Fundusze Europejskie
Wiedza Edukacja Rozwój

Unia Europejska
Europejski Fundusz Społeczny

Programowanie zespołowe 2.0

Biblioteka SFML – obsługa zdarzeń

Scenariusz zajęć

Czas realizacji: 180 minut
Cele ogólne: Obsługa zdarzeń związanych z oknem oraz urządzeniami zewnętrznymi.
Cel szczegółowy: Przygotowanie mechanizmu interakcji z użytkownikiem w przygotowywanym projekcie.
Konieczne wiadomości wstępne: Podstawy programowania z użyciem biblioteki SFML.
Metoda prowadzenia zajęć: Dyskusja z elementami wykładu.

Zdarzenia

Każde zdarzenie jest obiektem typu `sf::Event`. Jest to unia zwierająca pola związane z różnymi typami zdarzeń. Znając typ obsługiwanego zdarzenia (pole `event.type`) można wykorzystać odpowiednie pola do odczytania związanych z nim dodatkowych danych.

Instancje obiektów zdarzeń mogą być zwracane wyłącznie przez funkcje `waitEvent()` oraz `pollEvent()` z klasy `sf::Window`. Funkcja `waitEvent()` jest funkcją blokującą, t.j. po wywołaniu czeka aż pojawi się zdarzenie, które będzie można obsłużyć. Funkcja `pollEvent()` jest funkcją nieblokującą, co oznacza że po wywołaniu sprawdza czy zdarzenie zaszło zwracając `true` w przypadku pozytywnym (dodatkowo modyfikując obiekt `event`) oraz `false` w przypadku negatywnym (nie modyfikując obiektu `event`).

Pętla oczekująca na zdarzenia wygląda zatem następująco:

```
while (window.pollEvent(event)) {
    /* Obsłuż pojawiające się zdarzenia */
}
```

lub alternatywnie:

```
while (window.waitEvent(event)) {
    /* Obsłuż pojawiające się zdarzenia */
}
```

Obsługa zdarzeń

Obsługę konkretnych zdarzeń wewnątrz pętli oczekującej na zdarzenia najwygodniej jest przeprowadzić za pomocą struktury `switch`.

```
switch (event.type) {

    case sf::Event::Closed:         // Obsluga zamknięcia okna
        window.close();
        break;

    case sf::Event::KeyPressed:     // Obsługa naciśnięcia klawisza
        // Kod obsługi zdarzenia
        break;

    default:                        // Ignorowanie pozostałych zdarzeń
        break;
}
```

Zdarzenia związane z oknami

Wyróżniamy następujące zdarzenia związane z oknami:

`Closed` – Zdarzenie to pojawia się podczas próby zamknięcia okna dowolnym sposobem (kliknięcie na przycisk zamknięcia, użycie skrótu klawiszowego, itp.). W większości wypadków reakcją na to zdarzenie jest wywołanie funkcji składowej `close()` zamykającej okno. Przechwycenie zdarzenia umożliwia jednak wykonanie pewnych działań przed rzeczywistym zamknięciem okna (zamknięcie otwartych plików, prośba o potwierdzenie zamknięcia aplikacji, itp.).

`Resized` – Zdarzenie to pojawia się w momencie zmiany rozmiaru okna przez akcję użytkownika lub użycie funkcji `setSize()`. Nowy rozmiar przechowywany jest w strukturze `event.size` (pola `width` oraz `height`). Przechwycenie zdarzenia umożliwia n.p. dostosowanie wyglądu aplikacji do nowego rozmiaru okna.

`LostFocus` – Zdarzenie to pojawia się w momencie kiedy okno przestaje być aktywne. Jego przechwycenie umożliwia np. wstrzymanie działania animacji aplikacji do czasu kiedy okno znów będzie aktywne.

`GainedFocus` – Zdarzenie to pojawia się w momencie kiedy okno ponownie staje się aktywne. Jego przechwycenie umożliwia np. ponowne uruchomienie działania animacji aplikacji zatrzymanej kiedy okno przestało być aktywne.

