ELEKTROTEHNIČKI FAKULTET
UNIVERZITET U ISTOČNOM SARAJEVU

SEMINARSKI RAD

PROJEKAT: SudokuSolver

Studije: I ciklus
Odsjek: Računarstvo i informatika
Predmet: Paralelni računarski sistemi

Mentor:
dr Nikola Davidović

Studenti:
Danijela Milanović 2003,
Stefan Jokić 2002

Istočno Sarajevo, Jun 2022

1. Uvod

Dokument detaljno predstavlja rad i izradu projekta iz predmeta „Projektovanje informacionog softvera“ na temu SudokuSolver.

Posao je proveden kroz korake, od analize i izbora tehnologije, istraživanja zadate teme, implementacija i testiranja aplikacije.

Prvi korak kojim je započeta izrada aplikacije „SudokuSolver“ je analiziranje tehnologija u kojima je moguće ubrzati proces izvršavanja metodom višenitnosti. Izabran je Java programski jezik. Za vrstu izabrana je mobilna aplikacija, dok za IDE AndroidStudio.

Zatim se pristupilo razradi ideje i na koje sve načine je moguće ubrzati performanse aplikacije. Kada je izvršena izrada ideje i algoritma pristupilo se izradi aplikacije, njenom GUI – u i implementaciji logičke strane. Na kraju je izvršena integracija logičkog dijela u aplikaciju i njeno testiranje.

Alati koji su korišteni za izradu ovog projekta su:

- AndroidStudio – okruženje za izradu Android aplikacija,
- Eclipse IDE – korišten za izradu logičkog dijela aplikacije,

2. Tehnologije za izradu aplikacije

U nastavku dokumenta predstavljene su tehnologije i alati koji su korišteni za izradu SudokuSolver a, kao i na koji način podržavaju programiranje metodom višenitnosti.

2.1 JAVA programski jezik

JAVA je objektno-orijentisani programski jezik, koji je razvila kompanija *Sun Microsystems* početkom 1990-ih godina.

Za razliku od dotadašnjih programskih jezika izbačeni su koncepti modula i uvedeni paketi kakve danas znamo, koji se oslanjaju na Fajl sistem, uveli su formalno koncept klasa iz objektno-orijentisane paradigme. Osim toga, jezik ima sintaksu sličnu jezicima C i C++, ali je mnogo stroži pri prevođenju, dizajniran tako da bude nezavisan od platforme, i sa pojednostavljenim upravljanjem memorijom. Pretpostavlja se da je ovo urađeno zbog popularnosti jezika C, ali i zbog jednostavnosti nekih struktura. Prva verzija je zvanično objavljena 1995. godine.

2.1.1 Istorija JAVA programskog jezika

James Gosling i Patrick Naughton, grupa inžinjera kompanije *Sun*, su inicirali jezički projekat JAVA u junu 1991. Jezik je nazvan *Oak*, i bio je namenjen za programiranje kućnih elektronskih uređaja (televizora, video rekordera, ...). Ime projekta *Oak* je kasnije promijenjeno u JAVA, po brendu *Java Coffee*, koja je dobila ime po istoimenom ostrvu *Java*, tako da logo jezika *JAVA* prikazuje čašu vruće kafe. Postoji još jedna verzija porijekla imena jezika, koja se odnosi na aluziju na aparat za kafu kao primer kućnog uređaja, za koji je programski jezik prvobitno kreiran. James Gosling je JAVA dizajnirao sa istom sintaksom kao u C/C++ kako bi sistemski i aplikativni programeri lakše naučili jezik.

Sun Microsystems je prvu javnu implementaciju objavio kao JAVA 1.0 1996. godine. Obećavao je **Write Once, Run Anywhere (WORA)**, obezbjeđujući izvršavanje koda na popularnim platformama bez utroška vremena. Prilično bezbjedan i sa sigurnosnim podešavanjima, dozvoljavao je ograničenja pristupa mreži i datotekama. Glavni web pregledači ubrzo su ugradili mogućnost pokretanja JAVA applet a na web stranicama i JAVA je brzo postala popularna. Artur Van Hof je ponovo napisao prevodilac za JAVA 1.0 u Javi kako bi se strogo pridržavao specifikacije jezika JAVA 1.0. Sa pojavom JAVA 2 (prvobitno objavljen kao J2SE 1.2 u decembru 1998. - 1999), nove verzije su imale višestruke konfiguracije izgrađene za različite tipove platformi. J2EE je uključivao tehnologije i API za poslovne aplikacije koje se obično izvršavaju u server okruženju, dok J2ME sadrži API optimizovan za mobilne aplikacije. Desktop verzija je preimenovana u J2SE. 2006. godine, u marketinške svrhe, Sun je preimenovao nove verzije J2 u JavaEE, JavaME i JavaSE.

Godine 1997. *Sun Microsystems* obratio se ISO/IEC JTC 1 uredu za standardizaciju, a kasnije *Ecma International* kako bi formalizovao Javu, ali se ubrzo povukao iz procesa. JAVA ostaje *de facto* standard, koji se kontroliše kroz procese JAVA zajednice. U jednom trenutku, *Sun* je većinu svojih JAVA implementacija učinio dostupnim bez naplate, uprkos

licenciranom statusu softvera. *Sun* je stvarao prihod od Jave prodajom licenci za specijalizovane proizvode kao što je Java Enterprise System.

Dana 13. novembra 2006, *Sun* je izdavao veći deo svoje Java virtuelne mašine (**JVM**) kao besplatni i open-source softver (FOSS), pod uslovima navedenim u GNU General Public Licence (GPL). 8. maja 2007., *Sun* je završio postupak, stavljajući sav svoj JVM osnovni kod dostupan kao free software/open source distribuciju, osim malog dela koda na koji *Sun* nije imao autorska prava.

Nakon što je kompanija Oracle Corporation preuzela Sun Microsystems u 2009–10, Oracle se opisao kao upravitelj Java tehnologije sa velikom posvećenošću podsticanju učešća i transparentnosti zajednice. To nije sprečilo Oracle da podnese tužbu protiv Google-a ubrzo nakon toga zbog upotrebe JAVA programskog jezika unutar **Android SDK-a**.

U januaru 2016. Oracle je objavio da će za Java run-time okruženja proizvedena iz JDK 9 prekinuti dodatak web pregledača.

2.1.2 Principi

U stvaranju JAVA jezika oslanjalo se na pet osnovnih ciljeva:

1. Mora biti jednostavan, objektno-orijentisan i prepoznatljiv.
2. Mora biti temeljan i siguran.
3. Mora biti arhitektonski neutralan i prenosiv.
4. Mora da se izvršava sa visokim performansama.
5. Mora biti interpretiran, praćen i dinamičan.

2.1.3 Karakteristike JAVE

Programski jezik JAVA treba da bude:

- **jednostavan** – da bude sistem u kome bi se lako programiralo, bez potrebe za komplikovanim uhodavanjem i koji koristi postojeći način razmišljanja. Sintaksa jezika Java je unapređena verzija sintakse C++
- **objektno-orijentisan** – objektno-orijentisano projektovanje predstavlja tehniku programiranja fokusiranu na podatke(objekte) i na interfejse ka tim objektima.
- **distribuiran** – JAVA poseduje iscrpnu biblioteku rutina za rad sa TCP/IP protokolima, kao što su HTTP i FTP. JAVA aplikacije mogu da pristupaju objektima preko mreže i preko URL-a, sa podjednakom lakoćom kao da pristupaju lokalnom sistemu datoteka.
- **robustan** – JAVA je namenjena za pisanje programa koji moraju biti pouzdani na mnogo načina. Ističe se u ranoj proveri mogućih problema, kasnijoj dinamičkoj provjeri(tokom izvršavanja) i eliminaciji situacija u kojima lako dolazi do pojave grešaka.
- **bezbjedan** – JAVA je namenjena korišćenju u mrežnim/distribuiranim okruženjima. Prema tome, mnogo je truda uloženo u bezbjednost. Omogućava konstrukciju sistema

koji su zaštićeni od virusa i zlonamerne modifikacije. Od samog početka, Java je projektovana da potpuno onemogući određene vrste napada, kao što su prekoračenje izvršnog steka, pristup memoriji izvan dela dodeljenog procesu, čitanje ili upisivanje datoteka bez dozvole...

- **neutralan** – kompajler stvara objektnu datoteku, čiji je format nezavisan od operativnog sistema na kome se pokreće. Kompajlirani kod se može izvršavati na mnogim procesorima, pod pretpostavkom prisustva izvršnog sistema Java. JAVA kompajler ostvaruje ovo tako što generiše bajtkod instrukcije, koje nemaju nikakve veze sa arhitekturom korišćenog računara, već se podjednako lako interpretiraju na svakoj mašini, a isto tako lako se prevode i u odgovarajući mašinski kod.
- **prenosiv** – ovde ne postoje aspekti koji su zavisni od implementacije. Veličine primitivnih tipova su fiksne, kao i njihovo ponašanje u aritmetici. Takođe, biblioteke koje su deo sistema definišu interfejse koji su prenosivi.
- **interpretiran** – isti JAVA bytecode se može izvršavati na svakom kompjuteru za koji postoji JAVA interpreter. Budući da je linkovanje postepen i lakši postupak, sam razvoj može biti brži
- **performantan** – mada su performanse prevedenog JAVA bytecode obično više nego dovoljne, postoje situacije kada su potrebne bolje performanse. Bajtkod može da bude preveden tokom izvršavanja u mašinski jezik
- **višenitan –** prednosti višenitne obrade su bolji interaktivni odzivi i ponašanje u realnom vremenu
- **dinamičan** – JAVA je projektovana tako da se prilagođava okruženju koje se stalno unapređuje. Biblioteke mogu slobodno da dodaju nove metode i polja, bez uticaja na klijente. U JAVA programskom jeziku je prilično jednostavno pronalaženje informacija prilikom izvršavanja programa.

