
	<p>UNIVERZITET U NOVOM SADU FAKULTET TEHNIČKIH NAUKA</p>		

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
 Departman za računarstvo i automatiku
 Odsek za računarsku tehniku i računarske komunikacije

ISPITNI RAD

Kandidat: Ivana Radovanović
Broj indeksa: SV 23/2022

Predmet: Objektno orijentisano programiranje 2
Tema rada: Sudoku

Mentor rada: dr Miodrag Đukić

Novi Sad, decembar, 2023.

SADRŽAJ

1. Uvod.....	1
1.1 Problem Sudoku zagonetke	1
1.1.1 Pronalaženje rješenja	2
1.1.2 Generisanje table za igru	2
1.2 Zadatak	2
2. Analiza problema	4
3. Koncept rješenja	5
4. Opis rješenja	7
4.1 Sekvencijalni program – moduli i osnovne metode	7
4.1.1 Argumenti komandne linije	7
4.1.2 Modul glavnog programa (Main)	8
4.1.3 Modul FileHandler (FileHandler)	8
4.1.4 Modul Sudoku (Sudoku)	9
5. Testiranje	14
5.1 SudokuTest	14
5.2 ¬SudokuTest	14
5.3 areMatricesEqual	14
5.4 testSolveSudokuGood	15
5.5 testSolveSudokuBad	15
5.6 testValidateSudokuGood	15
5.7 testValidateSudokuBad	16
5.8 testValidateSudokuEmpty	16

5.9	testGenerateSudoku	16
5.10	testGenerateSudokuWithLimit	17
5.11	testSetUpGood	17
5.12	testSetUpBad	17
5.13	testSaveAndLoadSudoku	17
5.14	runAllTests	18
6.	Zaključak	19

SPISAK SLIKA

Slika 1: Postavka jedne Sudoku table	1
Slika 2: Rješenje date Sudoku table	1
Slika 3: Postavki jednog Sudoku rješenja u standardnom formatu.....	3
Slika 4: Argumenti komandne linije	7

1. Uvod

1.1 Problem Sudoku zagonetke

Sudoku zagonetka jeste logička zagonetka u obliku kvadratne rešetke. Standardna tabla se sastoji od 9 redova i 9 kolona, što je ukupno 81 polje. Svako polje treba da sadrži jednu od cifara između 1 i 9 ali u takvom rasporedu da se poštuju tri osnovna pravila Sudoku igre. To znači da je potrebno da se svaki element:

- pojavljuje tačno jednom u svakom redu
- pojavljuje tačno jednom u svakoj koloni
- pojavljuje tačno jednom u svakoj podmatrici 3x3

Kako bi počela igra, neophodno je da postoji početna tabla. Cilj ove igre jeste da se popune sva polja tako što se prate tri osnovna pravila.

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



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

Slika 1: Postavka jedne
Sudoku table

Slika 1 Rješenje date
Sudoku table

1.1.1 Pronalaženje rešenja

Rješenja je potrebno pronaći poštujući pravila Sudoku igre. Algoritam koji se koristi naziva se „bektreking“ algoritam (backtracking algorithm). On koristi rekurziju kako bi došao do rješenja. Algoritam radi na sljedeći način:

- a) pronalazi se prazno polje na tabli,
- b) pronađeno polje se popunjava onim brojem koji poštuje pravila Sudoku igre.

Algoritam se nastavlja sve dok se sva polja ne popunjena ili ukoliko popunjavanje svih polja prema pomenutim pravilima nije moguće. U tom slučaju algoritam se rekurzivno vraća, briše neuspjeli pokušaj rješavanja i ide u novu iteraciju, pokušavajući opet da riješi zagonetku. Ovom algoritmu ponekad treba malo više vremena da riješi Sudoku zagonetku zbog broja rekurzivnih poziva, koji u nekim slučajevima mogu biti veliki, u zavisnosti od kompleksnosti zadate zagonetke.

1.1.2 Generisanje table za igru

Popunjavanje određenog broja polja table smatra se generisanjem Sudoku table, koju će kasnije biti moguće riješiti primjenom već pomenutih pravila igre Sudoku. Glavni problem jeste koji je pravi način za generisanje table tako da ona bude rješiva. Jedan od mogućih načina, koji je implementiran u ovom projektu, jeste da se to obezbjedi generisanje popunjene table, po pravilima Sudoku-a, te da se zatim uklone određeni elementi te table.

Generisanje popunjene table vrši se tako što se najprije prazna tabla, odnosno matrica 9x9, dijagonalno popuni sa slučajno izabranim brojevima iz opsega [1, 9], te se zatim pozove „bektreking“ algoritam koji tako postavljenu tablu riješi. Nakon toga, iz svake podmatrice 3x3 sa table briše se, opet slučajno izabrani broj elemenata.

