Web aplikacija logičke igre Kakuro

Jurić, Ivan

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:602406

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

zir.nsk.hr

Image not found or type unknown

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij

WEB APLIKACIJA LOGIČKE IGRE KAKURO

Diplomski rad

Ivan Jurić

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 06.07.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Ivan Jurić
Studij, smjer:	DRD - Informacijske i podatkovne znanosti
Mat. br. studenta, godina upisa:	D 780 R, 09.10.2015.
OIB studenta:	15452218651
Mentor:	Doc.dr.sc. Tomislav Rudec
Sumentor:	Doc.dr.sc. Irena Galić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Alfonzo Baumgartner
Član Povjerenstva:	Doc.dr.sc. Irena Galić
Naslov diplomskog rada:	Web aplikacija logičke igre Kakuro
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U aplikaciji će biti omogućen ručni unos kakura i unos kakura sa slike. Aplikacija će biti povezana s bazom podataka pomoću koje će se analizirati brzina izvođenja različitih primjeraka igre.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	06.07.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 17.07.2017.

Ime i prezime studenta:	Ivan Jurić
Studij:	DRD - Informacijske i podatkovne znanosti
Mat. br. studenta, godina upisa:	D 780 R, 09.10.2015.
Ephorus podudaranje [%]:	5 %

Ovom izjavom izjavljujem da je rad pod nazivom**: Web aplikacija logičke igre Kakuro**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Rudec

i sumentora Doc.dr.sc. Irena Galić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Zahvale

Zahvaljujem se mentoru doc.dr.sc. Tomislavu Rudecu na ukazanom povjerenju, vodstvu i savjetima tijekom izrade ovog rada.

Zahvaljujem se sumentorici doc.dr.sc. Ireni Galić na savjetima u području obrade slike i prepoznavanja znakova.

Također zahvaljujem se svim profesorima i asistentima sa smjera Računarstvo koji su me tijekom studija pripremili za izradu ovog rada.

Hvala svim mojim radnim kolegama i kolegama iz fakultetskih klupa koji su mi svojim društvom omogućili ugodniji i lakši završetak studija.

Na kraju zahvalio bi svojim roditeljima i braći koji su mi bili velika podrška tijekom mog studiranja.

Hvala Vam!

Sadržaj

1. UVOD 1
   1.1. O kakuru 1
2. RJEŠAVANJE 2
   2.1. Pravila 3
   2.2. Metode 4
      2.2.1. Magični brojevi 4
      2.2.2. Okomiti i vodoravni zbrojevi 6
      2.2.3. Isključivanje 7
3. KORIŠTENE TEHNOLOGIJE 8
   3.1. .NET Framework 8
   3.2. MVC 9
   3.3. Entity Framework 10
   3.4. Tesseract OCR 12
   3.5. D3.js 13
4. STRUKTURA APLIKACIJE 14
   4.1. Baza podataka 15
   4.2. Modeli 16
      4.2.1. Modeli baze podataka 16
      4.2.2. Modeli za prikaz 18
   4.3. Upravljač 20
   4.4. Pogledi 23
   4.5. Optičko prepoznavanje znakova 26
      4.5.1. Pretprocesiranje 26
      4.5.2. Prepoznavanje znakova 29
   4.6. Algoritam 30
      4.6.1. Opis 30
      4.6.2. Početak 31
      4.6.3. Iteracije 31
      4.6.4. Završetak 32
5. RAD APLIKACIJE 33
   5.1. Rješavač 33
      5.1.1. Ručni unos 33
      5.1.2. Unos sa slike 35
      5.1.3. Ispis rješenja 36

5.2. Pomagač 37
5.3. Analiza 38
5.3.1. Učitavanje 38
5.3.2. Rješavanje 39
6. ZAKLJUČAK 41
7. LITERATURA 42
8. SAŽETAK 43
9. ŽIVOTOPIS 44
10. PRILOZI 45

1. UVOD

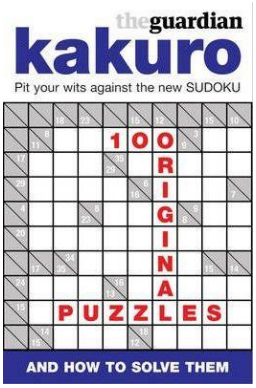
Tema diplomskoga rada izrada je web aplikacije logičke igre kakuro. Aplikacija korisniku omogućuje unos kakura, a zatim, pomoću algoritma, rješava kakuro i ispisuje rješenje. Ukoliko algoritam ne daje potpuno rješenje kakura, ispisuje se djelomično rješenje odnosno ispisuju se samo poznate vrijednosti polja.
Osim rješavanja kakura aplikacija korisniku omogućuje pomoć pri rješavanju na način da ispisuje moguće kombinacije za zadani broj polja i zbroj. Aplikacija omogućuje analizu brzine učitavanja i rješavanja različitih primjeraka kakura.

1.1. O kakuru

Kakuro se pojavljuje 1966. godine u SAD-u kada Kanađanin Jacob E. Funk prvi put u enigmatskom časopisu *Dell Magazine* objavljuje kakuro. Kakuro se često uspoređuje s puno poznatijom logičkom igrom sudoku, a zanimljivo je istaknuti da se sudoku u istom časopisu pojavljuje tek 1979. godine. Izvorno ime kakura je *Cross Sums* što u slobodnom prijevodu znači križaljka s brojevima. [1]

Popularnost kakura raste 80-tih godina 20-og stoljeća kada izdavačka kuća *Nikoli* prvi put objavljuje kakuro u Japanu. Nakon izvornog naziva *Cross Sums*, u Japanu dobiva naziv *Kasan Kurosu*, a 1986. godine naziv je skraćen na *Kakuro.*

Popularnost kakura na zapadu raste nakon 2005. godine kada poznate novine *Daily Mail* i *The Guardian* kakuro uvode u svakodnevni tisak. (slika. 1.1.)

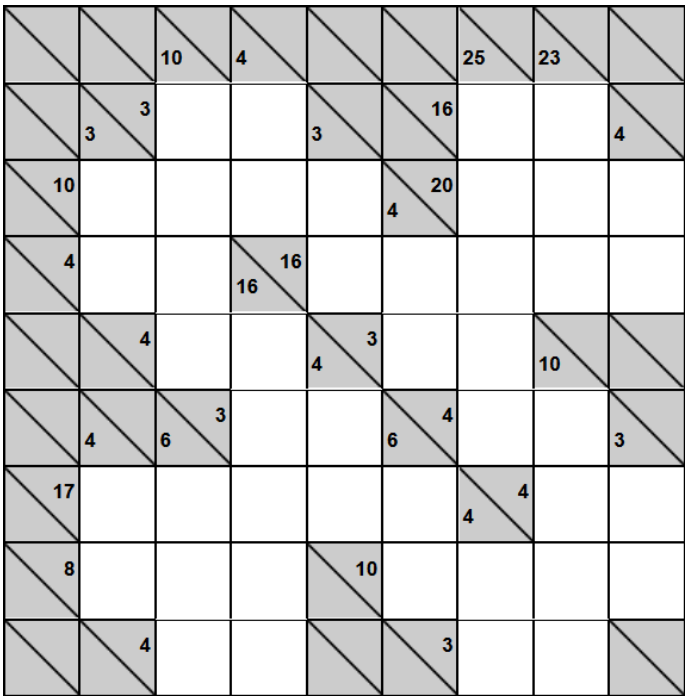


Sl. 1.1. The Guardian kakuro [2]

2. RJEŠAVANJE

Kao i kod većine logičkih igara i kod kakura postoji više taktika i metoda za rješavanje. Osim uobičajenog kakura postoji nekoliko različitih tipova kao što su *Cross Sums Sudoku* i *Cryptic Kakuro* pa oblik, izgled i veličina nisu strogo određeni. Uobičajeno je da se kakuro, kao i obične križaljke, sastoji od tamnijih i svjetlijih polja. Tamnija polja najčešće označavaju okvir kakura i dijagonalom koja povezuje gornji desni i donji lijevi ugao podijeljena su na dva dijela. Svijetlija polja su prazna i predviđeno je da se u njih upisuje rješenje, odnosno vrijednost koja pripada tom polju.

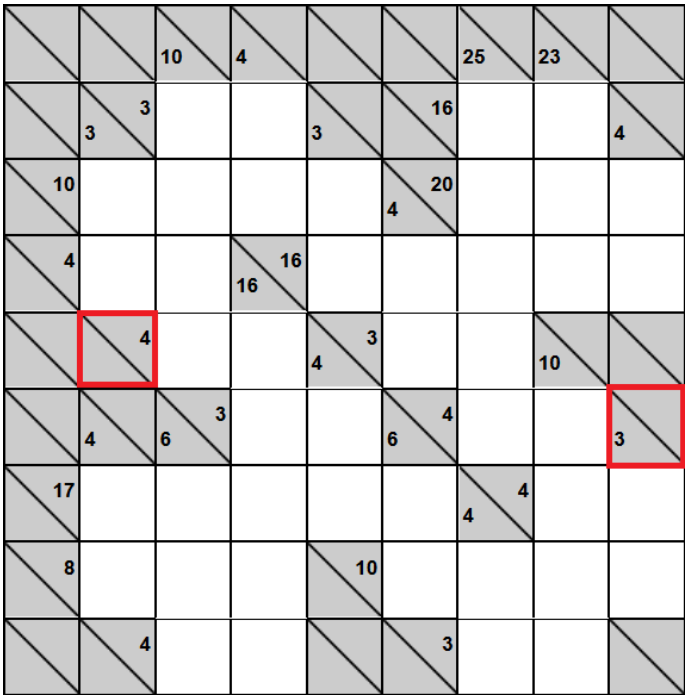
Kakuro je najčešće kvadratnog oblika (N x N), ali mogući su i pravokutni oblici kakura (M x N). Izvorni kakuro je kvadratnog oblika sa 16 stupaca i 16 redaka, ali puno je popularniji kakuro manjih dimenzija kakav se često pojavljuje u novinama. Na slici 2.1. prikazan je kvadratni kakuro s devet stupaca i devet redaka.



Sl. 2.1. Primjer 9x9 kakura [3]

2.1. Pravila

U tamnijim poljima kakura koja označavaju okvir mogu se nalaziti brojevi koji označavaju vodoravne i okomite zbrojeve. Na slici 2.2. crveno su označena dva polja okvira.



Sl. 2.2. Vodoravni i okomiti zbrojevi

U označenom polju na lijevoj strani kakura broj četiri nalazi se iznad dijagonale što znači da predstavlja vodoravni zbroj. Taj vodoravni zbroj odnosi se na dva bijela polja desno od označenog polja. Na desnoj strani kakura označeno je polje s brojem tri ispod dijagonale. Broj tri predstavlja okomiti zbroj i odnosi se na dva bijela polja koja se nalaze ispod označenog polja. Ako je vodoravni zbroj nepopunjen znači da se desno od tog polja ne nalaze bijela polja, odnosno ako je okomiti zbroj nepopunjen ispod tog polja ne nalaze se bijela polja. Zbroj svih pojedinačnih vodoravnih i okomitih zbrojeva kakura mora biti jednak.

Bijela polja smiju se popunjavati samo brojevima od jedan do devet, a pribrojnici pojedinog zbroja moraju biti različiti brojevi, odnosno ne smiju se ponavljati [4]. Budući da u stupcu ili retku gdje postoji zbroj mora biti najmanje dva bijela polja slijedi da najveći zbroj može biti 45, a najmanji 3.

2.2. Metode

U ovom poglavlju ukratko su opisane metode rješavanja kakura. Neke od tih metoda korištene su pri izradi algoritma. Cilj kakura je uspješno riješiti kakuro u što kraćem vremenskom periodu. Kod težih primjeraka kakura važno je na početku rješavanja odrediti mjesta koju su najlakša za riješiti i početi rješavati kakuro upravo od tih mjesta.