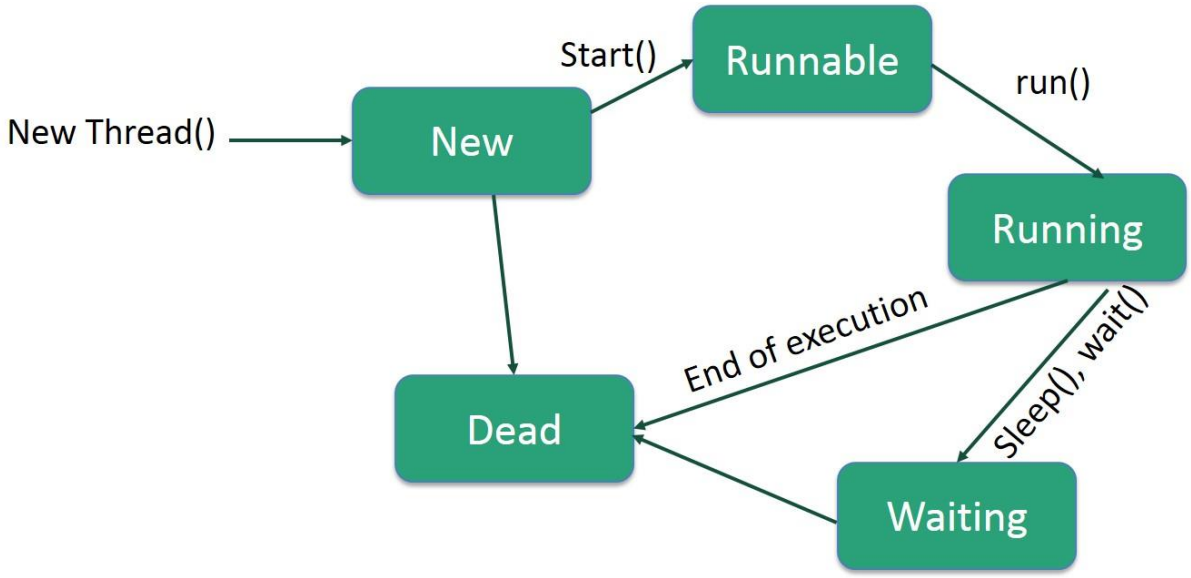
2.1.4 JAVA višenitnost

JAVA je višenitni programski jezik. Višenitni program sadrži dvije ili više niti koje se mogu izvoditi istovremeno i svaka nit može istovremeno raditi sa različitom svrhom, optimalno koristeći dostupne resurse.

Po definiciji, MultiTasking je kada više procesa dijele zajedničke resurse obrade kao što je CPU. Višenitnost proširuje ideju MultiTasking-a u aplikacije gdje možete podijeliti specifične operacije unutar jedne aplikacije u pojedinačne niti. Svaka od niti može raditi paralelno. Operativni sistem dijeli vrijeme obrade, ne samo između različitih aplikacija, već i između svake niti unutar aplikacije.

2.1.4.1 Životni ciklus niti

Nit prolazi kroz različite faze u svom životnom ciklusu. Na primjer, nit se rađa, pokreće, pokreće, a zatim umire. Sljedeći dijagram prikazuje kompletan životni ciklus niti.



Faze životnog ciklusa niti:

- **Novo** – Nova nit započinje svoj životni ciklus u novom stanju. Ostaje u ovom stanju sve dok program ne pokrene nit. Takođe se naziva **rođena nit** .
- **Runnable** – Nakon što se pokrene novorođena nit, nit postaje runnable. Smatra se da nit u ovom stanju izvršava svoj zadatak.
- **Waiting** – Ponekad nit prelazi u stanje čekanja dok nit čeka da druga nit izvrši zadatak. Nit prelazi nazad u stanje koje se može pokrenuti samo kada druga nit signalizira nit koja čeka da nastavi s izvršavanjem.
- **Timed Waiting** – Nit koja se može pokrenuti može ući u vremenski ograničeno stanje čekanja za određeni vremenski interval. Nit u ovom stanju prelazi nazad u stanje koje se može pokrenuti kada taj vremenski interval istekne ili kada se dogodi događaj na koji čeka.
- **Završena (mrtva)** – Nit koja se može pokrenuti ulazi u prekinuto stanje kada završi svoj zadatak ili je prekinuta na neki drugi način.

2.1.4.2 Implementacija višenitnosti u JAVA programskom jeziku

Ako je klasa predviđena da se izvršava kao nit, to možete postići implementacijom interfejsa **Runnable.**

Kao prvi korak, implementira se metoda run() koju pruža **Runnable** interfejs. Ova metoda pruža ulaznu tačku za nit i sva kompletna poslovna logika izvršava se unutar ove metode.

Kao drugi korak, instancirat ćete **Thread** objekt koristeći sljedeći konstruktor - Thread(Runnable threadObj, String threadName);

Jednom kada je objekt Thread kreiran, pokreće se pozivanjem **start()** metode, koja izvršava poziv run() metode.

Implementacija višenitnosti u JAVA programskom jeziku može se izvršiti i na sledeći način.

Klasa koja je predviđena da se izvršava kao nit može da naslijedi klasu Thread koja se koristi i u prethodnom načinu implementacije. Ostala implementacija se obavlja na sličan način kao u prethodnom primjeru sa modifikovanim drugim korakom.

SudokuSolver primjenjuje drugi način implementacije višenitnosti.

2.2 Android operativni sistem

Android je mobilni operativni sistem kompanije Google zasnovan na Linux jezgru, prvenstveno dizajniran za mobilne uređaje sa ekranom osjetljivim na dodir, kao što su pametni telefoni i tablet uređaji. Korisnički interfejs Androida je zasnovan na direktnoj manipulaciji objektima na ekranu, korišćenjem ulaza u vidu dodira koji odgovaraju pokretima u realnom svetu kao što su prevlačenje, pritiskanje, zumiranje kao i unos teksta pomoću virtuelne tastature. Kao dodatak uređajima osetljivim na dodir, Google je razvio i Android TV za televizore, Android Auto za automobile i Wear OS za ručne satove, svaki od njih sa prilagođenim korisničkim interfejsom. Varijante Android operativnog sistema se koriste i na igračkim konzolama, digitalnim kamerama, personalnim računarima i drugim elektronskim uređajima.

Android je razvila istoimena kompanija koju je kompanija Google kupila 2005. godine. Android je predstavljen 2007. godine zajedno sa osnivanjem udruženja Open Handset Alliance, OHA, konzorcijuma hardverskih, softverskih i telekomunikacionih kompanija posvećenih razvoju otvorenih standarda za mobilne uređaje. Prvi Android telefon je prodat u septembru 2008. godine i od tada je predstavljeno više izdanja ovog operativnog sistema, sa najnovijom verzijom 8.0 "Oreo", koja je predstavljena u avgustu 2017. godine. Android aplikacije se mogu preuzeti sa GooglePlay prodavnice, na kojoj zaključno sa februarom 2017. godine, ima preko 2,7 miliona aplikacija. Android je najprodavaniji operativni sistem na tablet računarima od 2013. godine, i pokreće se na većini pametnih telefona. Od maja 2017. godine, Android ima 2 milijarde aktivnih korisnika mesečno, i poseduje najveću bazu korisnika od svih operativnih sistema.

Izvorni kod Androida je objavljen od strane Googl-a i dostupan je pod licencom otvorenog koda, iako se većina Android uređaja isporučuje u kombinaciji softvera otvorenog koda i vlasničkih licenci, između ostalog i vlasničkog softvera neophodnog za pristup Google servisima. Android je popularan među tehnološkim kompanijama koje zahtevaju gotove, jeftine, prilagodljive i lake operativne sisteme za svoje visokotehnološke uređaje. Otvorenost platforme je ohrabrila ogromnu zajednicu programera i entuzijasta da koriste softver otvorenog koda kao osnovu za svoje zajedničke projekte, kao i da kreiraju zakrpe za starije uređaje, dodaju nove funkcionalnosti za naprednije korisnike. Velike razlike u hardveru koji se nalazi u Android uređajima dovodi do velikih kašnjenja u nadogradnji softvera, gde je obično potrebno da prođe i po nekoliko meseci između različitih verzija operativnog sistema i sigurnosnih zakrpa, pre nego što se isporuči korisnicima, ili se ponekad uopšte i ne isporuči.

Od novembra 2013. godine, trenutne verzije Android operativnog sistema sadrže jezgro bazirano na Linux jezgru verzije 3.x, dok je na verzijama starijim od Android 4.0 Ice Cream Sandwich jezgro bazirano na Linux jezgru verzije 2.6.x.Android Linux jezgro poseduje dublje izmene arhitekture Linuks jezgra u odnosu na tipičan razvojni ciklus Linux jezgra.

2.3 AndroidStudio

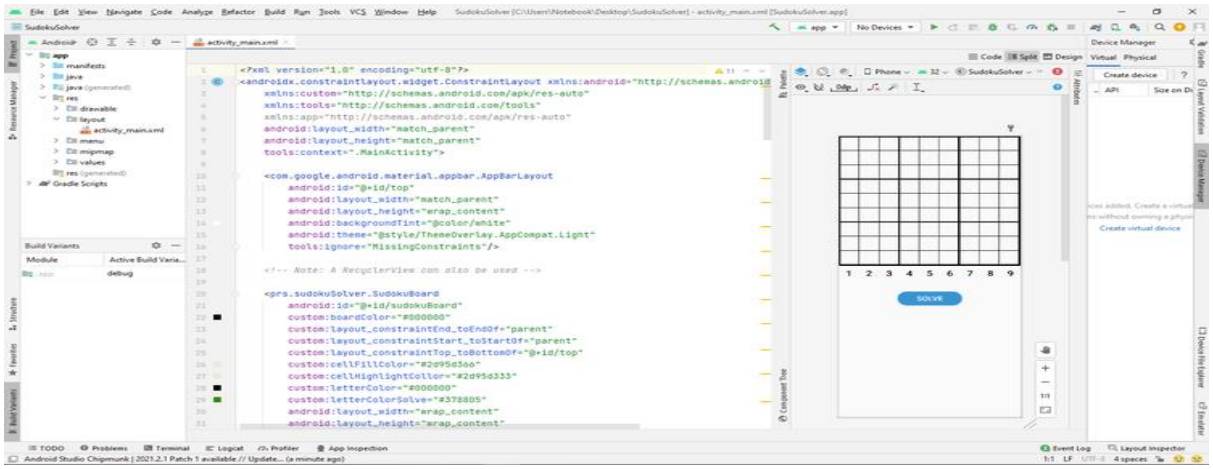
Android Studio je službeno integrisano razvojno okruženje (IDE) za *Googleov Android operativni sistem*, izgrađeno na JetBrains - ovom IntelliJ IDEA softveru i dizajnirano posebno za razvoj *Androida* . Dostupan je za preuzimanje na Windows , macOS i Linux operativnim sistemima. *Android Studio* je zamjena za Eclipse Android razvojne alate (E-ADT) kao primarni IDE za razvoj izvornih Android aplikacija.

