

Fakulteta za Matematiko in Fiziko v Ljubljani

Problem 8-ih kraljic

Opis razvoja algoritma pri predmetu Programiranje 3

Jakob Valič

Ljubljana, avgust 2018

Vsebina

Uvod	1
Problem	1
Način iskanja rešitev	1
Logika.....	2
Predstavitev šahovnice in figur na njej.....	2
Iskanje prostih polj	2
Postavljanje nove kraljice	3
Premikanje prejšnje kraljice	4
Grafični vmesnik.....	5
Risanje šahovnice in figur	5
Časovnika	5
Časovnik vseh rešitev	5
Časovnik za posamezno rešitev	6
Gumbi	6
Gumb za ustavitev vizualizacije	6
Prikaz delovanja.....	7
Prikaz vseh rešitev	7
Rešitev po korakih	7
Zaključek.....	9
Možne izboljšave	9
Viri	9

Uvod

Problem

Na polje velikosti $n \times n$ želimo postaviti n kraljic, tako da se med seboj ne bodo napadale. V našem primeru se bomo osredotočili na polje velikost 8×8 .

Način iskanja rešitev

Rešitve iščemo s pomočjo sestopanja ali vračanja (ang. backtracking). To je iterativni način, ki sistematično pregleduje vse možnosti. Algoritem se izvaja dokler ne postavimo vseh 8 kraljic na šahovnico. Drži se dveh preprostih pravil:

1. Kraljico v novi vrstici postavimo v prvi prost stolpec.
2. Ko v novi vrstici ni nobenega prostega mesta za kraljico, začnemo premikati kraljico v zgornji vrstici do prvega prostega polja. Če v koraku 2 ne dobimo prostega polja, ga ponovimo (torej prestavljamo kraljico v višji vrstici). Ko korak 2 uspe, nadaljujemo s korakom 1.

Logika

Dobro načelo programiranja je, da je logika aplikacije neodvisna od grafičnega vmesnika. Tudi v našem primeru smo ločili delovanje logike in grafičnega prikaza. Logika poskrbi za izračun vseh rešitev problema, ki jih je 92. Shrani jih v seznam `vseRešitve`. Na zahtevo uporabnika, ki pritisne na gumb, logika izračuna posamezne korake rešitev in jih shrani v seznam `korakiPosamezneRešitve`. Oba navedena seznama sta seznama, v katerem hranimo šahovnice. Pri korakih posamezne rešitve logika za vsak korak shrani tudi opis koraka, in sicer v seznam nizov z imenom `opisKorakovPosamezneRešitve`.

Predstavitev šahovnice in figur na njej

Šahovnico predstavimo z dvodimenzijskim seznamom števil, torej `int[,]`. Prva komponenta predstavlja vrstice, druga stolpce. Obojih je 8, potekajo pa od števila 0 do 7. Šahovnica ima torej 64 polj. Vasako polje ima bodisi vrednost 0, 1, 2 ali 3. Ta števila izberemo glede na ustrezni grafični prikaz kraljic.

- Vrednost 0 pomeni, da na polju ni kraljice.
- Vrednost 1 pomeni, da je na polju kraljica. Prikažemo jo z belo kraljico.
- Vrednost 2 pomeni, da na polje ne smemo dati kraljice, ker je polje napadeno od ene druge že obstoječe kraljice. Na tem polju se bo prikazala rdeča kraljica.
- Vrednost 3 označuje kraljico, ki napada, torej preprečuje novi kraljici, da bi jo lahko postavili na ploščo. Takih kraljic je lahko na plošči več.

Osnovna šahovnica, na katero začnemo postavljati kraljice, je prazna.

```
int[,] sahovnica0 = {{0, 0, 0, 0, 0, 0, 0, 0},  
                    {0, 0, 0, 0, 0, 0, 0, 0},  
                    {0, 0, 0, 0, 0, 0, 0, 0},  
                    {0, 0, 0, 0, 0, 0, 0, 0},  
                    {0, 0, 0, 0, 0, 0, 0, 0},  
                    {0, 0, 0, 0, 0, 0, 0, 0},  
                    {0, 0, 0, 0, 0, 0, 0, 0},  
                    {0, 0, 0, 0, 0, 0, 0, 0}};
```