Ovako generisane table će u velikom broju slučajeva biti različite.

1.2 Zadatak

Napraviti C++ program koji ispunjava postavljene funkcionalne zahtjeve. Funkcionalni zahtjevi su:

1. Učitavanje argumenata komandne linije
2. Učitavanje Sudoku zagonetke
3. Čuvanje Sudoku zagonetke
4. Provjera ispravnosti Sudoku rješenja
5. Automatsko rješavanje Sudoku zagonetke
6. Postavka Sudoku zagonetke
7. Ispis statističkih informacija

8. Testiranje programa

Tok programa je sljedeći:

1. Korisniku se nude dvije opcije:
 - I. učitavanje početne Sudoku zagonetke iz datoteke

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

Slika 2: Jedna od postavki
Sudoku rešenja u standardnom
formatu

- II. generisanje početne Sudoku zagonetke od strane programa
2. Na konzoli se ispisuje postavljena Sudoku zagonetka u standardnom formatu (Slika 3)
3. Korisniku se nude 2 opcije:
 - I. učitavanje sopstvenog rješenja Sudoku zagonetke iz datoteke
 - II. rješavanje Sudoku zagonetke od strane programa
4. Nakon završene partije, vrši se validacija rješenja i prikazivanje statističkih podataka odigrane partije

Unutar tekstualne datoteke, u prazna tj. nepopunjena mjesta Sudoku table (matrice) upisana je cifra 0, a cifre svakog reda razdvojene se zapetom. Međutim, da bi čitljivost bila što veća i tabla što jasnija, prilikom prikaza te table, cifra 0 zamijenjena je praznim stringom, tj. „space-om“.

2. Analiza problema

Algoritmi za rješavanje Sudoku zagonetke nisu jedini problem koji se pojavljuje prilikom rješavanja zadatka. Pitanje koje se pojavljuje jeste sama organizacija koda, kao i testiranje funkcionalnosti programa.

Razmatranje modularnosti programa, tj. njegovo dijeljenje na module ili klase, omogućava čistiji i lakše održiv kod. Ovo takođe doprinosi bržem izvršavanju i boljoj upravljivosti koda. Treba posebno voditi računa o efikasnom upravljanju memorijom kako bi se izbjeglo nepotrebno trošenje resursa.

U narednom koraku analize Sudoku table, identifikacija dijelova gdje se mogu primijeniti optimizacije postaje ključna. Proučavanje redova, kolona i podgrupa otkriva potencijalna mjesta za poboljšanja izvršavanja programa. Različite tehnike, poput optimizacije petlji, smanjenja nepotrebnih uslova ili primjene efikasnijih algoritama za provjeru pravila Sudoku-a, mogu značajno ubrzati proces rješavanja.

Sinhronizacija i komunikacija između različitih dijelova koda takođe igraju ključnu ulogu u efikasnom izvršavanju. Pravilno upravljanje dijeljenim promjenljivama i smanjenje potrebe za čestom komunikacijom doprinosi cjelokupnoj performansi sistema.

Kao dodatak, preporučljivo je implementirati i odgovarajuće strategije za testiranje funkcionalnosti programa. Ovo uključuje jedinice testiranja, integraciona testiranja i testiranja performansi kako bi se osiguralo da promjene u kodu ne naruše postojeću funkcionalnost i optimizacije.

3. Koncept rešenja

Rješavanje ovog problema prati sekvencijalni pristup, često korišćen u rješavanju Sudoku problema. Ovaj pristup obuhvata rekurzivnu implementaciju koja istražuje različite kombinacije brojeva kako bi pronašla ispravno rješenje.

Kod se deli na tri klase: *FileHandler* i *Sudoku*, kao i na *SudokuTest* klasu koja će služiti za verifikaciju u modula glavnog programa. Klasa *FileHandler* ne sadrži privatne attribute već služi za rad sa tekstualnim datotekama, pa zato sadrži *static* funkcije, da se instanca klase ne bi pravila. Klasa *Sudoku* ima više privatnih atributa: *int numOfErrors* koji sadrži broj grešaka jedne partije, *int numOfGoodValues* koji sadrži broj dobro odigranih poteza (gdje se polja koja su već bila popunjena ne broje), *int numOfGames* koji sadrži broj odigranih partija, *int numOfFilled* koji sadrži broj popunjenih polja na početku igre, *int counter* koji sadrži broj mogućih rješenja jedne Sudoku table, kao i jedan javni atribut, *vector<vector<int>> matrix* koji predstavlja samu matricu programa i srž zauzeća memorije. Obje klase imaju odgovarajuće javne metode neophodne za funkcionisanje programa koje su opisane u daljem tekstu. *SudokuTest* predstavlja klasu funkcija, gdje je najbitnija *runAllTests()* funkcija koja služi za njihovo pokretanje.

