric keypad
DIK_NUMPAD4	0x4B	4 on numeric keypad
DIK_NUMPAD5	0x4C	5 on numeric keypad
DIK_NUMPAD6	0x4D	6 on numeric keypad
DIK_ADD	0x4E	+ on numeric keypad
DIK_NUMPAD1	0x4F	1 on numeric keypad
DIK_NUMPAD2	0x50	2 on numeric keypad
DIK_NUMPAD3	0x51	3 on numeric keypad
DIK_NUMPAD0	0x52	0 on numeric keypad
DIK_DECIMAL	0x53	. on numeric keypad
DIK_F11	0x57	F11
DIK_F12	0x58	F12
DIK_F13	0x64	(NEC PC98)
DIK_F14	0x65	(NEC PC98)
DIK_F15	0x66	(NEC PC98)
DIK_KANA	0x70	(Japanese keyboard)
DIK_CONVERT	0x79	(Japanese keyboard)
DIK_NOCONVERT	0x7B	(Japanese keyboard)
DIK_YEN	0x7D	(Japanese keyboard)
DIK_NUMPADEQUALS	0x8D	= on numeric keypad (NEC PC98)
DIK_CIRCUMFLEX	0x90	(Japanese keyboard)
DIK_AT	0x91	(NEC PC98)
DIK_COLON	0x92	(NEC PC98)
DIK_UNDERLINE	0x93	(NEC PC98)
DIK_KANJI	0x94	(Japanese keyboard)
DIK_STOP	0x95	(NEC PC98)
DIK_AX	0x96	(Japan AX)
DIK_UNLABELED	0x97	(J3100)
DIK_NUMPADENTER	0x9C	Enter on numeric keypad

DIK_RCONTROL 0x9D Right Ctrl
DIK_NUMPADCOMMA 0xB3 , on numeric keypad (NEC PC98)
DIK_DIVIDE 0xB5 / on numeric keypad
DIK_SYSRQ 0xB7 SysRq
DIK_RMENU 0xB8 Right Alt
DIK_PAUSE 0xC5 Pause
DIK_HOME 0xC7 Home on arrow keypad
DIK_UP 0xC8 Up arrow on arrow keypad
DIK_PRIOR 0xC9 Page Up on arrow keypad
DIK_LEFT 0xCB Left arrow on arrow keypad
DIK_RIGHT 0xCD Right arrow on arrow keypad
DIK_END 0xCF End on arrow keypad
DIK_DOWN 0xD0 Down arrow on arrow keypad
DIK_NEXT 0xD1 Page Down on arrow keypad
DIK_INSERT 0xD2 Insert on arrow keypad
DIK_DELETE 0xD3 Delete on arrow keypad
DIK_LWIN 0xDB Left Windows key
DIK_RWIN0xDC Right Windows key
DIK_APPS 0xDD App menu key
DIK_POWER 0xDE Power
DIK_SLEEP 0xDF Sleep

Alternate names for keys, to facilitate transition from DOS
DIK_BACKSPACE DIK_BACK Backspace
DIK_NUMPADSTAR DIK_MULTIPLY * on numeric keypad
DIK_LALT DIK_LMENU Left Alt
DIK_CAPSLOCK DIK_CAPITAL CapsLock
DIK_NUMPADMINUS DIK_SUBTRACT - on numeric keypad
DIK_NUMPADPLUS DIK_ADD + on numeric keypad
DIK_NUMPADPERIOD DIK_DECIMAL . on numeric keypad
DIK_NUMPADSLASH DIK_DIVIDE / on numeric keypad
DIK_RALT DIK_RMENU Right Alt
DIK_UPARROW DIK_UP Up arrow on arrow keypad
DIK_PGUP DIK_PRIOR Page up on arrow keypad
DIK_LEFTARROW DIK_LEFT Left arrow on arrow keypad
DIK_RIGHTARROW DIK_RIGHT Right arrow on arrow keypad
DIK_DOWNARROW DIK_DOWN Down arrow on arrow keypad
DIK_PGDN DIK_NEXT Page Down on arrow keypad

· Una clase que controle teclado y ratón

---

cDI.h

```
#include <dinput.h>

#pragma comment(lib, "dxguid.lib")
#pragma comment(lib, "dinput8.lib")

#define KEYDOWN(name, key) (name[key] & 0x80)

struct STMouse{
	float x;
	float y;
	BOOL button[3];
	BOOL rueda[2];
};

class cDI
{
public:
	cDI(HINSTANCE hInstance, HWND hwnd);
	IniMous();
	IniKeyb();
	UCHAR *LookKeyb();
	STMouse LookMous();
	STMouse LookMouZ(float rango, float anc, float alt, float vel, bool top);
private:
	HWND hwnd;
	LPDIRECTINPUT8 pDirectInput;
	LPDIRECTINPUTDEVICE8 Keyb, Mous;
	UCHAR buffer[256];
	DIMOUSESTATE2 MousStat;
	STMouse MousDat;
	STMouse MousTemp;
	int MovZ;
	BOOL MousUpdate();
	BOOL KeybUpdate();
	Button(bool temp);
};
```