Slika 1: Začetna šahovnica

Iskanje prostih polj

Novo kraljico lahko postavimo le na prosto polje. Polje je prosto, če ga nobena druga kraljica ne napada. Z drugimi besedami to pomeni, da v vrstici in stolpcu polja ni nobene druge kraljice. Prav tako ne sme biti kraljice na nobeni diagonali polja. Funkciji, ki išče dovoljena polja, določimo vektorje premikov kraljice, kot jih omogočajo šahovska pravila. Z danega polja se pomikamo v smeri vseh vektorjev premikov tako dolgo, dokler ne zadenemo ob drugo kraljico ali ne pademo iz šahovnice. Če smo zadeli ob kraljico vemo, da je polje napadeno, torej ni prosto. Hkrati smo izvedeli za koordinati napadajoče kraljice. Če v vseh smereh ne naletimo na nobeno drugo kraljico vemo, da je polje prosto. Zaradi narave algoritma bi nam bilo potrebno preverjati le vektorje premika naravnost gor, levo gor in desno gor. Funkcija `Dovoljeno_polje` nam vrne seznam kraljic, ki napadajo določeno polje. Če je seznam prazen, je polje prosto.

```

public static List<int[]> Dovoljeno_polje(int[,] sahovnica, int[] kraljica)
{
    List<int[]> pozicijaNapadalnihKraljic = new List<int[]>();
    if (sahovnica[kraljica[0], kraljica[1]] != 0) // Če polje, na katero želimo dati kraljico slučajno ni prosto.
    {
        pozicijaNapadalnihKraljic.Add(kraljica);
    }
    // 4 smeri premih premikov + 4 smeri diagonalnih premikov
    int[][] premiki = new int[][] { new int[] { 0, -1 }, new int[] { 0, 1 }, new int[] { 1, 0 }, new int[] { -1, 0 },
        new int[] { -1, -1 }, new int[] { -1, 1 }, new int[] { 1, -1 }, new int[] { 1, 1 } };
    int n = (int)Math.Sqrt(sahovnica.Length);
    foreach (int[] vektor_premika in premiki)
    {
        int i = kraljica[0];
        int j = kraljica[1];
        int i_vektorja_premika = vektor_premika[0];
        int j_vektorja_premika = vektor_premika[1];
        while (0 <= i + i_vektorja_premika && i + i_vektorja_premika < n && 0 <= j + j_vektorja_premika && j + j_vektorja_premika < n)
        {
            i = i + i_vektorja_premika;
            j = j + j_vektorja_premika;
            if (sahovnica[i, j] != 0)
            {
                pozicijaNapadalnihKraljic.Add(new int[] { i, j });
            }
        }
    }
    return pozicijaNapadalnihKraljic;
}

```

Slika 2: Funkcija za iskanje prostih polj

Postavljanje nove kraljice

Funkcija `Postavljaj_kraljice` v novi vrstici postavi kraljico na prvo prosto polje. Če tako polje obstaja, nanj postavi kraljico in gre postavljati novo kraljico v eno vrsto nižje. To doseže z rekurzivnim klicem. Kako funkcija ve, katera je zadnja vrstica s kraljico? Na pomoč ji priskoči funkcija `Poisci_zadnjo_kraljico`, ki vrne koordinate zadnje kraljice.

Če računamo korake posamezne rešitve, shranimo posamezen korak in opis koraka. Če računamo vse rešitve, tega ne storimo.

```

List<int[]> napadajoceKraljice = Dovoljeno_polje(sahovnica, new int[] { i, j });
if (napadajoceKraljice.Count == 0) // Dobili smo mesto, na katero lahko postavimo kraljico.
{
    sahovnica[i, j] = 1;
    if (shraniKorake)
    {
        ShraniSahovnico(sahovnica, korakiPosamezneRešitve);
        if (i == 7) // Postavili smo kraljico v zadnji vrstici
        {
            opisKorakovPosamezneRešitve.Add("Dobili smo rešitev. :");
        }
        else
        {
            opisKorakovPosamezneRešitve.Add("Prosto polje.");
        }
    }
    break;
}

```

Slika 3: Funkcija za postavljanje kraljic - dobili smo prosto polje

Če polje ni prosto, nanj ne moremo postaviti kraljice. Če iščemo korake posamezne rešitve, označimo napadajoče kraljice in kraljico, ki jo ne moremo postaviti na napadeno polje.

```

else if (shraniKorake) // Na želeno polje ne moremo postaviti kraljice.
    //Shranimo korak s ponazoritvijo nedovoljenega polja.
{
    označiNapadajočeKraljice(sahovnica, napadajočeKraljice);
    sahovnica[i, j] = 2; // Premikajočo kraljico obarvamo rdeče.
    ShraniŠahovnico(sahovnica, korakiPosamezneRešitve);
    opisKorakovPosamezneRešitve.Add("Napadeno polje.");
    sahovnica[i, j] = 0; // Ponastavimo
    odznačiNapadajočeKraljice(sahovnica, napadajočeKraljice);
}