Ćwiczenie 1. Napisz program obsługujący przedstawione powyżej zdarzenia związane z oknem. Dla każdego przechwyconego zdarzenia program powinien wydrukować na ekranie odpowiednią informację.

Klawiatura

Programowa obsługa naciśnięć klawiszy na klawiaturze jest możliwa na dwa sposoby: przez przechwytywanie zdarzeń związanych z klawiaturą oraz przez bezpośrednie odpytywanie fizycznego urządzenia.

Zdarzenia związanie z klawiaturą

Podobnie jak w przypadku zdarzeń związanych z oknami pojawiające się zdarzenie związane z klawiaturą (naciśnięcie/zwolnienie przycisku, itp.) może zostać obsłużone przez program jeżeli zostanie przechwycone w pętli oczekującej na zdarzenia (główne okno programu). Możemy wyróżnić następujące zdarzenia związane z klawiaturą:

`KeyPressed` – Zdarzenie to pojawia się w momencie naciśnięcia dowolnego klawisza na klawiaturze. Jeśli klawisz zostaje wciśnięty przez dłuższą chwilę pojawi się kolejno wiele zdarzeń oznaczających jego naciśnięcie. Aby temu zapobiedz można wywołać funkcję `setKeyRepeatEnabled(false)` z klasy `sf::Window`. Przywrócenie działania domyślnego uzyskamy wywołując tę samą funkcję z argumentem `false`.

Obiekt zdarzenia posiada pole `event.key` zawierające dodatkowe informacje o kodzie naciśniętego klawisza oraz stanie klawiszy kontrolnych (`ctrl`, `alt`, `shift`, `system`).

```
if (event.key.code == sf::Keyboard::A) {
    cout << "Klawisz [A] zostal nacisniety" << endl;
    cout << "ctrl: " << event.key.control << endl;
    cout << "alt: " << event.key.alt << endl;
    cout << "shift: " << event.key.shift << endl;
    cout << "system: " << event.key.system << endl;
}
```

Lista nazw stałych oznaczających kody klawiszy jest dostępna na stronie https://www.sfml-dev.org/documentation/2.5.0/classsf_1_1Keyboard.php.

`KeyReleased` – Zdarzenie to pojawia się w momencie zwolnienia nacisku na dowolny klawisz na klawiaturze. Szczegóły dotyczące użytego klawisza można uzyskać identycznie jak w przypadku zdarzenia `KeyPressed`.

`TextEntered` – Zdarzenie to pojawia się w momencie wpisywania znaku alfanumerycznego. Wartość wpisanego znaku znajduje się w polu `event.text`. W odróżnieniu od zdarzenia `KeyPressed` znaki uzyskiwane poprzez kombinację kilku klawiszy (np. znak `Ą` w języku polskim) są obsługiwane jako pojedyncze zdarzenie. Nie ma więc dodatkowej konieczności sprawdzania stanu klawiszy kontrolnych.

Odpytywanie klawiatury w czasie rzeczywistym

Odpytywanie klawiatury w czasie rzeczywistym może być zrealizowane przy użyciu klasy `sf::Keyboard`. Posiada ona statyczną metodę `isKeyPressed()` zwracającą wartość logiczną reprezentującą stan wciśnięcia klawisza lub jego brak. Działanie tej funkcji jest niezależne od aktywności okna, co oznacza że może sprawdzać stan klawiszy nawet jeśli okno nie jest aktywne.

```
if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
    cout << "Klawisz [<-] jest wcisniety" << endl;
    robot.idzWLewo(10);
}
```

Uwaga. W przypadku obsługi zdarzenia związanego z klawiaturą w momencie jego pojawienia się program jest informowany *następujący przycisk został wciśnięty*, natomiast w przypadku odpytywania w czasie rzeczywistym program pyta w dowolnym momencie *czy następujący przycisk jest naciśnięty*.