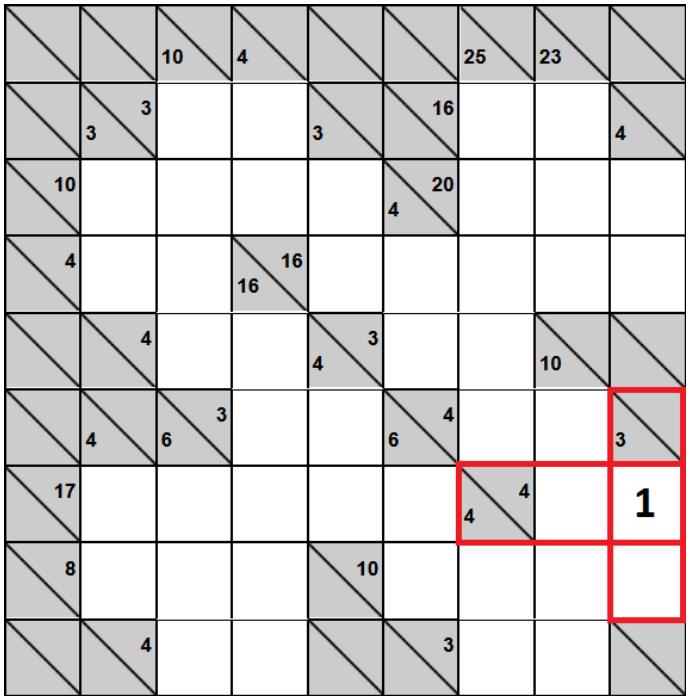
2.2.1. Magični brojevi

Magični brojevi su zbrojevi koji imaju samo jednu moguću kombinaciju u zadanom broju polja [5]. Npr. broj četiri je jedan od takvih brojeva jer je jedino kombinacija 1+3 moguća u dva polja. Mjesta na kojima se nalaze magični brojevi najčešće su najbolja mjesta za početak rješavanja kakura. U tablici 2.1. prikazani su svi magični brojevi, broj polja i moguće kombinacije.

Zbroj	Broj polja	Kombinacija	Zbroj	Broj polja	Kombinacija
3	2	1+2	22	6	1+2+3+4+5+7
4	2	1+3	38	6	3+5+6+7+8+9
16	2	7+9	39	6	4+5+6+7+8+9
17	2	8+9	28	7	1+2+3+4+5+6+7
6	3	1+2+3	29	7	1+2+3+4+5+6+8
7	3	1+2+4	41	7	2+4+5+6+7+8+9
23	3	6+8+9	42	7	3+4+5+6+7+8+9
24	3	7+8+9	36	8	1+2+3+4+5+6+7+8
10	4	1+2+3+4	37	8	1+2+3+4+5+6+7+9
11	4	1+2+3+5	38	8	1+2+3+4+5+6+8+9
29	4	5+7+8+9	39	8	1+2+3+4+5+7+8+9
30	4	6+7+8+9	40	8	1+2+3+4+6+7+8+9
15	5	1+2+3+4+5	41	8	1+2+3+5+6+7+8+9
16	5	1+2+3+4+6	42	8	1+2+4+5+6+7+8+9
34	5	4+6+7+8+9	43	8	1+3+4+5+6+7+8+9
35	5	5+6+7+8+9	44	8	2+3+4+5+6+7+8+9
21	6	1+2+3+4+5+6	45	9	1+2+3+4+5+6+7+8+9

Tab. 2.1. Magični brojevi

Na slici 2.3. prikazana je najjednostavnija primjena magičnih brojeva. Označeni su stupac i redak koji se križaju, a zbrojevi su im magični brojevi. Budući da je jedina moguća kombinacija za zbroj tri 1+2, a za zbroj četiri 1+3, rješenje polja na kojemu se križaju je jedan.

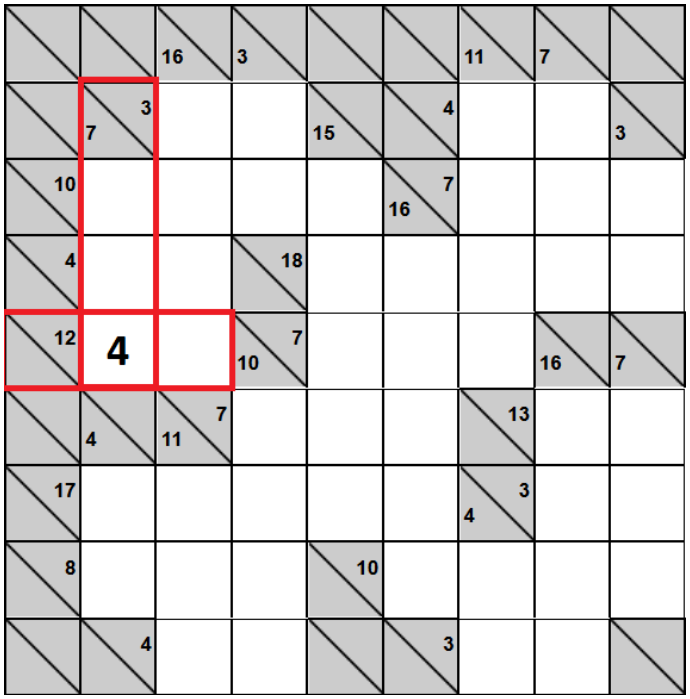


Sl. 2.3. Primjena magičnih brojeva

Budući da se na slici 2.3. nalazi jednostavan primjerak kakura, osim već spomenutog mjesta, postoji puno mjesta s kojih se kakura može početi rješavati. Najčešće su kombinacije magičnih brojeva tri i četiri međutim na gornjoj desnoj strani kakura nalaze se zbrojevi 23 u 3 polja i 16 u 2 polja koji se križaju. To su također magični brojevi sa samo jednom kombinacijom, a jedini broj koji je presjek mogućih kombinacija je broj 9.

2.2.2. Okomiti i vodoravni zbrojevi

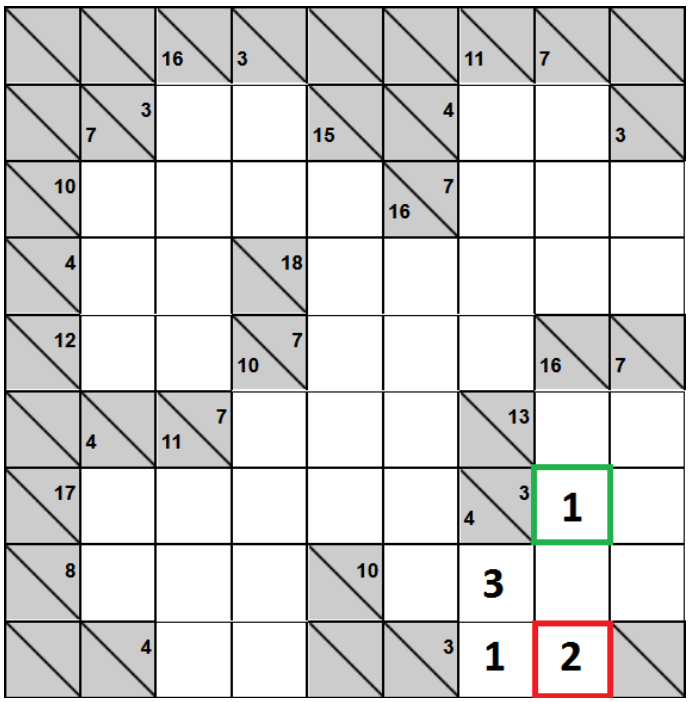
Metoda okomitih i vodoravnih zbrojeva najčešće je korištena metoda za rješavanje.. Primjer na slici 2.4. pokazuje stupac sa zbrojem 7 u 3 polja koji se križa s retkom sa zbrojem 12 u 2 polja. Na polju koje se nalazi na križanju tog stupca i retka, mogući okomiti brojevi su jedan, dva i četiri jer je jedina kombinacija za zbroj sedam u tri polja 1+2+4. Za zbroj 12 u 2 polja ima više mogućih kombinacija pa samim time i više mogućih brojeva. Moguće kombinacije su 3+9, 4+8 i 5+7, a mogući vodoravni brojevi tri, četiri, pet, sedam i osam. Kada se napravi presjek mogućih okomitih i vodoravnih brojeva ako mogući broj ostaje samo broj četiri i upravo je to rješenje tog polja.



Sl. 2.4. Okomiti i vodoravni zbrojevi

2.2.3. Isključivanje

Metoda isključivanja brojeva koristi se nakon što se pomoću drugih metoda riješi nekoliko polja. Na slici 2.5. prikazan započeto je rješavanje kakura. Nakon rješavanje prva tri broja metodom isključivanja riješeno je i četvrto polje koje je na slici označeno zeleno. Budući da je jedina moguća kombinacija za zbroj tri u dva polja 1+2, a u stupcu na crveno označenom mjestu već postoji broj dva metodom isključivanja jedini mogući broj koji preostaje je broj jedan.



Sl. 2.5. Isključivanje

Na slici 2.5. prikazan je najjednostavniji oblik isključivanja. Isključivanje može biti višestruko i može se odnositi na kombinacije. Ukoliko je broj siguran u pojedinom retku ili stupcu metodom isključivanja može se isključiti cijela kombinacija i tako ukloniti više brojeva odjednom.

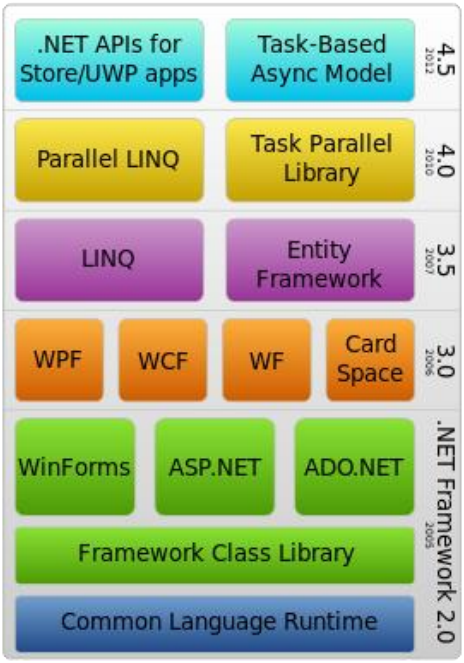
3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju ukratko su objašnjene tehnologije korištene za izradu web aplikacije. Web aplikacija napravljena je kao jednostranična ASP.NET MVC aplikacija.

3.1. .NET Framework

.NET Framework okvir je za programiranje razvijen je od tvrtke Microsoft. Programerima omogućuje lakše razvijanje i održavanje web stranica, aplikacija i servisa. Namijenjen je prvenstveno pokretanju na Microsoft Windows operacijskom sustavu.

Razvoj .NET Framework-a započeo je još 90-tih godina 20-og stoljeća, a prvi put se pojavljuje početkom 2002. godine u verziji 1.0. Tijekom godina Microsoft je razvijao .NET Framework i 2017. godine razvijena je verzija 4.7. U izradi web aplikacije logičke igre kakuro korištena je verzija 4.5.2. Na slici 3.1. prikazan je razvoj sastavni dijelova .NET Framework-a kroz verzije.



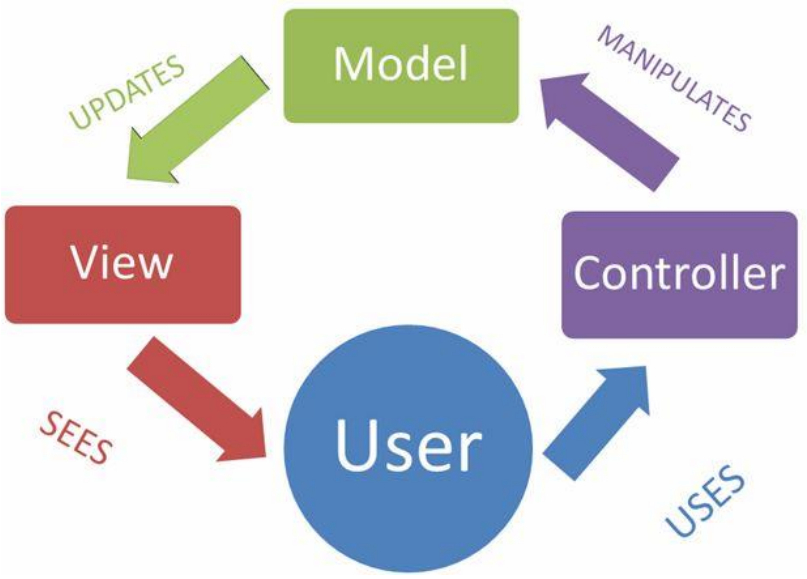
Sl. 3.1. Razvoj .NET Framework-a [6]

3.2. MVC

MVC način je dizajna softverske arhitekture. Koristi se za odvajanje pojedinih dijelova aplikacije u komponente ovisno o njihovoj namjeni. U svojim se začetcima MVC koristio za arhitekturu grafičkog sučelja korisnika kod Desktop aplikacija, ali s vremenom postaje vrlo popularan u web i mobilnim aplikacijama. Gotovo svi popularni programski jezici imaju svoje MVC okvire pa tako postoji i ASP.NET MVC okvir. MVC koji se sastoji od tri dijela:

- Model
    - veza s bazom podataka
    - podaci i poslovna logika
- Pogled (*View*)
    - prikaz podataka
    - korisničko sučelje
- Upravljač (*Controller*)
    - upravlja korisničkim zahtjevima

Na slici 3.2. prikazan je odnos između dijelova MVC-a i korisnika.



