SFML - Zmija (nastavak)

Objektno programiranje - 8. vježbe

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

14. svibnja 2025. godine

Nastavljamo rad na kodu s prošlih vježbi

- kod s prošlih vježbi može se preuzeti na web-stranici kolegija
- u tom kodu došli smo do pisanja funkcije `Reset` (u datoteci `Zmija.h`)

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

- naredbe koje bi inače stavili u konstruktor stavili smo u funkciju `Reset` jer će nam one trebati ne samo na početku nego i pri svakom ponovnom pokretanju igre (kad zmija izgubi sve živote ili se zabije u zid)
- prema tome, funkciju `Reset` ćemo osim u konstruktoru pozivati i kasnije (u funkciji `update` klase `Igra`)

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`

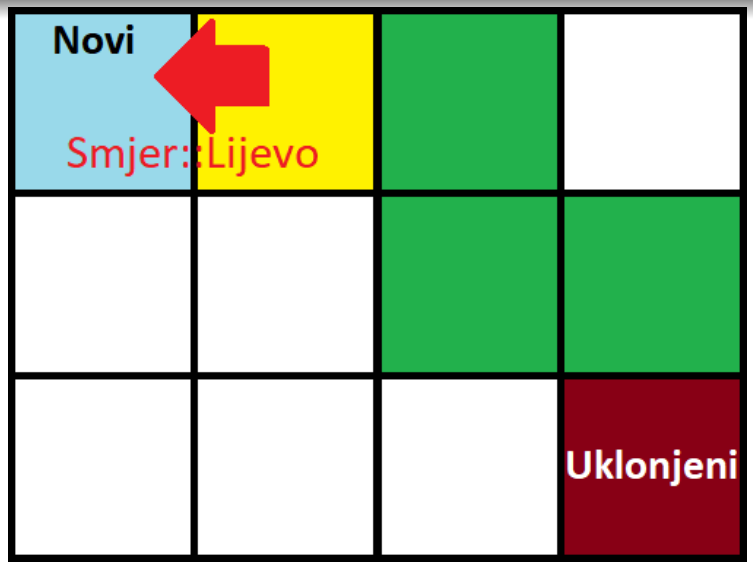
```
class Zmija {
  public:
    void Korak();
    ...
};

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

Funkcija `Pomakni`

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



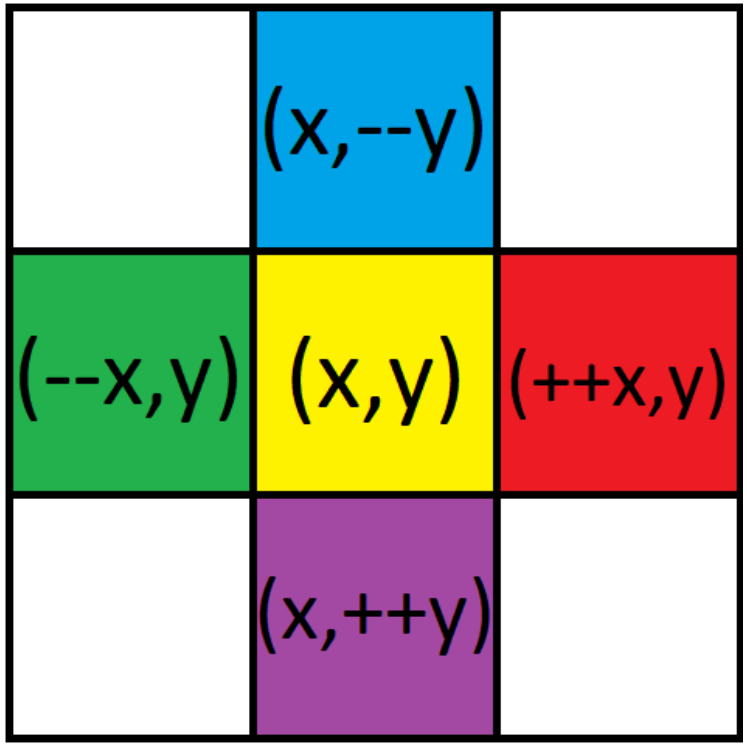

