Osnovi računarskih sistema

Jelena Tomašević i Milena Vujošević–Janičić

Čas 1

Vežbe—— 08.04.2005

1.1 Life

Zadatak 1 *Prisetimo se stare igre Život (Life).*
Na tabli sa kvadratnim poljima postavljaju se figure. U svakoj iteraciji figure ***"preživljavaju"****,* ***"umiru"****, ili* ***"se rađaju"****, zavisno od broja susednih figura. Figura je susedna ako se nalazi na jednom od 8 (osam) susednih polja. Naredna iteracija se formira u celosti na osnovu rasporeda figura u prethodnoj iteraciji i to:*

- *figura "preživljava" ako ima dve ili tri susedne figure;*
- *figura "umire" od prenaseljenosti ako u susedstvu ima više od tri, a od usamljenosti ako u susedstvu ima manje od dve druge figure;*
- *ako u susedstvu praznog polja table postoje tacno tri figure, na tom mestu se "rađa" nova figura.*

Napisati klasu `IgraLife` *koja kao podatak član sadrži tablu za igru. Tabla je "okrugla", tj. ako figura izađe na jednu stranu, pojavljuje se sa suprotne strane table. Metod* `Iteracija()` *formira narednu iteraciju, uklanjajući i figure koje nisu preživele i kreirajući one koje su rođene (najpre se za sva polja matrice izračunava da li figura koja se nalazi na tom polju preživljava ili ukoliko je polje prazno da li se na tom polju rađa nova figura, a zatim se formira nova matrica koja predstavlja narednu iteraciju i izvodi se premeštanje figura sa matrice koja predstavlja prethodnu iteraciju na matricu koja predstavlja narednu iteraciju i kreiranje novorođenih figura i uklanjanje figura koje nisu preživele; na kraju se formirana nova iteracija proglašava za aktuelnu, a matrica sa prethodnom iteracijom se uklanja.)*

Napisati operator prosleđivanja:
`ostream& operator << ( ostream&, IgraLife& igra )`
koji ispisuje tablu za igru, pri čemu se prazna polja označavaju tačkom, a figure onako kako je to za njih definisano.

Napisati operator izdvajanja:
`istream& operator >> ( istream&, IgraLife& igra )`
koji čita tablu za igru i pri tome generiše potreban raspored figura na njoj, uz pretpostavku da je veličina table odgovarajuća.

U prvom redu datoteke `life.dat` *zapisana su dva cela broja: širina table i visina table. Nakon toga, od početka drugog reda, sledi zapis same table i rasporeda figura. Pretpostaviti da se figura označava znakom* `'x'` *a prazno polje sa* `'.'`*. Na primer:*

```
10 8
. . . . . . . . . .
. . . . . . . . . .
. . . . . x x . . .
. . . . . x . x . .
. . . . . x . . . .
. . . . . . . . . .
. x x x . . . . . .
. . . . . . . . . .
```

Napisati program koji na osnovu sadržaja datoteke `life.dat` *formira tablu za igru i potrebne figure, a zatim omogućava praćenje iteracija tako što ispiše iteraciju na standardnom izlazu i zatim čeka na pritisak odgovarajućeg tastera: bilo koji taster označava narednu iteraciju i ispis table na izlaz a* `k` *je kraj.*

