

SFML - Zmija

Objektno programiranje - 11. vježbe

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

29. svibnja 2024. godine



Za predjelo: Enumeracije

- korisnički definirani tip podatka koji se sastoji od cjelobrojnih konstanti - ključna riječ pri definiranju je **enum**
- npr. smjer kretanja mogli bi u kodu pamtiti tako da:
 - smjer gore pamtimo kao 0,
 - smjer dolje pamtimo kao 1,
 - smjer lijevo pamtimo kao 2,
 - smjer desno pamtimo kao 3

Primjer. Isto možemo postići ovako:

```
enum smjer { gore, dolje, lijevo, desno };
```

Prednosti korištenja enumeracije:

- povećava apstrakciju - možemo se usredotočiti na vrijednosti, a ne na to kako ih spremamo
- povećava čitljivost i olakšava dokumentiranje koda

Vrste enumeracija

- dvije vrste enumeracija:
 - bez dosega (*unscoped*) - samo ključna riječ **enum**
 - s dosegom (*scoped*) - ključne riječi **enum class** (ili ekvivalentno: **enum struct**) - nakon njih isto slijedi ime i popis **enumeratora** odvojenih zarezom:

Primjer.

```
enum class smjer { gore, dolje, lijevo, desno };
```

- ako je ime enumeracije izostavljeno, varijable tog tipa mogu se definirati samo kao dio `enum` definicije

```
enum smjer { gore, dolje, lijevo, desno };
```

```
smjer a = gore;
```

```
enum { naprijed, nazad } b;
```

```
b = nazad;
```

```
cout << a << " " << b << endl; //ispis: 0 1
```

Vrijednosti enumeratora

- po *defaultu* vrijednosti počinju od 0 i svaki ima za 1 veću vrijednost od prethodnog
- možemo im pri deklaraciji navesti vrijednost (ostali slijede pravilo: 1 više od prethodnog)
- vrijednosti ne moraju biti jedinstvene

Primjer.

```
enum x { a = -3, b, c = 12, d, e = 12 };  
cout << a << ", " << b << ", " << c << ", "  
      << d << ", " << e << endl;
```

Ispis: -3, -2, 12, 13, 12

- također, enumeratori su `const` i mogu se inicijalizirati konstantnim izrazom

```
int n;  
enum x { a = n, b, c = 12, d, e = 12 };  
c = 5; X
```



Zašto trebamo i enum class?

Problem 1.

- dvije enumeracije ne mogu imati enumeratore istog imena

Primjer.

```
enum smjerA { gore, dolje, lijevo, desno };
enum smjerB { lijevo, desno, naprijed, nazad };

error: redeclaration of 'lijevo'
error: redeclaration of 'desno'
```

Rješenje korištenjem enum class:

```
enum class smjerA { gore, dolje, lijevo, desno };
enum class smjerB { lijevo, desno, naprijed, nazad };
```

Zašto trebamo i `enum class`? (nastavak)

Problem 2.

- *unscoped* enumeracije nisu *type-safe* - implicitno imamo konverziju u cijeli broj te je moguće npr. usporediti enumeratore različitih enumeracija

Primjer.

```
enum smjerA { gore, dolje, lijevo, desno };
enum smjerB { naprijed, nazad };
smjerA a = dolje;
if(a == nazad) //daje samo warning
    cout << "Isti!" << endl;
```

Ispis: Isti!

Primjer.

```
enum class smjerA { gore, dolje, lijevo, desno };  
enum class smjerB { naprijed, nazad };  
smjerA a = smjerA::dolje;  
if(a == smjerB::nazad)  
    cout << "Isti!" << endl;
```

Prethodni kod se sad neće kompajlirati.

```
error: no match for 'operator==' (operand types are  
'main()::smjerA' and 'main()::smjerB')
```

Primjer enumeracije iz SFML-a: Keyboard::Key

Dio iz datoteke `Keyboard.hpp`:

