EBOOK

Michael Andre Franiatte

C# and C++ Codes in Gamepad Libraries to Play PC Games

Wiimote and Xbox360 Controller on PC Games

EBOOK

C# and C++ Codes in Gamepad Libraries to Play PC Games

Wiimote and Xbox360 Controller on PC Games

Michael Franiatte
06/21/2018

C# and C++ Codes in Gamepad Libraries to Play PC Games

Michael Franiatte*

Abstract

The gamepads of consoles have invaded PC to play games becoming the replacers of keyboard and mouse, with the same accuracy and more easy to use for a best comfortable experience of gameplay. The codes presented here are for beginner programmers wanting to know how to simulate keyboard and mouse events in order to play PC games using XBox360 gamepad, Wiimote/Nunchuck/Sensor bar, Wii classic controller, Wii guitar hero 3 drums and guitar. This paper gives some information that wasn't finding previously by other authors. It correspond to make run keyboard events to simulate a key press down and up, and assembling snippets to be recognized by all PC games. Viewers can take this paper as a study on the best codes and equations to play the best in all games. Also, the gamers can take this paper to customize their own gamepad with scripts.

Keywords: *gamepads, PC, gameplay, games, codes, scripts*

* Author correspondence: michael.franiatte@gmail.com

1. Introduction

After the first chapter to introduce what kind of materials are needed, how to connect a Wiimote on a PC and how to create a C# program for using Wii hardware, the other chapters are for giving entire C# and C++ codes to play very well at PC games as FPS and Racing with Wiimote/Nunchuck/Sensor bar, Wiimote alone as a gamepad and Xbox360 controller.

1.1. Materials needed

For programming and playing, what are needed, it's a good newsiest computer, a wireless sensor bar or a Wii console for infrared using by the Wiimote to replace the mouse axis, a Wiimote and Nunchuck, a Bluetooth key to connect the Wiimote to the PC, a Xbox360 controller, Visual studio C# express edition with WiimoteLib_1.7_src, Visual Studio C++ express edition with XBOX360test_src (visual studio version 2008 or 2010), PC games and this book.

1.2. Connect a Wiimote

A simple Bluetooth key inserted in a usb port can be used to connect a Wiimote on a PC. Connect the Wiimote by adding a peripheral with the Bluetooth icon without code. Bluetooth key is auto-recognized since windows vista, after using the Wiimote, disconnect it by the same way when viewing the peripherals added. Sometimes the Nunchuck doesn't run with the program made, thus the Nunchuck need to be disconnected/reconnected while the program running, which must be closed with Alt+Tab, Alt+F4, for open again the program a second time, after it run (full screen game with Alt+Enter).

1.3. Create a C# Program

The example for make a C# program presented here use the library WiimoteLib.DLL as a reference to add in a new project. The dll must be copy in the folder obj of the new project created after make a build. It's found with the WiimoteLib solution in the debug folder of WiimoteLib_1.7_src. The new project is composed of a Form application. The solution is called WiimoteOnCoD7. To add a background picture, in the properties of the project, in the tab resources, you need to add a picture with the button new picture in add a resource, then load a picture, and after to go on the form design and select the background image in the tab properties of the form. With the toolbox you can insert textbox for add information, also insert a trackbar like a motion track bar to adjust an autocentering when you center the Wiimote in the middle of the sensor bar, like to stop the axis moves in FPS when you going to the middle of the sensor bar. In the designer code page of the form, you must verify the + sign in this.Load += new System.EventHandler(this.WiimoteOnCoD7_Load); In this page set to static declaration of combobox, checkbox, textbox, label and remove this. before these terms. Under properties of the Form, in the tab Events, you can double click on FormLoad and FormClosed to enable event functions.

You can change the name of the project to publish with another name the program in the bin\release folder. You must copy this name, in the properties of the project, in the tab application, in the box assembly name. The picture of the icon can be converting on the site http://convertir-une-image.com, to change in properties of the project and in Form1 design. You can use http://www.online-image-editor.com to make transparent Wiimote picture and use PowerPoint and Paint to edit the background image of the program. The Background image layout can be set as stretch in properties of the form. You can set RGB colour of comboboxes, checkboxes, command buttons by selecting a somehow colour and change the numbers shown by numbers shown with Paint when you pick up a colour and display colour panel. The Radio buttons must be set with property AutoCheck false, otherwise check of radio buttons in group boxes would be exclusive.

2. List of Key Codes

```
public enum VirtualKeyCode : ushort // UInt16
{
    LBUTTON = 0x01,
    RBUTTON = 0x02,
    CANCEL = 0x03,
    MBUTTON = 0x04,
    XBUTTON1 = 0x05,
    XBUTTON2 = 0x06,
    BACK = 0x08,
    TAB = 0x09,
    CLEAR = 0x0C,
    RETURN = 0x0D,
    SHIFT = 0x10,
    CONTROL = 0x11,
    MENU = 0x12,
    PAUSE = 0x13,
    CAPITAL = 0x14,
    KANA = 0x15,
    HANGEUL = 0x15,
    HANGUL = 0x15,
    JUNJA = 0x17,
    FINAL = 0x18,
    HANJA = 0x19,
    KANJI = 0x19,
    ESCAPE = 0x1B,
    CONVERT = 0x1C,
    NONCONVERT = 0x1D,
    ACCEPT = 0x1E,
    MODECHANGE = 0x1F,
    SPACE = 0x20,
    PRIOR = 0x21,
```

```
NEXT = 0x22,
END = 0x23,
HOME = 0x24,
LEFT = 0x25,
UP = 0x26,
RIGHT = 0x27,
DOWN = 0x28,
SELECT = 0x29,
PRINT = 0x2A,
EXECUTE = 0x2B,
SNAPSHOT = 0x2C,
INSERT = 0x2D,
DELETE = 0x2E,
HELP = 0x2F,
APOSTROPHE = 0xDE,
VK_0 = 0x30,
VK_1 = 0x31,
VK_2 = 0x32,
VK_3 = 0x33,
VK_4 = 0x34,
VK_5 = 0x35,
VK_6 = 0x36,
VK_7 = 0x37,
VK_8 = 0x38,
VK_9 = 0x39,
VK_A = 0x41,
VK_B = 0x42,
VK_C = 0x43,
VK_D = 0x44,
VK_E = 0x45,
VK_F = 0x46,
VK_G = 0x47,
VK_H = 0x48,
VK_I = 0x49,
VK_J = 0x4A,
VK_K = 0x4B,
VK_L = 0x4C,
VK_M = 0x4D,
VK_N = 0x4E,
VK_O = 0x4F,
VK_P = 0x50,
VK_Q = 0x51,
VK_R = 0x52,
VK_S = 0x53,
VK_T = 0x54,
VK_U = 0x55,
VK_V = 0x56,
VK_W = 0x57,
VK_X = 0x58,
VK_Y = 0x59,
VK_Z = 0x5A,
LWIN = 0x5B,
RWIN = 0x5C,
APPS = 0x5D,
SLEEP = 0x5F,
NUMPAD0 = 0x60,
NUMPAD1 = 0x61,
NUMPAD2 = 0x62,
NUMPAD3 = 0x63,
NUMPAD4 = 0x64,
NUMPAD5 = 0x65,
NUMPAD6 = 0x66,
NUMPAD7 = 0x67,
NUMPAD8 = 0x68,
NUMPAD9 = 0x69,
MULTIPLY = 0x6A,
```

```
ADD = 0x6B,
SEPARATOR = 0x6C,
SUBTRACT = 0x6D,
DECIMAL = 0x6E,
DIVIDE = 0x6F,
F1 = 0x70,
F2 = 0x71,
F3 = 0x72,
F4 = 0x73,
F5 = 0x74,
F6 = 0x75,
F7 = 0x76,
F8 = 0x77,
F9 = 0x78,
F10 = 0x79,
F11 = 0x7A,
F12 = 0x7B,
F13 = 0x7C,
F14 = 0x7D,
F15 = 0x7E,
F16 = 0x7F,
F17 = 0x80,
F18 = 0x81,
F19 = 0x82,
F20 = 0x83,
F21 = 0x84,
F22 = 0x85,
F23 = 0x86,
F24 = 0x87,
NUMLOCK = 0x90,
SCROLL = 0x91,
LSHIFT = 0xA0,
RSHIFT = 0xA1,
LCONTROL = 0xA2,
RCONTROL = 0xA3,
LMENU = 0xA4,
RMENU = 0xA5,
BROWSER_BACK = 0xA6,
BROWSER_FORWARD = 0xA7,
BROWSER_REFRESH = 0xA8,
BROWSER_STOP = 0xA9,
BROWSER_SEARCH = 0xAA,
BROWSER_FAVORITES = 0xAB,
BROWSER_HOME = 0xAC,
VOLUME_MUTE = 0xAD,
VOLUME_DOWN = 0xAE,
VOLUME_UP = 0xAF,
MEDIA_NEXT_TRACK = 0xB0,
MEDIA_PREV_TRACK = 0xB1,
MEDIA_STOP = 0xB2,
MEDIA_PLAY_PAUSE = 0xB3,
LAUNCH_MAIL = 0xB4,
LAUNCH_MEDIA_SELECT = 0xB5,
LAUNCH_APP1 = 0xB6,
LAUNCH_APP2 = 0xB7,
OEM_1 = 0xBA,
OEM_PLUS = 0xBB,
OEM_COMMA = 0xBC,
OEM_MINUS = 0xBD,
OEM_PERIOD = 0xBE,
OEM_2 = 0xBF,
OEM_3 = 0xC0,
OEM_4 = 0xDB,
OEM_5 = 0xDC,
OEM_6 = 0xDD,
OEM_7 = 0xDE,
```

```
    OEM_8 = 0xDF,
    OEM_102 = 0xE2,
    PROCESSKEY = 0xE5,
    PACKET = 0xE7,
    ATTN = 0xF6,
    CRSEL = 0xF7,
    EXSEL = 0xF8,
    EREOF = 0xF9,
    PLAY = 0xFA,
    ZOOM = 0xFB,
    NONAME = 0xFC,
    PA1 = 0xFD,
    OEM_CLEAR = 0xFE,
}
public enum bScan : ushort // UInt16
{
    LBUTTON = 0x01,
    RBUTTON = 0x02,
    CANCEL = 0x39,
    MBUTTON = 0x04,
    XBUTTON1 = 0x05,
    XBUTTON2 = 0x06,
    BACK = 0x0E,
    TAB = 0x0F,
    CLEAR = 0x0C,
    RETURN = 0x1C,
    SHIFT = 0x2A,
    CONTROL = 0x1D,
    MENU = 0x38,
    PAUSE = 0x13,
    CAPITAL = 0x3A,
    KANA = 0x15,
    HANGEUL = 0x15,
    HANGUL = 0x15,
    JUNJA = 0x17,
    FINAL = 0x18,
    HANJA = 0x19,
    KANJI = 0x19,
    ESCAPE = 0x01,
    CONVERT = 0x1C,
    NONCONVERT = 0x1D,
    ACCEPT = 0x1E,
    MODECHANGE = 0x1F,
    SPACE = 0x39,
    PRIOR = 0xC9,
    NEXT = 0xD1,
    END = 0xCF,
    HOME = 0xC7,
    LEFT = 0xCB,
    UP = 0xC8,
    RIGHT = 0xCD,
    DOWN = 0xD0,
    SELECT = 0x29,
    PRINT = 0x2A,
    EXECUTE = 0x2B,
    SNAPSHOT = 0x2C,
    INSERT = 0xD2,
    DELETE = 0xD3,
    HELP = 0x3B,
    APOSTROPHE = 0xDE,
    VK_0 = 0x0B,
    VK_1 = 0x02,
    VK_2 = 0x03,
    VK_3 = 0x04,
    VK_4 = 0x05,
    VK_5 = 0x06,
```

```
VK_6 = 0x07,
VK_7 = 0x08,
VK_8 = 0x09,
VK_9 = 0x0A,
VK_A = 0x1E,
VK_B = 0x30,
VK_C = 0x2E,
VK_D = 0x20,
VK_E = 0x12,
VK_F = 0x21,
VK_G = 0x22,
VK_H = 0x23,
VK_I = 0x17,
VK_J = 0x24,
VK_K = 0x25,
VK_L = 0x26,
VK_M = 0x32,
VK_N = 0x31,
VK_O = 0x18,
VK_P = 0x19,
VK_Q = 0x10,
VK_R = 0x13,
VK_S = 0x1F,
VK_T = 0x14,
VK_U = 0x16,
VK_V = 0x2F,
VK_W = 0x11,
VK_X = 0x2D,
VK_Y = 0x15,
VK_Z = 0x2C,
LWIN = 0x5B,
RWIN = 0x5C,
APPS = 0x5D,
SLEEP = 0x5F,
NUMPAD0 = 0x52,
NUMPAD1 = 0x4F,
NUMPAD2 = 0x50,
NUMPAD3 = 0x51,
NUMPAD4 = 0x4B,
NUMPAD5 = 0x4C,
NUMPAD6 = 0x4D,
NUMPAD7 = 0x47,
NUMPAD8 = 0x48,
NUMPAD9 = 0x49,
MULTIPLY = 0x37,
ADD = 0xDD,
SEPARATOR = 0x6C,
SUBTRACT = 0x4A,
DECIMAL = 0x6E,
DIVIDE = 0xB5,
F1 = 0x3B,
F2 = 0x3C,
F3 = 0x3D,
F4 = 0x3E,
F5 = 0x3F,
F6 = 0x40,
F7 = 0x41,
F8 = 0x42,
F9 = 0x43,
F10 = 0x44,
F11 = 0x57,
F12 = 0x58,
F13 = 0x64,
F14 = 0x65,
F15 = 0x66,
F16 = 0x67,
```

```
        F17 = 0x80,
        F18 = 0x81,
        F19 = 0x82,
        F20 = 0x83,
        F21 = 0x84,
        F22 = 0x85,
        F23 = 0x86,
        F24 = 0x87,
        NUMLOCK = 0x45,
        SCROLL = 0x46,
        LSHIFT = 0x2A,
        RSHIFT = 0x36,
        LCONTROL = 0x1D,
        RCONTROL = 0x9D,
        LMENU = 0xA4,
        RMENU = 0xA5,
        BROWSER_BACK = 0xA6,
        BROWSER_FORWARD = 0xA7,
        BROWSER_REFRESH = 0xA8,
        BROWSER_STOP = 0xA9,
        BROWSER_SEARCH = 0xAA,
        BROWSER_FAVORITES = 0xAB,
        BROWSER_HOME = 0xAC,
        VOLUME_MUTE = 0xAD,
        VOLUME_DOWN = 0xAE,
        VOLUME_UP = 0xAF,
        MEDIA_NEXT_TRACK = 0xB0,
        MEDIA_PREV_TRACK = 0xB1,
        MEDIA_STOP = 0xB2,
        MEDIA_PLAY_PAUSE = 0xB3,
        LAUNCH_MAIL = 0xB4,
        LAUNCH_MEDIA_SELECT = 0xB5,
        LAUNCH_APP1 = 0xB6,
        LAUNCH_APP2 = 0xB7,
        OEM_1 = 0xBA,
        OEM_PLUS = 0xBB,
        OEM_COMMA = 0xBC,
        OEM_MINUS = 0xBD,
        OEM_PERIOD = 0xBE,
        OEM_2 = 0xBF,
        OEM_3 = 0xC0,
        OEM_4 = 0xDB,
        OEM_5 = 0xDC,
        OEM_6 = 0xDD,
        OEM_7 = 0xDE,
        OEM_8 = 0xDF,
        OEM_102 = 0xE2,
        PROCESSKEY = 0xE5,
        PACKET = 0xE7,
        ATTN = 0xF6,
        CRSEL = 0xF7,
        EXSEL = 0xF8,
        EREOF = 0xF9,
        PLAY = 0xFA,
        ZOOM = 0xFB,
        NONAME = 0xFC,
        PA1 = 0xFD,
        OEM_CLEAR = 0xFE,
    }
```