Rešenje:

```
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

//-------------------------------------------------------------
template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v )
        : _Podaci(s)
        {
        for( unsigned i=0; i<s; i++ )
            _Podaci[i].resize(v);
        }

    const vector<T>& operator[] ( unsigned i ) const
        { return _Podaci[i]; }

    vector<T>& operator[] ( unsigned i )
```

```
        { return _Podaci[i]; }

    unsigned Sirina() const
        { return _Podaci.size(); }

    unsigned Visina() const
        { return _Podaci.size() ? _Podaci[0].size() : 0; }

private:
    vector< vector<T> > _Podaci;
};

//-------------------------------------------------------------
class IgraLife
{
public:

    //Konstruktor bez argumenata
    IgraLife()
        {}

    // Metod BrojSuseda()izracunava broj figura
    // na 8 susednih pozicija u odnosu
    // na poziciju (x,y) kako bi se koristeci tu
    // informaciju ispitalo da li figura
    // prezivljava ili se mozda na tom mestu radja nova figura.
    int BrojSuseda( int x, int y ) const
        {
        int bs = 0;
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();
        for( int i=-1; i<=1; i++ )
            for( int j=-1; j<=1; j++ )
                if( i || j ){
                    int xx = (x+i+sirina) % sirina;
                    int yy = (y+j+visina) % visina;
                    if( _Tabla[xx][yy] )
                        bs++;
                }
        return bs;
        }

    // Metod koji predstavlja sledecu iteraciju.
    void Iteracija()
```

```
        {
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();

        //Kreira se nova tabla.
        Matrica<bool> novaTabla( sirina, visina );

        //Za svako polje stare table...
        for( int i=0; i<sirina; i++ )
            for( int j=0; j<visina; j++ )
                {
            //...izracunava se broj suseda...
                int bs = BrojSuseda( i, j );
            //...i ukoliko na tom polju postoji figura onda ona prezivljava
            // ukoliko je (bs>=2 && bs<=3), a ako ne postoji figura onda se
            // ona radja ukoliko je (bs==3).
                novaTabla[i][j] =
                    _Tabla[i][j] ? (bs>=2 && bs<=3) : (bs==3);
            }
        _Tabla = novaTabla;
        }

    void Ucitaj( istream& istr )
        {
        int sirina, visina;
        istr >> sirina >> visina;

        // Matrica<bool> a( sirina, visina );
        // _Tabla = a;
        _Tabla = Matrica<bool>( sirina, visina );

        for( int i=0; i<visina; i++ )
            for( int j=0; j<sirina; j++ ){
                char c;
                istr >> c;
                //Ako je (c=='x') onda je _Tabla[j][i]=true sto znaci da tu
                //postoji figura, inace je false.
                _Tabla[j][i] = (c=='x');
            }
        }

    void Zapisi( ostream& ostr ) const
        {
        int sirina = _Tabla.Sirina();
```

```
        int visina = _Tabla.Visina();

        ostr << sirina << ' ' << visina << endl;
        for( int i=0; i<visina; i++ ){
            for( int j=0; j<sirina; j++ )
                ostr << (_Tabla[j][i] ? 'x' : '.');
            ostr << endl;
            }
        }

private:

    // S obzirom da za svako polje table razlikujemo dva slucaja (figura
    // postoji ili ne postoji) tablu mozemo interpretirati kao matricu
    // logickih vrednosti:
    // _Tabla[x][y]=true ako se na tom polju nalazi figura;
    // _Tabla[x][y]= false inace.

    Matrica<bool> _Tabla;
};

//-------------------------------------------------------------
istream& operator >> ( istream& istr, IgraLife& igra )
{
    igra.Ucitaj( istr );
    return istr;
}

ostream& operator << ( ostream& ostr, const IgraLife& igra )
{
    igra.Zapisi( ostr );
    return ostr;
}

//-------------------------------------------------------------
main()
{
    IgraLife igra;
    ifstream f( "life.dat");
    f >> igra;
    cout << igra;
    while(1){
        char c;
        cin >> c;
```

```
        if( c=='k')
            break;
        igra.Iteracija();
        cout << igra;
    }
    return 0;
}
```

Zadatak 2 *Modifikujmo igru uvođenjem dve vrste figura:*

- *čekalice miruju na jednom polju od rođenja do smrti (poput svih figura u uobičajenoj igri);*
- *šetalice bezuslovno preživljavaju.*

Napisati apstraktnu klasu `Jedinka` *i njene naslednice* `Cekalica`, `Setalica` *i* `PraznoPolje`. *Napisati klasu* `Tabla` *koja predstavlja tablu za igru a realizovati je kao matricu pokazivača na Jedinke.*
U klasi Jedinka obezbediti:

- *metod za ispitivanje da li jedinka preživljava,*
- *metod koji vraća pokazivač na novu jedinku,*
- *metod koji vraća izgled jedinke (čekalicu označiti sa 'x', šetalicu sa 's', a prazno polje sa '.').*

Ukoliko se na praznom polju "rađa" nova figura, koja će biti rođena utvrđuje se slučajnim izborom.

