

SFML - Događaji i strukturiranje koda

Objektno programiranje - 10. vježbe (2. dio)

Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

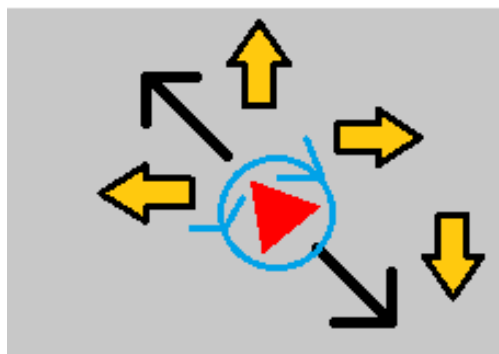
24. svibnja 2023. godine



Zadatak

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).

Prozor



sf::Event klasa

- ▶ **unija** ⇒ samo jedan njegov član valjan (onaj koji odgovara tipu događaja)
- ▶ samo funkcije `pollEvent` i `waitEvent` daju valjane događaje

Tipična petlja za događaje:

```
sf::Event event;
while (prozor.pollEvent(event)) {
    switch (event.type) {
        case sf::Event::Closed:
            prozor.close();
            break;
        case sf::Event::KeyPressed:
            ...
            break;
        default: //ako nas ne zanimaju ostali
            break;
    }
}
```



Događaj promjene veličine prozora

- ▶ pri promjeni veličine prozora (ili korisnik ručno ili u programu pozivom `prozor.setSize(...)`)
- ▶ omogućuje promjenu postavki renderiranja nakon događaja

Primjer. Ispis dimenzija prozora pri promjeni veličine prozora:

```
while (prozor.isOpen()) {
    sf::Event event;
    while (prozor.pollEvent(event)) {
        switch (event.type) {
            case sf::Event::Closed:
                prozor.close();
                break;
            case sf::Event::Resized:
                cout << "(" << prozor.getSize().x << ", "
                    << prozor.getSize().y << ")" << endl;
                break;
        }
    }
}
```



Događaji dobivanja i gubljenja fokusa

- ▶ promjena fokusa događa se kad promijeni trenutno aktivni prozor
- ▶ prozor koji nije u fokusu ne dobiva ulaz s tipkovnice
- ▶ primjer upotrebe: pauziranje igre kad prozor nije u fokusu

Primjer.

```
...
case sf::Event::GainedFocus:
    cout << "Dobio fokus!" << endl;
    break;
case sf::Event::LostFocus:
    cout << "Izgubio fokus!" << endl;
    break;
...
```



Ulazak miša u prozor i izlazak miša iz prozora

- ▶ **sf::Event::MouseEntered** - prilikom ulaska pokazivača miša u prozor
- ▶ **sf::Event::MouseLeft** - prilikom izlaska pokazivača miša iz prozor

Primjer.

```
...
case sf::Event::MouseEntered:
    cout << "Usao u prozor!" << endl;
    break;
case sf::Event::MouseLeft:
    cout << "Napustio prozor!" << endl;
    break;
...
```



Pomak miša

- ▶ **sf::Event::MouseMove** - pri pomicanju miša **unutar prozora** (ne računa se naslovna traka niti rub)
- ▶ radi čak i ako prozor nije u fokusu
- ▶ dobivanje koordinata pokazivača miša (unutar prozora!):
(`mouseMove.x`, `mouseMove.y`)

Primjer.

```
...
case sf::Event::MouseMove:
    cout << "(" << event.mouseMove.x << ", "
        << event.mouseMove.y << ")" << endl;
    break;
...
```



Pritisak i otpuštanje tipke miša

- ▶ **sf::Event::MouseButtonPressed** - pritisak tipke miša
- ▶ **sf::Event::MouseButtonReleased** - otpuštanje tipke miša
- ▶ koordinate pokazivača dobivamo pomoću **mouseButton**
- ▶ tipke miša koje SFML podržava: **Left** (lijevo), **Right** (desno), **Middle** (kotačić), **XButton1**, **XButton2** (tipke sa strane)

