Programowanie kontrolerów gier i aplikacji w systemie Windows

Łukasz Stefanowicz, Grzegorz Łabiak

Streszczenie: Artykuł w sposób kompletny przedstawia metodę obsługi kontrolerów aplikacji i gier. Komponent DirectX, jako element składowy systemu operacyjnego Windows, jest technologią bezpośrednio wspierającą komunikację systemu operacyjnego z urządzeniami peryferyjnymi. Spośród wielu dostępnych na rynku manipulatorów do gier w referacie skupiono się na takich urządzeniach jak: joypad, joystick i kierownica. Dodatkowo opisano sposób oprogramowania obsługi klawiatury i myszy. Kody programów zostały przedstawione w języku C.

Słowa kluczowe: kontroler, Windows, DirectX, Direct Input, ANSI C, programowanie, urządzenia peryferyjne

1. WSTĘP

Początkujący programista może zetknąć się z problemem obsługi kontrolerów, umożliwiających sterowanie danym obiektem w sposób nie kolidujący z żadnymi innymi mechanizmami aplikacji. Generalnie wymaga się, by zdarzenie wygenerowane przez dane urządzenie nie miało wpływu na ilość wyświetlanych klatek oraz nie powodowało jakichkolwiek niezamierzonych działań.

Pomocne okazuje się wykorzystanie biblioteki DirectX firmy Microsoft. Wśród użytecznych modułów udostępnia ona Direct Input, umożliwiający bezproblemową obsługę urządzeń wejściowych, takich jak mysz, klawiatura, joystick, joypad, kierownica itp [3].

Niniejszy artykuł stanowi przedstawienie implementacji metody obsługi w/w kontrolerów, wraz z teoretycznym wstępem, mającym na celu przybliżenie historii rozwoju, a także wyszczególnienie podziału urządzeń.

2. KONTROLERY GIER I APLIKACJI

2.1. DEFINICJA

Za kontroler gier uważa się każde urządzenie wejściowe, które służy do kontrolowania gier komputerowych (PC) oraz wideo (konsole). W przypadku komputera PC funkcjonalność urządzeń takich, jak mysz czy klawiatura jest znacznie większa i sprowadza się także do obsługi systemu operacyjnego (są to podstawowe urządzenia wejściowe).

2.2. HISTORIA

Format kontrolera gier ulegał licznym modyfikacjom na przełomie wielu lat. Bezpośredni wpływ na to miał rozwój konsol, którego początek sięga lat 70. wieku XX. Pierwszym prymitywnym kontrolerem była tarcza, obsługująca konsolę Odyssey. Funkcjonalność urządzenia pozostawiała wiele do życzenia, wskutek czego wraz z Atari 2600 pojawiło się kolejne, mające drążek oraz przycisk. Początek lat 80. nie wprowadził nic nowego - wprowadzane kontrolery były odświeżoną wersją poprzednich pomysłów. Przełom nastąpił dopiero po 4 latach, kiedy firma Nintendo zaprezentowała konsolę NES oraz całkiem nowe urządzenie umożliwiające kontrolę rozgrywki. Można rzec, iż ukazano prekursor przyszłego standardu. Przedstawiony prostokątny joypad zawierał przycisk czerofunkcyjny, służący do wskazywania kierunku ruchu oraz dwa przyciski akcji, a także dwa przyciski wyboru (start, select). Pod koniec lat 80. konsola Sega Genesis ugruntowała wizję standardowego kontrolera gier, zmieniając jedynie kształt z prostokąta na bumerang. Początek lat 90. to także era rozwoju kontrolerów PC, które de facto były niczym innym jak kopią wersji konsolowych. 5 lat później firma Sony wprowadziła PlayStation wraz ze zmodyfikowanymi urządzeniami, wzbogaconymi o analogowe gałki, dające możliwość bardzo precyzyjnego wyznaczania toru ruchu. Firma Microsoft wprowadza system Windows oraz zaczyna się era w pełni trójwymiarowych gier. Rok później Sega ze swoją konsolą Dreamcast wprowadziła nową wizję kontrolerów wyspecjalizowanych, generalnie przypisanych do danego typu gier (kierownica, pistolet, wędka itp). Kolejne lata przyniosły rozwój systemu Windows 95 o pełną obsługę standardu USB, co zaowocowało wysypem wyspecjalizowanych urządzeń, zaczynając od joysticków i joypadów, kończąc na kierownicach [6].

