5.3. Implementación

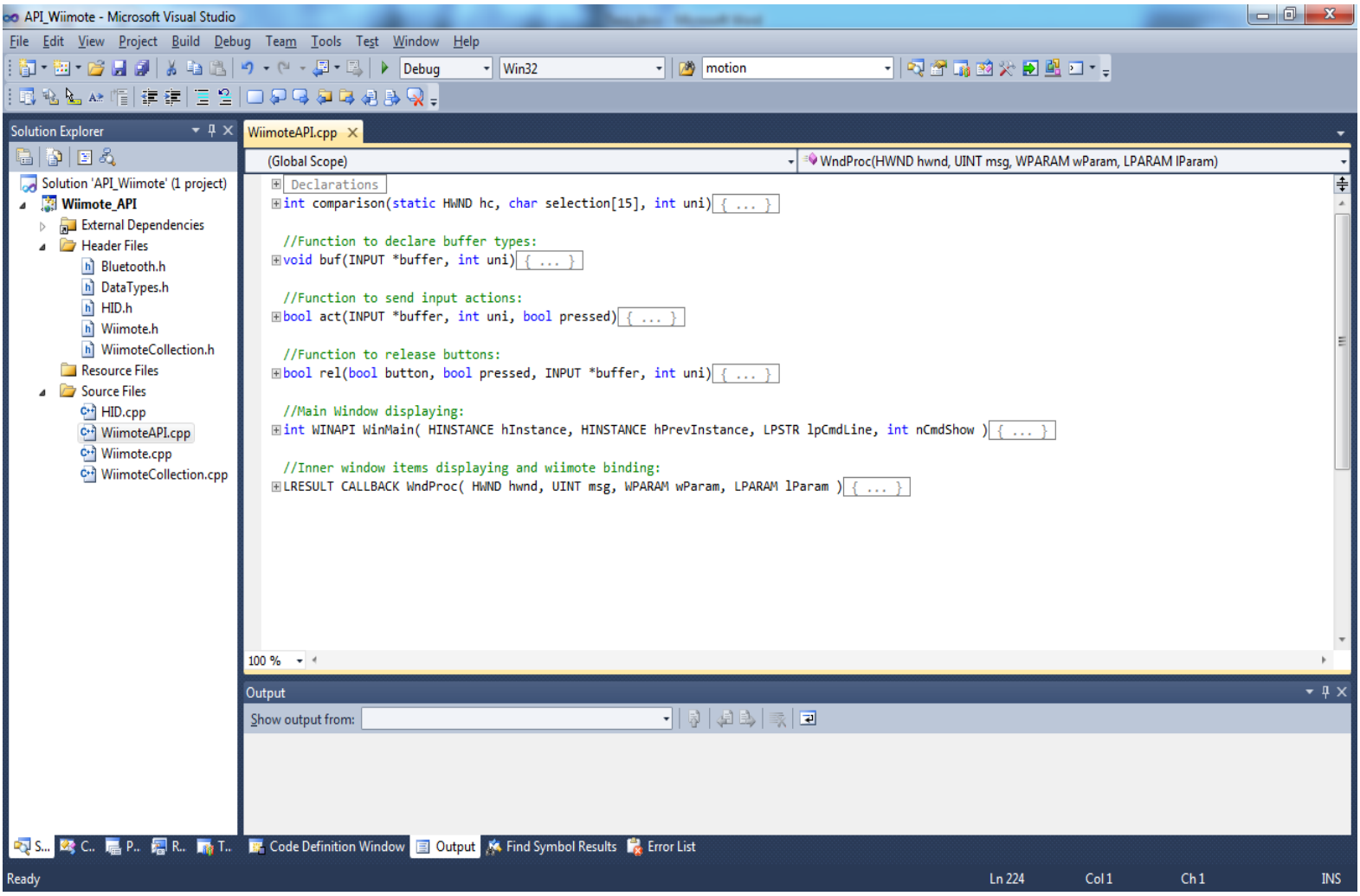
Para desarrollar este proyecto de tesis se utilizó Visual Studio 2010 y se utilizó el SDK para Windows 7 y .NET Framework versión 4, lo cual permite el uso de las diversas funciones de programación de interfaces gráficas a través de WinAPI.