Ćwiczenie 2. Napisz program, który dla każdego naciśnięcia klawisza alfanumerycznego (reprezentującego literę lub cyfrę) umieści wewnątrz okna element tekstowy z odpowiadającą mu zawartością. Współrzędne oraz kolory elementów tekstowych powinny być wybierane w sposób losowy.

Ćwiczenie 3. Napisz program umożliwiający sterowanie elementem graficznym znajdującym się wewnątrz okna za pomocą klawiszy strzałek. Zadbaj o to, żeby poruszający się element nigdy nie opuścił obszaru okna.

Mysz

Podobnie jak w przypadku klawiatury programowa obsługa myszy jest możliwa na dwa sposoby: przez przechwytywanie zdarzeń związanych z myszą oraz przez bezpośrednie odpytywanie fizycznego urządzenia. Aktualnie biblioteka SFML jest w stanie obsłużyć wyłącznie jedną mysz podłączoną do komputera.

Zdarzenia związane z myszą

Możemy wyróżnić następujące zdarzenia związane z myszą:

`MouseButtonPressed` – Zdarzenie to pojawia się w momencie naciśnięcia klawisza myszy. Biblioteka SFML obsługuje klawisze lewy, prawy i środkowy, oraz dwa klawisze dodatkowe (boczne). Obiekt zdarzenia posiada pole `event.mouseButton` zawierające informację o kodzie naciśniętego klawisza oraz współrzędne kursora myszy. Nazwy stałymi odpowiadających poszczególnym klawiszom myszy są następujące: `Left`, `Right`, `Middle`, `XButton1` oraz `XButton2`.

```
if(event.mouseButton.button == sf::Mouse::Right) {
    std::cout << "Prawy przycisk myszy nacisniety" << std::endl;
    std::cout << "x: " << event.mouseButton.x << std::endl;
    std::cout << "y: " << event.mouseButton.y << std::endl;
 }
```

`MouseButtonReleased` – Zdarzenie to pojawia się w momencie zwolnienia klawisza myszy. Dane dotyczące zwolnionego klawisza oraz współrzędnych kursora myszy uzyskiwane są identycznie jak w przypadku zdarzenia `MouseButtonPressed`.

`MouseWheelScrolled` – Zdarzenie to pojawia się w momencie obrotu kółka myszy. Biblioteka SFML obsługuje zarówno obrót pionowy jak i poziomy. Obiekt zdarzenia posiada pole `event.mouseWheelScroll` zawierające informacje o kierunku przewijania, liczbie skoków kółka oraz współrzędnych kursora myszy.

```
if(event.mouseWheelScroll.wheel == sf::Mouse::VerticalWheel)
    cout << "Przewijanie pionowe" << endl;
else if(event.mouseWheelScroll.wheel == sf::Mouse::HorizontalWheel)
    cout << "Przewijanie poziome" << endl;

cout << "skok kolka: " << event.mouseWheelScroll.delta << endl;
cout << "x: " << event.mouseWheelScroll.x << endl;
cout << "y: " << event.mouseWheelScroll.y << endl;
```

`MouseMoved` – Zdarzenie to pojawia się kiedy kursor myszy porusza się nad oknem (nawet jeśli okno nie jest aktywne). Obiekt zdarzenia posiada pole `event.mouseMove` zawierający informacje o współrzędnych kursora myszy (`event.mouseMove.x` oraz `event.mouseMove.y`).

`MouseEntered` – Zdarzenie to pojawia się w momencie kiedy kursor myszy znajdzie się nad oknem programu.

`MouseLeft` – Zdarzenie to pojawia się w momencie kiedy kursor myszy znajdzie się poza oknem programu.

Uwaga. Dla wszystkich zdarzeń dotyczących myszy współrzędne kursora wyznaczane są względem pozycji okna i odpowiadają punktom położonym wewnątrz okna.

Odpytywanie myszy w czasie rzeczywistym

Odpytywanie myszy w czasie rzeczywistym jest realizowane za pomocą klasy `sf::Mouse`. Zawiera ona zestaw statycznych funkcji składowych pozwalających sprawdzić stan naciśnięcia jej klawiszy oraz pozycję kursora.