2.3. PODZIAŁ KONTROLERÓW

Wyróżnić można następujące kontrolery:

- **joypad** – typ urządzenia, które ewoluowało wraz z rozwojem konsol do gier. Ma postać prostokąta, zawierającego przycisk czterofunkcyjny, obsługiwany przez lewy kciuk oraz przyciski akcji, z reguły cztery, obsługiwane przez kciuk prawy;
- **joystick** – przestrzenna wersja joypada, zawierająca drążek, w większości przypadków pełniący rolę przycisku czterofunkcyjnego oraz

przyciski akcji. Można wyróżnić dwa typy joysticków:
  - *analogowy* – kąt obrotu wynosi 360 stopni,
  - *cyfrowy – kąt obrotu definiują cztery kierunki główne oraz cztery pośrednie;*
- **mysz** i **klawiatura** – podstawowe urządzenia wejściowe umożliwiające obsługę systemu operacyjnego, pełniące także rolę kontrolerów gier i aplikacji;
- **kierownica** – kontroler wykorzystywany głównie w grach wyścigowych. Na cały zestaw składają się:
  - *koło kierownicy* – mające z reguły kąt obrotu od 200 do 900 stopni,
  - *pedały*: gaz, hamulec,
  - *skrzynia biegów*.
- **trackball** – potocznie zwany manipulatorem kulkowym, zawierający jedynie kulkę, trzymany w jednej ręce;
- **paddle** – kontroler zawierający pokrętło oraz jeden lub więcej przycisków. Kąt obrotu wynosił zazwyczaj 330 stopni. Obecnie tego typu kontrolery są niezwykle rzadko spotykane.

3. DirectX, a programowanie kontrolerów

3.1. Wstęp

Wraz z ukazaniem się systemu operacyjnego Windows 95 firmy Microsoft, udostępnione zostały pierwsze biblioteki DirectX w wersji 1.0 (rys. 1). Od początku istnienia miały one dostarczać programistom dostępu do multimediów. Wraz z biegiem lat sam pakiet DirectX ewoluował w bibliotekę dostarczającą rozwiązań umożliwiających obsługę grafiki 2D, 3D, dźwięku oraz urządzeń we-wy. Określone zadania realizują poszczególne komponenty:

- **Direct Graphics**, w skład którego wchodzą:
  - *Direct Draw (2D)* – obsługujący grafikę 2D,
  - *Direct 3D* – obsługujący grafikę 3D,
- **Direct Input** – odpowiedzialny za obsługę kontrolerów gier i aplikacji,
- **Direct Play** – wykorzystywany we wszelakich grach sieciowych,
- **Direct Sound** – odpowiedzialny za odtwarzanie oraz nagrywanie dźwięku,
- **Direct Show** – odpowiedzialny za odtwarzanie plików audio oraz wideo,
- **Direct Music** – umożliwiający odtwarzanie plików muzycznych stworzonych w programie Direct Music Producer,
- **Direct Setup** – odpowiedzialny za instalację komponentów biblioteki, jak i sprawdzenie ich aktualnych wersji.

Rys.1 Umiejscowienie DirectInput w warstwowym modelu systemu operacyjnego

3.2. DirectInput, a XInput

Wraz z wprowadzeniem DirectX w wersji 9, firma Microsoft dodała moduł XInput, umożliwiający także obsługę kontrolerów gier. Miał on nie tyle zastąpić DirectInput, co wspomóc w obsłudze wyspecjalizowanych kontrolerów nowej generacji. W rzeczywistości jego funkcjonalność ogranicza się do pełnej obsługi kontrolera konsoli XBox i na tym koniec. Z uwagi na spore ograniczenia, do których należą chociażby:

- obsługa 4 kontrolerów na raz,
- wsparcie 4 osi kontrolera, 10 przycisków i 2 dźwignie,
- niemożliwość obsługi klawiatury, myszy oraz w zasadzie innego kontrolera niż ten znany z konsoli XBox,

staje się niemal oczywiste, iż wykorzystanie modułu XInput jest pozbawione sensu wszędzie tam, gdzie zachodzi konieczność dostarczenia funkcjonalności obsługi standardowych kontrolerów gier i aplikacji [5]. W gruncie rzeczy do obsługi w/w kontrolera XBox z powodzeniem użyć można Direct Input, jednak i ta opcja niesie za sobą pewne ograniczenia:

- lewa i prawa dźwignia zachowywać się będzie jak zwykła oś, a nie jak niezależne osie analogowe,
- nie będą dostępne wibracje,
- urządzenia słuchawkowe nie będą dostępne.

4. Programowanie kontrolera typu joystick

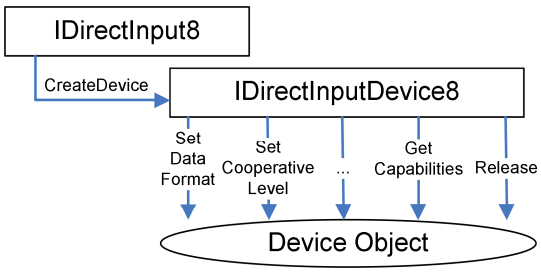
4.1. Obiekt główny, urządzenia, funkcja zwrotna oraz struktura danych

Obsługa kontrolerów typu joystick, czyli joypad, joystick i kierownica rozpoczyna się od deklaracji obiektu głównego typu *IDirectInput8* (rys. 2). Bez niego niemożliwa byłaby jakakolwiek interakcja z zainstalowanymi kontrolerami gier [4]. Deklaracja ta ma postać:

```
LPDIRECTINPUT8 g_lpDI;
```

Krok następny polega na zadeklarowaniu obiektu urządzenia (obsługa jednego urządzenia) lub tablicy obiektów (obsługa wielu urządzeń), w sposób następujący:

```
LPDIRECTINPUTDEVICE8  g_lpDIJoystick[10];
```



Rys.2 Interfejs DirectInput COM

Wyliczaniem i inicjalizacją urządzeń zajmuje się funkcja zwrotna, której zadaniem jest utworzenie obiektu wszystkich podłączonych kontrolerów. W przypadku

podłączenia kilku urządzeń to właśnie ona ma rozwiązać problem, umożliwiając dostęp do każdego z nich. Jej postać może być następująca:

```
BOOL CALLBACK EnumJoysticksCallback(const
  DIDEVICEINSTANCE* pdidInstance,VOID* pContext)
{
  HRESULT hr;
  if(noJoys == 9)
   noJoys -= 1;
  hr = g_lpDI->CreateDevice(pdidInstance->
   guidInstance, &g_lpDIJoystick[++noJoys], 0);
  if(FAILED(hr))
   return DIENUM_STOP;
  else
   return DIENUM_CONTINUE;
}
```

Funkcja ta pobiera dwa argumenty, gdzie: pierwszy jest referencją do wykrytego urządzenia, natomiast drugi referencją do liczby 32-bitowej, najczęściej ustawiany na *0*. Jak można zauważyć, funkcja zwrotna w przypadku wykrycia urządzenia tworzy jego obiekt o indeksie *noJoys* (zmienna globalna typu *int*) automatycznie inkrementowanym. Prosty warunek kontroluje, by wartość indeksu nie przekroczyła dozwolonego zakresu. Jeśli inicjalizacja urządzenia zakończy się sukcesem, funkcja zwraca wartość umożliwiającą kontynuowanie wyszukiwania urządzenia, natomiast w przeciwnym wypadku (w systemie nie ma już kontrolerów do zainicjalizowania) zwracana jest wartość, która kończy wyszukiwanie. Kiedy żaden kontroler nie zostanie odnaleziony, funkcja zwrotna kończy swoje działanie.

Przechowywanie stanu urządzenia można zrealizować na dwa sposoby:

- poprzez strukturę *DIJOYSTATE*,
- poprzez strukturę *DIJOYSTATE2*.