3. XBox360 controller (C++ Windows Console)

```
#include <math.h>
#include <iostream>
#include <process.h>
#include <windows.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <string>
#ifndef _XBOX_CONTROLLER_H_
#define _XBOX_CONTROLLER_H_
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <XInput.h>
#pragma comment(lib, "XInput.lib")
class CXBOXController
{
private:
	XINPUT_STATE _controllerState;
	int _controllerNum;
public:
	CXBOXController(int playerNumber);
	XINPUT_STATE GetState();
	bool IsConnected();
	void Vibrate(int leftVal = 0, int rightVal = 0);
};
#endif
CXBOXController::CXBOXController(int playerNumber)
{
	_controllerNum = playerNumber - 1;
}
XINPUT_STATE CXBOXController::GetState()
{
	ZeroMemory(&_controllerState, sizeof(XINPUT_STATE));
	XInputGetState(_controllerNum, &_controllerState);
	return _controllerState;
}
bool CXBOXController::IsConnected()
{
	ZeroMemory(&_controllerState, sizeof(XINPUT_STATE));
	DWORD Result = XInputGetState(_controllerNum, &_controllerState);
	if(Result == ERROR_SUCCESS)
	{
		return true;
	}
	else
	{
		return false;
	}
}
void CXBOXController::Vibrate(int leftVal, int rightVal)
{
	XINPUT_VIBRATION Vibration;
	ZeroMemory(&Vibration, sizeof(XINPUT_VIBRATION));
	Vibration.wLeftMotorSpeed = leftVal;
	Vibration.wRightMotorSpeed = rightVal;
	XInputSetState(_controllerNum, &Vibration);
}
static int WidthS = 400;
	 static int HeightS = 300;
	 static int WA = 2;
	 static int WJ1NJXWD = 2;
	 static int WJ2NJXWD = 2;
	 static int WJ1NJYWD = 2;
	 static int WJ2NJYWD = 2;
	 static int WJ1NJXWU = 2;
	 static int WJ2NJXWU = 2;
	 static int WJ1NJYWU = 2;
	 static int WJ2NJYWU = 2;
	 static int WJ1NAXWU = 2;
	 static int WJ2NAXWU = 2;
	 static int WJ1NAXWD = 2;
```

```
static int WJ2NAXWD = 2;
static int WAYSU = 2;
static int WAYSD = 2;
static int WHU = 2;
static int WPU = 2;
static int WMU = 2;
static int WOU = 2;
static int WTU = 2;
static int WHD = 2;
static int WPD = 2;
static int WMD = 2;
static int WOD = 2;
static int WTD = 2;
static int WJNAYSU = 2;
static int WJNAYSD = 2;
static int WDU = 2;
static int WLU = 2;
static int WRU = 2;
static int WUD = 2;
static int WDD = 2;
static int WLD = 2;
static int WRD = 2;
static double Rand2swp = double();
static double Rand2swyp = double();
static double Rand2swm = double();
static double Rand2swym = double();
static double irx = double();
static double iry = double();
static double irx2e = double();
static double iry2e = double();
static double irx3e = double();
static double iry3e = double();
static int mousex = int();
static int mousey = int();
static double mousexm = double();
static double mouseym = double();
static double mousexr = double();
static double mouseyr = double();
static double mouseyp = double();
static double mousexp = double();
static int WAA = 2;
static int WBBU = 2;
static int WBBD = 2;
static int WUU = 2;
static int WZZU = 2;
static int WZCU = 2;
static int WCCU = 2;
static int WZZD = 2;
static int WZCD = 2;
static int WCCD = 2;
static int keys123 = int();
static int keys456 = int();
      static const int _INPUT_MOUSE = 0;
      static const int _MOUSEEVENTF_MOVE = 0x0001;
      static const int _MOUSEEVENTF_ABSOLUTE = 0x8000;
      static const int _MOUSEEVENTF_LEFTDOWN = 0x0002;
      static const int _MOUSEEVENTF_LEFTUP = 0x0004;
      static const int _MOUSEEVENTF_RIGHTDOWN = 0x0008;
      static const int _MOUSEEVENTF_RIGHTUP = 0x0010;
static const int _KEYEVENTF_EXTENDEDKEY = 0x0001;
      static const int _KEYEVENTF_KEYUP = 0x0002;
      static int Mousedirectinputx = int();
static int Mousedirectinputy = int();
static int varxout = int();
static int varyout = int();
      static int varxin = int();
```

```
	static int varyin = int();
	static int varpx = int();
	static int varpy = int();
	static int connecting = int();
	static double randirx1 = double();
	static double randiry1 = double();
		static int WF1U = 2;
	static int WF1D = 2;
		static bool F1B = false;
		static int WF2U = 2;
	static int WF2D = 2;
		static bool F2B = false;
		static int WF3U = 2;
	static int WF3D = 2;
		static bool F3B = false;
		static int WF4U = 2;
	static int WF4D = 2;
		static bool F4B = false;
			int signx = int();
			int signy = int();
CXBOXController* Player1;
int main(int argc, char* argv[])
{
	printf("*Controller by Mic Frametaux*");
	printf("\nSTART+UP = MW3");
	printf("\nSTART+RIGHT = BRINK");
	printf("\nSTART+DOWN = METRO");
	printf("\nSTART+LEFT = TITANFALL");
	printf("\nLEFT STICK = WASD");
	printf("\nRIGHT TRIGGER = E");
	printf("\nLEFT TRIGGER = Q");
	printf("\nA BUTTON = F(R)");
	printf("\nB BUTTON = V");
	printf("\nX BUTTON = SPACE");
	printf("\nY BUTTON = SHIFT");
	printf("\nRIGHT SHOULDER = LEFT CLICK");
	printf("\nLEFT SHOULDER = RIGHT CLICK");
	printf("\nBACK = CTRL");
	printf("\nSTART = G");
	printf("\nDOWN = C");
	printf("\nUP = X");
	printf("\nLEFT = 1,2,3");
	printf("\nRIGHT = 4,5,6");
	printf("\nSTART+SELECT = CLOSE");
	Player1 = new CXBOXController(1);
	while(true)
	{
		Player1 = new CXBOXController(1);
			if(Player1->IsConnected())
		{
			if (Player1->GetState().Gamepad.sThumbRX<0) {signx=-1;}
			if (Player1->GetState().Gamepad.sThumbRX>=0) {signx=1;}
			if (Player1->GetState().Gamepad.sThumbRY<0) {signy=-1;}
			if (Player1->GetState().Gamepad.sThumbRY>=0) {signy=1;}
	    mousex = - (Player1->GetState().Gamepad.sThumbRX*Player1-
>GetState().Gamepad.sThumbRX)*signx/2000000;
	    mousey = - (Player1->GetState().Gamepad.sThumbRY*Player1-
>GetState().Gamepad.sThumbRY)*signy/4000000;
                                        INPUT input[1];
                                        if (F1B == true)
			{
                                                ::ZeroMemory(input, sizeof(input));

                                                input[0].type = _INPUT_MOUSE;
```

```
                                        input[0].mi.dx = 65555 / 2 - mousex *
65555 / 400 / 2;
                                        input[0].mi.dy = mousey * 65555 / 300 / 2
+ 65555 / 2;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                        SendInput(1, input, sizeof(INPUT));
                    }
                    if (F3B == true)  //Metro
                    {
                        mousexp = mousexp + mousex / 12;
                        mouseyp = mouseyp + mousey / 12;
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -(int)mousexp;
                                        input[0].mi.dy = (int)mouseyp;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                        SendInput(1, input, sizeof(INPUT));
                    }
                    if (F2B == true)  //Brink slow
                    {
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -mousex / 8;
                                        input[0].mi.dy =  mousey / 8;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE;
                                        SendInput(1, input, sizeof(INPUT));
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -mousex / 40;
                                        input[0].mi.dy = mousey / 40;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                        SendInput(1, input, sizeof(INPUT));
                    }
                    if (F4B == true) //Titanfall
                    {
                        if (Player1->GetState().Gamepad.wButtons &
XINPUT_GAMEPAD_LEFT_SHOULDER)
                        {
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -mousex / 8;
                                        input[0].mi.dy = mousey / 8;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                        SendInput(1, input, sizeof(INPUT));
                        }
                        else
                        {
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -mousex / 8;
                                        input[0].mi.dy = mousey / 8;
```

```
                                                input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                                SendInput(1, input, sizeof(INPUT));
                            }
                            if (Player1->GetState().Gamepad.wButtons &
XINPUT_GAMEPAD_LEFT_SHOULDER)
                            {
                                if (WAA <= 3)
                                    WAA = WAA + 1;
                                WA = 0;
                            }
                            else
                            {
                                if (WA <= 3)
                                    WA = WA + 1;
                                WAA = 0;
                            }
                            if (WAA == 1)
                            {
                                mouse_event(0x0008, 0, 0, 0, 0);;
                            }
                            if (WA == 1)
                            {
                                mouse_event(0x0010, 0, 0, 0, 0);;
                            }
                        }
                       if ((Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_UP) &&
(Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_START))
            {
               if (WF1D <= 3)
                    WF1D = WF1D + 1;
                WF1U = 0;
            }
            else
            {
                if (WF1U <= 3)
                    WF1U = WF1U + 1;
                WF1D = 0;
            }
            if (WF1D == 1)
            {
                            if (F1B == false)
                                    F1B=true;
                            else
                                    F1B=false;
                       }
                       if ((Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_RIGHT)
&& (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_START))
            {
                if (WF2D <= 3)
                    WF2D = WF2D + 1;
                WF2U = 0;
            }
            else
            {
                if (WF2U <= 3)
                    WF2U = WF2U + 1;
                WF2D = 0;
            }
            if (WF2D == 1)
            {