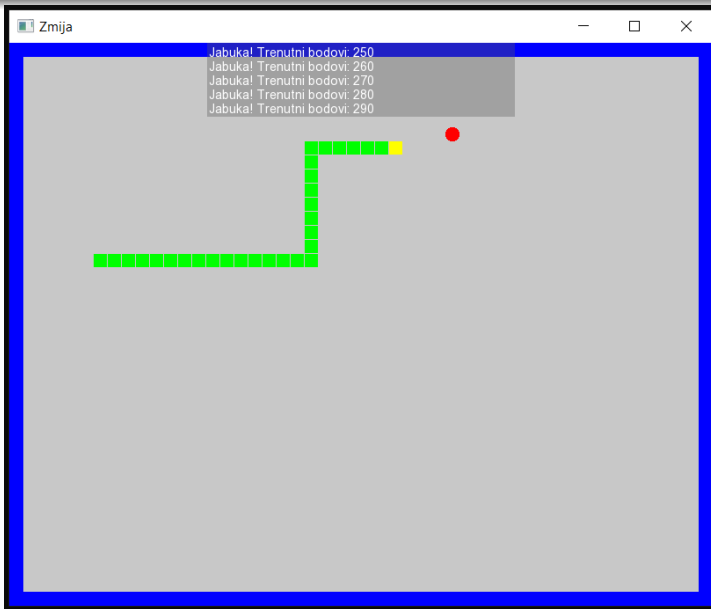
```
class SFML_WINDOW_API Keyboard {
public:
    enum Key {
        Unknown = -1,
        A = 0,
        B,
        C,
        ...
        Hyphen,
        ...
        // Deprecated values:
        Dash = Hyphen,
        ...
    };
    ...
};
```


Primjer.

```
sf::Event event;
while (prozor.pollEvent(event)) {
    switch (event.type) {
        ...
        case sf::Event::KeyPressed:
            if(event.key.code == sf::Keyboard::Up)
                cout << "Tipka gore!" << endl;
            break;
        ...
    }
}
```

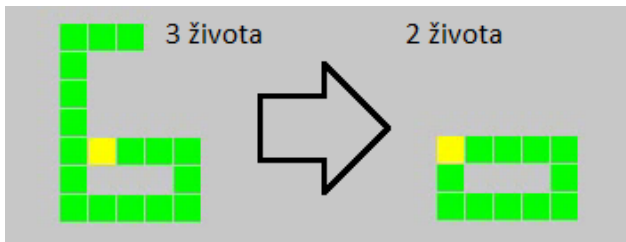
Napomena. Kad bi na prethodnom slajdu imali `enum class Key` umjesto `enum Key` tada bismo umjesto uokvirenog pisali:

```
sf::Keyboard::Key::Up
```



Pravila koja ćemo imati

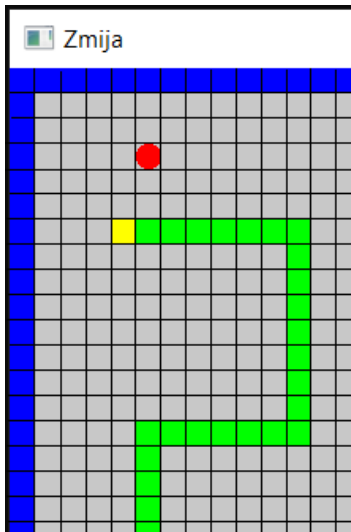
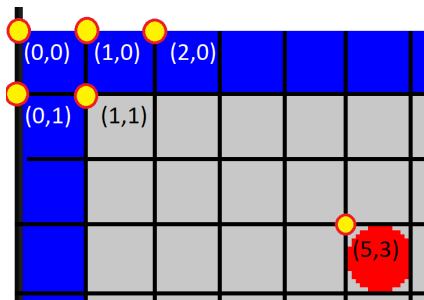
- Zmija se može kretati u četiri smjera: gore, dolje, lijevo i desno.
- Ako zmija pojede jabuku tada joj se duljina poveća za 1, te igrač dobiva 10 bodova (nakon toga sljedeća jabuka pojavi se na slučajnom mjestu).
- Zmija ima tri života - ako izgubi sva tri, igra također završava. Život se gubi u slučaju da se zmija zaleti u samu sebe. Tada se također uklanja dio zmije od mjesta sudara do njenog repa.



- Ako se zmija zabije u zid, igra odmah završava.
- Svakim novim korakom brzina zmije se povećava.

