

Rešavanje Sudoku slagalice Backtracking algoritmom

Solving Sudoku puzzles using backtracking algorithm

Kandidat

Jovanoski Bojan

Mentor

Mikloš Pot

Subotica, 2019. godine

Izvod

Zadatak je napraviti algoritam koji uspešno rešava Sudoku slagalicu. Takođe je potrebno napraviti jednostavan interfejs pomoću kojeg korisnici unose parcijalno rešenu slagalicu. Aplikacija je potrebno da ima i mogućnost rešavanja pojedinačnog polja koje korisnik prethodno odabere, kao i prikazivanje vremena koje je potrošeno za rešavanje, i broj utrošenih koraka (ukupan broj, kao i broj koraka u napred i broj koraka u nazad). Celokupno rešenje slagalice treba da se obavlja pomoću backtracking algoritma, i jednom nađeno rešenje da se prikaže korisniku na pregledan i lako čitljiv način.

Sadržaj

Izvod.....	2
Sadržaj.....	3
Zahvale.....	4
Zadatak završnog rada	5
1. Uvod.....	6
1.1. Opis problema.....	6
1.2. Ciljevi rada.....	6
2. Sudoku	7
2.1. Pravila igre.....	7
2.2. Istorijat	7
2.3. Sudoku slagalice.....	8
3. Nizovi.....	9
3.1. Jednodimenzionalni niz.....	9
3.2. Dvodimenzionalni niz.....	9
3.3. Trodimenzionalni niz.....	9
4. Backtracking.....	10
5. Aplikacija	14
6. Kôd programa	18
6.1. Front – end.....	18
6.2. Back – end.....	21
6.2.1. Kreiranje niza	21
6.2.2. Predefinisana polja.....	21
6.2.3. Validacija polja	21
6.2.4. Kreiranje mogućih rešenja	23
6.2.5. Rešavanje problema.....	23
7. Vremenski zahtevi.....	27
8. Zaključak.....	29
Literatura	30

Zahvale

Hvala Milanu i Bojanu, bez kojih ne bih uspeo da završim sve ispite još neko vreme. Takođe hvala i profesoru Potu na svim savetima i konsultacijama vezanih za ovaj rad i predmet uopšte.

Zadatak završnog rada

Zadatak diplomskog rada je izrada softverskog rešenja za automatskog igrača logičke igre Sudoku. Igru Sudoku treba prikazati prvo teorijski, onda razmotriti moguće strategije rešenja i na osnovu toga napisati funkcionalan program koji će dati tačno rešenje na osnovu prethodno zadatah, predefinisanih brojeva. Dobijeno rešenje je potrebno ispiati korisniku na jednostavan i pregledan način.

1. Uvod

U ovom radu je opisana Sudoku slagalica, načini rešavanja, najbolje strategije, kao i njen istorijat. Takođe je opisan backtracking algoritam sa primerima, kao i drugi pojmovi koji su korišteni za razvoj ove aplikacije. Rad takođe sadrži delove koda koje opisuju kako algoritam funkcioniše.

1.1. Opis problema

Aplikacija je potrebno da reši Sudoku slagalice na efikasan i brz način, koristeći backtracking algoritam. Aplikacija takođe treba da prikazuje vreme rešavanja, kao i broj potrebnih koraka za dolazak do rešenja. Takođe, ukoliko korisnik ne želi da dobije rešenje cele slagalice, već samo rešenje za određeno polje, potrebno je da postoji opcija i za to, gde će se korisniku pokazati moguća rešenja za odabrano polje u datom trenutku.

1.2. Ciljevi rada

Cilj rada je napraviti funkcionalnu aplikaciju za rešavanje Sudoku slagalice. Za logiku koristiti Backtracking algoritam. Interfejs aplikacije praviti pomoću HTML i CSS tehnologije, a samu logiku rešavanja koristeći PHP. Za komunikaciju između korisnika i servera, koristiti JavaScript i jQuery.

2. Sudoku

Sudoku (数独, na japanskom znači jedan broj) ili učestalo sudoko je vrsta matematičke zagonetke čije je rešavanje temeljeno na logici. Sastoji se od jednog velikog kvadratnog polja, podeljenog na 81 manjih kvadratića. Zatim, unutar velikog kvadrata, označeno je 9 odeljaka veličine 3x3 polja. Postoje različite težine te igre, a koristi se za zabavu ili testiranje inteligencije [1].

2.1. Pravila igre

Cilj igre je ispuniti sva polja brojevima od 1 do 9, s time da se svaki broj sme pojaviti tačno 9 puta. Problematika je u tome što se jedan broj sme pojaviti samo jednom u svakom redu, svakoj kolini i svakom odeljku od 3x3 polja. Na početku igre, otkriveni su određeni brojevi, a igrač mora otkriti gde se nalaze svi ostali brojevi i kako su raspoređeni. Sudoku može imati više rešenja, ako je na početku otkriveno malo brojeva. Zato treba paziti da se greška ne načini na početku rešavanja. Jedini način rešavanja Sudokua je metoda eliminacije, a tu se koristi svojstvom da se jedan broj sme pojaviti samo jednom u svakom redu, koloni i bloku. Rešavanje postaje lakše ukoliko se otkrije više brojeva.

2.2. Istorijat



Igra potiče iz Japana gde se igra već 1000 godina, dok u modernom svetu postaje popularnija kad počinje da se štampa u Francuskom magazinu La France, 6. jula 1895 godine (Slika 1). U to vreme Sudoku slagalica nije izgledala identično današnjoj, jer se od igrača nije tražilo da poštuje da i u jednom kvadratu od 3x3 bude jedinstvenih brojeva, već samo po kolonama i redovima.

Moderni Sudoku nastaje 1979. godine kad Howard Garns predlaže časopisu Dell Magazines da štampa ove slagalice. U Ujedinjenom Kraljevstvu Sudoku počinje da se pojavljuje u novinama The Times 2004. godine. Wayne Gould je zaslužan za štampanje Sudokua u Ujedinjenom Kraljevstvu, a posle je i razvio softver koji je generisao Sudoku slagalice, koji je olakšao posao svima u štampariji [2].

Slika 1

2.3. Sudoku slagalice

Običan Sudoku ima jedinstveno rešenje. Minimalnim Sudokom se smatraju one Sudoku slagalice iz kojih se ne može obrisati ni jedno polje, a da ona i dalje ostane običan Sudoku. Mnoge obične Sudoku slagalice imaju po 17 predefinisanih projeva, iako pronalaženje njihovog rešenja nije lak posao. Najviše predefinisanih polja za minimalnu Sudoku slagalicu je 40. Ukoliko bi se bilo koje predefinisano polje uklonilo iz ove slagalice, ona bi imala više od jednog rešenja. Do sada je pronađeno samo dve minimalne Sudoku slagalice sa 40 predefinisanih polja. Jedna od njih je prikazana na slici 2. Rešivost Sudoku slagalici ne zavisi samo od količine predefinisanih polja, već i od načina na koji su brojevi raspoređeni.