                            if (F2B == false)
                                    F2B=true;
                            else
                                    F2B=false;
                       }
```

```
                    if ((Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_DOWN)
&& (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_START))
            {
                if (WF3D <= 3)
                    WF3D = WF3D + 1;
                WF3U = 0;
            }
            else
            {
                if (WF3U <= 3)
                    WF3U = WF3U + 1;
                WF3D = 0;
            }
            if (WF3D == 1)
            {
                            if (F3B == false)
                                F3B=true;
                            else
                                F3B=false;
                    }
                    if ((Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_LEFT)
&& (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_START))
            {
                if (WF4D <= 3)
                    WF4D = WF4D + 1;
                WF4U = 0;
            }
            else
            {
                if (WF4U <= 3)
                    WF4U = WF4U + 1;
                WF4D = 0;
            }
            if (WF4D == 1)
            {
                            if (F4B == false)
                                F4B=true;
                            else
                                F4B=false;
                    }
   ////////////////////////////////////////
            if (Player1->GetState().Gamepad.sThumbLX * 2 >= 50*7000 / 100)
            {
                if (WJ1NJXWD <= 3)
                    WJ1NJXWD = WJ1NJXWD + 1;
                WJ1NJXWU = 0;
            }
            else
            {
                if (WJ1NJXWU <= 3)
                    WJ1NJXWU = WJ1NJXWU + 1;
                WJ1NJXWD = 0;
            }
            if (WJ1NJXWD == 1)
            {
                keybd_event(0x44, 0x20, 0, 0);
            }
            if (WJ1NJXWU == 1)
            {
                keybd_event(0x44, 0x20, 0x0002, 0);
            }
            if (Player1->GetState().Gamepad.sThumbLX * 2 <= -50*7000 / 100)
            {
                if (WJ2NJXWD <= 3)
                    WJ2NJXWD = WJ2NJXWD + 1;
                WJ2NJXWU = 0;
```

```
}
else
{
    if (WJ2NJXWU <= 3)
        WJ2NJXWU = WJ2NJXWU + 1;
    WJ2NJXWD = 0;
}
if (WJ2NJXWD == 1)
{
    keybd_event(0x41, 0x1E, 0, 0);
}
if (WJ2NJXWU == 1)
{
    keybd_event(0x41, 0x1E, 0x0002, 0);
}
if (Player1->GetState().Gamepad.sThumbLY * 2 >= 50*7000 / 100)
{
    if (WJ1NJYWD <= 3)
        WJ1NJYWD = WJ1NJYWD + 1;
    WJ1NJYWU = 0;
}
else
{
    if (WJ1NJYWU <= 3)
        WJ1NJYWU = WJ1NJYWU + 1;
    WJ1NJYWD = 0;
}
if (WJ1NJYWD == 1)
{
    keybd_event(0x57, 0x11, 0, 0);
}
if (WJ1NJYWU == 1)
{
    keybd_event(0x57, 0x11, 0x0002, 0);
}
if (Player1->GetState().Gamepad.sThumbLY * 2 <= -50*7000 / 100)
{
    if (WJ2NJYWD <= 3)
        WJ2NJYWD = WJ2NJYWD + 1;
    WJ2NJYWU = 0;
}
else
{
    if (WJ2NJYWU <= 3)
        WJ2NJYWU = WJ2NJYWU + 1;
    WJ2NJYWD = 0;
}
if (WJ2NJYWD == 1)
{
    keybd_event(0x53, 0x1F, 0, 0);
}
if (WJ2NJYWU == 1)
{
    keybd_event(0x53, 0x1F, 0x0002, 0);
}
      if (Player1->GetState().Gamepad.bRightTrigger >= 0.2) //E
{
    if (WJ1NAXWD <= 3)
        WJ1NAXWD = WJ1NAXWD + 1;
    WJ1NAXWU = 0;
}
else
{
    if (WJ1NAXWU <= 3)
        WJ1NAXWU = WJ1NAXWU + 1;
    WJ1NAXWD = 0;
```

```
            }
            if (WJ1NAXWD == 1)
            {
                keybd_event(0x45, 0x12, 0, 0);
            }
            if (WJ1NAXWU == 1)
            {
                keybd_event(0x45, 0x12, 0x0002, 0);
            }
            if (Player1->GetState().Gamepad.bLeftTrigger >= 0.2)//Q
            {
                if (WJ2NAXWD <= 3)
                    WJ2NAXWD = WJ2NAXWD + 1;
                WJ2NAXWU = 0;
            }
            else
            {
                if (WJ2NAXWU <= 3)
                    WJ2NAXWU = WJ2NAXWU + 1;
                WJ2NAXWD = 0;
            }
            if (WJ2NAXWD == 1)
            {
                keybd_event(0x51, 0x10, 0, 0);
            }
            if (WJ2NAXWU == 1)
            {
                keybd_event(0x51, 0x10, 0x0002, 0);
            }
            if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_B)//V
            {
                if (WJNAYSD <= 3)
                    WJNAYSD = WJNAYSD + 1;
                WJNAYSU = 0;
            }
            else
            {
                if (WJNAYSU <= 3)
                    WJNAYSU = WJNAYSU + 1;
                WJNAYSD = 0;
            }
            if (WJNAYSD == 1)
            {
                keybd_event(0x56, 0x2F, 0, 0);
            }
            if (WJNAYSU == 1)
            {
                keybd_event(0x56, 0x2F, 0x0002, 0);
            }
                    if (Player1->GetState().Gamepad.wButtons &
XINPUT_GAMEPAD_RIGHT_SHOULDER)
            {
                if (WBBD <= 3)
                    WBBD = WBBD + 1;
                WBBU = 0;
            }
            else
            {
                if (WBBU <= 3)
                    WBBU = WBBU + 1;
                WBBD = 0;
            }
            if (WBBD == 1)
            {
                mouse_event(0x0002, 0, 0, 0, 0);
            }
```

```
if (WBBU == 1)
{
    mouse_event(0x0004, 0, 0, 0, 0);
}
          if (F4B == false) //Titanfall
{
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_LEFT_SHOULDER)
{
    if (WAA <= 3)
        WAA = WAA + 1;
    WA = 0;
}
else
{
    if (WA <= 3)
        WA = WA + 1;
    WAA = 0;
}
if (WAA == 1)
{
    mouse_event(0x0008, 0, 0, 0, 0);
}
if (WA == 1)
{
    mouse_event(0x0010, 0, 0, 0, 0);
}
          }
          if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_BACK) //CTRL
{
    if (WZCD <= 3)
        WZCD = WZCD + 1;
    WZCU = 0;
}
else
{
    if (WZCU <= 3)
        WZCU = WZCU + 1;
    WZCD = 0;
}
if (WZCD == 1)
{
    keybd_event(0x11, 0x1D, 0, 0);
}
if (WZCU == 1)
{
    keybd_event(0x11, 0x1D, 0x0002, 0);
}
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_X) //SPACE
{
    if (WCCD <= 3)
        WCCD = WCCD + 1;
    WCCU = 0;
}
else
{
    if (WCCU <= 3)
        WCCU = WCCU + 1;
    WCCD = 0;
}
if (WCCD == 1)
{
    keybd_event(0x20, 0x39, 0, 0);
}
if (WCCU == 1)
{
    keybd_event(0x20, 0x39, 0x0002, 0);
```

```
}
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_Y) //SHIFT
{
    if (WZZD <= 3)
        WZZD = WZZD + 1;
    WZZU = 0;
}
else
{
    if (WZZU <= 3)
        WZZU = WZZU + 1;
    WZZD = 0;
}
if (WZZD == 1)
{
    keybd_event(0x10, 0x2A, 0, 0);
}
if (WZZU == 1)
{
    keybd_event(0x10, 0x2A, 0x0002, 0);
}
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_DOWN)
{
    if (WDD <= 3)
        WDD = WDD + 1;
    WDU = 0;
}
else
{
    if (WDU <= 3)
        WDU = WDU + 1;
    WDD = 0;
}
if (WDD == 1)
{
    keybd_event(0x43, 0x2E, 0, 0);
}
if (WDU == 1)
{
    keybd_event(0x43, 0x2E, 0x0002, 0);
}
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_A)//F & R
{
    if (WHD <= 3)
        WHD = WHD + 1;
    WHU = 0;
}
else
{
    if (WHU <= 3)
        WHU = WHU + 1;
    WHD = 0;
}
if (WHD == 1)
{
    keybd_event(0x46, 0x21, 0, 0);
    keybd_event(0x52, 0x13, 0, 0);
}
if (WHU == 1)
{
    keybd_event(0x46, 0x21, 0x0002, 0);
    keybd_event(0x52, 0x13, 0x0002, 0);
}
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_START) //G
{
    if (WPD <= 3)
```

```
        WPD = WPD + 1;
    WPU = 0;
}
else
{
    if (WPU <= 3)
        WPU = WPU + 1;
    WPD = 0;
}
if (WPD == 1)
{
    keybd_event(0x47, 0x22, 0, 0);
}
if (WPU == 1)
{
    keybd_event(0x47, 0x22, 0x0002, 0);
}
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_LEFT)
{
    if (WLD <= 3)
        WLD = WLD + 1;
    WLU = 0;
}
else
{
    if (WLU <= 3)
        WLU = WLU + 1;
    WLD = 0;
}
if (WLD == 1)
{
    if (keys123 == 0)
    {
        keybd_event(0x31, 0x02, 0, 0);
    }
    if (keys123 == 1)
    {
        keybd_event(0x32, 0x03, 0, 0);
    }
    if (keys123 == 2)
    {
        keybd_event(0x33, 0x04, 0, 0);
    }
}
if (WLU == 1)
{
    if (keys123 == 0)
    {
        keybd_event(0x31, 0x02, 0x0002, 0);
        keys123 = 1;
    }
    else
    {
        if (keys123 == 1)
        {
            keybd_event(0x32, 0x03, 0x0002, 0);
            keys123 = 2;
        }
        else
            if (keys123 == 2)
            {
                keybd_event(0x33, 0x04, 0x0002, 0);
                keys123 = 0;
            }
    }
}
```

```
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_RIGHT)
{
    if (WRD <= 3)
        WRD = WRD + 1;
    WRU = 0;
}
else
{
    if (WRU <= 3)
        WRU = WRU + 1;
    WRD = 0;
}
if (WRD == 1)
{
    if (keys456 == 0)
    {
        keybd_event(0x34, 0x05, 0, 0);
    }
    if (keys456 == 1)
    {
        keybd_event(0x35, 0x06, 0, 0);
    }
    if (keys456 == 2)
    {
        keybd_event(0x36, 0x07, 0, 0);
    }
}
if (WRU == 1)
{
    if (keys456 == 0)
    {
        keybd_event(0x34, 0x05, 0x0002, 0);
        keys456 = 1;
    }
    else
    {
        if (keys456 == 1)
        {
            keybd_event(0x35, 0x06, 0x0002, 0);
            keys456 = 2;
        }
        else
            if (keys456 == 2)
            {
                keybd_event(0x36, 0x07, 0x0002, 0);
                keys456 = 0;
            }
    }
}
if (Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_DPAD_UP)
{
    if (WUD <= 3)
        WUD = WUD + 1;
    WUU = 0;
}
else
{
    if (WUU <= 3)
        WUU = WUU + 1;
    WUD = 0;
}
if (WUD == 1)
{
    keybd_event(0x58, 0x2D, 0, 0);
}
if (WUU == 1)
```

```
            {
                keybd_event(0x58, 0x2D, 0x0002, 0);
            }
                    if ((Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_START) &&
(Player1->GetState().Gamepad.wButtons & XINPUT_GAMEPAD_BACK))
                            break;
              Sleep(1);
              }
              else
              {
                    std::cout << "\n\tERROR! PLAYER 1 - XBOX 360 Controller Not Found!\n";
              Sleep(1);
              }
}
}
```

