SFML - Upotreba slike i strukturiranje koda

Objektno programiranje - 5. vježbe

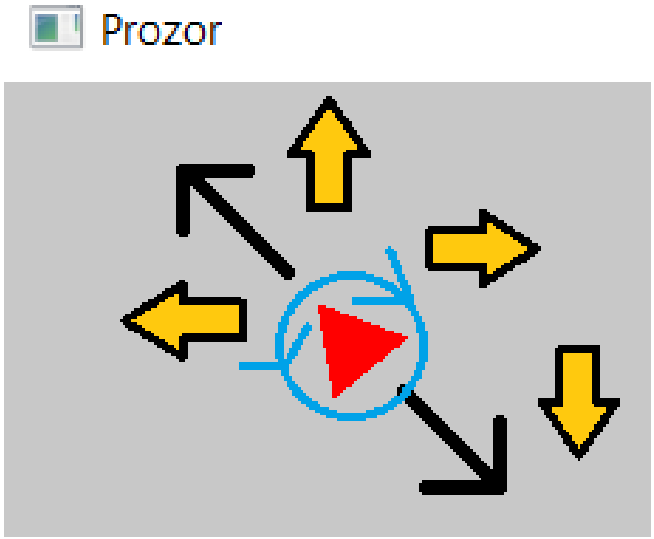
dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

9. travnja 2025. godine

Zadatak riješen na prethodnim vježbama

Zadatak. Nacrtati crveni jednakostranični trokut (radijus opisane kružnice je 20 piksela) koji se početno nalazi na sredini 640×480 sivog prozora te koji se može pomicati pritiskanjem strelica na tipkovnici (kao na slici ispod: strelica gore je naprijed, strelica dolje je natrag, strelica lijevo je rotacija u smjeru obrnutom od smjera kazaljke na satu, a strelica desno je rotacija u smjeru kazaljke na satu).

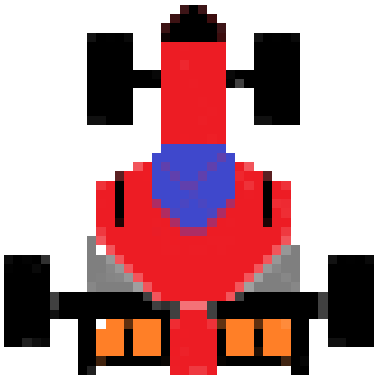


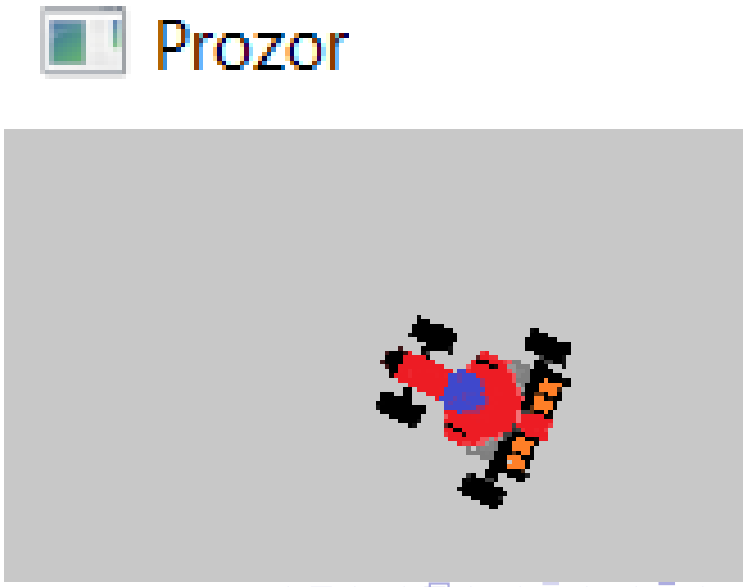
Rješenje se može preuzeti na web-stranici kolegija (pod „Rješenje zadatka s pomicanjem trokuta” - jedna datoteka `formula.cpp`).

Upotreba slike

Zadatak. Promijeniti trokut iz prošlog zadatka u formulu. Datoteka "formula.png" može se preuzeti na web-stranici kolegija.

