SFML - Prozori

Objektno programiranje - 1. vježbe (2. dio)

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

12. ožujka 2025. godine

SFML prozor

- objekt klase **sf::Window** (SFML koristi imenički prostor **sf**)

Primjer 1.

```
#include <iostream>
#include <SFML/Window.hpp>
using namespace std;

int main() {
  sf::Window prozor(sf::VideoMode(800,600),"Prozor");
  while (prozor.isOpen()) { }
  return 0;
}
```

- u gornjem smo stvorili i otvorili jedan prozor
- kako bi nešto vidjeli dodali petlju koja se vrti dok god je taj prozor otvoren (inače program odmah završi)

Komentar vezan uz prethodni `#include`

Uočimo da smo na prethodnom slajdu imali

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

umjesto, primjerice, `#include <Window.hpp>`. Objašnjenje:

- u svojstvima projekta smo pod *Additional Include Directories* naveli *C:\SFML-2.6.1\include*
- u toj mapi je unutar mape **`SFML`** datoteka **`Window.hpp`**

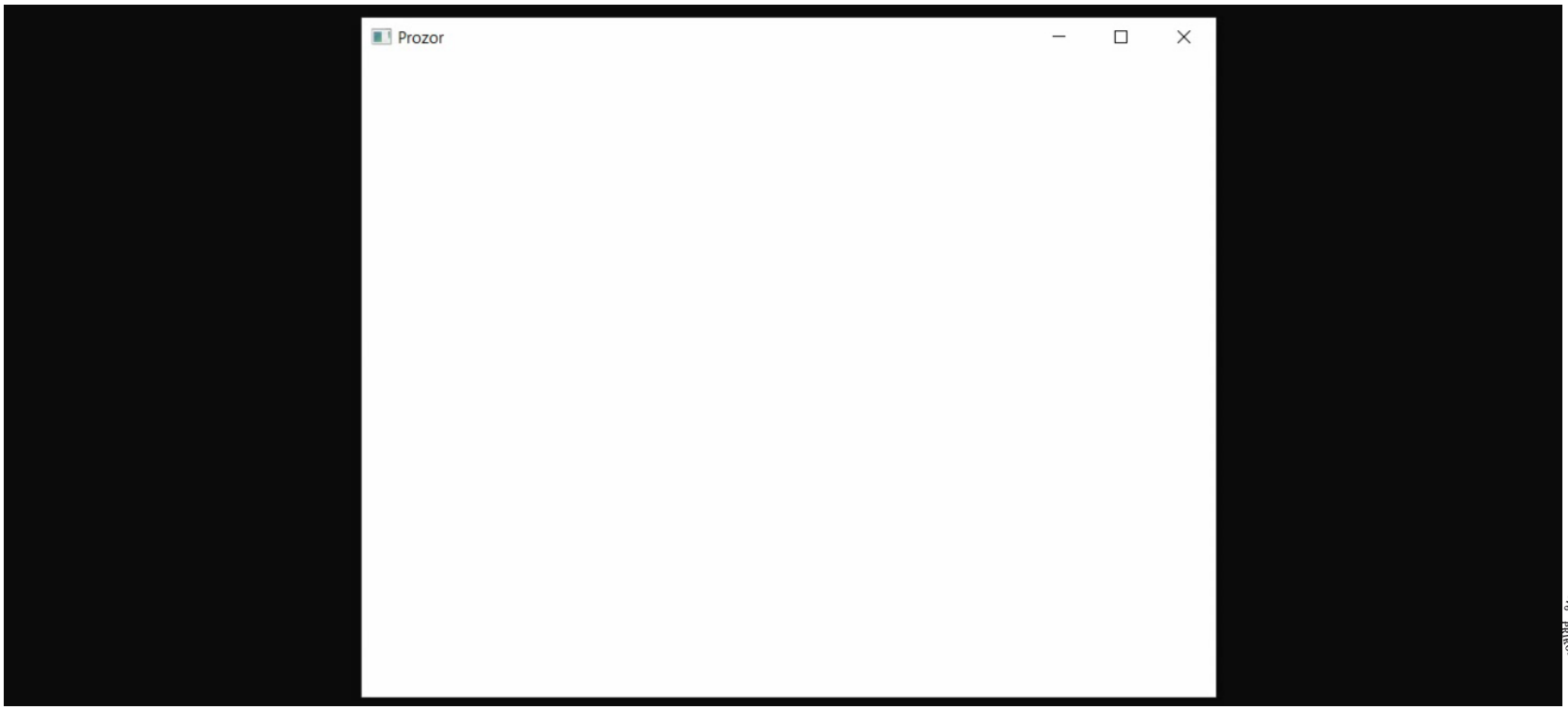
Napomena. Ne možemo samo pod *Additional Include Directories* promijeniti navedeno *C:\SFML-2.6.1\include\SFML* jer u datoteci `Window.hpp` između ostalog piše sljedeće:

```
#include <SFML/System.hpp>
#include <SFML/Window/Clipboard.hpp>
#include <SFML/Window/Context.hpp>
```