Android Studio je najavljen 16. maja 2013. na Google I/O konferenciji. Bio je u fazi pregleda ranog pristupa počevši od verzije 0.1 u maju 2013., a zatim je ušao u beta fazu počevši od verzije 0.8 koja je objavljena u junu 2014. Prva stabilna verzija objavljena je u decembru 2014., počevši od verzije 1.0.

Dana 7. maja 2019, *Kotlin* je zamijenio Javu kao Googleov preferirani jezik za razvoj Android aplikacija. **JAVA** je još uvijek podržana, kao i C++.

Nakon što se aplikacija kompajlira pomoću *Android Studia*, može se objaviti na Google Play prodavnici . Aplikacija mora biti u skladu sa *politikom sadržaja za programere Google Play trgovine*.

Android Studio ima ugrađen Android virtuelni uređaj (emulator) za pokretanje i otklanjanje grešaka u aplikacijama. Prikaz AndroidStudia predstavljen je na sledećoj slici.



Prilikom izrade SudokuSolvera nije korišten ugrađen Android emulator. Rađeno je sa mogućnosti debugovanja na fizičkom uređaju.

3. Analiza problema

3.1 Sudoku

Sudoku je vrsta matematičke zagonetke čije je rješavanje temeljeno na logici. Sastoji se od jednog velikog kvadratnog polja, podijeljenog na 81 manjih kvadratića. Unutar velikog kvadrata, označeno je 9 blokova velikih 3x3 polja. Sudoku može biti različite težene u zavisnosti koliko brojeva je dato u postavci i na kojim pozicijama. Postoji mnogo strategija za riješavanje sudooku table. Sudoku igra treba da ima jedinstveno rješenje. Međutim, može biti slučajeva da Sudoku igra ima više rešenja ili da nema nijedno. Obično igra sa višestrukim rješenje se smatra težim od onog koje ima jedinstveno rješenje. Postoje više vrsta sudokua kao što su killer sudoku, alfabetni sudoku i šahovski sudoku. U projektu sudokuSolver korišten je klasičan sudoku.

3.1.1 Pravila igre

Cilj igre je ispuniti sva polja brojevima od 1 do 9, s time da se svaki broj smije pojaviti točno 9 puta. Problematika je u tome što se jedan broj smije pojaviti samo jednom u svakom redu, svakoj koloni i svakom bloku od 3x3 polja. Na početku igre, otkriveni su određeni brojevi, a onaj koji riješava mora otkriti gdje se nalaze svi ostali brojevi i kako su raspoređeni. Sudoku može imati više rješenja, ako je na početku otkriveno malo brojeva. Zato treba paziti da se greška ne načini na početku rješavanja. Jedini način rješavanja sudokua je metoda eliminacije, a tu se koristi svojstvom da se jedan broj smije pojaviti samo jednom u svakom redu, koloni i bloku. Rješavanje postaje lakše ukoliko se otkrije više brojeva.

3.1.2 Istorija

Sudoku je nastao krajem 16. veka. La France je razvila osnovni oblik moderne sudoku mreže 9x9 sa 9 unutrašnjih blokova dimenzija 3x3, a kasnije se verovalo da je moderni Sudoku anonimno dizajnirao Hauard Garns 1979. Međutim, ovo Igra je postala svetski fenomen tek kada ju je Nikoli predstavio Japanu 1984. godine. Sudoku je zaštitni znak u Japanu i slagalica se obično naziva Number Place, ili Ukratko NumPla.

3.2 Sudoku solver

Sudoku solver je termin koji se koristi za aplikacije koje na osnovu unaprijed zadatih virjednosti sudoku table daju njeno moguće rješenje. Postoji nekoliko algoritama koji će rešiti zagonetke 9×9 (N=9) u delićima sekunde ali vremenska složenost eksponencijalno raste kako se N povećava. Jedan od najčešćih algoritama koji se koristi za automatsko riješavanje sudoku table je Brute-Force algoritam.

4. Logički dio aplikacije

4.1 Postavka table i algoritma

U izradi aplikacije SudokuSolver korišten je Backtracking algoritam koji je tip Brute-Force algoritma. Ovaj pristup nudi dvije glavne prednosti a to su:

- Može potvrditi ako nema riješenja
- Može prikazati listu svih mogućih riješenja

Aplikacija je rađena kao višenitna. Da bi se proces mogao standardizovati, korišten je primjer iste table za sva testiranja. Zadata tabla je izabrana jer je ocijenjena kao „veoma teška“.

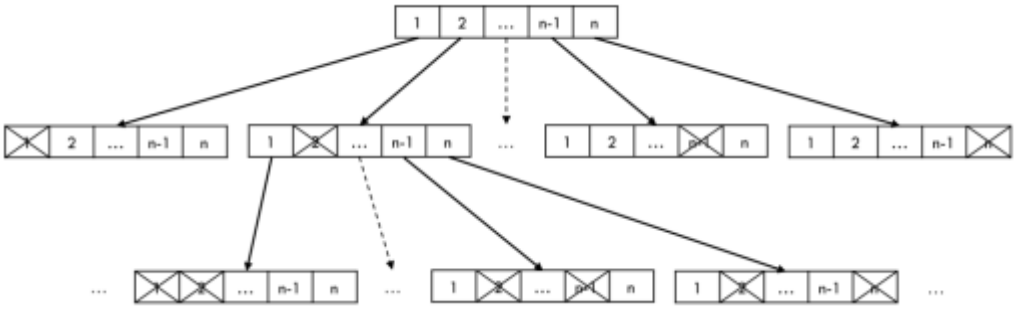
1								2
	9		4				5	
		6				7		
	5		9		3			
				7				
			8	5			4	
7						6		
	3				9		8	
		2						

Ispitivanje se sastojalo iz ponovnog riješavanja zadate table korištenjem istog algoritma sa različitim brojem niti.

Algoritam koristi rekurzivni poziv na pristupu test-and-backtrack odnosno provjeri-i-vrati se. Sve ćelije u tabli će biti posjećene.

4.2 Brute-Force algoritam

Pretraga Brute-Force algoritmom se sastoji iz toga da on napravi stablo sa maksimalnom visinom N^2 što je u našem slučaju 81, iz razloga što mora da se posjeti svaka ćelija. Apstraktna struktura stabla je data slikom ispod gdje svaki nivo predstavlja svaku ćeliju.



Međutim, ako pogledamo igru red po red, za svaki red imamo N mogućih vrijednosti za prvu ćeliju, i $N - 1$ mogućih vrijednost za drugu ćeliju, $N - 2$ mogućih vrijednosti za treću ćeliju, i 1 vrijednost za poslednju ćeliju. Na osnovu toga, računska složenost za popunjavanje jednog reda je $O(N \times (N-1) \times \ldots \times 1) = O(N!)$. Kako postoji N redova, ukupna složenost proračuna je $O((N!)^2)$.

4.3 Koraci pri rješavanju table

Koraci pri riješavanju table su sledeći:

1. Ukoliko se naiđe na ćeliju koja nije prazna, ta ćelija se preskače. U suprotnom, privremena vrijednost je dodjeljena ćeliji u opsegu od 1 do 9. Na slici ispod vidimo da je prva ćelija (0,0) preskočena i da je vrijednost „1“ upisana i narednu ćeliju (0,1).

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

2. Nakon toga, vrši se test, da se utvrdi da li je postavljena vrijednost dozvoljena. Nova vrijednost se poredi sa svim vrijednostima u njenom redu, njenoj koloni i njenom bloku. Ako se naiđe na istu vrijednost, data vrijednost se povećava za jedan i ponavlja se postupak testiranja. Ako se prođe kroz sve vrijednosti te ćelije, algoritam se vrati na prethodnu ćeliju.
3. Proces se ponavlja dok program ne nađe element koji potencijalno može da stoji na toj lokaciji.

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

4. Kada je uslov zadovoljen, funkcija se rekurzivno poziva na narednu lokaciju. Da bi optimizovli softver, kada se funkcija pozove drugi put, prvi element koji gleda je prethodni element uvećan za 1. Na primer, ovde, pošto smo upravo pretpostavili da je „3“ na lokaciji (0,1), algoritam će pogoditi „4“ sledeću lokaciju.
5. Kada dubina rekurzije dostigne vrijednost 81, znamo da su sve ćelije posjećene, tako da ako riješenje nije pronađeno, algortam se vraća u nazad i pokušava sa svim mogućim kombinacijama pozivajući rekurziju.
6. Ovaj postupak se ponavlja sve do onog momenta kada su sve moguće kombinacije isprobane. Ako riješenje nije pronađeno, ispisuje se poruka da tabla nema riješenje (Za testiranje aplikacije znamo da tabla ima rješenje, tako da će za određeno vrijeme doći do rješenja.)

4.4 Optimizacija brute-force algoritma

Po svojoj definiciji, brute-force algoritam ima najlošiju vremensku složenost, ali garantuje rezultate. Vrijeme potrebno da se riješi tabla najviše zavisi od pozicije od koje se kreće, i koji elemnti su prvi posjećeni. Čak je moguće, ali ne vjerovatno da vremenska složenost bude O(1) ako je tačno rješenje ono dobijeno u prvom prolasku.

Može biti primamljivo da se solver pokuša optimizovati tako što će se testirati razne početne pozicije i različite početne vrijednosti ali ova optimizacija bi bila validna samo za određenu tablu. Najbolji način koji se pokazao za optimizaciju je da početnu lokaciju uzmemo nasumično kao i početnu vrijednost. Da bi se ovo implementiralo, korištene su ciklične petlje da pretražuju redove i koolone i testirane vrijednosti. Da bi se pazilo koliko duboko ide algoritam, inkrementiramo indeks na svaki poziv funkcije tako da kad dođe do 81, što je maksimalna dužina table, funkcija se prekine.