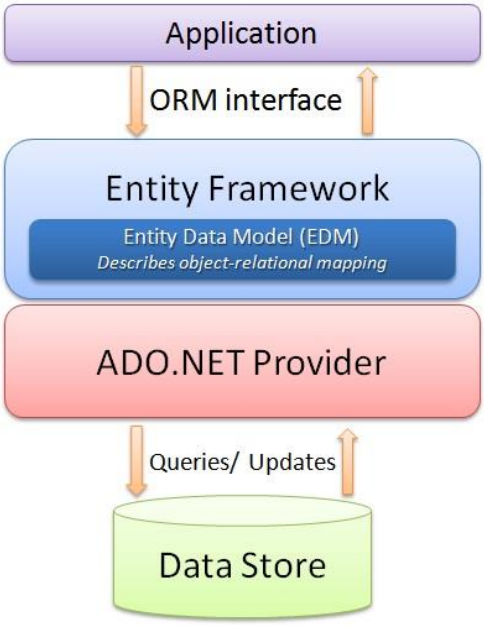
Sl. 3.2. MVC [7]

3.3. Entity Framework

Entity Framework je *open-source* okvir za objektno relacijsko mapiranje (ORM) i koristi ADO.NET. ADO.NET tehnologija je pristupa podacima razvijena od tvrtke Microsoft. Entity Framework napisan je u C# programskom jeziku. Prva verzija Entity Framework-a uključena je u .Net Framework 3.5. SP1 2008. godine. Od verzije 6 Entity Framework više nije dio .Net Framework-a.

Kao i drugi ORM okviri Entity Framework programerima olakšava rad s podacima, smanjuje kod koji je potreban za osnovne operacije s podacima (*CRUD metode*) i omogućuje jednostavnu izradu aplikacija orijentiranim podacima. Prilikom upotrebe Entity Framework-a programeri moraju modelirati entitete, veze između entiteta i logiku potrebnu za rješavanje problema.

Na slici 3.3. prikazana je arhitektura Entity Frameworka.

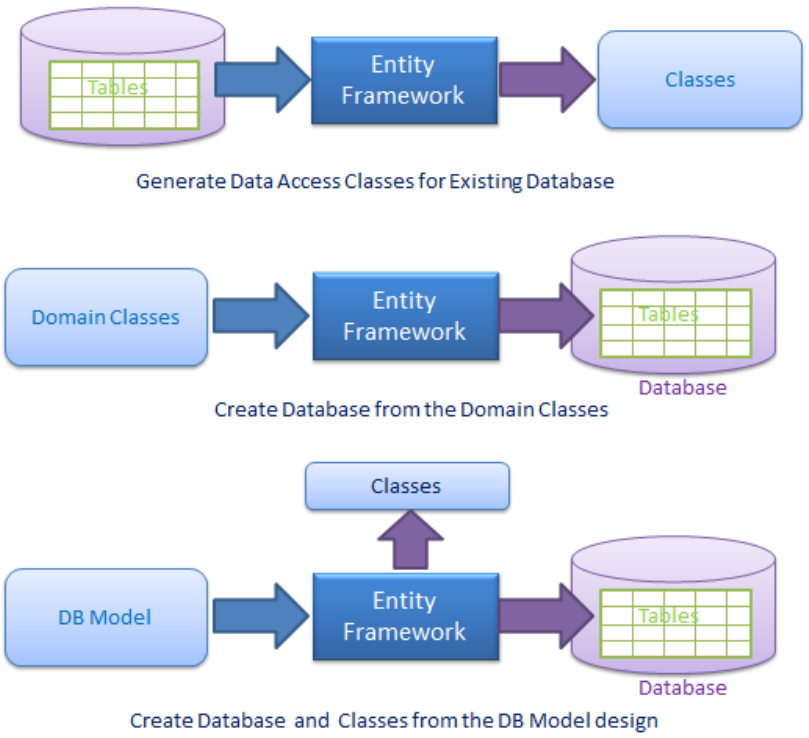


Sl. 3.3. Arhitektura Entity Framework-a [8]

Postoji nekoliko slučajeva odnosno nekoliko mogućih scenarija u kojima se Entity Framework koristi:

- ako već postoji baza podataka ili se prvo želi dizajnirati baza podataka prije drugih dijelova aplikacije. Taj slučaj naziva se *Database first* i koristi se *Code-first from database* ili *EF designer from database* pristup.
- ako se baza podataka želi napraviti iz programskog koda. Koristi se *Code first to a new database* pristup i većina programera koji nisu stručnjaci za baze podataka često koriste upravo ovaj pristup. *Code First* pristup korišten je i u izradi baze podataka koja se koristi za web aplikacije kakuro.
- koji se najmanje koristi u praksi je *Model first* u kojemu se prvo dizajnira shema baze podataka u vizualnom dizajneru i onda se prave klase i baze podataka.

Na slici 3.4. prikazana su 3 navedena slučaja.



Sl. 3.4. Entity Framework pristupi [9]

3.4. Tesseract OCR

Tesseract je *open source engine* za prepoznavanje znakova. Može koristiti u brojnim programskim jezicima. Tesseract nema izrađeno grafičko sučelje, nego se koristi isključivo kao *engine*. Postoje dva dijela instalacije Tesseracta, *engine* i podaci za treniranje. Također, postoji i nekoliko verzija Tesseracta. Za izradu web aplikacije logičke igre kakuro korištena je 3.0.2. verzija.

Problematika optičkog prepoznavanja znakova prisutna je od sredine 20-og stoljeća. Optičko prepoznavanje znakova razvilo se iz potrebne slijepih osoba za korištenjem računala. U današnje vrijeme postoji čitav niz gotovih metoda i načina na koji se OCR može ostvariti.

Postoje 3 dijela optičkog prepoznavanja znakova:

- pretprocesiranje
- prepoznavanje znakova
- postprocesiranje

U pretprocesiranju cilj je sliku što bolje obraditi i pripremiti za prepoznavanje znakova. Obrada uključuje postupke binarizacije, uklanjanja šumova i linija, segmentacije, normalizacije i svih ostalih postupaka koje mogu pomoći da se znakovi lakše pročitaju.

Kod prepoznavanja znakova dvije najčešće korištene metode su prepoznavanje uzoraka i međusobna povezanost slika.

Postprocesiranje odnosi se na postupke koje povećavaju preciznost OCR-a nakon prepoznavanja znakova. Jedan od najčešćih postupaka je postaviti *whitelist* odnosno liste mogućih izlaza kod prepoznavanja znakova. Npr. ukoliko se radi od prepoznavanju riječi kao pravilo može se staviti da mogući izlazi mogu biti samo riječi engleskog jezika ili ako se radi o prepoznavanju brojeva, što je slučaj u ovoj web aplikaciji, može se staviti da se radi o prepoznavanju znakova od 0 do 9.

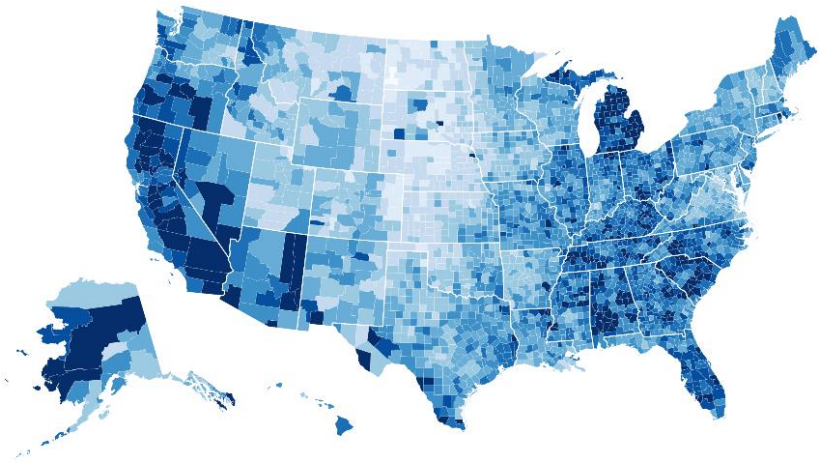
3.5. D3.js

D3.js je JavaScript biblioteka za upravljanje dokumentima na temelju podataka. Pomoću D3.js biblioteke mogu se napraviti dinamičke, interaktivne vizualizacije podataka u internet preglednicima. Za prikaz podataka koristi HTML, CSS i SVG. Verzija 1.0 pojavljuje se početkom 2011. godine, a u izradi web aplikacije korištena verzija 4.7.

Mnogobrojne web stranice koriste D3.js za vizualizaciju podataka. Najpopularnije primjene su različite kontrolne ploče, interaktivni dijagrami na stranicama za vijesti i web karte. Na primjeru na slici 3.5. prikazana je kontrolna ploča izrađena pomoću D3.js biblioteke, a na slici 3.6. interaktivna web karta.



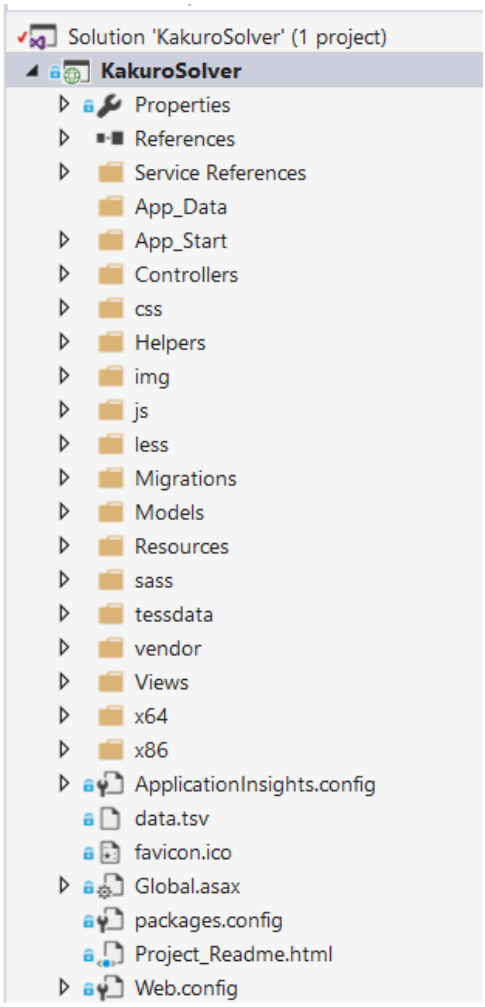
Sl. 3.5. Kontrolna ploča [10]



Sl. 3.6. Interaktivna karta [11]

4. STRUKTURA APLIKACIJE

Aplikacija je dizajnirana prema MVC uzorku. Sastoji se od modela, upravljača i pogleda. Dodatno u aplikaciji postoji optičko prepoznavanje znakova koje se poziva prilikom odabira unosa kakura sa slike. Također, prilikom odabira rješavanja kakura poziva se algoritam za rješavanje. Aplikacija je povezana na bazu podataka u kojoj se prilikom svakog unos kakura sa slike i rješavanja kakura pohranjuju podaci. Na slici 4.1. prikazana je struktura projekta u Visual Studio 2015 razvojnom okruženju u kojemu je aplikacija razvijena.



Sl. 4.1. Struktura projekta

Osim navedenih dijelova, projekt sadrži datoteke potrebne za prikaz podataka, CSS i JavaScript datoteke, slike i resurse koji se koriste za lokalizaciju web aplikacije. Također, sadrži i potrebne konfiguracijske datoteke u kojima se postavljaju parametri potrebni za rad aplikacije.