Odpowiednio DIJOYSTATE2 umożliwia obsługę większej ilości przycisków, dlatego też sugerowane jest użycie właśnie tej wersji [2]. Przykładowa deklaracja jest następująca:

```
DIJOYSTATE2 js;
```

Wszelakie struktury zdefiniowane są w pliku nagłówkowym *dinput.h*, który jest składową DirectX SDK.

4.2. INICJALIZACJA URZĄDZEŃ

4.2.1. UTWORZENIE OBIEKTU GŁÓWNEGO

Krokiem pierwszym jest inicjalizacja obiektu głównego za pomocą funkcji *DirectInput8Create* w sposób następujący:

```
DirectInput8Create(GetModuleHandle(NULL),
DIRECTINPUT_VERSION,          IID_IDirectInput8,
(void**)&g_lpDI, NULL);
```

Funkcja ta zwraca wartość typu *HRESULT* oraz przyjmuje pięć argumentów:

- **pierwszy**: uchwyt do programu,
- **drugi**: wersja DirectInput,
- **trzeci**: zwracany interfejs,
- **czwarty**: referencję do obiektu głównego,
- **piąty**: referencję do IUnknown, przeważnie *NULL*.

4.2.2. UTWORZENIE OBIEKTÓW URZĄDZEŃ

Następnie należy wyliczyć dostępne urządzenia za pomocą *EnumDevices* w sposób następujący:

```
g_lpDI->EnumDevices(DI8DEVCLASS_GAMECTRL,
EnumJoysticksCallback, NULL,
DIEDFL_ATTACHEDONLY);
```

Funkcja ta zwraca także wartość typu *HRESULT* oraz przyjmuje cztery argumenty:

- **pierwszy**: filtr urządzenia - w tym przypadku stała określająca kontrolery gier,
- **drugi**: funkcja zwrotna,
- **trzeci**: wartość 32-bitowa przekazywana do funkcji zwrotnej, w tym przypadku nieużywana,
- **czwarty**: zakres wyliczania, który może przyjmować następujące, predefiniowane wartości:
  - **DIEDFL_ALLDEVICES** − wszystkie zainstalowane kontrolery,
  - **DIEDFL_ATTACHEDONLY** − aktualnie podłączone i zainstalowane kontrolery,
  - **DIEDFL_FORCEFEEDBACK** − urządzenia wspierające force feedback, czyli efekty specjalne (np. wibracje),
  - **DIEDFL_INCLUDEALIASES** − obsługa kontrolerów łącznie z tymi, które są aliasami do innych urządzeń,
  - **DIEDFL_INCLUDEHIDDEN** − obsługa kontrolerów łącznie z urządzeniami ukrytymi,
  - **DIEDFL_INCLUDEPHANTOMS** − obsługa kontrolerów łącznie z urządzeniami typu Phantom.

Na tym etapie możliwe jest reagowanie na brak urządzeń. W tym celu wystarczy sprawdzić, czy obiekty urządzeń są równe *NULL*.

4.2.3. USTAWIENIE TRYBÓW PRACY

Następnie należy ustawić format danych za pomocą *SetDataFormat*. Przyjmuje ona jeden argument:

- wskaźnik do struktury *DIDATAFORMAT* [2].

W przypadku joysticka i tej funkcji można użyć predefiniowanej zmiennej globalnej: c_dfDIJoystick2, która umożliwia wykorzystanie struktury *DIJOYSTATE2* dla uzyskiwania danych. Postać wywołania jest następująca:

```
if(FAILED(g_lpDIJoystick->SetDataFormat(
&c_dfDIJoystick2))){
    //wszelakie reakcje na błąd
}
```

Oczywiście chcąc używać struktury *DIJOYSTATE* należy użyć predefiniowanej zmiennej *c_dfDI_Joystick*.