Primjer.

```
...
case sf::Event::MouseButtonPressed:
    if(event.mouseButton.button == sf::Mouse::Left)
        cout << "Lijevi klik na ("
            << event.mouseButton.x << ", "
            << event.mouseButton.y << ")" << endl;
    break;
...
```



Napomena. Ostali događaji (poput `MouseWheelScrolled`): [link](#)

Pritisak i otpuštanje tipke na tipkovnici

- ▶ događaji tipa **KeyPressed** i **KeyReleased**
- ▶ to koristimo za **reakciju** na pritisak tipke (npr. pomicanje lika u igri), a ne `TextEntered` događaje (za unos teksta - njih ćemo raditi kasnije)

Primjer.

...

```
case sf::Event::KeyPressed:  
    cout << "Tipka!" << endl;  
    break;
```

...

- ▶ probati gornji kod - pritisnuti i držati neku tipku
- ▶ generira se više `KeyPressed` događaja, s *defaultnim kašnjenjem* (isto kao pri pisanju u neki dokument)
- ▶ za onemogućavanje ponavljanja `KeyPressed` događaja pri držanju pritisnute tipke treba na prozoru izvršiti:

```
prozor.setKeyRepeatEnabled(false)
```



9/44

Kako provjeriti koja tipka je pritisnuta

- ▶ **key.code** daje kod pritisnute tipke
- ▶ kodovi - `sf::Keyboard::`
 - ▶ A, B, C, ..., X, Y, Z, Num0, Num1, ..., Num9
 - ▶ LControl, LShift, LAlt, LSystem, RControl, RShift, RAlt, RSystem
 - ▶ Escape, Menu, Pause
 - ▶ LBracket ([), RBracket (]), Semicolon (;), Comma (,), Period (.), Quote ('), Slash (/), Backslash (\), Tilde (~), Equal (=), Hyphen (-)
 - ▶ Space, Enter, Backspace, Tab, PageUp, PageDown, End, Home, Insert, Delete
 - ▶ Add (+), Subtract (-), Multiply (*), Divide (/)
 - ▶ Left (←), Right (→), Up (↑), Down (↓)
 - ▶ Numpad0, Numpad1, ..., Numpad9
 - ▶ F1, F2, ..., F15
- ▶ sljedeći kodovi su zastarjeli (u zagradama je navedeno čime su zamijenjeni):
 - ▶ Dash (Hyphen)
 - ▶ BackSpace (Backspace)
 - ▶ BackSlash (Backslash)
 - ▶ SemiColon (Semicolon)
 - ▶ Return (Enter)



10/44

Još malo o tipkama

- ▶ funkcija za provjeru je li određena tipka trenutno pritisnuta:

```
static bool sf::Keyboard::isKeyPressed(Key key)
```

Napomena.

- ▶ neke tipke imaju posebno značenje (ovisno o operacijskom sustavu) pa to može dovesti do neočekivanog ponašanja
- ▶ primjerice, na Windowsu i Visual studiu tipka F12 pokreće *debugger*



Primjer

```
...  
case sf::Event::KeyPressed:  
    if(event.key.code == sf::Keyboard::Up)  
        cout << "Tipka gore!" << endl;  
    break;  
...  

```



Rješenje početnog zadatka

```
sf::Vector2f pocetni_pomak(0.f, -0.1f); //gore
int main() {
    sf::RenderWindow prozor(sf::VideoMode(640,480),
                            "Prozor");
    float radijus = 20.f;
    sf::CircleShape t(radijus,3);
    t.setFillColor(sf::Color::Red);
    t.setOrigin(radijus, radijus);
    t.setPosition(prozor.getSize().x / 2.f,
                 prozor.getSize().y / 2.f);
    sf::Vector2f pomak(pocetni_pomak);
    float kut = 0;
    while (prozor.isOpen()) {
        sf::Event event;
        ...sljedeći slajd...
    }
    return 0;
}
```



Nastavak rješenja (unutar while petlje)

```
while (prozor.pollEvent(event))
    if(event.type == sf::Event::Closed)
        prozor.close();
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
    t.move(pomak);
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
    t.move(-pomak);
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
    t.setRotation(kut -= 0.1);
    updejt_pomaka(pomak, t.getRotation());
}
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
    t.setRotation(kut += 0.1);
    updejt_pomaka(pomak, t.getRotation());
}
prozor.clear(sf::Color(200, 200, 200, 255));
prozor.draw(t);
prozor.display();
```