```

Slika 4: Funkcija za postavljanje kraljic - polje ni prosto

Premikanje prejšnje kraljice

Če v posamezni vrstici ni nobenega prostega polja, na katero bi lahko postavili novo kraljico, začnemo premikati kraljico v prejšnji vrstici do prvega prostega polja. Če tako polje dobimo, znova pokličemo funkcijo za postavljanje kraljic. Če takega polja ni, začnemo premikati kraljico v eni vrstici višje, torej rekurzivno pokličemo funkcijo.

```

sahovnica[i, j_zadnje_kraljice] = 0; // Odmaknemo kraljico
for (int j = j_zadnje_kraljice + 1; j < 8; j++)
{
    List<int[]> napadajočeKraljice = Dovoljeno_polje(sahovnica, new int[] { i, j });
    if (napadajočeKraljice.Count == 0) // Nobena kraljice ne napada polja.
    {
        sahovnica[i, j] = 1; // Na prosto polje postavimo kraljico.
        return Postavljaj_kraljice(sahovnica, shraniKorake); // Postavljamo naprej kraljice.
    }
    else if (shraniKorake)
    {
        označiNapadajočeKraljice(sahovnica, napadajočeKraljice);
        sahovnica[i, j] = 2; // Premikajočo kraljico obarvamo rdeče.
        ShraniŠahovnico(sahovnica, korakiPosamezneRešitve);
        opisKorakovPosamezneRešitve.Add("Napadeno polje.");
        sahovnica[i, j] = 0; // Ponastavimo
        odznačiNapadajočeKraljice(sahovnica, napadajočeKraljice);
    }
}
// Zanka se je iztekla in v vrstici nismo dobili polja, na katerega bi lahko postavili kraljico.
// Zato začnemo premikati kraljico, ki je v zgornji vrstici.
return Premakni_kraljico_za_eno_naprej(sahovnica, shraniKorake);

```

Slika 5: Funkcija za premikanje kraljice v prejšnji vrstici

Grafični vmesnik

Risanje šahovnice in figur

Šahovnico narišemo le enkrat. Rišemo jo z dvema zankama. V vrsticah izmenjujemo barve, prav tako v stolpcih. Pri risanju figur si zapomnimo okvir slike, v katero vstavimo kraljico določene barve. Okvir slike shranimo v seznam okvirjev slik. To je dobro zato, da lahko ob sledeči potezi stare slike pobrišemo.

```
private PictureBox DodajKraljico(int vrstica, int stolpec, int vrstaKraljice)
{
    Image slika;
    if (vrstaKraljice == 1) // Navadna, belo obarvana kraljica.
    {
        slika = Properties.Resources.kraljica_bela;
    }
    else if (vrstaKraljice == 2) // Napadena, rdeče obarvana kraljica.
    {
        slika = Properties.Resources.kraljica_rdeca;
    }
    else // Napadajoča, belo-rdeče obarvana kraljica.
    {
        slika = Properties.Resources.kraljica_bela_rdeca;
    }

    PictureBox okvirSlike = new PictureBox
    {
        BackColor = Color.Transparent,
        Size = new Size(razmik, razmik),
        Location = new Point((stolpec + 1) * 100, (vrstica + 1) * 100),
        Image = slika
    };
    Controls.Add(okvirSlike);
    return okvirSlike;
}
```

Slika 6: Funkcija za risanje šahovnice

Časovnika

Imamo dva časovnika, ki nam v intervalu 1 sekunde izrisujeta bodisi novo rešitev bodisi korak rešitve na zaslon. Interval lahko tudi ročno popravimo.