W dalszej kolejności wymagane jest ustawienie trybu współpracy urządzenia z aplikacją:

- **DISCL_BACKGROUND** − dostęp do urządzenia, nawet gdy program jest nieaktywny,
- **DISCL_FOREGROUND** − urządzenie jest aktywne, wtedy gdy aplikacja,
- **DISCL_NOWINKEY** − klawisz Windows nieaktywny,
- **DISCL_NONEXCLUSIVE** − wiele programów ma dostęp do urządzenia,
- **DISCL_EXCLUSIVE** − ekskluzywny dostęp do urządzenia (tylko 1 program).

Ustawiany jest za pomocą *SetCooperativeLevel*, przyjmującej dwa argumenty:

- **pierwszy**: uchwyt do okna

- **drugi**: kombinacja wyżej wymienionych flag

Wywołanie jest następujące:

```
if(FAILED(g_lpDIJoystick->
SetCooperativeLevel(NULL,
DISCL_EXCLUSIVE|DISCL_FOREGROUND))){
     //wszelakie reakcje na błąd
}
```

4.2.4. POBRANIE INFORMACJI O KONTROLERZE

Ostatnią czynnością podczas inicjalizacji jest pobranie informacji o kontrolerze. Służy do tego *GetCapabilities*, która jako argument przyjmuje referencję do obiektu typu *DIDEVCAPS* [2].

Na potrzeby przykładu w pierwszej kolejności tworzony globalnie jest obiekt *DIDEVCAPS*. Wymagane jest wyczyszczenie jego zawartości za pomocą *ZeroMemory*, a następnie ustawienie wymaganego pola *dwSize*. Przykładowy sposób realizacji tego zadania:

```
DIDEVCAPS g_diDevCaps;
ZeroMemory(&g_diDevCaps,sizeof(g_diDevCaps));
g_diDevCaps.dwSize = sizeof(DIDEVCAPS);
if(FAILED(g_lpDIJoystick->
GetCapabilities(&g_diDevCaps))){
     //reakcja na błąd
    }
```

Struktura *DIDEVCAPS* umożliwia pobranie szczegółowych informacji na temat inicjalizowanego kontrolera, a w szczególności:

- ilość dostępnych osi (*dwAxes*),
- ilość dostępnych przycisków (*dwButtons*),
- typ urządzenia (*dwDevType*),
- wersja firmware (*dwFirmwareRevision*),
- wersja hardware (*dwHardwareRevision,*)
- wersja sterownika (*dwFFDriverRevision*).

Przykład uzyskania informacji na temat ilości przycisków urządzenia:

```
unsigned long NoButtons =  g_diDevCaps.dwButtons;
```

W tym przypadku zmiennej *NoButtons* przypisywana jest wartość pola *dwButtons* obiektu *g_diDevCaps* struktury typu *DIDEVCAPS*.

Następnie możliwe jest pobranie informacji o obiekcie urządzenia. Struktura *DIDEVICEINSTANCE* przechowuje chociażby nazwę urządzenia, które zostało zainicjalizowane [2]. Przykład realizacji jest niemalże identyczny jak powyżej - zamiast *GetCapabilities* wywołać należy *GetDeviceInfo*.

Oprócz nazwy urządzenia *DIDEVICEINSTANCE* przechowuje także:

- typ urządzenia,
- unikalny id urządzenia nadany przez producenta,
- unikalny id obiektu urządzenia.

4.3. POBIERANIE STANU URZĄDZENIA

Pobranie stanu urządzenia ogranicza się jedynie do wywołania funkcji *GetDeviceState*, która przyjmuje dwa argumenty:

- **pierwszy**: wielość struktury,
- **drugi**: referencja do struktury przechowującej stan urządzenia.

Dobrym zwyczajem jest wywołanie funkcji Pool oraz Acquire, umożliwiających uzyskanie dostępu do urządzenia. Przykładowe wywołanie może być następujące:

```
hr=g_lpDIJoystick->Poll();
if(FAILED(hr))
 g_lpDIJoystick->Acquire();
g_lpDIJoystick->
GetDeviceState(sizeof(DIJOYSTATE2), &js);
```

4.4. REAGOWANIE NA ZDARZENIA

Struktura *DIJOYSTATE2* przechowuje informacje o stanie przycisków akcji, przycisku czterofunkcyjnego itp. i w prosty sposób można zareagować, jeśli zostało wygenerowane jakieś zdarzenie.