Ovaj pristup takođe daje veliku svestranost u pogledu višenitnosti. Iz razloča što algoritam počinje od nasumične lokacije, funkcija rješavanja može pozvati veliki broj niti i generisati startne pozicije koje su dovoljno odmaknute jedna od druge za nesmetan rad. Ova osobina, korisniku daje mogućnost da slobodno dodaje niti bez modifikacije algoritma.

4.5 Višenitnost i sinhronizacija

Kako je zadatak testirati veoma veliki skup vrijednosti, upotreba paralelnog računarstva je od velike pomoći. Za ovaj konkretan problem, korišteno je mnogo niti da bi se dobio validan set vrijedosti. Java je izabrana kao platforma zbog svoje lakoće implemetiranja višenitnosti. Glavna klasa brine o kreiranju niti, a svaka nit preuzima dio table koju riješava. Čim jedna nit riješi tablu, štampa rješenje i vrijeme koje je potrebno da se riješi tabla.

Da bi se upravljalo sinhronizacijom, koristi se globalna varijabla pod nazivom „finished“. Kada se glavna funkcija pokrene, postavlja vrijednost varijable „finished “ na „false“. Kako niti pokušavaju da riješe tablu, one gledaju u varijabu „finished“ i prestaju sa radom ukoliko je vrijednost varijable postavljena na „true“, u suprotnom one nastavljaju sa svojim radom. Čim jedna od niti riješi tablu, ta nit postavlja vrijednost varijable na „true“, što prisiljava ostale niti da prekinu sa radom. U međuvremenu, u glavnoj funkciji, nakon što su sve niti inicijalizovane, program pada u while(!finished) petlju. Čim se prekine petlja, zna se da je jedna od niti riješila tablu, tako da se sve ostale niti uništavaju.

5. Izrada GUI dijela aplikacije

Gui dio aplikacije sastoji se iz xml dijela i klase SudokuBoard koja pomoću android grafike iscrtava tablu na ekran.

5.1 Android Grafika

Za prikaz table i brojeva na ekran korištena je android grafika. Android pruža bokat skup API-ja za iscrtavanje 2D elemenata. Android grafika sadrži grafičke alate niskog nivoa kao što su kanvasi, boje, tačke i pravougaonici koji direktno upravlaju crtanjem na ekranu. Postoje dva načina za iscrtavanje 2D grafike

- Iscrtavanje na View objekat koji se nalazi u paketu layout
- Iscrtavanje direktno na kanvasu

Za iscrtavanje table za sudoku korišten je način iscravanja direktno na kanvas iz razloga jer se tabla svaki put kad se unese nova vrijednost tabla se iscrtava ponovo.

5.2 Iscrtavanje table

Iscrtavanje table se vrši pomoću klase SudokuBoard koja iscrtava sudoku tablu na ekran. Klasa je organizovana na način da se prvo nalaze biblioteke koje se koriste, pa sama klasa. Sve varijable korištene u kodu se nalze na početku klase, zatim konstruktor i nakon toga metode.

5.2.1 Konstruktor

Konsruktor klase SudokuBoard prima dva parametra: konzekst i atribute definisane radi lakše organizacije koda. Atributi se smještaju u varijable radi daljeg rada programa.

5.2.2 onMeasure metoda

Metoda onMeasure je sistemski ugrađena metoda koja je predefinisana za potrebe programa. Metoda prima atribute širinu i visinu. Potrebno je da dimenzija table bude kvadratnog oblika i da je prilagodljiva za sve veličine ekrana. U ovoj metodi postavljamo veličinu table tako što i širinu i visinu table postavljamo na manju dimenziju ekrana. Takođe, postavljamo i dimenziju jedne ćelije na 1/9 postavljene dimenzije table.

5.2.3 onDraw metoda

Metoda onDraw je sistemski ugrađena metoda koja je predefinisana za potrebe programa. Atribut koji prima je kanvas na kom se iscrtava tabla. U ovoj metodi postavljamo sve vrijednosti varijablama koje su potrebne da bi se iscrtala tabla kao što su atributi za pozadinu table, za ispunu ćelije, za oznaku ćelije i za boju broja.

Sve metode koje smo mi definisali za iscrtavanje dijelova table se pozivaju u ovoj metodi a to su metoda za farbanje ćelija koje su selektovane, za crtanje okvira table, za crtanje same table i za iscrtavanje brojeva na njoj.

5.2.4 onTouchEvent metoda

Metoda onTouchEvent je sistemska metoda koja se odnosi na događaj koji se desio prilikom dodira ekrana. Svrha ove metode je da registuje ćeliju koju je korisnik izabrao. Registovanje ćelije dobijaju se podaci u kom redu i u kojoj koloni se ona nalazi što korisniku omogućava da se lakše

snalazi u prostoru table. Metoda je tipa „boolean“ pa ako je korisnik kliknuo na nešto je ćelija vratiće se „true“ a u suprotnom „false“.

5.2.5 drawNumbers metoda

Metoda drawNumbers je metoda koja iscrtava brojeve na kanvas. Brojeve koje ispisuje čita iz matrice u kojoj su smješteni brojevi u odgovarajućim redovima i kolonama. Ova metoda iscrtava brojeve koje korisnik unosi kao i brojeve koje tabla riješi s tim što im promjeni boju da korisnik zna koji brojevi predstavljaju riješenje.

5.2.6 writeNumber metoda

Metoda writeNumbers je metoda koja ispisuje brojeve koji se nalaze u matrici tako što uzima kao atribute red i kolonu u kom se nalazi broj. Njega konvertuje u String tip podatka i ispisuje ga na tačno mjesto u sudoku tabli na kojoj se on nalazi. Ovdje je potrebno voditi računa o veličini brojeva koji se ispisuju jer nije ista veličina za sve uređaje. Kako smo predefinisali metodu onMeasure znamo veličinu table i ćelije i na osnovu toga ova metoda računa koje veličine i na kojim pozicijama će ispisati brojeve.

5.2.7 colorCell metoda

Metoda colorCell je metoda koja farba red i kolonu i ćeliju koju je korisnik dodirnuo. Ćelija se farba u drugoj boji radi lakšeg snalaženja korisnika u prostoru sudoku table. Metodu je moguće proširiti da se registruje i blok u kojoj se nalazi ali mi smo smatrali da je dovoljno da se registruju samo red kolona i ćelija i da bi registrovanje bloka bilo doprinjelo lošijijem utisku orjentacije.

5.2.8 drawThickLine metoda

Metoda drawThickLine je pomoćna metoda kod iscrtavanja table gdje se linije table iscrtavaju deblje za pozicije koje odvaju blokove. Svrha ove pomoćne metode je smanjenje koda radi njegove redudantnosit jer se metoda poziva za iscrtavanje redova i kolona.

5.2.9 drawThinLine metoda

Metoda drawThinLine je pomoćna metoda kod iscrtavanja table gdje se linije table iscrtavaju tanje za pozicije koje odvaju ćelije. Svrha ove pomoćne metode je smanjenje koda radi njegove redudantnosit jer se metoda poziva za iscrtavanje redova i kolona.

5.2.10 drawBoard metoda

Metoda drawBoard je metoda koja iscrtava tablu pozivajući metode za cranje tankih i debelih linija na odgovarajućim mjestima.

5.3 Layout

Layout je aplikacije je definisan tako što je tabla dodata pomoću android grafike a sve ostale komponente se dodate pomoću grafičkog dizajnera.

Layout se sastoji od menija, table, niza dugmića na kojima se nalaze brojevi od 1 do 9, kao i dugme koje pokreće program. Sve metode koje su vezane za dugmiće su onClick metode koje su definisane u izvršnom kodu.

5.4 Meni

U paketu menu ubačeni su radioButtons koji se koriste pri izboru broja niti. Kako će se izabrati najoptimalnija nit ona će biti čekirana za početnu vrijednost. Korisniku je omogućeno da bira

različit broj niti radi uočavanja razlike u brzini izvršenja koda. Omogućeno je da samo jedno polje može biti izabrano tj. da korisnik ne može istovremeno izabrati 1 i 4 niti u isto vrijeme.

5.5 Vrijednosti

5.5.1 Atributi

Radi lakšeg upravljanja aplikacijacijom kreiran je xml za resurse vezane za tablu i njen izgled. U xml-u su je definisano ime pod koji će se dohvatati atributi kao i sve boje koje su potrebne za iscrtavanje table.

5.5.2 Stringovi

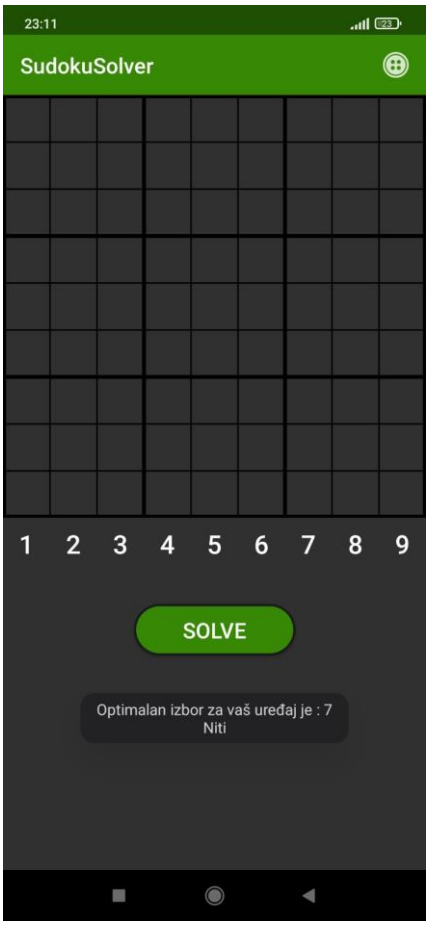
U xml-u string su definisani svi natpisi koji se koriste u aplikaciji radi lakšeg snalaženja u kodu i izbjegavanja „hardkodiranja“.

5.5.3 Boje

U xml-u colors su definisane sve boje koje se koriste u aplikaciji radi lakšeg snalaženja u kodu i izbjegavanja „hardkodiranja“.