Program ima dva osnovna zadatka: rješavanje i generisanje Sudoku zagonetki. U suštini, oba zadatka se svode na prvi - implementaciju algoritma koji, na osnovu postavke Sudoku table, može generisati validno rješenje. Tabla može biti potpuno prazna ili djelimično popunjena poljs, ali program mora biti sposoban da riješi svaku konfiguraciju. Drugi zadatak proizlazi iz prvog, gdje je potrebno generisati rješenje prazne table i zatim ukloniti određeni broj polja. Na taj način se dobija početna postavka Sudoku zagonetke koja je garantovano rješiva.

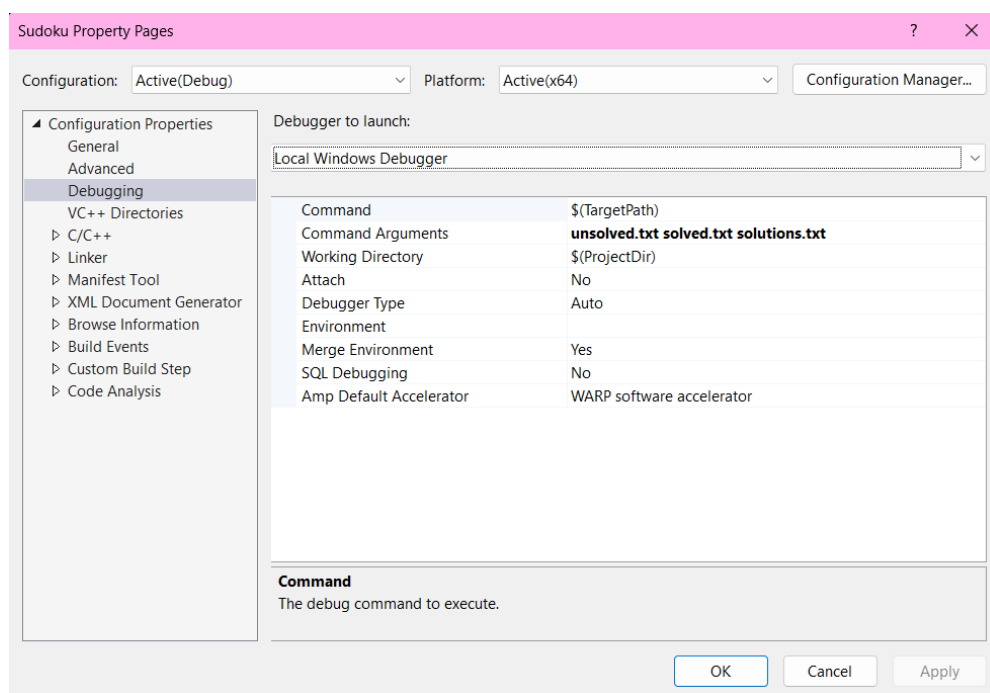
Rješavanje Sudoku-a se svodi na rekurzivni proces. Na prazno mjesto se postavlja odgovarajući broj, a zatim nova tabla, sa tom promjenom, postaje novi "problem" za rješavanje. Ovaj proces se ponavlja sve dok se ne riješi kompletna tabla.

Metode klase trebaju imati odgovarajuće parametre, provjere i postavljene kriterijume zaustavljanja kako bi se osiguralo ispravno funkcionisanje programa. U nastavku su definisane i detaljno objašnjene sve korištene metode u programu.

4. Opis rješenja

4.1 Sekvencijalan program – moduli i osnovne metode

4.1.1 Argumenti komandne linije



Slika 3: Argumenti komandne linije

Program za svoje izvršavanje koristi datoteke koje su zadate kao argumenti komandne linije preko *argv* argumenta main funkcije.

4.1.2 Modul glavnog programa (Main)

```
int main(int argc, char* argv[]);
```

Ovo je glavna funkcija programa. Kroz nju je moguće pokrenuti sam program koji započinje igranje Sudoku igre ili testove za program, ukoliko je to potrebno.

4.1.3 Modul FileHandler (FileHandler)

Modul koji vrši operacije nad fajlovima, čitanje iz datoteke i upisivanje u datoteku, gdje se pri tome dodjeljuje Sudoku klasi njena matrica ili se iz Sudoku klase upisuje matrica.

4.1.3.1 Konstruktor klase FileHandler (FileHandler)

```
FileHandler();
```

Podrazumevani (default) konstruktor klase FileHandler.

4.1.3.2 Destruktor klase Igra (~FileHandler)

```
~FileHandler();
```

Podrazumevani (default) destruktorklase Igra.