Wciśnięty przycisk ma wartość dziesiętną 128 (80 hex), natomiast zwolniony 0. Reagowanie na wciśnięty przycisk sprowadza się do sprawdzenia określonej wartości tablicy, np.:

```
int number = 0; //numer przycisku, start od 0
if (js.rgbButtons[number] & 0x80){
     //reagowanie na wciśnięty przycisk
}
else{
     //reagowanie na zwolniony przycisk
}
```

Przycisk czerofunkcyjny jest reprezentowany za pomocą dwóch osi:

- oś X:
  - przycisk wciśnięty w lewo: wartość dziesiętna 0,
  - przycisk wciśnięty w prawo: wartość dziesiętna 65535 (hex: FFFF),
- oś Y:
  - przycisk wciśnięty w górę: wartość dziesiętna 0,
  - przycisk wciśnięty w dół: wartość dziesiętna 65535 (hex: FFFF).

Przykład obsługi:

```
#define LEWO 0
#define DOL 65535

if(js.lX == LEWO){
      //reakcja na wciśnięcie klawisza 4-ro
      //funkcyjnego w lewo
    }
if(js.lY == DOL){
     //wciśnięcie w dół
    }
```

W stanie wolnym (przycisk czterofunkcyjny znajduje się w pozycji początkowej) oś X oraz Y ma wartość połówkową, a mianowicie dziesiętnie 32767 (hex: 7FFF). Oczywiście w przypadku kontrolera kierownicy - wartość osi X odpowiada aktualnemu wychyleniu koła kierownicy, wskutek czego możliwe jest właściwe oprogramowanie modelu reagowania danego obiektu, sterowanego tym urządzeniem.

4.5. ZAKOŃCZENIE PRACY Z URZĄDZENIEM

Zakończenie pracy z urządzeniem polega na:

- zwolnieniu dostępu do urządzenia,
- zwolnieniu urządzenia,
- zwolnieniu obiektu DirectInput.

Zadanie to może zostać zrealizowane następująco:

```
g_lpDIJoystick->Unacquire();
g_lpDIJoystick->Release();
g_lpDIJoystick = NULL;
g_lpDI->Release();
```

Po wykonaniu tych czynności praca z urządzeniem zostaje zakończona, wskutek czego nie jest możliwe dalsze odczytywanie stanu urządzenia.

5. PROGRAMOWANIE KLAWIATURY

5.1. OBIEKT GŁÓWNY, URZĄDZENIA ORAZ STRUKTURA DANYCH

Programowanie urządzenia klawiatury za pomocą Direct Input jest nieco podobne do sposobu przedstawionego we wcześniejszych wersach [4]. Obiekt główny oraz urządzenia wygląda identycznie. Nie jest natomiast wymagane użycie funkcji zwrotnej, aby wyliczyć urządzenia, ponieważ komputer PC domyślnie posiada jedną klawiaturę systemową. Aby zachować jednak integralność artykułu, deklaracja obiektów jest następująca:

```
LPDIRECTINPUT8 g_lpDI;
LPDIRECTINPUTDEVICE8 g_lpDIKeyboard;
```

W przypadku urządzenia klawiatury do przechowywania stanu urządzenia nie jest potrzebna żadna wyspecjalizowana struktura. Wystarczy tablica 256-cio elementowa, mogąca przechować pojedynczy znak. Deklaracja jest postaci:

```
unsigned char keystate[256];
```

5.2. INICJALIZACJA URZĄDZENIA

W pierwszej kolejności wymagana jest inicjalizacja obiektu głównego, za pomocą *DirectInput8Create*.