5.5.4 Teme

Teme su stilizovane tako što je na ugrađenu temu dodalo dio gdje je sistemski ugrađen meni zamjenjen sa menijem koji smo mi definisali tj. sa ikonicom needle4. Tema se atomatski bira na osnovu moda koji je postavljen na korisničkom uređaju. Odrađene su svijetla i tamna tema.



5.6 Custom komponente

Dugme solve je custom komponenta definisana u paketu drawable. Dugme je stilizovano tako što je crtano od početka sa svim parametrima potrebni da bi se iscrtalo dugme.

6. Integracija logičkog dijela

6.1 Implementacija logičkog dijala

Prema analizi iz poglavlja 4. izvršena je implementacija algoritma u JAVA programskom jeziku kako bi bilo moguće integrisati logički deo u Android aplikaciju. Implementacija je predstavljena u sledećem dijelu. Odrađena je sa test tabelama u okviru JAVA konzolne aplikacije.

Bitno je napomenuti dva karakteristična atributa koja su riješila probleme prilikom implementacije:

- Kada prvi thread stigne do izvršenja metode koja je *synchronized* svi ostali threadovi ne mogu da je izvršavaju. Dolaze u red za čekanje.
- Statička varijabla za svaki thread se kešira. Prilikom brze promjene ove statičke varijable dva thread a neće vidjeti istu vrijednost. To dovodi do trke dva thread a, odnosno do beskonačne petlje. Da bi se to izbjeglo koristi se *volatile*.

```
package test;

import java.util.Random;

public class MainClass {

	public static void main(String[] args) {
		Solver board = new Solver();

		int[][] newBoard = new int[9][9];
		int i, j, n = 0;
		int[] puzzle = { 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 9, 0, 4, 0, 0, 0, 5, 0, 0, 0, 6, 0, 0, 0, 7, 0, 0, 0, 5, 0, 9, 0,
				3, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 8, 5, 0, 0, 4, 0, 7, 0, 0, 0, 0, 0, 6, 0, 0, 0, 3, 0, 0,
				0, 9, 0, 8, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0 };
		for (i = 0; i < 9; i++)
			for (j = 0; j < 9; j++)
				newBoard[i][j] = puzzle[n++];
		board.setBoard(newBoard);

		SolveBoardThread.finished = false;

		SolveBoardThread solve = new SolveBoardThread(board);
		SolveBoardThread solve2 = new SolveBoardThread(board);
		SolveBoardThread solve3 = new SolveBoardThread(board);
		SolveBoardThread solve4 = new SolveBoardThread(board);
		SolveBoardThread solve5 = new SolveBoardThread(board);

		solve.start();
		solve2.start();
		solve3.start();
		solve4.start();
		solve5.start();

		while (!SolveBoardThread.finished)
			;

		try {
			solve.interrupt();
```

```
                solve.interrupt();
                solve2.interrupt();
                solve3.interrupt();
                solve4.interrupt();
                solve5.interrupt();

                solve = null;
                solve2 = null;
                solve3 = null;
                solve4 = null;
                solve5 = null;
} catch (Exception e) {
}

int[][] newBoard2 = new int[9][9];
n = 0;
int[] puzzle2 = { 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 9, 0, 4, 0, 0, 0, 5, 0, 0, 0, 6, 0, 0, 0, 7, 0, 0, 0, 5, 0, 9,
                        0, 3, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 8, 5, 0, 0, 4, 0, 7, 0, 0, 0, 0, 0, 6, 0, 0, 0, 3, 0,
                        0, 0, 9, 0, 8, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0 };
for (i = 0; i < 9; i++)
                for (j = 0; j < 9; j++)
                        newBoard2[i][j] = puzzle2[n++];
board.setBoard(newBoard2);

solve = new SolveBoardThread(board);
solve2 = new SolveBoardThread(board);
solve3 = new SolveBoardThread(board);
solve4 = new SolveBoardThread(board);
solve5 = new SolveBoardThread(board);

solve.start();
solve2.start();
solve3.start();
solve4.start();
solve5.start();

while (!SolveBoardThread.finished)
                ;
```

```
		try {
			solve.interrupt();
			solve2.interrupt();
			solve3.interrupt();
			solve4.interrupt();
			solve5.interrupt();

			solve = null;
			solve2 = null;
			solve3 = null;
			solve4 = null;
			solve5 = null;
		} catch (Exception e) {
		}

}

static class SolveBoardThread extends Thread {
		volatile static boolean finished;

		private Solver board;
		int[][] boardMatrix;

		SolveBoardThread(Solver board) {
			this.board = board;
		}

		@SuppressWarnings("static-access")
		@Override
		public void run() {
			if (Thread.currentThread().interrupted()) {
				return;
			}
			int i, j, value;
			Random generator = new Random(System.currentTimeMillis());
			i = generator.nextInt(8);
			j = generator.nextInt(8);
```

```
                i = generator.nextInt(8);
                j = generator.nextInt(8);
                value = generator.nextInt(8) + 1;

                int[][] matrix = board.getBoard();
                boardMatrix = new int[matrix.length][];

                for (int brojac = 0; brojac < matrix.length; brojac++) {
                        int[] aMatrix = matrix[brojac];
                        int aLength = aMatrix.length;
                        boardMatrix[brojac] = new int[aLength];
                        System.arraycopy(aMatrix, 0, boardMatrix[brojac], 0, aLength);
                }

                if (solve(i, j, boardMatrix, 0, value)) {
                        if (finished == false) {
                                printBoard(board);
                        }
                }

}

static boolean solve(int row, int col, int[][] board, int xTimes, int startV) {
                if (SolveBoardThread.finished == true) {
                        return false;
                }
                if (xTimes == 81)
                        return true;
                int f = 1;
                for (int p = 0; p <= 8; p++) {
                        for (int q = 0; q <= 8; q++) {
                                if (board[p][q] == 0) {
                                        f = 0;
                                }
                        }
                }
                if (f == 1)
                        return true;
```

```
                return true;

        if (++col == 9) {
                col = 0;
                if (++row == 9)
                        row = 0;
        }

        if (board[row][col] != 0) {
                return solve(row, col, board, xTimes + 1, startV);
        }

        for (int val = 1; val <= 9; ++val) {
                if (++startV == 10)
                        startV = 1;

                if (allowedHere(row, col, startV, board)) {
                        board[row][col] = startV;
                        if (solve(row, col, board, xTimes + 1, startV))
                                return true;
                }
        }

        board[row][col] = 0;
        return false;
}

synchronized void printBoard(Solver board) {
        if (finished == false) {
                finished = true;
                board.setBoard(boardMatrix);

                int r = 1;
                int i, j;

                if (r == 1) {
                        r = 0;
```

```
                                        for (i = 0; i < 9; i++) {
                                                if (i % 3 == 0)
                                                        System.out.println(" ------------");

                                                for (j = 0; j < 9; j++) {
                                                        if (j % 3 == 0)
                                                                System.out.print("| ");
                                                        if (board.getBoard()[i][j] == 0)
                                                                System.out.print("* ");
                                                        else
                                                                System.out.print(
                                                                Integer.toString(
                                                                board.getBoard()[i][j]) + " ");
                                                }
                                                System.out.println("|");
                                        }
                                        System.out.println(" ------------");
                                }
                        }
                }

                static boolean allowedHere(int row, int col, int value, int[][] board) {
                        int i;

                        for (i = 0; i < 9; i++) {
                                if (board[row][i] == value)
                                        return false;

                                if (board[i][col] == value)
                                        return false;

                                if (board[row / 3 * 3 + i % 3][col / 3 * 3 + i / 3] == value)
                                        return false;
                        }
                        return true;
                }
        }
}
```

Takođe bitno je napomenuti da je svaki thread imao svoju posebnu kopiju matrice, da bi svaki mogao za sebe da je rješava. Prije pokretanja metode solve izvršeno je kloniranje matrice kako ne bi dolazilo do preplitanja dva thread a u izvršavanju.

6.2 Integracija logičkog dijela u Android aplikaciju

Integracija logičkog dijela izvršena je u MainActivity klasi Android aplikacije. U tom dijelu kao podklasa klasi MainActivity integrisana je klasa SolveBoardThread.

Izvršenje rješavanja Sudoku tablice poziva se u metodi solve. Prema izabranom broju thread ova pokreće se izvršenje i čeka se na rješenje sudoku tablice.

Kada se aplikacija otvara izabran broj thred ova je onaj optimalni broj za uređaj. Maksimalan izbor je 8 thred ova. Optimalan broj thred ova se računa prema formuli: Runtime.getRuntime().availableProcessors() - 1;