```
void Zmija::Pomakni() {
    sf::Vector2i novi = KoordinateGlave();
    koordUklonjenog = koordinate.back();
    koordinate.pop_back();
```

(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;
    }
    koordinate.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;
            }
}
```

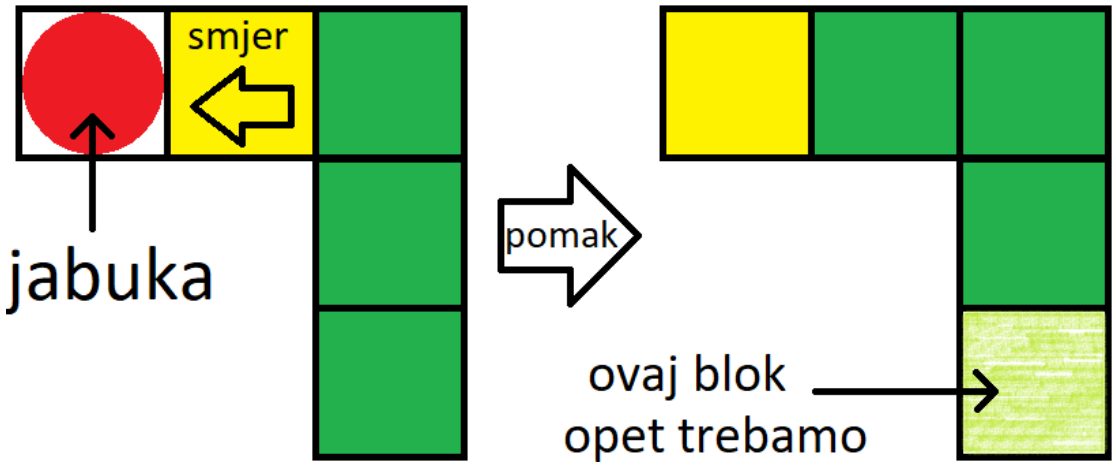
Funkcija `Odrezi`

- uklanjamo sve blokove zmije 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; i < velicina; ++i)