4. Wiimote Controller Program in C++ using C# Wiimote Library (C++ Form)

```
#pragma once
#include <iostream>
#include <process.h>
#include <windows.h>
#include "stdafx.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <string>
using namespace std;
namespace WiimoteByMic {
       using namespace System;
       using namespace System::ComponentModel;
       using namespace System::Collections;
       using namespace System::Windows::Forms;
       using namespace System::Data;
       using namespace System::Drawing;
       using namespace WiimoteLib;
       using namespace System::Runtime::InteropServices;
       using namespace System::Threading;
 public ref class Form1 : public System::Windows::Forms::Form
 {
        static int WidthS = 400;
        static int HeightS = 300;
              static int Randir;
        static int WA = 2;
        static int WJ1NJXWD = 2;
        static int WJ2NJXWD = 2;
        static int WJ1NJYWD = 2;
        static int WJ2NJYWD = 2;
        static int WJ1NJXWU = 2;
        static int WJ2NJXWU = 2;
        static int WJ1NJYWU = 2;
        static int WJ2NJYWU = 2;
        static int WJ1NAXWU = 2;
        static int WJ2NAXWU = 2;
        static int WJ1NAXWD = 2;
        static int WJ2NAXWD = 2;
        static int WAYSU = 2;
        static int WAYSD = 2;
        static int WHU = 2;
        static int WPU = 2;
        static int WMU = 2;
        static int WOU = 2;
        static int WTU = 2;
        static int WHD = 2;
        static int WPD = 2;
        static int WMD = 2;
        static int WOD = 2;
```

```
static int WTD = 2;
static int WJNAYSU = 2;
static int WJNAYSD = 2;
static int WDU = 2;
static int WLU = 2;
static int WRU = 2;
static int WUD = 2;
static int WDD = 2;
static int WLD = 2;
static int WRD = 2;
static double Rand2swp = double();
static double Rand2swyp = double();
static double Rand2swm = double();
static double Rand2swym = double();
static double irx = double();
static double iry = double();
static double irx2e = double();
static double iry2e = double();
static double irx3e = double();
static double iry3e = double();
static int mousex = int();
static int mousey = int();
static double mousexm = double();
static double mouseym = double();
static double mousexr = double();
static double mouseyr = double();
static double mouseyp = double();
static double mousexp = double();
static int WAA = 2;
static int WBBU = 2;
static int WBBD = 2;
static int WUU = 2;
static int WZZU = 2;
static int WZCU = 2;
static int WCCU = 2;
static int WZZD = 2;
static int WZCD = 2;
static int WCCD = 2;
static int keys123 = int();
static int keys456 = int();
      static const int _INPUT_MOUSE = 0;
      static const int _MOUSEEVENTF_MOVE = 0x0001;
      static const int _MOUSEEVENTF_ABSOLUTE = 0x8000;
      static const int _MOUSEEVENTF_LEFTDOWN = 0x0002;
      static const int _MOUSEEVENTF_LEFTUP = 0x0004;
      static const int _MOUSEEVENTF_RIGHTDOWN = 0x0008;
      static const int _MOUSEEVENTF_RIGHTUP = 0x0010;
static const int _KEYEVENTF_EXTENDEDKEY = 0x0001;
      static const int _KEYEVENTF_KEYUP = 0x0002;
      static int Mousedirectinputx = int();
static int Mousedirectinputy = int();
static int varxout = int();
static int varyout = int();
      static int varxin = int();
static int varyin = int();
static int varpx = int();
static int varpy = int();
static int connecting = int();
static int connecting1 = int();
static int connecting2 = int();
static double randirx1 = double();
static double randiry1 = double();
      static int WF1U = 2;
static int WF1D = 2;
      static bool F1B = false;
      static int WF2U = 2;
```

```
        static int WF2D = 2;
               static bool F2B = false;
               static int WF3U = 2;
        static int WF3D = 2;
               static bool F3B = false;
               static int WF4U = 2;
        static int WF4D = 2;
               static bool F4B = false;
               static int WF5U = 2;
        static int WF5D = 2;
               static bool F5B = false;
               #define GAMENAME6 "Borderlands 2 (32-bit, DX9)"
 public: WiimoteLib::Wiimote^ wm;
 public: WiimoteLib::Wiimote::WiimoteCollection^ mWC;
public:Thread^ oThread;
 public:Thread^ nThread;
 public:
 Form1(void)
 {
   InitializeComponent();
   mWC = gcnew WiimoteLib::Wiimote::WiimoteCollection();
   mWC->FindAllWiimotes();
   while (connecting < 13)
   for each (wm in mWC)
   {
          if (connecting < 13)
       {
          wm->SetReportType();
       connecting = connecting + 1;
       }
   }
   oThread = gcnew Thread(gcnew ThreadStart(this,&Form1::t1));
   oThread->Start();
   nThread = gcnew Thread(gcnew ThreadStart(this,&Form1::t2));
   nThread->Start();
 }
 void t1()
 {
   mWC = gcnew WiimoteLib::Wiimote::WiimoteCollection();
   mWC->FindAllWiimotes();
   for each (wm in mWC)
   {
                 connecting1 = 0;
          while (connecting1 < 1)
          {
          if (connecting1 < 1)
       {
          wm->SetReportType();
       connecting1 = connecting1 + 1;
          }
          wm->WiimoteChanged +=
       gcnew System::EventHandler<WiimoteChangedEventArgs^>(
   this, &Form1::wm_WiimoteChanged1);
          Sleep(10);
          }
   }
 }
 void t2()
 {
   mWC = gcnew WiimoteLib::Wiimote::WiimoteCollection();
   mWC->FindAllWiimotes();
   for each (wm in mWC)
   {
          connecting2 = 0;
          while (connecting2 < 1)
          {
```

```
         if (connecting2 < 1)
       {
          wm->SetReportType();
       connecting2 = connecting2 + 1;
          }
          wm->WiimoteChanged +=
       gcnew System::EventHandler<WiimoteChangedEventArgs^>(
   this, &Form1::wm_WiimoteChanged2);
          Sleep(10);
          }
   }
 }
public: void  wm_WiimoteChanged1(Object^ sender,WiimoteLib::WiimoteChangedEventArgs^ args){
   WiimoteState^ ws;
   ws = args->WiimoteState;
                     if (ws->IRSensors1.X >= 1 & ws->IRSensors1.X <= 1022)
            {
                irx3e = ws->IRSensors1.X - 1024 / 2 + randirx1 / 2;
            }
            if (ws->IRSensors0.X >= 1 & ws->IRSensors0.X <= 1022)
            {
                irx2e = ws->IRSensors0.X - 1024 / 2 - randirx1 / 2;
            }
            if (ws->IRSensors1.Y >= 1 & ws->IRSensors1.Y <= 766)
            {
                iry3e = ws->IRSensors1.Y - 768 / 2 + randiry1 / 2;
            }
            if (ws->IRSensors0.Y >= 1 & ws->IRSensors0.Y <= 766)
            {
                iry2e = ws->IRSensors0.Y - 768 / 2 - randiry1 / 2;
            }
                     irx = (irx2e + irx3e) / 20;
           mousexm = Math::Pow(irx, 3) / 1000;
           mousexr = Math::Abs(irx) + Math::Abs(Math::Pow(mousexm, 3)) / 1000;
           mousex = (int)(mousexr) * Math::Sign(irx);
           iry = (iry2e + iry3e) / 20;
           mouseym = Math::Pow(iry, 2) / 15;
           mouseyr = Math::Abs(iry) + Math::Abs(Math::Pow(mouseym, 2)) / 15;
           mousey = (int)(mouseyr) * Math::Sign(iry);
                     Randir = Randir + 1;
                     if (Randir >= 3)
                    {
                                        INPUT input[1];
                        Randir = 0;
                        if (F3B == true)  //Metro
                        {
                            mousexp = mousexp + mousex / 12 + irx / 6;
                            mouseyp = mouseyp + mousey / 12 + iry / 18;
                                                ::ZeroMemory(input, sizeof(input));

                                                input[0].type = _INPUT_MOUSE;

                                                input[0].mi.dx = -(int)mousexp;
                                                input[0].mi.dy = (int)mouseyp;
                                                input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                                SendInput(1, input, sizeof(INPUT));
                        }