4.1. Baza podataka

Baza podataka omogućuje kasniju analizu brzine učitavanja i rješavanja kakura. Vrsta baze podataka je Microsoft SQL (MS SQL), a jezik koji se koristi za upite na bazu podataka je Transact SQL (T-SQL). Baza podataka stvorena je iz programskog koda pomoću Entity Framework-a, točnije *Code-First* pristupa u Entity Framework-u. Baza podataka sadrži jednu tablicu naziva *KakuroStatistics.* Tablica sadrži sljedeće atribute:

- **Id**
  - primarni ključ (jedinstvena vrijednost)
  - tip: Guid
- **Rows**
  - broj redaka kakura
  - tip: integer (cjelobrojni)
- **Columns**
  - broj stupaca kakura
  - tip: integer (cjelobrojni)
- **Loaded**
  - je li kakuro učitan
  - tip: bool
- **LoadTime**
  - vrijeme učitavanja u milisekundama
  - tip: integer (cjelobrojni)
- **Solved**
  - je li kakuro riješen do kraja
  - tip: bool
- **SolveTime**
  - vrijeme rješavanja u milisekundama
  - tip: integer

Na slici 4.2. prikazana je tablica *KakuroStatistics* s popunjenim podacima.

Id	SolveTime	Rows	Columns	LoadTime	Loaded	Solved
2659a578-a...	0	5	5	14732	True	False
c22aa3f8-1...	45	3	3	5558	True	True
b2257364-b...	17	3	3	2872	True	True
3a7b71d7-1...	0	9	8	25205	True	False
c2ddd0bb-f...	11031	15	15	83864	True	True
0aac3a6c-1...	41	5	5	12963	True	True
f3b95713-0...	0	4	4	5648	False	False
0ac7756b-0...	6	4	4	3937	True	True
b80c2dd7-9...	4637	13	13	66623	True	True
043ff794-dc...	1738	9	8	0	False	False
be475df7-8...	0	3	3	4516	False	False
03ce0f86-4...	3085	12	10	0	False	True
bd0d5baf-1...	3010	12	10	31184	False	True
f96c8eb7-9...	35	3	3	0	False	True
f494f15a-e9...	0	4	4	7563	True	False
8ffae688-ca...	0	3	3	5812	True	False
37252274-c...	108	7	7	0	False	True
d317e361-c...	0	9	9	16095	False	False
83b0ee2d-d...	0	3	3	5104	True	False
28ef14ba-fe...	51	5	5	13638	True	True
741361a1-2...	1645	9	8	28888	True	False
9ba0778c-5...	1762	9	8	32585	True	False
b73b9c6c-8...	70	5	5	14115	True	True
28a1855b-f...	1657	9	8	28594	True	False

Sl. 4.2. *KakuroStatistics* tablica s podacima

4.2. Modeli

Razlikuju se modeli baze podataka i modeli za prikaz. Iako jednako izgledaju, modeli za prikaz ne dodaju se u kontekst u kojemu se kreira baza podataka.

4.2.1. Modeli baze podataka

Da bi se baza podataka mogla stvoriti iz programskog koda potrebno je prije svega napraviti modele koji predstavljaju tablice u bazi podataka. Budući da je predviđeno da u bazi podataka postoji samo jedna tablica potrebno je napraviti jedan takav model. Budući da se tablica u bazi podataka zove *KakuroStatistics* model je nazvan u jednini *KakuroStatistic*. Na slici 4.3. prikazani je programski kod modela. U programskom kodu definirani su atributi koje tablica sadrži.

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace KakuroSolver.Models.DBModels
{
    7 references | Ivan Juric, 11 days ago | 1 author, 1 change
    public class KakuroStatistic : BaseEntity
    {
        [Required]
        4 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public int Rows { get; set; }
        [Required]
        4 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public int Columns { get; set; }
        [NotMapped]
        0 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public int NumberOfFields
        {
            get
            {
                return Rows * Columns;
            }
            set { }
        }
        1 reference | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public long LoadTime { get; set; }
        4 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public long SolveTime { get; set; }
        1 reference | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public bool Loaded { get; set; }
        4 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public bool Solved { get; set; }
        3 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
        public KakuroStatistic()
        {
            Id = Guid.NewGuid();
        }
    }
}
```

Sl. 4.3. *KakuroStatistic* model

Osim atributa iz tablice u bazi podataka definiran je i atribut koji se koriste samo u programskom kodu. To je atribut *NumberOfFields* koji označava broj polja kakura i izračunava se množenjem broja redaka i stupaca, a koristi za se analizu. Budući da se atribut *NumberOfFields* neće nalaziti u bazi podataka nužno je dodati mu vlastiti atribut *NotMapped*.

Atributi *Rows* i *Columns* sadrže vlastiti atribut *Required* koji označava da je prilikom unosa u tablicu *KakuroStatistics* neophodno postaviti vrijednost tih atributa.

4.2.2. Modeli za prikaz

Druga vrsta modela su *View* modeli odnosno modeli za prikaz. Koriste se za prikaz podataka na korisničkom sučelju. Modeli za prikaz popunjavaju se ovisno o odabiru akcije na korisničkom sučelju. U aplikaciji postoje tri modela za prikaz:

- *Read model*
    - služi za učitavanje kakura
    - sadrži datoteku, broj redaka, broj stupaca

```
8   namespace KakuroSolver.Models
9   {
        4 references | Ivan Juric, 12 days ago | 1 author, 3 changes
10      public class KakuroReadModel
11      {
12          [Display(Name = "File", ResourceType = typeof(Localization))]
            5 references | Ivan Juric, 12 days ago | 1 author, 3 changes | 0 exceptions, ? live
13          public HttpPostedFileBase File { get; set; }
14
15          [Required(ErrorMessageResourceName = "FieldRequired", ErrorMessageResourceType = typeof(Localization))]
16          [Display(Name = "NumberOfRows", ResourceType = typeof(Localization))]
17          [Range(3, 15, ErrorMessageResourceName = "ValueOutOfRange_3_15", ErrorMessageResourceType = typeof(Localization))]
            12 references | Ivan Juric, 97 days ago | 1 author, 2 changes | 0 exceptions, ? live
18          public int NumberOfRows { get; set; }
19
20          [Required(ErrorMessageResourceName = "FieldRequired", ErrorMessageResourceType = typeof(Localization))]
21          [Display(Name = "NumberOfColumns", ResourceType = typeof(Localization))]
22          [Range(3, 15, ErrorMessageResourceName = "ValueOutOfRange_3_15", ErrorMessageResourceType = typeof(Localization))]
            20 references | Ivan Juric, 97 days ago | 1 author, 2 changes | 0 exceptions, ? live
23          public int NumberOfColumns { get; set; }
24      }
25  }
```

Sl. 4.4. *Read model*

- *Helper model*
    - sadrži broj polja, zbroj i kombinacije

```
8   namespace KakuroSolver.Models
9   {
        1 reference | Ivan Juric, 97 days ago | 1 author, 2 changes
10      public class KakuroHelperModel
11      {
12          [Required(ErrorMessageResourceName = "FieldRequired", ErrorMessageResourceType = typeof(Localization))]
13          [Display(Name = "NumberOfFields", ResourceType = typeof(Localization))]
14          [Range(2, 9, ErrorMessageResourceName = "ValueOutOfRange_2_9", ErrorMessageResourceType = typeof(Localization))]
            1 reference | Ivan Juric, 97 days ago | 1 author, 2 changes | 0 exceptions, ? live
15          public int NumberOfFields { get; set; }
16
17          [Required(ErrorMessageResourceName = "FieldRequired", ErrorMessageResourceType = typeof(Localization))]
18          [Display(Name = "Sum", ResourceType = typeof(Localization))]
19          [Range(3, 45, ErrorMessageResourceName = "ValueOutOfRange_3_45", ErrorMessageResourceType = typeof(Localization))]
            1 reference | Ivan Juric, 97 days ago | 1 author, 2 changes | 0 exceptions, ? live
20          public int Sum { get; set; }
21
22          [Display(Name = "Combinations", ResourceType = typeof(Localization))]
            1 reference | Ivan Juric, 103 days ago | 1 author, 1 change | 0 exceptions, ? live
23          public List<List<int>> Combinations { get; set; }
24      }
25  }
```

Sl. 4.5. *Helper model*

- *Statistics model*
    - analiza
    - sadrži listu *KakuroStatistic* modela, ukupan broj i broj riješenih kakura

```
7   namespace KakuroSolver.Models
8   {
        14 references | Ivan Juric, 11 days ago | 1 author, 1 change
9       public class StatisticsModel
10      {
            14 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
11          public List<KakuroStatistic> KakuroStatistics { get; set; }
12
            0 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
13          public int KakuroNumber
14          {
15              get
16              {
17                  return KakuroStatistics.Count();
18              }
19              set { }
20          }
            0 references | Ivan Juric, 11 days ago | 1 author, 1 change | 0 exceptions, ? live
21          public int SolvedKakuros
22          {
23              get
24              {
25                  var returnValue = 0;
26                  foreach (var item in KakuroStatistics)
27                  {
28                      if (item.Solved)
29                      {
30                          returnValue++;
31                      }
32                  }
33                  return returnValue;
34              }
35              set { }
36          }
37      }
38  }
```

Sl. 4.6. *Statistics model*

4.3. Upravljač

Aplikacija sadrži samo jedan upravljač koji se zove *HomeController*. Upravljač sadrži sedam metoda koje kao izlaznu vrijednost vraćaju različite poglede. Budući da se radi o jednostraničnoj aplikaciji, sve metode vraćaju pogled na istu stranicu, ali na različiti dio stranice. Osim pogleda, metode vraćaju i pripadajući model.

Od sedam metoda koje upravljač sadrži četiri su *HttpGet* metode, a tri *HttpPost* metode. *HttpGet* metode samo potražuju podatke, dok *HttpPost* metode prvo predaju određene podatke pa zatim kao rezultat dobivaju pogled s modelom.

Četiri *HttpGet* metode su:

- *Index()*
  - vraća početni pogled i *Statistics* model
- *Solver()*
  - vraća pogled na Rješavača i *Statistics* model
- *Helper()*
  - vraća pogled na Pomagača i *Statistics* model
- *ChangeLanguage()*
  - vraća trenutni pogled, ali mijenja jezik u cijeloj aplikaciji

Metoda *ChangeLanguage()* zanimljiva je jer mijenja jezik u cijeloj aplikaciji na način da promijeni kulturu. Kao ulazni parametre prima jezik na koji se aplikacija treba promijeniti i trenutnu adresu aplikacije. Na slici 4.7. prikazan je programski kod *ChangeLanguage()* metode.

```
public ActionResult ChangeLanguage(string language, string returnUrl)
{
    Session["Culture"] = new CultureInfo(language);
    return Redirect(returnUrl);
}
```

Sl. 4.7. *ChangeLanguage()* metoda

HttpPost metode čine glavni dio upravljača. Kao i *HttpGet* metode vraćaju pogled i pripadajući model. Postoje tri *HttpPost* metode:

- *LoadKakuro()*
- *SolveKakuro()*
- *Helper()*

LoadKakuro() metoda služi za učitavanje kakura. Kao parametar prima *KakuroRead* model u kojemu se nalaze broj redaka, broj stupaca i slika kakura koji se želi učitati. Metoda zatim poziva optičko prepoznavanje znakova koje uz pomoć *Tesseract OCR Engine* pokušava prepoznati okvir kakura, vodoravne i okomite zbrojeve. Nakon učitavanja kakura u bazu podataka spremaju se dimenzije kakura, podatak je li kakuro uspješno učitan i vrijeme učitavanja. Ukoliko korisnik ne preda sliku kakura metoda će vratiti samo osnovni okvir bez pripadajućih zbrojeva i u bazu podataka neće ništa spremati.

SolveKakuro() metoda služi za rješavanje kakura i predstavlja glavnu metodu u upravljaču. Kao parametre prima broj redaka, broj stupaca i pročitane ili unesene vrijednosti za vodoravne i okomite zbrojeve. Na temelju tih podataka poziva algoritam i prosljeđuje mu navedene parametre. Nakon što dobije rješenje od algoritma metoda provjerava je li rješenje potpuno odnosno jesu li sva prazna polja popunjena. Metoda zatim u bazu podataka sprema dimenzije kakura, podatak je li kakuro potpuno riješen i vrijeme rješavanja. Na kraju, metoda vraća pogled i rješenje kakura. Na slici 4.8. prikazan je dio *SolveKakuro()* metode koji inicijalizira novi algoritam i mjeri vrijeme potrebno za izvršavanje algoritma.