4.1.3.3 Metoda za učitavanje iz datoteke (loadSudoku)

```
static void loadSudoku(Sudoku& sudoku, const string& filename);
```

Metoda služi za učitavanje Sudoku matrice iz datoteke u već proslijeđenu Sudoku instancu, tj. u njenu matricu. U datoteci, svaki red matrice smješten je u novi red datoteke, a svaka kolona odvojena je zapetom, pa se tako kroz ovu metodu i učitava. Na osnovu naziva datoteke koja joj se proslijeđuje kao parametar, kreira objekat klase *ifstream* koji koristi za učitavanje.

Parametri:

- datoteka – naziv datoteke iz koje se vrši učitavanje
- Sudoku instanca – instanca u čiju matricu se upisuju učitane vrijednosti iz datoteke

4.1.3.4 Metoda za upisivanje u datoteku (saveSudoku)

```
static void saveSudoku(const Sudoku& sudoku, const string& filename);
```

Metoda služi za upisivanje Sudoku matrice u datoteku iz već proslijeđene Sudoku instance, tj. u njene matrice. U datoteku se svaki red matrice smješta u novi red datoteke, a svaka kolona odvoja se zapetom, pa se tako kroz ovu metodu i upisuje. Na osnovu naziva datoteke koja joj se proslijeđuje kao parametar, kreira objekat klase *ofstream* koji koristi za upisivanje.

Parametri:

- datoteka – naziv datoteke u koju se vrši upisivanje
- Sudoku instanca – instanca iz čije matrice se učitavaju vrijednosti u datoteku

4.1.3.5 Metoda za učitavanje iz datoteke (loadLastMatrix)

```
vector<vector<int>> FileHandler::loadLastMatrix(const string& filename);
```

Metoda služi za učitavanje samo matrice iz datoteke, tj. neriješene matrice koju je korisnik ranije proslijedio programu jer želi da je riješi. U datoteci, svaki red matrice smješten je u novi red datoteke, a svaka kolona odvojena je zapetom, pa se tako kroz ovu metodu i učitava. Na osnovu naziva datoteke koja joj se prosljeđuje kao parametar, kreira objekat klase *ifstream* koji koristi za učitavanje. Metoda vraća učitanu matricu radi dalje obrade u kodu.

Parametri:

- datoteka – naziv datoteke iz koje se vrši učitavanje

4.1.4 Modul Sudoku (Sudoku)

Modul klase Sudoku9 koji sadrži metode date u nastavku, kao i sljedeće privatne attribute: *int numOfErrors*, *int numOfGoodValues*, *int numOfGames*, *int numOfFilled*, *int counter*. Tabla, tj. sama matrica korišćena je kao javni atribut, radi lakšeg kodiranja, *vector<vector<int>> matrix*.

4.1.4.1 Konstruktor klase Sudoku (Sudoku)

```
Sudoku(int size);
```

Konstruktor klase Sudoku. Konstruktor zauzima `sizeof(int)` memorijsku lokaciju za tip *int*, gdje je originalna ideja bila da se dozvoli mogućnost mijenjanja veličine matrice, odnosno da Sudoku igra ima više *nxn* mogućnosti igranja. Takođe, konstruktor zauzima memorijske lokacije i za pet privatnih atributa, što je ukupno 86 memorijskih lokacija.

4.1.4.2 Destruktor klase Sudoku (~Sudoku)

```
~Sudoku();
```

Oslobađa 86 memorijskih lokacija ranije zauzetih u konstruktoru.

4.1.4.3 Metoda za brojanje elemenata (countNonZeroElements)

```
void Sudoku::countNonZeroElements();
```

Metoda prolazi kroz matricu Sudoku instance i broji koliko ima elemenata koji nisu jednaki cifri 0.

4.1.4.4 Metoda za ispis matrice (print)

```
void print();
```

Metoda koja ispisuje vrijednosti matrice Sudoku instance nad kojom je pozvana tako što umjesto svih 0 cifara u datoteci ispisuje u konzolu prazan string, odnosno "space".

4.1.4.5 Metoda za anuliranje matrice (resetMatrix)

```
void resetMatrix();
```

Metoda koja postavlja vrijednosti polja matrice Sudoku instance nad kojom je pozvana na 0.

4.1.4.6 Metoda za provjeru vrijednosti u matrici (checkMatrix)

```
bool checkMatrix();
```

Metoda prolazi kroz svaki red i kolonu matrice Sudoku instance i provjerava da li su sve vrijednosti različite od 0, tj. da li je matrica popunjena. Ukoliko se pronađe barem jedna vrijednost koja je jednaka cifri 0, metoda vraća *false*, u suprotnom vraća *true*.