	1	2		3	4	5	6	7
	3	4	5		6	1	8	2
		1		5	8	2		6
		8	6					1
	2				7		5	
		3	7		5		2	8
	8			6		7		
2		7		8	3	6	1	5

Slika 2

Određene Sudoku slagalice mogu da imaju cele redove, kolone i kvadrate od 3x3 polja prazne. Moguće je da se napravi takva Sudoku slagalica da ima prazne grupe (pod grupom se smatra ceo red, kolona ili kvadrat od 3x3 polja), ali se veruje da ukupan broj ovih praznih grupa ne može da bude veći od 9, inače slagalica neće biti obična (imaće više od jednog rešenja). Na Slici 2 je prikazana Sudoku slagalica koja ima 9 praznih sa jedinstvenim rešenjem. [3].

1	4		9	3				
9	3		1					
5	6							4
8	9						2	7
			2	5			1	8
			7	4			3	2

Slika 3

3. Nizovi

Za razliku od promenljivih, koje čuvaju pojedinačne vrednosti, nizovi se koriste za čuvanje skupa ili sekvence vrednosti. Svaki niz se sastoji od dva dela, indeksa i elementa. Indeks niza se koristi za identifikaciju i pristup elementu. U programskom jeziku PHP nizovi mogu da sadrže više tipova podataka. Veličina niza ne mora da se definiše pre korišćenja, i sama veličina niza je ograničena memorijom računara. Brojanje elemenata niza uvek počinje od nule.

Ovaj algoritam za rešavanje Sudoku slagalice se oslanja na dvodimenzionalne i trodimenzionalne nizove. Postoje i nizovi većih dimenzija, ali oni neće biti obrađeni u ovom radu. [4]

3.1. Jednodimenzionalni niz

Ova vrsta niza je najjednostavnija. U njoj se čuvaju podaci, razdvojeni zarezom. Da bi smo pristupili nekom elementu, potreban je samo jedan podatak, pozicija elementa u nizu. Da bi smo pristupili elementu, potrebno je da njegov redni broj zapišemo u uglaste zagrade pored imena niza.

3.2. Dvodimenzionalni niz

Ovaj niz predstavlja niz koji se sastoji od nizova, to jest, niz koji za svoje članove ima nizove. Na kraju, ti nizovi imaju neke podatke u njima. Ovakvi nizovi se još nazivaju i *matrice*. Dvodimenzionalni niz se sastoji od redova i kolona, i da bi smo pristupili jednom članu, potrebno je zadati redni broj reda i kolone.

Dvodimenzionalni niz savršeno odgovara Sudoku slagalici. Algoritam ga koristi da čuva konačno rešenje čitave slagalice. U početku, sadržaj matrice odgovara delimično rešenoj Sudoku slagalici, sa tim da su prazna mesta obeležena nulom.

Da bi algoritam znao koja polja su predefinisana, i koja ne sme da menja, postoji još jedna matrica. Ona se sastoji samo od binarnih vrednosti (nula i jedinica), koji govore da li se na datoj poziciji nalazi predefinisana vrednost. Ukoliko se nalazi, na tom mestu će se naći jedinica, i algoritam će da preskoči to polje u traženju rezultata.

3.3. Trodimenzionalni niz

Trodimenzionalni nizovi su slični dvodimenzionalnim, sa tom razlikom da najdublji niz za svoje vrednosti ima nizove. Ovi nizovi se mogu zamisliti da imaju redove, kolone i još dodatne „fioke“ za svakog člana. Da bi se pristupilo jednom članu trodimenzionalnog niza, potrebne su 3 informacije: redni broj kolone, redni broj reda, i redni broj člana.

Algoritam koristi trodimenzionalne nizove za čuvanje mogućih rešenja za svako polje. Ova moguća rešenja se generišu na početku, pre početka backtracing algoritma. Da bi smo dobili sva moguća rešenja za jedno polje, moramo znati u kom redu i u kojoj koloni se to polje nalazi. Backtracking algoritam onda proverava jedno po jedno rešenje.

4. Backtracking

Backtracking algoritme karakteriše traganje za rešenjem ne po zadatim pravilima rešavanja, već metodom pokušaja i greške. Ovi problemi se često najprirodnije rešavaju rekurzivno i zahtevaju ispitivanje konačnog broja problema. Zadaci koji se rešavaju backtrackingom, po pravilu pripadaju jednom od sledeće tri klase:

- Naći barem jedno moguće rešenje;
- Naći sva moguća rešenja;
- Po zadatom kriterijumu odrediti optimalno rešenje.

Jedan on najpopularnijih problema za backtracking je određivanje položaj N kraljica na ploči veličine $N \times N$ tako da ni jedna kraljica ne napada drugu. Ovaj problem se najlakše rešava backtracingom. Takođe se rešavanje Sudokua najlakše rešava ovom metodom [5].

Backtracking funkcioniše prilično jednostavno. Prvo se uzima jedna vrednost i pokušava da se postavi na prvo slobodno mesto (u primeru Sudokua, uzima se broj jedan, i postavlja se na prvo slobodno mesto). Ukoliko odabrana cifra odgovara na datoj poziciji, prelazi se na sledeće prazno mesto i pokušava se postaviti sledeći broj. Ukoliko se potroše sve mogućnosti za dato polje (u slučaju sudoka, ukoliko ni jedna cifra od 1 do 9 ne može da se smesti u prazno polje), algoritam se vraća na prethodno polje, i pokušava sledeću mogućnost. Ukoliko uspe da nađe odgovarajuću vrednost, nastavlja dalje, a ukoliko ne može, opet se vraća korak u nazad, i opet pokušava sledeću vrednost. Vraćanje u nazad se može izvršiti koliko god puta je potrebno, i moguće je doći do samog početka.

Za velike probleme ovaj način pretraživanja može da bude prilično vremenski zahtevno i zato postoji mogućnost dodavanja nekog kriterijuma za pronalaženje optimalnog rešenja. Na primer, moguće je ograničiti broj koraka vraćanja u nazad ili vreme provedeno u pretrazi.

Ovakvi problemi često mogu da imaju i više od jednog rešenja. Pomoću backtracking algoritma je moguće pronaći sva moguća rešenja.

Konkretno ovaj algoritam za pronalaženje Sudoku problema ima vremensko ograničenje, koje dolazi iz prirode servera. Svaka skripta koja se izvršava duže od 30 sekundi se automatski prekida. Takođe, algoritam pronalazi jedno rešenje, i ne može da zna da li postoje i druga rešenja.