Rešenje:

```
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;


//-------------------------------------------------------------
template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v )
        : _Podaci(s)
```

```
        {
        for( unsigned i=0; i<s; i++ )
            _Podaci[i].resize(v);
        }

    const vector<T>& operator[] ( unsigned i ) const
        { return _Podaci[i]; }

    vector<T>& operator[] ( unsigned i )
        { return _Podaci[i]; }

    unsigned Sirina() const
        { return _Podaci.size(); }

    unsigned Visina() const
        { return _Podaci.size() ? _Podaci[0].size() : 0; }

private:
    vector< vector<T> > _Podaci;
};

//---------------------------------------------------------
class Jedinka;

class Tabla : public Matrica<Jedinka*>
{
public:
    Tabla( int s, int v )
        : Matrica<Jedinka*>(s,v)
        {}

    Tabla()
        : Matrica<Jedinka*>()
        {}

    int BrojSuseda( int x, int y ) const;
};


//---------------------------------------------------------
class Jedinka
{
public:
    virtual bool Prezivljava( const Tabla& tabla, int& x, int& y ) = 0;
```

```
    virtual Jedinka* NovaJedinka();
    virtual char Izgled() const = 0;
};


//-------------------------------------------------------------
class Cekalica : public Jedinka
{
public:
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
        {
        int bs = tabla.BrojSuseda( x, y );
        return bs>=2 && bs<=3;
        }
    Jedinka* NovaJedinka()
        { return this; }
    char Izgled() const
        { return 'x'; }
};

// Kreira se po jedan globalni objekat svakog primerka Jedinke
// i polja u tabli su zapravo pokazivaci na te objekte.
Cekalica cekalica;

//-------------------------------------------------------------
class Setalica : public Jedinka
{
public:
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
        { return true; }
    Jedinka* NovaJedinka()
        { return this; }
    char Izgled() const
        { return 's'; }
};

Setalica setalica;

//-------------------------------------------------------------
class PraznoPolje : public Jedinka
{
public:
    // Prazno polje prezivljava ako ce se tu u narednoj iteraciji
    // roditi nova figura.
```

```
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
        {
        int bs = tabla.BrojSuseda( x, y );
        return bs==3;
        }

    Jedinka* NovaJedinka()
        {
        // Izbor radjanja nove figure na praznom polju
        // je slucajan.
        // Funkcija random vraca slucajan broj iz
        // intervala [0, 100)
        int r = random(100);
        Jedinka* p;
        if( r<50 )
            p = &setalica;
        else
            p = &cekalica;
        return p;
        }

    char Izgled() const
        { return '.'; }
};

PraznoPolje praznoPolje;

//------------------------------------------------------------
int Tabla::BrojSuseda( int x, int y ) const
{
    int bs = 0;
    int sirina = Sirina();
    int visina = Visina();
    for( int i=-1; i<=1; i++ )
        for( int j=-1; j<=1; j++ )
            if( i || j ){
                int xx = (x+i+sirina) % sirina;
                int yy = (y+j+visina) % visina;
                if( (*this)[xx][yy] != &praznoPolje )
                    bs++;
            }
    return bs;
}
```

```
//-------------------------------------------------------------
class IgraLife
{
public:
    IgraLife()
        {}

    // Metod potreban prilikom ucitavanja
    // i inicijalizacije table
    Jedinka* NovaJedinka( char c )
        {
        switch( c ){
            case 's':
                return &setalica;
            case 'x':
                return &cekalica;
            default:
                return &praznoPolje;
            }
        }

    void Iteracija()
        {
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();

        // Pravimo novu tablu cije pokazivace
        // inicijalizujemo na 0.
        Tabla novaTabla( sirina, visina );
        for( int i=0; i<sirina; i++ )
            for( int j=0; j<visina; j++ )
                novaTabla[i][j] = 0;

        // Racunaju se vrednosti nakon tekuce iteracije.
        for( int i=0; i<sirina; i++ )
            for( int j=0; j<visina; j++ ){
                int x = i;
                int y = j;
                Jedinka* a = _Tabla[x][y];
                if( a->Prezivljava(_Tabla,x,y) )
                    novaTabla[x][y] = a->NovaJedinka();
                else
                    novaTabla[i][j] = &praznoPolje;
            }
```

```
        // U tablu se upisuje novo stanje
        _Tabla = novaTabla;
        }

    void Ucitaj( istream& istr )
        {
        int sirina, visina;
        istr >> sirina >> visina;
        _Tabla = Tabla( sirina, visina );
        for( int i=0; i<visina; i++ )
            for( int j=0; j<sirina; j++ ){
                char c;
                istr >> c;
                _Tabla[j][i] = NovaJedinka(c);
            }
        }

    void Zapisi( ostream& ostr ) const
        {
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();

        ostr << sirina << ' ' << visina << endl;
        for( int i=0; i<visina; i++ ){
            for( int j=0; j<sirina; j++ )
                ostr << _Tabla[j][i]->Izgled();
            ostr << endl;
            }
        }

private:
    Tabla _Tabla;
};


istream& operator >> ( istream& istr, IgraLife& igra )
{
    igra.Ucitaj( istr );
    return istr;
}

ostream& operator << ( ostream& ostr, const IgraLife& igra )
{
```

```
    igra.Zapisi( ostr );
    return ostr;
}


//-------------------------------------------------------
main()
{
    IgraLife igra;
    ifstream f( "life.dat");
    f >> igra;
    cout << igra;
    while(1){
        char c;
        cin >> c;
        if( c=='k')
            break;
        igra.Iteracija();
        cout << igra;
    }
    return 0;
}
```