Pomoćna funkcija

```
void updejt_pomaka(sf::Vector2f& pomak, float kut){  
    float rad = kut / 180 * 3.14;  
    pomak.x = cos(rad) * pocetni_pomak.x  
              - sin(rad) * pocetni_pomak.y;  
    pomak.y = sin(rad) * pocetni_pomak.x  
              + cos(rad) * pocetni_pomak.y;  
}
```

- ▶ ovo zahtijeva `#include<cmath>`
- ▶ `kut` je u stupnjevima, a za funkcije `sin` i `cos` trebamo u radijanima - zato smo prvo pretvorili stupnjeve u radijane
- ▶ o formuli korištenoj za rotaciju vektora smjera: [link](#)

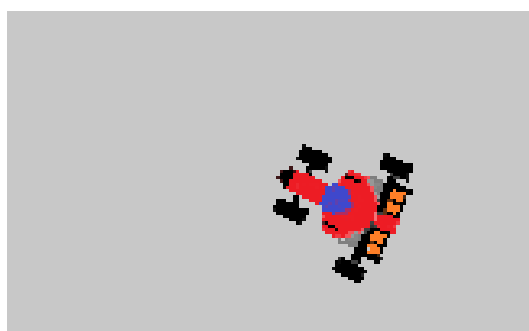
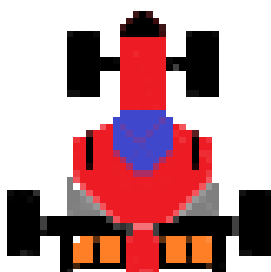


Upotreba slike

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)

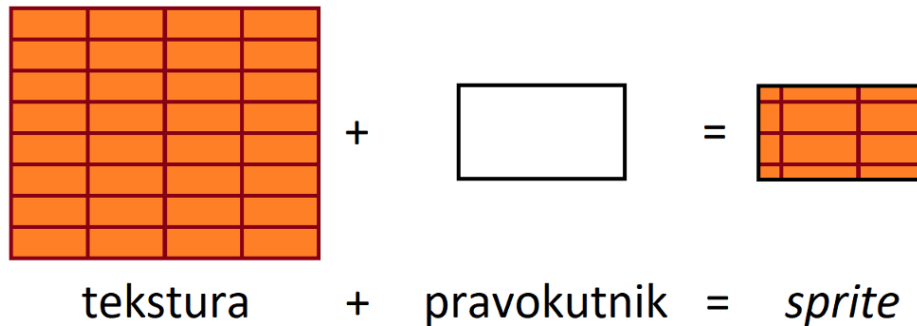
Prozor



Spriteovi i teksture

- ▶ **tekstura** = slika (nazivamo ju tekstura jer se prelikava na 2D objekt)
- ▶ **sprite** = pravokutnik s teksturom (YouTube video: [What are sprites](#))

Ilustracija:



Zašto naziv *sprite* (duh):

