

**Data wykonania ćwiczenia: 23.10.2017 r.**

 **Data oddania sprawozdania: 31.10.2017 r.**

Prowadzący: dr inż. Jan Nikodem Wydział Elektroniki **Informatyka**

Grupa: D

Termin: Pn TN 14:00

Kacper Pabian 226123

Paweł Szczęch 226160

**Sprawozdanie z laboratorium**

**Urządzenia Peryferyjne**

*Ćwiczenie 10*

*Obsługa Joysticka z wykorzystaniem DirectInput*

1. **Cel ćwiczenia**

Celem ćwiczenia było napisanie programu, który stwierdzi obecność joysticka podłączonego do portu usb. Po podłączeniu, wykorzystując API DirectInput program powinien odczytać nazwę urządzenia i symulować działanie myszy (przesuwanie kursora, naciśnięcie lewym, prawym przyciskiem). Dodatkowo do zrobienia mieliśmy prosty edytor graficzny, w którym można rysować za pomocą joysticka.

1. **Wstęp teoretyczny**

* **Warstwa HAL** (*Hardware Abstraction Layer*) – jest to sterownik umożliwiający sterowanie urządzeniem na płycie głównej. Stanowi ogniwo pośredniczące między sprzętem a jądrem systemu operacyjnego. Dla programisty jest sposobem komunikacji ze sprzętem przez udostępniane funkcje biblioteczne i sterowniki.
* **USB** (*Universal Serial Bus*) – komputerowy interfejs do urządzeń zastępujący stare porty szeregowe i porty równoległe. Opracowane przez firmy Microsofr, Intel, Compaq, IBM i DEC.

USB 2.0 (*high speed*) - Urządzenia zgodne z warunkami nowej specyfikacji mogą pracować z maksymalną szybkością 60 MB/s. Rzeczywista szybkość przesyłu danych zależy od konstrukcji urządzenia. Transmisja danych przez port odbywa się w trybie half duplex na jednej parze przewodów.

* **HID** (*Human Interface Driver)* – nazwa kodowa dla urządzeń peryferyjnych (tematu kursu), które służą do wprowadzania informacji przez człowieka do komputera. Podłączane są głównie przez port USB. Przykłady: joystick, klawiatura, mysz, drukarka.
* **DirectX** – zestaw funkcji API wspomagających generowanie grafiki (2D i 3D), dźwięku oraz innych zadań zwykle z grami i innymi aplikacjami multimedialnymi. Najczęściej widywany i wykorzystywany w obsłudze grafiki w grach komputerowych.
* **DirectInput** – biblioteka z DirectX obsługująca urządzenia peryferyjne. Zawiera metody odczytujące dane z tych urządzeń. Posiada możliwośc przyporządkowania określonych akcji do konkretnych przycisków tzw. Action Mapping.
* **COM** (*Component Object Model)* – standard definiowania i tworzenia interfejsów programistycznych na poziomie binarnym dla komponentów oprogramowania wprowadzony przez firmę Microsoft wraz z bibliotekami zapewniającymi podstawowe ramy i usługi dla współdziałania komponentów COM i aplikacji.

1. **Przebieg zajęć**

Na zajęciach do dyspozycji mieliśmy Joystick Logitech Attack 3. Pracowaliśmy na swoich komputerach, wykorzystując język C#, więc pierwsze co musieliśmy zrobić to odpowiednio dodać do projektu w Visual Studio 2015 odpowieni plik z DirectInput. Dzięki wcześniejszym zapoznaniu się z tematem wiedzieliśmy co mamy zrobić i szybko potrafiliśmy stworzyć odpowiednią metodę, dzięki której odczytaliśmy joystick. Linijka kodu, która pozwaliła nam odczytać podłączone urządzenia:

DeviceList gameControllerList = Manager.GetDevices(DeviceClass.GameControl, EnumDevicesFlags.AttachedOnly);

Zarówno DeviceList jak i Manager to klasy wzięte z biblioteki DirectInput.

Następnie musieliśmy sprawdzić czy zostały podłączone jakieś urządzenia (przynajmniej jedno). Funkcja zwracała nam string z informacjami o podłączonym urzadzeniu.

systemJoysticks = joystickDevice.DeviceInformation.InstanceName;

Dzięki informacjom o podłączonym urządzeniu mogliśmy zainicjalizować joystick w kolejnej funkcji nazwanej przez nas AcquireJoystick z argumentem string name (informacje o urządzeniu). Kilka odpowiednich operacji pozwoliło nam na ustawienie wszystkich potrzebnych informacji o joysticku oraz na ustawienie odpowiedniego zasięgu możliwych ruchów joysticka, jakie program odczytuje.

