SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET

PROBLEM OSAM KRALJICA

Stefan Dunić

Mentor:

doc.dr.sc. Ani Grubišić

Split, listopad 2013.

Sadržaj

1 Uvod

Problem osam kraljica, kao i njegovo poopćenje, problem N kraljica veoma je poznat problem i u matematici i u računalnim znanostima. Problematikom su se još u 19. stoljeću počeli baviti poznati šahisti, a kasnije zbog zanimljivosti slučaja počeli su se baviti i matematičari i informatičari, na što ćemo se samo kratko osvrnuti u trećem poglavlju.

Temeljna ideja problema je kako smjestiti N kraljica na ploču dimenzija NxN tako da se one međusobno ne napadaju. Potrebno je naći sva moguća rješenja koja bi zadovoljila zadano pravilo. Za očekivati je da će se vrijeme rješavanja problema povećavati s porastom broja kraljica, što će se kasnije u radu i pokazati. Danas postoje razne metode rješavanja problema N kraljica ovisno o kompleksnosti samog problema, od rješavanja „na prste“ pa do modernog pristupa koristeći se heurističkim algoritmima.

Za svrhu ovog rada, odnosno za traženje rješenja problema osam kraljica koristi se rekurzivna metoda unatražnog pretraživanja (eng. backtracking). Stoga u drugom poglavlju odmah prije svega će se općenito objasniti pojam unatražnog pretraživanja, dok će se u četvrtom poglavlju prikazati kroz primjer problema četiri kraljice.

U petom poglavlju izvršiti će se implementacija unatražnog pretraživanja pomoću JavaScript programskog jezika na problem osam kraljica.

Kroz šesto poglavlje će se razjasniti razlika između ukupnog broja rješenja i jedinstvenog broja rješenja problema osam, a i općenitijeg, N kraljica.

U dodatku se nalazi kod napisan upotrebom JavaScript programskog jezika i osnovnih HTML naredbi.

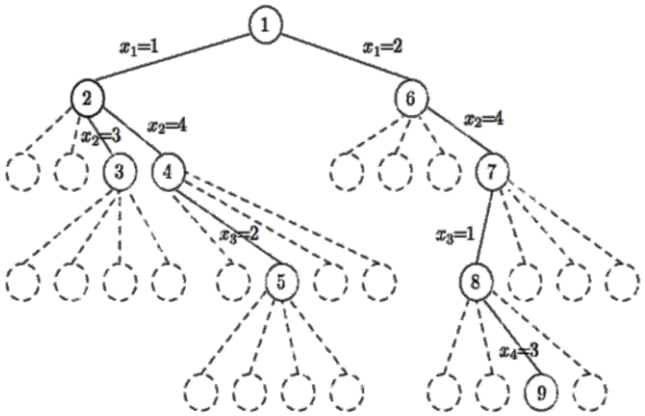
2 Unatražno pretraživanje (eng. Backtracking)

Unatražno pretraživanje je općeniti algoritam koji se temelji na sistematskom pretraživanju mogućih rješenja nekog računalnog problema. Kroz postupak pronalaženja pravih rješenja, algoritam stvara moguće kandidate dok istodobno uvjetovano odbacuje potencijalne kandidate rješenja koji ne mogu više na nikakav način stvoriti valjano rješenje. Samim time broj mogućih rješenja se smanjuje što omogućava brži pronalazak pravih rješenja.

Pozitivna strana upotrebe unatražnog pretraživanja jest to da se prije izvođenja algoritma dopušta postavljanje početnih uvjeta čime se u samom početku uvjetovano smanjuje broj mogućih rješenja.

Glavni zahtjev algoritma koji se treba ispuniti jest taj da mora postojati hijerarhija u sistematskoj proceduri traženja rješenja tako da se skup rješenja koji ne zadovoljava određeni uvjet odbaci prije nego li se stvori konačno potencijalno rješenje. Upravo zbog toga ispitivanje i stvaranje rješenja prati model stablastog prikaza. Korijen stabla predstavlja skup svih rješenja. Čvorovi u nižim razinama predstavljaju manje skupove izoliranih rješenja, a međusobno se razlikuju po svojstvima. Stablo se stvara kroz testiranje potencijalnih rješenja na način da se više ne stvaraju unaprijed odbačena rješenja (Slika 1). Kad se jedan čvor u stablu odbaci, cijelo njegovo podstablo je odbačeno i mi se vraćamo roditelju odbačenog čvora gdje testiramo mogućnost postojanja mogućeg rješenja na idućem djetetu.

Budući da metoda kroz proces testiranja stvara podrazine rješenja koje je teško obraditi, metoda sama po sebi nije toliko popularna (*Wikipedia, Backtracking; Engelhardt, 2007.*).



Slika 1 Stablasti prikaz algoritma unatražnog pretraživanja. Stablo se sastoji od korijena (kružnica s brojem 1), te potomskih čvorova (kružnice s brojevima 2 i 6) koji se i sami sastoje od svojih potomskih čvorova (kružnice s brojevima 3,4,7), itd. Iscrtkane kružnice predstavljaju one čvorove nakon kojih je algoritam unatražnog pretraživanja već unaprijed odbacio rješenja jer ne zadovoljavaju određene postavljene uvjete.

3 Problem N kraljica

Problem N kraljica je vrlo poznat matematički problem, no prva istraživanja se nisu mogla izvršiti sve do početka upotrebe jednostavnih algoritama.

Prvi poznati pokušaj davanja rješenja za osam kraljica 1848. godine je izdao tadašnji poznati šahist Max Bezzel pod pseudonimom Schachfreund (Slika 2: lijevo). Do 1854. godine predstavljeno je ukupno 40 rješenja od strane šahista. Nedugo nakon toga, problematika je postala popularna i drugoj vrsti publike, pa je tako 1850. godine Franz Nauck objavio i svoja rješenja. U svom radu on je dodao da postoje ukupno 60 rješenja glavnog problema. Kasnije se ispravio sa izjavom da ima ukupno 92 rješenja problema osam kraljica. U raspravu se uključio i Johann Carl Friedrich Gauss (Slika 2 desno) s idejom da postoje 72 rješenja, ali i da postoji mogućnost da ih ima još i više. Gauss je bio prvi koji je pokušao generalizirati problem osam kraljica na problem N kraljica. 1874. godine S. Günther je predložio metodu pronalaska rješenja problema osam kraljica koristeći se determinantama čiji je pristup kasnije uljepšao engleski matematičar J. W. L. Glaisher. U raspravu se kasnije uključilo i još nekoliko matematičara, ali nažalost ima jako malo zapisa o njihovoj međusobnoj raspravi (*Campbell, 1977.*).

Novi zamah u istraživanju problema N kraljica uvelike su napravila dostupnost računala 1960-tih godina. Edsger Djikstra 1972. godine je iskoristio problem osam kraljica da ilustrira moć onog što je on tada zvao strukturno programiranje. On je objavio veoma detaljan opis razvoja „depth-first backtracking algorithm“ (dubinsko unatražno pretraživanje) (*Campbell, 1997.*).

Danas se problem osam kraljica, kao i njegovo poopćenje na problem N kraljica, koristi u računalnim znanostima i često se upotrebljava kao praktična vježba. Najčešća upotreba problema je kao klasičan kombinatorni problem u području umjetne inteligencije, a pogotovo u slučaju prikazivanja algoritma unatražnog pretraživanja.

Slika 2 Lijevo: Max Bezzel, poznati njemački šahist iz 19. stoljeća. Desno: Johann Carl Friedrich Gauss, poznati njemački znanstvenik iz 19. stoljeća.

4 Primjena unatražnog pretraživanja na problem četiri kraljice

Za očekivati je da kompleksnost traganja za rješenjem raste sa N. Što bi značilo da za $N = 4$ možemo brže i jednostavnije pronaći sva rješenja nego li za neki veći broj N . Upravo zbog tog razloga za prikazivanje algoritma kroz primjer razmještaja N kraljica na ploču dimenzije $N\ x\ N$ koristimo vrijednost $N = 4$.

Dakle, problem se sastoji od šahovske ploče dimenzija $4x4$ na koju trebamo postaviti četiri kraljice tako da se one međusobno ne napadaju. Zbog potrebe u daljnjem tekstu, koristit ćemo skraćene nazive za četiri kraljice. Tako ćemo prvu kraljicu koju treba smjetiti na ploču nazvati K_1, drugu K_2, itd.

Prije upotrebe algoritma moramo definirati skup uvjeta i rješenja koji zadovoljavaju početna pravila. Na primjer, postavljanje K_1 na neko mjesto uvjetuje njen napad na neka druga mjesta. Stoga postavljanje K_2 na istu ploču mora zadovoljiti prijašnji uvjet i zauzeti mjesto tako da nije pod napadom K_1. Smještanje K_3 je malo teže jer istodobno mora zadovoljiti postavljene uvjete od K_1 i K_2. Nastavljajući takvim načinom može se dogoditi da dođemo do točke gdje skup svih prijašnjih ograničenja ne dozvoljava da se iduća kraljica smjesti na ploču. Zbog toga je potrebeno „ublažiti“ uvjete i pronaći nova rješenja. „Ublažavanje“ se vrši na način da se vratimo barem jedan korak unatrag i pronađemo novo dozvoljeno rješenje, odnosno slijedeću moguću poziciju za prethodnu kraljicu. Drugim riječima, ako postavimo K_i na i-tu kolonu i ne možemo pronaći rješenje za K_{i+1} mi se vraćamo korak unatrag i pronalazimo prvo slijedeće dozvoljeno rješenje za K_i (*Engelhardt, 2007.*).