Kretanje zmiје po prozoru

- zmiја će se kretati po mreži koja se sastoji od kvadratića dimenzije 16×16 piksela
- kretanje će biti u fiksnim vremenskim koracima (u svakom koraku se pomakne za 1 kvadratić)



Datoteka zmi_ja.cpp (main funkcija)

- može se preuzeti na web-stranici kolegija
- kod kao na prethodnim vježbama (**s mjerenjem vremena**)

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

using namespace std;

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

- može se preuzeti na web-stranici kolegija
- ista kao na prethodnim vježbama
- podsjetnik na sučelje klase Prozor:

Konstruktori:

- **Prozor()** - otvara prozor dimenzije 640×480 s naslovom "Prozor"
- **Prozor(const std::string&, const sf::Vector2u&)** - otvara prozor sa zadanim naslovom i zadanih dimenzija

Dobivanje informacija o prozoru:

- **sf::Vector2u dohvatiVelicinu()** - veličina prozora
- **bool jelCijeli()** - je li preko cijelog zaslona
- **bool jelGotov()** - je li bilo zahtjeva za zatvaranje prozora

Iz naziva bi trebalo biti jasno čemu služe preostale funkcije članice klase `Prozor`:

- `void crtaj(sf::Drawable&)`
- `void ocisti()`
- `void prebaciNaCijeli()`
- `void prikazi()`
- `void update()`

- može se preuzeti na web-stranici kolegija
- tu datoteku ćemo nadopuniti kad napravimo potrebne klase za zmiju

```
class Igra {  
    public:  
        Igra();  
        ~Igra();  
        void obradiUlaz();  
        void update();  
        void renderiraj();  
        Prozor* dohvatiProzor() {  
            return &p;  
        }  
}
```

(nastavak na sljedećem slajdu...)

Nastavak koda s prethodnog slajda - klasa Igra

```
        sf::Time protekloVrijeme();
        void restartSata();
private:
    Prozor p;
    //ovdje ćemo dodati neke dijelove
    sf::Clock sat;
    sf::Time vrijeme;
};

sf::Time Igra::protekloVrijeme() {
    return vrijeme;
}

void Igra::restartSata() {
    vrijeme += sat.restart();
}

Igra::~Igra() {}
```

- uočite dijelove za mjerenje vremena (**fiksna br. iteracija u sek**)

Nastavak - što ćemo dopuniti kasnije

```
void Igra::renderiraj() {
    p.ocisti();
    //tu ćemo crtati što treba
    p.prikazi();
}

void Igra::update() {
    p.update();
    //tu korak zmiije (ako je vrijeme za to)
}

void Igra::obradiUlaz() {
    //pogledati što je korisnik pritisnuo
}

Igra::Igra() : p("Zmija", sf::Vector2u(800, 640)) {
    //što još početno treba postaviti
}
```

Zadatak. Napraviti novu datoteku `Zmija.h` u našem projektu. Početni sadržaj prikazan je ispod na ovome slajdu.

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

class Zmija {
public:
private:
};
```

Podaci koje pamtimo o zmiji

```
class Zmija {  
    public:  
    private:  
        int brZivota;  
        int bodovi;  
        bool izgubio;  
        float brzina;  
        int velBloka;  
        sf::RectangleShape blok;  
};
```

- broj preostalih života, broj postignutih bodova
- je li igrač izgubio (ili igra još traje)
- veličina blok (zmija je izgrađena od njih) i oblik tog bloka koji će se crtati (pravokutnik)
- `brzina` je broj prijeđenih polja (ne piksela!) u sekundi, tj. broj koraka u sekundi (povećavat će se za npr. 0.01 po koraku)

Smjer u kojem će zmija napraviti sljedeći korak

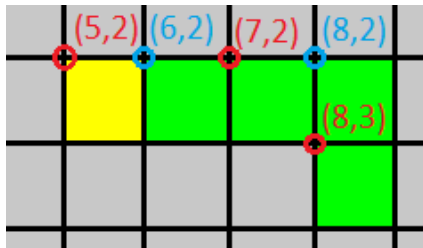
```
enum class Smjer{Nema, Gore, Dolje, Lijevo, Desno};