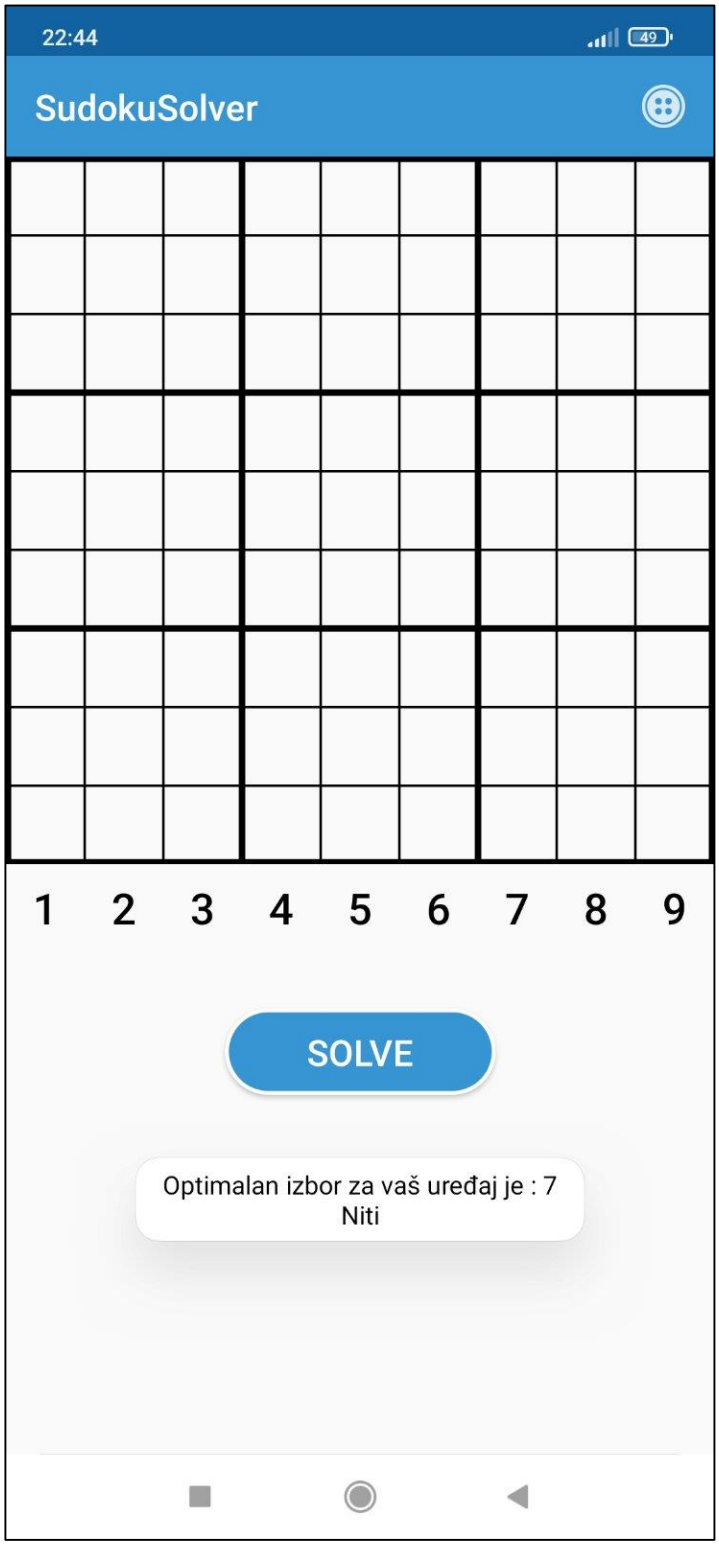
-1 u izrazu predstavlja glavni MainThread u kome se izvršava aplikacija.

Kod Android aplikacije dostavljen je kao prilog.

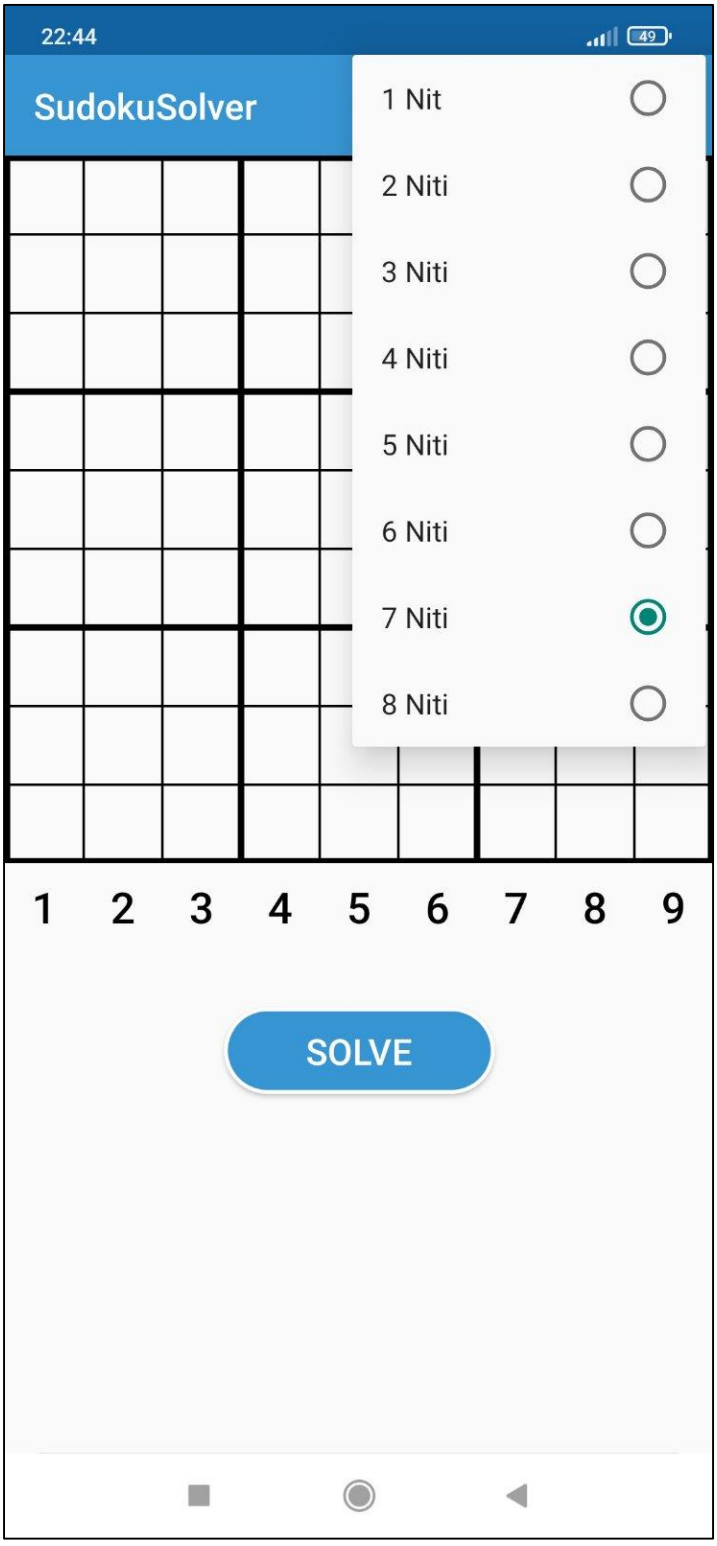
7. Pregled aplikacije i analiza poboljšanja

7.1 Pregled aplikacije

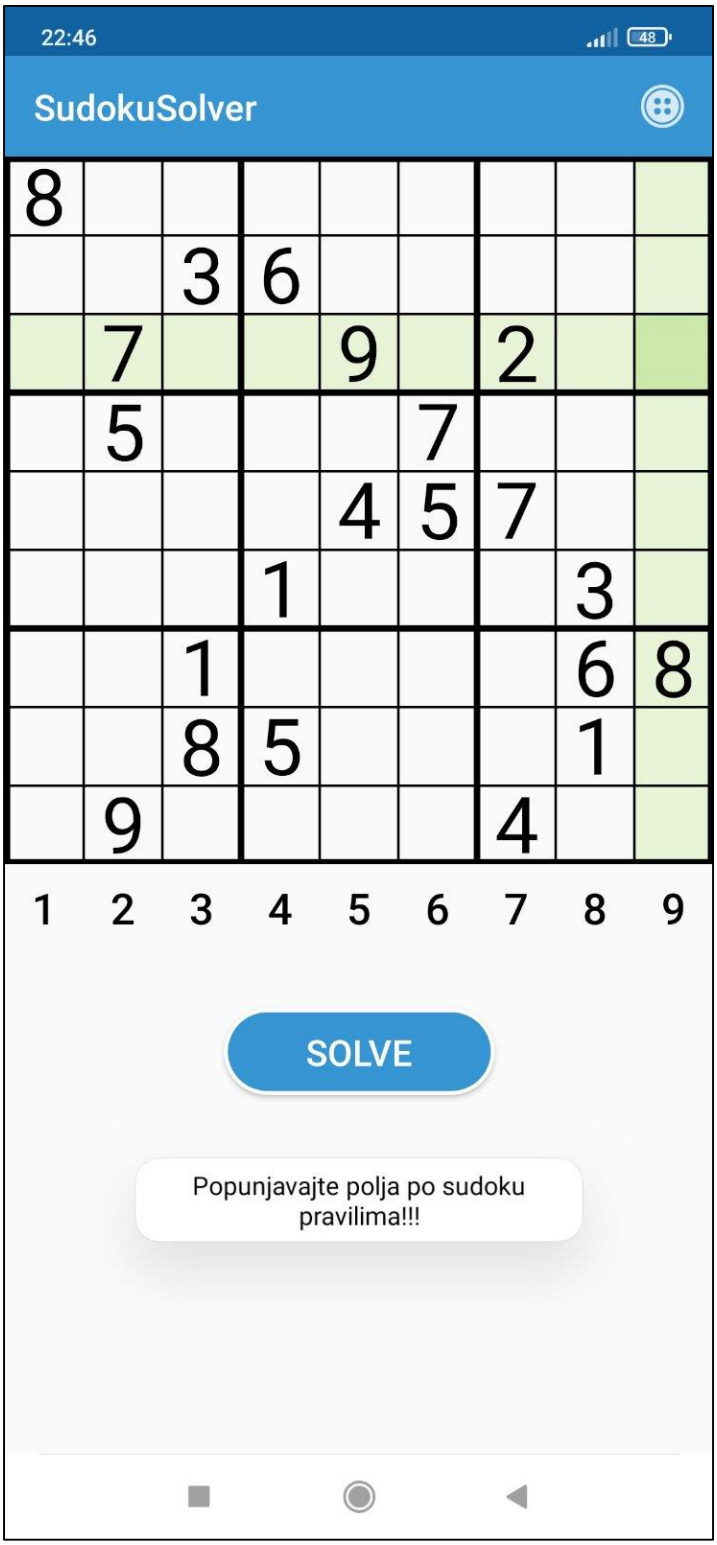
Početna stranica SudokuSolver aplikacije predstavljena je na sledećoj slici. Kada se aplikacija pokrene, imamo indikaciju koji je najoptimalniji izbor za uređaj. Inicijalni izbor niti je najoptimalniji broj.



Korisnik ima mogućnost izbora najviše 8 niti. Ukoliko je optimalan broj za uređaj veći od 8, aplikacija će izabrati najveći mogući broj niti, tj. 8.

