(Pseudo)slučajni brojevi

Objektno programiranje - 3. vježbe (1. dio)

Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

26. ožujka 2025. godine

Napravimo novi projekt

- datoteka `main.cpp` (još koda ćemo dodavati na idućim slajdovima):

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

using namespace std;

int main() {

      ...

      return 0;
}
```

Prozor s kojim ćemo raditi

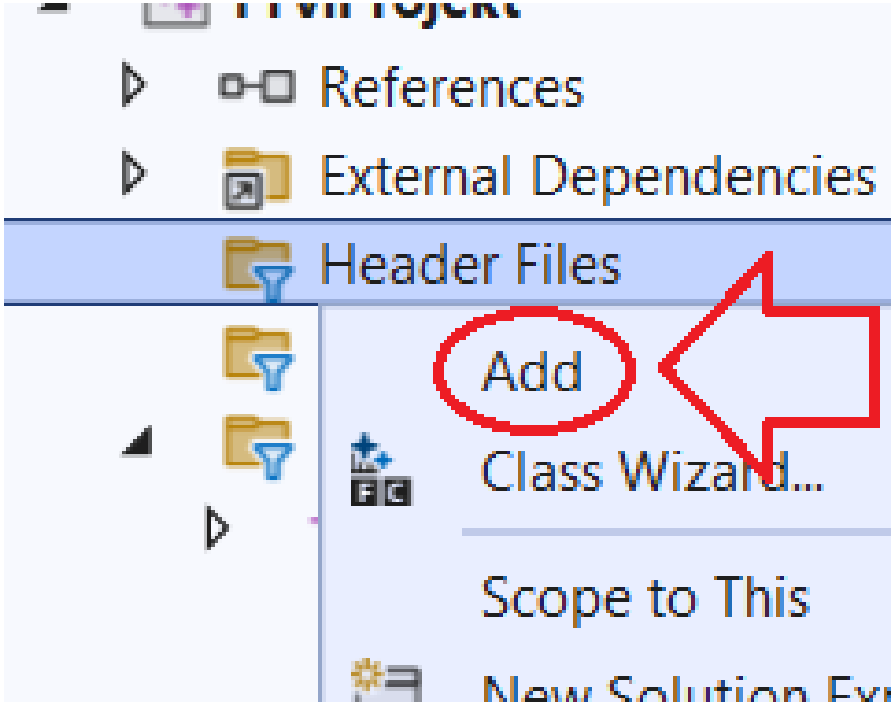
- dodati u `main` funkciju:

```
sf::RenderWindow prozor(sf::VideoMode(640, 640),
                        "Random krugovi");