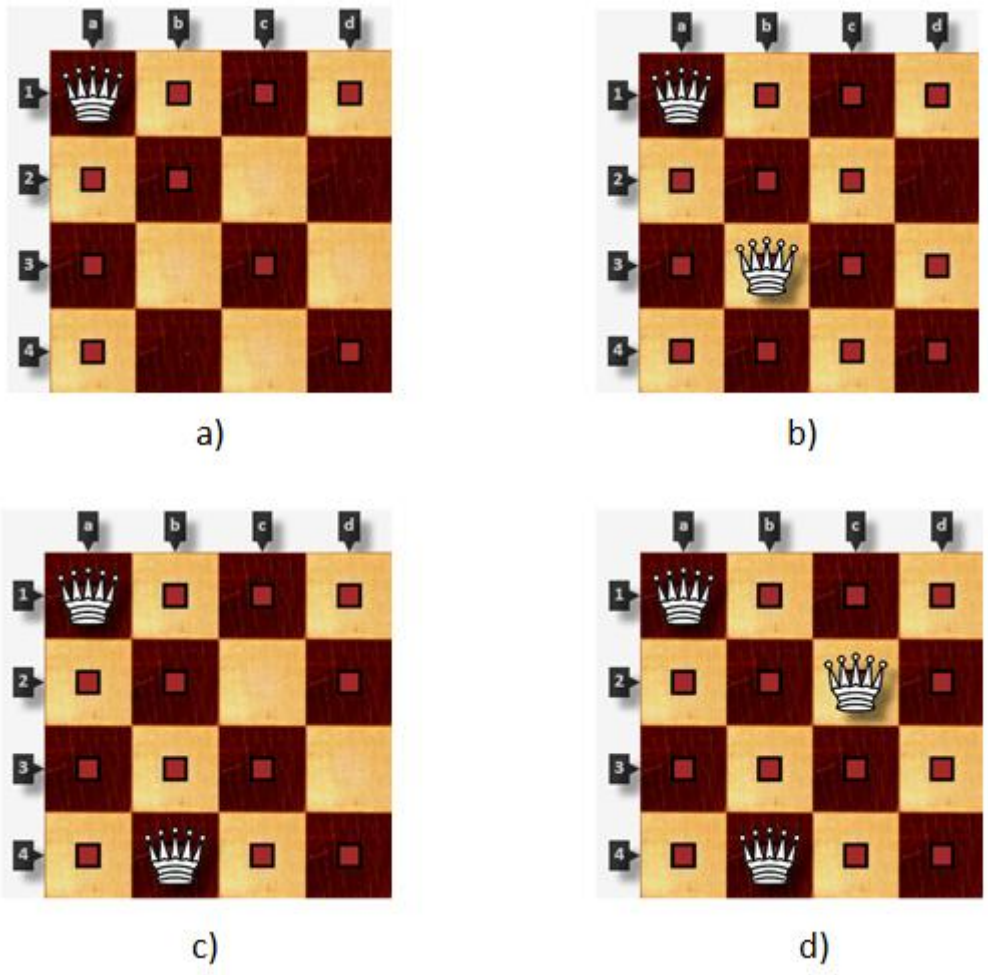
Vratimo se našem primjeru raspodjele četiri kraljice na ploču dimenzija $4x4$ tako da se one međusobno ne napadaju. Za početak postavljamo pravilo da svaka kraljica mora biti u svojoj koloni, što će biti naš početni uvjet. Zatim kreiramo pseudo kod za algoritam:

PSEUDO KOD ZA ALGORITAM PRIJE UNATRAŽNOG PRETRAŽIVANJA

1. Stavimo K_1 na mjesto a1.
2. Nastavimo sa slijedećom kraljicom K_2 i stavljamo je na mjesto b1.
3. U koloni b tražimo prvo dozvoljeno mjesto za K_2.
4. Postupak nastavljamo sa slijedećom kraljicom.

Krećemo s traženjem rješenja. Kraljicu K_1 stavljamo na poziciju a1 (Slika 3 a)). Sljedeći korak je traženje prvog slobodnog mjesta za K_2. U ovom slučaju to je mjesto b3 (Slika 3 b)). Budući da postavljanje K_2 na mjesto b3 uvjetuje nemogućnost postavljanja K_3 u treću kolonu moramo se vratiti korak unatrag te potražiti sljedeće moguće slobodno mjesto za K_2 u drugoj koloni. Drugim rječima, postavljanjem K_2 na mjesto b3 postavili smo uvjete na način da ih daljnjim korakom više ne možemo zadovoljiti, a da istodobno pronađemo cjeloukupno rješenje. Zato smo morali preispitati uvjete i preurediti ploču od one točke gdje smo zapeli (postavljanje K_2 u drugoj koloni). Uzimamo sljedeće moguće mjesto za K_2, a to je mjesto b4 (Slika 3 c)).

Sad možemo krenuti korak dalje s postavljanjem K_3 u treću kolonu na jedino slobodno mjesto c2. Time smo došli do dijela kad shvaćamo da ne možemo smjestiti K_4 u četvrtu kolonu (Slika 3 d)). Dakle, postavljanje K_2 na zadnje moguće mjesto nije pomoglo. Stoga jedan korak unatrag nije bio dovoljan. Moramo se vratiti još jedan korak unatrag. Dakle, na K_1.

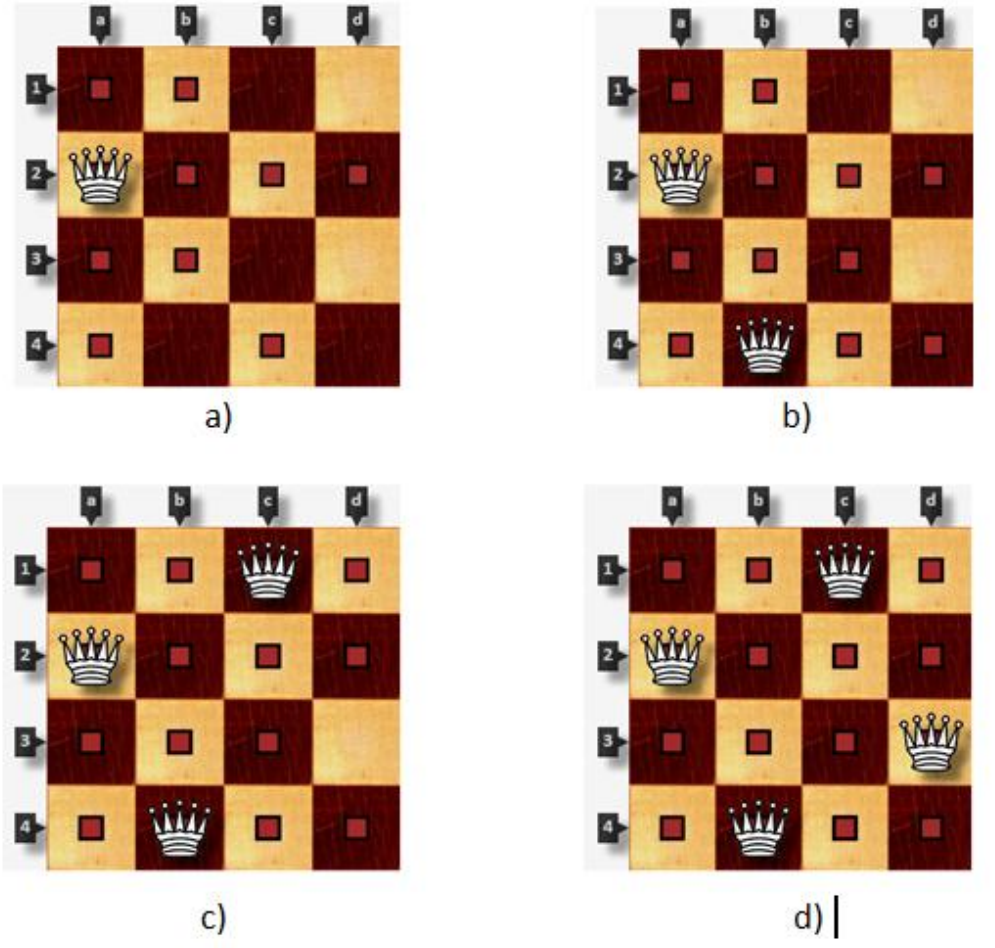


Slika 3 Prikaz postavljanja četiri kraljice na šahovsku ploču dimenzija $4x4$ počevši s prvim slobodnim mjestom za K_1. a) postavljanje K_1 u prvu kolonu na prvo slobodno mjesto a1, b) postavljanje K_2 u drugu kolonu na prvo slobodno mjesto uvažavajući uvjete algoritma, c) vršenje prvog unatražnog pretraživanje zbog nemogućnosti pronalaska daljnjeg rješenja stvorenim prijašnjim uvjetom i d) postavljanje K_3 u treću kolonu na jedino slobodno mjesto i uočavanje nemogućnosti daljnjeg rješavanja.

