Brzi heuristički algoritam za Circular Sudoku u programskom jeziku C#

Bježančević, Božo

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

Sveučilišni studij

BRZI HEURISTIČKI ALGORITAM ZA CIRCLE SUDOKU U PROGRAMSKOM JEZIKU C#

Završni rad

Božo Bježančević

Osijek, 2023.

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

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 15.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Božo Bježančević
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	R4039, 27.07.2017.
OIB Pristupnika:	10862840380
Mentor:	doc. dr. sc. Tomislav Rudec
Sumentor:	izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Naslov završnog rada:	Brzi heuristički algoritam za Circular Sudoku u programskom jeziku C#
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Student će u programskom jeziku C# kreirati heuristiku za Posebnu vrstu zagonetke sudoku - Circular sudoku. Tema rezervirana za: Božu Bježenčevića
Prijedlog ocjene završnog 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: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2023.
Datum potvrde ocjene od strane Odbora:	24.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 26.09.2023.

Ime i prezime studenta:	Božo Bježančević
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. studenta, godina upisa:	R4039, 27.07.2017.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Brzi heuristički algoritam za Circular Sudoku u programskom jeziku C#**

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

i sumentora izv. prof. dr. sc. Alfonzo Baumgartner

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:

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

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Božo Bježančević_____, OIB: __10862840380_____, student/ica Fakulteta elektrotehnike,
računarstva i informacijskih tehnologija Osijek na studiju _Sveučilišni prijediplomski studij Računarstvo_, kao autor/ica ocjenskog rada pod naslovom: _Brzi heuristički algoritam za Circular Sudoku u programskom jeziku C#**,**

dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima/icama i djelatnicima/ama ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 26.09.2023._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1. UVOD

Igra sudoku je matematička zagonetka, te je zamišljena da se rješava logičkim razmišljanjem. Klasični sudoku zadan je u velikoj kvadratnoj mreži koja je podijeljena u 9 potpolja kvadratnog oblika. Svako potpolje odgovara veličini *3×3* kvadratića što u potpunosti daje 81 kvadratić ili polje. Zagonetka se smatra riješenom kada se popuni svaki kvadratić s brojevima od jedan do devet tako da se isti broj ponovi točno jednom u svakom retku, stupcu ili *3×3* potpolju. Circle sudoku je jedan od tipova sudoku zagonetke i zadatak ovog završnog rada je osmisliti i napisati algoritam koji će pronaći točno rješenje zagonetke. Algoritam treba biti heuristički što znači da će biti brz i dovesti do zadovoljavajućeg rješenja, te izveden u programskom jeziku C#.

Prvo poglavlje ukratko opisuje zadatak završnog rada. U drugom poglavlju opisana je logička zagonetka sudoku i tipovi sudoku-a. Najčešće korišteni algoritmi koji se koriste za rješavanje sudoku zagonetki objašnjeni su u trećem poglavlju. Četvrto poglavlje opisuje C# programski jezik i pojam heurističkog algoritma upotrebljenih za potrebe rješavanja zadatka završnog rada. Peto poglavlje detaljno opisuje implementaciju i testiranje konačnog algoritma koji rješava circle sudoku zagonetke.

1.1. Zadatak završnog rada

U ovom završnom radu zadatak je napisati brzi heuristički algoritam koji može riješiti circle sudoku zagonetke različitih dimenzija. Rješenje mora biti brzo pronađeno.

2. LOGIČKA ZAGONETKA SUDOKU

2.1. Pregled područja teme

Klasična zagonetka sudoku sastoji se od 81 polja koje treba popuniti poštivajući pravila igre, te je još 2005. godine pronađeno da je zagonetku moguće popuniti na približno $6{,}7 \times 10^{21}$ mogućih načina. [1]

Za postavljanje zagonetke s jedinstvenim rješenjem potrebno je zadati minimalno 17 polja. Još uvijek se traže jedinstvene zagonetke s manje od 17 zadanih polja ali još nisu pronađene. [2]

Igrači koriste brojne metode za rješavanje sudoku zagonetki, a mnoge od njih objasnio je Denis Berthier (francuski profesor, 1948. -) u knjizi „*The Hidden Logic of Sudoku*“ 2007. godine. Neke od metoda koje objašnjava su „*skrivene dvojke*“, „*XY-veze*“ i „*XY-lanci*“. [3]

Računalno se koriste mnogi algoritmi za rješavanje i testiranje sudoku zagonetki. Najčešće korišteni algoritam je unatražni algoritam koji je vrsta *brute-force* algoritma. Najveći problem unatražnog algoritma je što povećanjem dimenzija *n* zagonetke, vrijeme izvršenja algoritma eksponencijalno raste. Prednost algoritma je što je vrlo jednostavan i sigurno će pronaći rješenje pravilno zadane zagonetke. [4]

Jedan od algoritama koji je testiran za rješavanje sudoku zagonetki je i metaheuristički algoritam harmonijskog pretraživanja koji se koristi za rješavanje raznih optimizacijskih problema. Metaheuristika je procedura više razine koja služi za odabir ili generiranje heuristike (djelomičnog algoritma pretraživanja) koja može dati zadovoljavajuće rješenje problema optimizacije ili problema strojnog učenja. Koristi se kada je računalni kapacitet ograničen ili kada su informacije nepotpune. Pristup harmonijskog pretraživanja inspiriran je jazz glazbenicima. Ovaj pristup ima tri faze: inicijalizacija, improvizacija i ažuriranje memorije. Inicijalizacija označava postavljanje početnih uvjeta. Improvizacija je izgradnja rješenja koristeći nasumični odabir. Ovisno o kvaliteti (točnosti) rješenja, memorija će se ažurirati tako da odbaci najlošije rješenje, a bolje rješenje će spremiti za sljedeću generaciju rješenja. Algoritam se sam po sebi nije pokazao vrlo uspješnim i korisnim, ali se raznim modifikacijama i kombinacijama s drugim algoritmima poboljšava korisnost algoritma, ali još uvijek zadržava manu da nekada neće naći moguće rješenje zagonetke. [5]