                        if (F2B == true)  //Brink slow
                        {
                                                ::ZeroMemory(input, sizeof(input));

                                                input[0].type = _INPUT_MOUSE;

                                                input[0].mi.dx = -mousex / 24 - (int)irx
/ 2;
```

```
                                        input[0].mi.dy =  mousey / 24 + (int)iry
/ 6;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE;
                                        SendInput(1, input, sizeof(INPUT));
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -mousex / 40 - (int)irx
/ 12;
                                        input[0].mi.dy = mousey / 40 + (int)iry /
12;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                        SendInput(1, input, sizeof(INPUT));
                    }
                    if (F4B == true) //Titanfall
                    {
                        if (ws->ButtonState.A)
                        {
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -mousex / 8;
                                        input[0].mi.dy = mousey / 8;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                        SendInput(1, input, sizeof(INPUT));
                        }
                        else
                        {
                                        ::ZeroMemory(input, sizeof(input));

                                        input[0].type = _INPUT_MOUSE;

                                        input[0].mi.dx = -mousex / 8;
                                        input[0].mi.dy = mousey / 8;
                                        input[0].mi.dwFlags = MOUSEEVENTF_MOVE |
MOUSEEVENTF_ABSOLUTE;
                                        SendInput(1, input, sizeof(INPUT));
                        }
                        if (ws->ButtonState.A)
                        {
                            if (WAA <= 3)
                                WAA = WAA + 1;
                            WA = 0;
                        }
                        else
                        {
                            if (WA <= 3)
                                WA = WA + 1;
                            WAA = 0;
                        }
                        if (WAA == 1)
                        {
                            mouse_event(0x0008, 0, 0, 0, 0);;
                        }
                        if (WA == 1)
                        {
                            mouse_event(0x0010, 0, 0, 0, 0);;
                        }
                    }
                }
                 if (F1B == true)
                 {
```

```
                    POINT coords;
                    coords.x = WidthS - (mousex * WidthS) / 512;
            coords.y = (mousey * HeightS) / 384 + HeightS;
                    //HWND hWnd = ::GetForegroundWindow();
                    //ClientToScreen(hWnd, &coords);
                    SetCursorPos(coords.x,coords.y);
                    }
                    if (F5B == true)
                    {
                    POINT coords;
                    coords.x = WidthS - (mousex * WidthS) / 512;
            coords.y = (mousey * HeightS) / 384 + HeightS;
                    SetCursorPos(coords.x,coords.y);
                    PostMessage(FindWindow(NULL, L"Borderlands 2 (32-bit, DX9)"),
WM_MOUSEMOVE, 0, MAKELPARAM(100000000, 1000000000));
                    }
              }
              public: void  wm_WiimoteChanged2(Object^
sender,WiimoteLib::WiimoteChangedEventArgs^ args){
   WiimoteState^ ws;
   ws = args->WiimoteState;
   ////////////////////////////////////
                    if (ws->ButtonState.Up & ws->ButtonState.One)
            {
               if (WF1D <= 3)
                    WF1D = WF1D + 1;
                WF1U = 0;
            }
            else
            {
                if (WF1U <= 3)
                    WF1U = WF1U + 1;
                WF1D = 0;
            }
            if (WF1D == 1)
            {
                            if (F1B == false)
                                    F1B=true;
                            else
                                    F1B=false;
                    }
                    if (ws->ButtonState.Right & ws->ButtonState.One)
            {
                if (WF2D <= 3)
                    WF2D = WF2D + 1;
                WF2U = 0;
            }
            else
            {
                if (WF2U <= 3)
                    WF2U = WF2U + 1;
                WF2D = 0;
            }
            if (WF2D == 1)
            {
                            if (F2B == false)
                                    F2B=true;
                            else
                                    F2B=false;
                    }
                    if (ws->ButtonState.Down & ws->ButtonState.One)
            {
                if (WF3D <= 3)
                    WF3D = WF3D + 1;
                WF3U = 0;
            }
```

```
            else
            {
                if (WF3U <= 3)
                    WF3U = WF3U + 1;
                WF3D = 0;
            }
            if (WF3D == 1)
            {
                            if (F3B == false)
                                   F3B=true;
                            else
                                   F3B=false;
                     }
                     if (ws->ButtonState.Left & ws->ButtonState.One)
            {
                if (WF4D <= 3)
                    WF4D = WF4D + 1;
                WF4U = 0;
            }
            else
            {
                if (WF4U <= 3)
                    WF4U = WF4U + 1;
                WF4D = 0;
            }
            if (WF4D == 1)
            {
                            if (F4B == false)
                                   F4B=true;
                            else
                                   F4B=false;
                     }
                     if (ws->ButtonState.Home & ws->ButtonState.One)
            {
               if (WF5D <= 3)
                    WF5D = WF5D + 1;
                WF5U = 0;
            }
            else
            {
                if (WF5U <= 3)
                    WF5U = WF5U + 1;
                WF5D = 0;
            }
            if (WF5D == 1)
            {
                            if (F5B == false)
                                   F5B=true;
                            else
                                   F5B=false;
                     }
//////////////////////////////////////////
Rand2swp = Rand2swp + 7;
            if (ws->NunchukState.RawJoystick.X - 255 / 2 > Rand2swp)
            {
                if (WJ1NJXWD <= 3)
                    WJ1NJXWD = WJ1NJXWD + 1;
                WJ1NJXWU = 0;
            }
            else
            {
                if (WJ1NJXWU <= 3)
                    WJ1NJXWU = WJ1NJXWU + 1;
                WJ1NJXWD = 0;
            }
            if (WJ1NJXWD == 1)
```

```
{
    keybd_event(0x44, 0x20, 0, 0);
}
if (WJ1NJXWU == 1)
{
    keybd_event(0x44, 0x20, 0x0002, 0);
}
if (Rand2swp >= 60)
{
    Rand2swp = 10;
}
Rand2swm = Rand2swm - 7;
if (ws->NunchukState.RawJoystick.X - 255 / 2 < Rand2swm)
{
    if (WJ2NJXWD <= 3)
        WJ2NJXWD = WJ2NJXWD + 1;
    WJ2NJXWU = 0;
}
else
{
    if (WJ2NJXWU <= 3)
        WJ2NJXWU = WJ2NJXWU + 1;
    WJ2NJXWD = 0;
}
if (WJ2NJXWD == 1)
{
    keybd_event(0x41, 0x1E, 0, 0);
}
if (WJ2NJXWU == 1)
{
    keybd_event(0x41, 0x1E, 0x0002, 0);
}
if (Rand2swm <= -60)
{
    Rand2swm = -10;
}
Rand2swyp = Rand2swyp + 7;
if (ws->NunchukState.RawJoystick.Y - 255 / 2 > Rand2swyp)
{
    if (WJ1NJYWD <= 3)
        WJ1NJYWD = WJ1NJYWD + 1;
    WJ1NJYWU = 0;
}
else
{
    if (WJ1NJYWU <= 3)
        WJ1NJYWU = WJ1NJYWU + 1;
    WJ1NJYWD = 0;
}
if (WJ1NJYWD == 1)
{
    keybd_event(0x57, 0x11, 0, 0);
}
if (WJ1NJYWU == 1)
{
    keybd_event(0x57, 0x11, 0x0002, 0);