5.3.1. Bibliotecas y cabeceras.

Existen varias bibliotecas y cabeceras que se utilizaron para mejorar la compatibilidad de la aplicación, para poder utilizar estructuras de datos específicas y para llamar funciones y métodos que permiten la comunicación con el Wiimote. Estas son las principales cabeceras y librerías utilizadas por el código principal de la aplicación, WiimoteAPI.cpp:

```
#pragma comment(lib, "setupapi.lib")
#include <iostream>
#include <assert.h>
#include "WiimoteCollection.h"
#include <windows.h>
```

Para tener una idea de las bibliotecas y demás códigos de ayuda en el desarrollo de esta aplicación a continuación se muestra una vista del proyecto en Visual Studio:



La cabecera principal es Wiimote.h, sin embargo el código principal WiimoteAPI.cpp se comunica con esta cabecera a través de WiimoteCollection.h, la cual se muestra a continuación:

```
#include "Wiimote.h"

// Function to convert everything to a void pointer
void* ToVoidPointer( void* Dummy, ... );

// Class for a vector with 'Wiimote' objects
// This class is for multiple wiimote connection
class WiimoteCollection:Wiimote{
public:
	WiimoteCollection();
	~WiimoteCollection();
	void FindAllWiimotes();
	bool WiimoteFound(LPSTR devicePath);
	vector<Wiimote*> wmCollection;

private:
	Wiimote* myWiimote;
};
```

Esta cabecera se encarga de detectar los Wiimotes conectados y disponibles en el equipo, para realizar la conexión con aquellos que desee el programador.

La cabecera Wiimote.h en conjunto con el código Wiimote.cpp, contienen la definición de todos los registros del Wiimote, desde la activación de los LEDs y la detección de los botones presionados, hasta la lectura de los acelerómetros. De aquí se obtuvieron las funciones utilizadas en el desarrollo de la WiimoteAPI, las cuales se muestran en la siguiente tabla:

Función	Objetivo
`WiimoteCollection()`	Define los vectores de los Wiimotes a utilizar.
`FindAllWiimotes()`	Encuentra los Wiimotes reconocidos por Windows.
`wmCollection.begin()`	Inicia el conteo de Wiimotes del sistema.
`Connect()`	Conecta el Wiimote definido por el usuario.
`mWiimoteState.ExtensionType`	Define el tipo de extensión a utilizar en el puerto de expansión del Wiimote.
`mWiimoteState.Extension`	Bandera que indica si existe una extensión conectada en el puerto de expansión del Wiimote.
`~Wiimote()`	Desconecta el Wiimote definido por el usuario.
`SetReportType(InputReport::ExtensionAccel, true)`	Define el tipo de muestreo; en este caso se activan el puerto de expansión y los acelerómetros.
`SetLEDs`	Activa los LEDs.
`mWiimoteState.ButtonState`	Conjunto de banderas que indican el estado de los botones del Wiimote.
`mWiimoteState.AccelState.RawValues`	Valores enteros que indican las posiciones de los ejes X, Y y Z del acelerómetro.
`mWiimoteState.NunchukState`	Conjunto de banderas que indican el estado de los botones C y Z del Nunchuck.
`mWiimoteState.NunchukState.RawJoystick`	Valores enteros que indican la posición, en un plano cartesiano con coordenadas X y Y, del joystick del Nunchuck.

5.3.2. Programación de la WiimoteAPI en lenguaje C++ con WinAPI.

En la programación de esta aplicación, se realizaron comentarios de código en inglés con el propósito de hacer un programa al alcance de cualquiera, internacionalmente si es necesario. El código principal consta de varias partes, la primera es la definición de bibliotecas a utilizar:

```
#pragma comment(lib, "setupapi.lib")
#include <iostream>
#include <assert.h>
#include "WiimoteCollection.h"
#include <windows.h>
```