2.2. Povijest i razvoj sudoku zagonetke

Sudoku je vrsta matematičke zagonetke koja se rješava na temelju logičkog razmišljanja. Sudoku je popularizirao Howard Garns (američki arhitekt, 1905. - 1989.) koji ga po prvi puta izdaje u *Dell Magazines* magazinu. U magazinu je objavljen sudoku dimenzija *9×9* sastavljen od 9 kvadrata dimenzija *3×3*, s ukupno 81 poljem. Sudoku zagonetke mogu biti različitih težina rješavanja i najčešće se koristi za testove inteligencije ili kao razonoda. Wayne Gould (novo zelandski sudac, 1945. -) razvio je računalni program koji brzo zadaje nove jedinstvene sudoku zagonetke. 2004. godine londonske novine *The Times* i kasnije iste godine američke novine počinju tjedno tiskati zagonetke koristeći Gould-ov program za generiranje istih. U posljednje vrijeme postaje sve popularniji te je 2006. godine u talijanskom gradu Lucca održano i prvo svjetsko prvenstvo u rješavanju sudoku zagonetki koje je osvojila Jana Tylova (češka ekonomistica, 1974. -). Slike 2.1 i 2.2 prikazuju primjer i rješenje sudoku zagonetke. [6]

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

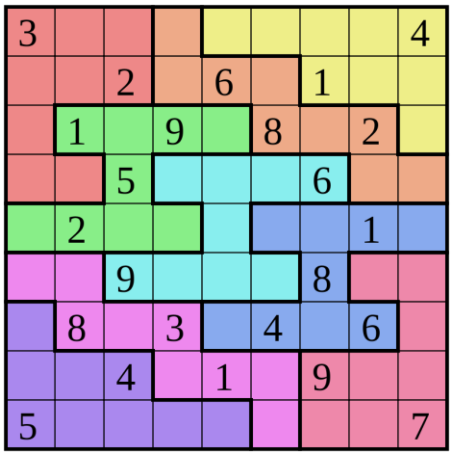
Sl. 2.2. Primjer sudoku zagonetke.

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

Sl. 2.1. Rješenje sudoku-a sa slike 2.2.

2.3. Tipovi sudoku-a

Različiti tipovi sudoku zagonetki razlikuju se po veličini i obliku mreže, te dodatku novih ograničenja. Jigsaw sudoku iako ima mrežu veličine *9×9* kvadratića, razlikuje se od uobičajenog sudoku-a po obliku potpolja veličine 9 kvadratića sveukupno. Na slikama 2.3 i 2.4 su primjer i rješenje jigsaw sudoku zagonetke. Svaka boja označava jedno potpolje veličine 9 kvadratića unutar kojih se ne smije pojaviti isti broj više od jedan puta. [7]

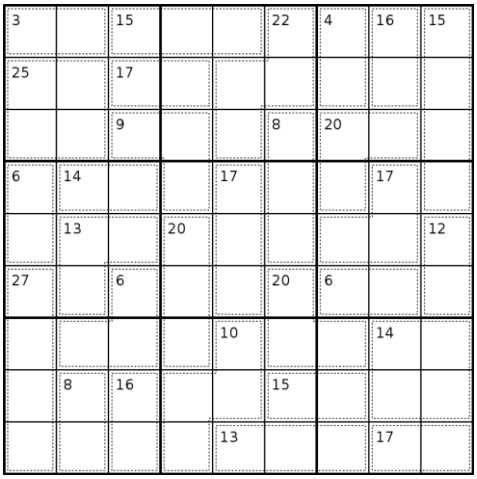


Sl. 2.3. Primjer jigsaw sudoku zagonetke.

Sl. 2.4. Rješenje jigsaw sudoku zagonetke sa slike 2.3.

Mini sudoku je tip sudoku zagonetke veličine mreže *6×6* kvadratića sa potpoljima veličine *3×2* kvadratića. Kako je mreža manja unose se samo brojevi od jedan do šest. Prvi puta pojavila se u novinama *USA Today*. [7]

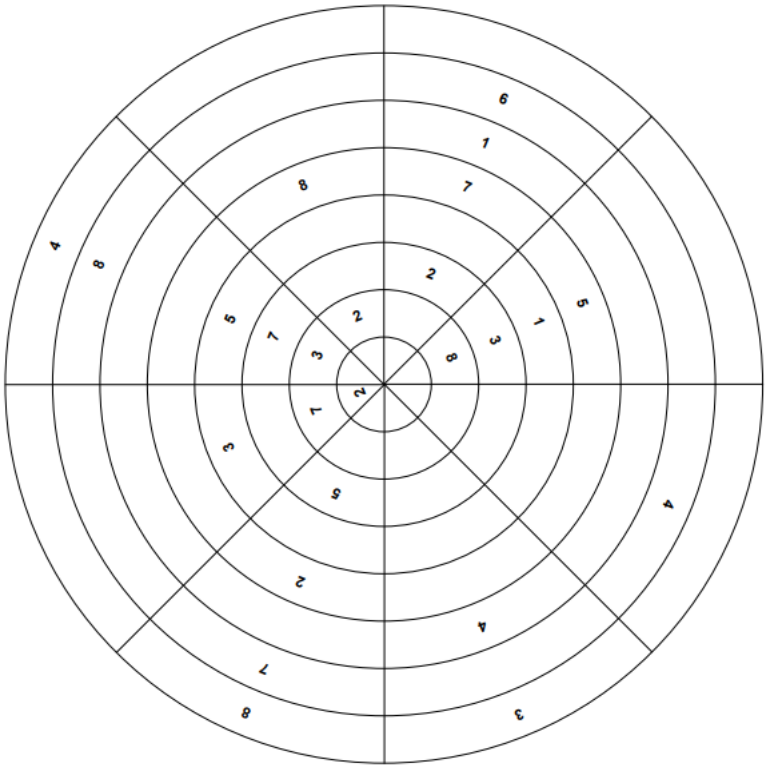
Killer sudoku ujedinjuje elemente sudoku i kakuro zagonetki. Ovisno o osobi koja rješava, killer sudoku može biti lakši ili teži od klasične sudoku zagonetke. Obilježava ga dodano ograničenje koje označava ukupni zbroj unutar isto obojanih ili isprekidanom crtom obrubljenih polja. Unutar tih polja se također ne smiju ponavljati isti brojevi. Primjer i rješenje killer sudoku zagonetke, prikazani su slikama 2.5 i 2.6. [7]