---

cDI.cpp

```
#include "cdi.h"

cDI::cDI(HINSTANCE hInstance, HWND hWnd)
{
```

```
	this->hwnd = hWnd;
	this->MousTemp.x=0;
	this->MousTemp.y=0;
	this->MovZ=0;
	if (FAILED(DirectInput8Create(hInstance, DIRECTINPUT_VERSION, IID_IDirectInput8, (void **) &pDirectInput, NULL)))
		{
			MessageBox(hwnd, "Error creando DI", NULL, NULL);
		}
}

cDI::IniKeyb()
{
	pDirectInput->CreateDevice(GUID_SysKeyboard, &Keyb, NULL);
	Keyb->SetDataFormat(&c_dfDIKeyboard);
	Keyb->SetCooperativeLevel(hwnd, DISCL_BACKGROUND | DISCL_NONEXCLUSIVE);

	if (FAILED(Keyb->Acquire() ))
	{
		MessageBox(hwnd, "Error adquiriendo dispositivo DI 01", NULL, NULL);
	}
}

cDI::IniMous ()
{
	pDirectInput->CreateDevice(GUID_SysMouse, &Mous, NULL);
	Mous->SetDataFormat(&c_dfDIMouse2);
	Mous->SetCooperativeLevel(hwnd, DISCL_FOREGROUND | DISCL_NONEXCLUSIVE);

	if (FAILED(Mous->Acquire() ))
	{
		MessageBox(hwnd, "Error adquiriendo dispositivo DI 02", NULL, NULL);
	}
}

UCHAR *cDI::LookKeyb ()
{
	if (KeybUpdate()) Keyb->GetDeviceState(sizeof(buffer),(LPVOID) &buffer);
	return buffer;
}

STMouse cDI::LookMous ()
{
	if (MousUpdate())
	{
	Mous->GetDeviceState (sizeof(DIMOUSESTATE2), &MousStat);
	MousDat.x = MousStat.lX;
	MousDat.y = MousStat.lY;
	this->Button(false);
	}
	return MousDat;
}

BOOL cDI::MousUpdate ()
{
	if (FAILED(Mous->Acquire())) return FALSE;
	return TRUE;
}

BOOL cDI::KeybUpdate()
{
	if (FAILED(Keyb->Acquire())) return FALSE;
	return TRUE;
}

STMouse cDI::LookMouZ (float rango, float anc, float alt, float vel, bool top)
{
	float MedX, MedY;

   if (MousUpdate()) {
			Mous->GetDeviceState (sizeof(DIMOUSESTATE2), &MousStat);
			MousDat.x = MousStat.lX;
			MousDat.y = MousStat.lY;

			MedX=(anc/rango)*vel;
			MedY=(alt/rango)*vel;
			MousTemp.x += MousDat.x*MedX;
			MousTemp.y -= MousDat.y*MedY;
			if (top) {
				if (MousTemp.x<-rango) MousTemp.x=-rango;
				if (MousTemp.x>rango) MousTemp.x=rango;
				if (MousTemp.y>rango) MousTemp.y=rango;
				if (MousTemp.y<-rango) MousTemp.y=-rango;
			}
			this->Button(true);
	}
	return MousTemp;
```

```
}

cDI::Button (bool temp)
{
	BOOL BoT[5]={0,0,0,0,0};
	if (MousStat.rgbButtons[0]!=0) BoT[0]=1;
	if (MousStat.rgbButtons[1]!=0) BoT[1]=1;
	if (MousStat.rgbButtons[2]!=0) BoT[2]=1;
	if (MousStat.lZ>0) BoT[3]=1;
	if (MousStat.lZ<0) BoT[4]=1;
	if (temp) {
		MousTemp.button[0]=BoT[0];
		MousTemp.button[1]=BoT[1];
		MousTemp.button[2]=BoT[2];
		MousTemp.rueda[0]=BoT[3];
		MousTemp.rueda[1]=BoT[4];
	}
	else {
		MousDat.button[0]=BoT[0];
		MousDat.button[1]=BoT[1];
		MousDat.button[2]=BoT[2];
		MousDat.rueda[0]=BoT[3];
		MousDat.rueda[1]=BoT[4];
	}
}
```