A continuación se definen las variables globales a utilizar:

```
//Variables for WINAPI:
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
HINSTANCE g_hinst;

//Variables
	POINT xyp;	//Variable to get cursor position.
	int cx = GetSystemMetrics(SM_CXSCREEN);  //Screen width.
	int cy = GetSystemMetrics(SM_CYSCREEN);  //Screen height.
	int pace, sens;		//Mouse sensibility parameters.
	bool pressed1, pressed2, pressed3, pressed4, pressed5, pressed6, pressed7,
	pressed8, pressed9, pressed10, pressed11, pressed12, pressed13, pressed14,
	pressed15, pressed16;		//Pressed buttons flags.
	int x, y;	//Accels calibration coordinates.
	int dx, dy;	//Distance between accels coordinates and cursor position.
	int unic;	//Counter for keys.

//Texts items:
	static HWND hwndCombo, hwndCombo2, hwndCombo3, hwndCombo4, hwndCombo5, hwndCombo6,
	hwndCombo7, hwndCombo8, hwndCombo9, hwndCombo10, hwndCombo11, hwndCombo12, hwndCombo13,
	hwndCombo14, hwndCombo15, hwndCombo16;
	const CHAR *items[] = {"", "LEFT CLICK", "RIGHT CLICK", "LEFT", "UP", "RIGHT", "DOWN",
	"PAGE UP", "PAGE DOWN", "END", "HOME", "ENTER",  "A","B", "C", "D", "E", "F", "G", "H", "I",
	"J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "0",
	"1", "2", "3", "4", "5", "6", "7", "8", "9", ".", "-", "/", "BACKSPACE", "TAB", "PAUSE",
	"ESC", "SPACEBAR", "PRINT", "INSERT", "DELETE"};
	const CHAR *buttons[] =  { "UP", "DOWN", "LEFT", "RIGHT", "MINUS", "PLUS", "HOME", "ONE",
	"A", "B", "TWO", "C", "Z"};
	char choice[15];
	int B1=0x00, B2=0x00, B3=0x00, B4=0x00, B5=0x00, B6=0x00, B7=0x00, B8=0x00, B9=0x00,
	B10=0x00, B11=0x00, B12=0x00, B13=0x00, B14=0x00, B15=0x00;
	const CHAR *scale[] = {"1","2","3","4","5","6","7","8","9","10"};
	int i,j=0;
	bool nunflag;
	int itemsize=sizeof(items)/4;

//KEYBOARD AND MOUSE ACTIONS DEFINITIONS:
	char empty[15]="", leftk[15]="LEFT", up[15]="UP", rightk[15]="RIGHT", down[15]="DOWN",
	pageup[15]="PAGE UP", pagedown[15]="PAGE DOWN", endk[15]="END", home[15]="HOME",
	enter[15]="ENTER", lclick[15]="LEFT CLICK", rclick[15]="RIGHT CLICK", period[15]=".",
	minusk[15]="-", slash[15]="/", bspace[15]="BACKSPACE", tab[15]="TAB", pause[15]="PAUSE",
	esc[15]="ESC", spacebar[15]="SPACEBAR", print[15]="PRINT", insert[15]="insert",
	deletek[15]="DELETE";
```

Después vienen las declaraciones de funciones.

Esta primera función se encarga de identificar los datos ingresados por el usuario en los distintos campos tipo combobox; cada dato se convierte en su valor UNICODE, para posteriormente ser reconocido por el teclado:

```
//Function to compare defined keys on Wiimote to real keyboard values:
int comparison(static HWND hc, char selection[15], int uni) {

	GetWindowText(hc, selection, 15);
	if (strcmp (empty, selection) == 0){
		uni=0x00;
	}
	else if (strcmp (lclick, selection)== 0){
		uni=0x01;
	}
	else if (strcmp (rclick, selection)== 0){
		uni=0x02;
	}
	else if (strcmp (bspace, selection)== 0){
		uni=0x08;
	}
	else if (strcmp (tab, selection)== 0){
		uni=0x09;
	}
	else if (strcmp (enter, selection) == 0){
		uni=0x0D;
	}
	else if (strcmp (pause, selection) == 0){
		uni=0x13;
	}
	else if (strcmp (esc, selection) == 0){
		uni=0x1b;
	}
	else if (strcmp (spacebar, selection) == 0){
		uni=0x20;
	}
	else if (strcmp (pageup, selection)== 0){
		uni=0x21;
	}
	else if (strcmp (pagedown, selection)== 0){
		uni=0x22;
	}
	else if (strcmp (endk, selection)== 0){
		uni=0x23;
	}
	else if (strcmp (home, selection)== 0){
		uni=0x24;
	}
	else if (strcmp (leftk, selection)== 0){
		uni=0x25;
	}
	else if (strcmp (up, selection)== 0){
		uni=0x26;
	}
	else if (strcmp (rightk, selection)== 0){
		uni=0x27;
	}
```

```
	else if (strcmp (down, selection)== 0){
		uni=0x28;
	}
	else if (strcmp (print, selection)== 0){
		uni=0x2C;
	}

	else if (strcmp (insert, selection)== 0){
		uni=0x2D;
	}
	else if (strcmp (deletek, selection)== 0){
		uni=0x2E;
	}
	else if (strcmp (period, selection)== 0){
		uni=0x6E;
	}
	else if (strcmp (minusk, selection)== 0){
		uni=0x6D;
	}
	else if (strcmp (slash, selection)== 0){
		uni=0x6F;
	}
	else{
		uni = cin.widen(selection[0]);
	}
	return uni;
}
```

