Interfejsy Naturalne, Zadanie 1

Paweł Aszklar
P.Aszklar@mini.pw.edu.pl

Warszawa, 18 kwietnia 2016

Część I
Część laboratoryjna

1 Wstęp

Dany mamy program wyświetlający trójwymiarową scenę. Scenę obserwujemy z widoku pierwszoosobowego pewnej postaci znajdującej się wewnątrz domku. Naszym zadaniem jest umożliwienie użytkownikowi programu wydostanie postaci z pomieszczenia, w którym jest zamknięta. Procedury odpowiedzialne za wykonywanie różnych akcji są już zaimplementowane. Do napisania zostało wywołanie odpowiednich procedur w odpowiedzi na akcje wykonywane przez użytkownika za pomocą klawiatury i myszy.

Klasą zarządzającą animacjami oraz wyświetlaniem sceny jest `INScene` zdefiniowana w plikach `in_scene.h` oraz `in_scene.cpp`. Tą klasę modyfikować będziemy na potrzeby wykonania zadania (można oczywiście definiować własne funkcje i klasy pomocnicze). Za pomocą metod i pól klasy można wykonać następujące akcje:

- Obrót kamery:

  ```
  m_camera.Rotate(dx, dy);
  ```

  umożliwia spoglądanie w górę i w dół (`dx` - obrót wokół osi OX układu lokalnego kamery) oraz w lewo i w prawo (`dy` - obrót kamery wokół osi OY). Obroty wyrażone są jako kąty w radianach.

- Poruszanie postacią:

  ```
  MoveCharacter(dx, dz);
  ```

  umożliwia ruch postaci w przód i w tył (`dz`) oraz w lewo i w prawo (`dx`) - postać porusza się zawsze równolegle do podłoża, a wymienione kierunki zależą od tego, jak obrócona jest kamera.

- Otwierania i zamykanie drzwi:

  ```
  OpenDoor();
  CloseDoor();
  ToggleDoor();
  ```

  umożliwiają odpowiednio włączenie animacji otwierania lub zamykania drzwi lub zmianę aktualnej animacji na przeciwną.

Mamy też zdefiniowane dwie metody pomocnicze pozwalające na ustalenie pozycji i orientacji postaci względem drzwi:

- Metoda sprawdzająca, czy postać zwrócona jest w kierunku drzwi:

  ```
  bool FacingDoor();
  ```

- Metoda obliczająca odległość postaci od drzwi:

  ```
  float DistanceToDoor();
  ```

Zanim przejdziemy dalej, ustalmy schemat sterowania, jakiego będziemy używać.

- Obroty kamery wykonywane powinny być poprzez ruchy myszą z wciśniętym lewym przyciskiem.
- Przesuwanie postaci wykonywane powinno być klawiszami W, S, A, D (odpowiednio: w przód, w tył, w lewo, w prawo).
- Do otwierania i zamykania drzwi służyć powinien klawisz E. Dodatkowo zakładamy, że drzwi otworzyć lub zamknąć można tylko jeżeli postać zwrócona jest w kierunku drzwi i nie znajduje się od nich dalej niż 1 jednostkę długości.

Powyższy schemat jest tylko propozycją. Można zastosować dowolny inny pod warunkiem zachowania tej samej funkcjonalności.

2 Obsługa zdarzeń systemowych

W pierwszym ćwiczeniu umożliwimy sterowanie programem poprzez obsługę zdarzeń systemowych. System Windows informuje program o zdarzeniach za pomocą wiadomości, które dostarczane są do definiowanej przez programistę funkcji zwrotnej `WindowProc`

```
LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg,
                            WPARAM wParam, LPARAM lParam);
```

W naszym programie wiadomości te trafiają do metody `ProcessMessage` klasy `INScene`:

```
bool ProcessMessage(WindowMessage& msg);
```

Informacje o wiadomości zawarte są w strukturze:

```
struct WindowMessage
{
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    LRESULT result;
};
```

Pole `message` zawiera typ wiadomości, natomiast jej parametry zapisane są w polach `wParam` oraz `lparam` (ich znaczenie zależy od typu wiadomości). Jeżeli wiadomość zostanie obsłużona w metodzie `ProcessMessage` powinna ona zwrócić wartość `true`. Ponadto do pola `result` należy przypisać wartość jaka zwrócona powinna być z funkcji `WidnowProc`. Informacje o wiadomościach dostarczanych przez system dla klawiatury można znaleźć tutaj, a dla myszy tutaj.