```
var algorithm = new Algorithm();
var watch = System.Diagnostics.Stopwatch.StartNew();
var resultsCells = algorithm.GetResult(cells, rows, columns);
watch.Stop();
var elapsedMs = watch.ElapsedMilliseconds;
```

Sl. 4.8. Dio metode *SolveKakuro()*

Posljednja *HttpPost* metoda je *Helper()*. Služi za prikazivanje mogućih kombinacija za zadani broj polja i zbroj. Prema tome kao parametre prima broj polja kakura i pripadajući zbroj. Metoda zatim poziva funkciju za traženje svih kombinacija bez ponavljanja brojeva od jedan do devet koje su dugačke koliko je zadan broj polja, a zbroj brojeva u kombinaciji je predani zbroj. Funkcija vraća odgovarajuće kombinacije bez ponavljanja, a metoda ih zatim prosljeđuje na pogled. Na slici 4.9. prikazana je metoda *Helper()*.

```
public ActionResult Helper(KakuroModel model)
{
    @ViewBag.Position = "helper";
    using (var db = new ApplicationDbContext())
    {
        model.StatisticsModel = new StatisticsModel()
        {
            KakuroStatistics = db.KakuroStatistics.ToList()
        };
    };
    if (!ModelState.IsValid)
    {
        return View("Index");
    }
    model.KakuroHelper.Combinations = Algorithm.GetAllCombinations(new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9 },
    return View("Index", model);
}
```

Sl. 4.9. Metoda *Helper()*

4.4. Pogledi

Aplikacija sadrži osam pogleda. Glavni pogled naziva se *_Layout* pogled i sadrži *Navigation* i *Index* pogled. *Index* pogled sadržava sve ostale poglede koji postoje u aplikaciji. Svi pogledi osim prva tri navedena su parcijalni pogledi i zasebno se učitavaju. Parcijalni pogledi redom su:

- *Intro*
- *Solver*
- *Helper*
- *Analysis*
- *Footer*

Na slici 4.10. prikazan je programski kod *Intro* pogleda, a na slici 4.11. izgled na web aplikaciji. U *Intro* pogledu nalaze se ime i opis aplikacije.

```
<header class="intro">
    <div class="intro-body">
        <div class="container">
            <div class="row">
                <div class="col-md-8 col-md-offset-2">
                    <h1 class="brand-heading">@Resources.Localization.ApplicationName</h1>
                    <p class="intro-text">
                        @Resources.Localization.Description
                    </p>
                    <a href="#solver" class="btn btn-circle page-scroll">
                        <i class="fa fa-angle-double-down animated"></i>
                    </a>
                </div>
            </div>
        </div>
    </div>
</header>
```

Sl.4.10. Programski kod *Intro*



Sl. 4.11. *Intro* izgled na web aplikaciji

Solver pogled je složeniji i sastoji se od forme za unos kakura i mreže kakura kada se kakuro učita sa slike ili ručno unese. Na slici 4.12. prikazan je dio HTML forme za unos kakura.

```
@using (Html.BeginForm("LoadKakuro", "Home", FormMethod.Post, new { enctype = "multipart/form-data" }))
{
    <div class="form-horizontal">
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.KakuroRead.NumberOfRows, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10" style="text-align:left">
                @Html.EditorFor(model => model.KakuroRead.NumberOfRows, new { htmlAttributes = new { @class = "form-control form-numbers",
                @Html.ValidationMessageFor(model => model.KakuroRead.NumberOfRows, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.KakuroRead.NumberOfColumns, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10" style="text-align:left">
                @Html.EditorFor(model => model.KakuroRead.NumberOfColumns, new { htmlAttributes = new { @class = "form-control form-numbers
                @Html.ValidationMessageFor(model => model.KakuroRead.NumberOfColumns, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10" id="buttonDiv">
                <input type="submit" value="@Resources.Localization.CreateButton" class="btn btn-lg btn-default btn-success" style="float:l
                <a onclick="return toggleForm()" class="btn btn-lg btn-default btn-success" style="float:left;margin-left:20px">@Resources.
            </div>
        </div>
    </div>
    <div class="form-group" id="loadFormDiv" style="display:none">
```

Sl. 4.12. Forma za unos kakura

Helper pogled sastoji se od forme za unos broja polja i zbroja te liste u kojoj se ispisuju moguće kombinacije. Na slici 4.13. prikazan je programski kod koji služi za ispis mogućih kombinacija u *Helper* pogledu.

```
<div style="text-align:left">
    @Resources.Localization.Combinations
    <ul style="list-style-type:none">
        @foreach (var combination in Model.KakuroHelper.Combinations)
        {
            <li>
                @String.Join(", ", combination.ToArray())
            </li>
        }
    </ul>
</div>
```

Sl. 4.13. Ispis mogućih kombinacija

Analysis pogled sadrži dva dijagrama. Prvi opisuje ovisnost vremena potrebnog za učitavanje o broju polja kakura, a drugi opisuje ovisnost vremena potrebnog za rješavanje o broju polja kakura. Za prikaz dijagrama korištena je D3.js biblioteka. D3.js funkciji predaju se modeli koji sadrže statističke podatke o učitavanja i rješavanju kakura, a funkcija zatim na dijagramima crta krugove. Ako je kakuro uspješno učitan ili riješen crta se zeleni krug, a ako nije crta se crveni krug. Na slici 4.14. nalazi se dio JavaScript koda za dodavanje osi na dijagram.

```
svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis)
  .append("text")
    .attr("class", "label")
    .attr("x", width)
    .attr("y", -6)
    .style("text-anchor", "end")
    .text('@Resources.Localization.Graph_NumberOfFields');

svg.append("g")
    .attr("class", "y axis")
    .call(yAxis)
  .append("text")
    .attr("class", "label")
    .attr("transform", "rotate(-90)")
    .attr("y", 6)
    .attr("dy", ".71em")
    .style("text-anchor", "end")
    .text('@Resources.Localization.Graph_Time_s');
```

Sl. 4.14. Dodavanje osi na dijagram

Na samom dnu stranice nalazi se *Footer* pogled. Sadržava samo ime autora i godinu izrade aplikacije. *Footer* pogled prikazan je na slici 4.15.



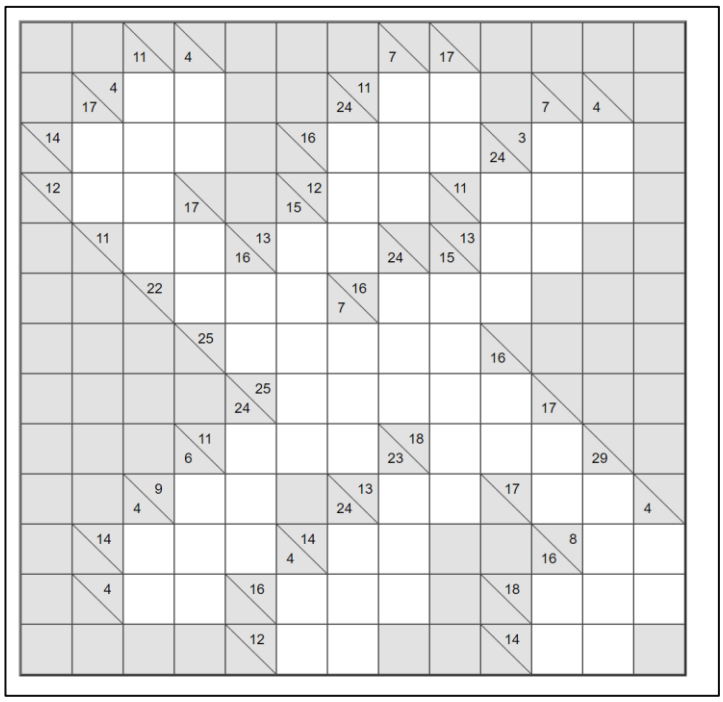
Sl. 4.15. *Footer*

4.5. Optičko prepoznavanje znakova

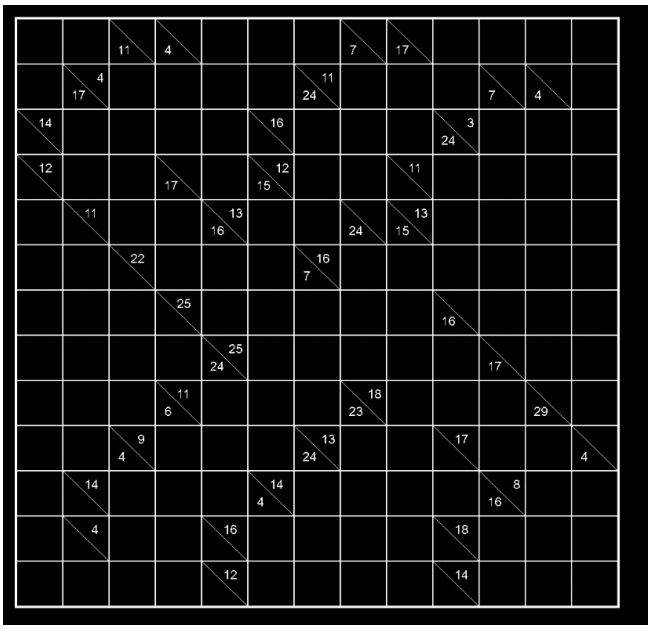
Optičko prepoznavanje znakova (OCR) poziva se kada korisnik želi unijeti kakuro sa slike. Metoda *LoadKakuro()* s upravljača poziva OCR i kao parametre šalje broj stupaca i redaka te sliku kakura.

4.5.1. Pretprocesiranje

U procesu pretprocesiranja događa se čitav niz postupaka pripremanja slike za prepoznavanje znakova. Originalna slika prikazana na slici 4.16. klonira se i na kloniranoj slici napravi se binarizacija i komplementiranje što je prikazano na slici 4.17.

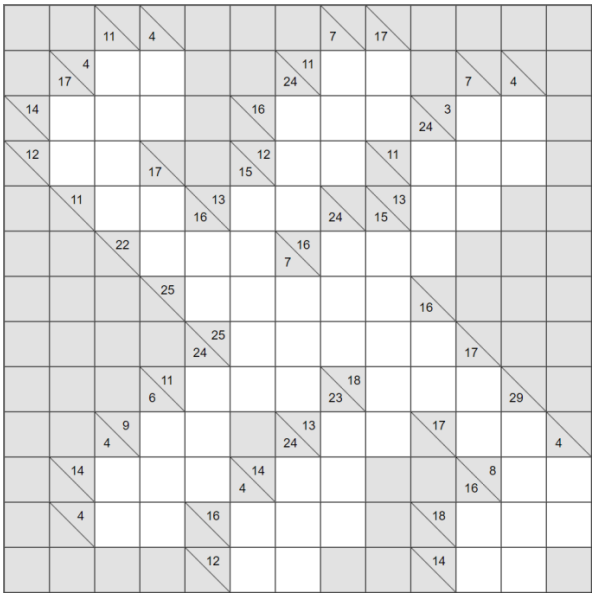


Sl. 4.16. Originalna slika

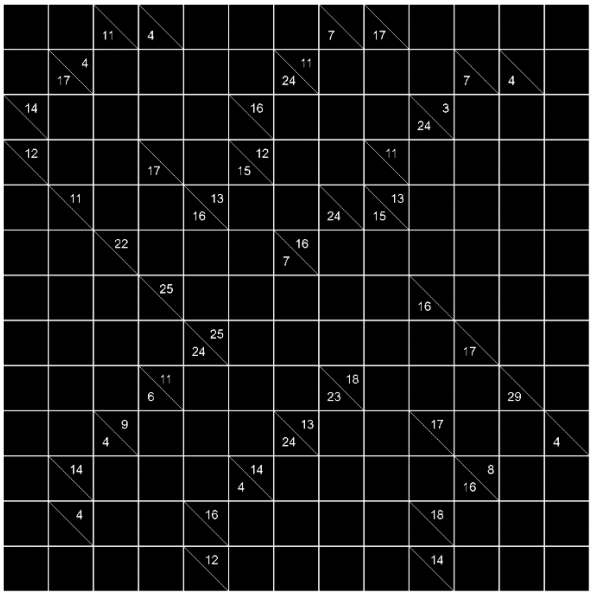


Sl. 4.17. Binarizirano-komplementirana slika

Na binarizirano-komplementiranoj slici pronalaze se koordinate koje označavaju granice kakura. Npr. za lijevu granicu krene se od lijevog ruba na pola visine slike što znači da početne koordinate imaju vrijednost (0, visina/2) te se provjerava vrijednost piksela i pomiče udesno sve dok se ne dođe do bijelog piksela. Kada se dođe do bijelog piksela x vrijednost spremi se kao vrijednost lijeve granica kakura. Sukladno tome određuju se i ostale tri granice kakura. Kada su poznate sve četiri granice kakura iz originalne slike i binarizirano-komplementirane izreže se kakuro. (slika 4.18. i slika 4.19.)

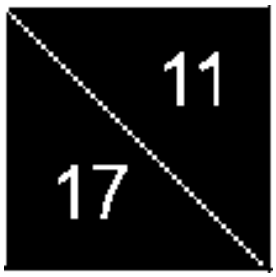


Sl. 4.18. Izrezana originalna slika



Sl. 4.19. Izrezana binarizirano-komplementirana slika

Budući da su poznate dimenzije kakura, iz izrezane originalne slike za svako polje provjerava se je li određeni piksel u polju bijele boje. Ukoliko piksel je bijele boje radi se o polju za unos rješenja, a ukoliko nije radi se o polju okvira kakura. Ovim postupkom saznaje se okvir kakura. Kada je okvir kakura poznat, izrezana binarizirano-komplementirana slika dijeli se na polja. Konkretno gore prikazan kakuro 13x13 dijeli se na 169 polja. Primjer jednog polja okvira kakura prikazan je na slici 4.20.

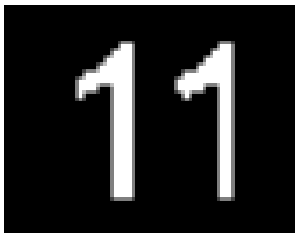


Sl. 4.20. Polje okvira kakura

Nadalje iz svih takvih slika polja okvira izrezuju se dva pravokutnika unutar kojih se nalaze brojevi. (slika 4.21. i slika 4.22.)



Sl. 4.21. Donji lijevi pravokutnik



Sl. 4.22. Gornji desni pravokutnik

Ovako pripremljene slike spremne su za optičko prepoznavanje znakova.

4.5.2. Prepoznavanje znakova

Izrezani pravokutnici pojedinačno se predaju *Tesseract* metodi za prepoznavanje znakova. Metoda za prepoznavanje znakova može se prilagoditi pomoću nekoliko parametara. U web aplikaciji korištena su 2 parametra: Segmentacija stranice i lista dopuštenih znakova. Zadana segmentacija stranice kod *Tesseract*-a je čitanje cijele stranice, a s obzirom da se u slučaju kakura radi o samo jednom ili dva znaka u istom redu segmentacija je postavljena na *SingleWord*. Također skup znakova dosta je ograničen. Radi se samo o znamenkama od nula do devet pa je *whitelist* postavljen na „0123456789“. *Whitelist* predstavlja listu dopuštenih znakova. Tim parametrima poboljšana je točnost prepoznavanja znakova. Na slici 4.23. prikazan je programski kod metode koja poziva *Tesseract Engine*,

