SFML - Crtanje

Objektno programiranje - 9. vježbe (2. dio)

Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

17. svibnja 2023. godine

SFML prozor

- objekt klase **`sf::Window`** (SFML koristi **`sf`** *namespace*)

Primjer.

```
#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-2.5.1\include*
- u toj mapi je unutar mape **`SFML`** datoteka **`Window.hpp`**

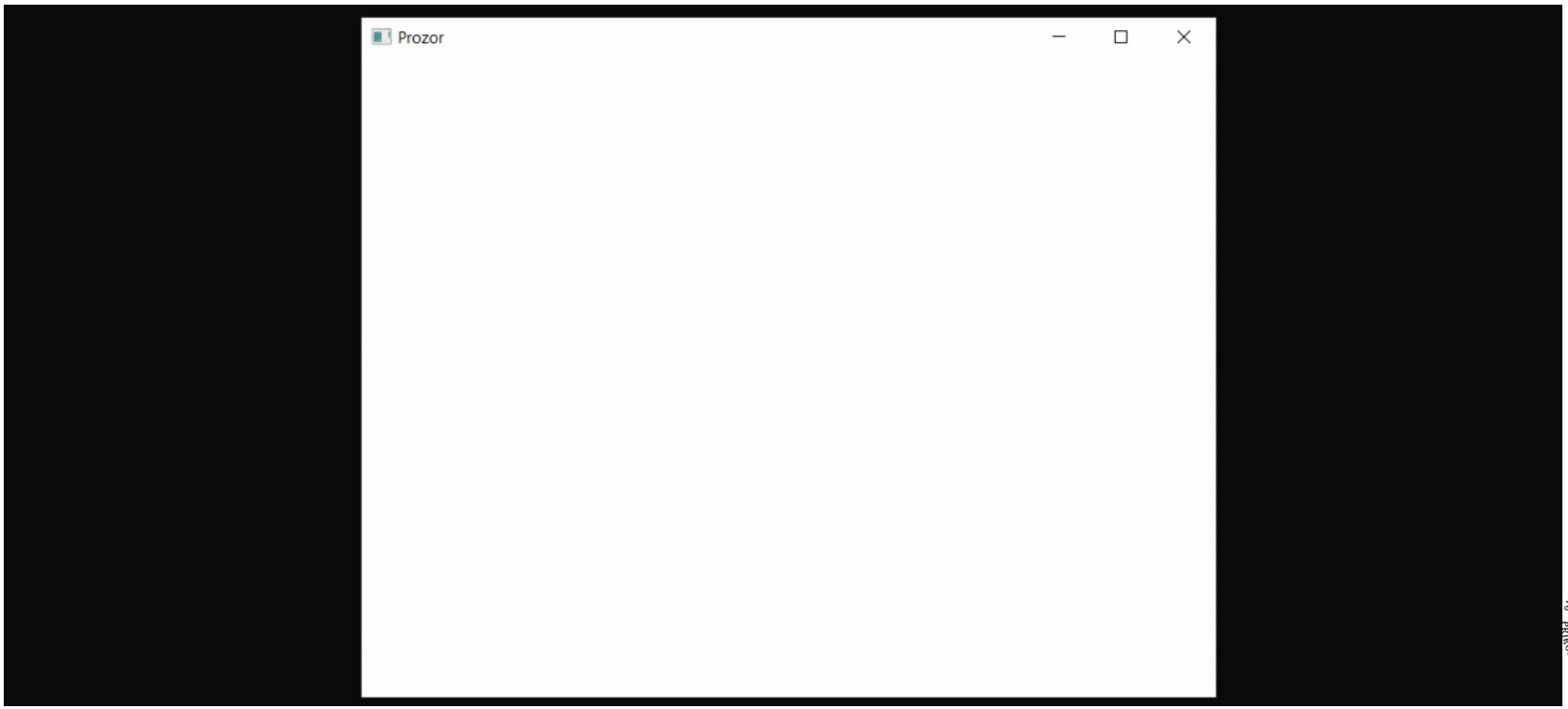
Napomena. Ne možemo samo pod *Additional Include Directories* promijeniti navedeno *C:\SFML-2.5.1\include\SFML* jer u datoteci `Window.hpp` između ostalog piše sljedeće:

```
#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;
while (prozor.isOpen()) {
    sf::Event d;
    while (prozor.waitEvent(d)) {
        if (d.type == sf::Event::Closed) {
            prozor.close();
        }
        cout << i++ << 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

Zadatak

Zadatak. Napisati program koji početno otvara jedan prozor. Kad korisnik odluči zatvoriti neki od otvorenih prozora, otvoriti još jedan prozor. Kad ukupno bude otvoreno 5 prozora, na zahtjev korisnika za zatvaranje bilo kojeg od njih, zatvoriti ih sve (i tako završiti program). U naslovnim trakama prozora treba pisati `"Prozor"` + redni broj tog prozora.

Napomena.