Iako je ovaj algoritam za pretragu zasnovan na backtrackingu, ima mala poboljšanja, jer ne proverava sve moguće brojeve, već samo one koji su mogući u određenim poljima. Takođe, u koliko je u jedno polje moguće upisati samo jednu cifru, ta cifra se upisuje i više se ne proverava. U tom slučaju ta cifra se smatra kao predefinisanim.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7			9	2				6
	6					2	8	
			4	1	9			5
				8			7	9

Slika 4

Posmatrajmo Sudoku slagalicu sa Slike 4. Trenutno je unešeno 30 predefinisanih polja. Na ovom primeru će biti prikazan backtracking algoritam. Zamislamo da je algoritam započeo sa radom, i da se zaustavio u četvrtom redu, četvrtoj koloni, kao na Slici 5.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	2	3	4	5	6	7
8	1	5		6				3
4			8		3			1
7			9	2				6
	6					2	8	
			4	1	9			5
				8			7	9

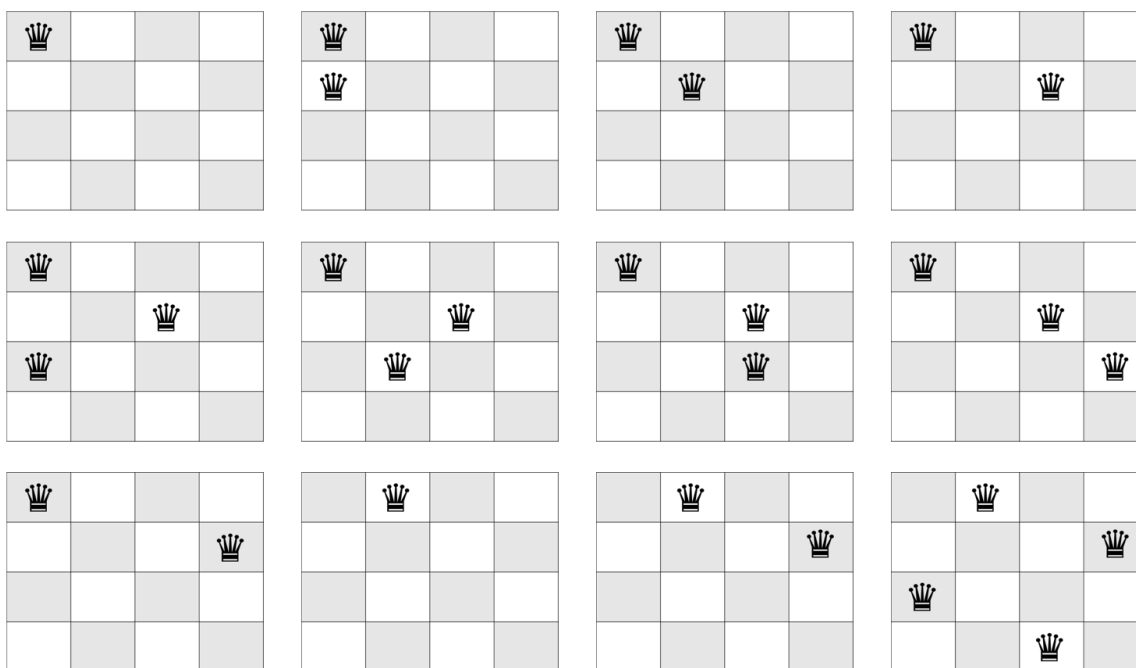
Slika 5

U sledeće polje je moguće upisati 5 i 7. Budući da je cifra pet već zauzeta u tom redu, proverava se broj sedam. Njega je moguće upisati, i on se i upisuje. U sledeće slobodno polje moguće je upisati 1, 4 i 7. Algoritam prvo upisuje prvi broj. On ne može da stoji u tom polju, jer se već javlja u istom redu. Zatim se prelazi na sledeći broj, četiri. On se javlja u koloni, pa ni njega nije moguće upotrebiti. Na kraju dolazi sedam. On se takođe pojavljuje u istoj koloni, pa ni ovu cifru nije moguće iskoristiti. U ovom slučaju dolazi do backtrackinga. Algoritam briše poslednje upisan broj u toj ćeliji (sedam), i vraća se korak u nazad. U tom polju je poslednje upisao sedam. To je zapamtio, i nastavlja sa proverom brojeva od sedmice. Ali, budući da je iskoristio sve moguće brojeve za to polje, mora još jednom da se vrati, u četvrti red, drugu kolonu, ali pre toga prazni polje u kome se trenutno nalazi. U tom polju mogu da stoje brojevi 1, 2 i 5. Budući da je algoritam već probao jedinicu, nju trenutno ignoriše i prelazi na sledeći broj – dva. Zatim prelazi na sledeće polje gde može da upiše jedinicu. Iako u ovom polju mogu da stoje cifre 1, 2, 5 i 9, i do sad je stajala petica, budući da petica, kao ni svaki sledeći broj nisu odgovarali tom polju, prilikom vraćanja na prethodno polje i njegovo menjanje, provera kreće od početka.

Algoritam se ovako kreće i zaustavlja se u dva slučaja. Prvi slučaj je ako dođe do poslednjeg polja, i u njega uspešno upiše broj. To znači da je slagalica uspešno rešena.

Drugi slučaj je ukoliko backtrackingom dođe do prvog polja, i iskoristi sve moguće cifre za to polje. Ukoliko se to desi, to znači da ta slagalica nije rešiva, tojest da nema rešenje.