}
if (Rand2swyp >= 60)
{
    Rand2swyp = 10;
}
Rand2swym = Rand2swym - 7;
if (ws->NunchukState.RawJoystick.Y - 255 / 2 < Rand2swym)
{
    if (WJ2NJYWD <= 3)
        WJ2NJYWD = WJ2NJYWD + 1;
```

```
    WJ2NJYWU = 0;
}
else
{
    if (WJ2NJYWU <= 3)
        WJ2NJYWU = WJ2NJYWU + 1;
    WJ2NJYWD = 0;
}
if (WJ2NJYWD == 1)
{
    keybd_event(0x53, 0x1F, 0, 0);
}
if (WJ2NJYWU == 1)
{
    keybd_event(0x53, 0x1F, 0x0002, 0);
}
if (Rand2swym <= -60)
{
    Rand2swym = -10;
}
          if ((float)(ws->AccelState.RawValues.X - 270 / 2) / 25 > 0.8)
{
    if (WJ1NAXWD <= 3)
        WJ1NAXWD = WJ1NAXWD + 1;
    WJ1NAXWU = 0;
}
else
{
    if (WJ1NAXWU <= 3)
        WJ1NAXWU = WJ1NAXWU + 1;
    WJ1NAXWD = 0;
}
if (WJ1NAXWD == 1)
{
    keybd_event(0x45, 0x12, 0, 0);
}
if (WJ1NAXWU == 1)
{
    keybd_event(0x45, 0x12, 0x0002, 0);
}
if ((float)(ws->AccelState.RawValues.X - 270 / 2) / 25 < -0.8)
{
    if (WJ2NAXWD <= 3)
        WJ2NAXWD = WJ2NAXWD + 1;
    WJ2NAXWU = 0;
}
else
{
    if (WJ2NAXWU <= 3)
        WJ2NAXWU = WJ2NAXWU + 1;
    WJ2NAXWD = 0;
}
if (WJ2NAXWD == 1)
{
    keybd_event(0x51, 0x10, 0, 0);
}
if (WJ2NAXWU == 1)
{
    keybd_event(0x51, 0x10, 0x0002, 0);
}
if ((float)(ws->AccelState.RawValues.Y - 270 / 2) / 25 > 0.98)
{
    if (WAYSD <= 3)
        WAYSD = WAYSD + 1;
    WAYSU = 0;
}
```

```
else
{
    if (WAYSU <= 3)
        WAYSU = WAYSU + 1;
    WAYSD = 0;
}
if (WAYSD == 1)
{
    keybd_event(0x52, 0x13, 0, 0);
}
if (WAYSU == 1)
{
    keybd_event(0x52, 0x13, 0x0002, 0);
}
if ((float)(ws->NunchukState.AccelState.RawValues.Y - 255 / 2) / 25 > 0.98)
{
    if (WJNAYSD <= 3)
        WJNAYSD = WJNAYSD + 1;
    WJNAYSU = 0;
}
else
{
    if (WJNAYSU <= 3)
        WJNAYSU = WJNAYSU + 1;
    WJNAYSD = 0;
}
if (WJNAYSD == 1)
{
    keybd_event(0x56, 0x2F, 0, 0);
}
if (WJNAYSU == 1)
{
    keybd_event(0x56, 0x2F, 0x0002, 0);
}
          if (ws->ButtonState.B)
{
    if (WBBD <= 3)
        WBBD = WBBD + 1;
    WBBU = 0;
}
else
{
    if (WBBU <= 3)
        WBBU = WBBU + 1;
    WBBD = 0;
}
if (WBBD == 1)
{
    mouse_event(0x0002, 0, 0, 0, 0);
}
if (WBBU == 1)
{
    mouse_event(0x0004, 0, 0, 0, 0);
}
          if (F4B == false) //Titanfall
{
if (ws->ButtonState.A)
{
    if (WAA <= 3)
        WAA = WAA + 1;
    WA = 0;
}
else
{
    if (WA <= 3)
        WA = WA + 1;
```

```
    WAA = 0;
}
if (WAA == 1)
{
    mouse_event(0x0008, 0, 0, 0, 0);
}
if (WA == 1)
{
    mouse_event(0x0010, 0, 0, 0, 0);
}
          }
          if (ws->NunchukState.Z & ws->NunchukState.C)
{
    if (WZCD <= 3)
        WZCD = WZCD + 1;
    WZCU = 0;
}
else
{
    if (WZCU <= 3)
        WZCU = WZCU + 1;
    WZCD = 0;
}
if (WZCD == 1)
{
    keybd_event(0x11, 0x1D, 0, 0);
}
if (WZCU == 1)
{
    keybd_event(0x11, 0x1D, 0x0002, 0);
}
if (!ws->NunchukState.Z & ws->NunchukState.C)
{
    if (WCCD <= 3)
        WCCD = WCCD + 1;
    WCCU = 0;
}
else
{
    if (WCCU <= 3)
        WCCU = WCCU + 1;
    WCCD = 0;
}
if (WCCD == 1)
{
    keybd_event(0x20, 0x39, 0, 0);
}
if (WCCU == 1)
{
    keybd_event(0x20, 0x39, 0x0002, 0);
}
if (ws->NunchukState.Z & !ws->NunchukState.C)
{
    if (WZZD <= 3)
        WZZD = WZZD + 1;
    WZZU = 0;
}
else
{
    if (WZZU <= 3)
        WZZU = WZZU + 1;
    WZZD = 0;
}
if (WZZD == 1)
{
    keybd_event(0x10, 0x2A, 0, 0);
```

```
}
if (WZZU == 1)
{
    keybd_event(0x10, 0x2A, 0x0002, 0);
}
if (ws->ButtonState.Down)
{
    if (WDD <= 3)
        WDD = WDD + 1;
    WDU = 0;
}
else
{
    if (WDU <= 3)
        WDU = WDU + 1;
    WDD = 0;
}
if (WDD == 1)
{
    keybd_event(0x43, 0x2E, 0, 0);
}
if (WDU == 1)
{
    keybd_event(0x43, 0x2E, 0x0002, 0);
}
if (ws->ButtonState.Home)
{
    if (WHD <= 3)
        WHD = WHD + 1;
    WHU = 0;
}
else
{
    if (WHU <= 3)
        WHU = WHU + 1;
    WHD = 0;
}
if (WHD == 1)
{
    keybd_event(0x46, 0x21, 0, 0);
    keybd_event(0x52, 0x13, 0, 0);
}
if (WHU == 1)
{
    keybd_event(0x46, 0x21, 0x0002, 0);
    keybd_event(0x52, 0x13, 0x0002, 0);
}
if (ws->ButtonState.Plus)
{
    if (WPD <= 3)
        WPD = WPD + 1;
    WPU = 0;
}
else
{
    if (WPU <= 3)
        WPU = WPU + 1;
    WPD = 0;
}
if (WPD == 1)
{
    keybd_event(0x47, 0x22, 0, 0);
}
if (WPU == 1)
{
    keybd_event(0x47, 0x22, 0x0002, 0);
```

```
}
if (ws->ButtonState.Minus)
{
    if (WMD <= 3)
        WMD = WMD + 1;
    WMU = 0;
}
else
{
    if (WMU <= 3)
        WMU = WMU + 1;
    WMD = 0;
}
if (WMD == 1)
{
    keybd_event(0x54, 0x14, 0, 0);
}
if (WMU == 1)
{
    keybd_event(0x54, 0x14, 0x0002, 0);
}
if (ws->ButtonState.Left)
{
    if (WLD <= 3)
        WLD = WLD + 1;
    WLU = 0;
}
else
{
    if (WLU <= 3)
        WLU = WLU + 1;
    WLD = 0;
}
if (WLD == 1)
{
    if (keys123 == 0)
    {
        keybd_event(0x31, 0x02, 0, 0);
    }
    if (keys123 == 1)
    {
        keybd_event(0x32, 0x03, 0, 0);
    }
    if (keys123 == 2)
    {
        keybd_event(0x33, 0x04, 0, 0);
    }
}
if (WLU == 1)
{
    if (keys123 == 0)
    {
        keybd_event(0x31, 0x02, 0x0002, 0);
        keys123 = 1;
    }
    else
    {
        if (keys123 == 1)
        {
            keybd_event(0x32, 0x03, 0x0002, 0);
            keys123 = 2;
        }
        else
            if (keys123 == 2)
            {
                keybd_event(0x33, 0x04, 0x0002, 0);
```

```
                keys123 = 0;
            }
    }
}
if (ws->ButtonState.Right)
{
    if (WRD <= 3)
        WRD = WRD + 1;
    WRU = 0;
}
else
{
    if (WRU <= 3)
        WRU = WRU + 1;
    WRD = 0;
}
if (WRD == 1)
{
    if (keys456 == 0)
    {
        keybd_event(0x34, 0x05, 0, 0);
    }
    if (keys456 == 1)
    {
        keybd_event(0x35, 0x06, 0, 0);
    }
    if (keys456 == 2)
    {
        keybd_event(0x36, 0x07, 0, 0);
    }
}
if (WRU == 1)
{
    if (keys456 == 0)
    {
        keybd_event(0x34, 0x05, 0x0002, 0);
        keys456 = 1;
    }
    else
    {
        if (keys456 == 1)
        {
            keybd_event(0x35, 0x06, 0x0002, 0);
            keys456 = 2;
        }
        else
            if (keys456 == 2)
            {
                keybd_event(0x36, 0x07, 0x0002, 0);
                keys456 = 0;
            }
    }
}
if (ws->ButtonState.Up)
{
    if (WUD <= 3)
        WUD = WUD + 1;
    WUU = 0;
}
else
{
    if (WUU <= 3)
        WUU = WUU + 1;
    WUD = 0;
}
if (WUD == 1)
```

```
            {
                keybd_event(0x58, 0x2D, 0, 0);
            }
            if (WUU == 1)
            {
                keybd_event(0x58, 0x2D, 0x0002, 0);
            }
            if (ws->ButtonState.One)
            {
                if (WOD <= 3)
                    WOD = WOD + 1;
                WOU = 0;
            }
            else
            {
                if (WOU <= 3)
                    WOU = WOU + 1;
                WOD = 0;
            }
            if (WOD == 1)
            {
                keybd_event(0x09, 0x0F, 0, 0);
                keybd_event(0x0D, 0x1C, 0, 0);
            }
            if (WOU == 1)
            {
                keybd_event(0x09, 0x0F, 0x0002, 0);
                keybd_event(0x0D, 0x1C, 0x0002, 0);
            }
            if (ws->ButtonState.Two)
            {
                if (WTD <= 3)
                    WTD = WTD + 1;
                WTU = 0;
            }
            else
            {
                if (WTU <= 3)
                    WTU = WTU + 1;
                WTD = 0;
            }
            if (WTD == 1)
            {
                keybd_event(0x1B, 0x01, 0, 0);
            }
            if (WTU == 1)
            {
                keybd_event(0x1B, 0x01, 0x0002, 0);
                     WidthS = Screen::PrimaryScreen->WorkingArea.Width / 2;
                     HeightS = Screen::PrimaryScreen->WorkingArea.Height / 2;
                randirx1 = ws->IRSensors0.X - ws->IRSensors1.X;
                randiry1 = ws->IRSensors0.Y - ws->IRSensors1.Y;
            }
          }
#pragma region Windows Form Designer generated code
#pragma endregion
};
}
```