Časovnik vseh rešitev

Zaporedno prikazuje vse rešitve. Ko pride do zadnje rešitve, začne prikazovati rešitve od začetka. Skrbi za spreminjanje napisa zaporedne številke.

```
private void ČasovnikVseRešitve_Tick(object sender, EventArgs e)
{
    if (zaporednaRešitev >= vseRešitve.Count()) // Na zaslonu ohranimo zadnjo rešitev.
    {
        zaporednaRešitev = 1; // Začnemo gledati rešitve znova, od začetka.
    }
    OdstraniVseKraljice();
    časovnikKorakiRešitve.Enabled = false;
    DodajVseKraljice(vseRešitve[zaporednaRešitev]);
    spremeniNapisRešitve(zaporednaRešitev);
    zaporednaRešitev++;
}

private void spremeniNapisRešitve(int zaporednaRešitev)
{
    labelŠtevecRešitev.Text = String.Format("Rešitev številka: {0}", zaporednaRešitev.ToString());
}
```

Slika 7: Časovnik vseh rešitev

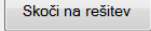
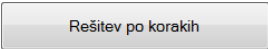
Časovnik za posamezno rešitev

Skrbi za prikaz korakov posamezne rešitve. Prav tako skrbi za prikaz opisa posameznega koraka. Korake posamezne rešitve oštevilči, da uporabnik ve, na katerem koraku se nahaja.

```
private void časovnikKorakiRešitve_Tick(object sender, EventArgs e)
{
    OdstraniVseKraljice();
    DodajVseKraljice(korakiRešitve[števecKorakovRešitve]); // Dodamo kraljice na šahovnico
    labelOpisKoraka.Text = opisKorakovRešitve[števecKorakovRešitve]; // Opis posameznega koraka rešitve
    labelŠtevecKorakov.Text = String.Format("Korak številka: {0}", števecKorakovRešitve);
    števecKorakovRešitve++;
    if (števecKorakovRešitve >= korakiRešitve.Count())
    {
        časovnikKorakiRešitve.Enabled = false;
        števecKorakovRešitve = 1;
        spremeniNapisRešitve(zaporednaRešitev);
        zaporednaRešitev++; // Prišli smo do naslednje rešitve
    }
}
```

Slika 8: Časovnik za posamezno rešitev

Gumbi

Z gumbi lahko uporabnik upravlja potek vizualizacije. Gumb  poskrbi, da se pomaknemo na izbrano rešitev. Gumb  poskrbi za prikaz naslednje rešitve po korakih.

Gumb za ustavitev vizualizacije

Zanimiv je gumb `Ustavi`, s katerim lahko zaustavimo vizualizacijo. Ta gumb zaustavi oba časovnika, če vsaj eden od njiju teče. Hkrati se mu napis spremeni v `Naslednja rešitev`. Isti gumb namreč uporabimo za nadaljevanje vizualizacije. Ob ponovnem pritisku na gumb se prikaže naslednja rešitev.

```
private void gumbUstavi_Click(object sender, EventArgs e)
{
    bool časovnikaUstavljena = !časovnikVseRešitve.Enabled && !časovnikKorakiRešitve.Enabled;
    if (!časovnikaUstavljena) // Ustavimo oba časovnika in ponastavimo vrednost napisov
    {
        časovnikVseRešitve.Enabled = false;
        časovnikKorakiRešitve.Enabled = false;
        gumbUstaviZacni.Text = "Naslednja rešitev";
    }
    else // Poženemo časovnik za prikaz naslednje rešitve.
    {
        časovnikVseRešitve_Tick(sender, e);
        časovnikVseRešitve.Enabled = true;
        časovnikKorakiRešitve.Enabled = false;
        gumbUstaviZacni.Text = "Ustavi";
    }
    ponastaviOznakeKorakov();
}
```