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



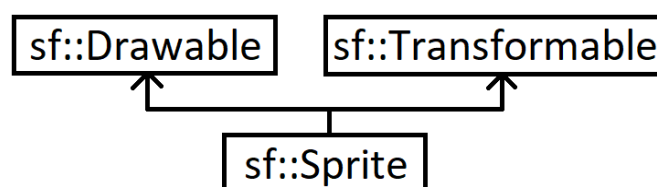
Klase za *sprite* i za teksturu

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
```



Sprite

- ▶ u kodu umjesto `sf::CircleShape t... it.setFillC...`:

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

Uočimo da ostalo u kodu ne trebamo mijenjati:

- ▶ crtanje je i dalje ovako: `prozor.draw(t);`
- ▶ `setOrigin`, `setPosition`, `move`, `setRotation` klasa `sf::Sprite` također ima jer nasljeđuje klasu `sf::Transformable`



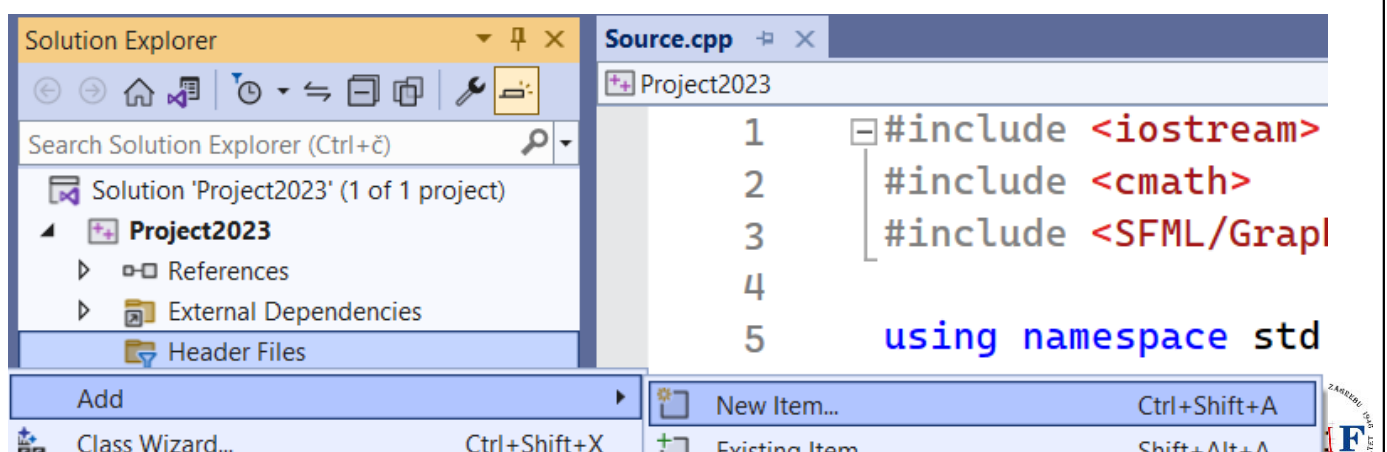
Zašto nam treba strukturiranje

- ▶ iz prethodnog koda mogli bi napraviti neku igru - možemo dodati:
 - ▶ provjeru kolizije s rubom ekrana,
 - ▶ stazu za utrke,
 - ▶ protivničke formule,
 - ▶ prepreke na stazi,
 - ▶ bodovanje, ...
- ▶ no, porastom količine koda on postaje **težak za održavanje** (tzv. [spaghetti kod](#) - na linku opisani i ostali „kodovi-tjestenine“)

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



```
1 #include <iostream>
2 #include <cmath>
3 #include <SFML/Grapph>
4
5 using namespace std
```

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 cijeliZaslona;  
};
```

- ▶ trebamo neki `sf::RenderWindow` za crtanje
- ▶ želimo imati podatke o veličini i naslovu prozora
- ▶ dva `bool` podatka - je li prozor otvoren/zatvoren i je li u *full screen* načinu



Pomoćne funkcije za stvaranje/uništavanje prozora

- ▶ bit će korištene u, primjerice, konstruktoru i destrukturu

```
class Prozor {  
    private:  
        void Stvori();  
        void Unisti();  
        ...  
};  
  
void Prozor::Stvori() {  
    auto stil = (cijeliZaslona ? 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)

```
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;
    cijeliZaslona = 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 cijeliZaslona;
        }
        sf::Vector2u dohvatiVelicinu() {
            return velicina;
        }
        ...
};
```



Funkcija za crtanje (onog što se može crtati)

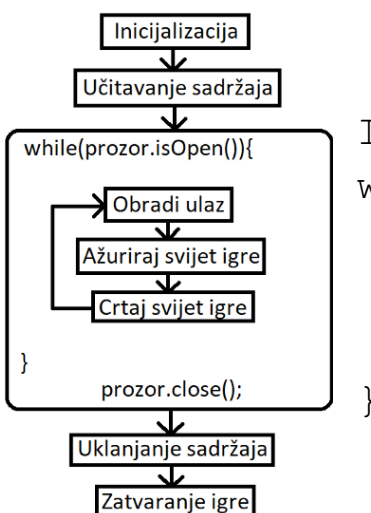
```
class Prozor {
    public:
        ...
        void crtaj(sf::Drawable&);
        ...
};