Zadatak 3 *Uvedimo sada još jednu vrstu figura koje ćemo nazvati* `Skakalica` *i definišimo preciznije ponašanje svake od figura.*

Naime, u svakom koraku šetalica mora da pređe na jedno od četiri, slučajno izabrana, susedna prazna polja (levo, desno, gore ili dole). Ako ni jedno od ovih polja nije prazno, ona umire, inače preživljava. Ukoliko preživi, može se pretvoriti u čekalicu a može ostati i dalje šetalica.

Skakalice se ponašaju slično šetalicama, ali umesto da prelaze na susedno polje, one pokušavaju da preskoče jedno od osam susednih polja (skakalica S sme da skoči na polja označena sa X):

```
X   .   X   .   X
.   .   .   .   .
X   .   S   .   X
.   .   .   .   .
X   .   X   .   X
```

Ukoliko skakalica preživi može se pretvoriti u šetalicu.

Ukoliko bilo koje dve figure stanu na isto polje u jednoj iteraciji, opstaje ona koja na to polje stigne poslednja.

Rešenje:

```
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

//-------------------------------------------------------------
template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v )
        : _Podaci(s)
        {
        for( unsigned i=0; i<s; i++ )
            _Podaci[i].resize(v);
        }

    const vector<T>& operator[] ( unsigned i ) const
        { return _Podaci[i]; }

    vector<T>& operator[] ( unsigned i )
        { return _Podaci[i]; }

    unsigned Sirina() const
        { return _Podaci.size(); }

    unsigned Visina() const
        { return _Podaci.size() ? _Podaci[0].size() : 0; }

private:
    vector< vector<T> > _Podaci;
};

//-------------------------------------------------------------
class Jedinka;

class Tabla : public Matrica<Jedinka*>
{
public:
    Tabla( int s, int v )
```

```
        : Matrica<Jedinka*>(s,v)
        {}

    Tabla()
        : Matrica<Jedinka*>()
        {}

    int BrojSuseda( int x, int y ) const;

    // U niz prazna[] upisuju se pozicije iz matrice pozicije[][] koje su prazne.
    int SlobodnaPolja( int x, int y, int pozicije[][2], int n, int prazna[] );
};


//-------------------------------------------------------------
class Jedinka
{
public:
    virtual bool Prezivljava( const Tabla& tabla, int& x, int& y ) = 0;
    virtual Jedinka* NovaJedinka();
    virtual char Izgled() const = 0;
};


//-------------------------------------------------------------
class Cekalica : public Jedinka
{
public:
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
        {
        int bs = tabla.BrojSuseda( x, y );
        return bs>=2 && bs<=3;
        }
    Jedinka* NovaJedinka()
        { return this; }
    char Izgled() const
        { return 'x'; }
};

Cekalica cekalica;

//-------------------------------------------------------------
class Setalica : public Jedinka
{
```

```
public:
    // Ispituje se da li setalica prezivljava i ako prezivljava
    // onda se na slucajan nacin bira jedno od praznih polja
    // na kojem ce u sledecoj iteraciji da zivi.
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
        {
        // Matrica pozicija na koje setalica moze da stigne
        // ukoliko su ta polja prazna
        int pozicije[][2] = { {0,1}, {0,-1}, {1,0}, {-1,0} };
        int prazna[4];

        // Niz prazna se puni pozicijama na koje setalica moze da stigne.
        // Na primer:
        // Ako je npraznih = 2 i prazna[0]=1 i prazna[1]=3
        // to znaci da postoje dve prazne pozicije i to su