Stan naciśnięcia klawisza sprawdzamy za pomocą funkcji `isButtonPressed()`.

```
if(sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
    gun.fire();
}
```

Pozycję kursora pobieramy za pomocą funkcji `getPosition()`. Jeśli podamy jako argument obiekt typu `sf::Window` zwrócona zostanie pozycja kursora wewnątrz okna, w przeciwnym przypadku – pozycja kursora na ekranie.

```
// Wspolrzedne kursora na ekranie
sf::Vector2i globalPosition = sf::Mouse::getPosition();

// Wspolrzedne kursora wewnatrz okna
sf::Vector2i localPosition = sf::Mouse::getPosition(window);
```

Funkcja `setPosition()` pozwala umieścić kursor myszy na wybranej pozycji. Jako pierwszy obowiązkowy argument przyjmuje ona współrzędne kursora. Drugi opcjonalny argument określa czy współrzędne dotyczą całego ekranu czy też wyłącznie wnętrza okna.

```
// Ustaw wspolrzedne kursora na ekranie
sf::Mouse::setPosition(sf::Vector2i(10, 50));

// Ustaw wspolrzedne kursora wewnatrz okna
sf::Mouse::setPosition(sf::Vector2i(10, 50), window);
```

Ćwiczenie 4. Napisz program otwierający puste okno w trybie pełnoekranowym. Program powinien obsługiwać kliknięcia myszy umieszczając niewielki czerwony kwadrat w miejscu kliknięcia lewym przyciskiem oraz niewielkie zielone koło w miejscu kliknięcia prawym przyciskiem.

Ćwiczenie 5. Napisz program tworzący puste okno dowolnych rozmiarów oraz obsługujący przewijanie kółka myszy. Przewijanie w pionie powinno zmieniać pionowy rozmiar okna, natomiast przewijanie w poziomie – jego rozmiar poziomy. Jeśli Twoja mysz nie obsługuje przewijania poziomego przewijanie pionowe powinno zmieniać rozmiar okna w obu kierunkach.

Ćwiczenie 6. Napisz program uniemożliwiający zamknięcie okna za pomocą kliknięcia myszą. Po zbliżeniu kursora myszy do przycisku zamykania okna powinien on zostać przeniesiony tuż za ten przycisk (poruszając się w tym samym kierunku).

Joystick

Podobnie jak w przypadku klawiatury i myszy programowa obsługa joysticka jest możliwa na dwa sposoby: przez przechwytywanie zdarzeń związanych z joystickiem oraz przez bezpośrednie odpytywanie fizycznego urządzenia. Biblioteka SFML jest w stanie obsłużyć jednocześnie 8 joysticków oraz 32 przyciski na każdym z nich.

Zdarzenia związane z joystickiem

Wyróżniamy następujące zdarzenia związane z joystickiem:

`JoystickButtonPressed` – Zdarzenie to pojawia się kiedy jeden z przycisków joysticka zostanie wciśnięty. Obiekt zdarzenia posiada pole `event.joystickButton` zawierające identyfikator joysticka oraz identyfikator naciśniętego przycisku.

```
if (event.type == sf::Event::JoystickButtonPressed) {
    cout << "Przycisk wcisniety" << endl;
    cout << "joystick id: "
         << event.joystickButton.joystickId << endl;
    cout << "przycisk: "
         << event.joystickButton.button << endl;
}
```

`JoystickButtonReleased` – Zdarzenie to pojawia się kiedy jeden z przycisków joysticka zostanie zwolniony. Identyfikatory joysticka oraz zwolnionego przycisku mogą zostać uzyskane identycznie jak w przypadku zdarzenia `JoystickButtonPressed`.