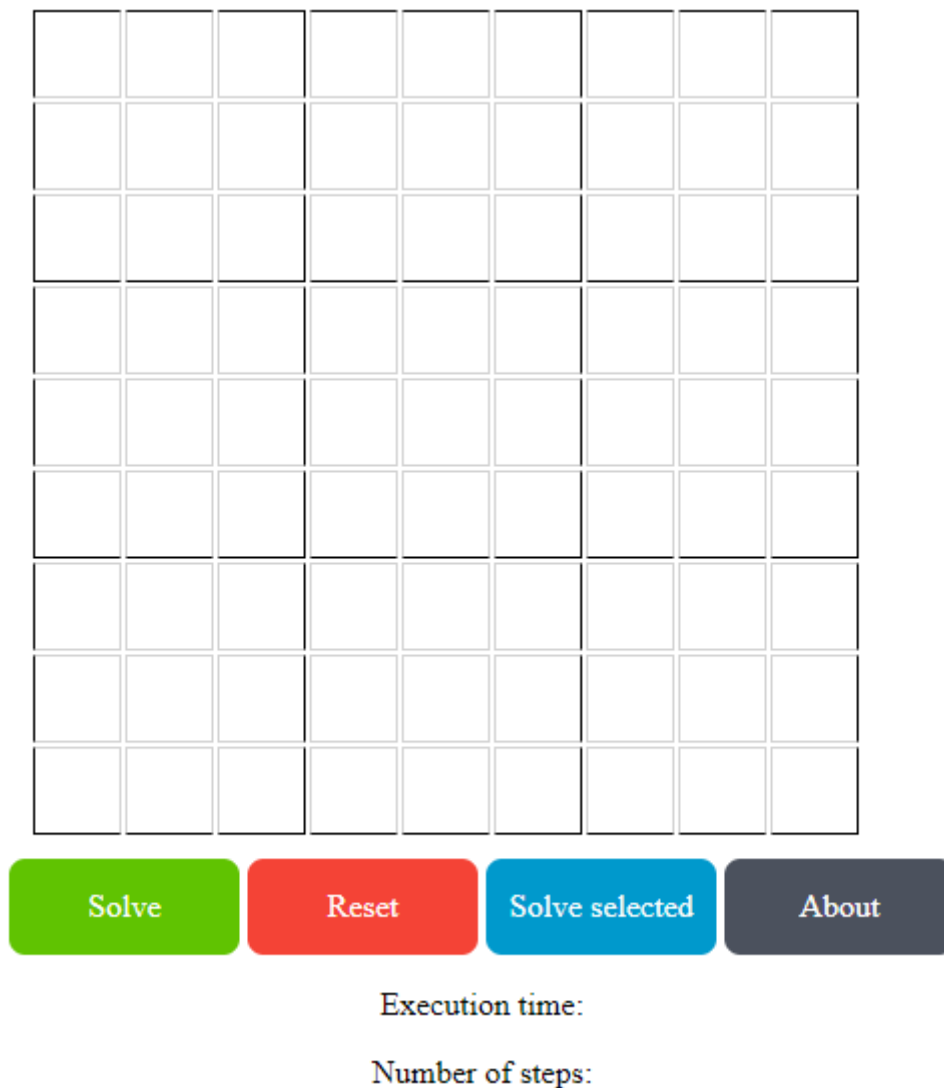
Još jedan primer za backtracking je problem sa N kraljica. Cilj je postaviti N kraljica na polje veličine $N \times N$, tako da se se ni jedna kraljica ne napada (to jest da se u jednom redu, jednoj koloni i dijagonali pojavi samo jedna kraljica). Najmanji mogući broj koji N može da bude je 4, pa će i ovaj primer da bude baziran na 4 kraljice. Prvo će se u prvi red u prvu kolonu staviti jedna kraljica. Budući da ona može da stoji u toj koloni, prelazi se na sledeći red. U sledećem redu se stavlja sledeća kraljica na prvo polje. Budući da ne može tamo da bude postavljena pomera se na sledeće polje. Kraljice se napadaju i po dijagonali, pa je drugu kraljicu potrebno pomeriti za još jedno mesto. Budući da ovde može da stoji, prelazi se na sledeći red i sledeću kraljicu. Kraljica u trećem redu, u ovom trenutku ne može da stoji ni u jendom polju, jer je u svakom polju napadnuta. U ovom trenutku dolazi do backtrackinga. Kraljica iz prethodnih reda se pomera za jedno polje u napred. U ovom polju može da stoji, pa se prelazi na sledeći red. Budući da se prethodna kraljica pomerila za jedno polje, ova kraljica mora da ispita sva polja od početka. Ni u ovom lučaju kraljica u trećem redu ne može da nađe „slobodno“ mesto, pa je opet dolazi do backtrackinga. Druga kraljica više nema gde da se pomeri, pa je potreban još jedan korak u nazad. Prva kraljica se pomera za jedno mesto. U ovom trenutku, svaka kraljica može da bude smeštena tako da ne bude napadnuta. Proces je prikazan na slici 6.



Slika 6

5. Aplikacija

Aplikacija za rešavanje Sudokua (Slika 7) se nalazi na sledećoj adresi: <http://ai.jovanoskibojan.com/>. Dostupna je za svakoga, i trenutno je na engleskom jeziku. *Front – end* je pravljen koristeći *HTML*, *CSS*, kao i pomoću *jQuery* – ja. Backend, kao i celokupan algoritam za rešavanje je napravljen u *PHPu*.



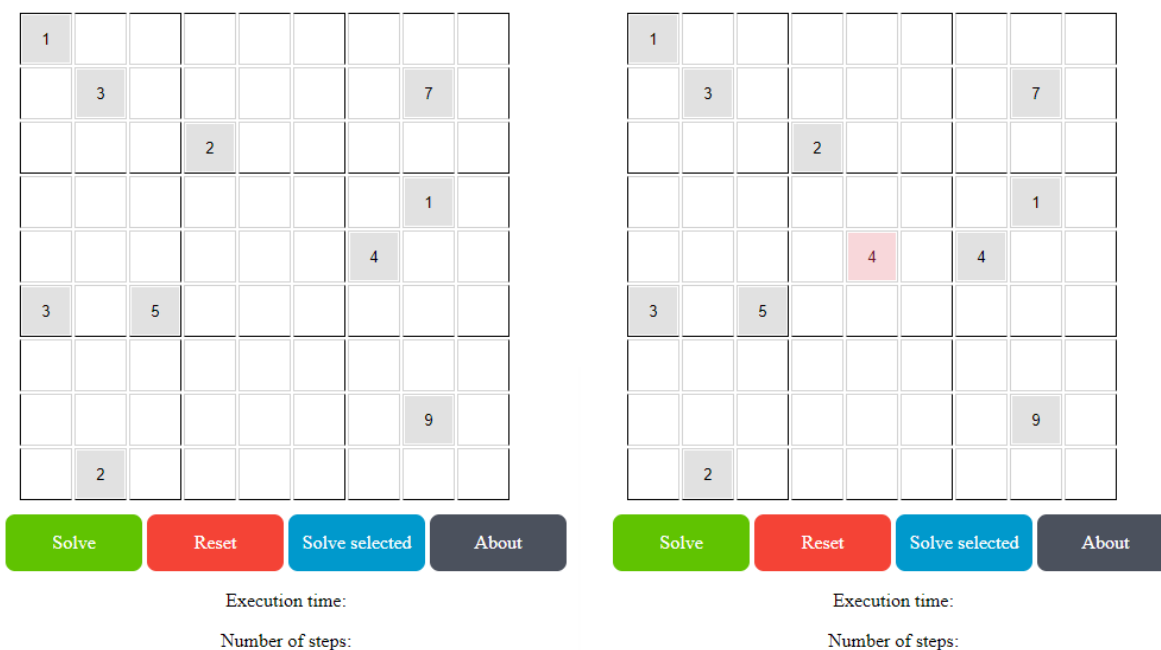
SolveResetSolve selectedAbout

Execution time:

Number of steps:

Slika 7

Aplikacija je pravljena da bude jednostavna za korišćenje. Prvo što se primeti su polja za unos brojeva. Korisnik treba da unese početne brojeve u željena polja. Sva polja koja sadrže validne brojeve će promeniti pozadinu u svetlo sivu boju. Ukoliko neko polje sadrži ilegalan broj, polje će da pocrveni, da stavi korisniku do znanja da je uneo pogrešan broj (Slika 8).



Slika 8

Da bi se moglo pristupiti rešavanju slagalice, polja ne smeju da sadrže pogrešno unesen broj. Ukoliko se pokuša pristupiti rešavanju dok postoje nepravila polja, aplikacija javlja da postoji greška, i da se ta polja moraju ispraviti. Kada su sva željena polja unešena, potrebno je kliknuti zeleno *Solve* dugme. Posle nekoliko trenutaka, pojavljuje se rešenje u belim poljima (Slika 9). Aplikacija takođe prikazuje koliko je vremena algoritmu trebalo da reši problem, kao i ukupan broj koraka koji je trebalo algoritmu da dođe do rešejna (broj ukupnih koraka, broj koraka u nazad, i broj koraka u napred). U prikaz vremena ne spada vreme koje je potrebno da se podaci pošalju sa lokalnog računara na server, kao ni prikaz rešenja, već samo vreme koje je potrebno serveru da izvrši algoritam za pretragu rešenja. Takođe je prikazano koliko se puta algoritam najviše pomerio nazad, odnosno napred u kontinuitetu, takoreći, koliko polja je bio najveći „skok“. Pod jednim korakom se smatra pomeranje sa jednog polja na drugo, bez obzira da li je u napred ili u nazad.