- datoteka je nastala u programu *Paint*, te joj je dodatnim programom pozadina stavljena na transparentnu
- dimenzije slike su 40×40 piksela (što odgovara omeđujućem okviru trokuta iz prethodnog zadatka)

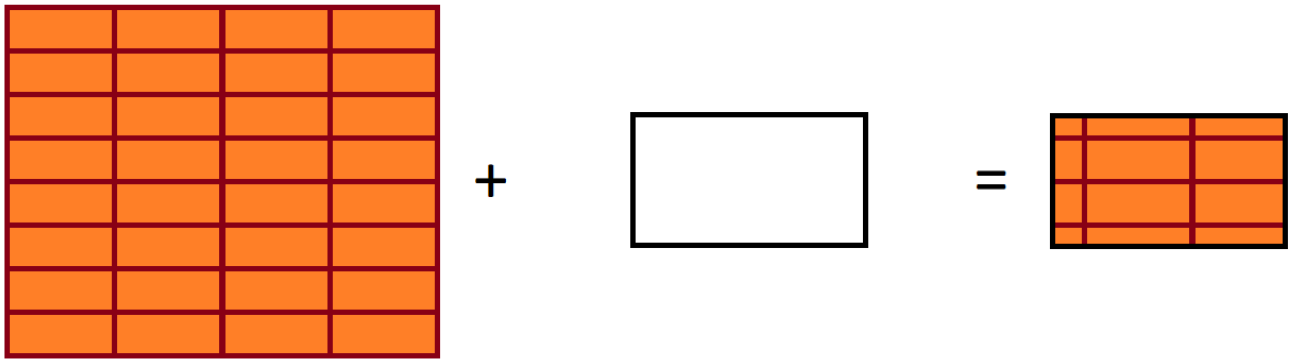




Spriteovi i teksture

- **tekstura** = slika (nazivamo ju tekstura jer se prelikava na 2D objekt)
- ***sprite*** = pravokutnik s teksturom

Ilustracija:



Zašto naziv *sprite* (duh):

- *sprite* nije dio pozadinske slike, nego „lebdi" nad tim

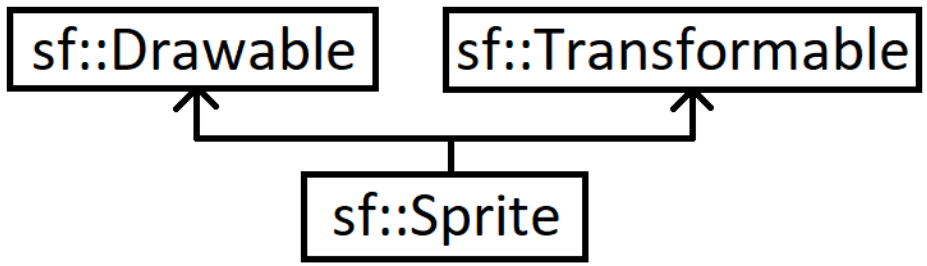
Klase za *sprite* i za teksturu

Tekstura

- slika koja se nalazi na grafičkoj kartici i može se koristiti za crtanje
- `sf::Texture` - sprema piksele koji se mogu crtati (npr. pomoću *spritea*)
- no, to nije objekt koji npr. pomičemo tijekom igre

Sprite

- slika koja se može koristiti kao 2D objekt, koji ima koordinate, boju i može se pomicati, uništiti ili stvoriti tijekom igre
- klasa `sf::Sprite` - dijagram nasljeđivanja:



Tekstura

- s obzirom da je jedina uloga teksture učitavanje i preslikavanje na grafički objekt, većina funkcija koje ima su za učitavanje i ažuriranje teksture

U kodu na početku `main` funkcije dodati:

```
sf::Texture texture;
texture.loadFromFile("formula.png");
```

Napomena. Datoteka `"formula.png"` mora se nalaziti **u istoj mapi kao i projekt** (tamo gdje se nalazi `.vcxproj` datoteka). U protivnom se na standardni izlaz ispiše poruka:

```
Failed to load image "formula.png".
Reason: Unable to open file
```

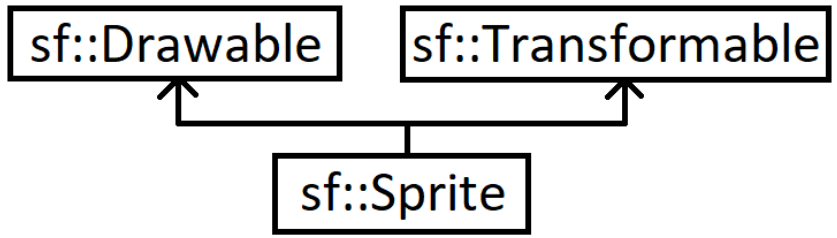
Upotreba klase `sf::Sprite`

- u kodu umjesto `sf::CircleShape t...` i `t.setFillC...`:

```
sf::Sprite t;
t.setTexture(texture);
```

