Fundusze Europejskie
Wiedza Edukacja Rozwój

Unia Europejska
Europejski Fundusz Społeczny

Programowanie zespołowe 2.0

Biblioteka SFML – wprowadzenie

Scenariusz zajęć

Czas realizacji: 180 minut
Cele ogólne: Zapoznanie z biblioteką SFML. Tworzenie prostych programów graficznych.
Cel szczegółowy: Projekt głównego okna dla przygotowywanego projektu.
Konieczne wiadomości wstępne: Podstawy programowania obiektowego.
Metoda prowadzenia zajęć: Dyskusja z elementami wykładu.

Biblioteka SFML

Biblioteka **SFML** (Simple and Fast Multimedia Library) jest wieloplatformową biblioteką graficzną. Jest ona dostępna na otwartej licencji (Zlib/png) umożliwiającej swobodne tworzenie opartego na niej oprogramowania (w tym również komercyjnego). W skład biblioteki wchodzi pięć modułów: *system*, *window*, *graphics*, *audio* oraz *network*. Udostępniają one interfejs do wielu zasobów systemowych znacząco ułatwiając tworzenie gier oraz innych aplikacji multimedialnych.

Instalacja i konfiguracja

Biblioteka SFML może zostać uruchomiona na wielu platformach systemowych (Linux, Windows, MacOS). Wiele środowisk programistycznych (m.in. Visual Studio, Code::Blocks czy Xcode) może zostać skonfigurowanych do korzystania z SFML. Szczegółowe instrukcje instalacji oraz konfiguracji biblioteki dostępne są na stronie projektu
https://www.sfml-dev.org/tutorials/2.5/

Uwaga. Podczas wyboru ścieżki instalacji istotne jest dopasowanie pobieranej wersji biblioteki do używanej przez nas wersji systemu operacyjnego oraz kompilatora języka C++ (32-bitowa czy 64-bitowa).

Ćwiczenie 1. Skonfiguruj używane przez Ciebie środowisko programistyczne do korzystania z biblioteki SFML.

Pierwszy program

Okno jest obiektem klasy `sf::Window` dostępnej w pliku nagłówkowym `SFML/Window.hpp` (lub jednej z klas ją rozszerzających, np. `sf::RenderWindow`). Aby utworzyć okno należy stworzyć obiekt podając odpowiednie parametry konstruktora (rozmiar oraz tytuł okna):

```
sf::Window window(sf::VideoMode(800, 600), "Nowe okno");
```

lub po stworzeniu obiektu za pomocą konstruktora domyślnego wywołać funkcję `create()`.

```
sf::Window window;
window.create(sf::VideoMode(800, 600), "Nowe okno");
```

Po utworzeniu okna należy uruchomić pętlę oczekującą na zdarzenia związane z oknem i obsłużyć jego zamknięcie.

Uwaga. Pierwszy argument konstruktora oraz funkcji `create()` określa rozmiar okna liczony bez uwzględnienia jego brzegów oraz paska tytułowego.

Przykład 1. Kod programu tworzącego puste okno i oczekującego na jego zamknięcie.

```
#include <SFML/Window.hpp>

int main() {
    sf::Window window(sf::VideoMode(800, 600), "Nowe okno");

    while(window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {        // Oczekuj na zdarzenia
            if (event.type == sf::Event::Closed) // Zamknij okno
                window.close();

        }
    }

    return 0;
}
```

Uwaga. Dodanie pętli oczekującej na zdarzenia jest konieczne do prawidłowego działania programu. W przeciwnym przypadku okno programu przestanie odpowiadać.

Dekoracje okien

Podczas tworzenia okna możliwe jest przekazanie do konstruktora oraz funkcji `create()` opcjonalnego trzeciego argumentu odpowiedzialnego za oczekiwany styl okna. Możliwe są następujące jego wartości:

- `sf::Style::None` – Brak dekoracji (paska tytułowego, ramek, przycisków, itp.); opcja użyteczna np. dla ekranów powitalnych; nie może być łączona z innymi.
- `sf::Style::Titlebar` – Okno posiada pasek tytułowy.
- `sf::Style::Resize` – Okno posiada przycisk maksymalizacji; rozmiar okna może być zmieniany za pomocą myszy.
- `sf::Style::Close` – Okno posiada przycisk zamknięcia.
- `sf::Style::Default` – Styl domyślny będący skróconym zapisem dla połączenia `Titlebar | Resize | Close`.