A continuación se declara una función para identificar el tipo de buffer a definir, la siguiente función se encarga de clasificar los buffers para los casos posibles de teclado o ratón:

```
//Function to declare buffer types:
void buf(INPUT *buffer, int uni){

	if(uni>0x07){
		buffer->type		= INPUT_KEYBOARD;
		buffer->ki.wVk		= uni;
		buffer->ki.wScan	= 0;
		buffer->ki.time		= 0;
		buffer->ki.dwExtraInfo	= 0;
	}

	else {
		buffer->type		= INPUT_MOUSE;
		buffer->mi.dx		= 0;
		buffer->mi.dy		= 0;
		buffer->mi.mouseData	= 0;
		buffer->mi.time		= 0;
		buffer->mi.dwExtraInfo	= 0;
	}

	return;
}
```

Posteriormente se define la función encargada de enviar el comando ya sea al ratón o al teclado:

```
//Function to send input actions:
bool act(INPUT *buffer, int uni, bool pressed){

	if(uni>0x07 && pressed==false){
		buffer->ki.dwFlags = KEYEVENTF_UNICODE;
		SendInput(1,buffer,sizeof(INPUT));
		pressed=true;
		return pressed;
	}

	else if(uni==0x02 && pressed==false){
		buffer->mi.dwFlags = MOUSEEVENTF_RIGHTDOWN;

		SendInput(1,buffer,sizeof(INPUT));
		pressed=true;
		return pressed;
	}

	else if(uni==0x01 && pressed==false){
		buffer->mi.dwFlags = MOUSEEVENTF_LEFTDOWN;

		SendInput(1,buffer,sizeof(INPUT));
		pressed=true;
		return pressed;
	}
}
```

Después está definida una función que se encarga de "liberar" los botones, ya sean de ratón o de teclado. Esta función es muy importante para realizar una simulación adecuada de las tareas de estos dispositivos periféricos:

```
//Function to release buttons:
bool rel(bool button, bool pressed, INPUT *buffer, int uni){

	if (!button && pressed){
			if(uni==0x01){
				buffer->mi.dwFlags = MOUSEEVENTF_LEFTUP;
				SendInput(1,buffer,sizeof(INPUT));
				pressed=false;
				return pressed;
			}
			else if(uni==0x02){
				buffer->mi.dwFlags = MOUSEEVENTF_RIGHTUP;
				SendInput(1,buffer,sizeof(INPUT));
				pressed=false;
				return pressed;
			}
			else if (uni>0x07){
				pressed=false;
				return pressed;
			}
	}
}
```

