UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD
Softversko inženjerstvo i informacione tehnologije

PROJEKTNI ZADATAK

Kandidat: **Marina Ivanović**
Broj indeksa: **SV6-2022**

Predmet: **Objektno orijentisano programiranje 2**
Tema rada: **Sudoku**

Novi Sad, decembar, 2023.

SADRŽAJ

SPISAK SLIKA

1. Uvod

1.1 Sudoku – pravila igre

Sudoku je logička zagonetka u obiku kvadratne mreže. U najvećem broju slučajeva, postavka zagonetke se nalazi u matrici 9x9, preciznije rečeno, sačinjena je od 9 podmatrica 3x3. Cilj igre je da se dopune prazna polja tako da budu ispoštovana tri pravila:

1. Dva ista broja ne smeju da se nađu u istom redu
2. Dva ista broja ne smeju da se nađu u istoj koloni
3. Dva ista broja ne smeju da se nađu u istoj podmatrici, 3x3

Na Slika 1 se vidi klasična postavka sudoku igre.

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

Slika 1 Klasična sudoku igra

Veliki broj sudoku zagonetki može da sadrži više od jednog rešenja, a postoje i slučajevi kada je zagonetka nerešiva. Cilj moje implementacije nije nalaženje svih mogućih rešenja, već efikasno i kratko vreme rešavanja.

1.2 Zadatak

Glavni zadatak je omogućiti korisniku rešavanje sudoku zagonetke, kao i mogućnost samog programa da je reši. Korišćenjem i isprobavanjem algoritama potrebno je naći optimalan način rešavanje zagonetke, gde sam se ja opredelila za backtracking algoritam (algoritam vraćanja nazad).

Učitavanje sudoku zagonetke (rešene ili neupotpunjene) kao i čuvanje, a i omogućavanje korisniku da unese naziv fajla u koji bi hteo da upiše ili sačuva stanje table, praćenje stanja table uz praćenje pravilnih i nepravilnih korisničkih unosa (kao i pravilnih i nepravilnih unosa samog programa), a i testiranje su neke od bitnijih funkcionalnosti potrebnih za adekvatno rešavanje problema, o kojima detaljnije u nastavku.

2. Rad U/I sistema

Predviđeno je da korisnik može da bira fajl iz kog želi da učita već postojeću igru ili u koji fajl bi želeo da sačuva stanje table. Pored tih opcija, korisnik može da omogući programu da reši zagonetku, ili da generiše novu, rešivu, zakonetku.

Sve ove, a i dodatne funkcionalnosti sam ostvarila uz Game modul. Klasa Game upravlja korisničkim interakcijama i služi kao glavni posrednik za interakciju sa Sudoku slagalicama, obrađuje ulaze, prikazuje informacije i koordinira akcije između korisnika i komponenti za rešavanje slagalice.

Posebni modul koji omogućava rad sa fajlovima – čitanje iz fajla i upisivanje u fajl, je modul FileHandler. U njemu se nalaze i potrebne provere, da bi se sa sigurnošću radilo sa fajlovima.

2.1 Struktura datoteke

Struktura ulazne i izlazne datoteke se u mojoj implementaciji ne razlikuju. U svakoj datoteci, rešenoj ili nerešenoj se nalazi matrica 9x9, gde je svaki red odvojen novim redom, a svaki član u redu razmakom. Ako je u pitanju nerešena zagonetka, upotrebljen je znak 0 za mesta gde nije popunjeno polje.

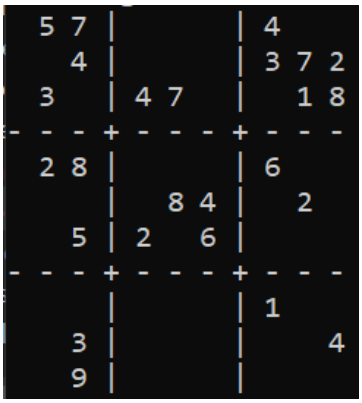
```
0 0 0 0 4 5 8 0 7
4 0 9 3 8 0 1 2 0
0 1 8 6 9 0 3 4 0
0 0 0 2 0 3 0 0 4
3 0 2 0 5 0 0 0 1
0 0 0 7 0 8 5 3 2
8 0 0 5 7 0 0 0 0
0 0 7 0 0 1 0 0 9
5 0 0 9 0 6 7 0 8
```