- `sf::Style::Fullscreen` – Okno jest wyświetlane na pełnym ekranie; ten tryb nie może być łączony z innymi; może wymagać zmiany rozdzielczości okna.

Ćwiczenie 2. Wypróbuj w praktyce wszystkie dostępne style okien.

Podstawowe operacje na oknach

Obiekty klasy `sf::Window` umożliwiają wykonanie podstawowych operacji związanych z oknami. Zmiana pozycji okna na ekranie możliwa jest przez wywołanie funkcji składowej

```
window.setPosition(sf::Vector2i(10, 50));
```

gdzie jako parametr podajemy współrzędne lewego górnego rogu okna. Pobranie pozycji okna na ekranie jest natomiast możliwe za pomocą funkcji składowej

```
sf::Vector2u size = window.getPosition();
unsigned int x = size.x;
unsigned int y = size.y;
```

Możliwa jest również zmiana rozmiaru okna. Uzyskujemy ją za pomocą wywołania

```
window.setSize(sf::Vector2u(1024, 768));
```

Podobnie jak w przypadku pozycji okna na ekranie, jego aktualny rozmiar również może zostać pobrany za pomocą odpowiedniej funkcji składowej

```
sf::Vector2u size = window.getSize();
unsigned int width = size.x;
unsigned int height = size.y;
```

Zmianę tytułu okna możemy uzyskać wywołując

```
window.setTitle("Nowy tytul");
```

Jeśli chcemy aby kursor myszy był niewidoczny kiedy znajduje się nad oknem używamy funkcji składowej

```
window.setMouseCursorVisible(false);
```

Aby przywrócić wyświetlanie kursora myszy nad oknem należy wywołać tę samą funkcję z argumentem `true`.

Pełną dokumentację klasy `sf::Window` zawierającą szczegółowy opis wszystkich funkcji składowych (podobnie jak dla pozostałych klas dostępnych w bibliotece SFML) znajdziemy na stronie: https://www.sfml-dev.org/documentation/2.5.1/annotated.php.

Ćwiczenie 3. Wypróbuj w praktyce przedstawione wyżej operacje na oknach.

Rysowanie wewnątrz okna

Biblioteka SFML dostarcza zestaw prostych narzędzi umożliwiających rysowanie grafiki wewnątrz okna. W tym celu należy użyć klasy `sf::RenderWindow` dostępnej w pliku nagłówkowym `SFML/Graphics.hpp`. Jest ona rozszerzeniem klasy `sf::Window`, zatem zawiera cały jej publiczny interfejs. Co więcej, obsługa zdarzeń związanych z oknami w obu przypadkach jest identyczna.

Przykład 2. Szkielet programu umożliwiającego rysowanie wewnątrz okna.

```
#include <SFML/Graphics.hpp>

int main() {
    sf::RenderWindow window(sf::VideoMode(800, 600), "Nowe okno");

    while(window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {        // Oczekuj na zdarzenia
            if (event.type == sf::Event::Closed) // Zamknij okno
                window.close();
        }

        // Czyszczenie zawartości okna oraz ustawienie koloru tła
        window.clear(sf::Color::Black);

        // Rysowanie wewnątrz okna za pomocą funkcji draw()

        // Odświeżenie wyświetlanej zawartości okna
        window.display();
    }

    return 0;
}
```

Funkcja `clear()` powoduje wyczyszczenie zawartości okna oraz ustawienie tła o zadanym kolorze. Jej wywołanie jest konieczne przed rozpoczęciem rysowania. W przeciwnym przypadku poprzednia zawartość okna będzie widoczna pod nowym rysunkiem.

Funkcja `dislpay()` powoduje wyświetlenie w oknie programu wszystkiego co zostało narysowane od czasu jej poprzedniego wywołania. Jej wywołanie jest konieczne, ponieważ wszystkie rysowane elementy nie są umieszczane bezpośrednio w oknie programu lecz w specjalnym buforze. Wywołanie funcji `display()` powoduje skopiowanie zawartości tego bufora do okna programu.

Funkcja `draw()` powoduje „narysowanie” elementu graficznego podanego jako argument poprzez umieszczenie go w specjalnym buforze. Nie będzie on widoczny do momentu wywołania funkcji `display()`.

Kolor

Kolor rysowanego elementu możemy zmienić za pomocą funkcji

```
element1.setFillColor(sf::Color(100, 250, 50));
element2.setFillColor(sf::Color::Blue);
```

gdzie kolor podawany jest jako jedna z predefiniowanych stałych lub przez określenie wartości jego składowych w modelu RGB.