1	4	2	3	5	7	6	8	9
5	3	6	1	8	9	2	7	4
7	8	9	2	4	6	1	3	5
2	6	4	5	3	8	9	1	7
8	1	7	6	9	2	4	5	3
3	9	5	4	7	1	8	2	6
4	5	1	9	2	3	7	6	8
6	7	3	8	1	4	5	9	2
9	2	8	7	6	5	3	4	1



Execution time: 0.09991 sec

Number of steps: **139** (Backtracking steps: **27**; forward steps: **112**).

Longest streak of back steps is **6**
 and longest streak of forward steps is **14**.

Slika 9

Ukoliko se klikne na crveno dugme *Reset*, cela slagalica se briše i vraća se na početno stanje.

Aplikacija takođe pruža asistenciju prilikom rešavanja. Ukoliko korisnik ne želi da vidi rešenje cele slagalice, već samo sva moguća rešenja za jedno polje, potrebno je obeležiti željeno polje, koje dobija plavu pozadinu. Nakon toga, potrebno je kliknuti plavo *Solve selected* polje, koje vraća sva moguća rešenja za odabrano polje u tom trenutku (Slika 10).

ai.jovanoskibojan.com says

Possible solutions for selected field are 4, 6, 7, 8 and 9

OK

1								
	3						7	
			2					
							1	
						4		
3		5						
							9	
	2							

Solve

Reset

Solve selected

About

Execution time:

Number of steps:

Slika 10

Klikom na dugme *About* se dobijaju informacije o korišćenju programa, o korištenim tehnologijama, upustvo za korišćenje i druge potrebne informacije.

6. Kôd programa

Kod se sastoji iz dva dela: front – end i back – end koda. Front – end je zadužen za prikaz polja za unos, proveru unesenih podataka, slanje podataka na server i prikaz rešenja. Back – end je zadužen za obradu svih podataka i traženje rešenja.

6.1. Front – end

Prilikom samog prizaka polja se nalazi jedna jednostavna petlja, koja prikazuje redove i kolone polja za unos. Ovo se postiže koristeći kombinacijom PHP – a i HTML – a.

```
1. <table id="sudoku">
2.   <tbody>
3.     <?php
4.       $id = 0;
5.       for($i = 0; $i < 9; $i++) {
6.         echo "\t\t\t<tr>\r\n";
7.         for($p = 0; $p < 9; $p++) {
8.           echo "\r\n\t\t\t" . '<td><input value="" id="id_' . $i . $p . '" type="text" name'
9.             . "value[]" data-row="'" . $i . '" data-col="'" . $p . '"></td>';
10.        }
11.        echo "\r\n\t\t\t</tr>\r\n";
12.        $id++;
13.      }
14.    </tbody>
15. </table>
```

Zatim sledi jQuery kod. Prvi deo proverava da li je uneseni karakter cifra, ili neki drugi ne dozvoljeni karakter:

```
1. // Check if the entered character is number
2. $("input").keydown(function (e) {
3.   var car_num = $(this).val().length;
4.   // Allow: backspace, delete, tab, escape, enter and .
5.   if ($.inArray(e.keyCode, [46, 8, 9, 27, 13, 110]) !== -1 ||
6.     // Allow: Ctrl+A, Command+A
7.     (e.keyCode === 65 && (e.ctrlKey === true || e.metaKey === true)) ||
8.     // Allow: home, end, left, right, down, up
9.     (e.keyCode >= 35 && e.keyCode <= 40)) {
10.    // let it happen, don't do anything
11.    $(this).removeClass("selected_field");
12.    return;
13.  }
14.  // Ensure that it is a number and stop the keypress
15.  if ((e.shiftKey || (e.keyCode < 49 || e.keyCode > 57)) && (e.keyCode < 97 || e.keyCode > 105) || (car_num >= 1)) {
16.    $(this).removeClass("selected_field");
17.    e.preventDefault();
18.  }
19. });
```

Zatim sledi deo koji šalje podatke na server, i proverava da li je unesena cifra validna u tom polju. Skripta vraća broj konflikata, i ukoliko je taj broj 0, znači da upisana cifra može da bude u tom polju:

```
20. // Checks if entered number is valid in that field
21. $("input").keyup(function() {
22.     var field = $(this);
23.     var this_val = $(this).val();
24.     $("#row").val($(this).attr("data-row"));
25.     $("#col").val($(this).attr("data-col"));
26.     field.removeClass("error");
27.     if(this_val !== "" || this_val.length > 0) {
28.         $.post("php/check_current.php", $('#sudoku_inputs').serialize(), function(data){
29.             if(data != 0) {
30.                 field.addClass("error");
31.                 $(this).removeClass("selected_field");
32.             }
33.             else {
34.                 field.removeClass("error");
35.                 field.addClass("manual_number");
36.             }
37.         });
38.     }
39.     if(this_val.length === 0)
40.         field.removeClass("manual_number");
41. });
```

Sledi deo koda koji proverava koje sve cifre mogu da se nađu u odabranom polju. Ovaj deo koda se poziva kada korisnik klikne na dugme *Solve selected*. Ceo tekst se formatira na serveru, tako da jQuery samo treba da pokaže rezultat.

```
42. // Solving selected
43. $(".selected").click(function() {
44.     var selected_field = $(".selected_field");
45.     var row = selected_field.attr("data-row");
46.     var col = selected_field.attr("data-col");
47.     if(row == null && col == null) {
48.         alert("Select an empty field first");
49.     }
50.     else {
51.         $("#row").val(row);
52.         $("#col").val(col);
53.         $.post("php/check_selected.php", $('#sudoku_inputs').serialize(), function(data){
54.             alert(data);
55.         });
56.     }
57. });
```

Na kraju dolazi deo koda za slanje podataka na rešavanje. Prvo se dugme za slanje onemogućí, da nije moguće ponovno slanje kad se dugme klikne jednom. Kada se podaci prime sa servera, razdvajaju se podaci o utrošenom vremenu, koracima, kao i samo rešenje slagalice. Pomoću jedne petlje, rešenje se upisuje u tabelu pored onih koje je korisnik uneo. Ukoliko postoje neka crvena polja, tj. polja sa greškom, korisnik dobija obaveštenje, i podaci se ne šalju na server.

```
58. //If there are no errors, data is sent for calculation
59. $(".send").click(function() {
60.     $('.send').prop('disabled', true);
61.     $('.send').addClass('submit-disabled');
62.     var errors = $('.error').length;
63.     var solution;
64.     var steps;
65.     var execution_time;
66.     if(errors == 0) {
67.         $.post("php/solve_sudoku.php", $('#sudoku_inputs').serialize(), function(data){
68.             $('.send').prop('disabled', false);
69.             $('.send').removeClass('submit-disabled');
70.             data = JSON.parse(data);
71.             if(data["solution_found"] == "0") {
72.                 alert(data["message"]);
73.             }
74.             else {
75.                 solution = data["matrix"];
76.                 steps = data["steps"];
77.                 execution_time = data["_time"];
78.                 for (i = 0; i < solution.length; i++) {
79.                     for (q = 0; q < solution[i].length; q++) {
80.                         $("#id_" + i + q).val(solution[i][q]);
81.                     }
82.                 }
83.                 $("#execution_time").html(execution_time.toFixed(5) + " sec");
84.                 $("#step_number").html(steps);
85.             }
86.         });
87.     }
88.     else {
89.         alert("There are errors");
90.     }
91. });
```

Postoje i drugi, manje bitni delovi programa koji ovde nisu prikazani. Oni su jednostavniji i manje bitni za celokupno rešavanje slagalice. Ne prikazani delovi koda služe za dodavanje pozadinske boje kada korisnik obeleži određeno polje i resetovanje forme na početno stanje.