Uwagi:

- Obracanie kamery oraz przełączanie animacji drzwi można wykonywać bezpośrednio w metodzie `ProcessMessage`.
- Przesunięcie myszy należy przeskalować przed użyciem go do obrotu kamery. Obrót o jeden radian przy przesunięciu o 1 piksel spowoduje zbyt szybkie ruchy kamery. Zastosowanie współczynnika skalowania około 0.002 powinno dać zadowalające rezultaty.
- W przeciwieństwie do pozostałych akcji, przesuwanie postaci realizowane powinno być w metodzie `Update`:

  ```
  void Update(float dt);
  ```

  Jej parametrem jest ilość czasu (w sekundach), jaka upłynęła od ostatniej klatki animacji. Dzięki niemu możemy zapewnić, że postać poruszać się będzie ze stałą prędkością niezależnie od tego ile klatek na sekundę wyświetlanych jest przez program. Prędkość poruszania postaci należy ustalić na około 3 jednostki na sekundę. Wyższe wartości utrudnią manewrowanie, a ekstremalnie duże wartości spowodować mogą błędy w wykrywaniu kolizji.

3 Obsługa urządzeń Direct Input

Drugie ćwiczenie związane będzie z obsługą myszy i klawiatury za pomocą biblioteki Direct Input. W celu uniknięcia powtarzania dwukrotnie tych samych akcji należy oczywiście wykomentować kod związany z obsługą wiadomości systemowych.

3.1 Inicjalizacja

Do funkcji biblioteki Direct Input dostęp mamy za pomocą interfejsu `IDirectInput8`, który musimy zainicjalizować. Służy do tego funkcja:

```
HRESULT DirectInput8Create(HINSTANCE hinst, DWORD dwVersion,
                           REFIID riidltf, LPVOID * ppvOut,
                           LPUNKNOWN punkOuter);
```

Pierwszym parametrem jest wartość `HINSTANCE` aplikacji (zwracana jest ona przez metodę `getHandle()` klasy `INScene`). Drugi parametr określa wersję biblioteki, którą chcemy użyć - zazwyczaj jako wartości używamy stałej `DIRECTINPUT_VERSION`. Trzecim parametrem jest identyfikator interfejsu - należy przekazać stałą `IID_DIDirectInput8`. Czwartym parametrem jest adres wskaźnika, w którym zapisany ma zostać adres tworzonego interfejsu. Ostatni parametr zwykle nie jest używany i ma wartość `NULL`. Wywołanie tej funkcji wyglądać będzie więc następująco:

```
IDirectInput8* di;
HRESULT result = DirectInput8Create(getHandle(),
                     DIRECTINPUT_VERSION, IID_IDirectInput8,
                     reinterpret_cast<void**>(&di), nullptr);
```

3.2 Inicjalizacja urządzeń

Dostęp do urządzeń (myszy, klawiatury, itp.) mamy za pomocą interfejsu `IDirectInputDevice8`. Tworzymy je za pomocą metody `CreateDevice` uzyskanego powyżej interfejsu `IDirectInput8`:

```
HRESULT CreateDevice(REFGUID rguid,
                     LPDIRECTINPUTDEVICE * lplpDevice,
                     LPUNKNOWN pUnkOuter);
```

Jako pierwszy parametr podajemy GUID urządzenia, do którego dostęp chcemy uzyskać. Dla myszy użyć należy stałej `GUID_SysMouse`, dla klawiatury `GUID_SysKeyboard`. Drugim parametem jest adres wskaźnika, w którym zapisany ma zostać adres tworzonego interfejsu. Trzeci zwykle nie jest używany i ma wartość `NULL`. Przykładowe wywołanie w celu uzyskania dostępu do myszy wyglądać będzie więc następująco:

```
IDirectInputDevice8* pMouse;
HRESULT result = di->CreateDevice(GUID_SysMouse, &pMouse,
                                  nullptr);
```

Następnie ustalić należy format w jakim odbierane będą dane z urządzenia. Służy do tego metoda `SetDataFormat` interfejsu `IDirectInputDevice8`:

```
HRESULT SetDataFormat(LPCDIDATAFORMAT lpdf);
```