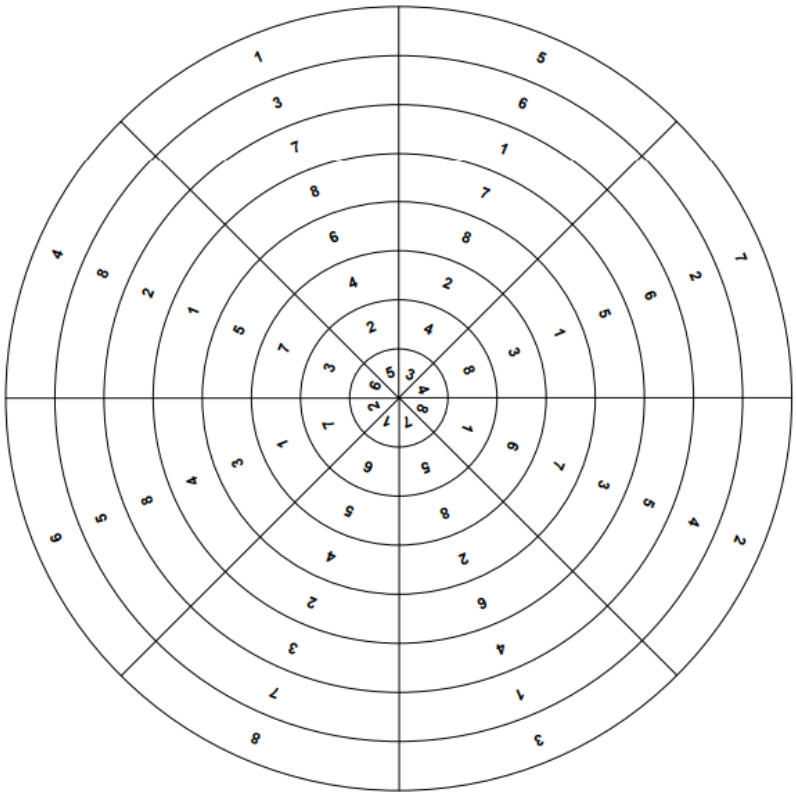
Sl. 2.5. Primjer crno-bijele killer sudoku zagonetke.

Sl. 2.6. Rješenje killer sudoku-a sa slike 2.5.

Circle sudoku je tip sudoku-a u kojemu je mreža zagonetke u obliku kruga, te je podijeljena u prstenove i kružne isječke. U svaki pojedini prsten i kružni isječak moraju se upisati brojevi od jedan do *n*, gdje *n* označava veličinu zagonetke, te se brojevi smiju upisati točno jedan puta. Circle sudoku razlikuje se od klasičnoga sudoku-a i po tome što ima jedno ograničenje manje. Pošto nema nikakvih potpolja, ograničenje da se broj ne smije ponoviti više puta u svakom potpolju ovdje se ne može primijeniti. Slike 2.7 i 2.8 prikazuju primjer i rješenje jedne circle sudoku zagonetke. [8]



Sl. 2.7. Primjer circle sudoku zagonetke.



Sl. 2.8. Rješenje circle sudoku zagonetke sa slike 2.7.

S vremenom se razvijaju različiti tipovi sudoku zagonetki. Razlikuju se po izgledu mreže koja može biti u obliku leptira, paukove mreže i slično. Također se razlikuju po dimenzijama mreže koje mogu biti od *4×4* pa čak i do *100×100* polja. Dodatkom novih ograničenja se dobije novi tip zagonetke pa se tako razlikuje i varijanta sudoku X u kojemu brojevi na glavnim dijagonalama također moraju biti različiti. Svake godine razvija se sve više novih tipova sudoku-a što ih čini još popularnijima. [7]

3. NAJPOZNATIJI ALGORITMI RJEŠAVANJA LOGIČKE ZAGONETKE SUDOKU

Standardni sudoku sadrži 81 polje u mreži dimenzija *9×9* i ima 9 potpolja. Svako potpolje je križanje prva, srednja ili zadnja tri retka ili stupca. Svako polje sadrži brojeve od jedan do devet i svaki broj se mora pojaviti točno jednom u svakom retku, stupcu ili potpolju. Sudoku započinje s nekim poljima koji već sadrže brojeve i cilj zagonetke je popuniti sva polja. Ispravne zagonetke imaju samo jedno rješenje. Igrači koriste mnoštvo računalnih algoritama kako bi riješili sudoku, proučavali njegova svojstva, te napravili nove zagonetke uključujući sudoku sa zanimljivim simetrijama. Nekolicina računalnih algoritama riješit će većinu *9×9* sudoku zagonetki u nekoliko trenutaka, no povećanjem dimenzija *n* raste i kompleksnost. Rast kompleksnosti ograničavaju svojstva sudoku-a koja se mogu konstruirati, analizirati i riješiti povećanjem *n*. Bez svojstava (pravila) sudoku-a, broj mogućih kombinacija rješenih zagonetki eksponencijalno bi rastao povećanjem dimenzija *n*, ali poštujući ta svojstva brzina rasta se znatno smanjuje. [9]

3.1. Stohastičko pretraživanje / metode optimizacije

Stohastičkim algoritmima se također može riješiti sudoku. Stohastički algoritmi su takvi algoritmi gdje je barem jednom neka odluka donesena nasumičnim odabirom tokom njegova izvođenja. Primjer ove metode je da se nasumično dodijeli broj praznim poljima, provjeri broj grešaka, te izmiješa umetnute brojeve sve dok broj grešaka ne postane nula. [9]