Para poder desplegar la ventana de la WiimoteAPI se necesita de una función **principal** que define las propiedades de la misma, como tamaño, nombre, color, etc.

```
//Main Window displaying:
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow ){

  MSG  msg;
  WNDCLASS wc = {0};
  wc.style         = CS_HREDRAW | CS_VREDRAW;
  wc.cbClsExtra    = 0;
  wc.cbWndExtra    = 0;
  wc.lpszMenuName =  0;
  wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
  wc.lpszClassName = "Wiimote";
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0,IDC_ARROW);

  RegisterClass(&wc);
  CreateWindow( wc.lpszClassName, TEXT("WIIMOTE API   -   -   -   -   -   BY ANDREW (DARKCYBORG)"),
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                350, 150, 550, 550, 0, 0, hInstance, 0);

        while( GetMessage(&msg, NULL, 0, 0)) {
               TranslateMessage(&msg);
               DispatchMessage(&msg);
        }

        return RegisterClass(&wc);
        return (int) msg.wParam;
}
```

Finalmente, se declara la función que define cada objeto de la interfaz gráfica y que contiene el código del ciclo de muestreo de las señales de entrada del Wiimote.

La primera parte de esta función consiste en la declaración de unas cuantas variables locales, como la bandera del Nunchuck, que indica si está conectado, y algunas variables de objetos de la interfaz gráfica.

```
//Inner window items displaying and wiimote binding:
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
  nunflag=false;

  HWND hwndsens;
  HWND hwndLeftLabel;
  HWND hwndRightLabel;
```

En la segunda parte se inicia una conexión rápida del Wiimote para verificar que el Nunchuck esté conectado al puerto de expansión del Wiimote y de ser así se guarda el valor de la bandera de conexión como verdadero:

```
switch(msg)
{
	case WM_CREATE:
	{
		//Nunchuk connection verification:
		WiimoteCollection* wmColl = new WiimoteCollection();
		wmColl->FindAllWiimotes();
		vector<Wiimote*>::const_iterator wmCounter;
		wmCounter = wmColl->wmCollection.begin();
		wmColl->wmCollection[0]->Connect();
		wmColl->wmCollection[0]->mWiimoteState.ExtensionType=Nunchuk;

		if(wmColl->wmCollection[0]->mWiimoteState.Extension){
			nunflag=true;
		}
```

La tercera parte consiste en crear las etiquetas, correspondientes a los distintos objetos tipo combobox, de cada botón del Nunchuck.

```
		if(nunflag){
			CreateWindow(TEXT("STATIC"), TEXT("- NUNCHUK BUTTONS BINDING -"),
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    230, 8, 250 25, hwnd, NULL, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[11],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 43, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[12],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 83, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[0],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 123, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[1],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 163, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[2],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 203, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[3],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 243, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);
```

La cuarta parte consta de la creación de los objetos tipo combobox para definir los botones del Nunchuck; en caso de no estar conectado, se crea un mensaje de aviso al usuario para conectarlo, aunque es opcional. Y se termina la conexión rápida al Wiimote la cual se realizó en fracciones de segundo:

```
                hwndCombo10 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 40, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo11 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 80, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo12 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 120, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo13 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 160, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo14 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 200, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo15 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 240, 110, 300, hwnd, NULL, g_hinst, NULL);
      }

      else{
             CreateWindow(TEXT("STATIC"), TEXT("PLEASE, CONNECT A NUNCHUK!"),
    WS_CHILD | WS_VISIBLE | SS_LEFT,
    270, 38, 130, 30, hwnd, NULL, NULL, NULL);
      }

      wmColl->wmCollection[0]->~Wiimote();
      delete wmColl;
```

Cabe destacar que esta conexión no se puede mantener activa mientras el usuario define las funciones de los botones del Wiimote, ya que el despliegue de la interfaz gráfica y el muestreo del Wiimote son dos procesos cíclicos, que no pueden trabajar en paralelo.

La quinta parte se encarga de crear las etiquetas de los botones del Wiimote para reconocer sus respectivos objetos tipo combobox:

```
            CreateWindow(TEXT("STATIC"), TEXT("- BUTTONS BINDING -"),
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                20, 8, 180, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[0],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 43, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[1],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 83, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[2],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 123, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[3],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 163, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[4],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 203, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[5],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 243, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[6],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 283, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[7],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 323, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[8],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 363, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[9],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 403, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);

            CreateWindow(TEXT("STATIC"), buttons[10],
                WS_CHILD | WS_VISIBLE | SS_LEFT,
                10, 443, 60, 30,
                hwnd, (HMENU) 1, NULL, NULL);
```