Szerokość oraz kolor brzegu elementu określamy za pomocą funkcji

```
element.setOutlineThickness(10.0);
element.setOutlineColor(sf::Color(250, 150, 100));
```

Brzeg rysowany jest na zewnątrz elementu. Oznacza to, że koło o promieniu 10 oraz brzegu grubości 5 będzie miało całkowity promień równy 15.

Figury geometryczne

Biblioteka SFML dostarcza wielu predefiniowanych kształtów umożliwiających rysowanie figur geometrycznych.

Koło o zadanym promieniu tworzymy następująco

```
sf::CircleShape kolo(200.0);
```

Rozmiar (promień) już istniejącego koła możemy zmienić za pomocą funkcji

```
kolo.setRadius(40.0);
```

Możliwa jest również zmiana dokładności rysowania koła przez ustawienie liczby rysowanych wierzchołków. W takim przypadku koło jest rysowane jako wielokąt foremny.

```
kolo.setPointCount(100);
```

Prostokąt o zadanym rozmiarze tworzymy następująco

```
sf::RectangleShape prostokat(sf::Vector2f(120.0, 50.0));
```

Rozmiar już istniejącego prostokąta możemy zmienić za pomocą funkcji

```
prostokat.setSize(sf::Vector2f(100.0, 100.0));
```

Dowolny wielokąt foremny możemy uzyskać podając dodatkowy argument dla konstruktora koła oznaczający liczbę wierzchołków

```
sf::CircleShape trojkat(80.0, 3);
sf::CircleShape kwadrat(80.0, 4);
sf::CircleShape osmiokat(80.0, 8);
```

Dowolną figurę wypukłą możemy uzyskać za pomocą klasy `sf::ConvexShape`. W tym celu należy określić liczbę wierzchołków, a następnie podać ich współrzędne. Kolejność wierzchołków jest istotna. Powinny one być podawane zgodnie z ruchem wskazówek zegara lub w kierunku przeciwnym. W przeciwnym przypadku figura może zostać narysowana niepoprawnie.

```
sf::ConvexShape figura;                          // Pusta figura
figura.setPointCount(5);                         // Liczba wierzcholkow
figura.setPoint(0, sf::Vector2f(0.0, 0.0));      // Wspolrzedne
figura.setPoint(1, sf::Vector2f(150.0, 10.0));
figura.setPoint(2, sf::Vector2f(120.0, 90.0));
figura.setPoint(3, sf::Vector2f(30.0, 100.0));
figura.setPoint(4, sf::Vector2f(0.0, 50.0));
```

Linię prostą możemy uzyskać na dwa sposoby. Jeśli chcemy kontrolować jej grubość użyjemy kształtu `sf::RectangleShape`

```
sf::RectangleShape linia(sf::Vector2f(150.0, 5.0));
```

Natomiast linię prostą bez kontroli grubości tworzymy następująco

```
sf::Vertex line[] = {
    sf::Vertex(sf::Vector2f(10.0, 10.0)),
    sf::Vertex(sf::Vector2f(150.0, 150.0))
};
window.draw(line, 2, sf::Lines);
```

Ćwiczenie 4. Wypróbuj w praktyce rysowanie rysowanie wewnątrz okna figur geometrycznych w różnych kształtach i kolorach.

Elementy tekstowe

Aby możliwe było tworzenie i rysowanie elementów tekstowych konieczne jest określenie używanego fontu (klasa `sf::Font`). Definicja fontu powinna zostać załadowana z pliku znajdującego się na dysku (np. jeden z fontów systemowych). Bardzo istotne jest poprawne określenie ścieżki dostępu do pliku.

```
sf::Font font;
if (!font.loadFromFile("arial.ttf")) {
    // Zgłoszenie błędu
}
```

Elementy tekstowe reprezentowane są przez obiekty klasy `sf::Text`. Tworzenie nowego elementu przebiega następująco

```
sf::Text text;
text.setFont(font);              // Wybór fontu
text.setString("Hello world");   // Ustawienie wartości
```

Aby ustalić rozmiar tekstu (w pikselach) oraz jego kolor możemy użyć funkcji

```
text.setCharacterSize(24);
text.setFillColor(sf::Color::Red);
```

Jest również możliwa zmiana stylu rysowanego elementu tekstowego

```
text.setStyle(sf::Text::Bold | sf::Text::Underlined);
```

Więcej informacji dotyczących elementów tekstowych znajduje się w dokumentacji biblioteki SFML https://www.sfml-dev.org/documentation/2.5.0/classsf_1_1Text.php.