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

      prozor.clear(sf::Color::Black);
      ...
      prozor.display();
}
```

Dodavanje novog zaglavlja u VS2022

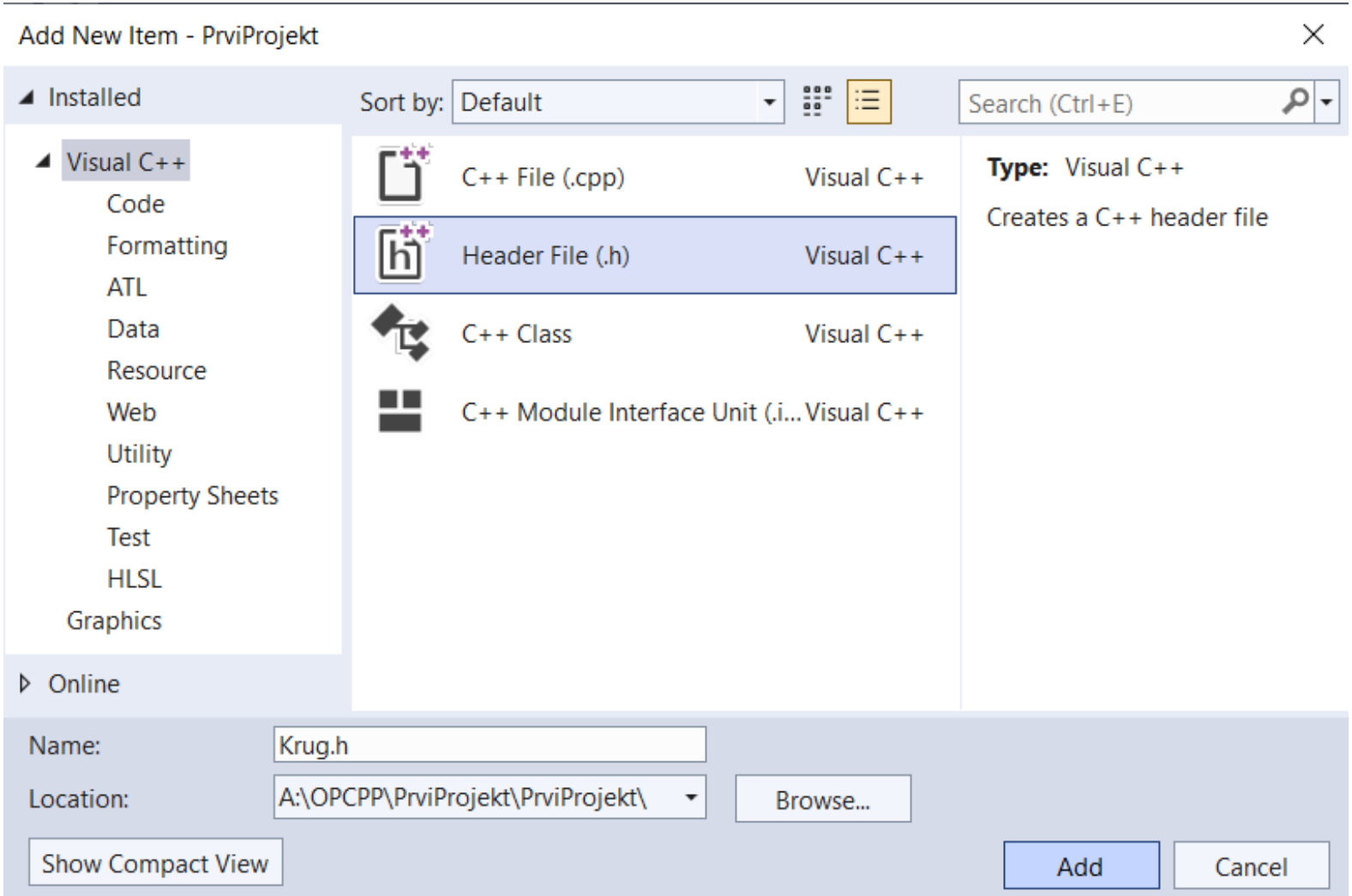
- dodati u projekt novu datoteku zaglavlja `Krug.h`

- u *Solution Explorer* prozoru napraviti desni klik na mapu *Header Files*, odabrati *Add* te zatim *New Item...*



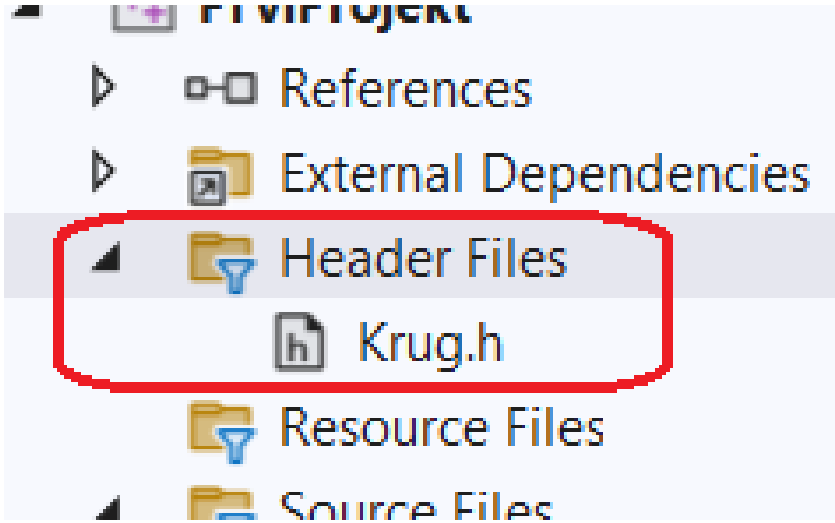
Dodavanje novog zaglavlja u VS2022

- odabrati *Header file (.h)* opciju, upisati naziv datoteke *Krug.h* te naposlijetku odabrati *Add*



Dodavanje novog zaglavlja u VS2022