En la sexta parte se crean los objetos tipo combobox de los botones del Wiimote:

```
hwndCombo = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 40, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo2 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 80, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo3 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 120, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo4 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 160, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo5 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 200, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo6 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 240, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo7 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 280, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo8 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 320, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo9 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 360, 110, 300, hwnd, NULL, g_hinst, NULL);
```

La séptima parte de esta función se encarga de crear varios objetos: las etiquetas de los dos botones con funciones fijas, las cuales son “Activar mouse” y “Detener Wiimote”; también la etiqueta del botón de inicio de funcionamiento del Wiimote; y las etiquetas correspondientes a la barra de ajuste de sensibilidad del Wiimote.

```
  CreateWindow(TEXT("static"), TEXT("ACTIVATE MOUSE"),
WS_CHILD | WS_VISIBLE | SS_LEFT,
75, 402, 150, 25, hwnd, NULL, NULL, NULL);

  CreateWindow(TEXT("static"), TEXT("STOP WIIMOTE"),
WS_CHILD | WS_VISIBLE | SS_LEFT,
75, 442, 150, 25, hwnd, NULL, NULL, NULL);

  CreateWindow(TEXT("button"), TEXT("START WIIMOTE!"),
      WS_VISIBLE | WS_CHILD | SS_CENTER,
      260, 402, 150, 50, hwnd, (HMENU)1, NULL, NULL);

  hwndCombo16 = CreateWindowEx(
  0, TRACKBAR_CLASS, "Trackbar Control",
  WS_CHILD | WS_VISIBLE |
  TBS_AUTOTICKS | TBS_ENABLESELRANGE,
  280, 320, 100, 30,
  hwnd, (HMENU) 3, NULL, NULL);

  hwndsens = CreateWindow("STATIC", "MOUSE SENSIBILITY:",
             WS_CHILD | WS_VISIBLE,
             260, 300, 150, 25,
             hwnd, (HMENU)1, NULL, NULL);

  hwndLeftLabel = CreateWindow("STATIC", "0",
             WS_CHILD | WS_VISIBLE,
             0, 0, 15, 15,
             hwnd, (HMENU)1, NULL, NULL);

  hwndRightLabel = CreateWindow("STATIC", "10",
             WS_CHILD | WS_VISIBLE,
             0, 0, 15, 15,
             hwnd, (HMENU)2, NULL, NULL);
```

La octava sección se encarga de crear los contenidos de cada objeto tipo combobox, para que el usuario seleccione de esta lista de funciones la que desee asignar a cada botón:

```
        for ( i = 0; i < itemsize; i++ ) {
                        SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo2, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo3, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo4, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo5, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo6, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo7, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo8, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo9, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo10, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo11, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo12, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo13, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo14, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo15, CB_ADDSTRING, 0, (LPARAM) items[i]);
         }

               SendMessage(hwndCombo16, TBM_SETRANGE,  TRUE,  MAKELONG(0, 10));
               SendMessage(hwndCombo16, TBM_SETPAGESIZE, 0,  1);
               SendMessage(hwndCombo16, TBM_SETTICFREQ, 1, 0);
               SendMessage(hwndCombo16, TBM_SETPOS, 5, 5);
               SendMessage(hwndCombo16, TBM_SETBUDDY, TRUE, (LPARAM) hwndLeftLabel);
               SendMessage(hwndCombo16, TBM_SETBUDDY, FALSE, (LPARAM) hwndRightLabel);


               SendMessage(hwndCombo, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo2, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo3, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo4, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo5, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo6, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo7, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo8, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo9, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo10, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo11, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo12, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo13, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo14, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo15, CB_SETCURSEL, 0, 0);
        }
```