En el cDI.h encontramos las definiciones para las variables y funciones del cDI.cpp:

```
cDI(HINSTANCE hInstance, HWND hwnd);
```
Para la creación de un objeto DI.
```
IniMous();
```
Para iniciar el mouse.
```
IniKeyb();
```
Para inicializar el teclado.
```
UCHAR *LookKeyb();
```
Para recoger las teclas apretadas del teclado.
```
STMouse LookMous();
```
Para calcular posición del mouse.
```
STMouse LookMouZ(float rango, float anc, float alt, float vel, bool top);
```
Para calcular posición relativa del mouse. Pasandole un rango (ya que lo toma como si el punto 0, 0 estubiese en el centro de la ventana), un alto y un ancho, una velocidad y si tiene límites en esa altura y anchura, él calculará la posición del ratón según estos parámetros.
```
HWND                    hwnd;
```
El HWND de la ventana principal, lo coge sólo al iniciar el objeto.
```
LPDIRECTINPUT8          pDirectInput;
```
Un puntero al objeto DI.
```
LPDIRECTINPUTDEVICE8    Keyb, Mous;
```
Objetos DI.
```
UCHAR                   buffer[256];
```
Un buffer que rellenará el teclado.
```
DIMOUSESTATE2           MousStat;
```
Donde se calculará la posición del ratón.
```
STMouse                 MousDat;
```
Para tener las coordenadas del ratón.
```
STMouse                 MousTemp;
```
Para tener las coordenadas relativas del ratón.
```
int                     MovZ;
```
Un indicador para el movimiento de la rueda.
```
BOOL MousUpdate();
```
Para actualizar el ratón y no perder así el dispositivo.
```
BOOL KeybUpdate();
```
Para actualizar el teclado.
```
Button(bool temp);
```
Para comprovar qué botones del ratón se han pulsado.

Una estructura de ratón (STMouse) tiene:
Un float para la posición x y otro para la y.
Tres booleanas que indican si se ha pulsado un botón o no.
Dos booleanas para el movimiento de la rueda. La primera representa hacia arriba y la segunda hacia abajo.

Y para usarla:

```
cDI *DI = new cDI(hInstance, hWnd);
DI->IniKeyb ();
DI->IniMous ();
```

Creamos un objeto DI, y un teclado y un ratón.
Luego, dentro del bucle principal del programa:

```
Key=DI->LookKeyb();
Raton=DI->LookMouZ (RangoXYZ, NewANC, NewALT, 0.05, true);
if (KEYDOWN(KB,DIK_RIGHT)) xpos+=0.1;
if (Mous.button[0]) color++;
if (Mous.rueda[0]) rueda++;
```

· DirectMusic

DirectMusic es un objeto COM del DirectX, por lo tanto, a diferencia del DirectInput necesitaremos inicializarlo: *CoInitialize(NULL);*
Tres son los objetos que necesitamos para usar de forma básica el DirectMusic:

IDirectMusicPerformance8*	El objeto principal del Direct Audio.
IDirectMusicLoader8*	El cargador del DirectMusic
IDirectMusicSegment8*	El sonido

Usaremos:

IDirectMusicPerformance8*	dmusicPerformance = NULL;
IDirectMusicLoader8*	dmusicLoader = NULL;
IDirectMusicSegment8*	dmusicSegment = NULL;

Crear el performance y el loader

```
CoCreateInstance(CLSID_DirectMusicPerformance, NULL, CLSCTX_INPROC,
        IID_IDirectMusicPerformance8, (void**)&dmusicPerformance);
```

De esta forma obtienes la interfaz del permormance.

```
dmusicPerformance->InitAudio(NULL, NULL, NULL, DMUS_APATH_SHARED_STEREOPLUSREVERB,
        64, DMUS_AUDIOF_ALL, NULL);
```

Inicializa el audio. Sus parámetros son: Puntero a la interfaz del directmusic (no necesario), al del direct audio (no necesario), el hwnd, el tipo de audiopath por defecto, numero de canales, características del sintetizador y los parámetros de este (NULL = por defecto).

```
CoCreateInstance(CLSID_DirectMusicLoader, NULL, CLSCTX_INPROC,
        IID_IDirectMusicLoader8, (void**)&dmusicLoader);
```

Crear y cargar el segmento

El path por defecto

No es necesario, pero si tubiesemos todos los archivos en un mismo path, con esto, no necesitariamos pasar toda la ruta.

```
WCHAR searchPath[MAX_PATH];
```

Necesitamos que el path no sea char, sino WCHAR.

```
MultiByteToWideChar(CP_ACP, 0, "\data", -1, searchPath, MAX_PATH);
```

Con esta función pasamos "\data" (que es el path que usaremos) a WCHAR.

```
dmusicLoader->SetSearchDirectory (GUID_DirectMusicAllTypes, searchPath, FALSE);
```

Y lo definimos como path por defecto.