class Zmija {
    ...
private:
    Smjer smjer;
    ...
};
```

- Smjer izvan klase Zmija (da ne moramo pisati npr. Zmija::Smjer::Gore nego samo Smjer::Gore)
- početno (prije no što korisnik pritisne neku tipku) zmija se neće kretati ⇒ smjer je Smjer::Nema

Koordinate svakog bloka zmiije

- pamtimo koordinate svakog bloka
- pomicanje za 1 polje \Rightarrow dodavanje na početak i izbacivanje s kraja (zbog toga i efikasnosti koristimo `deque`)



- koordinate (prve pripadaju glavi, a posljednje repu):
 $(5, 2), (6, 2), (7, 2), (8, 2), (8, 3)$
- koordinate zadnjeg bloka koji će biti „uklonjen” pri pomicanju korisno je zapamtiti (jer ako je pri pomicanju pojedena jabuka „vraćanje” tog bloka odgovara povećanju duljine zmiije za 1)

Pamćenje koordinata blokova zmije

```
#include <deque>

...

class Zmija {
    ...
private:
    std::deque<sf::Vector2i> koordinate;
    sf::Vector2i koordUklonjenog;
    ...
};
```

- koordinate su parovi cijelih brojeva

Sučelje: funkcije za dohvat podataka

```
class Zmija {  
    public:  
        Smjer DohvatiSmjer() {  
            return smjer;  
        }  
        int DohvatiZivote() {  
            return brZivota;  
        }  
        int DohvatiBodove() {  
            return bodovi;  
        }  
        bool JelIzgubio() {  
            return izgubio;  
        }  
        float DohvatiBrzinu() {  
            return brzina;  
        }  
        ...  
};
```


Ostale potrebne funkcije

- klasa `Zmija` može sama provjeriti je li se zabila u samu sebe, no kasnije će nam biti važno **gdje je glava** (npr. za provjeru zabijanja zmije u zid - u tom slučaju treba **zabilježiti da je igrač izgubio**)
- jedenjem jabuke treba moći **povećati bodove za 10**

```
class Zmija {
public:
    sf::Vector2i KoordinateGlave() {
        return koordinate.front();
    }
    void Izgubio() {
        izgubio = true;
    }
    void PovecajBodove() {
        bodovi += 10;
    }
    ...
};
```

Ostale potrebne funkcije (nastavak)

- funkcija `Igra::obradiUlaz` trebat će funkciju za promjenu smjera

```
class Zmija {  
    public:  
        void PostaviSmjer(Smjer s) {  
            smjer = s;  
        }  
        ...  
};
```

Konstruktor i destruktor

```
class Zmija {  
    public:  
        Zmija(int);  
        ~Zmija();  
        ...  
};
```



```
Zmija::Zmija(int) : velBloka(v) {  
    blok.setSize(sf::Vector2f(v-1, v-1));  
    Reset();  
}
```

```
Zmija::~~Zmija() {} //nemamo što tu staviti
```

- konstruktor prima veličinu bloka i postavlja pravokutnik koji se crta na dimenziju $(v - 1) \times (v - 1)$ piksela (-1 kako za „rubove”)
- Reset() - ostale postavke (kao i kod ponovnog pokretanja igre - npr. broj života opet postaviti na 3)

Funkcija Reset () - postavke pri pokretanju nove igre

```
class Zmija {  
    public:  
        void Reset();  
        ...  
};  
  
void Zmija::Reset() {  
    koordinate.clear();  
    koordinate.push_back(sf::Vector2i(10, 10));  
    PostaviSmjer(Smjer::Nema);  
    brzina = 10;  
    brZivota = 3;  
    bodovi = 0;  
    izgubio = false;  
}
```

- početno zmija ima jedan blok na koordinatama (10, 10)
- početno stoji na mjestu (Smjer::Nema)

Funkcija Korak

```
class Zmija {
    public:
        void Korak();
        ...
};

void Zmija::Korak() {
    if (brzina <= 20)
        brzina += 0.01f;
    if (smjer != Smjer::Nema) {
        Pomakni();
        ProvjeraSudara();
    }
}
```

U svakom koraku:

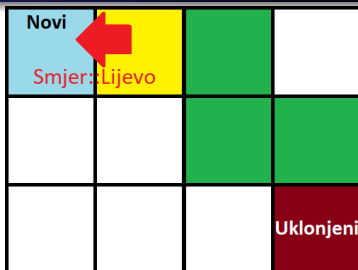
- povećamo brzinu (ali ne želimo prebrzu zmiju!)
- ako se zmija kreće (početno se ne kreće!), pomaknemo zmiju i provjerimo je li se zabila sama u sebe