6.2. Back – end

Back – end je podeljen u više funkcija koje su potrebne za uspešno rešavanje problema.

6.2.1. Kreiranje niza

jQuery podatke šalje serveru kao običan tekst. Da bi se rad sa podacima olakšao, ti podaci će biti konvertovani u dvodimenzionalni niz:

```
1. function create_matrix($numbers) {  
2.     $matrix = [];  
3.     $tmp = 9;  
4.     $count = 0;  
5.     $tmp_array = [];  
6.     foreach($numbers as $number) {  
7.         array_push($tmp_array, $number);  
8.         $count++;  
9.         if($count % $tmp == 0) {  
10.            array_push($matrix, $tmp_array);  
11.            $tmp_array = [];  
12.        }  
13.    }  
14.    return $matrix;  
15. }
```

6.2.2. Predefinisana polja

Da bi softver znao koja polja je potrebno da menja, potrebno je nekako naznačiti koja polja se ne smeju menjati, odnosno ona koja su predefinisana. Zato se kreira jedan dvodimenzionalni niz koji govori da li je u polju konstanta (1), ili se polje sme menjati (0):

```
1. function createGrid($matrix) {  
2.     $createdGrid = [[], []];  
3.     for($p = 0; $p < 9; $p++) {  
4.         for($q = 0; $q < 9; $q++) {  
5.             if($matrix[$p][$q] == 0)  
6.                 $createdGrid[$p][$q] = 0;  
7.             else  
8.                 $createdGrid[$p][$q] = 1;  
9.         }  
10.    }  
11.    return $createdGrid;  
12. }
```

6.2.3. Validacija polja

Funkcija za validaciju polja funkcioniše tako što prima celu matricu sa unetim podacima, i proverava da li određeno polje može da stoji na tom mestu. Provera kolone i reda je jednostavna, ukoliko se proverava kolona, red je fiksiran, i program prolazi kroz sve kolone. Za redove je obrnuto. 3x3 blok se proverava tako što se red i kolona pomeraju levo ili desno za jedno, dva ili ni jedno polje

```
1. function validate($matrix, $row, $col) {
2.     $current = $matrix[$row][$col];
3.     $matches = 0;
4.     // Checking for row
5.     for($i = 0; $i < 9; $i++) {
6.         if($matrix[$row][$i] == $current)
7.             $matches++;
8.     }
9.
10.    // Checking for column
11.    for($i = 0; $i < 9; $i++) {
12.        if($matrix[$i][$col] == $current)
13.            $matches++;
14.    }
15.
16.    // Checking for the 3x3 box
17.    $position_row = $row%3;
18.    $position_col = $col%3;
19.    $left = $position_col;
20.    $right = 2 - $position_col;
21.    $up = $position_row;
22.    $down = 2 - $position_row;
23.    $tmp = array();
24.
25.    for($i = 1; $i <= $left; $i++) {
26.        $rows_left = $col - $i;
27.        for($p = 1; $p <= $down; $p++) {
28.            $columns_left = $row + $p;
29.            array_push($tmp, $matrix[$columns_left][$rows_left]);
30.        }
31.        for($p = 1; $p <= $up; $p++) {
32.            $columns_left = $row - $p;
33.            array_push($tmp, $matrix[$columns_left][$rows_left]);
34.        }
35.    }
36.    for($i = 1; $i <= $right; $i++) {
37.        $rows_left = $col + $i;
38.        for($p = 1; $p <= $down; $p++) {
39.            $columns_left = $row + $p;
40.            array_push($tmp, $matrix[$columns_left][$rows_left]);
41.        }
42.        for($p = 1; $p <= $up; $p++) {
43.            $columns_left = $row - $p;
44.            array_push($tmp, $matrix[$columns_left][$rows_left]);
45.        }
46.    }
47.
48.    foreach($tmp as $check) {
49.        if($check == $current)
50.            $matches++;
51.    }
52.
53.    return $matches -= 2;
54. }
```

6.2.4. Kreiranje mogućih rešenja

Da softver ne bi morao da probava redom svaki broj u svakoj slobodnoj ćeliji, kreiraju se moguća rešenja. Ona se čuvaju u trodimenzionalnom nizu. Nulti član svakog niza predstavlja redni broj elementa niza koji treba da se proverí sledeći. Na primer, ukoliko su moguća rešenja 1, 3, 8 i 9, nulti član će uvek biti prvo jedan. Ukoliko prvi član niza (1), ne odgovara, nulti član niza će da se poveća za jedan, i sledeći put će se proveravati drugi član: 3

```
1. function possibleSolutions($matrix, $matrixGrid) {
2.     $possible_solutions = [[], [], []];
3.     for($i = 0; $i < 9; $i++) {
4.         for($j = 0; $j < 9; $j++) {
5.             if($matrixGrid[$i][$j] == 0) {
6.                 for($s = 1, $p = 0; $s < 10; $s++) {
7.                     $matrix[$i][$j] = $s;
8.                     $tmp = validate($matrix, $i, $j);
9.                     if($tmp == 0) {
10.                        // First integer of array stores the position that needs to be checked next
11.
12.                        $possible_solutions[$i][$j][0] = 1;
13.                        array_push($possible_solutions[$i][$j], $s);
14.                        continue;
15.                    }
16.                    $matrix[$i][$j] = 0;
17.                }
18.            }
19.        }
20.     return $possible_solutions;
21. }
```

6.2.5. Rešavanje problema

Kad su sve pomoćne funkcije definisane, moguće je pristupiti rešavanju problema. Prvo se preuzimaju podaci koje je jQuery prosledio, i pozivaju se funkcije za kreiranje potrebnih nizova. Takođe se definiše trenutno vreme u milisekundama, koje je potrebno da bi se znalo koliko je algoritmu trebalo da nađe rešenje.

```
1. $numbers = $_POST["value"];
2.
3. require 'create_matrix.php';
4. require 'create_grid.php';
5. require 'validate.php';
6. require 'create_possible_solutions.php';
7. // Creates an array from plain numbers from JS
8. $matrix = create_matrix($numbers);
9. // Creates a grid of numbers that are fixed (1), and the ones that needs changing (0)
10. $matrixGrid = createGrid($matrix);
11. // Creates an array of possible solutions for each field in 3d array
12. $possibleSolutions = possibleSolutions($matrix, $matrixGrid);
13.
14. // Starting measure of time
15. $start = microtime(true);
```