Respecto al segmento

```
WCHAR filename[MAX_PATH];
```

Necesitamos que el nombre del archivo no sea char, sino WCHAR.

```
MultiByteToWideChar(CP_ACP, 0, file, -1, filename, MAX_PATH);
```

Con esta función pasamos "file" (que es un char) a WCHAR.

```
dmusicLoader->LoadObjectFromFile(CLSID_DirectMusicSegment, IID_IDirectMusicSegment8,
        filename, (void**)&dmusicSegment);
```

Cargamos el archivo sobre el loader y el segmento.

```
dmusicSegment->Download(dmusicPerformance);
```

Preparamos el segmento para recibir los datos.

Manipular el sonido

Que suene

```
dmusicPerformance->PlaySegmentEx (dmusicSegment, NULL, NULL, 0,0, NULL, NULL, NULL);
```

Parámetros: Segmento con el archivo cargado, NULL, el segmento de transición, flags del metodo de comportamiento (¿?), el momento en que empieza a reproducirse, estado del segmento, cuando parará y el path en el que está (NULL es el de por defecto).

Pararlo

```
dmusicPerformance->StopEx(dmusicSegment, 0,0);
```

Parámetros: El segmento, el momento en el que para, 0.

```
dmusicPerformance->Stop(dmusicSegment, NULL, 0, 0);
```

Parámetros: El segmento (NULL son todos), NULL, el momento en el que para (0 inmediatamente).

Está sonando?

```
dmusicPerformance->IsPlaying (dmusicSegment, NULL)
```

Si el segmento introducido está sonando devolverá: S_OK.

Que se repita

```
dmusicSegment->SetRepeats(Loops);
```

Loops: Las veces que se repetirá ese segmento.

Y Cerrar DirectMusic

Las funciones usadas para ello son:

```
dmusicPerformance->CloseDown();
dmusicLoader->Release ();
dmusicPerformance->Release();
dmusicSegment->Release();
CoUninitialize();
```

Un ejemplo:

Código

```
#include <dmusicc.h>
#include <dmusici.h>

#pragma comment(lib, "dxguid.lib")

IDirectMusicPerformance8*          dmusicPerformance = NULL;
IDirectMusicLoader8*               dmusicLoader = NULL;
IDirectMusicSegment8*              dmusicSegment = NULL;

void IniDMusic(HWND hwnd)
{
            WCHAR searchPath[MAX_PATH];

            CoInitialize(NULL);

            CoCreateInstance(CLSID_DirectMusicPerformance, NULL, CLSCTX_INPROC,
                        IID_IDirectMusicPerformance8, (void**)&dmusicPerformance);

            dmusicPerformance->InitAudio(NULL, NULL, NULL, DMUS_APATH_SHARED_STEREOPLUSREVERB,
                        64, DMUS_AUDIOF_ALL, NULL);

            CoCreateInstance(CLSID_DirectMusicLoader, NULL, CLSCTX_INPROC,
                        IID_IDirectMusicLoader8, (void**)&dmusicLoader);

            MultiByteToWideChar(CP_ACP, 0, "\data", -1, searchPath, MAX_PATH);
            dmusicLoader->SetSearchDirectory (GUID_DirectMusicAllTypes, searchPath, FALSE);
}

void LoadWDMusic(char* file)
{
            WCHAR filename[MAX_PATH];
            MultiByteToWideChar(CP_ACP, 0, file, -1, filename, MAX_PATH);
            dmusicLoader->LoadObjectFromFile(CLSID_DirectMusicSegment, IID_IDirectMusicSegment8,
                        filename, (void**)&dmusicSegment);
            dmusicSegment->Download(dmusicPerformance);
}

void Play()
{
            dmusicPerformance->PlaySegmentEx (dmusicSegment, NULL, NULL, 0,0, NULL, NULL, NULL);
}

void Stop()
{
            dmusicPerformance->StopEx(dmusicSegment, 0,0);

}

bool Suena()
{
            if (dmusicPerformance->IsPlaying (dmusicSegment, NULL) == S_OK)
                        return true;
            else
                        return false;
}

void OtraVez()
{
            static int Loops = 1;
            Loops ++;
            dmusicSegment->SetRepeats(Loops);
}

void EndDMusc()
{
            dmusicPerformance->CloseDown();
            dmusicLoader->Release ();
            dmusicPerformance->Release();
            dmusicSegment->Release();
            CoUninitialize();
}
```

Llamadas

```
case VK_ESCAPE:
            PostQuitMessage (0);
            break;
case VK_RIGHT:
            Play();
            break;
case VK_LEFT:
            Stop();
            break;
case VK_DOWN:
            if (Suena())
                        MessageBox(hWnd, "ESTA SONANDO", NULL, NULL);
            else
                        MessageBox(hWnd, "NO SUENA NADA", NULL, NULL);
            break;
case VK_UP:
            OtraVez();
```