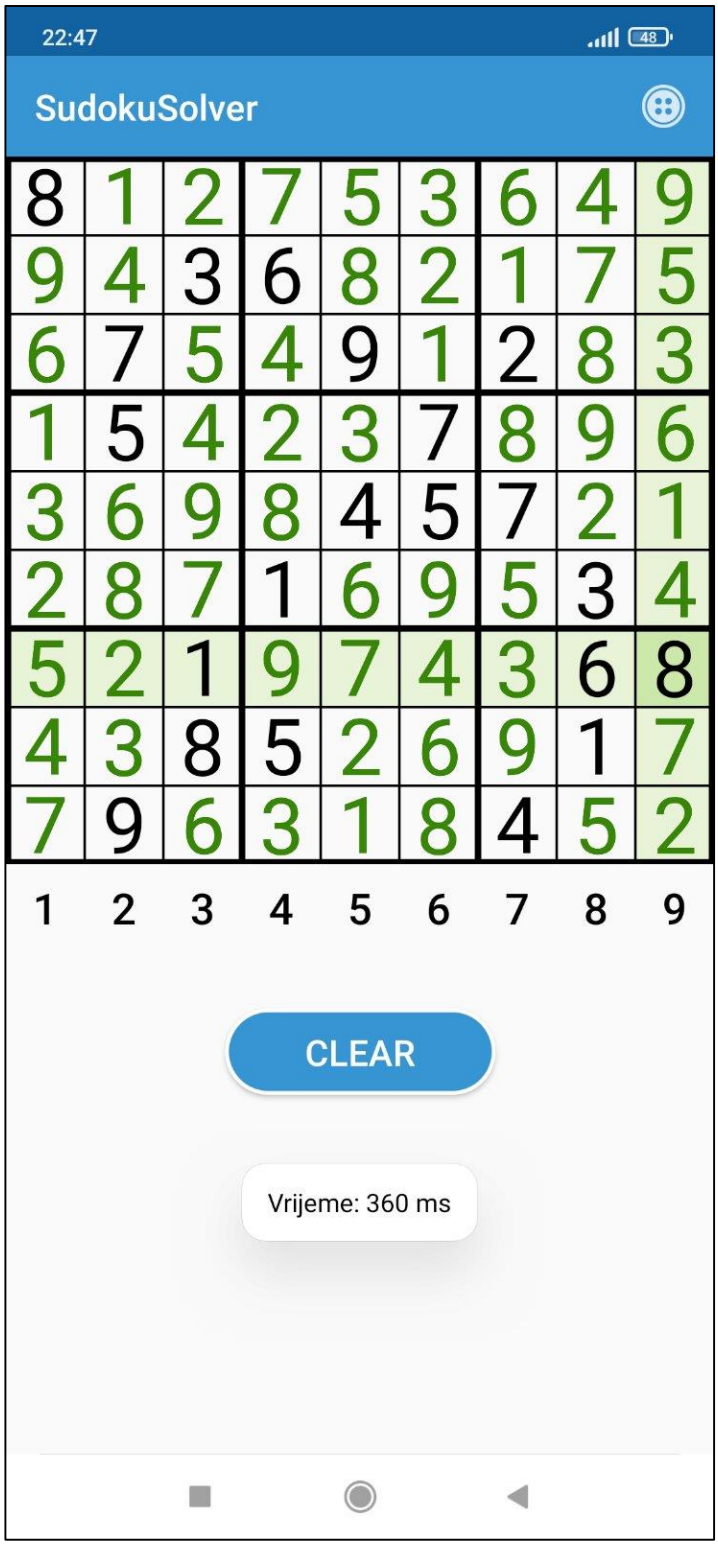


Korisniku je omogućen unos Sudoku tablice ali samo po pravilima popunjavanja. Ukoliko se popunjavanje polja pokušava vršiti protivno Sudoku pravilika korisnik će dobiti upozorenje da nije moguće izvršiti upis.



Takođe korisniku je omogućeno uklanjanje popunjene vrijednosti. Nakon što je izabrano polje iz koje želi da se obriše vrijednost, dodirom na istu vrijednost koja je popunjena u polju izvršava brisanje. Ukoliko dodirne bilo koju drugu vrijednost prebrisaće trenutni unos.

Nakon unosa Sudoku tablice, dodirom na Solve, pokreće se riješavanje. Kada se tablica riješi korisnik dobija prikaz i vrijeme koje je utrošeno.



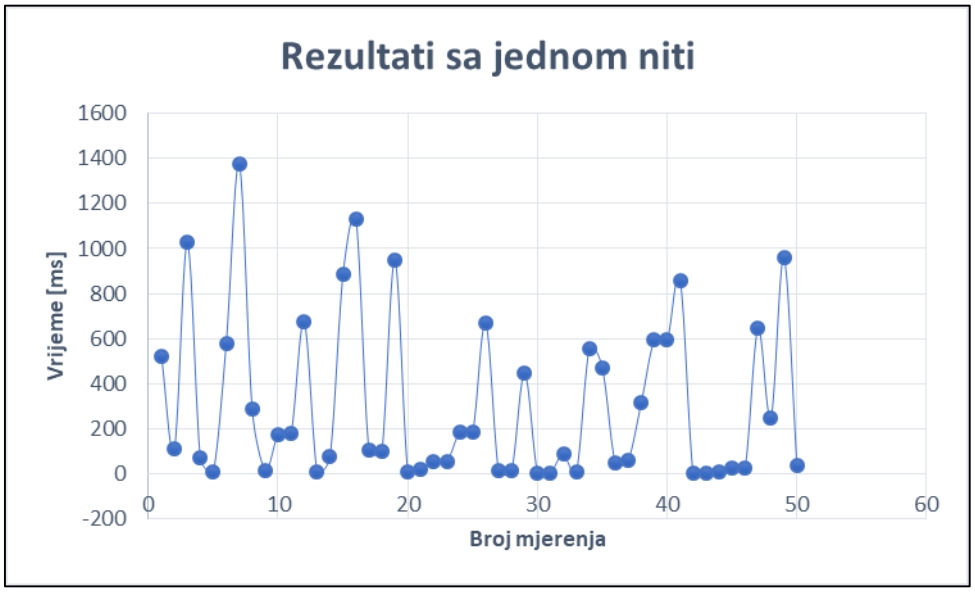
7.2 Analiza poboljšanja

Kako ne znamo koju će početnu poziciju izabrati algoritam potrebno je aplikaciju testirati više puta da bi se dobila što tačija procjena vremena algoritma. Takođe, vršeno je testiranje na više različitih niti. Postupak mjerenja za svaku nit se vršio 50 puta za istu tablu.

Testiranje je sprovedeno na uređaju sa procesorom **Octa-core Max2.00GHz** i verzijom androida **11 RPA1A.200720.011 .**

7.2.1 Analiza rezultata sa jednom niti

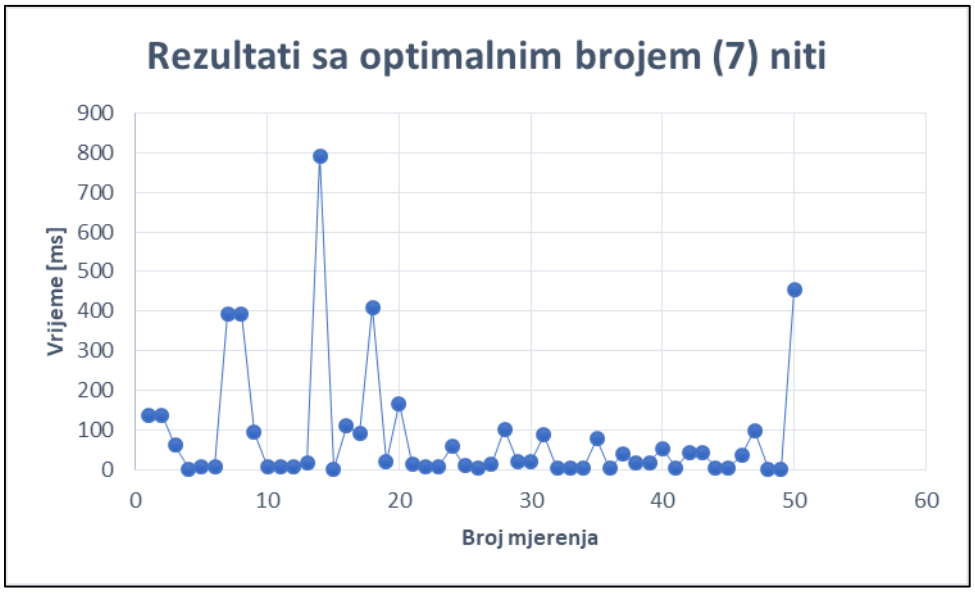
Prvia testiranja su bila sa uključenom samo jednom niti, odnosno bez paralelizacije. Svrha ovog testiranja je da bi se vidjelo koliko je vremena u proseku trebalo da se riješi postavljena tabla. Analiza rezultata je prikazana grafom ispod.



Ovi rezultati opravdavju pretpostavku da vrijeme riješenja Sudoku table, korištenjem **Brute-Force** algoritma, u potpunosti zavisi od izabrane početne lokacije. Možemo primjetiti nekoliko primjera gdje vrijeme riješavanja iznosi preko 1 sekundu što predstavlja najgore moguće odabranu početnu lokaciju. Kao drugu krajnost, uočavamo slučajeve gdje je riješenje postignuto za nekoliko milisekundi. Ovi slučajevi se dešavaju kada je početna pozicija izabrana sa minimalnim brojem vraćanja u nazad.