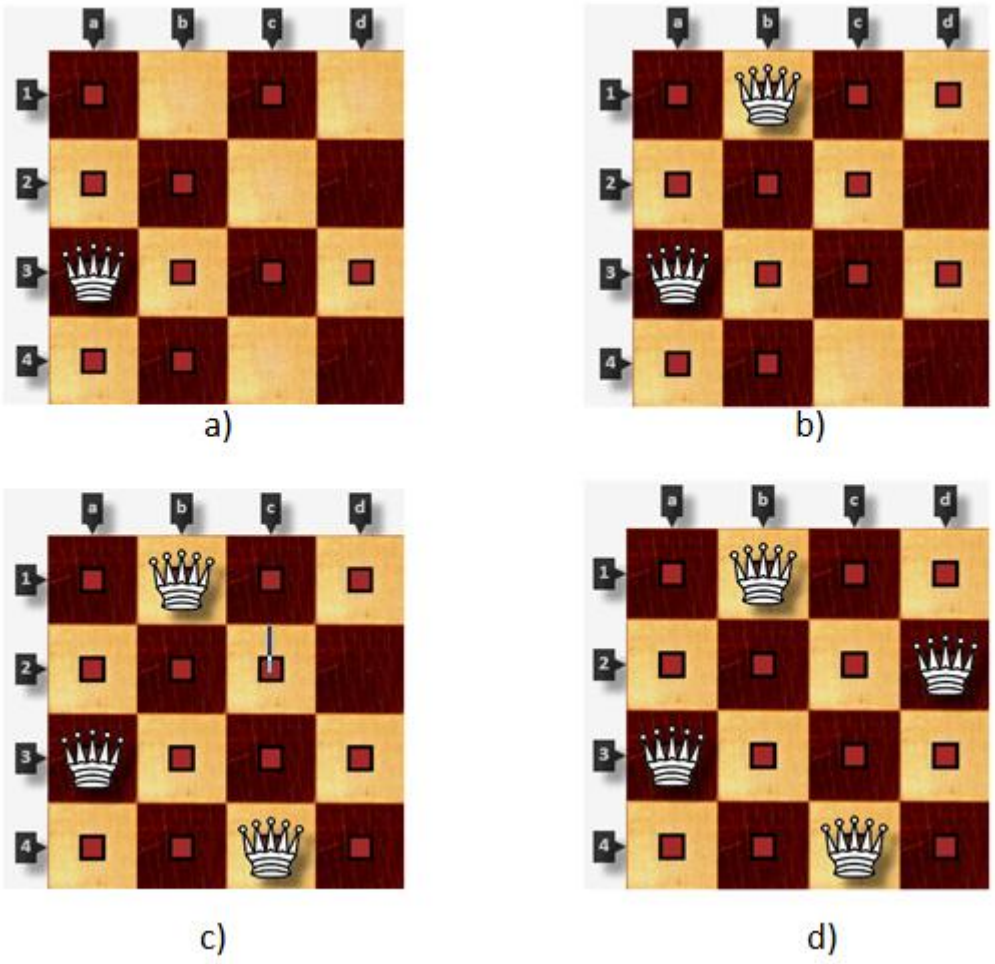
Stavljamo K_1 na mjesto a2 (Slika 4 a)) što uvjetuje postavljanje kraljice K_2 na jedino slobodno mjesto b4 (Slika 4 b)). Zatim u trećoj koloni postavimo K_3 na mjesto c1 (Slika 4 c)). Zadnji korak je postavljanje kraljice K_4 u četvrtu kolonu na mjesto d3 (Slika 4 d)).

Time smo došli do jednog rješenja problema. Za to nam je bilo potrebno obaviti dva vraćanja unatrag. Da bi pronašli sva moguća rješenja problema četiri kraljice također se koristimo metodom unatražnog pretraživanja.

Krenemo od četvrte kolone gdje odmah uočavamo da za K_4 ne postoji više nijedno slobodno mjesto. Vratimo se na K_3 i vidimo da i tu ne postoji drugo slobodno mjesto. Vraćajući se na K_2 vidimo da ni u drugoj koloni nema dozvoljenih slobodnih mjesta, čime smo se vratili na kraljicu K_1 u prvoj koloni. K_1 postavljamo na slijedeće slobodno mjesto, a3. Nastavljajući dalje s koracima koji su analogni kao i u prijašnjem slučaju pronalazimo još jedno rješenje (Slika 5) koje je u biti rotacija prvog rješenja. I zaista problem smještanja četiri kraljice na ploču dimenzije $4x4$ ima samo jedno jedinstveno rješenje.



Slika 4 Prikaz postavljanja četiri kraljice na šahovsku ploču dimenzija $\boldsymbol{4x4}$ nakon prvih unatražnih pretraživanja. a) postavljanje K_1 u prvu kolonu na drugo slobodno mjesto a2, b) postavljanje K_2 u drugu kolonu na jedino slobodno mjesto b4 uvažavajući uvjete algoritma, c) postavljanje K_3 na jedino slobodno mjesto c1 i d) postavljanje K_4 na jedino slobodno mjesto d3.



Slika 5 Prikaz traženja sljedećeg mogućeg rješenja problema četiri kraljice korištenjem algoritma unatražnog pretraživanja. a) postavljanje K_1 u prvu kolonu na sljedeće slobodno mjesto a3, b) postavljanje K_2 u drugu kolonu na jedino slobodno mjesto b1 uvažavajući uvjete algoritma, c) postavljanje K_3 na jedino slobodno mjesto c4 i d) postavljanje K_4 na jedino slobodno mjesto d2.

Ostala nam je još samo jedno slobodno mjesto za K_1 u prvoj koloni. To je mjesto a4. Daljnje traženje mogućih rješenja u tom slučaju anologno je kao i u prvom slučaju kad smo K_1 smjestili na mjesto A1. Ubrzo bi smo shvatili da u tom slučaju također ne postoji rješenja.

Na takav način smo istražili sva moguća rješenja. Kroz cijeli postupak koristili smo unatražno pretraživanje kod onih točaka kad nismo mogli pronaći dozvoljavajuća mjesta za kraljicu K_i u i-toj koloni. Dakle, mi smo u tim točkama odbacili sve moguće potomke čvora bez daljnjeg testiranja. To i jest bit unatražnog pretraživanja, kao što je prikazano na grafu na Slika 1. U suprotnom, išli smo dalje na sljedeću kolonu sve dok nismo postavili posljednju kraljicu u posljednju kolonu.

5 Implementacija unatražnog pretraživanja na problem 8 kraljica

Problem četiri kraljice se može riješiti doslovce „na prste“ kao što se može primijetiti u prijašnjem poglavlju. No što ako se broj kraljica poveća? Da li bi mi imali vremena „na prste“ proći kroz sve mogućnosti za pronalaženje riješenja ili problemu možemo pristupiti na jedan malo drugačiji način?

5.1 Moguća rješenja problema osam kraljica

Pogledajmo problem osam kraljica koje je potrebno razmjestiti na ploču dimenzija $8x8$. U takvom slučaju postoji $4\,426\,165\,368$ mogućih razmještaja. Broj se lako dobije primjenom osnova kombinatorne matematike. Budući da ploča ima 64 polja, postoje 64 slobodnih mjesta na koja možemo postaviti prvu kraljicu. Nakon postavljanja prve kraljice, za drugu ostaju 63 slobodna mjesta. Za ostalih šest kraljica primjenimo analogno pravilo. Tako za zadnju kraljicu (osmu) ostaje $(64 - 7)$ slobodnih mjesta. Kako se sve kraljice tretiraju kao isti objekt njihov raspored je nevažan stoga dobiveni broj moramo podjeliti sa $8!$.

$$\frac{64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57}{8!} = 4\,426\,165\,368$$

No rješenja problema unatoč velikom broju mogućih razmještaja ima samo 92. Kad bi mi išli preispitati svaki od mogućih razmještaja osam kraljica, pa čak koristeći se računalom, trebalo bi nam jako puno vremena. Da bi se smanjila računalna upotreba pri traženju rješenja mogu se koristiti prečaci kao što je primjena jednostavnog početnog pravila da se svaka kraljica ograniči na samo jednu kolonu (ili jedan redak). U tom slučaju broj mogućnosti će se smanjiti na $16\,777\,216$, odnosno 8^8 mogućih razmještaja što je skoro 264 puta manje mogućih razmještaja. Stvarajući permutacije koja daju moguća rješenja problema osam kraljica, a zatim još i provjeravanje na njihova dijagonalna napadanja smanjuje mogućnost razmješanja kraljica na samo $40\,320$ odnosno $8!$ mogućih razmještaja.

Naravno, to je još uvijek velik broj mogućnosti za testiranje „na prste“, stoga za ovakav problem koristimo računala. U daljnjem tekstu ovog rada nalazi se primjena unatražnog pretraživanja u JavaScript programskom jeziku na problem osam kraljica.

5.2 Programska podrška

Za implementaciju unatražnog pretraživanja na problem osam kraljica koristimo dvije programske podrške: JavaScript i KineticJS. JavaScript programski jezik nam služi za implementaciju algoritma unatražnog pretraživanja, te za pisanje dodatnih funkcija kojima kontroliramo samu izvedbu web aplikacije, dok nam KineticJS razvojno okruženje koristi da vizualno prikažemo definiranu šahovsku ploču, kraljice, te pokretanje i postavljanje kraljica na ploču.

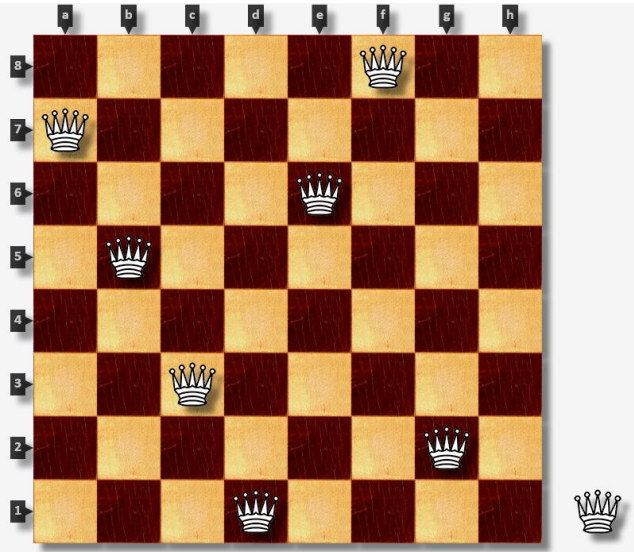
5.2.1 JavaScript

JavaScript (JS) je kompaktni, skriptni programski jezik. Najpoznatiji je po korištenju za razvoj web stranica i web aplikacija koje se izvršavaju unutar web preglednika na strani korisnika (eng. client). Time omogućava interakciju sa korisnikom, a samim time izmjenu dokumenta koji je prikazan. Također, JavaScript je moguće koristiti i na strani poslužitelja (eng. server), no to nije toliko čest slučaj. Jezik je osmislio i razvio Brendan Eich u tvrtci Netscape pod imenom *Mocha*, dok je službeni naziv bio LiveScript. Kasnije su ga preimenovali u JavaScript (*Wikipedia, JavaScript*).

Jedna od najznačajnijih upotreba JavaScript-a jest pisanje funkcija koje su uključene unutar HTML (eng. HyperText Markup Language) stranice, a koje HTML ne može izvršiti. To se često naziva DHTML (eng. Dynamic HTML).