3.2. Ograničeno programiranje

Sudoku se može modelirati kao *constraint satisfaction problem.* To su matematička pitanja određena kao objekti koji zadovoljavaju ograničenja. Ograničenim programiranjem se koriste algoritmi zaključivanja temeljeni na ograničenjima primjenjivim na modeliranju i rješavanju problema. Ograničeno programiranje je najbrža metoda, a može riješiti bilo koju sudoku zagonetku ako se koristi zajedno s ostalim metodama. [9]

3.3. Unatražni algoritam (engl. backtracking)

Unatražni algoritam radi tako da se isproba jedan od dostupnih brojeva poštujući zadane uvjete u svakom polju. Ako se nastavkom igre uspostavi da broj ne odgovara, vraća se nazad i ispravlja upisani broj. Princip vraćanja unazad algoritmu donosi naziv unatražni. Najveća prednost unatražnog algoritma je ta što će sigurno dovesti do točnog rješenja zagonetke ako je zagonetka

jednoznačno zadana. Težina zadane zagonetke ne utječe na vrijeme izvođenja algoritma. Kako bi u potpunosti riješio zagonetku potrebno mu je znatno više vremena. [9]

4. PROGRAMSKA PODRŠKA

4.1. C# programski jezik

C# je objektno-orijentirani programski jezik razvijen 2000. godine. Dizajnirao ga je Anders Hejlsberg (danski programski inženjer, 1960. -) sa svojim timom iz tvrtke Microsoft. Razvili su ga kako bi .NET platforma dobila programski jezik koji maksimalno iskorištava njene sposobnosti. Sličan je Java i C++ programskim jezicima. C# je zamišljen da bude jednostavan, moderan, svestran objektno-orijentirani programski jezik. U siječnju 1999. godine Anders je sastavio tim koji je razvio novi programski jezik koji je nazvao Cool (*C-like Object Oriented Language*). Microsoft je planirao zadržati ime Cool ali je na kraju javno objavljen kao C# (*C Sharp)*. U početku je C# bio smatran kao kopija programskog jezika Java ali od 2005. godine Java i C# razvili su se u dva vrlo različita programska jezika. [10]

Konstante i varijable oblik su podataka te program s njima radi. Varijable je prije korištenja potrebo definirati i inicijalizirati. Deklaracija je dodjeljivanje imena nekoj varijabli da ju program može prepoznati. Dodjeljivanje vrijednosti varijabli naziva se inicijalizacija. Posebna vrsta varijabli koje se ne mogu mijenjati nazivaju se konstante. One se mogu inicijalizirati samo jednom i ne mogu se mijenjati. Koriste se kada se vrijednost varijable želi zaštititi od ostalih programera kroz cijeli program. Ključna riječ „*const*“ koristi se ispred tipa podatka varijable koju se želi koristiti kao konstantu. Tablica 4.1 prikazuje osnovne tipove podataka s njihovim veličinama u bitovima. [11]

Tablica 4.1. Osnovni tipovi podataka

Tip podatka	Veličina podataka (u bitovima)
char	16
int	32
float	32
double	64

C# koristi više tipova operatora. Aritmetički operatori su najvažniji te se dijele na unarne i binarne. Unarni aritmetički operatori imaju samo jednog operanda i najviše se koriste za uvećavanje ili umanjivanje vrijednosti operanda, npr. „--i“.[12]

Poznatiji od unarnih su binarni aritmetički operatori. Binarni operatori su: „+, -, *, /, %“. Koriste dva operanda (varijable). Modulo operacija „%“ je operacija koja vraća ostatak cjelobrojnog dijeljenja kao rezultat. Operator dodjeljivanja vrijednosti je „=“. Pisanje koda može se olakšati pisanjem izvedenica dodavanjem binarnih operatora. Različite izvedenice su „+=, -=, *=, /=, %=“ [12]

Operatori usporedbe koriste se kako bi se usporedile dvije varijable. Najčešće se koriste kod grananja i petlji. Operatori usporedbe su: „<, >, <=, >=, ==, !=“ (manje, veće, manje ili jednako, veće ili jednako, potpuno jednako i različito).

Petlje omogućavaju da se jedna ili više linija koda izvršavaju određeni broj puta. Broj ponavljanja uglavnom nije odmah poznat. One omogućavaju da se kod izvršava sve dok zadovoljava određeni uvjet. Koriste se i ulančano. U C# programskom jeziku koriste se petlje *while, for* i *do-while.* [12]

Kada unaprijed znamo koliko puta se ponavlja određeni dio koda koristi se *for* petlja. Svaki puta kada se prođe kroz petlju mijenja se vrijednost kontrolnoj varijabli. Kontrolnu varijablu mora se definirati i inicijalizirati prije upotrebe petlje. Uvjet *for* petlje mora imati rezultat tipa *boolean,* to jest mora biti logički podatak. Petlja se prestaje izvoditi kada vrijednost uvjeta postane logička neistina. Zadnji podatak u *for* petlji predstavlja inkrement, odnosno iznos za koliko se kontrolna varijabla mijenja sa svakom izvedbom petlje. [12]

Ako se ne zna koliko puta će se kod unutar petlje morati izvesti, koristi se *while* petlja koja se izvodi sve dok je uvjet petlje istinit. [12] Ako se ne zna koliko puta će se kod unutar petlje morati izvesti, ali se zna da će se morati izvesti barem jednom, koristi se *do-while* petlja. Kod ove petlje prvo se navodi kod koji će se izvesti, a tek onda uvjet koji se treba zadovoljiti da se kod ponovno izvrši. [12] Razlika između *for, while,* i *do-while* petlji je u tome ako se zna koliko puta će se kod morati izvesti, te hoće li se uvijek morati barem jedanput izvesti.