```
class Zmija {  
    public:  
        void Pomakni();  
        ...  
};
```

```
void Zmija::Pomakni() {  
    sf::Vector2i novi = KoordinateGlave();  
    koordUklonjenog = koordinate.back();  
    koordinate.pop_back();
```

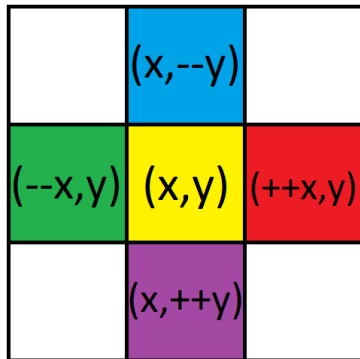
(nastavak koda ove funkcije na sljedećem slajdu...)

- koordinate novog dijela koji ćemo dobiti dobivamo povećanjem/smanjenjem x ili y koordinate glave za 1
- zadnje koordinate u deque prije uklanjanja spremimo jer u slučaju jedenja jabuke taj dio treba ponovo vratiti



Funkcija Pomakni (nastavak)

```
switch (smjer) {  
    case Smjer::Gore:  
        --novi.y;  
        break;  
    case Smjer::Dolje:  
        ++novi.y;  
        break;  
    case Smjer::Lijevo:  
        --novi.x;  
        break;  
    case Smjer::Desno:  
        ++novi.x;  
}  
koordinata.push_front(novi);  
}
```



Funkcija ProvjeraSudara

```
class Zmija {  
    public:  
        void ProvjeraSudara();  
        ...  
};  
  
void Zmija::ProvjeraSudara() {  
    auto velicina = koordinate.size();  
    sf::Vector2i glava = koordinate[0];  
    if (velicina > 4)  
        for(size_t i = 1; i < velicina; ++i)  
            if (koordinate[i] == glava) {  
                Odrezi(i); //ukloni od i-tog do kraja  
                return;  
            }  
}
```


Funkcija Odrezi

- uklanjamo sve blokove zmijske od i -tog (uključivo) od zadnjeg bloka (sad je $(i - 1)$). blok zadnji pa su njegove koordinate spremljene u `koordUklonjenog` + **provjera broja života**

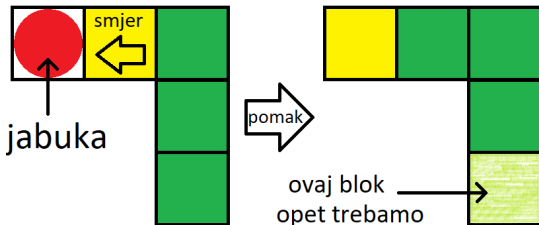
```
class Zmija {
public:
    void Odrezi(size_t);
    ...
};

void Zmija::Odrezi(size_t i) {
    auto velicina = koordinate.size();
    koordUklonjenog = koordinate[i-1];
    for (auto j = i; j < velicina; ++j)
        koordinate.pop_back();
    --brZivota;
    if (brZivota == 0)
        Izgubio();
}
```

Funkcija Produlji

```
class Zmija {  
    public:  
        void Produlji();  
        ...  
};
```

```
void Zmija::Produlji() {  
    coordinate.push_back(koordUklonjenog);  
}
```

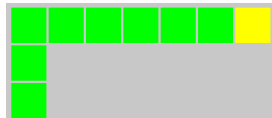


Crtanje zmije na ekran (koristimo `Prozor::crtaj`)