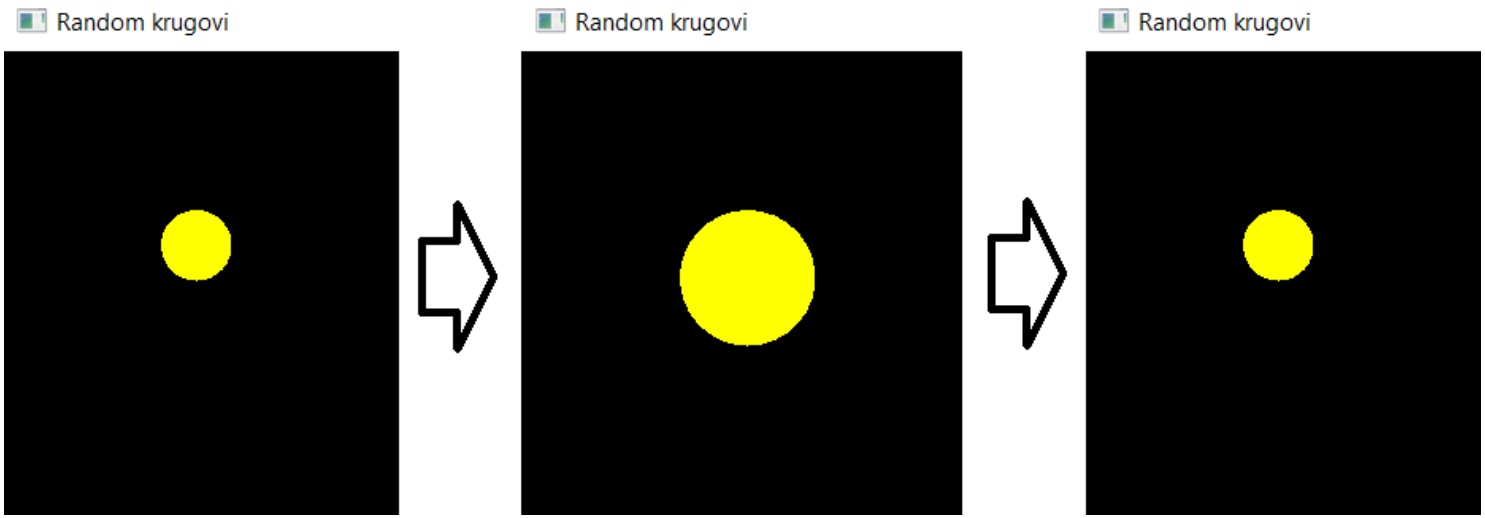
- datoteka bi se nakon tog postupka trebala pojaviti u *Solution Explorer* prozoru u mapi *Header Files*



- uočite kako je automatski u zaglavlje dodano `#pragma once`

Klasa `Krug` koju ćemo implementirati

- klasa `Krug` predstavlja žuti krug čiji se radijus postepeno povećava od 1 do 50 piksela te se zatim smanjuje (i tako dalje)
- ishodište kruga je na koordinatama (100,100))



Klasa `Krug` koju ćemo implementirati

- klasa `Krug` treba se moći brinuti o sebi (zna sama ažurirati veličinu kruga te se iscrtati na dani prozor)