Uočimo da ostalo u kodu ne trebamo mijenjati:

- crtanje je i dalje ovako: `prozor.draw(t);`
- podsjetnik na dijagram nasljeđivanja:



- kako klasa `sf::Sprite` nasljeđuje klasu `sf::Transformable`, onda također ima metode za transformacije:
  - `setOrigin`
  - `setPosition`
  - `move`
  - `setRotation`

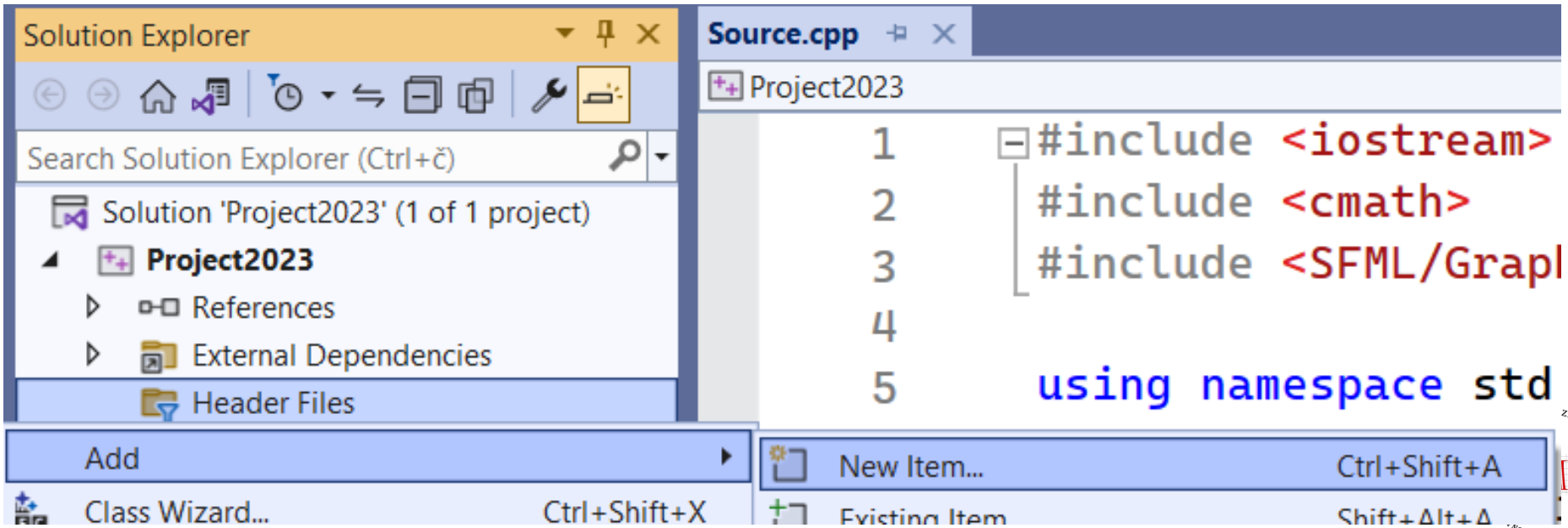
Zašto nam treba strukturiranje

- iz prethodnog koda mogli bi napraviti neku igru - možemo dodati:
  - provjeru kolizije formule s rubom ekrana,
  - stazu za utrke,
  - protivničke formule,
  - prepreke na stazi,
  - bodovanje,
  - ...
- no, porastom količine koda on postaje **težak za održavanje** (tzv. *spaghetti* kod)

Klasa `Prozor`

- svaka igra treba **prozor**
- mora se znati stvoriti, uništiti, obrađivati događaje koji se pojave (posebno zahtjeve za zatvaranje tog prozora), očistiti ekran, ažurirati što je nacrtano, pratiti je li ekran u *full-screen* načinu

Implementirat ćemo takvu klasu - napraviti unutar našeg projekta **`Prozor.h`** datoteku



Koje podatke želimo imati o prozoru:

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

class Prozor {
    private:
        sf::RenderWindow prozor;
        sf::Vector2u velicina;
        std::string naslov;
        bool gotov;
        bool cijeliZaslon;
};
```

- trebamo neki `sf::RenderWindow` za crtanje
- želimo imati podatke o veličini (širina i visina - dva *unsigned* broja držimo kao jedan vektor) i naslovu prozora (string)
- dva `bool` podatka - je li pripadni prozor `prozor` otvoren/zatvoren i je li u *full screen* načinu

Pomoćne funkcije za stvaranje/uništavanje prozora

- bit će korištene u, primjerice, konstruktoru i destruktoru - ali također i pri promjeni u (iz) *full screen* način(a)

```
class Prozor {
    private:
        void Stvori();
        void Unisti();
        ...
};

void Prozor::Stvori() {
    auto stil = (cijeliZaslon ? sf::Style::Fullscreen
                              : sf::Style::Default);
    prozor.create(sf::VideoMode(velicina.x,
                  velicina.y, 32), naslov, stil);
} //uokvireno je i default (dubina po pikselu)

void Prozor::Unisti() {
    prozor.close();
}
```

Destruktor

- konstruktori: jedan *defaultni* i jedan koji prima naslov i veličinu
- zbog količine privatnih podataka, pomoćna funkcija `Postavi`
- podsjetnik: korisnik ne mora znati za neke funkcije (npr. koje se bave detaljima implementacije) - zato je `Postavi()` privatna

```
class Prozor {
  public:
    Prozor();
    Prozor(const std::string&, const sf::Vector2u&);
    ~Prozor();
  private:
    void Postavi(const std::string&,
                 const sf::Vector2u&);
    ...
};

Prozor::~Prozor() {
    Unisti();
}
```

Konstruktori i pomoćna funkcija `Postavi`

```
Prozor::Prozor() {
    Postavi("Prozor", sf::Vector2u(640, 480));
}

Prozor::Prozor(const std::string& n,
               const sf::Vector2u& v){
    Postavi(n, v);
}

void Prozor::Postavi(const std::string& n,
                     const sf::Vector2u& v) {
    naslov = n;
    velicina = v;
    cijeliZaslon = false;
    gotov = false;
    Stvori();
}
```

Funkcija za prebacivanje na cijeli zaslon

- u tom slučaju potrebno je ponovo otvoriti prozor s novim postavkama

```
class Prozor {
    public:
        ...
        void prebaciNaCijeli();
        ...
};

void Prozor::prebaciNaCijeli() {
    cijeliZaslon = !cijeliZaslon;
    Unisti();
    Stvori();
}
```

Funkcije za čišćenje i prikaz prozora

```
class Prozor {
    public:
        ...
        void ocisti();
        void prikazi();
        ...
};

void Prozor::ocisti() {
    prozor.clear(sf::Color(200,200,200,255));
}

void Prozor::prikazi() {
    prozor.display();
}
```

Funkcija za ažuriranje (obrada događaja)

```
class Prozor {
  public:
    void update();
    ...
};

void Prozor::update() {
  sf::Event event;
  while (prozor.pollEvent(event)) {
    if (event.type == sf::Event::Closed) {
      gotov = true;
    }
    else if (event.type == sf::Event::KeyPressed &&
      event.key.code == sf::Keyboard::F5) {
        prebaciNaCijeli();
    }
  }
}
```

Funkcije za dobivanje informacija o prozoru

```
class Prozor {
    public:
        ...
        bool jelGotov() {
            return gotov;
        }
        bool jelCijeli() {
            return cijeliZaslon;
        }
        sf::Vector2u dohvatiVelicinu() {
            return velicina;
        }
        ...
};
```

- korisnik može pozvati `jelGotov()` kako bi saznao koja je vrijednost varijable `gotov`, no kako mu nismo dali pristup toj varijabli (privatna je!) on joj ne može izravno mijenjati vrijednost!

Funkcija za crtanje (onog što se može crtati)

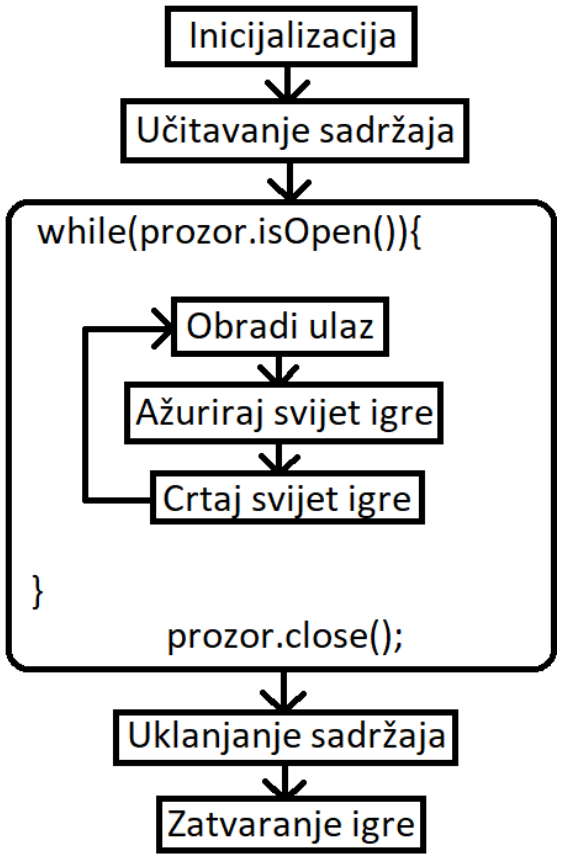
```
class Prozor {
    public:
        ...
        void crtaj(sf::Drawable&);
        ...
};