4.1.4.7 Metoda za provjeru mogućnosti upisa u određeno polje matrice (isSafe)

```
bool isSafe(int row, int col, int value) const;
```

Metoda provjerava da li je moguće postaviti vrijednost *value* u određenu kolonu *col* i u određeni red *row* tako što prolazi kroz taj red i kolonu i provjerava da li vrijede pravila Sudoku igre, kao i kroz podmatrixu 3x3. Ukoliko je moguć upis u proslijeđeni red i kolonu, metoda vraća *true*, inače vraća *false*.

Parametri:

- row – indeks reda polja u koje se upisuje
- col – indeks kolone polja u koje se upisuje
- value – vrijednost koja se upisuje u polje proslijeđenih koordinata

4.1.4.8 Metoda za rješavanje Sudoku table (solveSudoku)

```
bool solveSudoku();
```

Metoda pokušava da riješi matricu Sudoku instance nad kojom je pozvana tako što inicijalizuje privatni atribut *counter* na 0, kako bi pratila broj rješenja, ukoliko postoje. Zatim metoda poziva *solveSudokuHelper* metodu koja istražuje moguća rješenja. Kada ova metoda završi, provjerava se vrijednost *counter*-a. Ukoliko je pronađeno rješenje, *counter* će biti veći od 0, i metoda će vratiti *true*. Inače, rješenje nije pronađeno i metoda vraća *false*.

4.1.4.9 Metoda za implementaciju backtracking algoritma (solveSudokuHelper)

```
bool solveSudokuHelper();
```

Metoda koja rekurzivno pokušava da riješi matricu Sudoku instance nad kojom je pozvana tako što koristi već opisani „backtracking“ algoritam. Ona pronalazi polja sa vrijednošću 0 i pokušava da ih popuni važećom vrijednošću od 1 do 9. Za svaku važeću vrijednost, funkcija se

rekurzivno poziva kako bi nastavila rješavanje zagonetke. Ako se pronađe važeće rješenje, brojač se povećava, i funkcija vraća *true*. Ako važeće rješenje nije pronađeno, funkcija se vraća unazad (backtrack) i pokušava drugu vrijednost za trenutnu ćeliju. Proces se nastavlja sve dok ćelije nisu popunjene i dok se ne pronađe barem jedno rješenje.

4.1.4.10 Metoda za generisanje Sudoku table (generateSudoku)

```
void generateSudoku();
```

Metoda nad pozvanom instancom Sudoku klase poziva dvije druge metode, fillGrid i removeElementsFromGrid kako bi generisala novu tablu Sudoku instance koja će biti rješiva.

4.1.4.11 Metoda za popunjavanje Sudoku table (fillGrid)

```
bool fillGrid();
```

Metoda nad pozvanom instancom Sudoku klase, rekurzivno popunjava Sudoku matricu postavljajući brojeve u prazne ćelije. Iterira kroz svaku ćeliju u Sudoku matrici. Ako je ćelija prazna (sadrži 0), generiše nasumičnu permutaciju mogućih brojeva (od 1 do 9) i pokušava postaviti svaki broj u ćeliju. Provjerava da li je broj bezbjedno postaviti prema pravilima Sudoku-a. Ako se pronađe bezbjedan broj, postavlja ga i rekurzivno poziva metodu da popuni sljedeću ćeliju. Ako se Sudoku može uspješno popuniti, funkcija vraća *true*. Ako nijedan validan broj nije pronađen za određenu ćeliju, vrši povratak i resetuje vrijednost ćelije na 0, pokušavajući različite brojeve dok se ne pronađe validna kombinacija ili dok se sve mogućnosti ne iscrpe.

4.1.4.12 Metoda za brisanje elemenata iz Sudoku table (removeElementsFromGrid)

```
void removeElementsFromGrid();
```

Metoda uklanja nasumičan broj elemenata iz svake 3x3 podmatrice Sudoku instance nad kojom je pozvana. Iterira kroz svaku 3x3 podmatricu, generiše nasumičan broj između 4 i 9 kako bi odredio broj elemenata koje treba ukloniti. Zatim stvara listu jedinstvenih pozicija unutar matrice, nasumično ih miješa, i uklanja određeni broj elemenata iz promiješanih pozicija. Ako element već nije obrisan, postavlja ga na 0, što predstavlja praznu ćeliju u Sudoku matrici.

4.1.4.13 Metoda za validaciju Sudoku table (validateSudoku)

```
bool validateSudoku();
```

Metoda provjerava trenutno stanje Sudoku matrice nad čijom instancom je pozvana, provjeravajući greške u svakoj 3x3 matrici. Greška se javlja ako postoje duplikati vrijednosti unutar reda, kolone ili 3x3 matrice. Broji broj grešaka i čuva ga u promenljivoj numOfErrors. Vraća *true* ako je Sudoku matrica ispravna (bez grešaka), inače *false*.