5.2.2 KineticJS

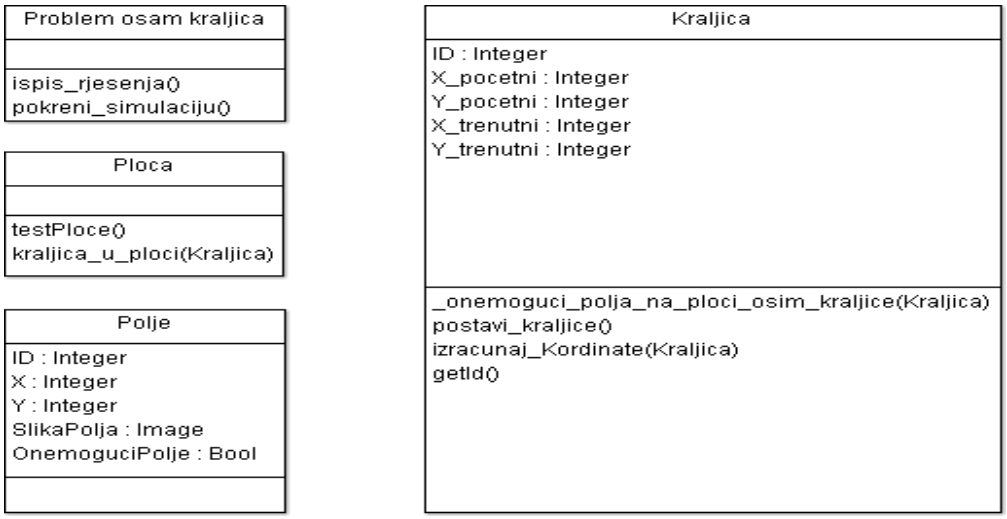
KineticJS je JavaScript razvojno okruženje koje na HTML5 „platnu“ (HTML5 Canvas) koristi objektno orijentiranu paradigmu programiranja te samim time omogućava jednostavno korištenje animacija, tranzicija i kontroliranja događaja nad objektima poput slika, teksta, kvadratnih i pravokutnih kontura te mnogih drugih unutar mobilnih ili desktop aplikacija (Slika 6) (*Wikipedia, KineticJS*).



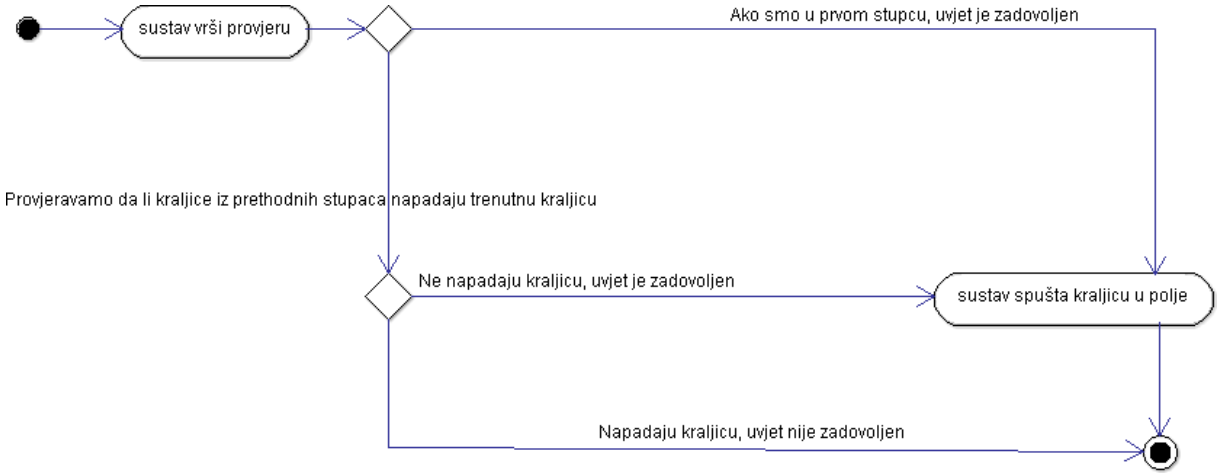
Slika 6 Primjer igre razvijene KineticJS razvojnim okruženjem.

5.3 Upotreba JavaScript-e na problem osam kraljica

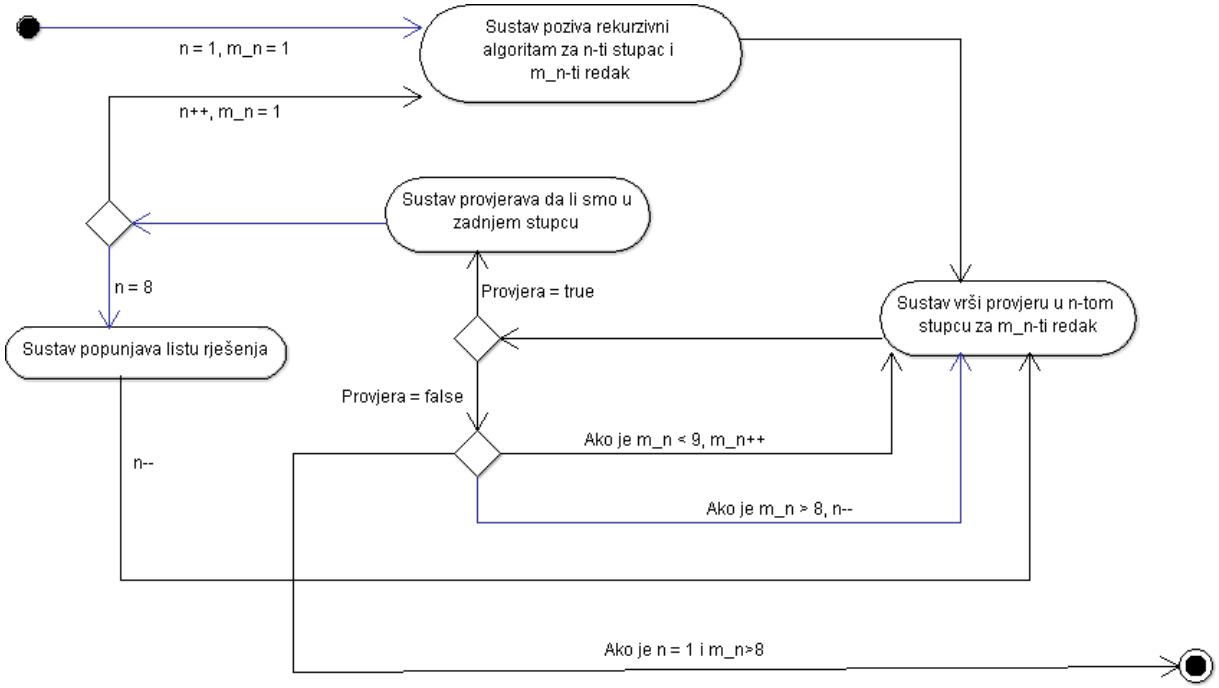
Za pronalaženje svih rješenja problema osam kraljica koristili smo implementaciju JavaScript programskog jezika na rekurzivni backtrack algoritam (algoritam unatražnog pretraživanja). U samom početku smo definirali klase objekata (Slika 7) te ploču dimenzija $8x8$. Zatim smo postavili početni uvjet prvim pravilom da se u jednom stupcu smije nalaziti samo jedna kraljica, uz proširenje na uvjet da u svakom slijedećem stupcu promatramo samo slobodna mjesta uvjetovana prvim pravilom. Na takav način smo unaprijed izbjegli ona mjesta koja napadaju kraljice iz prethodnih stupaca.



Slika 7 Na slici su prikazane četiri klase koje su korištene pri programskoj izvedbi rješenja. Klase „Problem osam kraljica“ i „Ploca“ u sebi sadrže po dvije metode. Klasa „Polje“ sadrži atribute koji opisuju svako polje ploče. Klasa „Kraljica“ sadrži atribute i metode potrebne za pomicanje kraljica. Primjena se nalazi u Prilogu I.



Slika 8 Dijagram aktivnosti koji prikazuje postupak provjere zadovoljenosti proširenog uvjeta pri pomicanju kraljica.



Slika 9 Dijagram aktivnosti koji prikazuje postupak izvođenja rekurzivnog backtrack algoritma za pronalaženje rješenja problema osam kraljica opisanog pseudo kodom.

Algoritam kojim pronalazimo rješenje problema se sastoji od dva važna dijela. U prvom dijelu se vrši provjera zadovoljavanja proširenog uvjeta (Slika 8), dok se u drugom dijelu algoritma vrši rekurzija kojom se pomičemo naprijed odnosno unatrag pri traženju rješenja. Dijagram aktivnosti na Slika 9 prikazuje postupak izvođenja rekurzivnog backtrack algoritma.

PSEUDO KOD ZA PROVJERU ZADOVOLJENOSTI UVJETA

1. Ako se nalazimo u prvom stupcu, uvjet je zadovoljen.
2. Za svaki idući stupac, provjeravamo da li prethodne kraljice napadaju trenutno mjesto.
3. Ako je trenutno mjesto napadnuto od nekih prethodnih kraljica, uvjet nije zadovoljen.
4. Ako nijedna od prethodnih kraljica ne napada trenutno mjesto, uvjet je zadovoljen.