Nazwę okienka ustawiliśmy jako nazwę podpiętego urządzenia.

Kolejnym krokiem było napisanie metody UpdateStatus. Pierwszą operacją w tej metodzie była operacja Poll, która odświerza status urządzenia. W UpdateStatus umieściliśmy odpowiednie opracje, które pozwoliły nam na obsługę myszki jouystickiem (odczytywanie zmiany położenia, inicjalizacja i wywołanie odpowiednich przycisków). Aby symulować kliknięcie myszką zaimportowaliśmy funkcję mouse\_event z pliku user32.dll:

[DllImport("user32.dll", CharSet = CharSet.Auto)]

public static extern void mouse\_event(uint flag, uint \_x, uint \_y, uint btn, uint exInfo);

private const int MOUSEEVENT\_LEFTDOWN = 0x0002; //numer odpowiada danej opracji na myszce

private const int MOUSEEVENT\_LEFTUP = 0x0004;

private const int MOUSEEVENT\_RIGHTDOWN = 0x0008;

private const int MOUSEEVENT\_RIGHTUP = 0x0010;

Do tej operacji potrzebowaliśmy użyć biblioteki systemowej Runtime.InteropServices.

Przyciski pobraliśmy metodą Get buttons z JoystickState i wrzuciliśmy je do tablicy bajtów, po czym utworzyliśmy nową tablicę boolów, do której wsadziliśmy informacje o przyciskach z poprzedniej tablicy, którą potem wykorzystywaliśmy do ich obsługi. Kod konwersji tablicy i obsługi przycisków:

public JoystickState state;

byte[] jsButtons = state.GetButtons();

buttons = new bool[jsButtons.Length];

int i = 0;

foreach (byte button in jsButtons)

{

buttons[i] = button >= 128;

i++;

}

if (buttons[0])

{

if (leftClicked == false)

{

mouse\_event(MOUSEEVENT\_LEFTDOWN, 0, 0, 0, 0);

leftClicked = true;

}

}

else if (buttons[1])

{

if (rightClicked == false)

{

mouse\_event(MOUSEEVENT\_RIGHTDOWN, 0, 0, 0, 0);

rightClicked = true;

}

}

else

{

if (leftClicked == true)

{

mouse\_event(MOUSEEVENT\_LEFTUP, 0, 0, 0, 0);

leftClicked = false;

}

if (rightClicked == true)

{

mouse\_event(MOUSEEVENT\_RIGHTUP, 0, 0, 0, 0);

rightClicked = false;

}

}

}

UpdateStatus wykonujemy w zaimplementowanym czasomierzu.

Został nam jeszcze do zrobienia edytor graficzny, który zrobiliśmy za pomocą biblioteki systemowej Drawing, wykorzystując z niej obiekty: Graphics, Pen oraz Point.

Okno programu tworzone jest za pomocą WindowsForms, więc do wykonania edytora wystarczyło dodać Panel oraz wybraną ilość umieszczonych w nim PictureBox’ów. Do każdego z nich został przypisany kolor, na jaki ma zmienić się obiekt rysujący Pen po kliknięciu na niego:

private void redBox\_Click(object sender, EventArgs e)

{

pen.Color = redBox.BackColor;

mainBox.BackColor = redBox.BackColor;

}

By móc rysować należało zaimplementować jeszcze trzy metody: MouseDown, MouseUp, MouseMove, które w naszym programie zajmują się sprawdzaniem stanu lewego przycisku myszy oraz wywoływaniem funkcji rysującej.

MouseDown jest wywoływana, gdy przycisk jest wciśnięty. Zapisuje ona pozycję kursora oraz zmienia stan zmiennej ‘k’ odpowiadającej za wciśnięcie. W przeciwnym wypadku metoda MouseUp ustawia jej na stan 0.

private void Form1\_MouseDown(object sender, MouseEventArgs e)

{

sp = e.Location;

if (e.Button == MouseButtons.Left)

k = 1;

}

private void Form1\_MouseUp(object sender, MouseEventArgs e)

{

k = 0;

}

W ten sposób metoda MouseMove dostaje informacje kiedy należy wywołać funkcję DrawLine obiektu Graphics g.

private void Form1\_MouseMove(object sender, MouseEventArgs e)

{

if (k == 1)

{

ep = e.Location;

g = this.CreateGraphics();

g.DrawLine(pen, sp, ep)

}

sp = ep;

}

1. Wnioski

Wykonując ćwiczenie nauczyliśmy się obsługi urządzeń za pomocą DirectInput. Ćwiczenie wymagało dużego zaznajomienia się z biblioteką i zastanowania się nad odpowiednią implementacją metod. Ostatecznie udało nam się wykonać wszystkie zadania przydzielone nam w ramach ćwiczenia.