`JoystickMoved` – Zdarzenie to pojawia się kiedy joystick poruszy się wzdłuż jednej z osi. Biblioteka SFML obsługuje 8 osi: `X`, `Y`, `Z`, `R`, `U`, `V`, `POV X` oraz `POV Y`. Ich dokładne mapowanie jest zależne od sterownika joysticka. Obiekt zdarzenia posiada pole `event.joystickMove` zawierające identyfikator joysticka, identyfikator osi ruchu oraz aktualną pozycję w zakresie [-100,100].

```
if (event.type == sf::Event::JoystickMoved) {
    if (event.joystickMove.axis == sf::Joystick::X) {
        cout << "Ruch wzdluz osi X" << endl;
        cout << "joystick id: "
             << event.joystickMove.joystickId << endl;
        cout << "aktualna pozycja: "
             << event.joystickMove.position << endl;
    }
}
```

`JoystickConnected` – Zdarzenie to pojawia się kiedy joystick zostaje podłączony do komputera. Obiekt zdarzenia posiada pole `event.joystickConnect` zawierające identyfikator podłączonego joysticka.

```
if (event.type == sf::Event::JoystickConnected)
    cout << "Podlaczono joystick: "
         << event.joystickConnect.joystickId << endl;
```

`JoystickDisconnected` – Zdarzenie to pojawia się kiedy joystick zostaje odłączony od komputera. Obiekt zdarzenia posiada pole `event.joystickConnect` zawierające identyfikator odłączonego joysticka.

```
if (event.type == sf::Event::JoystickDisconnected)
    cout << "Odlaczono joystick: "
         << event.joystickConnect.joystickId << endl;
```

Odpytywanie joysticka w czasie rzeczywistym

Odpytywanie joysticka w czasie rzeczywistym może być realizowane za pomocą klasy `sf::Joystick`. Zawiera ona zestaw statycznych funkcji składowych, które umożliwiają sprawdzenie aktualnego stan przycisków oraz osi joysticka.

Funkcja `isConnected()` pozwala sprawdzić czy joystick o identyfikatorze podanym jako parametr jest podłączony.

```
if (sf::Joystick::isConnected(0)) {
    cout << "Joystick nr 0 jest podlaczony" << endl;
}
```

Liczbę dostępnych przycisków możemy sprawdzić za pomocą funkcji `getButtonCount()`. Jej argumentem jest identyfikator joysticka.

```
unsigned int ilePrzyciskow = sf::Joystick::getButtonCount(0);
```

Dostępne osie możemy sprawdzić za pomocą funkcji `hasAxis()`. Pierwszym argumentem jest identyfikator joysticka, drugim natomiast identyfikator osi.

```
if (sf::Joystick::hasAxis(0, sf::Joystick::Z)) {
        cout << "Os Z jest dostepna" << endl;
}
```

Za pomocą funkcji `isButtonPressed()` możemy sprawdzić czy wybrany przycisk jest aktualnie wciśnięty. Argumentami funkcji są identyfikator joysticka oraz identyfikator przycisku.

```
if (sf::Joystick::isButtonPressed(0, 1)) {
    gun.fire();
}
```

Pozycję na wybranej osi możemy sprawdzić za pomocą funkcji `getAxisPosition()`. Jej argumentami są identyfikator joysticka oraz identyfikator przycisku.

```
double x = sf::Joystick::getAxisPosition(0, sf::Joystick::X);
double y = sf::Joystick::getAxisPosition(0, sf::Joystick::Y);
robot.move(x, y);
```

Ćwiczenie 7. Napisz program, który umieści w oknie dwa elementy graficzne o różnych kolorach. Ruch jednego z nich powinien być sterowany za pomocą klawiszy strzałek, natomiast ruch drugiego – za pomocą joysticka. Zderzenie obiektów powinno zakończyć działanie programu. Zadbaj o to, żeby żaden z obiektów nie mógł opuścić obszaru okna. Opracuj sposób wykrywania zderzenia obiektów.

Odnośniki i załączniki

[1] Biblioteka SFML: https://www.sfml-dev.org/

[2] Licencja Zlib/PNG: http://opensource.org/licenses/Zlib/