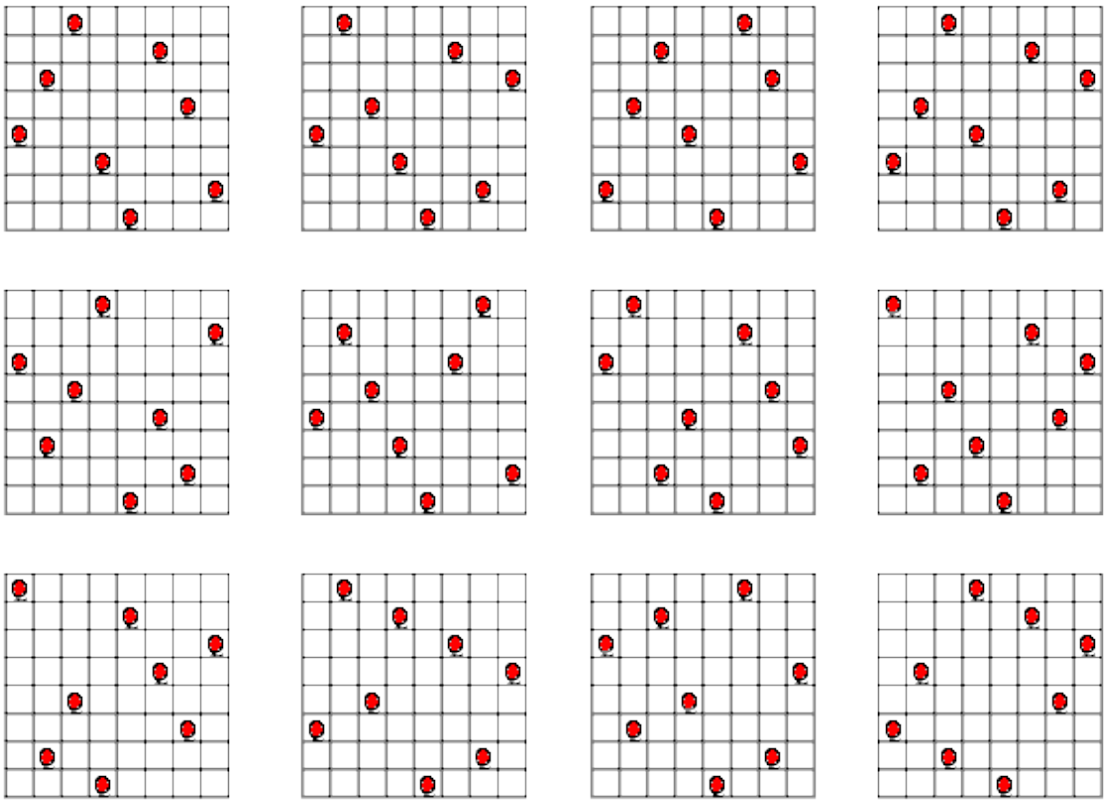
PSEUDO KOD ZA REKURZIVNI BACKTRACK

1. Krećemo od prvog retka u prvom stupcu.
2. Ako je prošireni uvjet zadovoljen, kraljica se postavlja na trenutno mjesto i poziva se algoritam za idući stupac. Vraćamo se na korak 2.
3. Ukoliko uvjet nije zadovoljen, u istom stupcu se pomičemo se na sljedeći redak i vraćamo se na korak 2.
4. Ako se uvjet nije uspio zadovoljiti niti na zadnjem redku u danom stupcu, algoritam se rekurzivno se vratića na prethodni stupac, gdje se postavljena kraljica pomiče na idući redak u tom stupcu i vraćamo se na korak 2.
5. Ukoliko smo stigli do zadnjeg stupca gdje je uvjet zadovoljen, algoritam je pronašao jedno rješenje te se vraća na prethodni stupac gdje se kraljica pomiče u idući redak. Vraćamo se na korak 2.

Najbitniji dio algoritma je onaj koji sadrži u sebi rekurziju. Stoga se u njegovom svakom koraku pozivamo na provjeru zadovoljenosti uvjeta. Ako je uvjet zadovoljen, program stavlja kraljicu na trenutno mjesto i nastavlja sa sljedećim korakom. U protivnom se nastavlja algoritam bez spuštanja kraljice na zabranjeno mjesto. Dakle, ni u jednom slučaju nema prekida izvođenja algoritma.

6 Rješenja problema osam kraljica

Iako problem osam kraljica ima 92 različitih rješenja samo je 12 od njih jedinstveno (Slika 10). Svako jedinstveno rješenje sadrži osam varijanti, uključujući i originalan oblik, koje se dobiju rotacijom za 90, 180 ili 270 stupnjeva te zrcaljenjem svake od četiri rotirajuće varijante (kao što je bio slučaj i za problem četiri kraljice). Međutim, ako je rješenje ekvivalentno njegovoj rotaciji za 90 stupnjeva onda jedinstveno rješenje ima samo dvije varijante, sebe i svoje zrcaljenje. Ako je rješenje ekvivalent rješenja dobivenog rotacijom za 180 stupnjeva (ali ne rotaciji za 90 stupnjeva) onda jedinstveno rješenje ima četiri varijante, sebe, svoje zrcaljenje, rotaciju za 90 stupnjeva te njeno zrcaljenje. Slučaj u kome je jedinstveno rješenje ekvivalentno svom zrcaljenju odbacujemo jer bi to značilo da se neke od kraljica međusobno napadaju. Od 12 jedinstvenih rješenja problema osam kraljica na ploči $8x8$, točno jedno je jednako svojoj rotaciji za 180 stupnjeva i niti jedno nije jednako svojoj rotaciji za 90 stupnjeva iz čega je ukupan broj različitih rješenja $11 \cdot 8 + 1 \cdot 4 = 92$ (*Wikipedia, Backtracking*).



Slika 10 Grafički prikaz 12 jedinstvenih rješenja kod problema osam kraljica.

U slučaju problema $\boldsymbol{N}$ kraljica broj jedinstvenih i ukupnih rješenja prikazani su u Tablica 1. Broj ukupnih rješenja raste s porastom broja $\boldsymbol{N}$, kao i broj jedinstvenih rješenja iako je on uvijek manji od broja ukupnog rješenja za dani $\boldsymbol{N}$. U slučaju $\boldsymbol{N \leq 4}$, $\boldsymbol{N \neq 2, 3}$, problem se može riješiti „na prste“, dok je za $\boldsymbol{N} > 4$ potrebna sofisticiranija metoda. Backtracking algoritam se pokazao kao korisna metoda za pronalazak rješenja do $\boldsymbol{N} < 20$, no za $\boldsymbol{N \geq 20}$ korištenje metode je sporo. Npr., pokazalo se da je za $\boldsymbol{N = 23}$, gdje ima ukupno 128 850 048 mogućih rješenja, korištenjem algoritma unatražnog pretraživanja potrebno 267 dana (*Rivin et al., 1992.*). U takvim slučajevima zgodnije se okrenuti modernijim pristupima heurističkim algoritmima koji daju rješenje na problem $\boldsymbol{N}$ kraljica bez ogromne upotrebe računala, kao npr. korištenje genetskog algoritma (*Homaifar et al., 1992.*).

Kao što je i bilo za očekivati za $\boldsymbol{N = 2}$ i $\boldsymbol{N = 3}$ ne postoji niti jedno rješenje. Također je zanimljivo primjetiti kako problem šest kraljica ima manje rješenja nego problem pet kraljica.

Tablica 1 Broj ukupnih i jedinstvenih rješenja za različite brojeve N problema N kraljica.

N	Br. ukupnih rješenja	Br. jedinstenih rješenja
1	1	1
2	0	0
3	0	0
4	2	1
5	10	2
6	4	1
7	40	6
8	92	12
9	352	46
10	724	92
11	2 680	341
12	14 200	1 787
13	73 712	9 233
14	365 596	45 752
…	…	…
25	2 207 893 435 808 352	275 986 683 743 434
26	22 317 699 616 364 044	2 789 712 466 510 589

7 Zaključak

Algoritam unatražnog pretraživanja (backtracking) je vrlo korisna metoda za pronalazak rješenja određenih problema, a ponekad i veoma kompleksnih. Metoda zahtijeva istraživanje svakog mogućeg slučaja danog problema, stoga možemo reći da je u tom pogledu veoma precizna jer ne dopušta mjesto pogrešci. Upravo zbog te odlike ona je relativno spora ovisno o danom problemu. Korištenje metode unatražnog pretraživanja u slučaju pronalaska svih mogućih rješenja problema 23 kraljice uz računalnu pomoć bilo je potrebno ukupno 267 dana. No za pronalazak svih rješenja problema N kraljica, gdje je N<20 metoda se pokazala veoma zadovoljavajuća u odnosu na ostale metode.

Svrha ovog rada je pronalazak rješenja problema osam kraljica. Kao što se pokazalo u 6. poglavlju, problem osam kraljica ima ukupno 92 rješenja, stoga pronalazak svih rješenja je teško obaviti „na prste“. Pokazalo se da je za taj problem najbolja opcija korištenje metode unatražnog pretraživanja uz pomoć računala. U Dodatku se nalazi programski kod pisan u JavaScript programskom jeziku koji uz pomoć KineticJS razvojnog okruženja stvara web aplikaciju čijim se pokretanjem lako dolazi do kombinacije svih mogućih rješenja.

Sljedeći korak bi bio istražiti korisnost korištenja algoritma unatražnog pretraživanja na problem pronalaska jedinstvenih rješenja problema N kraljica.

8 Literatura

Campbell P.J. (1977): Gauss and the Eight Queens problem: A study in miniature of the propagation of historical error, Historia Mathematica, WI 53511, 397-404, USA

Engelhardt M.R. (2007): A group-based search for solutions of the n-queens problem, Discrete Mathematics 307, 2535-2551, Nuernberg, Germany

Homaifar A., Turner J., Ali S. (1992.): Genetic Algorithms and The N-Queen Problem,The IEEE Proceedings of the Southeast Con, 262-267, Birmingham

Rivin I., Vardi I., Zimmermann P. (1992.): The n-Queens Problem, Mathematical Association of America, Vol. 101, No.7, 629-639, USA

Wikipedia (27. 6. 2013.), Backtracking, http://en.wikipedia.org/wiki/Backtracking

Wikipedia (12. 8. 2013.), Eight queens puzzle, http://en.wikipedia.org/wiki/Eight_queens_puzzle

Wikipedia (15. 9. 2013.), Java Script, http://en.wikipedia.org/wiki/JavaScript

Kinetic JS, www.kineticjs.com

Literatura slika:

Treebeard's Stumper Answer (4. 2. 2000.), www.rain.org/~mkummel/stumpers/04feb00a.html

Wikipedia (23. 4. 2013.), Max Bezzel, http://en.wikipedia.org/wiki/Max_Bezzel

Johann Carl Friedrich Gauss, www.rare-earth-magnets.com/t-johann-carl-friedrich-gauss.aspx