Slika 2 Primer nerešene zagonetke

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 0 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 0 1 5 3 7 2 8 4
2 8 7 4 1 9 0 3 5
3 4 5 2 8 6 1 7 0
```

Slika 3 Primer rešene zagonetke

Kada je u pitanju prikaz zagonetke u konzoli (prikaz pri izvršavanju koda), svaka kolona je odvojena znakom |, a red znakom -, dok se presek kolona i reda označava sa znakom +. Znak 0 se menja praznim poljem, radi bolje preglednosti.



Slika 4 Primer prikaza nerešene zagonetke

2.2 Argumenti komandne linije

Da bi se izbeglo dopuštanje korisnika da unosi naziv fajla u koji ili iz kog želi da učita zagonetku, nazive dozvoljenih fajlova (unsolved.txt i solved.txt) smo definisali kao argumente komandne linije. Pri pokretanju našeg programa, oni se prosleđuju u *argv* parametru, iz koga ih učitavamo na sledeći način:

```
if (argc < 3) {
        std::cerr << "Usage: " << argv[0] << " <input_file> <output_file>" << std::endl;
        return 1;
    }
    std::string fileInput = argv[1];
    std::string fileOutput = argv[2];
```

Bitna je provera da li ovih argumenata ima najmanje dva, jer u suprotnom naš program ne bi radio. Ove argumente dalje prosleđujem kao parametre konstruktoru Game objekta, o kome malo više kasnije.

3. Algoritami rešavanja i generisanja zagonetke

Rešavanje i nalaženje adekvatnog algoritma za rešavanje sudoku zagonetke može da bude zahtevan posao, ali nakon razmatranja mnogih mogućnosti odlučila sam se za Backtracking algoritam.

3.1 Korišćenje Backtracking algoritma za rešavanje sudoku slagalice

3.1.1 Opis Backtracking algoritma:

Backtracking algoritam je tehnika pretrage koja se koristi za sistematsko ispitivanje svih mogućih rešenja kako bi se pronašlo tačno rešenje za određeni problem. Ova tehnika radi rekurzivno, sistematski istražujući sve mogućnosti, vraćajući se unazad (backtracking) kada se dođe do situacije koja ne vodi ka ispravnom rešenju.

3.1.2 Primena Backtracking algoritma za rešavanje sudoku slagalice:

U kontekstu rešavanja Sudoku slagalice, backtracking algoritam se koristi za pronalaženje ispravnih brojeva koji mogu biti postavljeni na prazna mesta u tabli tako da zadovoljavaju pravila Sudoku igre - svaki broj mora biti jedinstven u svakom redu, koloni i podmatrici 3x3.

Algoritam sistematski prolazi kroz prazna mesta na tabli i pokušava da postavi brojeve od 1 do 9 na ta mesta. Pre postavljanja broja, algoritam proverava da li bi postavljanje određenog broja na datu poziciju bilo ispravno s obzirom na pravila Sudoku slagalice. Ukoliko se pravilo krši, algoritam se rekurzivno vraća unazad (backtrack), pokušavajući druge kombinacije brojeva, sve dok ne pronađe ispravno rešenje ili uoči da dato mesto ne može biti popunjeno ispravnim brojem.

3.1.3 Prednosti korišćenja Backtracking algoritma za sudoku:

Potpunost Rešenja: Backtracking algoritam garantuje pronalaženje rešenja za Sudoku slagalicu ako rešenje postoji.

Efikasnost: Iako može biti zahtevan po pitanju računarskih resursa za složenije probleme, backtracking algoritam je efikasan za većinu standardnih Sudoku slagalica.

Korišćenjem backtracking algoritma, implementacija rešavanja Sudoku slagalice postaje efikasna i osigurava tačnost pronalaska rešenja, pridržavajući se pravila igre.

3.2 Generisanje nove Sudoku zagonetke

Algoritam generisanja nove sudoku zagonetke se oslanja na pseudoslučajne vrednosti dobijene uz pomoć seed-a, koji nam predstavlja početno stanje generatora slučajnih brojeva. Da bismo obezbedili nepredvidivost programa, koristimo module *cstdlib* i *ctime* i njihove funkcije – *srand* i *stime*. *Stime* nam omogućava da dobijemo seed (početno stanje generatora slučajnih brojeva), koji nam zapravo predstavlja broj sekundi od početka epohe. *Srand* je krajnji činilac da bi dobili pseudoslučajan broj – zajedno sa *stime* dobijamo seed.

Pošto smo utvrdili kako možemo da dođemo do pseudoslučajne vrednosti koja nam omogućava popunjavanje sudoku zagonetke na nepredvidiv način, možemo da se vratimo na sam algoritam. Počinjemo popunjavanje prazne sudoku table (sva polja inicijalizovana na 0, tj. prazan član) popunjavanjem glavne dijagonale podmatrica matrice. Ovakvim popunjavanjem smanjujemo šansu kršenja pravila, jer su gornje leva, središnja, i donje desna podmatrica skroz nezavisne jedna od druge, tako da možemo samo da se fokusiramo na popunjavanje podmatrice nasumično brojevima 1-9. Sledeći korak u algoritmu je popunjavanje ostalih podmatrica, gde nam je jedini dodatni problem potreba proveravanja da li trenutno polje sadrži nedozvoljen broj u tom redu ili koloni, dok nam je do sada bilo bitno samo da li će se naći u istoj podmatrici.

Poslednji korak algoritma je glavna postavka, tj. brisanje brojeva iz table sve dok ne dostignemo željeni broj obrisanih brojeva iz bloka (najmanje 3 obrisana). Ponovo ćemo koristiti seed za pseudonasumičnu vrednost reda i kolone iz koje hoćemo da obrišemo broj. Na samom kraju želimo da proverimo da li je naša generisana sudoku zagonetka rešiva.

4. Opis rešenja

4.1 Moduli i osnovne metode

4.1.1 Modul glavnog programa (Source)

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

Glavna funkcija programa iz koje mogu sa se pokrenu test primeri ili sam program. Kao argumenti komandne linije prosleđeni su mogući fajlovi u koje je moguće upisivati i čitati sudoku zagonetke.

4.1.2 Modul za čuvanje globalnih konstanti (Constants)

Modul u kome se nalaze konstante N = 9 (broj polja u svakom redu/koloni), B = 3 (broj polja u svakom redu/koloni podmatrice) i EMPTY = 0 (koji koristimo pri upisivanju u fajl, tj. koji nam služi za popunjavanje praznih mesta).

4.1.3 Modul sudoku (Sudoku)

Modul koji čuva stanje sudoku tabele, kao i statistiku.

4.1.3.1 Parametri Sudoku modula

`std::vector<std::vector<int>> matrix;` - Matrica za čuvanje sudoku table

`int correctInput;` - Brojač ispravnih poteza

`int incorrectInput;` - Brojač neispravnih poteza

`int gamesPlayed;` - Redni broj partije

4.1.3.2 Geteri i seteri modula Sudoku