5. C# Wiimote Library to Import in C++ (C# Library)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WiimoteLib
{
```

```
    public static class HID
    {
        [System.Runtime.InteropServices.DllImport("hid.dll")]
        public static extern void HidD_GetHidGuid(out Guid gHid);
        [System.Runtime.InteropServices.DllImport("hid.dll")]
        public static extern Boolean HidD_GetAttributes(IntPtr HidDeviceObject, ref
HIDD_ATTRIBUTES Attributes);
        [System.Runtime.InteropServices.DllImport("hid.dll")]
        public extern static bool HidD_SetOutputReport(IntPtr HidDeviceObject, byte[]
lpReportBuffer, uint ReportBufferLength);
        [System.Runtime.InteropServices.DllImport("setupapi.dll")]
        public static extern IntPtr SetupDiGetClassDevs(ref Guid ClassGuid, string
Enumerator, IntPtr hwndParent, UInt32 Flags);
        [System.Runtime.InteropServices.DllImport("setupapi.dll")]
        public static extern Boolean SetupDiEnumDeviceInterfaces(IntPtr hDevInfo, IntPtr
devInvo, ref Guid interfaceClassGuid, Int32 memberIndex, ref SP_DEVICE_INTERFACE_DATA
deviceInterfaceData);
        [System.Runtime.InteropServices.DllImport("setupapi.dll")]
        public static extern Boolean SetupDiGetDeviceInterfaceDetail(IntPtr hDevInfo, ref
SP_DEVICE_INTERFACE_DATA deviceInterfaceData, IntPtr deviceInterfaceDetailData, UInt32
deviceInterfaceDetailDataSize, out UInt32 requiredSize, IntPtr deviceInfoData);
        [System.Runtime.InteropServices.DllImport("setupapi.dll")]
        public static extern Boolean SetupDiGetDeviceInterfaceDetail(IntPtr hDevInfo, ref
SP_DEVICE_INTERFACE_DATA deviceInterfaceData, ref SP_DEVICE_INTERFACE_DETAIL_DATA
deviceInterfaceDetailData, UInt32 deviceInterfaceDetailDataSize, out UInt32 requiredSize,
IntPtr deviceInfoData);
        [System.Runtime.InteropServices.DllImport("Kernel32.dll")]
        public static extern Microsoft.Win32.SafeHandles.SafeFileHandle CreateFile(string
fileName, System.IO.FileAccess fileAccess, System.IO.FileShare fileShare, IntPtr
securityAttributes, System.IO.FileMode creationDisposition, EFileAttributes flags, IntPtr
template);
    }
    public enum EFileAttributes : uint
    {
        Overlapped = 0x40000000
    };
    public struct SP_DEVICE_INTERFACE_DATA
    {
        public int cbSize;
        public Guid InterfaceClassGuid;
        public int Flags;
        public IntPtr RESERVED;
    }
    public struct SP_DEVICE_INTERFACE_DETAIL_DATA
    {
        public UInt32 cbSize;

[System.Runtime.InteropServices.MarshalAs(System.Runtime.InteropServices.UnmanagedType.ByVa
lTStr, SizeConst = 256)]
        public string DevicePath;
    }
    public struct HIDD_ATTRIBUTES
    {
        public int Size;
        public short VendorID;
        public short ProductID;
        public short VersionNumber;
    }
    public class WiimoteChangedEventArgs : EventArgs
    {
        public WiimoteState WiimoteState;
        public WiimoteChangedEventArgs(WiimoteState ws)
        {
            WiimoteState = ws;
        }
    }
```

```
    public class Wiimote : IDisposable
    {
        public void OnReadData(IAsyncResult ar)
        {
            byte[] buff = (byte[])ar.AsyncState;
            mStream.EndRead(ar);
            InputReport type2 = (InputReport)buff[0];
            if (v <= 13)
            {
                v = v + 1;
                switch (type2)
                {
                    case InputReport.Status:
                        WriteData(REGISTER_EXTENSION_INIT_1, 1, new byte[] { 0x55 });
                        WriteData(REGISTER_EXTENSION_INIT_2, 1, new byte[] { 0x00 });
                        byte[] ibuff = ReadData(REGISTER_EXTENSION_TYPE, 6);
                        long itype = ((long)ibuff[0] << 40) | ((long)ibuff[1] << 32) |
((long)ibuff[2]) << 24 | ((long)ibuff[3]) << 16 | ((long)ibuff[4]) << 8 | ibuff[5];
                        mWiimoteState.ExtensionType = (ExtensionType)itype;
                        mBuff[0] = (byte)OutputReport.IR;
                        mBuff[1] = (byte)(0x04);
                        HID.HidD_SetOutputReport(this.mHandle.DangerousGetHandle(), mBuff,
(uint)mBuff.Length);
                        WriteData(REGISTER_IR, 1, new byte[] { 0x08 });
                        WriteData(REGISTER_IR_MODE, 1, new byte[] { 0x01 });
                        mBuff[0] = (byte)OutputReport.Type;
                        mBuff[1] = (byte)(true ? 0x04 : 0x00);
                        mStatusDone.Set();
                        break;
                    case InputReport.ReadData:
                        int size = (buff[3] >> 4) + 1;
                        int offset = (buff[4] << 8 | buff[5]);
                        System.Array.Copy(buff, 6, mReadBuff, offset - mAddress, size);
                        if (mAddress + mSize == offset + size)
                            mReadDone.Set();
                        break;
                    case InputReport.OutputReportAck:
                        mWriteDone.Set();
                        break;
                }
            }
            if (WiimoteChanged != null)
                WiimoteChanged(this, new WiimoteChangedEventArgs(mWiimoteState));
            if (mStream != null && mStream.CanRead)
            {
                byte[] tBuff = new byte[REPORT_LENGTH];
                mStream.BeginRead(tBuff, 0, REPORT_LENGTH, new
System.AsyncCallback(OnReadData), tBuff);
            }
            mWiimoteState.IRSensors0.X = buff[6] | ((buff[8] >> 4) & 0x03) << 8;
            mWiimoteState.IRSensors0.Y = buff[7] | ((buff[8] >> 6) & 0x03) << 8;
            mWiimoteState.IRSensors1.X = buff[9] | ((buff[8] >> 0) & 0x03) << 8;
            mWiimoteState.IRSensors1.Y = buff[10] | ((buff[8] >> 2) & 0x03) << 8;
            mWiimoteState.ButtonState.A = (buff[2] & 0x08) != 0;
            mWiimoteState.ButtonState.B = (buff[2] & 0x04) != 0;
            mWiimoteState.ButtonState.Minus = (buff[2] & 0x10) != 0;
            mWiimoteState.ButtonState.Home = (buff[2] & 0x80) != 0;
            mWiimoteState.ButtonState.Plus = (buff[1] & 0x10) != 0;
            mWiimoteState.ButtonState.One = (buff[2] & 0x02) != 0;
            mWiimoteState.ButtonState.Two = (buff[2] & 0x01) != 0;
            mWiimoteState.ButtonState.Up = (buff[1] & 0x08) != 0;
            mWiimoteState.ButtonState.Down = (buff[1] & 0x04) != 0;
            mWiimoteState.ButtonState.Left = (buff[1] & 0x01) != 0;
            mWiimoteState.ButtonState.Right = (buff[1] & 0x02) != 0;
            mWiimoteState.AccelState.RawValues.X = buff[3];
            mWiimoteState.AccelState.RawValues.Y = buff[4];
```

```
            mWiimoteState.AccelState.RawValues.Width = buff[5];
            mWiimoteState.NunchukState.RawJoystick.X = buff[16];
            mWiimoteState.NunchukState.RawJoystick.Y = buff[17];
            mWiimoteState.NunchukState.AccelState.RawValues.X = buff[18];
            mWiimoteState.NunchukState.AccelState.RawValues.Y = buff[19];
            mWiimoteState.NunchukState.AccelState.RawValues.Width = buff[20];
            mWiimoteState.NunchukState.C = (buff[21] & 0x02) == 0;
            mWiimoteState.NunchukState.Z = (buff[21] & 0x01) == 0;
        }
        private int u;
        private int v;
        private int s;
        private int t;
        private byte[] ReadData(int address, short size)
        {
            if (u <= 13)
            {
                u = u + 1;
                mReadBuff = new byte[size];
                mAddress = address & 0xffff;
                mSize = size;
                Array.Clear(mBuff, 0, REPORT_LENGTH);
                mBuff[0] = (byte)OutputReport.ReadMemory;
                mBuff[1] = (byte)(((address & 0xff000000) >> 24));
                mBuff[2] = (byte)((address & 0x00ff0000) >> 16);
                mBuff[3] = (byte)((address & 0x0000ff00) >> 8);
                mBuff[4] = (byte)(address & 0x000000ff);
                mBuff[5] = (byte)((size & 0xff00) >> 8);
                mBuff[6] = (byte)(size & 0xff);
                HID.HidD_SetOutputReport(this.mHandle.DangerousGetHandle(), mBuff,
(uint)mBuff.Length);
                mReadDone.WaitOne(200, false);
            }
            return mReadBuff;
        }
        private void WriteData(int address, byte size, byte[] buff)
        {
            if (s <= 13)
            {
                s = s + 1;
                Array.Clear(mBuff, 0, REPORT_LENGTH);
                mBuff[0] = (byte)OutputReport.WriteMemory;
                mBuff[1] = (byte)(((address & 0xff000000) >> 24));
                mBuff[2] = (byte)((address & 0x00ff0000) >> 16);
                mBuff[3] = (byte)((address & 0x0000ff00) >> 8);
                mBuff[4] = (byte)(address & 0x000000ff);
                mBuff[5] = 1;
                Array.Copy(buff, 0, mBuff, 6, 1);
                HID.HidD_SetOutputReport(this.mHandle.DangerousGetHandle(), mBuff,
(uint)mBuff.Length);
            }
        }
        public static Int32 VID = 0x057e;
        public static Int32 PID1 = 0x0330;
        public static Int32 PID2 = 0x0306;
        public void SetReportType()
        {
            if (t <= 13)
            {
                t = t + 1;
                string devicePath = mDevicePath;
                mHandle = HID.CreateFile(devicePath, System.IO.FileAccess.ReadWrite,
System.IO.FileShare.ReadWrite, new System.IntPtr(), System.IO.FileMode.Open,
EFileAttributes.Overlapped, new System.IntPtr());
                HIDD_ATTRIBUTES attrib = new HIDD_ATTRIBUTES();
                attrib.Size = System.Runtime.InteropServices.Marshal.SizeOf(attrib);
```