```
Krug k;

while (prozor.isOpen()) {

        // ...  kao i prije ...

      prozor.clear(sf::Color::Black);

      k.updejt();
      k.prikaz(prozor);

      prozor.display();
}
```

Klasa `Krug` (u datoteci `Krug.h`)

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

class Krug {
private:
      sf::CircleShape krug;
      float rast = 0.01f;
      unsigned max_radijus = 50;
public:
      Krug(); // konstruktor
      void prikaz(sf::RenderWindow& prozor);
      void updejt();
};
```

- podsjetnik: klasa `sf::RenderWindow` nasljeđuje klasu `sf::Window` što znači da nasljeđuje i klasu `sf::NonCopyable`
- prema tome se naš prozor **ne može kopirati** (zato metoda `Krug::prikaz` prima referencu na dani prozor - tako ne dolazi do kopiranja prozora ✓)

Implementacija metoda klase `Krug`

- u *Solution Explorer* prozoru, u mapi *Source Files* napraviti novu datoteku *Krug**.cpp*** (na sličan način kako smo u mapi *Header Files* napravili datoteku *Krug.h*)
- konstruktor postavlja boju kruga, njegov (početni) radijus i njegovu poziciju

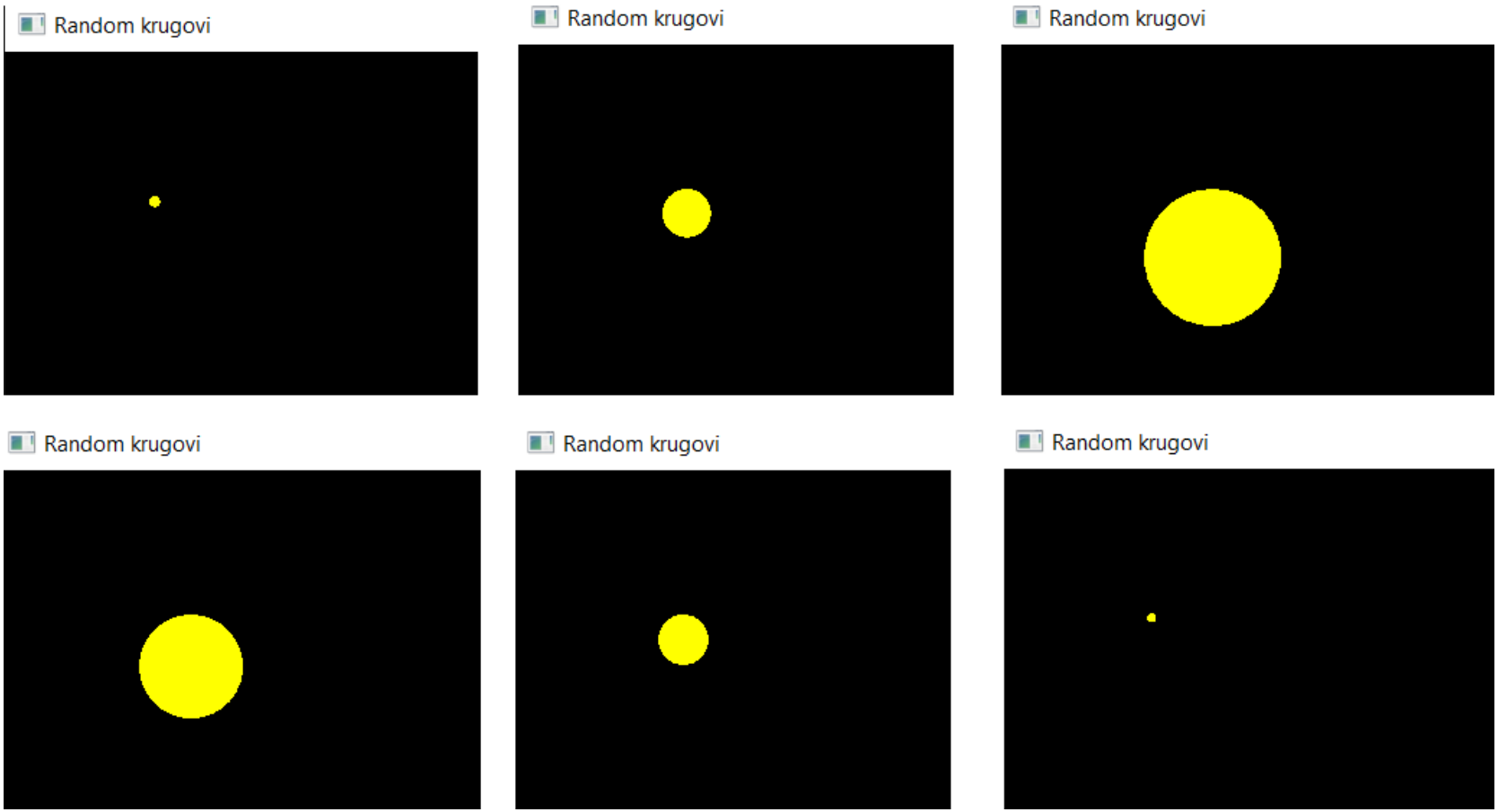
```
#include <SFML/Graphics.hpp>
#include "Krug.h"

using namespace std;

Krug::Krug() {
    krug.setFillColor(sf::Color::Yellow);
    krug.setRadius(1.f);
    krug.setPosition(sf::Vector2f(100.f,100.f));
}
```

Implementacija metoda klase `Krug`

```
void Krug::prikaz(sf::RenderWindow& prozor) {
     prozor.draw(krug);
}

void Krug::updejt() {
     float r = krug.getRadius();
     if (r < 1 || r > max_radijus)
          rast = -rast;
     krug.setRadius(r + rast);
}
```

- metoda `prikaz` prikazuje krug na danom prozoru
- metoda `updejt` provjerava je li trenutni radijus kruga prevelik ili premalen, te ovisno o tome mijenja predznak od `rast` (tako da se prevelik krug počne smanjivati, a premalen povećavati)

Prikaz dobivenog programa



- uočite u kojem se smjeru povećava/smanjuje krug obzirom da mu nismo mijenjali ishodište!