```
public static string ReadTextFromImage(Bitmap image)
{
    var dataPath = AppDomain.CurrentDomain.BaseDirectory + @"tessdata";

    //Creating the tesseract OCR engine with English as the language
    using (var tEngine = new TesseractEngine(dataPath, "eng", EngineMode.Default))
    {
        //Process the specified image
        using (var page = tEngine.Process(image, Tesseract.PageSegMode.SingleWord))
        {
            tEngine.SetVariable("tessedit_char_whitelist", "0123456789");

            //Gets the image's content as plain text
            var text = page.GetText();

            try
            {
                var value = Convert.ToInt32(text);
                if (value < 3 || value > 45)
                {
                    text = "?";
                }
                else
                {
                    text = value.ToString();
                }
            }
            catch (Exception)
            {
                text = "";
            }
            return text;
        }
    }
}
```

Sl. 4.23. Metoda za prepoznavanje znakova

4.6. Algoritam

Algoritam za rješavanje kakura poziva se iz upravljača u metodi *SolveKakuro()*. Algoritmu se predaju broj stupaca i redaka te pripadajući vodoravni i okomiti zbrojevi.

4.6.1. Opis

Za razliku od drugih algoritama rješavanja kakura, ideja ovog algoritma je inicijalno pronaći sve moguće kombinacije i sve moguće brojeve, a zatim kroz iteracije, pomoću nekoliko metoda, smanjivati broj mogućih brojeva sve dok se ne dođe do vrijednosti polja. Tada se u tom retku i stupcu smanjuju virtualna duljina i zbroj. Na slici 4.24. prikazane su sve moguće kombinacije za različite duljine.

2	3	12	12345	15	**5**
	4	13	12346	16	
	5	14 23	12347 12356	17	
	6	15 24	12348 12357 12456	18	
	7	16 25 34	12349 12358 12367 12457 13456	19	
	8	17 26 35	12359 12368 12458 12467 13457 23456	20	
	9	18 27 36 45	12369 12378 12459 12468 12567 13458 13467 23457	21	
	10	19 28 37 46	12379 12469 12478 12568 13459 13468 13567 23458 23467	22	
	11	29 38 47 56	12389 12479 12569 12578 13469 13478 13568 14567 23459 23468 23567	23	
	12	39 48 57	12489 12579 12678 13479 13569 13578 14568 23469 23478 23568 24567	24	
	13	49 58 67	12589 12679 13489 13579 13678 14569 14578 23479 23569 23578 24568 34567	25	
	14	59 68	12689 13589 13679 14579 14678 23489 23579 23678 24569 24578 34568	26	
	15	69 78	12789 13689 14589 14679 15678 23589 23679 24579 24678 34569 34578	27	
	16	79	13789 14689 15679 23689 24589 24679 25678 34579 34678	28	
	17	89	14789 15689 23789 24689 25679 34589 34679 35678	29	
3	6	123	15789 24789 25689 34689 35679 45678	30	
	7	124	16789 25789 34789 35689 45679	31	
	8	125 134	26789 35789 45689	32	
	9	126 135 234	36789 45789	33	
	10	127 136 145 235	46789	34	
	11	128 137 146 236 245	56789	35	
	12	129 138 147 156 237 246 345	123456	21	**6**
	13	139 148 157 238 247 256 346	123457	22	
	14	149 158 167 239 248 257 347 356	123458 123467	23	
	15	159 168 249 258 267 348 357 456	123459 123468 123567	24	
	16	169 178 259 268 349 358 367 457	123469 123478 123568 124567	25	
	17	179 269 278 359 368 458 467	123479 123569 123578 124568 134567	26	
	18	189 279 369 378 459 468 567	123489 123579 123678 124569 124578 134568 234567	27	
	19	289 379 469 478 568	123589 123679 124579 124678 134569 134578 234568	28	
	20	389 479 569 578	123689 124589 124679 125678 134579 134678 234569 234578	29	
	21	489 579 678	123789 124689 125679 134589 134679 135678 234579 234678	30	
	22	589 679	124789 125689 134689 135679 145678 234589 234679 235678	31	
	23	689	125789 134789 135689 145679 234689 235679 245678	32	
	24	789	126789 135789 145689 234789 235689 245679 345678	33	
4	10	1234	136789 145789 235789 245689 345679	34	
	11	1235	146789 236789 245789 345689	35	
	12	1236 1245	156789 246789 345789	36	
	13	1237 1246 1345	256789 346789	37	
	14	1238 1247 1256 1346 2345	356789	38	
	15	1239 1248 1257 1347 1356 2346	456789	39	
	16	1249 1258 1267 1348 1357 1456 2347 2356	1234567	28	**7**
	17	1259 1268 1349 1358 1367 1457 2348 2357 2456	1234568	29	
	18	1269 1278 1359 1368 1458 1467 2349 2358 2367 2457 3456	1234569 1234578	30	
	19	1279 1369 1378 1459 1468 1567 2359 2368 2458 2467 3457	1234579 1234678	31	
	20	1289 1379 1469 1478 1568 2369 2378 2459 2468 2567 3458 3467	1234589 1234679 1235678	32	
	21	1389 1479 1569 1578 2379 2469 2478 2568 3459 3468 3567	1234689 1235679 1245678	33	
	22	1489 1579 1678 2389 2479 2569 2578 3469 3478 3568 4567	1234789 1235689 1245679 1345678	34	
	23	1589 1679 2489 2579 2678 3479 3569 3578 4568	1235789 1245689 1345679 2345678	35	
	24	1689 2589 2679 3489 3579 3678 4569 4578	1236789 1245789 1345689 2345679	36	
	25	1789 2689 3589 3679 4579 4678	1246789 1345789 2345689	37	
	26	2789 3689 4589 4679 5678	1256789 1346789 2345789	38	
	27	3789 4689 5679	1356789 2346789	39	
	28	4789 5689	1456789 2356789	40	
	29	5789	2456789	41	
	30	6789	3456789	42	
			12345678	36	**8**
			12345679	37	
			12345689	38	
			12345789	39	
			12346789	40	
			12356789	41	
			12456789	42	
			13456789	43	
			23456789	44	
			123456789	45	**9**

Sl. 4.24. Moguće kombinacije

4.6.2. Početak

Prvi korak u algoritmu pronalazak je svih mogućih vodoravnih i okomitih kombinacija te spremanje brojeva iz tih kombinacija u moguće okomite i vodoravne brojeve pojedinih polja. Nakon toga presjekom ta dva skupa dobiva se skup mogućih brojeva pojedinih polja. Na slici 4.25. prikazana je funkcija za pronalazak svih mogućih kombinacija.