        koordinate.pop_back();
    --brZivota;
    if (brZivota == 0)
        Izgubio();
}
```

Funkcija `Produlji`

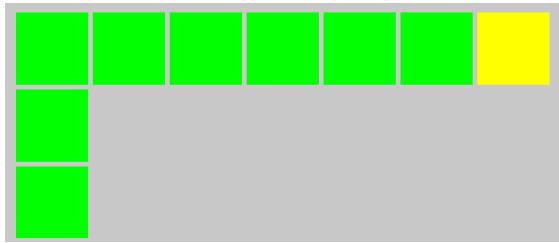
```
class Zmija {
  public:
    void Produlji();
    ...
};
```




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

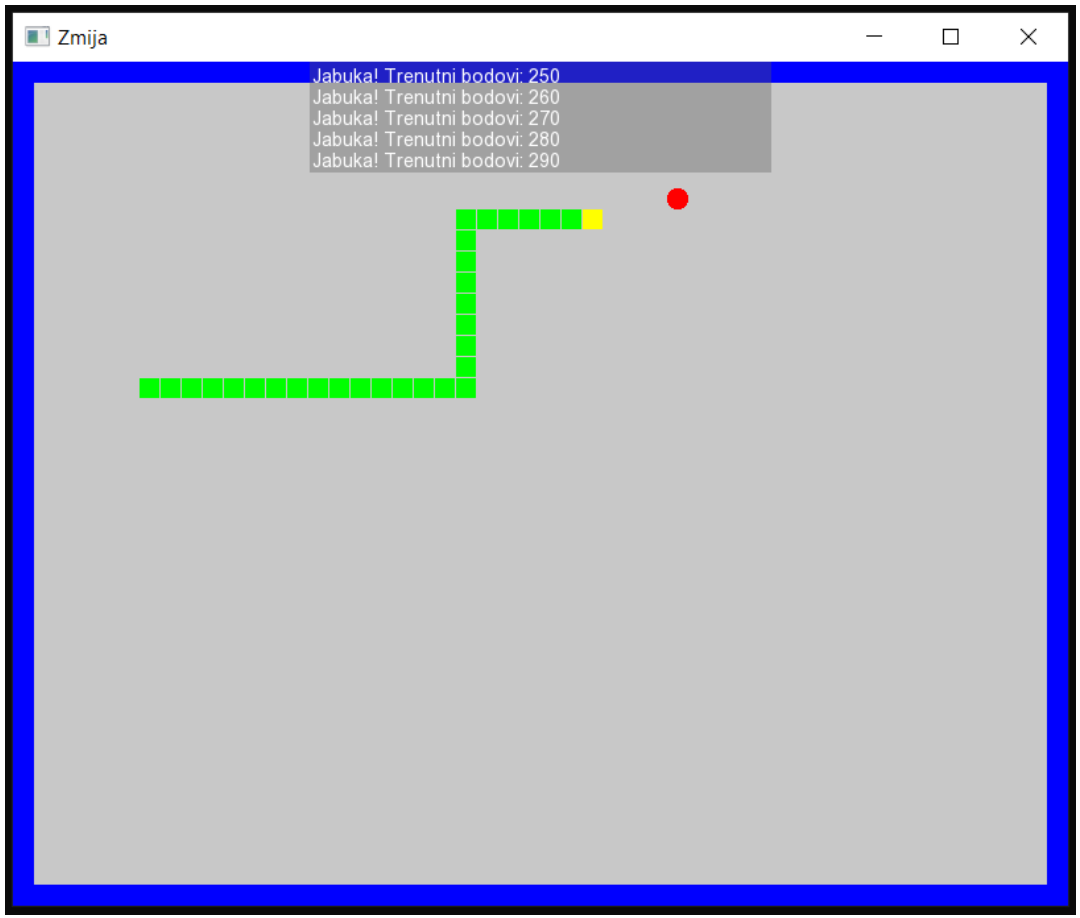
Crtanje zmije na ekran (koristimo `Prozor::crtaj`)