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.dohvatiProzor() -> 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();
    Prozor* dohvatiProzor() {
        return &p;
    }
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();
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();
}
```

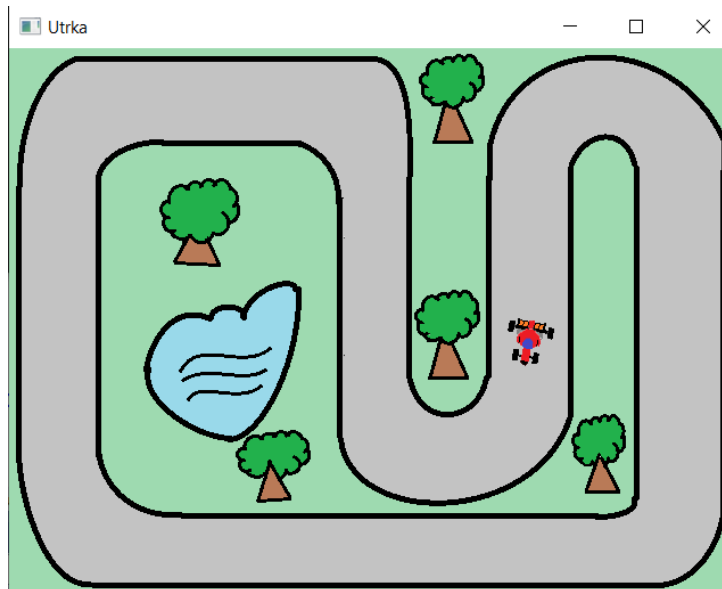
Glavni program (funkcija main)

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

int main() {
    Igra igra;
    while(!igra.dohvatiProzor()->jelGotov()){
        igra.obradiUlaz();
        igra.update();
        igra.renderiraj();
    }
    return 0;
}
```

Zadatak (za samostalno rješavanje)

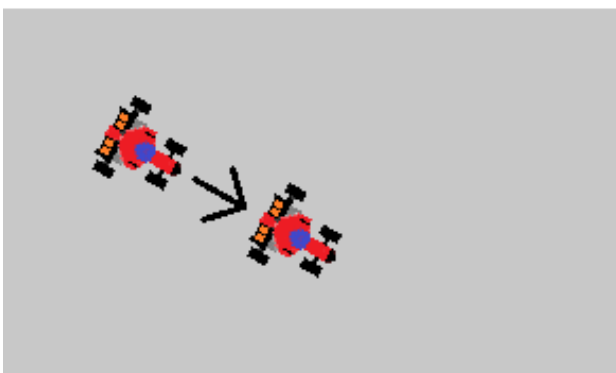
Zadatak. Dodati u pozadinu mapu (npr. može se iskoristiti datoteka `mapa.png` koja se može preuzeti na web-stranici kolegija). Dodati provjeru kolizije između formule i ruba prozora (tako da formula ne može napustiti područje prozora).



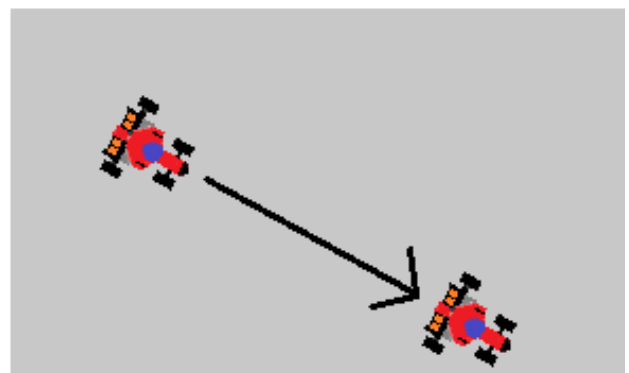
Problemi s vremenom izvršavanja

- ▶ 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

Utrka



Utrka



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 ⇒ 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 brzih (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);  
}
```