If, if-else i *switch* kontrole su toka koje omogućavaju izvršavanje operacija ako je zadovoljen određeni uvjet. Ako je određeni uvjet zadovoljen, izvest će se kod unutar *if* petlje, te je zbog toga i jednostavnosti korištenja upravo ona najkorištenija. [12]

Kontrola toka slična *if*-u je *if-else. Else* dijelom kontrole toka izvodi se dio koda ako uvjet za *if* kontrolu toka nije zadovoljen. Pisanje koda znatno je olakšano korištenjem ove kontrole toka. Više *if-else* kontroli toka može biti ulančano iako se ne preporuča osim ako je prijeko potrebno. [12]

Jednostruko grananje ovisno o vrijednosti uvjeta, koji može biti cjelobrojna varijabla ili izraz, omogućava nam *switch* kontrola toka. Provjerom vrijednosti uvjeta se izvršava dio koda pridružen konstanti kojoj odgovara. *Switch* kontrola toka uvijek mora imati *break* naredbu za svaku konstantu koja zaustavlja izvršavanje *switch* kontrole toka i nastavlja s izvođenjem koda nakon *switch* bloka. *Default* naredba izvršava pridruženi dio koda ako uvjet ne odgovara ni jednoj konstanti. [12]

Cjeline istovrsnih podataka označavaju polja. Prednost polja je pohranjivanje više podataka u memoriju sa jednakim imenom. Pojedinim elementima polja pristupa se pomoću indeksa, odnosno rednog broja pojedinog elementa u polju. Polja mogu biti jednodimenzionalna i višedimenzionalna. Jednodimenzionalna polja su zapravo niz elemenata, a njihov indeks je udaljenost pojedinog elementa od početnog elementa polja. Prvi element u nizu uvijek ima indeks „0“, a posljednji element ima indeks za jedan manji od veličine polja. Veličina polja se ne može mijenjati tokom izvođenja programa. Poljima se mogu dodijeliti vrijednosti (polje se može inicijalizirati) tako da se vrijednosti unesu u vitičaste zagrade i odvoje međusobno odvoje zarezom. [13]

4.2. Heuristički algoritam

Ako su uobičajene metode spore ili dolaze do netočnog rješenja koristi se heuristika. Heuristika je tehnika koja u tim slučajevima ubrzava pronalazak rješenja ili približava točnom rješenju. Heuristička funkcija u algoritmima pretraživanja kod svakog grananja vrednuje sve opcije i ovisno o njima odabire koju granu će slijediti. Zadatak heuristike je riješiti problem u zadovoljavajućem vremenu. Optimalnost, potpunost, točnost, preciznost i vrijeme izvršavanja su kriteriji po kojima se odlučuje hoće li se koristiti heuristika ili ne. [14] Algoritam za rješavanje circle sudoku zagonetke implementirat će heuristiku tako da prvo prolazi kroz dubinu tražeći prazna polja sa najmanjim brojem mogućih kandidata, prioritizirajući polja koja imaju samo jednog mogućeg kandidata s ciljem smanjenja broja rekurzivnih poziva.

5. PROGRAMSKO RJEŠENJE

5.1. Uvod u rješenje

Zadatak algoritma je ispuniti zadanu circle sudoku zagonetku brojevima zadovoljavajući kriterije rješavanja circle sudoku-a. Kako bi olakšali izvedbu algoritma, zadani circle sudoku spremit će se u dvodimenzionalnu matricu gdje svaki prsten odgovara retku matrice, a svaki isječak odgovara stupcu. Algoritam šeće poljima matrice i za svako prazno polje određuje moguće brojeve koji se mogu upisati. Algoritam će prvo proći kroz dubinu tražeći polje sa najmanjim brojem mogućih kandidata, te ako pronađe polje sa samo jednim kandidatom prekinut će daljnje pretraživanje i upisat će kandidata u polje. Nakon toga kreće od početka matrice ponovno tražeći prazno polje sa najmanjim brojem kandidata. Ako algoritam dođe do praznog polja za kojeg nema nijednog mogućeg kandidata vratit će se na polje koje je prethodno popunio i izmijeniti broj na sljedećeg mogućeg kandidata. Ako je algoritam prošao kroz sva moguća polja i isprobao sve moguće kombinacije, to jest vratio se na prvo polje koje je upisao i za njega više nema mogućih kandidata to znači da zagonetka nema rješenja.

5.2. Implementacija rješenja

5.2.1. Klasa Sudoku

Klasa *Sudoku* sadrži svojstvo atribut *Puzzle* pomoću kojega možemo dohvatiti ili spremiti sudoku zagonetku u dvodimenzionalno polje cijelih brojeva. Klasa također sadrži metode *FindCandidates*, *SolveSudoku* i *PrintSudoku* koje su sastavni dio algoritma za rješavanje circle sudoku zagonetke, te njeno ispisivanje na ekran. Klasa je prikazana na slici 5.1.

```
2 references
class Sudoku
{
    10 references
    public int[,] Puzzle { set; get; }

    1 reference
    private List<int> FindCandidates (int row, int column)...

    2 references
    public bool SolveSudoku()...

    2 references
    public void PrintSudoku()...

}
```

Sl. 5.1. Klasa *Sudoku* sa svojstvom i metodama

5.2.2. Metoda za pronalazak kandidata za određeno polje zagonetke

Metoda *FindCandidates*, prikazana na slici 5.2., pronalazi moguće kandidate za određeno polje circle sudoku zagonetke. Kao parametre prima broj retka i stupca, to jest indekse matrice koji određuju polje zagonetke. Kao rezultat vraća listu cjelobrojnih podataka koji označavaju sve moguće kandidate za zadano joj polje. *List<T>* lista je kojoj se ne mora zadati početnu veličinu, nego joj se tokom izvođenja programa veličina mijenja ovisno o količini podataka koji se u nju spreme. Jednostavna je za pretraživanje i dodavanje novih podataka na kraj liste. Prvo se inicijalizira lista cjelobrojnih podataka *candidates* i dohvaća dimenziju zagonetke koju sprema u varijablu *length*. U prvoj *for* petlji se uzimaju brojevi od 1 do dimenzija zagonetke. Druga *for* petlja provjerava nalazi li se već kandidat *number* u tom retku i stupcu. Varijabla *isCandidate* je zastavica kojom se signalizira da se broj još ne nalazi u tom stupcu ili retku, te je kandidat za zadano polje. Ako se broj već nalazi u tom retku ili stupcu, zastavici *isCandidate* se mijenja vrijednost na „*false*“ i naredbom „*break*“ završava se izvođenje *for* petlje. Ako se *number* ne nalazi u istom retku ili stupcu, zastavica zadržava vrijednost „*true*“ koja joj je dodijeljena svakim prolaskom kroz prvu *for* petlju, to jest promjenom kandidata *number* za zadano polje. *If* naredbom provjerava se vrijednost zastavice *isCandidate* i ako je uvjet zadovoljen, *number* se dodaje na kraj liste *candidates* koju metoda na kraju vraća kao rezultat.

```
private List<int> FindCandidates (int row, int column)
{
    List<int> candidates = new List<int>();
    int length = Puzzle.GetLength(0);

    for (int number = 1; number <= length; number++)
    {
        bool isCandidate = true;

        for (int i = 0; i < length; i++)
        {
            if (Puzzle[i, column] == number ||
                Puzzle[row, i] == number)
            {
                isCandidate = false;
                break;
            }
        }
        if (isCandidate)
            candidates.Add(number);
    }
    return candidates;
}
```