```
#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`**



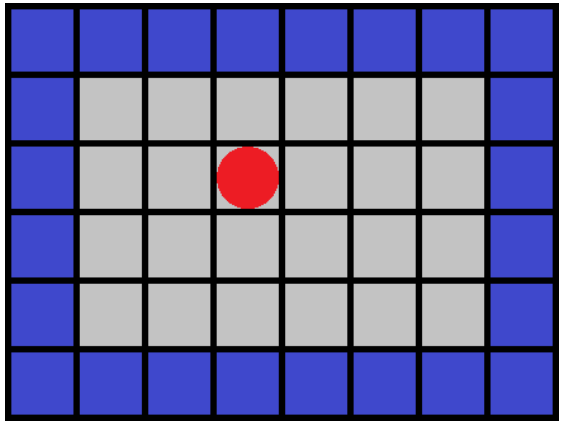
Klasa `Svijet` - što ćemo pamtiti

- 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() {}
```



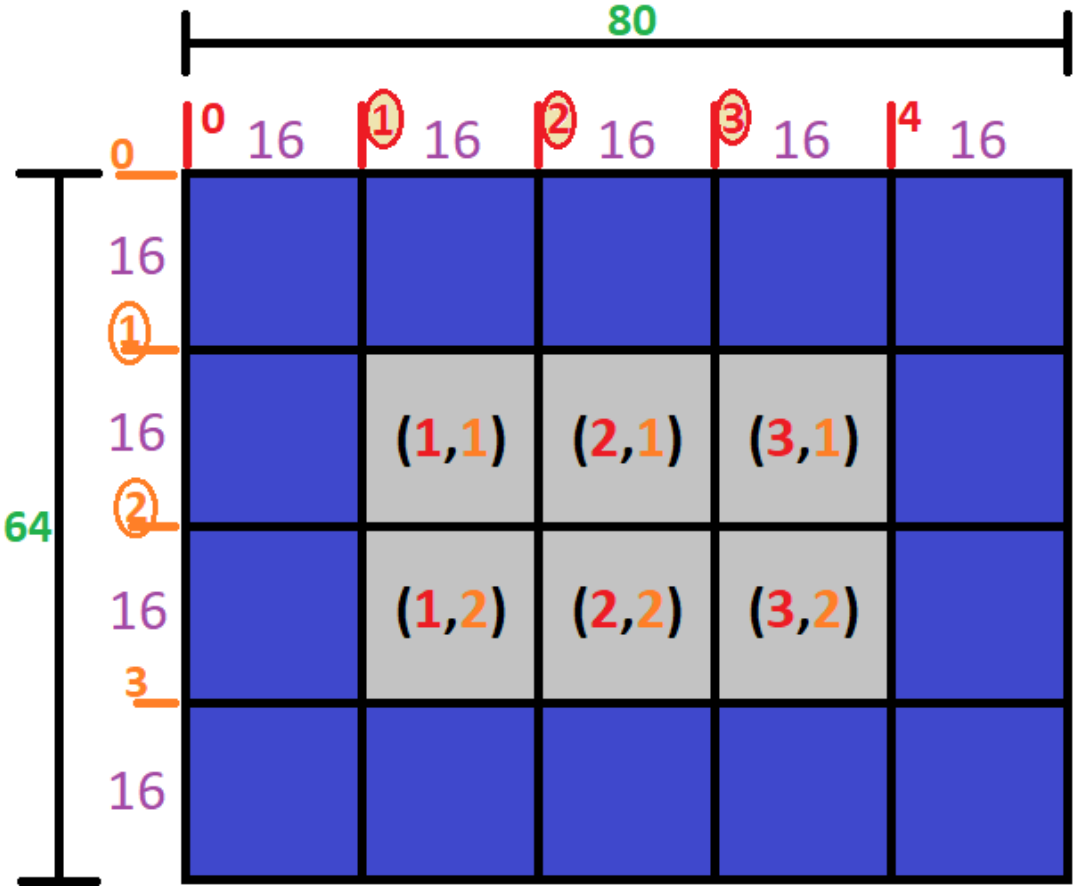
Postavljanje jabuke na prozoru

- početno, ali i nakon što zmija pojede jabuku (tj. glava zmije 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” $(x, y) \in \{(1,1), \ldots, (3,2)\}$

- $80/16 = 5 \rightarrow$ 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`

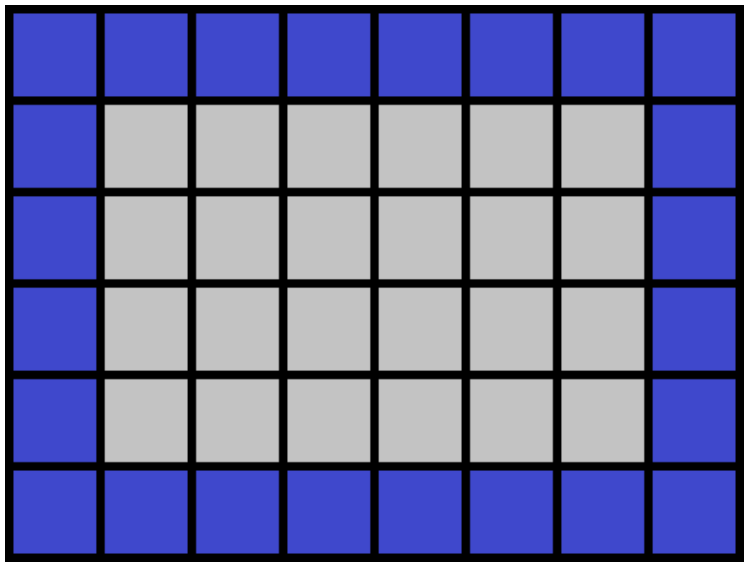
- u svakom koraku, zmija se pomakne i može detektirati ako se zabije sama u sebe
- no, treba pogledati: je li zmija pomicanjem pojela jabuku ili se zabila u zid (taj kod je na sljedećem slajdu)