Ukoliko trodimenzionalni niz koji sadrži moguća rešenja ima samo jednu vrednost, to znači da je u toj ćeliji moguća samo jedna vrednost. U tom slučaju, ta vrednost se upisuje u niz i prema njoj se ophodi kao da je vrednost predefinisana. Ta rešenja se uklanjaju iz niza.

```
16. // Creating possible solutions
17. for($r = 0; $r < 100; $r++) {
18.     for($i = 0; $i < 9; $i++) {
19.         for($j = 0; $j < 9; $j++) {
20.             if($matrixGrid[$i][$j] == 0) {
21.                 $possible_solutions_size = sizeof($possibleSolutions[$i][$j]);
22.                 if($possible_solutions_size == 2) {
23.                     $matrix[$i][$j] = $possibleSolutions[$i][$j][1];
24.                     $matrixGrid[$i][$j] = 1;
25.                     unset($possibleSolutions[$i][$j]);
26.                 }
27.             }
28.         }
29.     }
30.     $possibleSolutions = possibleSolutions($matrix, $matrixGrid);
31. }
```

Nakon toga, program prolazi kroz sve ćelije koje nisu predefinisane. Uzima prvu vrednost iz niza mogućih rešenja, i nastavlja na sledeću ćeliju. Ukoliko se desi da je iskoristio sve moguće vrednosti za tu ćeliju, znači da prethodna ćelija nema pravu vrednost. U tom slučaju se vraća na prethodnu ćeliju i uzima sledeću vrednost iz niza mogućih rešenja za tu ćeliju. Ukoliko je softver došao do početka, i ukoliko nema prethodnih ćelija čije vrednosti može da menja, smatra se da Sudoku puzla nerešiva, i pojavjuje se greška. Takođe, prilikom prelaska na sledeće ili prethodno polje, brojač se povećava kako bi znali koliko koraka je algoritmu ukupno trebalo da dođe do rešenja. Takođe se proverava koliko koraka je algoritam išao samo u jednom smeru. U ovom delu koda se vrši i provera utrošenog vremena. Budući da je vreme za izvršavanje limitirano na 30 sekundi, proverava se da li je toliko vremena isteklo. Ukoliko jeste, pojavljuje se greška.


```
32. $i = 0;
33. $j = 0;
34. $counter = 0;
35. $back_counter = 0;
36. $back_counter_tmp = 0;
37. $back_counter_max = 0;
38. $forward_counter = 0;
39. $forward_counter_tmp = 0;
40. $forward_counter_max = 0;
41. while ($i < 9) {
42.     while ($j < 9) {
43.         $back = false;
44.         $found = false;
45.         $counter++;
46.         if($matrixGrid[$i][$j] == 0) {
47.             $lastChecked = $possibleSolutions[$i][$j][0];
48.             for($c = $lastChecked; $c < sizeof($possibleSolutions[$i][$j]); $c++) {
49.                 $matrix[$i][$j] = $possibleSolutions[$i][$j][$c];
50.                 $tmp = validate($matrix, $i, $j);
51.                 if($tmp == 0) {
52.                     $found = true;
53.                     break;
54.                 }
55.                 else {
56.                     $found = false;
57.                 }
58.             }
59.             $possibleSolutions[$i][$j][0] = ($c + 1);
60.             if($found == false) {
61.                 $possibleSolutions[$i][$j][0] = 1;
62.                 $matrix[$i][$j] = 0;
63.                 do {
64.                     $j--;
65.                     if($j < 0) {
66.                         $j = 8;
67.                         $i--;
68.                         if($i < 0) {
69.                             $i = 0;
70.                             $results = [
71.                                 solution_found => "0",
72.                                 message => "Solution is not found, sudoku is probably impossible to
solve. Number of steps tried: " . $counter,
73.                             ];
74.                             $send_data = json_encode($results);
75.                             ob_clean();
76.                             echo $send_data;
77.                             die;
78.                         }
79.                     }
80.                 } while ($matrixGrid[$i][$j] == 1);
81.                 $back = true;
82.             }
83.         }
84.         if($back == false) {
85.             $j++;
86.             $forward_counter++;
87.             $forward_counter_tmp++;
88.         }
89.         else {
```

```
90.         $back_counter++;
91.         $back_counter_tmp++;
92.     }
93.     if($back_counter_max < $back_counter_tmp) {
94.         $back_counter_max = $back_counter_tmp;
95.         $back_counter_tmp = 0;
96.     }
97.     if($forward_counter_max < $forward_counter_tmp) {
98.         $forward_counter_max = $forward_counter_tmp;
99.         $forward_counter_tmp = 0;
100.    }
101.    }
102.    if($back == false)
103.        $i++;
104.    $j = 0;
105.    $loop_time = microtime(true) - $start;
106.    if($loop_time >= 28) {
107.        $results = [
108.            solution_found => "0",
109.            message => "Maximum execution time exceeded, solution not found. Number
of steps tried: " . $counter,
110.        ];
111.        $send_data = json_encode($results);
112.        ob_clean();
113.        echo $send_data;
114.        die;
115.    }
116. }
```