- s obzirom da su prozori izvedeni iz `sf::NonCopyable` nije ih moguće kopirati (jedna posljedica: funkcije umjesto prozora moraju primiti referencu ili pokazivač na njega)

Rješenje

```
sf::Window prozori[5];
size_t brojac = 1;
prozori[0].create(sf::VideoMode(800,600),
          string("Prozor")+to_string(brojac));
while (prozori[0].isOpen()) {
  sf::Event d;
  for(size_t i = 0; i < brojac; ++i)
    while (prozori[i].pollEvent(d)) {
      if(d.type == sf::Event::Closed) {
        if(brojac < 5)
           prozori[brojac-1].create
             (sf::VideoMode(800,600),
              string("Prozor")+to_string(++brojac));
        else for(size_t j = 0; j < brojac; ++j)
             prozori[j].close();
      }
    }
}
```

Ostali događaji

`sf::Event::`

- Resized
- LostFocus
- GainedFocus
- TextEntered
- KeyPressed
- KeyReleased
- MouseWheelScrolled
- MouseButtonPressed
- MouseButtonReleased
- MouseMoved
- MouseEntered
- MouseLeft
- JoystickButtonPressed
- JoystickButtonReleased
- JoystickMoved
- JoystickConnected
- JoystickDisconnected

Više informacija o pojedinom događaju na linku: link
Nekima od njih bavit ćemo se kasnije.

Napomena. `MouseWheelMoved` je od SFML-a 2.3. zamijenjen s `MouseWheelScrolled`.

Promjena postavki prozora

Promjena naslova prozora

```
prozor.setTitle("Novi naslov");
```

Promjena veličine prozora

```
prozor.setSize(sf::Vector2u(200,400));
```

- za manipulaciju dvodimenzionalnim vektorima: parametrizirana klasa `sf::Vector2<T>`
- ima jednostavnu implementaciju: link
- koristimo najčešće specijalizacije (sljedeće preuzeto iz implementacije):

```
typedef Vector2<int>          Vector2i;
typedef Vector2<unsigned int> Vector2u;
typedef Vector2<float>        Vector2f;
```

Primjer upotrebe klase `Vector2f`

```
void ispis(sf::Vector2f &v) {
    cout << "(" << v.x << "," << v.y << ")" << endl;
}

int main() {
    sf::Vector2f v1(16.5f, 24.f);
    v1.x = 18.2f;
    ispis(v1);      //(18.2,24)
    sf::Vector2f v2 = v1 * 2.f;
    ispis(v2);      //(36.4,48)
    sf::Vector2f v3;
    v3 = v1 + v2;
    if(v2 != v3)
        ispis(v3);  //(54.6,72)
    return 0;
}
```

Pozicija prozora

`void setPosition(const Vector2i &position)`

- promjena pozicije prozora

`Vector2i sf::Window::getPosition() const`

- daje poziciju prozora (u pikselima)

Primjer.

```
prozor.setPosition(sf::Vector2i(40,100));
auto p = prozor.getPosition();
cout << "Pozicija prozora:  (" << p.x
     << "," « p.y « ")" << endl;
```

Još o mogućnostima rada s prozorima: link

Crtanje - posebna klasa prozora

`sf::RenderWindow`

- klasa izvedena iz `sf::Window` (sve funkcije naslijeđene)

Primjer. U trenutnom kodu potrebne su minimalne promjene:

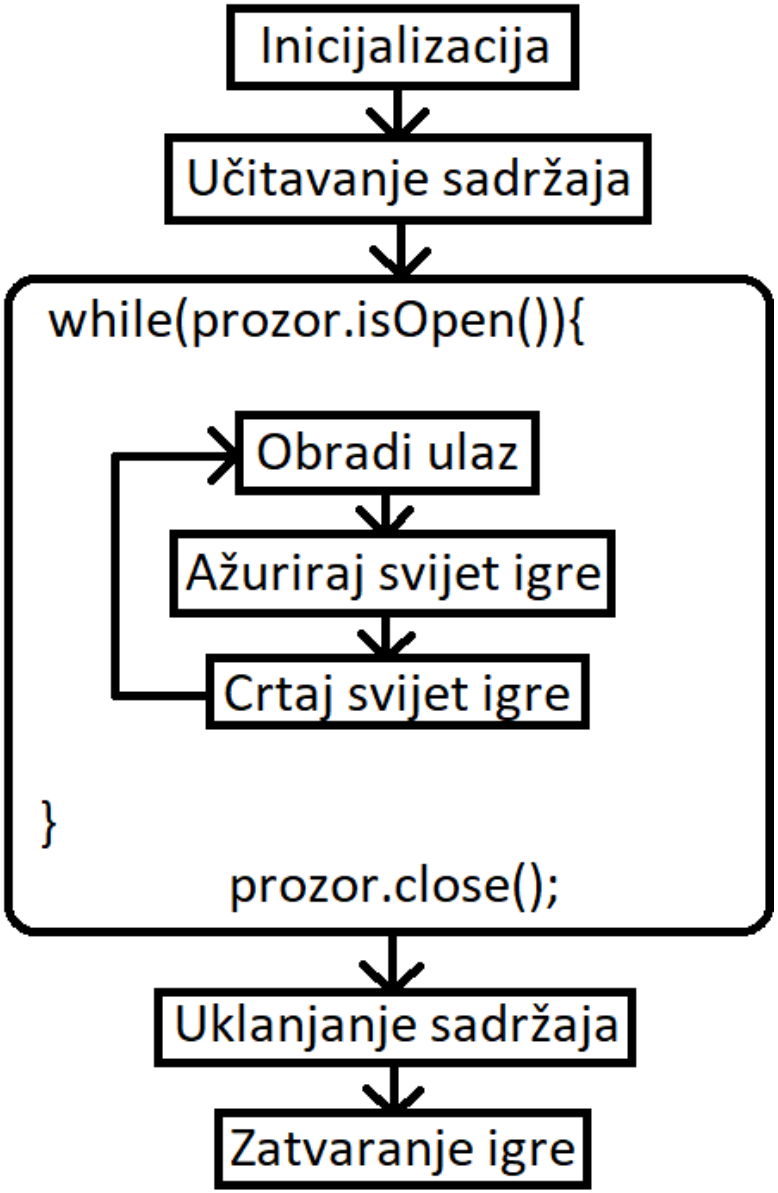
```
#include <SFML/Graphics.hpp>
...
  sf::RenderWindow prozor;
  prozor.create(sf::VideoMode(800,600),"Prozor");
  while (prozor.isOpen()) {
      sf::Event d;
      while (prozor.pollEvent(d)) {
          if (d.type == sf::Event::Closed) {
              prozor.close();
          }
      }
  }
```

Napomena uz prethodni `#include`

- dovoljno je samo uključiti datoteku **`SFML/Graphics.hpp`** (umjesto još i datoteke `SFML/Window.hpp`) jer u toj datoteci između ostalog piše:

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

Dijagram tipične igre



Čisti/crtaj/prikaži ciklus

- ***double-buffering*** metoda - standardna u igrama
- ne crtamo odmah na ekran nego u spremnik
- u prikladnom trenutku kopiramo to na ekran

```
while (prozor.isOpen()) {
    ...
    prozor.clear(sf::Color::Black);
    //prozor.draw(...);
    prozor.display();
}
```

- ne želimo da se crta jedno preko drugoga - prvo očistimo prozor
- `clear` prima enumerirani tip **`sf::Color`** (*default* je crna boja)
- nakon crtanja, prikažemo nacrtano `display` metodom

Boje - klasa `sf::Color`

- za manipuliranje **RGBA** bojama
- *defaultni* konstruktor daje crnu, neprozirnu boju

Primjer.

```
sf::Color crna;
prozor.clear(crna);
```

```
sf::Color::Color(Uint8 red, Uint8 green,
                 Uint8 blue, Uint8 alpha=255)
```

- crvena, zelena, plava i prozirnost komponenta - svaka cijeli broj iz raspona [0,255]