```
const std::vector<std::vector<int>>& getMatrix() const;
```

Funkcija koja vraća matricu po referenci.

```
void setValue(int row, int col, int value);
```

Funkcija koja postavlja vrednost polja u matrici na određen broj.

```
int getCorrectInput() const;
void setCorrectInput(int correct);
```

Geter i seter za broj dobrih poteza.

```
int getIncorrectInput() const;
void setIncorrectInput(int mistakes);
```

Geter i seter za broj loših poteza.

```
int getGamesPlayed() const;
void setGamesPlayed();
```

Geter i seter za broj odigranih igara.

4.1.3.3 Funkcija za inicijalizovanje statistike (initStats())

```
void initStats();
```

Funkcija koja postavlja vrednosti brojeva dobrih i loših poteza na nula, za početak nove igre.

4.1.3.4 Funkcije prikaza

```
void displayBoard();
void displayStats();
```

Funkcije koje su zadužene za prikaz tabele i statistike u konzoli.

4.2 Modul i metode osnovne logike programa - Computer

4.2.1 Parametri modula Computer

```
Sudoku* sudoku;
```

Modul Computer sadrži pointer ka već kreiranom sudoku objektu.

4.2.2 Metode rešavanja sudoku zagonetke

```
const bool solveSudoku()
```

Glavna metoda za rešavanje sudoku zagonetke, poziva metodu solveSudoku(int row, int col), gde prosleđuje nule kao parametre.

Povratna vrednost:

- True – Uspešno rešen sudoku/ima rešenje
- False – Neuspešno rešen sudoku/nema rešenje

4.2.3 Metode generisanja zagonetke