9 Prilozi

9.1 Prilog I – Programski kod za rješavanje problema osam kraljica

```
<!DOCTYPE HTML>
<!--
    Implementacija problema osam kraljica upotrebom HTML, JavaScript
programskih jezika te KineticJS razvojnog okruženja.
-->
<html>
<head>
<!--
    Unutar zaglavlja nalazi se :
    (1) poziv na KineticJS razvojno okruženje
    (2) funkcije potrebne za izvedbu igre, te rekurzivni backtrack
algoritam
-->
<title>Problem osam kraljica</title>
<link rel="shortcut icon" href="resources/Kraljica.ico">
<meta content="text/html; charset=windows-1250" http-equiv="Content-type"
/>
<meta name="application-name" content="Problem osam kraljica">
<meta name="description" content="Rasporedite svih osam kraljica na
šahovsku ploču tako da ni jedna ne napada drugu.">
<meta name="keywords" content="Problem osam kraljica, eigth queens problem,
HTML5, CSS3, JavaScript, KineticJS">
<meta name="author" content="Stefan Dunić">

<link href="css/master.css" rel="stylesheet" type="text/css"/>
<!-- (1) --><script type="text/javascript" src="js/kinetic-
v4.4.0.js"></script>
<!-- (2) --><script type="text/javascript">
function _2dpolje(temp) /* stvaramo dvodimenzionalno polje [temp][temp] */
{
    var array = [];
    for (var i=0;i<temp;i++)
    {
        array[i] = [];
    }
    return array;
}
function _postaviNiz() /* omogućujemo spuštanje kraljica na sva polja */
{
    for (var i=0;i<8;i++)
    {
        for (var j=0;j<8;j++)
        {
            nizOnemoguceniPolja[i][j] = false;
        }
    }
}
```

```
function kraljica_u_ploci(kraljica) /* provjeravamo da li je kraljica
unutar ploče ili izvan nje */
{
        if(kraljica.getX() > 100 && kraljica.getX() < (100 + 8*80) &&
kraljica.getY() > 40 && kraljica.getY() < (40 + 8*80))
        {
            return true;
        }
        else
        {
            return false;
        }
}
/*
    "podizanjem" kraljice se poziva ova funkcija koja nam vizualno prikaže
onemogućena polja
*/
function _onemoguci_polja_na_ploci_osim_kraljice(kraljica)
{
    onemoguciLayer.removeChildren();
    onemoguciLayer.draw();
    for(var j = 0; j < 8; j++)
    {
        for(var k = 0; k < 8; k++)
        {
            nizOnemoguceniPolja[j][k] = false;
        }
    }
    for(var i = 0; i < 8; i++)
    {
        if(kraljica.getId() != kraljice[i].getId())
        {
            if(kraljica_u_ploci(kraljice[i]))
            {
                var x_koordinata_kraljice = Math.floor((kraljice[i].getX()
- 100) / 80);
                var y_koordinata_kraljice = Math.floor((kraljice[i].getY()
- 40) / 80);

                var temp_x_desna_kosa;
                var temp_y_desna_kosa;
                var temp_x_lijeva_kosa;
                var temp_y_lijeva_kosa;

                if(x_koordinata_kraljice >= y_koordinata_kraljice)
                {
                    temp_x_desna_kosa = x_koordinata_kraljice -
y_koordinata_kraljice;
                    temp_y_desna_kosa = 0;
                }
                else
                {
                    temp_x_desna_kosa = 0;
                    temp_y_desna_kosa = y_koordinata_kraljice -
x_koordinata_kraljice;
                }

                for (var m=0; m<8; m++)
                {
                    for (var n=0; n<8; n++)
                    {
```

```
                        if( m == x_koordinata_kraljice || n==
y_koordinata_kraljice)
                        {
                            nizOnemoguceniPolja[m][n] = true;
                            var tempRect = new Kinetic.Rect({
                                x: 100 + 80*m + 30,
                                y: 40 + 80*n + 30,
                                width: 20,
                                height: 20,
                                fill: "brown",
                                stroke: "black",
                                strokeWidth: 3,
                                draggable: false
                              });
                            onemoguciLayer.add(tempRect);
                        }

                        if(m == temp_x_desna_kosa && n ==
temp_y_desna_kosa)
                        {
                            nizOnemoguceniPolja[m][n] = true;
                            var tempRect = new Kinetic.Rect({
                                    x: 100 + 80*m + 30,
                                    y: 40 + 80*n + 30,
                                    width: 20,
                                    height: 20,
                                    fill: "brown",
                                    stroke: "black",
                                    strokeWidth: 3,
                                    draggable: false
                                  });
                            onemoguciLayer.add(tempRect);
                            temp_x_desna_kosa++;
                            temp_y_desna_kosa++;
                        }
                    }
                }

                if(x_koordinata_kraljice >= y_koordinata_kraljice)
                {
                    temp_x_lijeva_kosa = x_koordinata_kraljice +
y_koordinata_kraljice;
                    temp_y_lijeva_kosa = 0;
                }
                else (x_koordinata_kraljice < y_koordinata_kraljice)
                {
                    temp_x_lijeva_kosa = 7;
                    temp_y_lijeva_kosa = x_koordinata_kraljice +
y_koordinata_kraljice - 7;
                }

                while(temp_x_lijeva_kosa >= 0 &&  temp_y_lijeva_kosa <= 7)
                {
                    if(temp_x_lijeva_kosa >= 0 && temp_x_lijeva_kosa <= 7
&& temp_y_lijeva_kosa >= 0 && temp_y_lijeva_kosa <= 7)
                    {

nizOnemoguceniPolja[temp_x_lijeva_kosa][temp_y_lijeva_kosa] = true;
                                var tempRect = new Kinetic.Rect({
                                        x: 100 + 80*temp_x_lijeva_kosa+
30,
```

```
                                                y: 40 + 80*temp_y_lijeva_kosa +
30,
                                                width: 20,
                                                height: 20,
                                                fill: "brown",
                                                stroke: "black",
                                                strokeWidth: 3,
                                                draggable: false
                                            });
                                        onemoguciLayer.add(tempRect);
                    }
                    temp_x_lijeva_kosa--;
                    temp_y_lijeva_kosa++;
                }
                onemoguciLayer.draw();
            }
        }
    }
}

function testPloce() /* koristimo kod izračuna računanja kordinata nakon
"spuštanja" kraljice */
{
    for(var j = 0; j < 8; j++)
    {
        for(var i = 0; i < 8; i++)
        {
            if(stage.getPointerPosition().x > nizPoljaPloce[j][i].getX() &&
stage.getPointerPosition().x < (nizPoljaPloce[j][i].getX() + 80) &&
stage.getPointerPosition().y > nizPoljaPloce[j][i].getY() &&
stage.getPointerPosition().y < (nizPoljaPloce[j][i].getY() + 80))
            {
                if(nizOnemoguceniPolja[i][j]==false)
                {
                    return true;
                }
            }
        }
    }
    return false;
}
/*
    Računamo kordinate kraljice nakon "spuštanja" u ovisnosti o tome da li
kraljicu
    spuštamo unutar ploče, izvan ploče ili na onemogućenom polju. Ako je na
omogućenom polju, računamo "centar"
    polja, u suprotnom vraćamo je na početnu poziciju.
*/
function izracunaj_Kordinate(kraljica)
{
    if(testPloce())
    {
        var temp1 = stage.getPointerPosition().x - 100;
        temp1 = Math.floor(temp1 / 80)*80 + 110;
        var temp2 = stage.getPointerPosition().y - 40;
        temp2 = Math.floor(temp2 / 80)*80 + 40 + 10;
        xy[0] = temp1;
        xy[1] = temp2;

    }
    else
```

```
    {
        if(stage.getPointerPosition().x > 100 &&
stage.getPointerPosition().x < (100 + 8*80) && stage.getPointerPosition().y
> 40 && stage.getPointerPosition().y < (40 + 8*80))
        {
            xy[0] = trenutniX[kraljica.getId()];
            xy[1] = trenutniY[kraljica.getId()];
        }
        else
        {
            xy[0] = niz_pocetniX[kraljica.getId()];
            xy[1] = niz_pocetniY[kraljica.getId()];
        }

    }

}
/* POSTAVI PLOCU */
function postavi_plocu(){
    for (var j=0; j<8; j++)
    {
        /* Vraćamo kraljice na njihove početne pozicije izvan ploče. */
        kraljice[j].transitionTo({
                        x: niz_pocetniX[j],
                        y: niz_pocetniY[j],
                        opacity: 1,
                        duration: 0.6,
                    });
        trenutniX[j] = niz_pocetniX[j];
        trenutniY[j] = niz_pocetniY[j];
    }
    broj_rjesenja = 0;
}
/* ALGORITAM */
var N = 8;
var position = new Array(N);
function Provjera(stupac, redak){
    for (var i = 1; i <= stupac; i++) {
                    var other = position[stupac - i];
                    if (other == redak || other == redak - i || other ==
redak + i) {
                        return false;
                    }
                }

    return true;
}
var korak = 0;
var broj_rjesenja = 0;
var lista_koraka = new Array();
function Algoritam(stupac) {
    if (stupac == N) {
        for(var m = 0; m < N; m++)
        {
            rjesenja += (position[m]+1) + " ";
        }
        rjesenja += ";  ";
        ispis_rjesenja();
    } else {
        for (var redak = 0; redak < N; redak++) {
            if (Provjera(stupac, redak)) {
```

```
                position[stupac] = redak;
                lista_koraka[korak++] = {x: (stupac + 1), y:
(position[stupac]+1)};
                Algoritam(stupac+1);
            }
        }
    }
}
/* SIMULACIJA */
function pokreni_simulaciju(temp){
    korak = temp;
    for( j = (lista_koraka[korak].x); j<8; j++){
            kraljice[j].transitionTo({
                            x: niz_pocetniX[j],
                            y: niz_pocetniY[j],
                            opacity: 1,
                            duration: 0.6
                        });
    }
    kraljice[(lista_koraka[korak].x) - 1].transitionTo({
                            x: (lista_koraka[korak].x-1) * 80 + 110,
                            y: (lista_koraka[korak].y-8) * (-80) + 50,
                            opacity: 1,
                            duration: 0.6,
                            callback: function() {
                                        if(lista_koraka[korak].x == 8) {
                                           broj_rjesenja++;
                                           alert("Pronađeno je " +
broj_rjesenja + ". rješenje!\nAko želite prekinuti simulaciju postavite
ploču.");