4.1.4.14 Metoda za validaciju podmatrice 3x3 Sudoku table (isValidMatrix)

```
bool isValidMatrix(int rowStart, int colStart) const;
```

Metoda provjerava ispravnost 3x3 podmatrice unutar Sudoku matrice. Metoda koristi nesortirani skup (unordered set) kako bi pratila jedinstvene vrijednosti unutar podmatrice. Iterira kroz podmatricu i provjerava prisustvo nevažećih vrijednosti (manjih od 1 ili većih od 9) ili duplikata. Metoda vraća *true* ako je podmatrica ispravna, inače vraća *false*.

Parametri:

- rowStart – indeks početnog reda polja 3x3 podmatrice
- colStart – indeks početne kolone polja 3x3 podmatrice
- value – vrijednost koja se upisuje u polje proslijeđenih koordinata

4.1.4.15 Metoda za validaciju reda Sudoku table (checkRowErrors)

```
bool checkRowErrors(int row, int col, int duplicate);
```

Metoda provjerava da li postoje duplikati vrijednosti u istom redu kao i proslijeđena ćelija (red, kolona) Sudoku matrice pretraživanjem udesno i ulijevo od navedene ćelije. Metoda vraća *true* ukoliko je pronađen duplikat, inače vraća *false*.

Parametri:

- row – indeks reda trenutnog polja
- col – indeks kolone trenutnog polja
- duplicate – vrijednost za provjeru pojave duplikata u redu

4.1.4.16 Metoda za validaciju kolone Sudoku table (checkColErrors)

```
bool checkColErrors(int row, int col, int duplicate);
```

Metoda provjerava da li postoje duplikati vrijednosti u istoj koloni kao i proslijeđena ćelija (red, kolona) Sudoku matrice pretraživanjem gore i dolje od navedene ćelije. Metoda vraća *true* ukoliko je pronađen duplikat, inače vraća *false*.

Parametri:

- row – indeks reda trenutnog polja
- col – indeks kolone trenutnog polja
- duplicate – vrijednost za provjeru pojave duplikata u redu

4.1.4.17 Metoda za provjeru početne vrijednosti Sudoku table (checkSetUp)

```
bool checkSetUp(vector<vector<int>>& mat);
```

Metoda provjerava da li data 9x9 matrica odgovara početnoj postavci Sudoku matrice upoređivajući svaki element date matrice sa odgovarajućim elementom interne matrice instance

objekta Sudoku nad kojim je pozvana metoda. Ako se nenula vrijednosti u datoj matrici razlikuju od interne matrice, funkcija vraća *false*, ukazujući na neslaganje. Ako se sve odgovarajuće vrijednosti poklapaju ili su nula, funkcija vraća *true*, označavajući ispravnu postavku.

Parametri:

- *mat* – 9x9 matrica koja se proverava u odnosu na internu Sudoku matricu

5. Testiranje

Testiranje je realizovano kroz klasu `SudokuTest`, koja sadrži javne metode unutar klase, gdje se unutar jedne statične metode pozivaju svi testovi. Cilj testiranja jeste da se kroz granične slučajeve i test primjere utvrdi da li program može da se izbor sa tim slučajevima prije nego što korisnik dobije mogućnost za korišćenjem. Sve metode, sem posljednje, vraćaju *boolean* vrijednost, odnosno vraćaju *true* ukoliko je test prošao, odnosno *false* ukoliko nije. Takođe, u konzoli se pri tom ispisuje koji testovi su prošli a koji ne.

5.1 `SudokuTest`

```
SudokuTest();
```

Podrazumevani (default) konstruktor klase `SudokuTest`.

5.2 `~SudokuTest`

```
~SudokuTest();
```

Podrazumevani destruktor klase `SudokuTest`.

5.3 `areMatricesEqual`

```
bool areMatricesEqual(const vector<vector<int>>& matrix1, const  
vector<vector<int>>& matrix2) const;
```

Dodatna metoda potrebna da provjeri da li su dvije date matrice jednake. Prvo se provjerava da li su obje matrice iste veličine (broj redova i broj kolona). Ako su dimenzije različite ili matrice nemaju elemenata (prazne su), metoda odmah vraća *false*, što znači da matrice nisu jednake.

Ako matrice imaju iste dimenzije, metoda prolazi kroz svaki element obje matrice u petlji. Ako pronađe bilo koji par elemenata koji se razlikuju, odmah vraća *false*, signalizirajući da

matrice nisu jednake. Ako se nijedan neispravan par elemenata ne pronađe tokom iteracije, metoda konačno vraća *true*, što znači da su matrice jednake.