Krokiem następnym - inicjalizacja obiektu urządzenia wykorzystując funkcję *CreateDevice*. Zwraca ona wartość *HRESULT* oraz przyjmuje trzy argumenty:

- **pierwszy**: typ GUID żądanego urządzenia:
  - *GUID_SysKeyboard* - klawiatura,
  - *GUID_SysMouse* - myszka,
- **drugi**: referencja zmiennej urządzenia,
- **trzeci**: referencja do obiektu *IUnknown*,

Przykład:

```
if(FAILED(g_lpDI->CreateDevice(GUID_SysKeyboard,
&g_lpDIKeyboard, NULL))){
        //reakcja na błąd
     }
```

W kolejnym kroku ustawiany jest format danych za pomocą *SetDataFormat*. Przyjmuje ona jeden argument:

- referencję do struktury *DIDATAFORMAT*.

W przypadku klawiatury i tej funkcji można użyć predefiniowanej zmiennej globalnej: *c_dfDIKeyboard*. Postać wywołania jest następująca:

```
if(FAILED(g_lpDIKeyboard->SetDataFormat(
&c_dfDIKeyboard))){
     //wszelakie reakcje na błąd
    }
```

W dalszej kolejności wymagane jest ustawienie trybu współpracy urządzenia z aplikacją za pomocą *SetCooperativeLevel.*

Krokiem ostatnim jest uzyskanie dostępu do urządzenia za pomocą bezargumentowej funkcji *Acquire* w następujący sposób:

```
if(FAILED(g_lpDIKeyboard->Acquire())){
    //reakcja na błąd
}
```

5.3. POBIERANIE STANU URZĄDZENIA

Aby pobrać stan urządzenia wystarczy wywołać funkcję *GetDeviceState*, przyjmującej dwa argumenty ty:

- pierwszy: wielkość struktury
- drugi: wskaźnik do struktury przechowującej stan urządzenia.

Sposób wywołania jest następujący:

```
g_lpDIKeyboard->GetDeviceState(sizeof(unsigned
char[256]), (LPVOID)keystate);
```

5.4. REAGOWANIE NA ZDARZENIA

Aby móc stwierdzić, czy dany przycisk został wciśnięty należy sprawdzić, czy pobrana wartość jest równa 128 (hex: 80). W tym wypadku wymagane jest porównanie wszystkich elementów tablicy 256-cio elementowej, przechowującej aktualny stan klawiszy. Kody klawiszy Direct Input są różne od skaningowych klawiatury, w związku z czym warto zaznajomić się z ich rozmieszczeniem - plik nagłówkowy *dinput.h*, wchodzący w skład DirectX SDK zawiera wszelkie wymagane dane.

5.5. ZWALNIANIE URZĄDZENIA

Zwalnianie zainicjalizowanego urządzenia odbywa się w sposób identyczny, jak w punkcie 4.5.

6. PROGRAMOWANIE MYSZY

6.1. OBIEKT GŁÓWNY, URZĄDZENIA ORAZ STRUKTURA DANYCH

Programowanie urządzenia myszy jest bardzo podobne [4]. Deklaracja obiektu głównego oraz urządzenia wygląda identycznie. Inna jest natomiast struktura danych - istnieją dwa typy, różniące się ilością obsługiwanych przycisków:

- *DIMOUSESTATE* - 4 przyciski,
- *DIMOUSESTATE2* - 8 przycisków.

Z uwagi na ograniczenia struktury pierwszej, sugerowane jest wykorzystywanie drugiej. Deklaracja ma następującą postać:

```
DIMOUSESTATE2     mouse_state;
```

W przypadku ruchu względem określonej osi, przy uwzględnieniu odpowiednich wartości można wyróżnić znak przesunięcia:

- ruch w kierunku lewym-górnym: (-x,-y),
- ruch w kierunku prawym-górnym: (x, -y),
- ruch w kierunku lewym-dolnym: (-x,y),
- ruch w kierunku prawym-dolnym: (x,y).

Im szybszy jest ruch, tym wartości bezwzględne punktów będą miały większą wartość.

6.2. INICJALIZACJA URZĄDZENIA

W pierwszej kolejności należy zainicjalizować obiekt główny (*DirectInput8Create*), a także urządzenia (*CreateDevice*). Następnie wymagane jest ustawienie formatu danych za pomocą *SetDataFormat*. W przypadku myszki i tej funkcji można użyć predefiniowanej zmiennej globalnej: *c_dfDIMouse* (jeśli wykorzystywana będzie struktura *DIMOUSESTATE*) lub *c_dfDIMouse2* (*DIMOUSESTATE2*).