Sl. 5.2. Metoda koja pronalazi moguće kandidate za zadano polje circle sudoku zagonetke.

5.2.3. Metoda za rješavanje circle sudoku zagonetke

Metoda *SolveSudoku* ne prima nikakve parametre. Kao rezultat vraća *boolean* vrijednost je li zagonetka riješena ili ne. Varijable *row*, *column* i *length* su pomoćne varijable u funkciji. Algoritam prvo prolazi kroz matricu tražeći prazno polje. Kada pronađe prazno polje sprema položaj praznog polja u varijable *row* i *column*. Unutar prve *for* petlje deklarirana je varijabla *onlyCandidate* koja služi kao zastavica kada algoritam pronađe prazno polje sa samo jednim mogućim kandidatom i postavljena je na vrijednost „*false*“. Algoritam pomoću metode *FindCandidates* kojoj predaje položaj trenutnog polja dobiva listu mogućih kandidata koje sprema u varijablu *newCandidates*. Uvjetom „*row < 0*“ omogućeno je da pomoćne varijable sadrže položaj prvog praznog polja. U pomoćnu varijablu *candidates* sprema se lista mogućih rješenja pronađenih metodom *FindCandidates*. Uvjet „*newCandidates.Count < candidates.Count*“ provjerava ima li nova lista manje mogućih kandidata od postojeće. Ako ima, trenutno provjereno polje se sprema u pomoćne varijable kao i lista kandidata za to polje. Drugi *if* uvjet provjerava postoji li samo jedno moguće rješenje za trenutno polje, to jest sadrži li lista *candidates* samo jednu vrijednost. Ako je kandidat jedini, zastavica *onlyCandidate* mijenja vrijednost u „*true*“ i prekida se izvođenje *for* petlji. Na slici 5.3. je prikazan prvi dio metode.

```
public bool SolveSudoku()
{
    int row = -1;
    int column = -1;
    int length = Puzzle.GetLength(0);
    List<int> candidates = new List<int>();

    for (int i = 0; i < length; i++)
    {
        bool onlyCandidate = false;
        for (int j = 0; j < length; j++)
        {
            if (Puzzle[i, j] == 0)
            {
                List<int> newCandidates = FindCandidates(i, j);
                if (row < 0 ||
                    newCandidates.Count < candidates.Count)
                {
                    row = i;
                    column = j;
                    candidates = newCandidates;
                }
                if (candidates.Count == 1)
                {
                    onlyCandidate = true;
                    break;
                }
            }
        }
        if (onlyCandidate)
            break;
    }
```

Sl. 5.3. Funkcija koja rješava circle sudoku zagonetku.

Nakon izvođena *for* petlji, ili njihovog prekida, provjerava se je li sudoku zagonetka riješena gledajući vrijednosti pomoćne varijable *row*. Ako je sudoku riješen, vrijednost varijable će biti "-1" i metoda će vratiti vrijednost „*true*". U slučaju da je položaj nekog polja spremljen u pomoćne varijable, *for* petljom prolazi se kroz listu *candidates* i u zadano polje upisuje prvi kandidat. Provjerava se je li sada sudoku zagonetka riješena rekurzivno pozivajući metodu *SolveSudoku*. Ako nije riješen, vrijednost polja vraća se na „0" označavajući da je to prazno polje. Nakon toga upisuje se sljedeći mogući kandidat s liste. Ako se nakon pokušaja svih mogućih kandidata ne nađe rješenje, metoda vraća vrijednost „*false*" označavajući da circle sudoku zagonetka nema rješenja. Slika 5.4. prikazuje nastavak metode *SolveSudoku*.

```
if (row < 0)
{
    return true;
}

else
    for (int i = 0; i < candidates.Count; i++)
    {
        Puzzle[row, column] = candidates[i];

        if (SolveSudoku())
        {
            return true;
        }
        else Puzzle[row, column] = 0;
    }
return false;
```

Sl. 5.4. Nastavak metode *SolveSudoku*.

5.3. Testiranje algoritma

Slika 2.7. prikazuje primjer circle sudoku zagonetke korištene kao test ispravnosti algoritma. Prvi red matrice u programu odgovara vanjskom prstenu zagonetke, a prvi stupac odgovara prvom isječku lijevo od najvišljeg.