Biblioteka posiada kilka formatów predefiniowanych. W przypadku myszy możemy przekazać stałą `c_dfDIMouse`, w przypadku klawiatury `c_dfDIKeyboard`. Przykładowe wywołanie dla myszy będzie miało więc postać:

```
HRESULT result = pMouse->SetDataFormat(c_dfDIMous);
```

Trzecim krokiem inicjalizacji urządzenia jest określenie trybu w jakim aplikacja chce współpracować z urządzeniem. Służy do tego metoda `SetCooperativeLevel` interfejsu `IDirectInputDevice8`:

```
HRESULT SetCooperativeLevel(HWND hwnd, DWORD dwFlags);
```

Pierwszym parametrem jest `HWND` okna aplikacji - w naszym programie można ją uzyskać wywołując `m_window.getHandle()`. Jako drugi parametr przekazujemy kombinacje flag określających tryb współpracy. Przykładowo można przekazać flagi `DISCL_FOREGROUND`, mówiącą, że urządzenie dostępne będzie, gdy okno aplikacji jest aktywne, oraz `DISCL_NONEXCLUSIVE` mówiącą, że urządzenie nie jest potrzebne aplikacji na wyłączność. Przykładowe wywołanie:

```
HRESULT result = pMouse->SetCooperativeLevel(
                     m_window.getHandle(),
                     DISCL_FOREGROUND | DISCL_NONEXCLUSIVE)
```

Ostatnim elementem jest uzyskanie dostępu do urządzenia, poprzez wywołanie metody `Acquire`, np.:

```
HRESULT result = pMouse->Acquire();
```

3.3 Odczyt stanu urządzenia

Stan urządzenia odczytać można za pomocą metody `GetDeviceState`:

```
HRESULT GetDeviceState(DWORD cbData, LPVOID lpvData);
```

Pierwszym parametrem jest rozmiar bufora w którym zapisane maja być dane urządzenia, drugim jest adres tego bufora.

Wywołanie tej metody może się nie powieść i nie zawsze jest to błąd krytyczny (aplikacja może tymczasowo stracić dostęp do urządzenia, np. gdy okno programu przestanie być aktywne). W przypadku, gdy wywołanie

zwróci wartość DIERR_INPUTLOST albo DIERR_NOTACQUIRED, należy wywołać metodę Acquire i spróbować pobrać dane ponownie. W przypadku, gdy nie da się natychmiastowo uzyskać ponownego dostępu do urządzenia, Acquire zwróci wartość DIERR_INPUTLOST lub E_ACCESSDENIED. W takim przypadku należy uznać, że w danej klatce nie udało się odczytać danych z urządzenia i kontynuować działanie programu. W celu obsługi tych wszystkich przypadków, zdefiniować możemy pomocniczą funkcję/metodę GetDeviceState:

```
const unsigned int GET_STATE_RETRIES = 2;
const unsigned int ACQUIRE_RETRIES = 2;
bool GetDeviceState(IDirectInputDevice8* pDevice,
                    unsigned int size, void* ptr)
{
    if (!m_device)
        return false;
    for (int i = 0; i < GET_STATE_RETRIES; ++i)
    {
        HRESULT result = pDevice->GetDeviceState(size, ptr);
        if (SUCCEEDED(result))
            return true;
        if (result != DIERR_INPUTLOST &&
            result != DIERR_NOTACQUIRED)
            ... //error! throw exeption
        for (int j = 0; j < ACQUIRE_RETRIES; ++j)
        {
            result = pDevice->Acquire();
            if (SUCCEEDED(result))
                break;
            if (result != DIERR_INPUTLOST &&
                result != E_ACCESSDENIED)
                ... //error! throw exeption
        }
    }
    return false;
}
```

Funkcja zwraca true jeśli uda się odczytać dane z urządzenia, lub false w przeciwnym wypadku.

To, jaka zmienna posłużyć powinna za bufor danych zależy od tego, jaki format danych przekazaliśmy do wywołania SetDataFormat.