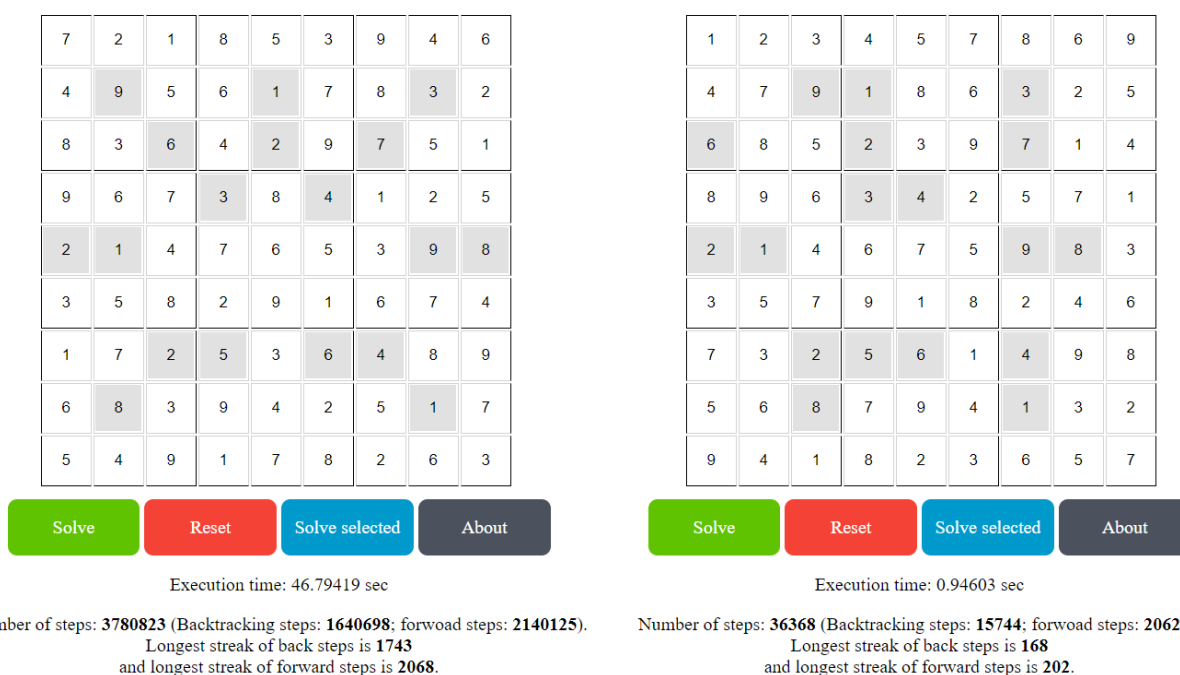
Na kraju se pravi nov niz, u koji se dodaje rešenje, broj koraka kao i utrošeno vreme. Ovaj niz se konvertuje u JSON, i takav se šalje jQuery, na ispis korisniku.

```
117.     $time_elapsed_secs = microtime(true) - $start;
118.     $results = [
119.         "solution_found" => "1",
120.         "matrix" => $matrix,
121.         "steps" => $counter,
122.         "_time" => $time_elapsed_secs,
123.         "backs" => $back_counter,
124.         "back_max" => $back_counter_tmp,
125.         "forwards" => $forward_counter,
126.         "forward_max" => $forward_counter_max,
127.     ];
128.     $send_data = json_encode($results);
129.     ob_clean();
130.     echo $send_data;
```

7. Vremenski zahtevi

Vreme rešavanja Sudoku slagalice zavisi od broja datih cifara, kao i od mogućih rešenja za svako polje. Takođe, jedan od faktora koje utiču na vreme je i način na koji su predefinisane cifre raspoređene.

Na Slici 11 možemo da vidimo dve slagalice, na prvi pogled jako slične. Desna slagalica je malo više nasumično raspoređena, dok su na levoj strani brojevi raspoređeni po *ortogonalnoj simetriji*. Razlika u vremenu potrebnog za rešavanje je drastična, i iznosi prilično 46 sekundi, a razlika u broju potrebnih koraka je 3.744.455. Ista problematika se javlja i kod slagalica sa diagonalnom simetrijom, automorfnih slagalica i sličnih problema. Rešavanje levog problema je moguće samo na lokalnom serveru, gde je maksimalno vreme za pokretanje skripte povećano na 2 minuta, dok na „live“ serveru ovu opciju nije moguće menjati, i maksimalno vreme izvršavanja je 30 sekundi.

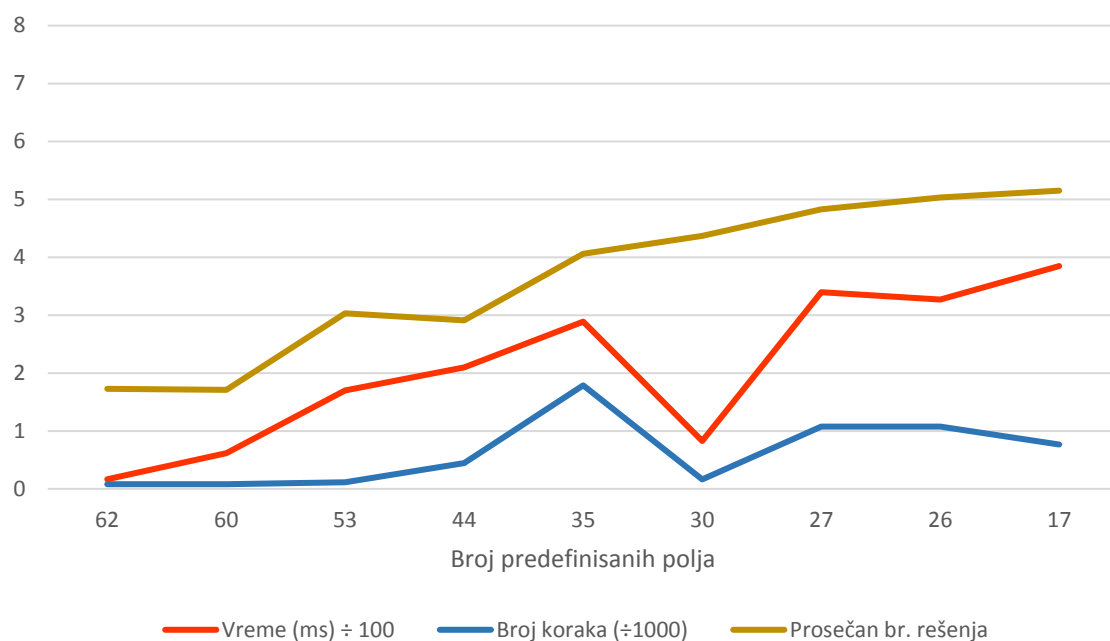


Slika 11

U Tabeli 1 su prikazana vremena potrebna za izvršavanje i broj ukupnih koraka u zavisnosti od ukupnog broja popunjenih polja. Takođe je prikazan i prosečan broj mogućih rešenja za jendo polje

Broj polja:	62	60	53	44	35	30	27	26	17
Vreme (ms)	16,6	61,52	169,92	209,96	289,06	82,61	339,84	327,15	384,77
Broj koraka	81	81	112	443	1788	166	1074	1074	765
Prosečan br. rešenja	1,73	1,71	3,03	2,91	4,06	4,37	4,83	5,03	5,15

Tabela 1



Iz priloženog grafika se vidi da, kako opada broj predefinisanih polja, raste prosečan broj rešenja po polju. Samim tim, algoritam ima više mogućnosti da ispita, pa i vreme raste kako broj polja opada. Broj koraka je najmanji na početku, jer je mnogo polja popunjeno, pa ne mora mnogo da se vraća nazad. Ukoliko postoji malo predefinisanih polja, takođe je broj koraka manji jer ima veći izbor brojeva za rešenje. Grafik je nastao merenjem 5 različitih sudoku slagalica za svaki broj predefinisanih polja posebno.

8. Zaključak

Backtracking algoritam je idealno rešenje za rešavanje Sudoku slagalica. U mnogim slučajevima se pokazalo efikasno i brzo, sa malim brojem koraka. U retkim slučajevima je algoritmu potrebno dugo vremena za kompletiranje, ali budući da je ova aplikacija namenjena pomaganju u rešavanju svakodnevnih slagalica, takvi problemi će veoma retko da naiđu.

Literatura

- [1] <https://sh.wikipedia.org/wiki/Sudoku>
- [2] <https://en.wikipedia.org/wiki/Sudoku>
- [3] https://en.wikipedia.org/wiki/Mathematics_of_Sudoku
- [4] Dražen Drašković, asistent Elektrotehnički fakultet Univerziteta u Beogradu
- [5] Milan Čabarkapa – Nastava računarstva