Ima i treći konstruktor: `sf::Color::Color(Uint32 color)`

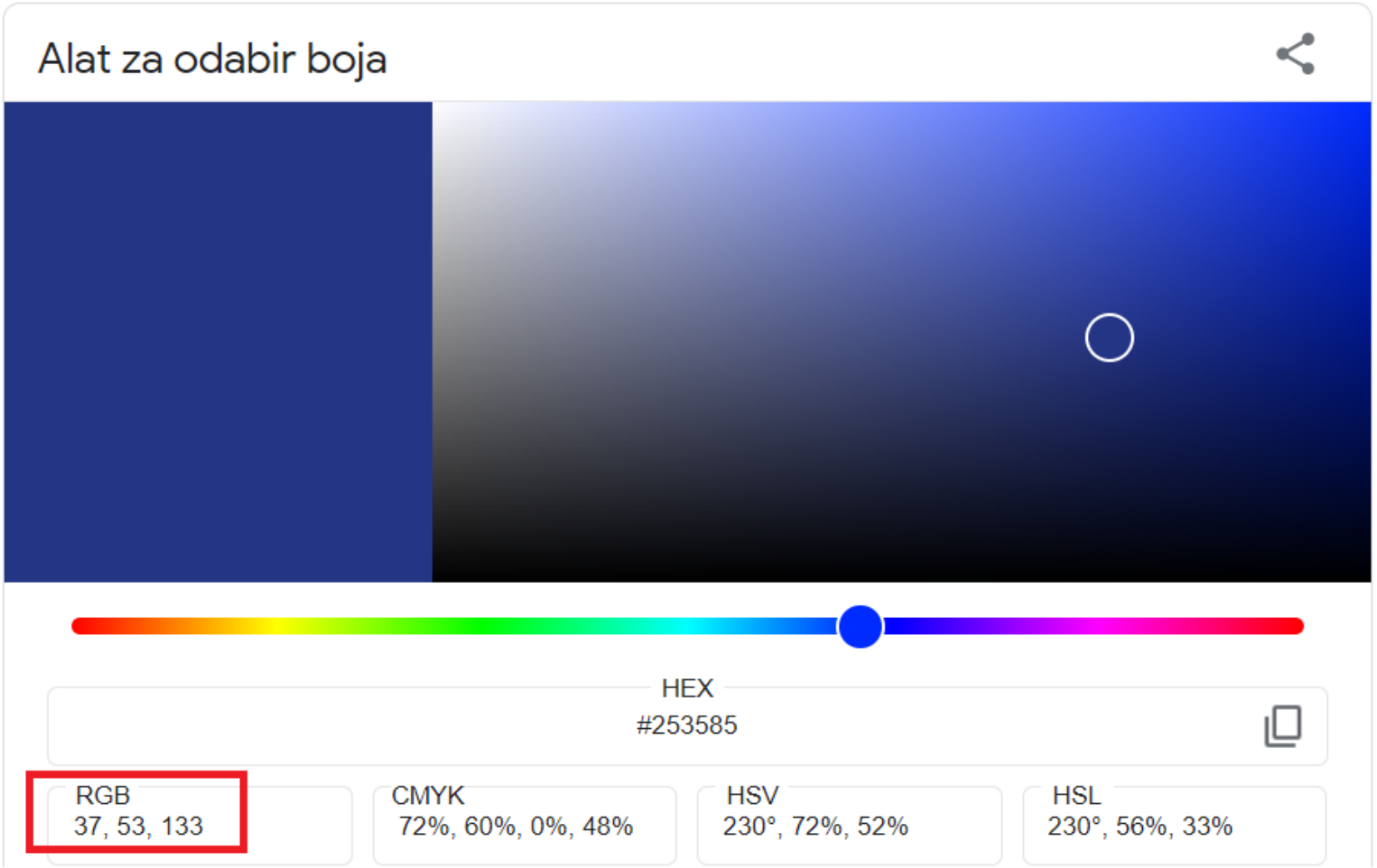
- parametar je 32-bitni nenegativni cijeli broj koji sadrži RGBA komponente (u tom redoslijedu)

Predefinirane boje

- `sf::Color::Black`
- `sf::Color::Blue`
- `sf::Color::Cyan`
- `sf::Color::Green`
- `sf::Color::Magenta`
- `sf::Color::Red`
- `sf::Color::Transparent`
- `sf::Color::White`
- `sf::Color::Yellow`

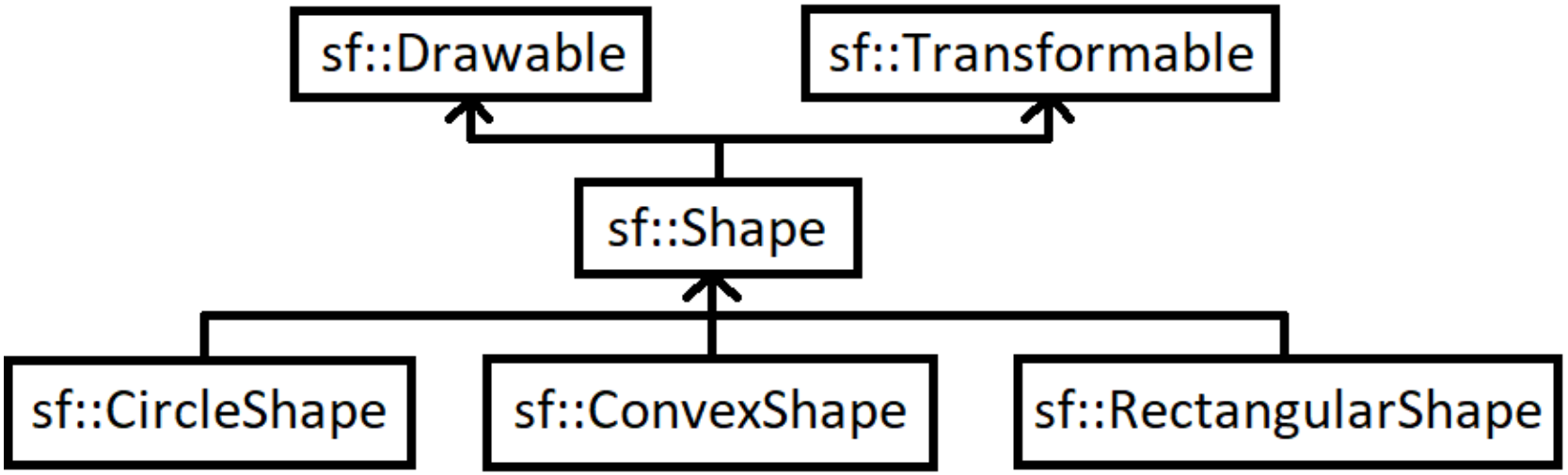
Boje pomoću RGB koda

- guglati *color picker* i odabrati željenu boju
- za RGB 37, 53, 133 $\rightarrow$ `sf::Color boja(37,53,133);`