```
#include "Prozor.h"
```

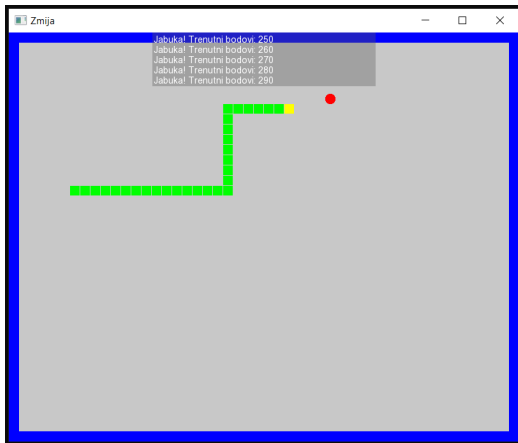
```
class Zmija {  
public:  
    void Renderiraj(Prozor*);  
    ...  
};
```

```
void Zmija::Renderiraj(Prozor* p) {  
    auto velicina = koordinate.size();  
    for (size_t i = 0; i < velicina; ++i) {  
        blok.setFillColor((i == 0) ?  
            sf::Color::Yellow : sf::Color::Green);  
        blok.setPosition(koordinate[i].x * velBloka,  
            koordinate[i].y * velBloka);  
        p->crtaj(blok);  
    }  
}
```



Klasa Svijet

- osim zmije trebamo **rub** i **jabuku**
- igra je jednostavna pa umjesto posebnih klasa imamo još jednu - klasu `Svijet`
- zbog jednostavnosti napisat ćemo kod u istu datoteku `Zmija.h`



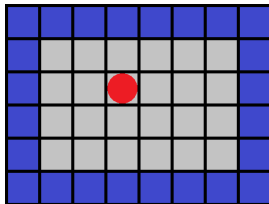
Klasa Svijet - što ćemo pamtiti

- osim oblika za rub i jabuku (s pripadnim koordinatama jabuke) pamtimo i **potrebne veličine**
- dodana i funkcija koja vraća veličinu bloka

```
class Svijet {  
    public:  
        int dohvatiVBloka() {  
            return velicinaBloka;  
        }  
    private:  
        sf::Vector2u velicinaProzora;  
        int velicinaBloka;  
        sf::Vector2i jabukaKoord;  
        sf::CircleShape jabuka;  
        sf::RectangleShape rub;  
};
```

Konstruktor i destruktor

```
class Svijet {  
    public:  
        Svijet (int, sf::Vector2u) ;  
        ~Svijet () ;  
    ...  
};
```



```
Svijet::Svijet (int vBloka, sf::Vector2u vProzora) :  
    velicinaBloka (vBloka), velicinaProzora (vProzora) {  
    PostaviJabuku () ;  
    jabuka.setFillColor (sf::Color::Red) ;  
    jabuka.setRadius (vBloka / 2.f) ;  
    rub.setFillColor (sf::Color::Transparent) ;  
    rub.setSize (sf::Vector2f (vProzora.x, vProzora.y)) ;  
    rub.setOutlineColor (sf::Color::Blue) ;  
    rub.setOutlineThickness (-vBloka) ;  
}  
Svijet::~~Svijet () {}
```

Postavljanje jabuke na prozoru

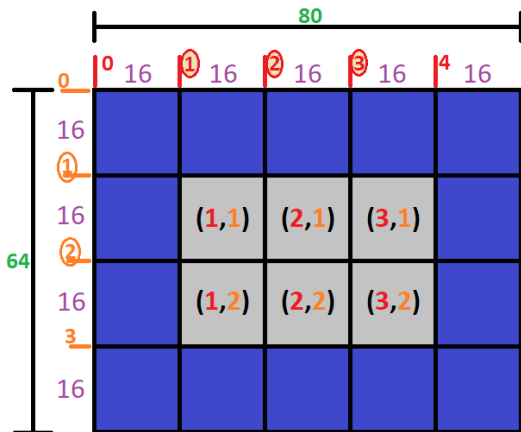
- početno, ali i nakon što zmija pojede jabuku (tj. glava zmiije dođe na koordinate jabuke), treba odrediti nove (**slučajno odabrane**) koordinate jabuke

```
#include <random>
```

```
class Svijet {  
    public:  
        void PostaviJabuku();  
        ...  
};
```

Kako odrediti te slučajne koordinate - u koje polje će se jabuka postaviti i gdje će se nacrtati?

Određivanje koordinata za postavljanje jabuke



Račun za određivanje „koordinata bloka” $(x, y) \in \{(1, 1), \dots, (3, 2)\}$

- $80/16 = 5 \rightarrow$ za $br \in \mathbb{N}$, $x' = br \% (5 - 2) \in \{0, 1, 2\}$
- $64/16 = 4 \rightarrow$ za $br \in \mathbb{N}$, $y' = br \% (4 - 2) \in \{0, 1\}$

\Rightarrow trebamo $(x' + 1, y' + 1)$

Funkcija PostaviJabuku