Kod *main* funkcije vidljiv na slici 5.5. služi kao test funkcija algoritma. Varijabla *puzzle* je zapravo dvodimenzionalno polje ili matrica. U nju se unosi zadana circle sudoku zagonetka gdje „0" označava prazna polja. Stvara se objekt *sudoku* koji predstavlja testnu zagonetku i postavlja mu se vrijednost pomoću svojstva *Puzzle* na vrijednost matrice *puzzle.* Ispisuje se neriješena zagonetka

na izlaz odvojena donjim crtama za ljepši prikaz. Ako metoda *SolveSudoku* točno riješi zagonetku, ispisuje ju na izlaz, a ako ne uspije riješiti na izlaz se ispisuje niz znakova „*No solution*“.

```
var watch = new System.Diagnostics.Stopwatch();
watch.Start();

int[,] puzzle = new int[,] {
{ 0, 0, 0, 0, 3, 8, 0, 4 },
{ 0, 6, 0, 4, 0, 7, 0, 8 },
{ 0, 1, 0, 0, 4, 0, 0, 0 },
{ 8, 7, 5, 0, 0, 2, 0, 0 },
{ 0, 0, 1, 0, 0, 0, 3, 5 },
{ 0, 2, 3, 0, 0, 5, 0, 7 },
{ 2, 0, 8, 0, 0, 0, 7, 3 },
{ 0, 0, 0, 0, 0, 0, 2, 0 }
};

Sudoku sudoku = new Sudoku();
sudoku.Puzzle = puzzle;

sudoku.PrintSudoku();
Console.Write("_____ \n\n");

if (sudoku.SolveSudoku())
{
    sudoku.PrintSudoku();
}
else
{
    Console.Write("No solution");
}


watch.Stop();
Console.WriteLine("\n" + watch.ElapsedMilliseconds + "ms");
```

Sl. 5.5. Kod *main* funkcije.

Slika 5.6. je metoda koja ne vraća nikakvu vrijednost. Sa dvije ugniježđene *for* funkcije prolazi kroz svako polje matrice, te koristeći mogućnost interpolacije stringova ispisuje jedno po jedno polje na izlaz sa znakom razmaka između svakoga. Kada dođe do kraja retka, prelazi u novi red kako bi na kraju izlaz zadržao izgled matrice i bio pregledniji. Na slici 5.7. prikazan je izlaz zadane circle sudoku zagonetke i ispod nje crtom je odvojena riješena zagonetka kako bi izlaz bio preglednije prikazan.

```
public void PrintSudoku()
{
    int length = Puzzle.GetLength(0);
    for (int row = 0; row < length; row++)
    {
        for (int column = 0; column < length; column++)
        {
            Console.Write($"{ Puzzle[row, column]} ");
        }
        Console.Write("\n");
    }
}
```

Sl. 5.6. Metoda koja ispisuje sudoku zagonetku na izlaz.

Sl. 5.7. Rješenje circle sudoku zagonetke vidljivo na izlazu.

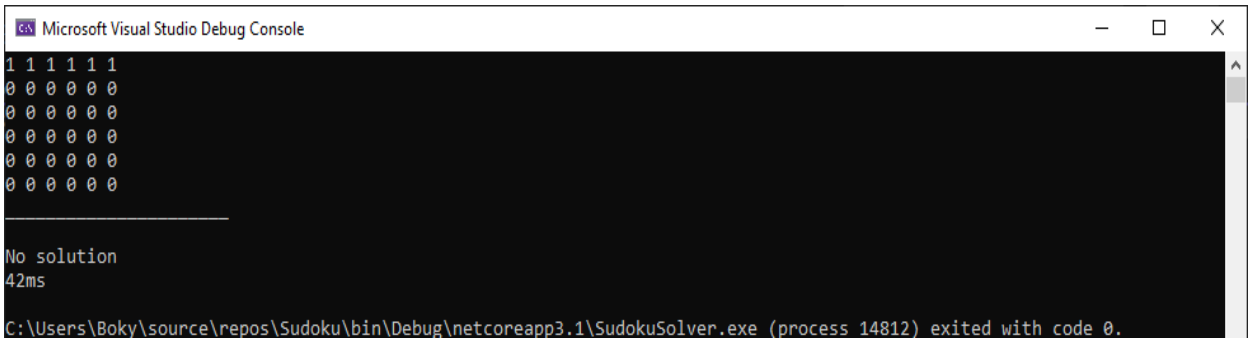
Sljedeće će se testirati vrijeme izvođena algoritma ovisno o dimenzijama zadane prazne zagonetke. Vrijeme se mjeri unutar programa i ispisuje na izlaz. Vrijeme se mjeri pet puta i uzima se prosječno trajanje izvođenja algoritma u milisekundama, a rezultati su prikazani tablicom 5.1. Iz rezultata je vidljivo da se povećanjem dimenzija zagonetke vrijeme izvođenja algoritma eksponencijalno povećava.

Tablica 5.1. Vrijeme izvođenja algoritma

Dimenzije zagonetke	Prosječno trajanje izvođenja algoritma [ms]
3×3	46,6
6×6	64
9×9	76,6
12×12	122,4
15×15	317,2
18×18	575,6
21×21	1193,4
24×24	2134,4
27×27	3751,8
30×30	5488

U sljedećim primjerima biti će prikazani rezultati algoritma ako mu se preda prazna zagonetka, zagonetka sa više mogućih rješenja, te nepravilno zadana zagonetka sa vremenom izvršavanja algoritma. Iz primjera sa slike 5.8. vidljivo je da će algoritam kada mu se preda prazna zagonetka vratiti prvo moguće rješenje koje pronađe, te će uvijek u prvom prstenu i kružnom isječku zagonetke biti upisani brojevi od jedan do *n*, a ostala polja će se popunjavati nakon toga prema algoritmu. Slika 5.9. pokazuje da će algoritam ako mu se preda krivo zadana zagonetka nakon dužeg izvođenja algoritma vratiti na izlaz kako nema mogućeg rješenja za zadanu zagonetku. Slikom 5.10. još jednom se pokazuje kako će algoritam vratiti prvo moguće rješenje koje pronađe ako mu se preda zagonetka sa više mogućih rješenja.

Sl. 5.8. Rješenje prazne circle sudoku zagonetke vidljivo na izlazu.



Sl.5.9. Rješenje krivo postavljene circle sudoku zagonetke vidljivo na izlazu.

Sl. 5.10. Rješenje circle sudoku zagonetke sa više mogućih rješenja vidljivo na izlazu.

Na kraju ćemo još prikazati nekoliko rješenja različitih zagonetki. Na slici 5.11. prikazan je izlaz ako su na glavnoj dijagonali matrice zadane sve jedinice. Slika 5.12. prikazuje rješenje zagonetke gdje se u posljednjem prstenu i kružnom isječku zadaju brojevi redom od jedan do *n*.