Crtanje oblika

- svaki oblika ima posebnu klasu
- većina svojstava ista jer imaju istu baznu klasu (**`sf::Shape`**)
- dijagram nasljeđivanja:



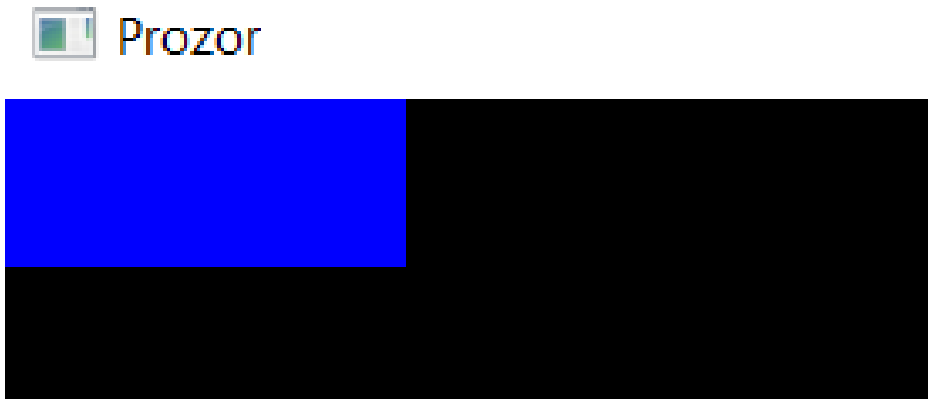
- `sf::Shape` je **apstraktna klasa** ($\Rightarrow$ mora se specijalizirati konkretnim oblikom)
- **`sf::Drawable`** - objekti koji se mogu crtati na ekran
- **`sf::Transformable`** - pruža funkcionalnosti pomicanja, skaliranja i rotiranja objekta

Klase oblika: pravokutnik

- klasa **`sf::RectangleShape`**
- konstruktor prima njegove dimenzije
- samo `setSize` je specifična za tu klasu

Primjer. Plavi pravokutnik dimenzija 120×50:

```
prozor.clear(sf::Color::Black);
sf::RectangleShape p(sf::Vector2f(120.f,50.f));
p.setFillColor(sf::Color::Blue);
prozor.draw(p);
prozor.display();
```



Klase oblika: pravokutnik - promjena veličine

- funkcija specifična za klasu `sf::RectangleShape` je `setSize`
- možemo nakon konstruktora mijenjati veličinu pravokutnika
- po *defaultu* je 0×0 - konstruktor ima ovakvu deklaraciju:

```
RectangleShape(const Vector2f &size=Vector2f(0,0))
```

Primjer. Umjesto prethodnog konstruktora može ovako:

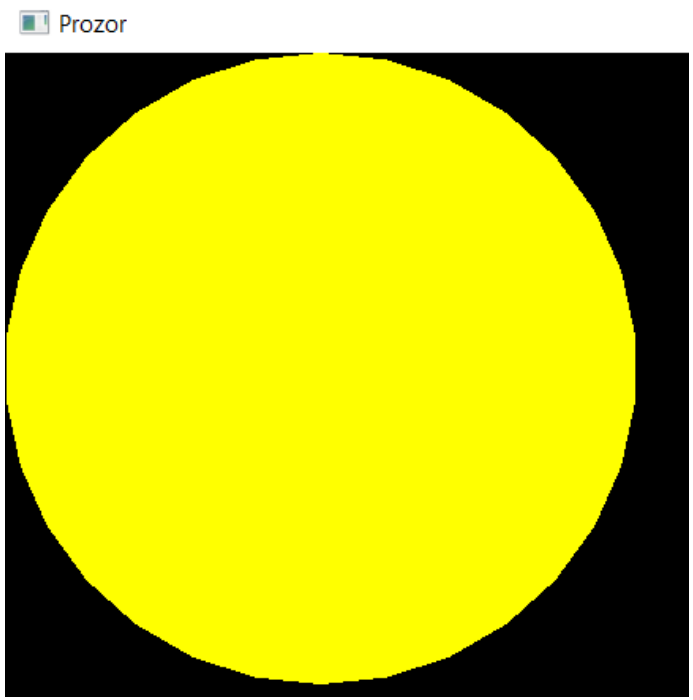
```
sf::RectangleShape p;
p.setSize(sf::Vector2f(120.f, 50.f));
```

Klase oblika: krug

- klasa **`sf::CircleShape`**
- konstrukor prima veličinu radijusa (polumjera)