void Prozor::crtaj(sf::Drawable& d) {
        prozor.draw(d);
}
```

- ovdje je referenca (`&`) obavezna (jer `sf::Drawable` je apstraktna klasa!)

Klasa `Igra`

- napraviti novu datoteku **`Igra.h`** u projektu
- prema slici lijevo (iz jedne od prethodnih prezentacija), desno je prikazana `main` funkcija kakvu bismo htjeli




```
Igra igra;
while(!igra.gotovo()){
    igra.obradiUlaz();
    igra.update();
    igra.renderiraj();
}
```

Klasa `Igra`

- jedna od mogućih implementacija: držimo instancu prozora

```
#include <SFML/Graphics.hpp>
#include <cmath>
#include "Prozor.h"

class Igra {
    public:
        Igra();
        ~Igra();
        bool gotovo() {
            return p.jelGotov();
        }
    private:
        Prozor p;
};

Igra::Igra() : p("Utrka", sf::Vector2u(640,480)){}
Igra::~Igra() {}
```

Što sve trebamo pamtiti o formuli

```
class Igra {
    ...
    private:
        ...
        sf::Texture tekstura;
        sf::Sprite sprite;
        sf::Vector2f pomak, pocetni_pomak;
        float kut;
        float smjer; //naprijed 1, stani 0, nazad -1
};
```

Inicijalizacija potrebnih dijelova u konstruktoru

```
Igra::Igra() :  p("Utrka", sf::Vector2u(640,480)) {
   tekstura.loadFromFile("formula.png");
   sprite.setTexture(tekstura);
   sprite.setOrigin(sprite.getLocalBounds().width/2,
                    sprite.getLocalBounds().height/2);
   sf::Vector2u velp = p.dohvatiVelicinu();
   sprite.setPosition(velp.x / 2.f, velp.y / 2.f);
   pomak = pocetni_pomak = sf::Vector2f(0.f,-0.1f);
   kut = 0;
   smjer = 0;
}
```

Funkcija za obradu ulaza

```
class Igra {
    public:
        ...
        void obradiUlaz();
    private:
        void updejtPomaka();
        ...
};

void Igra::updejtPomaka() {
    float rad = kut / 180 * 3.14f;
    pomak.x = cos(rad) * pocetni_pomak.x
              - sin(rad) * pocetni_pomak.y;
    pomak.y = sin(rad) * pocetni_pomak.x
              + cos(rad) * pocetni_pomak.y;
}
```

Funkcija za obradu ulaza (nastavak)

```
void Igra::obradiUlaz() {
  smjer = 0;
  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
      smjer = 1;
  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
      smjer = -1;
  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left)){
      kut -= 0.1f;
      updejtPomaka();
  }
  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
      kut += 0.1f;
      updejtPomaka();
  }
}
```

Funkcija za ažuriranje

```
class Igra {
    public:
        ...
        void update(); // koristi pomakniFormulu()
    private:
        void pomakniFormulu();
        ...
};

void Igra::update() {
    p.update();
    pomakniFormulu();
}

void Igra::pomakniFormulu() {
    sprite.setRotation(kut);
    sprite.move(pomak * smjer);
}
```

Funkcija za renderiranje