        // pozicije koje citamo iz matrice kao prvu {0,-1}
        // i kao trecu {-1,0} (u odnosu na polje sa kojeg se gleda)
        int npraznih = tabla.SlobodnaPolja( x, y, pozicije, 4, prazna );

        // Ako postoje prazna mesta onda u tom slucaju setalica prezivljava
        if( npraznih > 0 )
        {
            int s = tabla.Sirina();
            int v = tabla.Visina();

            // Generise se slucajan broj iz intervala [0, npraznih).
            int i = random(npraznih);

            // Obratiti paznju na to da su x i y preneti po referenci
            // pa da su na ovaj nacin izmenjene njihove vrednosti.
            x = (x + pozicije[prazna[i]][0] + s) % s;
            y = (y + pozicije[prazna[i]][1] + v) % v;
            return true;
        }
        else
            return false;
        }
    Jedinka* NovaJedinka()
        {
            // Setalica se moze u narednoj iteraciji
            // pretvoriti u cekalicu.
            if( random(100)<50 )
                return &cekalica;
            return this;
```

```
        }
    char Izgled() const
        { return 's'; }
};
Setalica setalica;

//-------------------------------------------------------------
class Skakalica : public Jedinka
{
public:
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
        {
        // Matrica pozicija na koje skakalica moze da stigne
        // ukoliko su ta polja prazna.
        int pozicije[][2] = { {-2,-2}, {-2,0}, {-2,2}, {0,-2},
                              {0,2}, {2,-2}, {2,0}, {2,2} };
        int prazna[8];

       // Niz prazna se puni pozicijama na koje skakalica moze da stigne.
         int npraznih = tabla.SlobodnaPolja( x, y, pozicije, 8, prazna );

        if( npraznih > 0 ){
            int s = tabla.Sirina();
            int v = tabla.Visina();
            int i = random(npraznih);
            x = (x + pozicije[prazna[i]][0] + s) % s;
            y = (y + pozicije[prazna[i]][1] + v) % v;
            return true;
        }
        else
            return false;
        }
    Jedinka* NovaJedinka()
        {
            //Skakalica se moze u narednoj iteraciji pretvoriti u setalicu.
            if( random(100)<50 )
                return &setalica;
            return this;
        }
    char Izgled() const
        { return 'k'; }
};

Skakalica skakalica;
```

```
//------------------------------------------------------------
class PraznoPolje : public Jedinka
{
public:
    bool Prezivljava( const Tabla& tabla, int& x, int& y )
        {
        int bs = tabla.BrojSuseda( x, y );
        return bs==3;
        }

    Jedinka* NovaJedinka()
        {

        // Prazno polje se moze u narednoj iteraciji
        // pretvoriti u cekalicu, setalicu ili skakalicu.
        int r = random(100);
        Jedinka* p;
        if( r<30 )
            p = &skakalica;
        else if( r<60 )
            p = &setalica;
        else
            p = &cekalica;
        return p;
        }

    char Izgled() const
        { return '.'; }
};

PraznoPolje praznoPolje;

//------------------------------------------------------------
int Tabla::BrojSuseda( int x, int y ) const
{
    int bs = 0;
    int sirina = Sirina();
    int visina = Visina();
    for( int i=-1; i<=1; i++ )
        for( int j=-1; j<=1; j++ )
            if( i || j ){
                int xx = (x+i+sirina) % sirina;
                int yy = (y+j+visina) % visina;
```

```