Primjer.

```
prozor.clear(sf::Color::Black);
sf::CircleShape p(200.f);
p.setFillColor(sf::Color::Yellow);
prozor.draw(p);
prozor.display();
```



Klase oblika: krug - specifične funkcije i konstruktor

- konstruktor ima sljedeću deklaraciju:

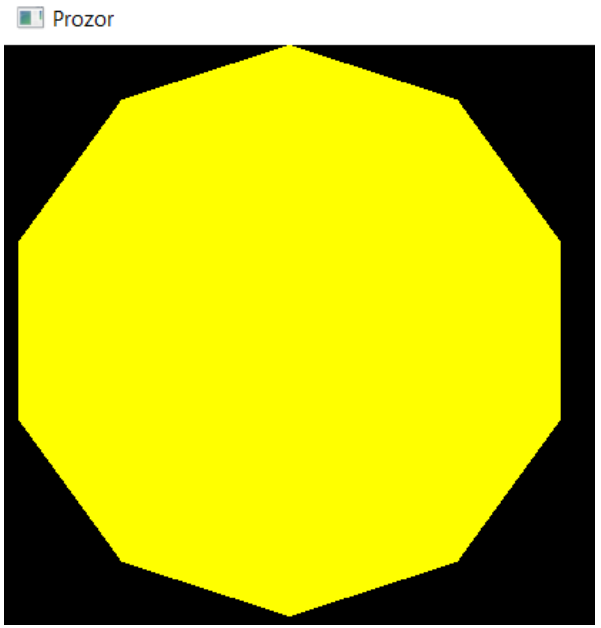
```
CircleShape(float radius=0,std::size_t pointCount=30)
```

- drugi (opcionalan) argument je broj stranica („kvaliteta" kruga; ne crta se savršen krug nego aproksimacija mnogokutom)
- za naknadno postavljanje radijus i broja stranica: funkcije `setRadius` i `setPointCount`

Primjer.

```
sf::CircleShape p;
p.setRadius(200.f);
p.setPointCount(10);
```

- uočite: dobili smo pravilni deseterokut

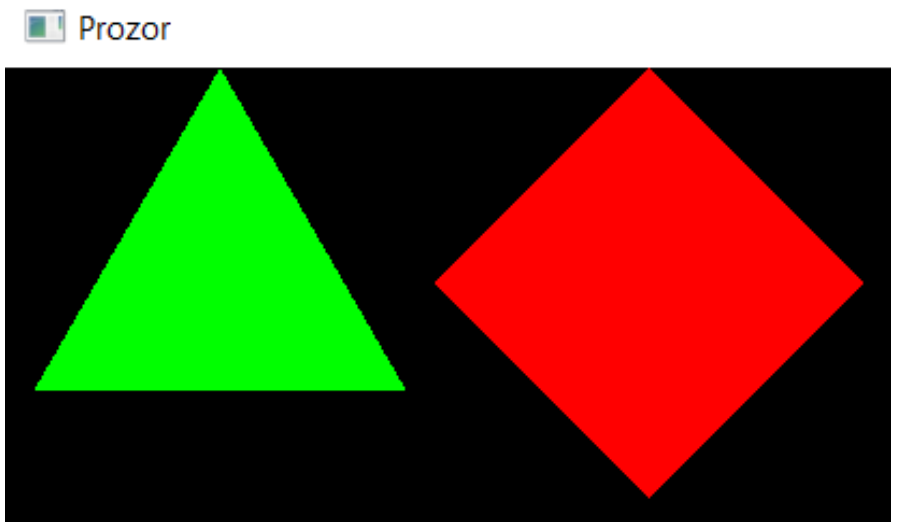


Pravilni mnogokuti

- ne postoji klasa za pravilne mnogokute - mogu se dobiti kao u prethodnom primjeru prilagodbom broja stranica