La novena parte inicia con la activación del botón de inicio del Wiimote. Primero se desactiva el botón para que el usuario identifique el inicio del funcionamiento del Wiimote. Posteriormente se hacen las comparaciones de las funciones para cada botón del Wiimote que asignó el usuario y así identificar de qué tipo son. Después se activa la conexión del Wiimote y se encienden dos LEDs para hacer aún más evidente que el Wiimote está activado y listo para usarse:

```
        case WM_COMMAND:
        {

               if (LOWORD(wParam) == 1) {

                      CreateWindow(TEXT("button"), TEXT("START WIIMOTE!"),
                   WS_VISIBLE | WS_CHILD | SS_CENTER,
                   260, 402, 150, 50, hwnd, (HMENU)1, NULL, NULL);

                      B1=comparison(hwndCombo,choice,B1);
                      B2=comparison(hwndCombo2,choice,B2);
                      B3=comparison(hwndCombo3,choice,B3);
                      B4=comparison(hwndCombo4,choice,B4);
                      B5=comparison(hwndCombo5,choice,B5);
                      B6=comparison(hwndCombo6,choice,B6);
                      B7=comparison(hwndCombo7,choice,B7);
                      B8=comparison(hwndCombo8,choice,B8);
                      B9=comparison(hwndCombo9,choice,B9);
                      B10=comparison(hwndCombo10,choice,B10);
                      B11=comparison(hwndCombo11,choice,B11);
                      B12=comparison(hwndCombo12,choice,B12);
                      B13=comparison(hwndCombo13,choice,B13);
                      B14=comparison(hwndCombo14,choice,B14);
                      B15=comparison(hwndCombo15,choice,B15);

               WiimoteCollection* wmColl = new WiimoteCollection();
               wmColl->FindAllWiimotes();
               vector<Wiimote*>::const_iterator wmCounter;
               wmCounter = wmColl->wmCollection.begin();

               wmColl->wmCollection[0]->Connect();
               wmColl->wmCollection[0]->SetReportType(InputReport::ExtensionAccel, true);
               wmColl->wmCollection[0]->SetLEDs(true, false, false, true);
```

Posteriormente, la décima parte consta de la creación de los buffers para cada tipo de acción correspondiente a los botones del Wiimote, cada buffer representa una señal de entrada al sistema, que puede ser de teclado o de ratón para este caso. Después se configura el puerto de expansión del Wiimote para reconocer al Nunchuck. A continuación se definen algunas variables para inicializar el puntero del ratón y se obtiene el valor de la sensibilidad definido por el usuario desde la interfaz gráfica:

```
        INPUT *buffer1       = new INPUT;
        buf(buffer1,B1);

        INPUT *buffer2       = new INPUT;
        buf(buffer2,B2);

        INPUT *buffer3       = new INPUT;
        buf(buffer3,B3);

        INPUT *buffer4       = new INPUT;
        buf(buffer4,B4);

        INPUT *buffer5       = new INPUT;
        buf(buffer5,B5);

        INPUT *buffer6       = new INPUT;
        buf(buffer6,B6);

        INPUT *buffer7       = new INPUT;
        buf(buffer7,B7);

        INPUT *buffer8       = new INPUT;
        buf(buffer8,B8);

        INPUT *buffer9       = new INPUT;
        buf(buffer9,B9);

        INPUT *buffer10      = new INPUT;
        buf(buffer10,B10);

        INPUT *buffer11      = new INPUT;
        buf(buffer11,B11);

        INPUT *buffer12      = new INPUT;
        buf(buffer12,B12);

        INPUT *buffer13      = new INPUT;
        buf(buffer13,B13);

        INPUT *buffer14      = new INPUT;
        buf(buffer14,B14);

        INPUT *buffer15      = new INPUT;
        buf(buffer15,B15);

        //Nunchuck activation:
        wmColl->wmCollection[0]->mWiimoteState.ExtensionType=Nunchuk;

        //Variables to initialize mouse movement:
        SetCursorPos(cx/2,cy/2);
        GetCursorPos(&xyp);
        x=120,y=120;

        //Mouse sensibility parameters:
        pace=SendMessage(hwndCombo16, TBM_GETPOS, 0, 0);
        sens=1;
```

La décima primera parte consiste en iniciar el ciclo de muestreo del acelerómetro del Wiimote:

```
while(!wmColl->wmCollection[0]->mWiimoteState.ButtonState.Two()) {

		Sleep(2*(10-pace));  //Sensibility assignment.
		dx=0,dy=0;

		//Mouse development:
		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.X<x-sens)){
			dx=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.X-x);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}

		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.X>x+sens)){
			dx=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.X-x);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}

		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.Y<y-sens)){
			dy=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.Y-y);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}

		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.Y>y+sens)){
			dy=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.Y-y);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}
```