                if( (*this)[xx][yy] != &praznoPolje )
                    bs++;
            }
    return bs;
}

// U odnosu na koordinate x0 i y0, na osnovu niza pozicija koji
// ima n elemenata puni se niz prazna indeksima pozicija koje su na
// tabli slobodne.
int Tabla::SlobodnaPolja( int x0, int y0, int pozicije[][2], int n,
                                                    int prazna[] )
{
    int npraznih = 0;
    int s = Sirina();
    int v = Visina();
    for( int i=0; i<n; i++ ){
        int x = (x0 + pozicije[i][0] + s) % s;
        int y = (y0 + pozicije[i][1] + v) % v;
        if( (*this)[x][y] == &praznoPolje )
        //Postoji po jedan objekat svake jedinke tako da je ovo ispravno poredjenje.
        {
            prazna[ npraznih++ ] = i;
        }
    }
    return npraznih;
}

//-------------------------------------------------------------
class IgraLife
{
public:
    IgraLife()
        {}

    Jedinka* NovaJedinka( char c )
        {
        switch( c ){
            case 's':
                return &setalica;
            case 'x':
                return &cekalica;
            case 'k':
                return &skakalica;
            default:
```

```
            return &praznoPolje;
        }
    }

void Iteracija()
    {
    int sirina = _Tabla.Sirina();
    int visina = _Tabla.Visina();
    Tabla novaTabla( sirina, visina );

    // Ovim je obezbedjeno da kada setalica odseta
    // ili skakalica skoci da na polju na kome su bile u prethodnoj
    // iteraciji ostane prazno polje
    for( int i=0; i<sirina; i++ )
        for( int j=0; j<visina; j++ )
            novaTabla[i][j] = &praznoPolje;

    for( int i=0; i<sirina; i++ )
        for( int j=0; j<visina; j++ ){
            int x = i;
            int y = j;
            Jedinka* a = _Tabla[x][y];
            // Metod Prezivljava moze da promeni vrednosti
            // koordinata x i y cime se u novuTablu upisuje
            // nova jedinka na tako izracunato mesto.
            // Ovime se postize setanje i skakanje.
            if( a->Prezivljava(_Tabla,x,y) )
                novaTabla[x][y] = a->NovaJedinka();
        }
    _Tabla = novaTabla;
    }

void Ucitaj( istream& istr )
    {
    int sirina, visina;
    istr >> sirina >> visina;
    _Tabla = Tabla( sirina, visina );
    for( int i=0; i<visina; i++ )
        for( int j=0; j<sirina; j++ ){
            char c;
            istr >> c;
            _Tabla[j][i] = NovaJedinka(c);
        }
    }
```

```
    void Zapisi( ostream& ostr ) const
        {
        int sirina = _Tabla.Sirina();
        int visina = _Tabla.Visina();

        ostr << sirina << ' ' << visina << endl;
        for( int i=0; i<visina; i++ ){
            for( int j=0; j<sirina; j++ )
                ostr << _Tabla[j][i]->Izgled();
            ostr << endl;
            }
        }

private:
    Tabla _Tabla;
};


istream& operator >> ( istream& istr, IgraLife& igra )
{
    igra.Ucitaj( istr );
    return istr;
}

ostream& operator << ( ostream& ostr, const IgraLife& igra )
{
    igra.Zapisi( ostr );
    return ostr;
}


//------------------------------------------------------------
main()
{

    randomize();
    IgraLife igra;
    ifstream f( "life.dat");
    f >> igra;
    cout << igra;
    while(1){
        char c;
        cin >> c;
```

```
        if( c=='k')
            break;
        igra.Iteracija();
        cout << igra;
    }
    return 0;
}
```

Sadržaj