```
void Svijet::PostaviJabuku() {  
    static std::uniform_int_distribution<unsigned>  
        u(0,10000);  
    static std::default_random_engine e(time(0));  
    int maxX = (velicinaProzora.x/velicinaBloka)-2;  
    int maxY = (velicinaProzora.y/velicinaBloka)-2;  
    jabukaKoord = sf::Vector2i(u(e) % maxX + 1,  
        u(e) % maxY + 1);  
    jabuka.setPosition(jabukaKoord.x * velicinaBloka,  
        jabukaKoord.y * velicinaBloka);  
}
```

- za blok (x, y) crtanje je na $(x \cdot velicinaBloka, y \cdot velicinaBloka)$ (ishodište je u gornjem lijevom kutu bloka!)
- uočite: broj blokova određen iz dimenzije prozora i bloka - idealno da dimenzije prozora višekratnici veličine bloka

Funkcija Update

- u svakom koraku, zmija se pomakne i može detektirati ako se zabije sama u sebe
- no, treba pogledati: je li zmija pomicanjem pojela jabuku ili se zabila u zid (taj kod je na sljedećem slajdu)

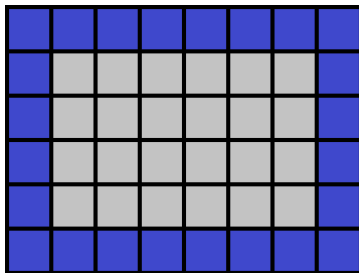
```
class Svijet {
public:
    void Update (Zmija&); //& => ne kopiramo zmiju!
    ...
};
```

```
void Svijet::Update (Zmija& igrac) {
    if (igrac.KoordinateGlave () == jabukaKoord) {
        igrac.Produlji ();
        igrac.PovecajBodove ();
        PostaviJabuku ();
    }
```

(nastavak koda funkcije je na idućem slajdu)

Funkcija Update (nastavak)

```
sf::Vector2i brPolja(velicinaProzora.x /  
    velicinaBloka,velicinaProzora.y/velicinaBloka);  
if (igrac.KoordinateGlave().x <= 0  
    || igrac.KoordinateGlave().y <= 0  
    || (igrac.KoordinateGlave().x >= brPolja.x-1)  
    || (igrac.KoordinateGlave().y >= brPolja.y-1))  
    igrac.Izgubio();  
}
```



- kao i u funkciji `Zmija::Renderiraj` koristimo `Prozor::crtaj`

```
class Svijet {  
    public:  
        void Renderiraj(Prozor*);  
        ...  
};  
  
void Svijet::Renderiraj(Prozor *p) {  
    p->crtaj(rub);  
    p->crtaj(jabuka);  
}
```

Povezivanje svih dosadašnjih dijelova u cjelinu

- dopunimo klasu `Igra` u datoteci "**Zmija.h**"
- podsjetnik na glavnu petlju (u `main` funkciji):

```
Igra igra;
while (!igra.dohvatiProzor() -> jelGotov()) {
    igra.obradiUlaz();
    igra.update();
    igra.renderiraj();
    igra.restartSata();
}
```

Dopuna konstruktora klase Igra

```
...
#include "Zmija.h"

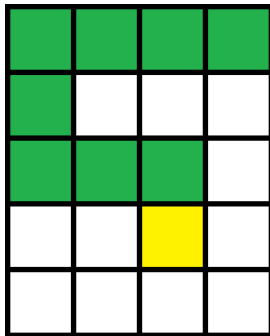
class Igra {
    ...
private:
    Svijet svijet;
    Zmija zmija;
    ...
};

Igra::Igra() : p("Zmija", sf::Vector2u(800, 640)),
              svijet(16, p.dohvatiVelicinu()),
              zmija(svijet.dohvatiVBloka()) {}
```

- u funkciji `obradiUlaz` mogli bismo napisati ovakve `if`-ove:

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {  
    zmija.PostaviSmjer(Smjer::Gore);  
}
```