`Window` konstruktor

```
sf::Window prozor(sf::VideoMode(800, 600),"Prozor");
```

- prvi argument: *video mode* (klasa `sf::VideoMode`)
- unutarnja veličina prozora (bez obruba i naslovne trake)
- u gornjem primjeru: 800 $\times$ 600 piksela
- drugi argument: naslov prozora



`Window` konstruktor - treći opcionalni argument

- **stil prozora** - bitovna ILI kombinacija `sf::Style` enumeratora:

`sf::Style::None`	bez dekoracija (ne može se kombinirati s ostalima)
`sf::Style::Titlebar`	prozor ima naslovnu traku
`sf::Style::Resize`	prozoru se može mijenjati veličina i ima gumb za maksimiziranje
`sf::Style::Close`	prozor ima gumb za zatvaranje
`sf::Style::Fullscreen`	*fullscreen* način
`sf::Style::Default`	*defaultni* stil (`Titlebar | Resize | Close`)

Window - create funkcija

- za stvaranje/mijenjanje prozora nakon konstrukcije
- ima iste argumente kao i konstruktor

Primjer. Umjesto konstruktora iz prošlog primjera (s istim efektom):

```
sf::Window prozor;  //defaultni konstruktor
prozor.create(sf::VideoMode(800,600),"Prozor");
```

- dobiveni prozor ne može se pomaknuti, zatvoriti, niti mu se može promijeniti veličina

Događaji (*events*)

Dodamo sljedeći kod u glavnu petlju koja osigurava ažuriranje aplikacije dok je prozor otvoren (*main* ili *game loop*):

```
while (prozor.isOpen()) {
    sf::Event d;
    while (prozor.pollEvent(d)) {
        if(d.type == sf::Event::Closed)
            prozor.close();
    }
}
```

- u petlji provjeravamo sve (zato `while`) događaje na čekanju
- funkcija `bool sf::Window::pollEvent(Event &event)` - vraća preko reference (i izbacuje) događaj s početka reda događaja (ako red nije prazan - tada vraća `true`)
- **nije blokirajuća funkcija** - ako je red prazan, vraća `false` (i ne mijenja `event`)

Događaji - `pollEvent` vs. `waitEvent`

`bool sf::Window::waitEvent(Event &event)`

- kao `pollEvent`, ali **blokirajuća funkcija** - čeka dok se ne dogodi neki događaj

Primjer. Što radi sljedeći kod (i što bi bilo drugačije kad bi stavili `pollEvent` umjesto `waitEvent`)?

```
int i = 0, j = 0;
while (prozor.isOpen()) {
    sf::Event d;
    while (prozor.waitEvent(d)) {
        if (d.type == sf::Event::Closed) {
            prozor.close();
        }
        cout << "i = " << i++ << endl;
    }
    cout << "j = " << j++ << endl;
}
```

Događaji: `sf::Event` tip

- **unija**
  - $\Rightarrow$ članovi dijele isti memorijski prostor, pa je samo jedan valjan u danom trenutku - onaj koji odgovara tipu događaja
  - $\Rightarrow$ nikad ne koristiti član događaja koji ne odgovara njegovom tipu
- koristimo samo događaje koje vraćaju `pollEvent` i `waitEvent` funkcije

Primjer. U prethodnom primjeru: **`sf::Event::Closed`**

- predstavlja **zahtjev** korisnika za zatvaranjem prozora

$\Rightarrow$ bez prethodnog `prozor.close()` prozor ostaje otvoren

- prije zatvaranja prozora, možemo spremiti stanje aplikacije ili pitati korisnika što učiniti