Krug na „slučajnim” koordinatama

- umjesto na koordinatama (100,100) želimo da se krug stvori na „slučajno” odabranim koordinatama
- zbog bolje preglednosti na ekranu, promijenimo maksimalan radijus koji krug može imati iz 50 u 5 piksela:

```
class Krug {
        ...
        unsigned max_radijus = 5;
        ...
};
```

- koristit ćemo zaglavlje **`random`** - ima dvije vrste tipova:
  - ***engine*** - stvaraju niz slučajnih nenegativnih cijelih brojeva
  - ***distribution*** - koriste *engine* za vraćanje brojeva prema određenoj vjerojatnosnoj distribuciji
- **generator slučajnih brojeva** = *engine* + *distribution*

Nova verzija konstruktora klase `Krug`

- promjene u datoteci `Krug.cpp`:

```
#include <random>
...
Krug::Krug() {
        default_random_engine e;
        krug.setFillColor(sf::Color::Yellow);
        krug.setRadius(1.f);
        krug.setPosition(sf::Vector2f(e(),e()));
}
```

- `e` je funkcijski objekt $\rightsquigarrow$ `e()` daje sljedeći slučajan broj
- ovo nam predstavlja problem - ne želimo proizvoljan slučajan broj za koordindate jer želimo koordinate unutar (vidljivog dijela) našeg prozora

Transformacija sirovih u upotrebljive slučajne brojeve

- sljedeći kod:

```
float a = e(), b = e();
cout << a << ", " << b << endl;
```

daje ispis poput:

```
3.49921e+09, 5.81869e+08
```

- to nije raspon koji smo željeli - koristimo distribucijski objekt - primjerice, normalna distribucija od 0 do 9 (uključivo):

```
Krug::Krug() {
    default_random_engine e;
    normal_distribution<float> u(320, 100);
    krug.setFillColor(sf::Color::Yellow);
    krug.setRadius(1.f);
    krug.setPosition(sf::Vector2f(u(e), u(e)));
}
```

Napomena o distribucijskim tipovima

- u kodu s prethodnog slajda imali smo

```
normal_distribution<float> u(320, 100);
```

- distribucijski tipovi su **predlošci** - imaju jedan parametar tipa za tip rezultata koji će tražena distribucija generirati (u gornjem primjeru to je `float`)
- distribucijski tipovi imaju restrikcije na tip koji možemo navesti kao tip predloška
  - primjerice, neki predlošci mogu se koristiti za generiranje samo realnih brojeva[1], a neki samo za cijele brojeve[2]

[1] float, double, long double

[2] short, int, long, long long, unsigned short, unsigned int, unsigned long, unsigned long long

Vektor krugova

- dodamo u `main.cpp` datoteku:

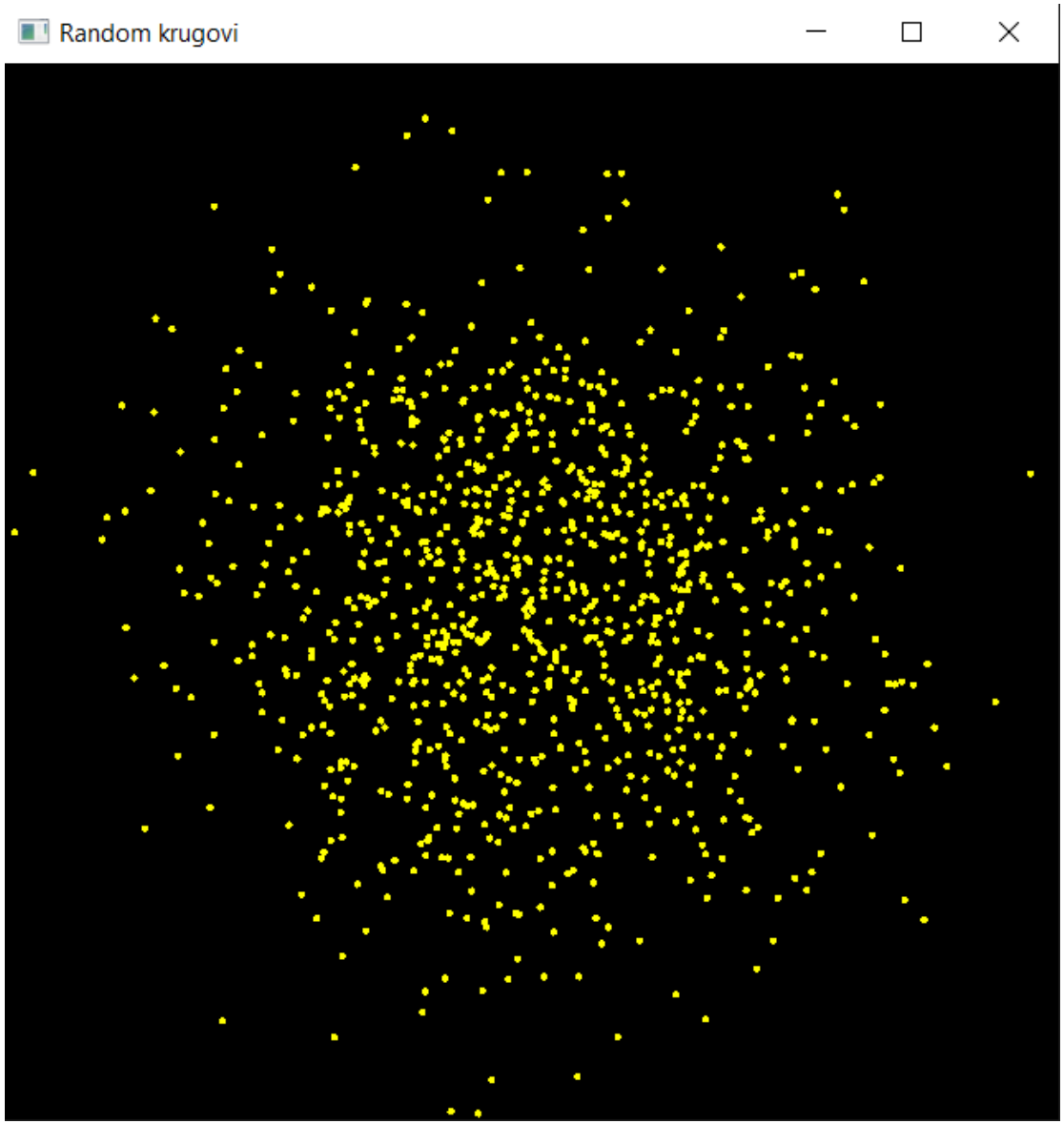
```
#include <vector>

int main() {
        ...
        vector<Krug> krugovi(1000);
        while (prozor.isOpen()) {
                ...
                for (auto& c : krugovi) {
                        c.updejt();
                        c.prikaz(prozor);
                }
                prozor.display();
        }
        return 0;
}
```

Više krugova - ali svi su isti!

- ako pokrenemo program, vidimo da su svi krugovi završili na istim koordinatama
- rješenje: upotreba ključne riječi **`static`**
- ako objekte stavimo kao `static`, oni će zadržati svoje stanje kroz funkcijske pozive

```
Krug::Krug() {
      static default_random_engine e;
      static normal_distribution<float> u(320, 100);
...
}
```

Dobiveni rezultat



Daljnji problem

- no, svako pokretanje istog programa daje iste koordinate
- **Rješenje**. dajemo sjeme (*seed*) - vrijednost koju *engine* koristi kako bi počeo generirati brojeve od nove vrijednosti
- umjesto fiksne vrijednosti (npr. `32540`), koristimo `time(0)`

```
Krug::Krug() {
      static default_random_engine e(time(0));
      ...
}
```

Druge razdiobe (distribucije)

Osim normalne distribucije, mogu se koristiti i ostale:

- uniformna, Bernoullijeva, Poissonova, Cauchyjeva, Fisherova, ...

Primjer. Biranje boje kruga na slučajan način:

```
Krug::Krug() {
  static default_random_engine e(time(0));
  static normal_distribution<float> u(320, 100);
  static uniform_int_distribution<unsigned> b(0,255);
  krug.setFillColor(sf::Color(b(e), b(e), b(e)));
  krug.setRadius(1.f);
  krug.setPosition(sf::Vector2f(u(e), u(e)));
}
```

Napomena. Rasponi kod distribucija su uključivi (tj. u gornjem primjeru imamo slučajan cijeli broj od 0 do 255, oba uključivo).

Dobiveni rezultat