W dalszej kolejności wymagane jest ustawienie trybu współpracy urządzenia za pomocą *SetCooperativeLevel*, by w końcu móc uzyskać dostęp do urządzenia za pomocą bezargumentowej funkcji *Acquire.*

6.3. POBIERANIE STANU URZĄDZENIA

Aby pobrać stan urządzenia wystarczy wywołać funkcję *GetDeviceState*, która przyjmuje dwa argumenty:

- **pierwszy**: wielkość struktury
- **drugi**: referencja do obiektu struktury przechowującej stan urządzenia.

Zwraca ona wartość typu *HRESULT* i w przypadku błędu należy:

- ponownie uzyskać dostęp do urządzenia
- spróbować powtórnie pobrać dane.

6.4. REAGOWANIE NA ZDARZENIA

Mając pobrany stan urządzenia do obiektu struktury DIMOUSESTATE2, w prosty sposób można zareagować na dane zdarzenie. Aby sprawdzić, czy został wciśnięty przycisk myszki (wartość dziesiętna 128, hex 80) należy odwołać się do tablicy nazwanej *rgbButtons* o indeksie równym numerze przycisku, gdzie:

- 0 − przycisk lewy
- 1 − przycisk prawy itp.

Przykład sprawdzenia:

```
int number = 0;
if(mouse_state.rgbButtons[number] & 0x80){
        //reakcja na wciśnięty lewy przycisk
    }
else
    {
    }
```

Wartość liczbowa przesunięcia względem osi X, Y lub Z (scroll) posiadają pola struktury *DIMOUSESTATE2* o nazwach:

- lX, lY oraz lZ.

6.4. ZWALNIANIE URZĄDZENIA

Zwalnianie zainicjalizowanego urządzenia odbywa się w sposób identyczny, jak w punkcie 4.5.

7. PODSUMOWANIE

Możliwość obsługi kontrolerów gier i aplikacji w systemie Windows jest umiejętnością pożądaną przez każdego kodera gier oraz niekiedy bardziej zaawansowanych programów, wymagających określonych właściwości danego kontrolera. Użycie w tym celu biblioteki DirectX firmy Microsoft dostarcza wszelkich funkcjonalności, pozwalających na bezproblemową obsługę tychże urządzeń. Kompletny opis obrazujący wykonanie krok po kroku określonych czynności sprawia, iż informacje przedstawione w tym artykule mogą zostać w powodzeniem wykorzystane przy realizacji obsługi kontrolerów gier i aplikacji.

W przypadku każdego urządzenia wymagane jest wykonanie pewnych kroków inicjalizacyjnych, umożliwiających w dalszym etapie pobieranie danych. Oczywiście stan urządzenia należy odświeżać co określoną ilość czasu. Z reguły wystarczające jest powtarzanie 10 razy na sekundę.

LITERATURA

[1] Tooth V., *Visual C++ 5 Second Edition Unleashed*. Indianapolis, 1997

[2] http://msdn.microsoft.com/en-us/library/ee416867%28VS.85%29.aspx

[3] http://msdn.microsoft.com/en-us/library/ee416842%28VS.85%29.aspx

[4] http://msdn.microsoft.com/en-us/library/ee416958%28VS.85%29.aspx

[5] http://en.wikipedia.org/wiki/DirectInput

[6] Silberschatz A., Galvin P. B., Gagne G., *Podstawy systemów operacyjnych*, WNT Warszawa 2007

inż. Łukasz Stefanowicz
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji

ul. prof. Z. Szafrana 2
65-246 Zielona Góra
e-mail: przeciwnik@yahoo.com

dr inż. Grzegorz Łabiak
Uniwersytet Zielonogórski
Wydział Elektrotechniki, Informatyki i Telekomunikacji
Instytut Informatyki i Elektroniki

ul. prof. Z. Szafrana 2
65-246 Zielona Góra
tel.: 68 328 26 16
e-mail: G.Labiak@iie.uz.zgora.pl