Jeżeli w przypadku myszy przekazaliśmy wartość c_dfDIMouse, dane myszy możemy odczytać do struktury DIMOUSESTATE. Struktura ta zawiera pola, które czy i które przyciski są wciśnięte oraz o ile przesunął się kursor myszy od ostatniego wywołania GetState. Przykład odczytu danych:

```
DIMOUSESTATE state;
if (GetDeviceState(pMouse, sizeof(DIMOUSESTATE), &state))
{
    ... //wykorzystanie danych urządzenia
}
```

Dla klawiatury biblioteka nie definiuje specjalnej struktury na dane. Jeżeli przy tworzeniu urządzenia użyliśmy formatu `c_dfDIKeyboard` za bufor posłuży tablica 256 elementów typu `BYTE`. Każdy element odpowiada jednemu klawiszowi i zawiera informację, czy jest on wciśnięty, czy nie. Przykład użycia:

```
BYTE state[256];
if (GetDeviceState(pKeyboard, sizeof(BYTE)*256, state))
{
    ... //wykorzystanie danych urządzenia
}
```

Biblioteka definiuje szereg stałych postaci `DIK_*` (np. `DIK_W`, `DIK_ESCAPE`, itd.) określające który element tablicy odpowiada któremu klawiszowi.

3.4 Zwolnienie zasobów

Przed zakończeniem działania aplikacji należy zwolnić uzyskane zasoby. W przypadku urządzeń (interfejs `IDirectInputDevice8`) należy zwolnić dostęp do urządzenia poprzez wywołanie metody `Unacquire`, a następnie zwolnić sam interfejs wywołując `Release()`. Te dwie operacje należy wykonać zarówno dla klawiatury jak i dla myszy. Przykład:

```
IDirectInputDevice8* pDevice;
...
pDevice->Unacquire();
pDevice->Release();
```

Zwolnić trzeba też interfejs `IDirectInput8` - wywołując `Release()`. Przykład:

```
IDirectInput8* di;
...
di->Release();
```

3.5 Podsumowanie

Użycie biblioteki DirectInput podzielone jest na trzy główne części : inicjalizację, odczyt danych i zwolnienie zasobów. W klasie `INScene` zdefiniowane są oddzielne metody, w których znaleźć się powinna każda z tych części. Inicjalizację należy przeprowadzić wewnątrz metody `InitalizeInput`,

zwolnienie zasobów w metodzie `Shutdown`, natomiast odczyt danych najlepiej zaimplementować wewnątrz metody `Update`

Część II
Część projektowa

4 Wymagania projektu

Projekt polega na rozszerzeniu aplikacji rozwijanej w poprzedniej części o możliwość przedefiniowania schematu sterowania postacią w programie z użyciem myszy i klawiatury oraz kontrolerów gier (joysticków, gamepadów, kierownic itp.).

1. W przypadku wykrycia podłączonego kontrolera gier, użytkownik powinien mieć możliwość wyboru sposobu sterowania, decydując się na kontroler lub mysz i klawiaturę. Jeżeli podłączonych jest więcej niż jeden kontroler, należy użyć któregokolwiek z nich.

2. Sterowanie myszą i klawiaturą – użytkownik programu powinien mieć możliwość wyboru które klawisze klawiatury służą do poruszania postaci i otwierania/zamykania drzwi. W tym przypadku można założyć, że sterowanie obrotami kamery za pomocą myszki pozostaje bez zmian.

3. Sterowanie kontrolerem – użytkownik powinien móc zdefiniować schemat sterowania wszystkimi akcjami (poruszanie postaci, otwieranie i zamykanie drzwi, obracanie kamerą) za pomocą jednego kontrolera. Do każdej akcji program powinien pozwalać przypisać dowolny element kontrolera (przyciski, D-PAD, przepustnice, osie analogowe itp.). *Uwaga!* Kilka osi analogowych może być sterowanych za pomocą tej samej gałki czy drążka. W przypadku przypisywania akcji do osi analogowych należy zastosować tzw. *martwą strefę*, gdzie małe odchylenia osi od położenia neutralnego nie są rejestrowane – zapobiegnie to przypadkowemu przypisaniu do akcji niepoprawnej osi.

Interfejs do przedefiniowania sterowania należy zaprojektować samemu, np.: można posłużyć się dołączoną biblioteką do renderowania tekstu do stworzenia prostego menu, lub wczytywać schemat sterowania z pliku – w takim wypadku należy stworzyć oddzielny program do tworzenia takich plików konfiguracyjnych.