```
class Igra {
    public:
        ...
        void renderiraj();
        ...
};

void Igra::renderiraj() {
    p.ocisti();
    p.crtaj(sprite);
    p.prikazi();
}
```

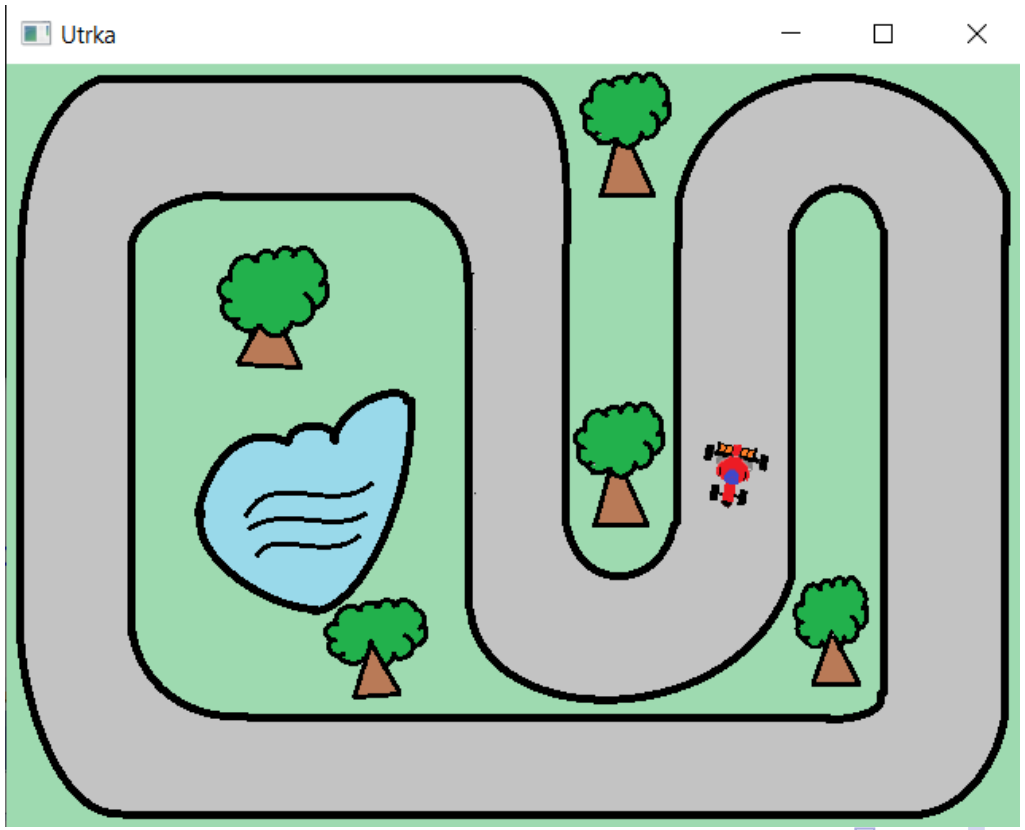
Glavni program (funkcija `main`)

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include "Prozor.h"
#include "Igra.h"
using namespace std;

int main() {
    Igra igra;
    while(!igra.gotovo()){
        igra.obradiUlaz();
        igra.update();
        igra.renderiraj();
    }
    return 0;
}
```