Parametri:

- *matrix1*— referenca na prvu matricu koja se upoređuje
- *matrix2*— referenca na drugu matricu koja se upoređuje

5.4 testSaveAndLoadSudoku

```
bool SudokuTest::testSaveAndLoadSudoku();
```

Test provjerava ispravnost funkcionalnosti čuvanja i učitavanja Sudoku igre iz datoteke. Kreira se instanca Sudoku objekta sa veličinom 9x9 i popunjava matricu nerješenom Sudoku tablom. Poziva se funkcija *saveSudoku* kako bi se sačuvala trenutna konfiguracija Sudoku igre u datoteku sa određenim imenom. Kreira se druga instanca Sudoku objekta, a zatim se poziva funkcija *loadSudoku* kako bi se učitala Sudoku konfiguracija iz prethodno sačuvane datoteke.

Upoređuje se matrica prvobitnog Sudoku objekta sa matricom učitanoj Sudoku objekta. Ako su matrice jednake, test je uspješan.

5.5 testSolveSudokuGood

```
bool SudokuTest::testSolveSudokuGood();
```

Test provjerava ispravnost funkcionalnosti rješavanja Sudoku igre. Kreira se instanca Sudoku objekta sa veličinom 9x9, matrica te instance se popuni sa testnom matricu. Čuva se kopija originalne konfiguracije Sudoku igre radi kasnije provjere. Poziva se funkcija *solveSudoku* kako bi se pokušala riješiti Sudoku tabla. Ako rješenje postoji, matrica Sudoku igre će biti promijenjena. Proverava se da li rješavanje Sudoku igre dovelo do promjene u matrici u odnosu na originalnu konfiguraciju. Ako nema promjena, test se smatra neuspješnim. Inače, test se smatra uspješnim.

5.6 testSolveSudokuBad

```
bool SudokuTest::testSolveSudokuBad();
```

Test provjerava ispravnost rada funkcije za rješavanje Sudoku igre u slučaju kada je unaprijed zadata tabla nerješiva. Kreira se instanca Sudoku objekta sa veličinom 9x9 i njena matrica se popuni matricom sa nerješivom Sudoku tablom. Čuva se kopija originalne konfiguracije Sudoku igre radi kasnije provjere. Poziva se funkcija *solveSudoku* kako bi se pokušao riješiti Sudoku. Ako rješenje postoji, test se smatra neuspješnim. Provjerava se da li je originalna konfiguracija Sudoku igre ostala nepromijenjena nakon pokušaja rješavanja. Ako se tabela promijenila (što nije očekivano u ovom slučaju), test se smatra neuspješnim.

5.7 testValidateSudokuGood

```
bool SudokuTest::testValidateSudokuGood();
```

Test ima tri testna slučaja koji provjeravaju ispravnost funkcije za validaciju Sudoku igre.

Test slučaj 1 (test1): Validna Sudoku tabla. Očekivano je da funkcija *validateSudoku* vrati *true*.

Test slučaj 2 (test2): Nevalidna Sudoku tabla sa duplikatom u posljednjem redu. Očekivano je da funkcija *validateSudoku* vrati *false*.

Test slučaj 3 (test3): Nevalidna Sudoku tabla sa duplikatom u posljednjoj koloni. Očekivano je da funkcija *validateSudoku* vrati *false*.

Svaki od testova se izvršava pozivom funkcije *validateSudoku* nad odgovarajućom instancom Sudoku igre, a rezultati se porede sa očekivanim vrijednostima. Na kraju, rezultat testa (uspješno ili neuspješno) se ispisuje u konzoli. Ako su svi testovi prošli, test je uspješan; u suprotnom, test nije uspješan.

5.8 testValidateSudokuBad

```
bool SudokuTest::testValidateSudokuBad();
```

Test ima tri testna slučaja koji provjeravaju neispravnosti u funkciji za validaciju Sudoku igre.

Test slučaj 1 (errors1): Dva ista broja u istom redu. Očekivano je da funkcija *validateSudoku* pronađe 1 grešku.

Test slučaj 2 (errors2): Dva ista broja u istoj koloni i istom redu. Očekivano je da funkcija *validateSudoku* pronađe 3 greške.

Test slučaj 3 (errors3): Jedan broj u istom redu i koloni kao i drugi broj. Očekivano je da funkcija *validateSudoku* pronađe 2 greške.

Svaki od testova se izvršava pozivom funkcije *validateSudoku* nad odgovarajućom instancom Sudoku igre, a broj pronađenih grešaka se upoređuje sa očekivanim vrijednostima. Na kraju, rezultat testa (uspješno ili neuspješno) se ispisuje u konzoli. Ako su svi testovi prošli, test je uspješan; u suprotnom, test nije uspješan.