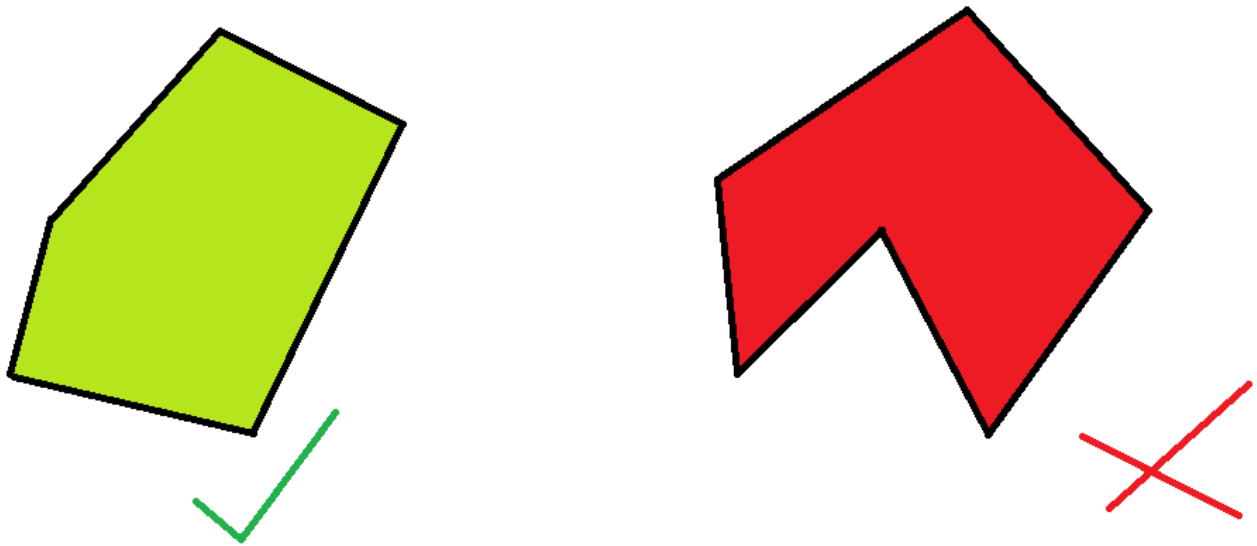
Primjer. Jednakostranični trokut i kvadrat:

```
sf::CircleShape t(100.f,3);
t.setFillColor(sf::Color::Green);
prozor.draw(t);
sf::CircleShape k(100.f,4);
k.setFillColor(sf::Color::Red);
k.move(200.f, 0.f);
prozor.draw(k);
```



Konveksni oblici

- klasa `sf::ConvexShape` - za crtanje konveksnih mnogokuta
- važno: **SFML ne može crtati konkavne mnogokute**
  $\Rightarrow$ treba ih crtati kao više konveksnih mnogokuta



- slika desno: konveksni mnogokut, slika lijevo: nekonveksni (konkavni) mnogokut

Pitanje. Kako biste nacrtali gornji desni mnogokut?

Konveksni oblici - crtanje

`sf::ConvexShape::ConvexShape(std::size_t pointCount=0)`

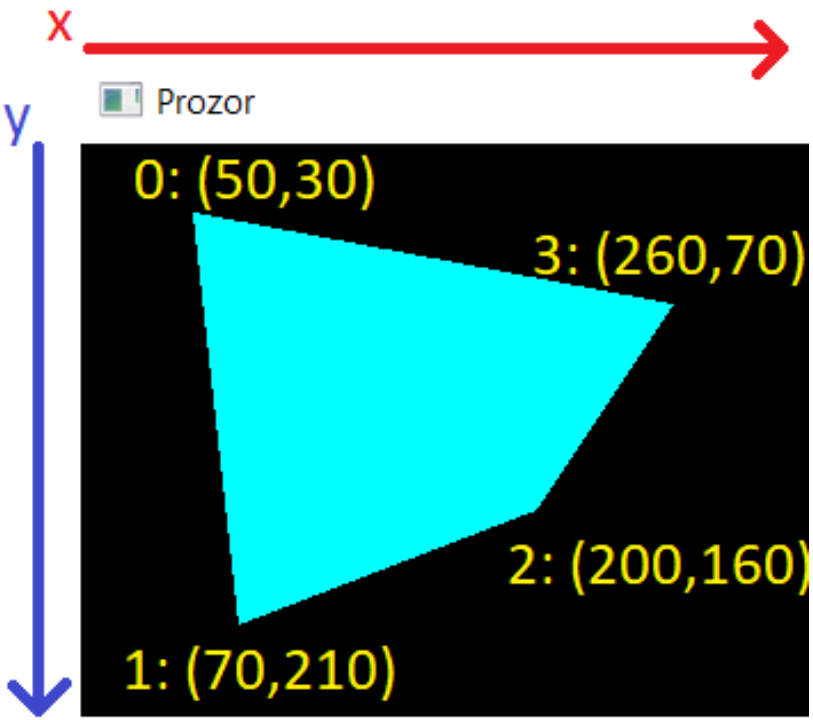
- konstruktor prima broj točaka mnogokuta (za valjani oblik treba biti $\geqslant 2$ točaka)
- funkcija `getPointCount()` vraća postavljeni broj točaka
- točke se nakon toga navode **po redu** (u smjeru kazaljke na satu ili u smjeru obratnom kazaljci na satu)
- **indeks mora biti u rasponu** $[0, \texttt{getPointCount()} - 1]$

Konveksni oblici - primjer

```
prozor.clear(sf::Color::Black);
sf::ConvexShape p(4);
p.setFillColor(sf::Color::Cyan);
p.setPoint(0,sf::Vector2f(50,30));
p.setPoint(1,sf::Vector2f(70,210));
p.setPoint(2,sf::Vector2f(200,160));
p.setPoint(3,sf::Vector2f(260,70));
prozor.draw(p);
prozor.display();
```

Napomena. Umjesto gornjeg konstruktora može ovako:

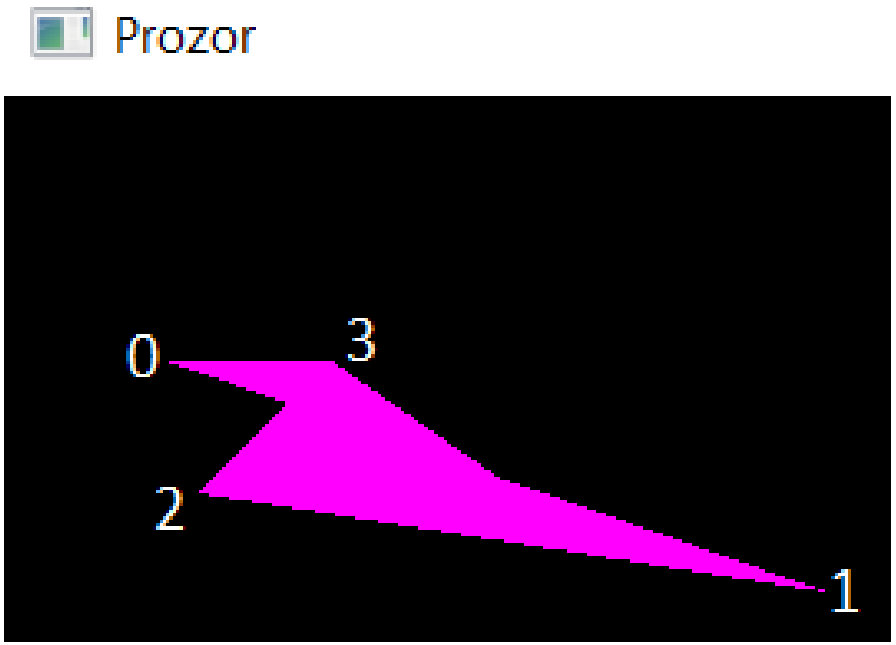
```
sf::ConvexShape p;
p.setPointCount(4);
```



Konveksni oblici - još jedan primjer

Pitanje. Što nije u redu u sljedećem primjeru?

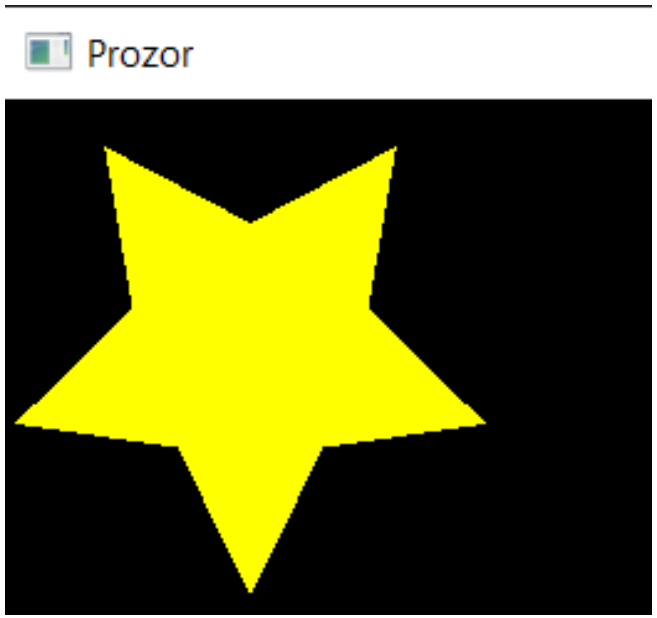
```
sf::ConvexShape p(4);
p.setFillColor(sf::Color::Magenta);
p.setPoint(0,sf::Vector2f(50,80));
p.setPoint(1,sf::Vector2f(250,150));
p.setPoint(2,sf::Vector2f(60,120));
p.setPoint(3,sf::Vector2f(100,80));
```



Triangle fans

- zapravo nije nužno crtanje konveksnih mnogokuta
- jedini zahtjev: ako crtamo linije iz težišta do svih navedenih točaka, one moraju biti nacrtane u istom redoslijedu
- konveksni oblici se zapravo crtaju korišenjem *triangle fans*
- zbog toga možemo, primjerice, nacrtati zvijezdu

Zadatak. Nacrtati zvijezdu sličnu onoj s donje slike.



Jedno od mogućih rješenja

```
sf::ConvexShape s(10);
s.setFillColor(sf::Color::Yellow);
float pi = float(atan(1)) * 4;
for (size_t k = 0; k <= 4; ++k) {
    float si = 2 * pi * k / 5 + pi / 2,
          co = 2 * pi * k / 5 + pi / 2;
    s.setPoint(2*k, sf::Vector2f
          (100*cos(co)+100,100*sin(si)+100));
    s.setPoint(2*k+1, sf::Vector2f
        (50*cos(co+pi/5)+100,50*sin(si+pi/5)+100));
}
prozor.draw(s);
```

- korištena literatura: link
- rješenje zahtijeva `#include<cmath>`