```
void generateSudoku();
```

Metoda (algoritam) za generisanje nove, pseudoslučajne table.

4.2.3.1 Funkcija popunjavanja table

```
void fillSudoku();
```

Poziva neophodne funkcije za popunjavanje tabele.

4.2.3.2 Funkcije popunjavanja podmatrica

```
void fillDiagonalBoxes();
void fillBox(int startRow, int startCol);
```

Pomoćne funkcije za popunjavanje sudoku table. Shodno algoritmu popunjavanja, prvo je potrebno popuniti dijagonalne podmatrice (fillDiagonalBoxes) uz pomoć fillBox funkcije, čiji su parametri:

- startRow – početni red podmatrice
- startCol – početna kolona podmatrice

Nakon što smo popunili podmatrice na dijagonali, potrebno je popuniti i ostale, rekurzivnim pozivanjem:

```
void fillRemainingBoxes(int row, int col);
```

Parametri:

- row – početni red podmatrice
- col – početna kolona podmatrice

4.2.3.3 Funkcija za resetovanje sudoku table

```
void eraseSudoku();
```

Funkcija postavlja sva polja na vrednost konstante EMPTY = 0.

4.3 Modul validacije - Validation

Validation modul je sačinjen od statičkih metoda koje vrše različite vrste provere nad sudoku tablom.

4.3.1 Metoda glavne validacije tabele

```
static const int validateBoard(const Sudoku& sudoku);
```

Validira celu tablu, vršeći neophodne provere. Usput drži evidenciju o broju nevalidnih polja.

Parametri:

- Sudoku – konstantna referenca nad sudoku objektom, nad kojim se obavlja validacija

Povratna vrednost:

- Broj nevalidnih polja, tj. broj polja koja ne ispunjavaju glavna pravila sudoku igre.

4.3.2 Metoda provere mogućnosti igranja poteza

```
static const bool isSafe(const Sudoku& sudoku, int row, int col, int val);
```

Određuje da li je moguće odigrati određeni potez.

Parametri:

- sudoku – konstantna referenca nad sudoku objektom, nad kojim se obavlja validacija
- row – red u kome se nalazi vrednost koju želimo da odigramo
- col – kolona u kojoj se nalazi vrednost koju želimo da odigramo
- val – vrednost koju želimo da odigramo.

4.3.3 Metode za proveravanje reda, kolone, i bloka

```
static const bool checkRow(const Sudoku& sudoku, int row, int val);
static const bool checkColumn(const Sudoku& sudoku, int col, int val);
static const bool checkBlock(const Sudoku& sudoku, int row, int col, int val);
```

Metode koje proveravaju da li se određena vrednost već nalazi u redu/koloni/podmatrici, i da li je validan taj red/kolona/blok.

Povratne vrednosti:

- True – ispravan red/kolona/blok, tj. ne pojavljuju se iste vrednosti više puta,
- False – neispravan red/kolona/blo, tj. iste vrednosti se pojavljuju više puta.

4.3.4 Metoda provere polja (isEmpty())

```
const bool isEmpty(const Sudoku& sudoku, int row, int col);
```

Na osnovu datog reda i kolone, proverava da li je polje u tabeli popunjeno.

Povratne vrednosti:

- True – polje je prazno.
- False – polje je popunjeno.

4.3.5 Metode za prebrojavanje praznih mesta u tabli

```
static const int numOfEmptySpacesBlock(const Sudoku& sudoku, int row, int col);
static const int numOfEmptySpaces(const Sudoku& sudoku);
static const bool emptySpaces(const Sudoku& sudoku, int minNumOfEmptySpaces);
static const bool isFilled(const Sudoku& sudoku);
```

NumOfEmptySpaces je funkcija koja poziva numOfEmptySpacesBlock za svaki blok u tabeli, da bi se utvrdilo da je izbačen dovoljan broj nula iz tabele.

IsFilled je funkcija koja poziva emptySpaces, koja prebrojava broj praznih mesta u tabeli, i ako je taj broj 0, onda je povratna vrednost isFilled funkcije true, u suprotnom je false.

4.3.6 Metoda provere dobrog rešenja u odnosu na datu bazu (postavku)

```
static const bool completedCorrectPuzzle(const Sudoku& baseSudoku, const Sudoku& solvedSudoku);
```

Metoda prihvata dve reference sudoku objekta i poredi da li je dato rešenje (solvedSudoku) baš rešenje date baze (postavke – baseSudoku)

Povratne vrednosti:

- True – u slučaju da to jeste rešenje te postavke
- False – u slučaju da nije rešenje te postavke

4.4 Modul za rukovanje fajlovima - FileHandler

Modul koji sadrži dve statičke funkcije – jednu za učitavanje sudoku zagonetke iz fajla, i druge za upisivanje u fajl.