```
                if (HID.HidD_GetAttributes(mHandle.DangerousGetHandle(), ref attrib))
                {
                    if (attrib.VendorID == VID && (attrib.ProductID == PID1 |
attrib.ProductID == PID2))
                    {
                        mStream = new System.IO.FileStream(mHandle,
System.IO.FileAccess.ReadWrite, REPORT_LENGTH, true);
                        if (mStream != null && mStream.CanRead)
                        {
                            byte[] vBuff = new byte[REPORT_LENGTH];
                            mStream.BeginRead(vBuff, 0, REPORT_LENGTH, new
System.AsyncCallback(OnReadData), vBuff);
                        }
                        byte[] iBuff = ReadData(0x0016, 7);
                        iBuff[0] = (byte)OutputReport.Status;
                        iBuff[1] = 0;
                        iBuff[0] = (byte)OutputReport.Status;
                        HID.HidD_SetOutputReport(this.mHandle.DangerousGetHandle(), iBuff,
(uint)iBuff.Length);
                        mStatusDone.WaitOne(200, false);
                    }
                    else
                    {
                        mHandle.Close();
                    }
                }
                mBuff[0] = (byte)OutputReport.IR;
                mBuff[1] = (byte)(0x04);
                WriteData(REGISTER_IR, 1, new byte[] { 0x08 });
                WriteData(REGISTER_IR_MODE, 1, new byte[] { 0x01 });
                Array.Clear(mBuff, 0, REPORT_LENGTH);
                mBuff[0] = (byte)OutputReport.Type;
                mBuff[1] = (byte)((false ? 0x04 : 0x00));
                mBuff[2] = (byte)WiimoteLib.InputReport.IRExtensionAccel;
                HID.HidD_SetOutputReport(this.mHandle.DangerousGetHandle(), mBuff,
(uint)mBuff.Length);
            }
        }
        public Wiimote(string devicePath)
        {
            mDevicePath = devicePath;
        }
        public class WiimoteCollection : System.Collections.ObjectModel.Collection<Wiimote>
        {
            public void FindAllWiimotes()
            {
                int q = new int();
                if (q <= 13)
                {
                    q = q + 1;
                    int index = 0;
                    System.Guid guid;
                    Microsoft.Win32.SafeHandles.SafeFileHandle mHandle;
                    HID.HidD_GetHidGuid(out guid);
                    System.IntPtr hDevInfo = HID.SetupDiGetClassDevs(ref guid, null, new
System.IntPtr(), 0x00000010);
                    SP_DEVICE_INTERFACE_DATA diData = new SP_DEVICE_INTERFACE_DATA();
                    diData.cbSize = System.Runtime.InteropServices.Marshal.SizeOf(diData);
                    while (HID.SetupDiEnumDeviceInterfaces(hDevInfo, new System.IntPtr(),
ref guid, index, ref diData))
                    {
                        System.UInt32 size;
                        HID.SetupDiGetDeviceInterfaceDetail(hDevInfo, ref diData, new
System.IntPtr(), 0, out size, new System.IntPtr());
                        SP_DEVICE_INTERFACE_DETAIL_DATA diDetail = new
SP_DEVICE_INTERFACE_DETAIL_DATA();
```

```
                        diDetail.cbSize = 5;
                        if (HID.SetupDiGetDeviceInterfaceDetail(hDevInfo, ref diData, ref
diDetail, size, out size, new System.IntPtr()))
                        {
                            System.Diagnostics.Debug.WriteLine(string.Format("{0}: {1} -
{2}", index, diDetail.DevicePath,
System.Runtime.InteropServices.Marshal.GetLastWin32Error()));
                            mHandle = HID.CreateFile(diDetail.DevicePath,
System.IO.FileAccess.ReadWrite, System.IO.FileShare.ReadWrite, new System.IntPtr(),
System.IO.FileMode.Open, EFileAttributes.Overlapped, new System.IntPtr());
                            HIDD_ATTRIBUTES attrib = new HIDD_ATTRIBUTES();
                            attrib.Size =
System.Runtime.InteropServices.Marshal.SizeOf(attrib);
                            if (HID.HidD_GetAttributes(mHandle.DangerousGetHandle(), ref
attrib))
                            {
                                if (attrib.VendorID == VID && (attrib.ProductID == PID1 |
attrib.ProductID == PID2))
                                {
                                    if (!WiimoteFound(diDetail.DevicePath))
                                        break;
                                }
                            }
                            mHandle.Close();
                        }
                        index++;
                    }
                }
            }
            protected bool WiimoteFound(string devicePath)
            {
                this.Add(new Wiimote(devicePath));
                return true;
            }
        }
        public void Dispose()
        {
            GC.SuppressFinalize(this);
        }
        public event EventHandler<WiimoteChangedEventArgs> WiimoteChanged;
        private const int REPORT_LENGTH = 22;
        private enum OutputReport : byte
        {
            Type = 0x12,
            IR = 0x13,
            Status = 0x15,
            WriteMemory = 0x16,
            ReadMemory = 0x17
        };
        public class WiimoteException : ApplicationException
        {
            public WiimoteException(string message)
                : base(message) { }
        }
        private const int REGISTER_IR = 0x04b00030;
        private const int REGISTER_IR_MODE = 0x04b00033;
        private const int REGISTER_EXTENSION_INIT_1 = 0x04a400f0;
        private const int REGISTER_EXTENSION_INIT_2 = 0x04a400fb;
        private const int REGISTER_EXTENSION_TYPE = 0x04a400fa;
        private Microsoft.Win32.SafeHandles.SafeFileHandle mHandle;
        private System.IO.FileStream mStream;
        private readonly byte[] mBuff = new byte[REPORT_LENGTH];
        private byte[] mReadBuff;
        private int mAddress;
        private short mSize;
        private readonly WiimoteState mWiimoteState = new WiimoteState();
```

```
        private readonly System.Threading.AutoResetEvent mReadDone = new
System.Threading.AutoResetEvent(false);
        private readonly System.Threading.AutoResetEvent mWriteDone = new
System.Threading.AutoResetEvent(false);
        private readonly System.Threading.AutoResetEvent mStatusDone = new
System.Threading.AutoResetEvent(false);
        private string mDevicePath = string.Empty;
        private readonly Guid mID = System.Guid.NewGuid();
        public delegate bool WiimoteFoundDelegate(string devicePath);
    }
    public enum InputReport : byte
    {
        Status = 0x20,
        ReadData = 0x21,
        OutputReportAck = 0x22,
        Buttons = 0x30,
        ButtonsAccel = 0x31,
        IRAccel = 0x33,
        ButtonsExtension = 0x34,
        ExtensionAccel = 0x35,
        IRExtensionAccel = 0x37
    };
    public enum ExtensionType : long
    {
        None = 0x000000000000,
        Nunchuk = 0x0000a4200000,
        ParitallyInserted = 0xffffffffffff
    };
    public class WiimoteState
    {
        public AccelState AccelState;
        public ButtonState ButtonState;
        public ExtensionType ExtensionType;
        public NunchukState NunchukState;
        public System.Drawing.Point IRSensors0;
        public System.Drawing.Point IRSensors1;
    }
    public struct NunchukState
    {
        public AccelState AccelState;
        public System.Drawing.Point RawJoystick;
        public bool C, Z;
    }
    public struct AccelState
    {
        public System.Drawing.Rectangle RawValues;
    }
    public struct ButtonState
    {
        public bool A, B, Plus, Home, Minus, One, Two, Up, Down, Left, Right;
    }
}
```

6. Use and Agreement Contract

Owner: Michael Andre Franiatte.

Contact: michael.franiatte@gmail.com.

Owning: All works from scratch of the owner.

Proof of Owning: Works published, and writings/speakings all over.

Requirements of Use: Pay the owner, quote the owner, agreement of the owner.

Availability of Works: Only under the shapes of the owner built, only for personal use.

Subjects of Claims: Works published by the owner on Google Play and Google Books.

Concerning Author Rights: Equations and codes from scratch of the owner, softwares built from it, all things of people arising from it.

End User License Agreement: A commercial license is required to use in personal manner. Do not redistributing in any manner, including by computer media, a file server, an email attachment, etc. Do not embedding in or linking it to another programs, source codes and assistances including internal applications, scripts, batch files, etc. Do not use for any kind of technical support including on customer or retailer computer, hardware or software development, research, discovery,

teachery, talk, speech, write, etc. Do not use for win money or for commercialisation of any products arising from my programs, source codes and assistances. Do not use and do not copy the way it run in other programs, source codes and assistances. Do not use without pay me, quote me and my agreement. Do not steal or copy or reproduce or modify or peer or share. Do not use in other manner than personal. It stand for my programs, source codes and assistances or programs, source codes and assistances stealing or copying or reproducing or modifying or peering or sharing my programs, source codes, and assistances. If you aren't agree you shall not use.

Terms of License and Price: The present contract acceptance is required to use works of the owner and built from it in all kind of manner. The price for each user shall be defined with the owner by contacting him and this for each subject of works the owner claims. Each user shall contact the owner for asking his agreement. It can be refused by the owner depending who asking and the price defined. People don't respecting the present contract shall not use the works of the owner.