                                        }
                                       korak++;
                                       pokreni_simulaciju(korak);
                                      }
                        });
}
</script>
</head>
<!--
    Unutar tijela stranice nalazi se okvir unutar kojeg je napravljena igra
na kojoj je
    implementiran backtrack algoritam za pronalaženje rješenja korištenjem
KineticJS
    razvojnog okruženja.
 -->
<body>
</br>
    <div id="container"></div>
    <script type="text/javascript">
        var x_polje = 100;
        var y_polje = 40;
        var xy = [];
        /*
            Stvaramo pozornicu na kojoj ćemo dodavati slojeve, grupe
slojeva na koje
            ćemo dodavati razne objekte poput slika (crno polje, bijelo
polje, kraljica),
            teksta i pravokutnika
        */
        var stage = new Kinetic.Stage({
                                    container: 'container',
```

```
                                    width: 1240,
                                    height: 800
                                });
        var plocaLayer = new Kinetic.Layer();
        var kraljiceLayer = new Kinetic.Layer();
        var onemoguciLayer = new Kinetic.Layer();
        var textLayer = new Kinetic.Layer();
        var sideTextLayer = new Kinetic.Layer();
        var rjesenjaTextLayer = new Kinetic.Layer();
        var group_postavi_plocu = new Kinetic.Group();
        var group_prikaz_rjesenja = new Kinetic.Group();
        var group_simuliraj = new Kinetic.Group();
        /*
            Definiramo varijable nizova koje ćemo koristiti.
        */
        var kraljice = [];
        var niz_pocetniX = [];
        var niz_pocetniY = [];
        var trenutniX = [];
        var trenutniY = [];
        var nizPoljaPloce = _2dpolje(8);
        var nizOnemoguceniPolja = _2dpolje(8);
        _postaviNiz();
        /*
            Dodajemo slojeve na našu pozornicu
        */
        stage.add(rjesenjaTextLayer);
        stage.add(sideTextLayer);
        stage.add(textLayer);
        stage.add(plocaLayer);
        stage.add(onemoguciLayer);
        stage.add(kraljiceLayer);
        /*
            Učitavamo slike koje ćemo prikazivati (crno polje, bijelo
polje, kraljica)
        */
        var imageObj = new Image();
        imageObj.onload = function() {
            stage.draw();
        }
        imageObj.src = 'resources/kraljica.png';
        var imageCrnoPolje = new Image();
        imageCrnoPolje.onload = function() {
            stage.draw();
        }
        imageCrnoPolje.src = 'resources/crnoPolje.png';
        var imageBijeloPolje = new Image();
        imageBijeloPolje.onload = function() {
            stage.draw();
        }
        imageBijeloPolje.src = 'resources/bijeloPolje.png';
        /*
            Koristeći objekte pravokutnika i teksta kreiramo tekst sa
strane tako da
            objekte strateški pozicioniramo po x, y kordinatama. Prvo
stvaramo naslov,
            zatim sam tekst koji se ispisuje i na kraju pravokutni okvir
koje dodajemo
            na već postojeći sloj na pozornici. Na isti način kreiramo
okvir za rješenja.
        */
```

```
        var sideText_naslov = new Kinetic.Text({
            x: 100 + 8*80 + 130,
            y: 40,
            text: 'Problem 8 kraljica',
            fontSize: 22,
            fontFamily: 'Calibri',
            fontStyle: 'bold',
            fill: '#555',
            width: 340,
            padding: 20,
            align: 'center'
        });
        var sideText = new Kinetic.Text({
            x: 100 + 8*80 + 130,
            y: sideText_naslov.getHeight(),
            text: '\nRasporedite svih 8 kraljica na šahovsku ploču tako da
ni jedna ne napada drugu.',
            fontSize: 18,
            fontFamily: 'Calibri',
            fill: '#555',
            width: 340,
            padding: 20,
            align: 'center'
        });
        var sideTextPolje = new Kinetic.Rect({
            x: 100 + 8*80 + 130,
            y: 40,
            stroke: '#555',
            strokeWidth: 3,
            fill: '#ddd',
            width: 340,
            height: sideText_naslov.getHeight() + sideText.getHeight() -
40,
            shadowColor: 'black',
            shadowBlur: 10,
            shadowOffset: 10,
            shadowOpacity: 0.2,
            cornerRadius: 3
        });
        sideTextLayer.add(sideTextPolje);
        sideTextLayer.add(sideText_naslov);
        sideTextLayer.add(sideText);
        var rjesenja = '';

        function ispis_rjesenja(){
            rjesenjaTextLayer.removeChildren();
            var rjesenjaText_naslov = new Kinetic.Text({
            x: 100 + 8*80 + 130,
            y: 225,
            text: '92 rješenja : ',
            fontSize: 16,
            fontFamily: 'Calibri',
            fontStyle: 'bold',
            fill: '#555',
            width: 340,
            padding: 20,
            align: 'left'
            });
            var rjesenjaText = new Kinetic.Text({
            x: 100 + 8*80 + 130,
            y: 190 + rjesenjaText_naslov.getHeight(),
```

```
            text: rjesenja,
            fontSize: 15,
            fontFamily: 'Calibri',
            fontStyle: 'bold',
            fill: '#555',
            width: 270,
            padding: 20,
            align: 'left'
            });
            var rjesenjaTextPolje = new Kinetic.Rect({
            x: 100 + 8*80 + 130,
            y: 225,
            stroke: '#555',
            strokeWidth: 3,
            fill: '#ddd',
            width: 340,
            height: rjesenjaText_naslov.getHeight() +
rjesenjaText.getHeight() - 40,
            shadowColor: 'black',
            shadowBlur: 10,
            shadowOffset: 10,
            shadowOpacity: 0.2,
            cornerRadius: 3
            });

            rjesenjaTextLayer.add(rjesenjaTextPolje);
            rjesenjaTextLayer.add(rjesenjaText);
            rjesenjaTextLayer.add(rjesenjaText_naslov);
            rjesenjaTextLayer.add(group_simuliraj);
            rjesenjaTextLayer.draw();
        }
        /*
            Kreiramo grupu objekata koji zajedno čine botun "Postavi ploču"
koji se sastoji od
            teksta i okvira. Nadalje grupa kao cjelina ima svoje događaje
koje smo stvorili
            nakon što smo dodali objekte u grupu. Unutar svakog događaja se
odvija zasebna
            JavaScript funkcija. Grupu na kraju dodajemo na već postojeći
sloj unutar pozornice.
        */
        var postavi_plocu_text = new Kinetic.Text({
            x: 100 + 8*80 + 130,
            y: 40 + sideTextPolje.getHeight() + 10,
            fontSize: 20,
            text: 'Postavi ploču',
            fontFamily: 'Calibri',
            fontStyle: 'bold',
            fill: 'grey',
            width: 160,
            padding: 5,
            align: 'center'
        });
        var postavi_plocu_okvir = new Kinetic.Rect({
            x: 100 + 8*80 + 130,
            y: 40 + sideTextPolje.getHeight() + 10,
            width:160,
            height: postavi_plocu_text.getHeight(),
            stroke: 'grey',
            strokeWidth: 3,
            fill: 'WhiteSmoke',
```

```
            shadowColor: 'black',
            shadowBlur: 10,
            shadowOffset: 5,
            shadowOpacity: 0.2,
            cornerRadius: 3
        });
        group_postavi_plocu.add(postavi_plocu_okvir);
        group_postavi_plocu.add(postavi_plocu_text);

        group_postavi_plocu.on('mouseover', function() {
            document.body.style.cursor = 'pointer';
            postavi_plocu_okvir.setFill('FloralWhite');
            sideTextLayer.draw();
        });
        group_postavi_plocu.on('mousedown', function() {
            document.body.style.cursor = 'pointer';
            postavi_plocu_text.setFill('green');
            postavi_plocu_okvir.setStroke('green');
            postavi_plocu_okvir.setFill('FloralWhite');
            postavi_plocu_okvir.setShadowColor('green');
            sideTextLayer.draw();
        });
        group_postavi_plocu.on('mouseup', function() {
            document.body.style.cursor = 'pointer';
            postavi_plocu_text.setFill('grey');
            postavi_plocu_okvir.setStroke('grey');
            postavi_plocu_okvir.setFill('FloralWhite');
            postavi_plocu_okvir.setShadowColor('grey');
            sideTextLayer.draw();
        });
        group_postavi_plocu.on('mouseout', function() {
            document.body.style.cursor = 'default';
            postavi_plocu_text.setFill('grey');
            postavi_plocu_okvir.setStroke('grey');
            postavi_plocu_okvir.setFill('WhiteSmoke');
            postavi_plocu_okvir.setShadowColor('grey');
            sideTextLayer.draw();
        });
        group_postavi_plocu.on('click', function() {
            postavi_plocu();
        });
        sideTextLayer.add(group_postavi_plocu);
        /*
            Kreiramo grupu objekata koji zajedno čine botun "Prikaz
rješenja" koji se sastoji od
            teksta i okvira. Nadalje grupa kao cjelina ima svoje događaje
koje smo stvorili
            nakon što smo dodali objekte u grupu. Unutar svakog događaja se
odvija zasebna
            JavaScript funkcija. Grupu na kraju dodajemo na već postojeći
sloj unutar pozornice.
        */
        var prikaz_rjesenja_text = new Kinetic.Text({
            x: 100 + 8*80 + 310,
            y: 40 + sideTextPolje.getHeight() + 10,
            fontSize: 20,
            text: 'Prikaz rješenja',
            fontFamily: 'Calibri',
            fontStyle: 'bold',
            fill: 'grey',
            width: 160,
```

```
    padding: 5,
    align: 'center'
});
var prikaz_rjesenja_okvir = new Kinetic.Rect({
    x: 100 + 8*80 + 310,
    y: 40 + sideTextPolje.getHeight() + 10,
    width:160,
    height: prikaz_rjesenja_text.getHeight(),
    stroke: 'grey',
    strokeWidth: 3,
    fill: 'WhiteSmoke',
    shadowColor: 'black',
    shadowBlur: 10,
    shadowOffset: 5,
    shadowOpacity: 0.2,
    cornerRadius: 3
});
group_prikaz_rjesenja.add(prikaz_rjesenja_okvir);
group_prikaz_rjesenja.add(prikaz_rjesenja_text);

group_prikaz_rjesenja.on('mouseover', function() {
    document.body.style.cursor = 'pointer';
    prikaz_rjesenja_okvir.setFill('FloralWhite');
    sideTextLayer.draw();
});
group_prikaz_rjesenja.on('mousedown', function() {
    document.body.style.cursor = 'pointer';
    prikaz_rjesenja_text.setFill('green');
    prikaz_rjesenja_okvir.setStroke('green');
    prikaz_rjesenja_okvir.setFill('FloralWhite');
    prikaz_rjesenja_okvir.setShadowColor('green');
    sideTextLayer.draw();
});
group_prikaz_rjesenja.on('mouseup', function() {
    document.body.style.cursor = 'pointer';
    prikaz_rjesenja_text.setFill('grey');
    prikaz_rjesenja_okvir.setStroke('grey');