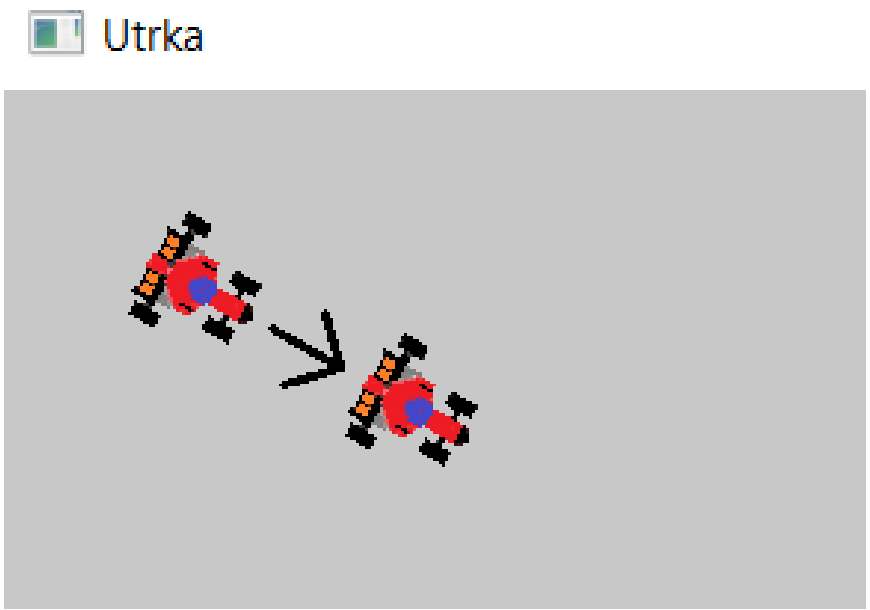
Zadatak

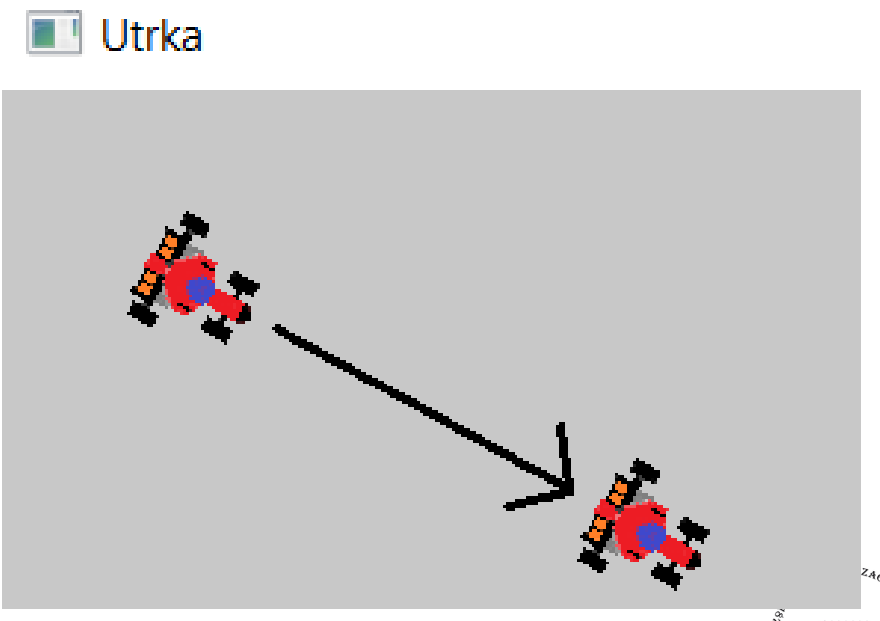
Zadatak. Dodati u pozadinu mapu (npr. može se iskoristiti datoteka `mapa.png` koja se može preuzeti na web-stranici kolegija). Dodati provjeru kolizije između formule i ruba prozora (tako da formula ne može napustiti područje prozora).



Problemi s vremenom izvršavanja

- problem: brzina kojom se *sprite* kreće ovisi o tome koliko je naše računalo opterećeno u tom trenutku
- „brže” skopovlje (*hardware*) $\Rightarrow$ više iteracije glavne petlje
- „sporije” skopovlje $\Rightarrow$ manje iteracija glavne petlje
- rješenje: možemo koristiti SFML-ovo upravljanje vremenom





Koliko formula prođe u 1 sekundi na sporijem i bržem računalu

Frame-rate

- sličice/okviri po sekundi (kratica **FPS**)
- klasa `sf::RenderWindow` nasljeđuje klasu `sf::Window` pa onda i sljedeću funkciju:

```
void sf::Window::setFramerateLimit(unsigned int limit)
```

- `limit` je maksimalni broj prikazanih okvira u sekundi (ili 0 kako bi onemogućili ograničenje)
- ako se koristi ograničenje, prozor koristi **kašnjenje nakon svakog poziva funkcije `display()`**
- tako osigurava da trenutni okvir traje dovoljno dugo za ispunjenje zahtjeva
- SFML zapravo tu koristi `sf::sleep` čija preciznost ovisi o OS-u $\Rightarrow$ može dovesti do nepreciznog ponašanja (npr. 62 FPS-a umjesto traženih 60)

Frame-rate

- iako je prethodna preciznost često dovoljno, to ne rješava problem sporijih računala, samo bržih (daje samo gornju ogradu na FPS!)
- zbog toga ćemo kasnije proučiti klase `sf::Time` i `sf::Clock`

Zadatak. Pogledati kako se prethodni program ponaša pri zadavanju različitih ograničenja.

```
void Prozor::Stvori() {
    ...
    prozor.setFramerateLimit(300);
}
```