5.9 testValidateSudokuEmpty

```
bool SudokuTest::testValidateSudokuEmpty();
```

Test provjerava funkcionalnost funkcije za validaciju Sudoku igre kada je tabla prazna.

Instanca Sudoku igre je inicijalizovana sa praznom matricom (svi elementi su 0). Očekuje se da funkcija *validateSudoku* vrati *false*, jer prazna tabla (sa svim 0 vrijednostima) ne zadovoljava pravila Sudoku igre.

5.10 testGenerateSudoku

```
bool SudokuTest::testGenerateSudoku();
```

Test provjerava funkcionalnost generisanja i validacije Sudoku igre. Generiše se instanca Sudoku igre (tabla) i popuni tabla koristeći funkciju *fillGrid*. Popunjena Sudoku tabla se zatim validira korišćenjem funkcije *validateSudoku*. Očekuje se da funkcija vrati *true*, što znači da je generisana tabla ispravna prema pravilima Sudoku igre.

5.11 testGenerateSudokuWithLimit

```
bool SudokuTest::testGenerateSudokuWithLimit();
```

Test provjerava funkcionalnost generisanja Sudoku igre sa ograničenim brojem mogućih brojeva po podmatrici 3x3. Generiše se instanca Sudoku igre (tabla) koristeći funkciju *generateSudoku*. Generisana Sudoku tabla se zatim validira korišćenjem funkcije *validateSudoku*. Očekuje se da funkcija vrati *true*, što znači da je generisana tabla ispravna prema pravilima Sudoku igre. Takođe se vrši dodatna provjera broja nepraznih (različitih od 0) ćelija u 3x3 podmatricama. Očekuje se da zbir broja nepraznih ćelija u cijeloj matrici bude manji od 54, što znači da se generišu table sa različitim nivoom težine uklanjanja brojeva.

5.12 testSetUpGood

```
bool SudokuTest::testSetUpGood();
```

Test provjerava funkcionalnost postavljanja (set-up) Sudoku igre koristeći funkciju *checkSetUp*. Kreira se instanca Sudoku igre (tabla) koristeći konstruktor i postavlja se matrica unaprijed definisanih vrijednosti. Zatim se poziva funkcija *solveSudoku* kako bi se popunila tabla i riješila Sudoku matrica. Poziva se funkcija *checkSetUp* sa originalnom matricom (prije rješavanja) kao argumentom. Očekuje se da funkcija vrati *true*, što znači da je postavljanje (set-up) Sudoku table ispravno.

5.13 testSetUpBad

```
bool SudokuTest::testSetUpBad();
```

Test provjerava funkcionalnost postavljanja (set-up) Sudoku igre koristeći funkciju *checkSetUp* kada se neispravna matrica riješi. Kreira se instanca Sudoku igre (tabla) koristeći konstruktor i postavlja se matrica unaprijed definisanih vrijednosti. Zatim se poziva funkcija *solveSudoku* kako bi se popunila tabla i riješila. Poziva se funkcija *checkSetUp* sa novom matricom (nakon rješavanja) koja sadrži jednu promijenjenu vrijednost u odnosu na originalnu matricu. Očekuje se da funkcija vrati *false*, što znači da je prepoznata neispravna postavka (set-up) Sudoku table.

5.14 runAllTests

```
void SudokuTest::runAllTests();
```

Metoda objedinjuje izvršavanje svih testova napisanih u klasi *SudokuTest*. Pozivanje ove metode izvršava sve testove i ispisuje rezultate. Ako svi testovi prođu, to ukazuje na to da je implementacija Sudoku igre ispravna. U suprotnom, ispisuje se poruka o neuspjelim testovima.

6. Zaključak

Napravljen je jedan koncept za realizaciju datog problema. Napravljen je i verifikovan sekvencijalni model koji obuhvata problem opisan u zadatku.

Verifikacija sekvencijalnog programa urađena je kroz pravljenje test funkcija. Time je potvrđena funkcionalnost svih metoda i funkcionalnosti programa u cjelini. Testovi su verifikacija da će program raditi ukoliko se korisnik ponaša u skladu sa pravilima igre Sudoku. Testovima nisu pokriveni slučajevi kada je fajl prazan ili kada korisnik prilikom unosa Sudoku matrice ili prilikom njenog rješavanja unese u matricu unutar datoteke nešto što nije broj, ili nešto što nije traženi format.

Program se ponaša u skladu sa očekivanjima i sposoban je da riješi sve Sudoku zagonetke, kao i da generiše rješive Sudoku zagonetke. U cjelini, program je ispunio sve funkcionalne zahtjeve navedene u specifikaciji. Naravno, postoji još prostora za poboljšavanje rada programa. Promjena algoritma rješavanja mogla bi da ubrza izvršavanje programa.