Sl. 5.11. Rješenje circle sudoku zagonetke sa jedinicama na glavnoj dijagonali matrice koja predstavlja zagonetku.

Sl. 5.12. Rješenje circle sudoku zagonetke sa uzlaznim brojevima u posljednjem prstenu i kružnom isječku.

6. ZAKLJUČAK

Zadatak ovoga rada je razraditi heuristički algoritam rješavanja circle sudoku zagonetke u C# programskom jeziku, te ga testirati. Algoritam se izvodi uz pomoć klase *Sudoku* u koju se sprema zagonetka i koja sadrži metode koje zapravo izvode algoritam. Pomoću metode *FindCandidates* provjerava se je li određeni broj mogući kandidat za popuniti prazno polje bez da se krše pravila za rješavanje circle sudoku zagonetke. Metoda *SolveSudoku* popunjava prazna polja tražeći prvo polja sa što manje mogućih kandidata. Kada ih pronađe, upisuje kandidate u polja, te ako dođe do pogreške, vraća se unazad pomoću rekurzije i mijenja krivo postavljenog kandidata u sljedećeg mogućeg. Nakon toga algoritam se ponavlja sve dok se zagonetka uspješno ne riješi ili ne isprobaju svi mogući kandidati.

U radu su prikazane i opisane funkcionalnosti C# programskog jezika korištena za razradu ovoga brzog heurističkog algoritma za rješavanje logičke zagonetke circle sudoku. Algoritam se temelji na rekurziji.

Sigurno rješenje circle sudoku zagonetki različitih dimenzija osigurano je korištenjem unatražnog algoritma ako je zagononetka jednoznačno zadana. Algoritam redom prolazi svim poljima zagonetke poštivajući osnovna pravila rješavanja circle sudoku-a. Algoritam omogućava rješavanje circle sudoku zagonetke različitih dimenzija i time je glavni zadatak rada izvršen.

SAŽETAK

Naslov: Brzi heuristički algoritam za circle sudoku u programskom jeziku C#

Zadatak je bio razraditi algoritam za riješavanje logičke zagonetke circle sudoku u C# programskom jeziku. Algoritam je realiziran koristeći dvije metode jedne klase. Bitan koncept algoritma je njegovo brzo izvođenje. Unatražni algoritam najkorišteniji je algoritam za rješavanje ovakvog tipa zagonetki. Ulaz programa je circle sudoku zapisan u obliku matrice kojeg korisnik unosi, a kao izlaz korisnik dobiva rješenje zagonetke, također u obliku matrice. Algoritam je osmišljen da rješava circle sudoku zagonetke različitih dimenzija. Struktura i opis algoritma napisanoga u C# programskom jeziku sadržani su u ovome radu. Kroz nekoliko poglavlja opisani su svi dijelovi algoritma za lakše shvaćanje i obrazloženje samog algoritma. Kako bi algoritam imao smisla, detaljno je opisana zagonetka sudoku i svi njeni tipovi kao i tip circle sudoku, za koji je algoritam početno namijenjen. Zbog popularnosti sudoku-a neprestano se razvijaju novi tipovi zagonetke kao i algoritama koji ih rješavaju.

Ključne riječi: C#, heuristika, matrica, rekurzija, sudoku

ABSTRACT

Title: Fast heuristic algorithm for circle sudoku in C# programming language

The task was to develop an algorithm for solving circle sudoku logic puzzle in C# programming language. The algorithm is realized by using two methods of the same class. An important concept of the algorithm is its fast execution. The backtracking algorithm is the most used algorithm for solving this type of puzzles. User input of the algorithm is a circle sudoku puzzle written in the form of a matrix and the output user gets is the solution of the puzzle, also in the matrix form. Algorithm is made to be able to solve circle sudoku puzzles of different dimensions. The structure and the description of the algorithm written in C# programming language is included in this paper. All parts of the algorithm are described through several chapters for its easier understanding and explanation. For the algorithm to make sense in solving circle sudoku puzzles, sudoku and all its variants are described in detail. Because of sudokus popularity new types of the puzzle as well as the algorithms used to solve them are being constantly developed.

Key words: C#, heuristics, matrix , recursion, sudoku

LITERATURA

[1] K. Y. Ling, Number of Sudokus, Journal of Recreational Mathematics, 33(2): 120-124, 2004

[2] Wikipedia: Mathematics of Sudoku
https://en.wikipedia.org/wiki/Mathematics_of_Sudoku [13.9.2022.]

[3] D. Berthier, The Hidden Logic of Sudoku, Lulu.com, 2007.

[4] E. Chi, K. Lange, Techniques for Solving Sudoku Puzzles, 2012.

[5] R. H. Chae, A. C. Regan, An analysis of Harmony Search for solving Sudoku puzzles, Soft Computing Letters, Vol. 3, 12.2021.

[6] Nezavisne novine: Sudoku
https://www.nezavisne.com/zabava/sudoku [29.6.2022.]

[7] Wikipedia: Sudoku
https://en.wikipedia.org/wiki/Sudoku [29.6.2022.]

[8] Clarity Media: Circle Sudoku
http://www.clarity-media.co.uk/circle-sudoku.php# [29.6.2022.]

[9] Wikipedia: Sudoku solving algorithms
https://en.wikipedia.org/wiki/Sudoku_solving_algorithms [29.6.2022.]

[10] Wikipedia: C# (programming languange)
https://en.wikipedia.org/wiki/C_Sharp_(programming_language) [29.6.2022.]

[11] TutorialsTeacher: C# - Data Types
https://www.tutorialsteacher.com/csharp/csharp-data-types [29.6.2022.]

[12] J. Albahari, C# 10 in a Nutshell, O'Riley Media, Inc., 2022.

[13] Microsoft: C# Documentation - Arrays
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/arrays [29.6.2022.]

[14] Wikipedia: Heuristic (computer science)
https://en.wikipedia.org/wiki/Heuristic_(computer_science) [29.6.2022.]