Ćwiczenie 5. Wypróbuj w praktyce rysowanie wewnątrz okna elementów tekstowych stosując różne fonty, style, rozmiary i kolory.

Transformacje elementów graficznych i tekstowych

Elementy rysowane w oknie takie jak opisane wyżej kształty oraz obiekty tekstowe posiadają wspólny zestaw funkcji składowych umożliwiających ich podstawowe transformacje.

Pozycja

Pozycję elementu w oknie określamy za pomocą funkcji

```
element.setPosition(10.0, 50.0);
```

Przesunięcie elementu względem jego aktualnej pozycji określamy za pomocą funkcji

```
element.move(2.0, 5.0);
```

Aktualną pozycję elementu uzyskamy za pomocą funkcji

```
sf::Vector2f position = element.getPosition();
```

Ćwiczenie 6. Zaprogramuj ruch elementu (np. koła) wewnątrz okna. W każdej iteracji głównej pętli programu element powinien przesunąć się w ustalonym kierunku o ustaloną liczbę pikseli. Zadbaj o to, żeby poruszający się element nigdy nie opuścił obszaru okna. W tym celu powinien się on odbijać od jego brzegów.

Rotacja

Ustawienie elementu pod ustalonym kątem uzyskiwany jest za pomocą funkcji

```
element.setRotation(45.0);
```

gdzie argumentem jest kąt w stopniach. Obrót wykonywany jest zgodnie z ruchem wskazówek zegara.

Obrót elementu względem jego aktualnego położenia określamy za pomocą funkcji

```
element.rotate(10.0);
```

Aktualny kąt ustawienia elementu uzyskamy za pomocą funkcji

```
double rotation = element.getRotation();
```

Skalowanie

Współczynnik skalowania określa sposób zmiany rozmiaru obiektu. Domyślnym współczynnikiem skalowania jest 1. Wartości powyżej 1 oznaczają powiększenie, natomiast mniejsze od 1 – zmniejszenie. Wartości ujemne powodują dodatkowo odbicie lustrzane. Dla wszystkich funkcji dotyczących skalowania określamy oddzielnie współczynniki skalowania w poziomie oraz w pionie.

Rozmiar elementu określamy za pomocą funkcji

```
element.setScale(4.f, 1.6f);
```

Zmianę rozmiaru elementu względem jego aktualnego rozmiaru określamy za pomocą funkcji

```
element.scale(2.0, 5.0);
```

Aktualną wartość skalowania elementu uzyskamy za pomocą funkcji

```
sf::Vector2f skala = element.getScale();
```

Ćwiczenie 7. Zmodyfikuj program z ćwiczenia 6 tak, żeby przy każdym odbiciu od brzegu okna poruszający się element zwiększał rozmiar.

Ćwiczenie 8. Zmodyfikuj program z ćwiczenia 8 określając minimalny oraz maksymalny rozmiar obiektu. Po osiągnięciu rozmiaru maksymalnego (odpowiednio minimalnego) każde odbicie od ściany powinno powodować zmniejszenie (odpowiednio powiększenie) poruszającego się elementu.

Ćwiczenie 9. Zmodyfikuj program z ćwiczenia 8 tak, żeby przy każdym odbiciu od brzegu okna poruszający się element razem ze swoim rozmiarem odpowiednio zwiększał lub zmniejszał swoją prędkość.

Punkt odniesienia

Punkt odniesienia jest specjalnie wyznaczonym punktem rysowanego elementu mającym istotne znaczenie dla wszystkich opisanych powyżej transformacji. Pozycja elementu jest określona przez współrzędne jego punktu odniesienia, rotacja elementu jest wykonywana wokół jego punktu odniesienia, zaś skalowanie elementu odbywa względem punktu odniesienia. Domyślnie punktem odniesienia każdego elementu jest jego prawy górny róg (punkt o współrzędnych (0,0)).

Współrzędne punktu odniesienia dla elementu określamy za pomocą funkcji

```
element.setOrigin(10.f, 20.f);
```

Aktualne współrzędne punktu odniesienia elementu uzyskamy za pomocą funkcji

```
sf::Vector2f origin = element.getOrigin();
```

Ćwiczenie 10. Wypróbuj w praktyce obroty dowolnego elementu graficznego wokół różnych punktów odniesienia (kolejne wierzchołki, środek, itp.).

Odnośniki

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

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