```
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();
}
```



Funkcija `Renderiraj`

- 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`

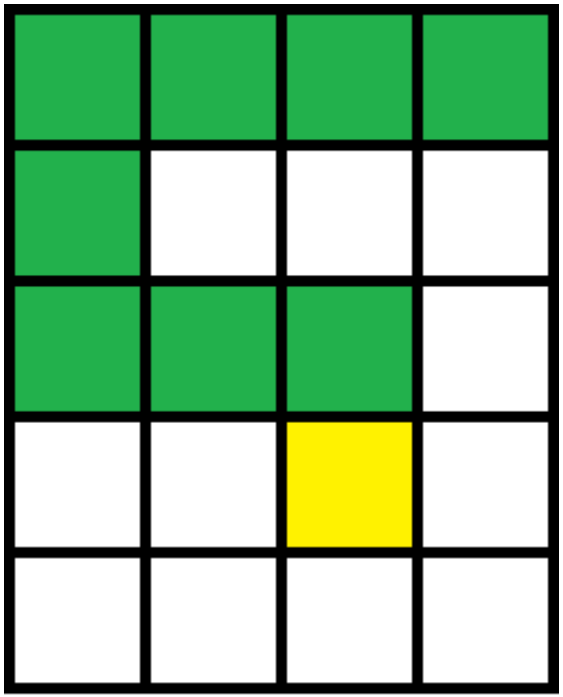
```
...
#include "Zmija.h"

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

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

Dopuna funkcije `Igra::obradiUlaz`

- 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)



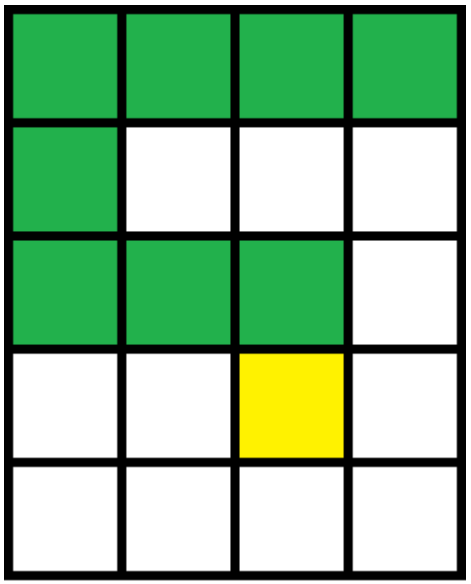
Dopuna funkcije `Igra::obradiUlaz` (nastavak)

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)
    && zmija.DohvatiSmjer() != Smjer::Dolje) {
        zmija.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 zmije 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 je važno vratiti nešto različito od Gore, Dolje, Lijevo, Desno, poput Nema).

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`

- pozivamo odgovarajuće funkcije `Renderiraj` za zmiju i svijet (za crtanje jabuke i plavog zida)
- argument tih funkcija je tipa `Prozor*` pa šaljemo adresu prozora `p` (prozora na koji će se zmija, jabuka i zid nacrtati)

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

Napomena: Reference umjesto pokazivača

- mogli smo umjesto pokazivača u funkcijama za renderiranje koristiti reference (čime bi dobili jednostavniji kod)

Primjer. Potrebne promjene u kodu ako bi funkcija `Renderiraj` klase `Svijet` primala referencu umjesto pokazivača na prozor:

```
class Svijet {    // u "Igra.h"
    ...
    void Renderiraj(Prozor&);
    ...
};
void Svijet::Renderiraj(Prozor& p) {
    p.crtaj(rub);
    p.crtaj(jabuka);
}
void Igra::renderiraj() {    // u "Igra.h"
    ...
    svijet.Renderiraj(p);
    ...
}
```