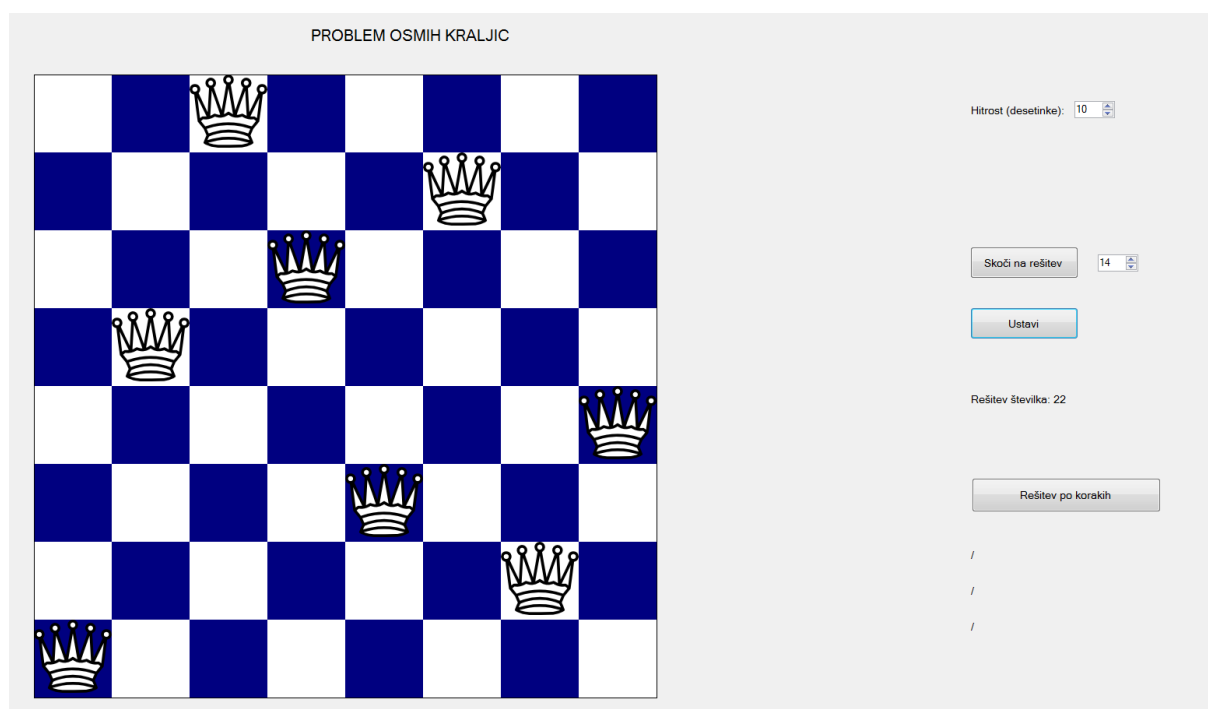
Slika 9: Gumb za ustavitev vizualizacije

Prikaz delovanja

Pri prikazu delovanja imamo v splošnem možnost prikaza zgolj rešitev ali prikaza posameznih korakov rešitve. Program začne prikaz vseh 92-ih rešitev. Lahko pa izberemo prikaz naslednje rešitve po korakih. Lahko tudi ustavimo trenutno pozicijo. Interval razmika med dvema slikama na šahovnici lahko spreminjamo. Privzeti interval je 1 sekunda.

Prikaz vseh rešitev

Zaporedno prikaže vseh 92 rešitev. Po zadnji prikaže spet prvo.



Slika 10: Rešitev 22

Rešitev po korakih

Na spodnji sliki smo najprej skočili na rešitev številka 14. Sedaj iščemo rešitev številka 15. Vseh korakov rešitve je 37. Trenutno se nahajamo na koraku 8. Vidimo, da kraljice v 7. vrstici ne moremo postaviti niti v zadnji, 8. stolpec. Zato jo bomo v koraku 9 umaknili s šahovnice in začeli premikati kraljico v 6. vrstici.

PROBLEM OSMIH KRALJIC

Hitrost (desetinke): 10

Skoči na rešitev 14

Ustavi

Iščemo rešitev št. 15

Rešitev po korakih

Vseh korakov: 37

Korak številka: 8

Napadeno polje.

Slika 11: Iskanje rešitve 15 po korakih

Zaključek

C# se mi zdi dober programski jezik za vizualizacijo algoritmov. Razlog je v tem, da ima dobro podporo za grafične elemente v urejevalniku Visual Studio. Izdelovati nalogo osmih kraljic v C# se mi je zdel zanimiv problem.

Možne izboljšave

- Popraviti bi bilo treba zaustavitveni pogoj za iskanje rešitev.
- Odpravili bi lahko utripanje, torej osveževanje vseh slik kraljic na šahovnici. Slike tistih kraljic, katerim se ne spremeni barva in pozicija, bi lahko pustili nedotaknjene, namesto da jih pobrišemo in na novo narišemo.
- Zanimivo bi bilo kodo prepisati v dogodkovno programiranje, da bi vsaka poteza v logiki povzročila določen prikaz na šahovnici.
- Poskrbeli bi lahko, da bi se velikost polj urejala dinamično glede na velikost okna aplikacije. V manjšem oknu bi bila polja manjša.
- Problem bi lahko reševali na poljubno veliki ($n \times n$) šahovnici.

Viri

- Idejo za vizualizacijo sem dobil na <http://eightqueen.becher-sundstroem.de/>.
- Sliko bele kraljice sem pridobil na spletni strani https://commons.wikimedia.org/wiki/File:Chess_qlt45.svg.
- Ikono za aplikacijo sem iz slike pretvoril na spletni strani <http://icoconvert.com/>.
- Kodo projekta sem napisal sam.