```
514         private static List<List<int>> GetAllCombinations(List<int> allPossibleNumbers, int arrayLength, int arraySum)
515         {
516             var allPossibleSolutions = new List<List<int>>();
517             var possibleSolutions = new List<List<int>>();
518             int setCounter = Convert.ToInt32(Math.Pow(2, allPossibleNumbers.Count()));
519             for (int number = 1; number < setCounter; number++)
520             {
521                 var nestedList = new List<int>();
522                 for (int j = 0; j < allPossibleNumbers.Count(); j++)
523                 {
524                     var binaryNumber = 1 << j;
525                     if ((number & binaryNumber) == binaryNumber) { nestedList.Add(allPossibleNumbers[j]); }
526                 }
527                 allPossibleSolutions.Add(nestedList);
528             }
529             foreach (List<int> possibleNumbers in allPossibleSolutions)
530             {
531                 if (arrayLength == possibleNumbers.Count())
532                 {
533                     int currentSum = 0;
534                     foreach (int number in possibleNumbers)
535                     {
536                         currentSum += number;
537                     }
538                     if (currentSum == arraySum)
539                     {
540                         possibleSolutions.Add(possibleNumbers);
541                     }
542                 }
543             }
544             return possibleSolutions;
545         }
```

Sl. 4.25. Funkcija za pronalazak svih kombinacija

4.6.3. Iteracije

Nakon što se početno izračunaju sve moguće kombinacije u stupcima i redcima, algoritam ulazi u petlju od maksimalno 15 iteracija. U iteracijama primjenjuju se standardne metode rješavanja kakura.

Prvi korak u iteracijama ispitivanje je nalazi li se samo jedan broj u skupu mogućih brojeva. Ukoliko je moguć samo jedan broj u polju upravo taj broj postavlja se na vrijednost polja, a iz ostalih skupova mogućih brojeva u retku i stupcu gdje se polje nalazi uklanja se taj broj. Također u tom retku i stupcu smanjuje se virtualni zbroj za vrijednost polja te virtualna duljina za jedan.

U stupcu ili retku, ovisno o broju polja, mogući su najmanji ili najveći zbrojevi. Drugi korak algoritma provjera je odgovara li virtualni zbroj nekom od najmanjih i najvećih zbrojeva za određenu duljinu odnosno je li virtualni zbroj magični broj.

Treći korak ponovno je računanje svih mogućih vodoravnih i okomitih kombinacija, međutim ovaj put s virtualnim duljinama i zbrojevima. Ukoliko je nakon ponovnog računanja moguća samo jedna kombinacija u retku ili stupcu na moguće brojeve svih polja tog retka ili stupca koji još nemaju svoju vrijednost stavljaju se brojevi iz te kombinacije.

Četvrti korak pokušaj je stavljanja svakog broja iz mogućih brojeva polja na vrijednost tog polja te računanje kombinacija sa smanjenim zbrojem u duljini za preostala polja. Ako ne postoji kombinacija za preostala polja odabrani broj nije moguć za to polje.

Ukoliko u pojedinom stupcu postoji N redaka s kojima se križa, koji imaju isti zbroj u N polja i broj mogućih brojeva im je N, onda su mogući brojevi bilo kojeg od tih redaka ili stupaca sigurni brojevi stupca. Pravilo vrijedi i suprotno.

Ovo pravilo koristi se za metodu sigurnih brojeva. Pomoću ove metode se mogu odrediti sigurni brojevi u stupcu ili retku, ali istovremeno isključiti ti isti brojevi iz drugih polja u stupcu ili retku.

4.6.4. Završetak

Na kraju svake iteracije provjerava se ima li svako polje svoju vrijednost i je li redni broj iteracije manji od 15. Ukoliko sva polja nemaju svoju vrijednost i redni broj iteracije je manji od 15 ulazi se u sljedeću iteraciju, a ako nije, ispisuje se konačno rješenje.

5. RAD APLIKACIJE

S obzirom da se radi o web aplikaciji, aplikacija se mora pokretati na poslužitelju. Dva su uobičajena načina pokretanja aplikacije:

- lokalno
  - *Internet Information Services*
  - primjer: Visual Studio IDE
- udaljeno
  - udaljeni poslužitelj / platforma
  - oblak računala
  - primjer: Microsoft Azure

Bez obzira na način pokretanja aplikacije, rad aplikacije je isti. Najveća razlika je što se pri pokretanju na računalu aplikacija povezuje s lokalnom bazom podataka, dok se kod pokretanja na udaljenoj platformi – oblaku računala aplikacija povezuje s bazom podataka koja se također nalazi u oblaku računala.

5.1. Rješavač

Rješavač predstavlja glavni dio aplikacije. Služi za unos i rješavanje kakura. Postoje dva načina unosa kakura: ručni unos i unos sa slike. U nastavku su detaljnije objašnjeni pojedini dijelovi rješavača.

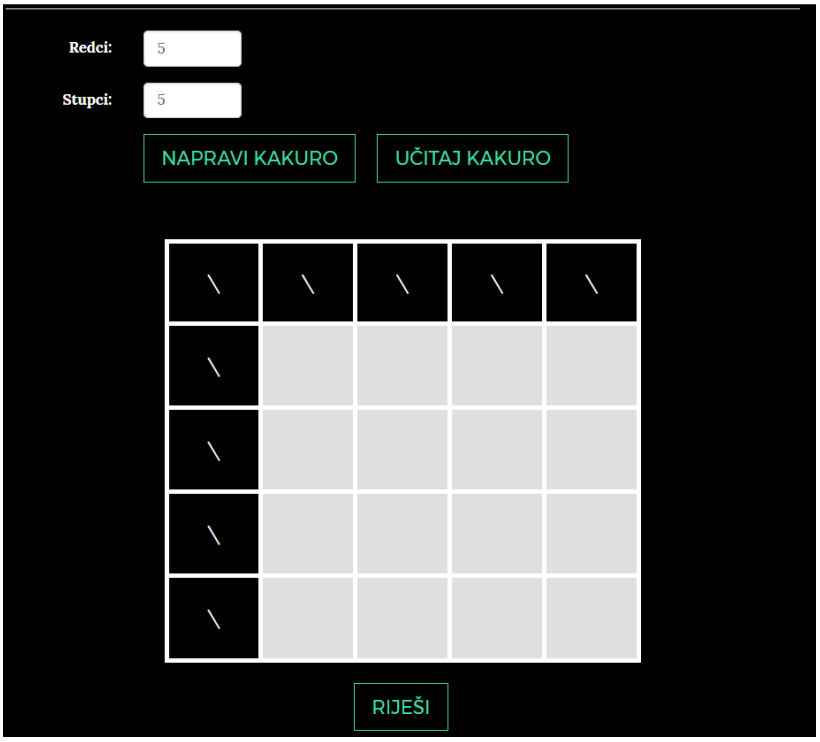
5.1.1. Ručni unos

Ručni unos odnosi se na popunjavanje svakog polja okvira kakura pojedinačno. Korisniku je na početku ponuđena forma za upis broj redaka i stupaca. (slika 5.1.)



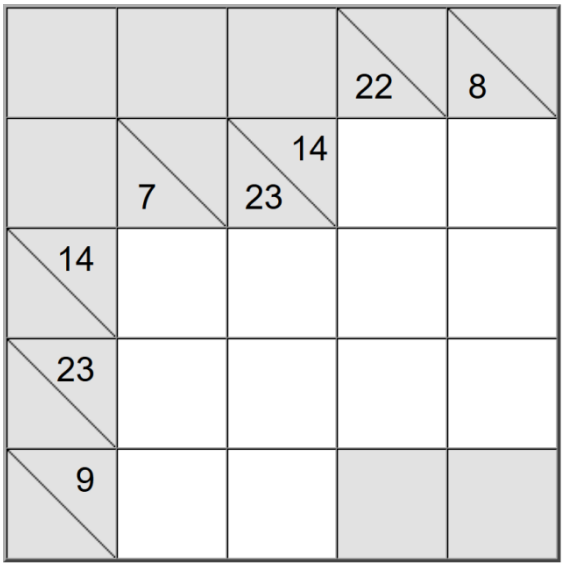
Sl. 5.1. Forma za unos kakura

Ukoliko korisnik želi ručno unijeti kakuro, nakon unosa broja redaka i stupaca, korisniku se pritiskom na gumb „Napravi kakuro“ crta mreža kakura zadanih dimenzija. (slika 5.2.)

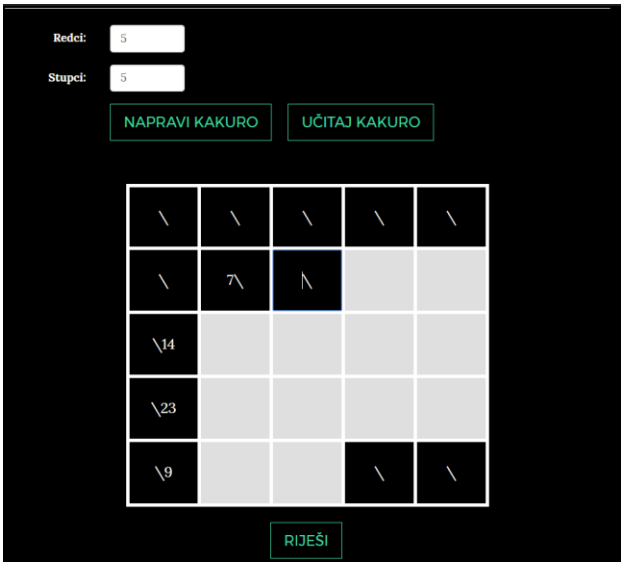


Sl. 5.2. Forma za unos kakura i mreža kakura

Korisnik zatim ručno unosi ostatak okvira i pripadajuće vodoravne i okomite zbrojeve. Nakon unesenih podataka pritiskom na gumb riješi događaju se provjere kakura. Ukoliko provjere daju pozitivan rezultat poziva se algoritam za rješavanje kakura. Na slici 5.3. prikazana je slika kakura koji korisnik ručno unosi, a na slici 5.4. dio postupka unosa kakura.



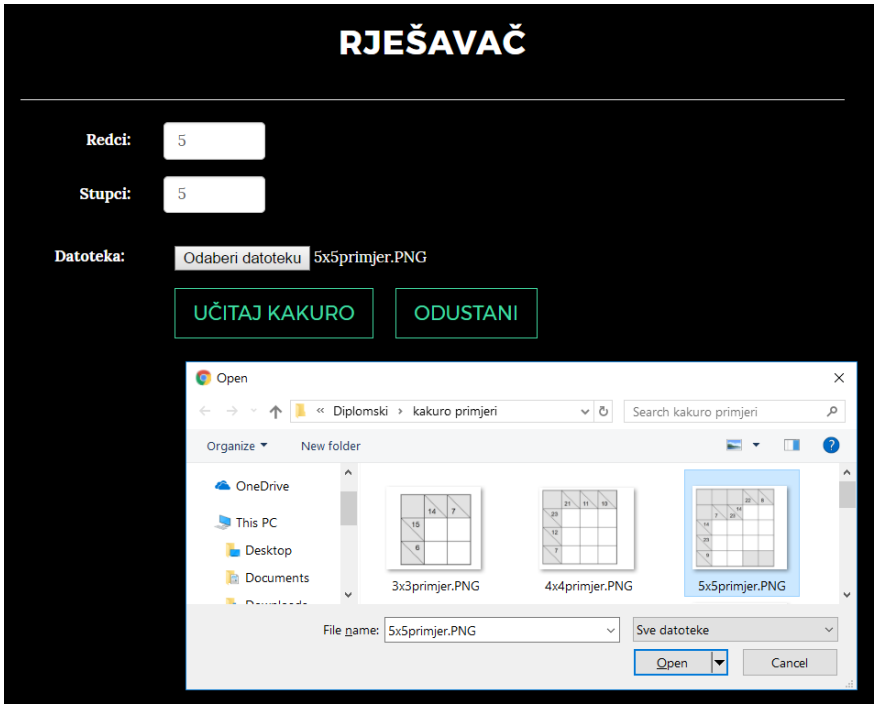
Sl. 5.3. Slika kakura



Sl. 5.4. Ručni unos kakura

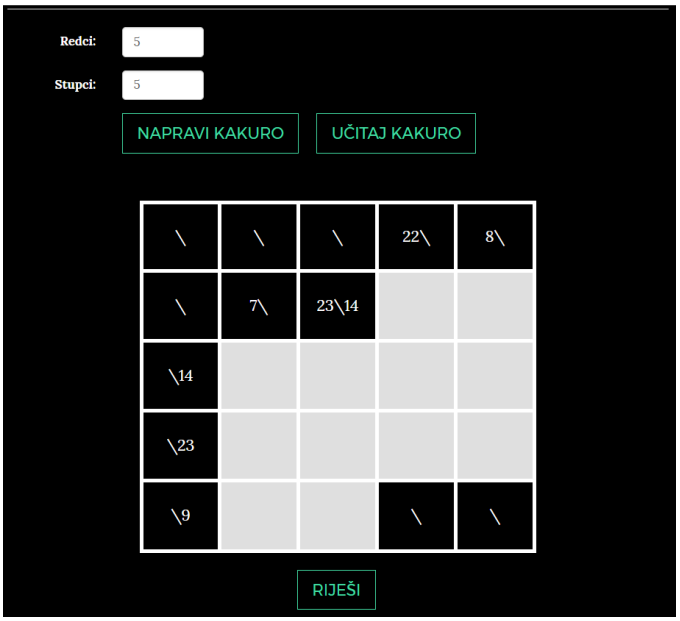
5.1.2. Unos sa slike

Osim ručnog unosa omogućen je unos kakura sa slike. Unosom kakura sa slike smanjena je mogućnost ljudske pogreške i smanjeno je ukupno vrijeme potrebno za unos kakura. Nakon popunjavanja forme sa slike 5.1. ako korisnik želi unijeti kakuro sa slike pritišće gumb „Učitaj kakuro“. Nakon pritiska na gumb otvara se dodatna forma za odabir datoteke. (slika 5.5.)



Sl. 5.5. Forma za odabir datoteke

Ponovnim pritiskom na gumb „Učitaj kakuro“ poziva se metoda za optičko prepoznavanje znakova. Nakon vremena potrebnog za učitavanje kakura na korisničko sučelje ispisuje se učitan kakuro (slika 5.6.). Nakon ispisa kakura korisniku je omogućen popravak pročitanih zbrojeva. Preporučeno je da korisnik nakon učitavanja kakura provjeri učitane zbrojeve jer postoji mogućnost da metoda za optičko prepoznavanje znakova nije prepoznala pravi broj. Na kraju, kao i kod ručnog unosa, pritiskom na gumb „Riješi“ poziva se algoritam za rješavanje kakura.



Sl. 5.6. Kakuro učitan sa slike

5.1.3. Ispis rješenja

Nakon što algoritam riješi kakuro potrebno je ispisati rješenje na korisničko sučelje. Rješenje se ispisuje na način da se u prazna polja ispisuju vrijednosti tih polja. Ukoliko algoritam ne riješi kakuro do kraja, za nepoznate vrijednosti ne ispisuje ništa. Na slici 5.7. prikazan je ispis potpuno riješenog kakura dok je na slici 5.8. prikazan ispis djelomično riješenog kakura.