    prikaz_rjesenja_okvir.setFill('FloralWhite');
    prikaz_rjesenja_okvir.setShadowColor('grey');
    sideTextLayer.draw();
});
group_prikaz_rjesenja.on('mouseout', function() {
    document.body.style.cursor = 'default';
    prikaz_rjesenja_text.setFill('grey');
    prikaz_rjesenja_okvir.setStroke('grey');
    prikaz_rjesenja_okvir.setFill('WhiteSmoke');
    prikaz_rjesenja_okvir.setShadowColor('grey');
    sideTextLayer.draw();
});
group_prikaz_rjesenja.on('click', function() {
    rjesenja = '';
    postavi_plocu();
    Algoritam(0);
});
sideTextLayer.add(group_prikaz_rjesenja);
var simuliraj_rjesenja_text = new Kinetic.Text({
    x: 100 + 8*80 + 315,
    y: 235,
    fontSize: 16,
    text: 'Simuliraj',
    fontFamily: 'Calibri',
```

```
            fontStyle: 'bold',
            fill: 'grey',
            width: 150,
            padding: 5,
            align: 'center'
        });
        var simuliraj_rjesenja_okvir = new Kinetic.Rect({
            x: 100 + 8*80 + 315,
            y: 235,
            width:150,
            height: simuliraj_rjesenja_text.getHeight(),
            stroke: 'grey',
            strokeWidth: 3,
            fill: 'WhiteSmoke',
            shadowColor: 'black',
            shadowBlur: 10,
            shadowOffset: 5,
            shadowOpacity: 0.2,
            cornerRadius: 3
        });
        group_simuliraj.add(simuliraj_rjesenja_okvir);
        group_simuliraj.add(simuliraj_rjesenja_text);

        group_simuliraj.on('mouseover', function() {
            document.body.style.cursor = 'pointer';
            simuliraj_rjesenja_okvir.setFill('FloralWhite');
            rjesenjaTextLayer.draw();
        });
        group_simuliraj.on('mousedown', function() {
            document.body.style.cursor = 'pointer';
            simuliraj_rjesenja_text.setFill('green');
            simuliraj_rjesenja_okvir.setStroke('green');
            simuliraj_rjesenja_okvir.setFill('FloralWhite');
            simuliraj_rjesenja_okvir.setShadowColor('green');
            rjesenjaTextLayer.draw();
        });
        group_simuliraj.on('mouseup', function() {
            document.body.style.cursor = 'pointer';
            simuliraj_rjesenja_text.setFill('grey');
            simuliraj_rjesenja_okvir.setStroke('grey');
            simuliraj_rjesenja_okvir.setFill('FloralWhite');
            simuliraj_rjesenja_okvir.setShadowColor('grey');
            rjesenjaTextLayer.draw();
        });
        group_simuliraj.on('mouseout', function() {
            document.body.style.cursor = 'default';
            simuliraj_rjesenja_text.setFill('grey');
            simuliraj_rjesenja_okvir.setStroke('grey');
            simuliraj_rjesenja_okvir.setFill('WhiteSmoke');
            simuliraj_rjesenja_okvir.setShadowColor('grey');
            rjesenjaTextLayer.draw();
        });
        group_simuliraj.on('click', function() {
            pokreni_simulaciju(0);
        });

        /*
            Na već postojeći sloj kraljica "kraljiceLayer" dodajemo 8
kraljica sa njihovim događajima.
            Nadalje na sloj ploče "plocaLayer" dodajemo crno, bijela polja.
Te na sloj teksta "textLayer"
```

```
            dodajemo vizualne pozicije polja.
            Petlja se pomiče tako da index j predstavlja redak, a index i
predstavlja stupac. Stupaca ima
            9 jer su u devetom stupcu početne pozicije kraljica sa strane
ploče.
        */
        for (var j=0; j<8; j++)
        {
            y_polje = 40 + j*80;
            for (var i=0; i<9; i++)
            {
                x_polje = 100 + i*80;
                if(i == 8) {
                    x_polje = x_polje + 40;
                    y_polje = y_polje + 10;
                    /*
                        Stvaramo 8 različitih objekata kraljica sa njihovim
atributima i događajima.
                        Atributi su početne i trenutne kordinate. Događaji
su "dragstart" i "dragend".
                    */
                    kraljice[j] = new Kinetic.Image({
                              id: "" + j,
                              x: x_polje,
                              y: y_polje,
                              image: imageObj,
                              width: 60,
                              height: 60,
                              draggable: true,
                              shadowColor: 'black',
                              shadowBlur: 10,
                              shadowOffset: 10,
                              shadowOpacity: 0.5
                            });
                    /*Atributi*/
                    niz_pocetniX[j] = x_polje;
                    niz_pocetniY[j] = y_polje;
                    trenutniX[j] = x_polje;
                    trenutniY[j] = y_polje;
                    /*Događaji*/
                    kraljice[j].on('dragstart', function() {
                        _onemoguci_polja_na_ploci_osim_kraljice(this);
                    });
                    kraljice[j].on('dragend', function() {
                    izracunaj_Kordinate(this);
                    this.transitionTo({
                                x: xy[0],
                                y: xy[1],
                                opacity: 1,
                                duration: 0.6,
                                });
                    trenutniX[this.getId()] = xy[0];
                    trenutniY[this.getId()] = xy[1];
                    onemoguciLayer.removeChildren();
                    onemoguciLayer.draw();
                    });
                    kraljice[j].on('mouseover', function() {
                        document.body.style.cursor = 'pointer';
                        this.setShadowColor('brown');
                        kraljiceLayer.draw();
                    });
```

```
                kraljice[j].on('mouseout', function() {
                    document.body.style.cursor = 'default';
                    this.setShadowColor('black');
                    kraljiceLayer.draw();
                });
                kraljiceLayer.add(kraljice[j]);
            }
            else
            {
                if(j == 0) /*Ako smo u prvom retku dodajemo slova a,b,c
...*/
                {
                    var donjiText = new Kinetic.Label({
                            x: x_polje + 40,
                            y: y_polje,
                            opacity: 0.75,
                            listening: false,
                            text: {
                              text: String.fromCharCode(97 + i),
                              fontFamily: 'Calibri',
                              fontSize: 18,
                              fontStyle: 'bold',
                              padding: 5,
                              fill: 'white'
                            },
                            rect: {
                              fill: 'black',
                              pointerDirection: 'down',
                              pointerWidth: 10,
                              pointerHeight: 10,
                              lineJoin: 'round',
                              shadowColor: 'black',
                              shadowBlur: 10,
                              shadowOffset: 10,
                              shadowOpacity: 0.5
                            }
                          });

                            textLayer.add(donjiText);
                }
                if(i == 0) /*Ako smo u prvom stupcu dodajemo brojke
8,7,6 ...*/
                {
                    var lijeviText = new Kinetic.Label({
                        x: x_polje,
                        y: y_polje + 40,
                        opacity: 0.75,
                        listening: false,
                        text: {
                          text: '' + (8-j),
                          fontFamily: 'Calibri',
                          fontSize: 18,
                          fontStyle: 'bold',
                          padding: 5,
                          fill: 'white'
                        },
                        rect: {
                          fill: 'black',
                          pointerDirection: 'right',
                          pointerWidth: 10,
                          pointerHeight: 10,
```

```
                                lineJoin: 'round',
                                shadowColor: 'black',
                                shadowBlur: 10,
                                shadowOffset: 10,
                                shadowOpacity: 0.5
                            }
                        });
                        textLayer.add(lijeviText);
                    }
                    /*
                        Logički raspisani uvjeti temeljem kojih dodajemo
crno odnosno bijelo polje.
                        Svako polje ima jedan atribut, koji je
"implementiran", pozivanjem funkcija iz zaglavlja,
                        u obliku niza koji nam govori na koja polja možemo,
a na koja ne možemo spusititi kraljicu.
                    */
                    if( ((i%2 == 1) && (j%2 == 1)) || ((i%2 == 0) && (j%2
== 0)) ) {
                        nizPoljaPloce[j][i] = new Kinetic.Image({
                                id: "" + (j + 1) + (i + 1),
                                x: x_polje,
                                y: y_polje,
                                width: 80,
                                height: 80,
                                image: imageCrnoPolje,
                                shadowColor: 'black',
                                shadowBlur: 10,
                                shadowOffset: 10,
                                shadowOpacity: 0.5
                            });
                        plocaLayer.add(nizPoljaPloce[j][i]);
                    }
                    if( ((i%2 == 1) && (j%2 == 0)) ||((i%2 == 0) && (j%2 ==
1)) ) {
                        nizPoljaPloce[j][i] = new Kinetic.Image({
                                                id: "" + (j + 1) + (i + 1),
                                                x: x_polje,
                                                y: y_polje,
                                                width: 80,
                                                height: 80,
                                                image: imageBijeloPolje,
                                                shadowColor: 'black',
                                                shadowBlur: 10,
                                                shadowOffset: 10,
                                                shadowOpacity: 0.5
                                            });
                        plocaLayer.add(nizPoljaPloce[j][i]);
                    }
                }
            }
        }
    </script>
</body>
</html>
```