- Problem:** sa slike vidimo da se trenutno zmija giba prema dolje (Zašto?) - ako korisnik pritisne tipku `Up`, zmija će se na takav čudan način zaletiti sama u sebe (tako nešto ne želimo)

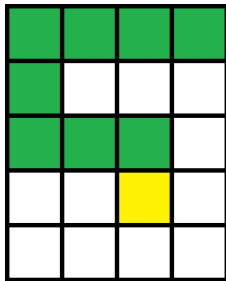


Dopuna funkcije Igra :: obradiUlaz (nastavak)

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)
    && zmiya.DohvatiSmjer() != Smjer::Dolje) {
    zmiya.PostaviSmjer(Smjer::Gore);
}
```

Problem: Što će se dogoditi ako se za situaciju prikazanu na slici unutar istog vremenskog intervala za jedan korak zmije pojave sljedeći događaji:

- (1.) u jednom prolasku glavne petlje pritisak na tipku `Right`,
- (2.) u sljedećem prolasku glavne petlje pritisak na tipku `Up`?



Rješenje navedenog problema

- na prethodnoj slici mogli smo vidjeti u kojem se smjeru gibala zmija
- ⇒ napisat ćemo funkciju `dohvatiFizickiSmjer` koja nam ne daje smjer koji je spremljen u varijabli `smjer`, nego smjer zmiye dobiven promatranjem položaja glave u odnosu na njen vrat

Dodamo u datoteku **Zmija.h**:

```
class Zmija {  
    public:  
        Smjer dohvatiFizickiSmjer();  
        ...  
};
```

Funkcija Zmija::dohvatiFizickiSmjer

- dodamo u datoteku **Zmija.h**:

```
Smjer Zmija::dohvatiFizickiSmjer() {
    if (koordinate.size() == 1)
        return Smjer::Nema;
    //odredimo razliku koordinata glave i vrata
    auto razlika = koordinate[0] - koordinate[1];
    if (razlika == sf::Vector2i(0, 1))
        return Smjer::Dolje;
    if (razlika == sf::Vector2i(0, -1))
        return Smjer::Gore;
    if (razlika == sf::Vector2i(1, 0))
        return Smjer::Desno;
    return Smjer::Lijevo; //else (razlika (-1,0))
}
```

Napomena: za duljinu 1 možemo bilo kamo (pa važno vratiti nešto različito od Gore, Dolje, Lijevo, Desno).

Kod funkcije obradiUlaz (u datoteci Igra.h)

```
void Igra::obradiUlaz() {  
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)  
        && zmija.dohvatiFizickiSmjer() != Smjer::Dolje) {  
        zmija.PostaviSmjer(Smjer::Gore);  
    } else if  
    (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)  
        && zmija.dohvatiFizickiSmjer() != Smjer::Gore) {  
        zmija.PostaviSmjer(Smjer::Dolje);  
    } else if  
    (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)  
        && zmija.dohvatiFizickiSmjer() != Smjer::Desno) {  
        zmija.PostaviSmjer(Smjer::Lijevo);  
    }  
}
```

(nastavak koda je na sljedećem slajdu)

Kod funkcije obradiUlaz (nastavak)

```
else if
    (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)
    && zmija.dohvatiFizickiSmjer() != Smjer::Lijevo) {
    zmija.PostaviSmjer(Smjer::Desno);
}
}
```

Dopunjena funkcija update

```
void Igra::update() {  
    p.update();  
    float vrijemeIteracije = 1.0f /  
        zmija.DohvatiBrzinu();  
    if (vrijeme.asSeconds() >= vrijemeIteracije) {  
        zmija.Korak();  
        svijet.Update(zmija);  
        vrijeme -= sf::seconds(vrijemeIteracije);  
        if (zmija.JelIzgubio()) {  
            zmija.Reset();  
        }  
    }  
}
```

Dopunjena funkcija renderiraj

```
void Igra::renderiraj() {  
    p.ocisti();  
    svijet.Renderiraj(&p);  
    zmiya.Renderiraj(&p);  
    p.prikazi();  
}
```