En la décima segunda parte se realizan las acciones si están presionados los botones del Wiimote:

```
//Functions binding:
if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Up()){

	pressed1=act(buffer1,B1,pressed1);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Down()){

	pressed2=act(buffer2,B2,pressed2);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Left()){

	pressed3=act(buffer3,B3,pressed3);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Right()){

	pressed4=act(buffer4,B4,pressed4);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Minus()){

	pressed5=act(buffer5,B5,pressed5);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Plus()){

	pressed6=act(buffer6,B6,pressed6);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Home()){

	pressed7=act(buffer7,B7,pressed7);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.One()){
	pressed8=act(buffer8,B8,pressed8);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.A()){

	pressed9=act(buffer9,B9,pressed9);
}

if(wmColl->wmCollection[0]->mWiimoteState.NunchukState.C){

	pressed10=act(buffer10,B10,pressed10);
}

if(wmColl->wmCollection[0]->mWiimoteState.NunchukState.Z){

	pressed11=act(buffer11,B11,pressed11);
}
```

La décima tercera parte se encarga de realizar el mapeo del movimiento del joystick del Nunchuck para poder asignarle cuatro botones, los cuales serían arriba, abajo, izquierda y derecha. Después se liberan los botones que fueron presionados, tanto del Wiimote como del Nunchuck:

```
	//Nunchuck direction mapping:
		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y>
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y>165) ){
			pressed12=act(buffer12,B12,pressed12);
		}

		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y<
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y<105) ){
			pressed13=act(buffer13,B13,pressed13);
		}

		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X<
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X<105) ){
			pressed14=act(buffer14,B14,pressed14);
		}

		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X>
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X>165) ){
			pressed15=act(buffer15,B15,pressed15);
		}

		//Buttons release:
		pressed1=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Up(),pressed1,buffer1,B1);
		pressed2=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Down(),pressed2,buffer2,B2);
		pressed3=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Left(),pressed3,buffer3,B3);
		pressed4=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Right(),pressed4,buffer4,B4);
		pressed5=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Minus(),pressed5,buffer5,B5);
		pressed6=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Plus(),pressed6,buffer6,B6);
		pressed7=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Home(),pressed7,buffer7,B7);
		pressed8=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.One(),pressed8,buffer8,B8);
		pressed9=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.A(),pressed9,buffer9,B9);
		pressed10=rel(wmColl->wmCollection[0]->mWiimoteState.NunchukState.C,pressed10,buffer10,B10);
		pressed11=rel(wmColl->wmCollection[0]->mWiimoteState.NunchukState.Z,pressed11,buffer11,B11);

		if(	(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y<165)
			&&
			(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y>105)
			&&
			(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X>105)
			&&
			(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X<165)
			){
				pressed12=rel(false,pressed12,buffer12,B12);
				pressed13=rel(false,pressed13,buffer13,B13);
				pressed14=rel(false,pressed14,buffer14,B14);
				pressed15=rel(false,pressed15,buffer15,B15);
		}
```

Y finalmente, se actualiza la posición del puntero. Una vez que el usuario presiona el botón “dos” del Wiimote, se finaliza el ciclo de muestreo y se eliminan los buffers creados para que el usuario pueda volver a definirlos si lo necesita; a continuación se desactiva el Wiimote y la WiimoteAPI vuelve a estar disponible. En caso de que el usuario de click en el botón de salir, como en cualquier otra ventana, se terminará la aplicación.

```
        GetCursorPos(&xyp);

        }

        delete buffer1;
        delete buffer2;
        delete buffer3;
        delete buffer4;
        delete buffer5;
        delete buffer6;
        delete buffer7;
        delete buffer8;
        delete buffer9;
        delete buffer10;
        delete buffer11;
        delete buffer12;
        delete buffer13;
        delete buffer14;
        delete buffer15;

        wmColl->wmCollection[0]->~Wiimote();

        delete wmColl;

                }

                break;
      }

         case WM_DESTROY:{
                PostQuitMessage(0);
          break;
         }

  }

  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```