Sl. 5.7. Potpuno riješen kakuro



Sl. 5.8. Djelomično riješen kakuro

5.2. Pomagač

Pomagač služi za prikazivanje mogućih kombinacija za zadani broj polja i zbroj. Prikazom kombinacija pomagač korisniku olakšava rješavanje kakura.

Pomagač se sastoji od forme za unos podataka i dijela za ispis kombinacija. Na slici 5.9. prikazana je forma za unos podataka u kojoj se od korisnika traži unos broja polja i pripadajući zbroj.



Sl. 5.9. Forma za unos broja polja i zbroja

Nakon unosa broja polja i zbroja pritiskom na gumb „Pronađi kombinacije“ ispisuje se lista svih mogućih kombinacija za dane parametre. Na slici 5.10. prikazane su kombinacije za zbroj 15 u 4 polja.



Sl. 5.10. Kombinacije

Ukoliko korisnik unese zbroj za koji ne postoji moguća kombinacija u zadanom broju polja na korisničko sučelje neće se ništa ispisati. Primjer je zbroj 5 u 3 polja.

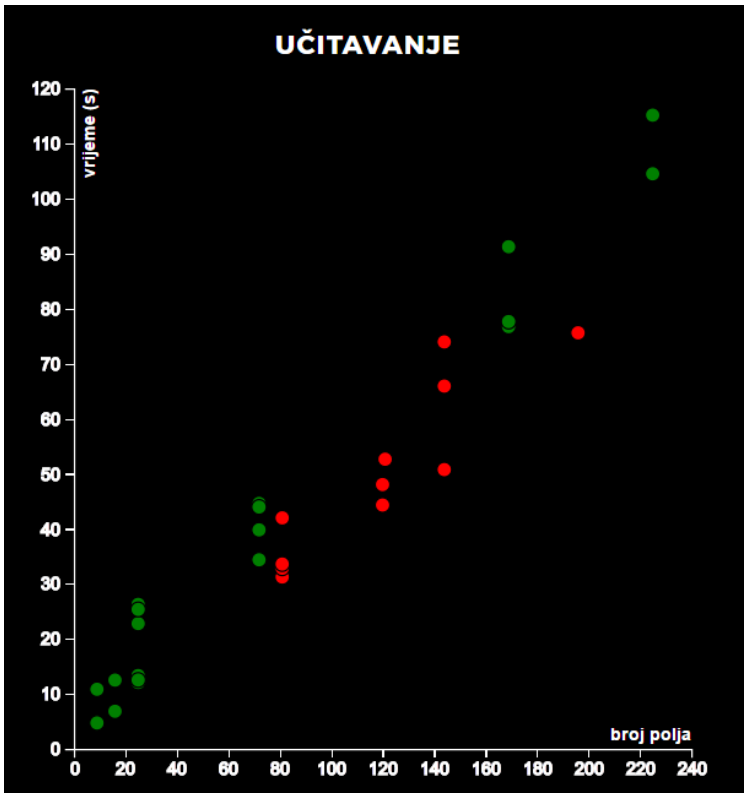
5.3. Analiza

Analiza sadrži dva dijagrama raspršenosti (*Scatter plots*): Dijagram učitavanja i dijagram rješavanja. U nastavku su dijagrami pojedinačno objašnjeni.

5.3.1. Učitavanje

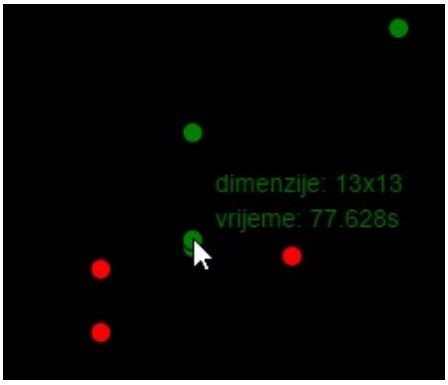
Analiza učitavanja odnosi se na vrijeme potrebno za unos kakura sa slike. Kada korisnik upiše broj stupaca i redaka te odabere datoteku iz koje će se kakuro učitati, pritiskom na gumb „Učitaj kakuro“ pokreće se vremenski brojač koji mjeri vrijeme. Brojač se zaustavlja kada je kakuro učitan.

Na dijagramu raspršenosti za učitavanje kakura prikazana je ovisnost vremena učitavanja o broju polja kakura. Vrijeme učitavanja prikazano je u sekundama. Ako je kakuro u potpunosti učitan prikazan je zelenom bojom, a ako se dogodila pogreška u čitanju prikazan je crvenom bojom. Na slici 5.11. prikazan je dijagram raspršenosti za učitavanje kakura.



Sl. 5.11. Dijagram raspršenosti za učitavanje

Prelaskom pokazivača preko točki koje označavaju kakuro prikazuju se dimenzije i vrijeme potrebno za učitavanje kakura (slika 5.12.).

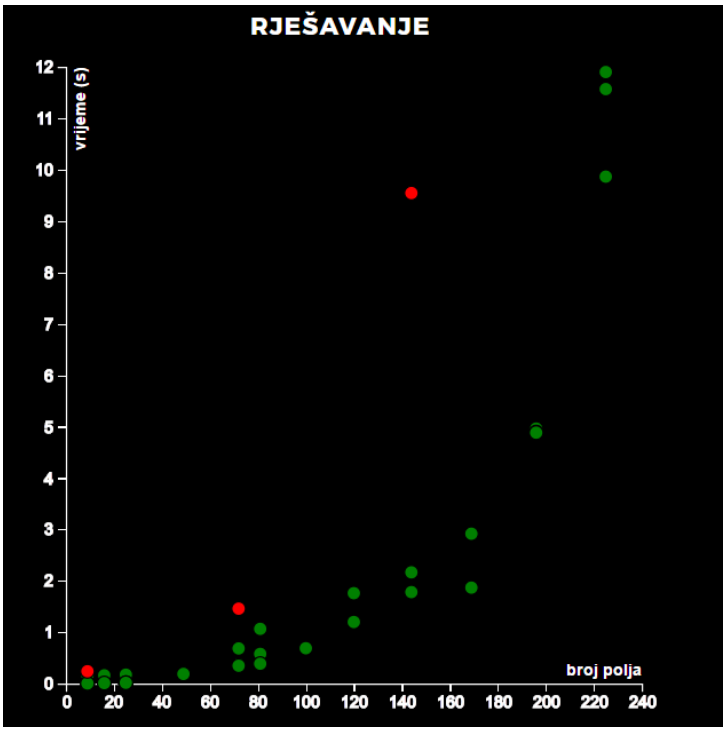


Sl. 5.12. Detalji učitavanja

Na dijagramu na slici 5.11. može se uočiti linearna zavisnost u odnosu broja polja i vremena potrebnog za učitavanje. Povećanjem broja polja raste vrijeme potrebno za učitavanje kakura što je i očekivano.

5.3.2. Rješavanje

Analiza rješavanja odnosi se na vrijeme potrebno algoritmu da riješi kakuro. Slično kao i kod učitavanja vremenski brojač pokreće se kada algoritam počne s radom, a zaustavlja se kada algoritam završi s radom. Na slici 5.13. prikazan je dijagram raspršenosti za rješavanje kakura.



Sl. 5.13. Dijagram raspršenosti za rješavanje

Kao što je očekivano povećanjem broja polja povećava se vrijeme potrebno za rješavanje kakura. Međutim, vidljivo je da zavisnost vremena o broju polja nije linearna kao kod učitavanja nego se može opisati eksponencijalnom funkcijom. Također iz dijagrama na slici 5.13. vidljivo je da se vrijeme rješavanja kakura znatno povećava ako algoritam ne može riješiti kakura do kraja (crvene točke).

6. ZAKLJUČAK

Web aplikacija za rješavanje logičke igre kakura uspješno je napravljena. Kakuro se rješava pomoću algoritma koji koristi standardne metode za rješavanje. U algoritam nisu uključene metode sirove snage („brute force“).

U aplikaciji je omogućen ručni unos i unos kakura sa slike. Unos sa slike koristi metode optičkog prepoznavanja znakova i znatno smanjuje vrijeme potrebno za unos kakura.

Analizom vremena potrebnog za učitavanje i rješavanje kakura potvrđena je pretpostavka da se povećanjem broja polja povećava vrijeme učitavanja i rješavanja kakura. Zavisnost vremena učitavanja o broj polja kakura je linearna, a zavisnost vremena rješavanja o broj polja kakura je eksponencijalna.

7. LITERATURA

[1] Povijest kakura http://www.conceptispuzzles.com/index.aspx?uri=puzzle/kakuro/history
21.06.2017.

[2] The Guardian https://www.bookdepository.com/Guardian-Book-Kakuro-Guardian/9780099499589
21.06.2017.

[3] Kakuro primjer http://www.kakuroconquest.com/8x8/expert
14.06.2017.

[4] Pravila kakura http://www.conceptispuzzles.com/?uri=puzzle/kakuro/rules
23.06.2017.

[5] Magični brojevi http://www.conceptispuzzles.com/resource/1/103.pdf
24.06.2017.

[6] .Net Framework https://en.wikipedia.org/wiki/.NET_Framework
25.06.2017.

[7] MVC http://www.c-sharpcorner.com/UploadFile/3d39b4/Asp-Net-mvc-introduction/
26.06.2017.

[8] Entity Framework https://www.codeproject.com/Articles/363040/An-Introduction-to-Entity-Framework-forAbsolute-B
26.06.2017.

[9] Entity Framework http://www.entityframeworktutorial.net/what-is-entityframework.aspx
26.06.2017.

[10] D3.js kontrolna ploča https://wrapbootstrap.com/theme/renaissance-responsive-admin-template-WB01591L0
29.06.2017.

[11] D3.js karta http://scottcheng.github.io/d3js-101/
29.06.2017.

8. SAŽETAK

U diplomskom radu napravljena je web aplikacija za rješavanje logičke igre kakuro. Osim rješavanja kakura omogućena je pomoć korisniku u rješavanju kakura u obliku prikaza mogućih kombinacija za zadane podatke. Omogućen je ručni unos i unos kakura sa slike. Vremena učitavanja i rješavanja kakura spremaju se u bazu podataka i analiziraju pomoću dva dijagrama. Za izradu aplikacije korištene su Microsoft tehnologije, a za optičko prepoznavanje znakova *Tesseract Engine.*

Ključne riječi: kakuro, web aplikacija, algoritam, OCR, MVC, dijagram

ABSTRACT

In the master thesis, a web application for solving the logical game kakuro was developed. In addition to solving kakuro, it is possible to assist the user in solving the kakuro by showing possible combinations for the given data. Manual input and input from the image are enabled. Loading and solving times are stored in the database and analyzed by using two diagrams. The application was created using Microsoft technology and Tesseract Engine for optical character recognition.

Key words: kakuro, web application, algorithm, OCR, MVC, diagram

9. ŽIVOTOPIS

Ivan Jurić rođen je 03.08.1993. u Osijeku. 2007. godine završava Osnovnu školu Antuna Mihanovića i upisuje Prirodoslovno-matematičku gimnaziju u Osijeku. 2011. godine nakon završetka Prirodoslovno-matematičke gimnazije upisuje preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku. 2015. godine završava preddiplomski studij i upisuje diplomski studij računarstva - izborni blok informacijske i podatkovne znanosti. Od 2004. godine aktivno trenira rukomet. U studenom 2014. godine pridružuje se informatičkom timu u tvrtki GDi GISDATA.

Potpis:

10. PRILOZI

Na CD-u priloženom uz Diplomski rad nalaze se:

Dokumenti:

Diplomski rad - Web aplikacija logičke igre Kakuro - Ivan Jurić.doc

Diplomski rad - Web aplikacija logičke igre Kakuro - Ivan Jurić.docx

Diplomski rad - Web aplikacija logičke igre Kakuro - Ivan Jurić.pdf

Datoteke:

Projekt „KakuroSolver“ preuzet iz Visual Studio 2015 razvojnog okruženja