```
static bool loadSudokuFromFile(const std::string& filename, Sudoku& sudoku);
```

Parametri:

- filename – izabrani argument komandne linije
- sudoku – referenca na objekat klase Sudoku

Povratna vrednost:

- True – ako je učitana tabela bez problema
- False – ako je nastao problem pri učitavanju

```
static bool saveSudokuToFile(const std::string& filename, const
Sudoku& sudoku);
```

Parametri:

- filename – izabrani argument komandne linije
- sudoku – referenca na objekat klase Sudoku

Povratna vrednost:

- True – ako je upisana tabela bez problema
- False – ako je nastao problem pri upisivanju

4.5 Modul za kontrolisanje toka igre – Game

4.5.1 Parametri modula Game

```
Sudoku* sudoku;
std::string fileInput;
std::string fileOutput;
```

Pointer na sudoku objekat i dva naziva fajla, koja smo prosledili modulu iz glavnog dela programa kroz argumente komandne linije.

4.5.2 Metode za učitavanje i pisanje u fajl

```
bool saveSudoku();
bool loadGame();
```

Ove metode su zadužene za pozivanje metoda iz FileHandler modula

4.5.3 Metoda početka igre

```
int startGame();
```

Metoda koju pozivamo iz glavnog dela programa, čiji je zadatak da pripremi program za igru.

Povratne vrednosti:

- 1 – ukoliko korisnik želi da nastavi igru
- 0 – ukoliko korisnik želi da završi sa igrom

4.5.4 Metode koje pozivaju rešavanje sudoku table

```
void startSolving(Computer& computer);
void computerSolving(Computer& computer);
void playerSolving();
```

Metodama startSolving i computerSolving prosleđujemo referencu na objekat Computer, koji je zadužen za rešavanje zagonetke. StartSolving nam dodeljuje mogućnost izbora između učitavanja našeg rešenja (playerSolving) ili generisanje rešenja kompjutera (computerSolving)

4.5.5 Metoda za kalkulisanje grešaka

```
void calculateStats();
```

Nakon završetka svake igre, potrebno je prikazati broj grešaka (do kog dolazimo krajnjom validacijom table). CalculateStats je zadužena za pozivanje te metode, kao i prikaz table i statistike.

5. Testiranje

Testiranje je izvršeno implementiranjem Testing modula.

5.1 Testovi

5.1.1 Test All

testAll() je funkcija koja pokreće sve testove.

5.1.2 testGeneratePuzzle

Ovaj test nam proverava da li je generisana tabla sigurno validna, koristeći funkcije validacije table.

5.1.3 testSolvePuzzle

Testovi koji proveravaju da li algoritam dobro rešava sudoku zagonetku.

5.1.4 testSolutionForTheBaseGame

Test koji nam testira funkciju completedCorrectPuzzle(), tačnije testira da li je dato rešenje baš rešenje date postavke.

5.1.5 Testovi validacije

Testovi testBoardValidation(), testRowValidation(), testColumnValidation(), testBlockValidation() testiraju istoimene funkcije iz Validation modula.

5.1.6 Testovi rukovanja sa fajlovima

Testovi testLoadFile() i testSaveFile() testiraju funkcije čuvanja i učitavanja iz FileHandler modula u nepredvidivim slučajevima (neispravna datoteka, nepostojeća datoteka...)

6. Zaključak

Tokom izrade projekta naišla sam na brojne probleme. Logika kod računanja nevalidnih polja mi je oduzela puno vremena, ali je takođe i koštala moj algoritam efikasnosti. U prvoj strukturi projekta koju sam isplanirala u planu je bilo da neke klase, poput Validation i Computer nasleđuju klasu Game, ali sam daljom izradom projekta shvatila da mi je to dodatno zakomplikovalo strukturu, i da su neki projekti bolji bez nasleđivanja, ako ono ne može na logičan način da se implementira.