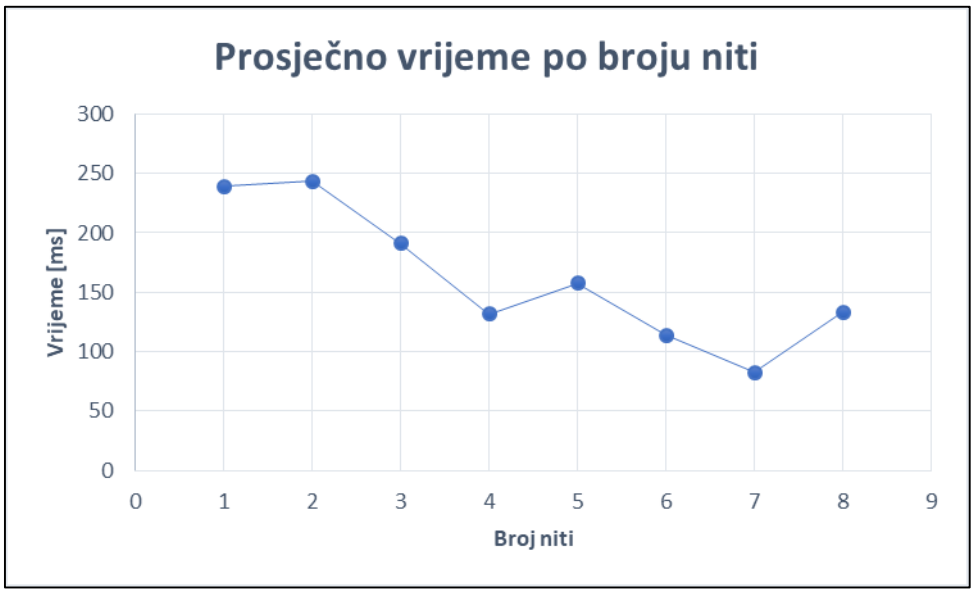
7.2.2 Analiza rezultata sa optimalnim brojem niti za korišteni uređaj

Aplikacija automatski vrši odabir broja niti za uređaj na kom se nalazi na osnovu formule **Runtime.getRuntime().availableProcessors() – 1**. Postupak analize je isti kao i sa jednom niti ali vidimo znatno poboljšanje u vremenu izvršenja kada je korištena samo jedna nit. Svi podaci su ispod 1 sekunde.



7.2.3 Analiza rezultata prosječnog vremena potrebog da se tabla riješi na osnovu izbora niti

Aplikacija je pokretana sa različitim brojem niti, od 1 do 8, i prosječna vremena koje je potrebno da se tabla riješi su prikazani u tabeli ispod. Broj mogućih početnih mjesta je 9x9 sa 9 različitih početnih vrijednosti što daje 729 mogućih počentih stanja za praznu tablu. Za tablu koju smo mi koristili pri testiranju aplikacije broj mogućih stanja, s tim da su unaprijed zadane vrijednosti na 21 ćeliji, iznosi 540. Prosječno vrijeme po broju niti dato je na grafu ispod.



Na grafiku iznad vidi se da kako se mjenja broj niti od 1 do 7 tako vrijeme izvršavanja opada sa malim odstupanjima. Odstupanja su rezultat nasumičnosti algoritma i malog skua podataka. Najoptimalniji izbor niti je 7 što dokazuje gore pomenutu formulu. Nakon 7. niti grafik ponovo počinje da raste iz razloga što se broj niti više ne može rasporediti na fizički broj jezgara već jedno jezgro mora da preuzme još jednu logičku nit. Kako bi se broj niti povećavao tako bi i grafik rastao.

7.3 Rezultati mjerenja

U tabeli ispod dati su podaci prikupljeni prilikom testiranja aplikacije. Prvi red označava broj niti koji korištenih pri testiranju. Poslednji red označava prosječno vrijeme koje je potrebno da se riješi zadata SudokuTabla. Ostala polja predstavljaju neobrađene podatke koji su prikupljeni tokom sprovođenja testa.

1	2	3	4	5	6	7	8
523	661	63	94	412	3	138	11
110	28	527	42	7	47	138	11
1026	41	237	50	36	107	61	8
69	171	17	50	512	3	2	414
8	263	4	328	512	14	8	240
580	491	2	680	558	23	8	240
1375	19	63	56	24	29	393	6
285	878	88	56	24	47	393	52
14	106	7	76	885	3	94	2
173	37	7	76	138	7	8	179
179	882	304	4	224	52	8	6
675	301	3	75	224	102	8	31
10	749	3	4	52	292	17	2
73	66	841	4	52	45	793	135
883	63	4	58	1	2	1	65
1129	306	467	3	116	3	110	8
105	35	467	3	116	3	90	761
97	44	161	3	133	190	410	8
949	44	44	96	40	29	19	66
9	43	44	96	52	226	166	16
16	315	39	274	8	80	15	92
52	695	53	23	48	69	7	13
52	3	53	370	43	25	8	78
182	259	970	58	8	9	58	135
182	57	9	603	8	25	9	145
671	35	4	131	126	3	4	3
11	6	4	43	1218	769	12	54
11	6	46	16	26	55	102	5
446	46	22	3	36	64	21	5
3	1148	22	55	8	36	21	11
3	50	6	3	163	58	88	100
89	8	65	74	163	47	3	613
6	865	65	169	313	47	3	613
556	38	501	3	53	166	4	1023
470	38	501	8	15	7	77	3
46	41	1125	10	3	475	3	3
61	135	65	400	114	145	41	163
315	6	82	131	53	35	17	43

594	3	39	93	53	4	17	43
594	473	148	340	165	559	53	43
858	51	60	55	165	559	5	3
3	53	839	55	77	185	43	3
3	13	22	297	82	5	43	263
10	451	166	42	2	37	4	7
22	451	17	45	110	5	4	174
22	581	3	164	36	122	35	56
644	581	1060	118	47	122	98	56
247	9	34	68	47	6	2	43
959	98	138	1026	10	731	2	74
38	422	45	55	567	6	454	137
238,8667	243,3	191,12	131,72	157,7	113,66	82,36	133,4667

7.4 Algoritam sa šablonskim izborom ili slučanjnim odabirom

Kako bi zaključili opravdanost upotrebe višenitnosti, izvršen je test na tabli koja ima jedinstveno rješenje. Test je vršen sa dvije varijacije algoritma. Prvi algoritam nam predstavlja analizirani algoritam sa slučajnim odabirom početne pozicije i početne vrijednosti.

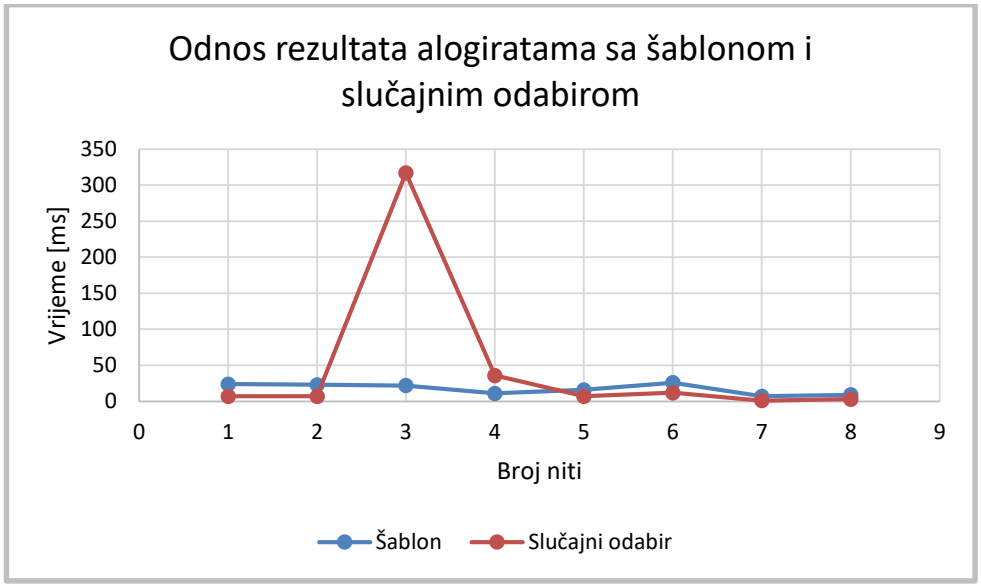
Drugi algoritam šablonski vrši dodjelu pozicija, tj prva nit kreće svoje izvršavanje od prvog slobodnog polja, druga nit od drugog slobodnog polja i tako analogno za odabran broj niti. Početna vrijednost je uvijek 1.

Analiza je vršena na sledećoj tablici:

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

Rezultati mjerenja predstavljeni su u sledećoj tabeli (prvi red algoritam sa šablonom, drugi red algoritam sa slučajnim odabirom):

1	2	3	4	5	6	7	8
24	23	22	11	16	26	7	9
7	7	317	36	7	12	1	3


Odnos rezultata alogiratama sa šablonom i slučajnim odabirom
Vrijeme [ms]
350
300
250
200
150
100
50
0
0
1
2
3
4
5
6
7
8
9
Broj niti
Šablon
Slučajni odabir

8. Zaključak

Nakon što je u dokumentu predstavljen čitav problem, izrada aplikacije, zaključujemo da je višenitno programiranje značajno ubrzalo rješavanje Sudoku table, ali samo ako je to slučaj sa nestabilnim algoritmom na više izmjerenih uzoraka.

Počeli smo sa sekvencijalnom verzijom aplikacije, a kasnije je taj način rada prebačen na paralelno izvršavanje. Prema rezultatima mjerenja primjetili smo značajnije poboljšanje, iako je analiza urađena na malom skupu podataka. Što bi skup podataka bio veći to bi poboljšanje bilo izraženije.

Ukoliko bi ipak izabrali veći broj niti od optimalnog bilo bi kontraproduktivno. Ipak, nismo uspjeli to dokazati, jer današnji mobilni uređaji su obično OctaCore (8 jezgara CPU a), optimalan broj niti nam je 7, a maksimalan izbor broja niti u aplikaciji je 8, što nam neće dati vidljive rezultate za poređenje.

Pošto je algoritam koji je primjenjen za rješavanje Brute-Force, sam po sebi spor, sa velikom vremenskom složenosti, višenitno programiranje je dalo značajno ubrzanje bez optimizacije.

U slučaju kada smo testirali algoritam na malom skupu podataka, sa tablicom sa jednim mogućim rješenjem došli smo do sledećih zaključaka:

- Algoritam sa slučajnim odabirom je izrazito nestabilan, ali gledajući veći skup podataka, daje brža rješenja.
- Algoritam sa šablonskim odabirom je stabilniji, ali je nešto i sporiji u većini slučajeva.
- Korištenje višenitnosti programiranja za rješavanje Sudoku a sa BruteForce algoritmom nije opravdano sa stanovišta uloženih resursa i dobijenog rezultata. Po dobijenim rezultatima najoptimalnije rješenje je sa 4 niti iz razloga što odnos broja uloženih resursa i brzine izvršavanja algoritma je najbolje.
- Gledajući iz ugla razvoja softvera višenitnost je dala doprinos ubrzanju, zbog toga je njegovo korištenje opravdano.

Prema svim navedenim dokazima dolazimo do toga da višenitno programiranje je značajno za ubrzavanje rješavanja problema, u slučaju da optimizacija nije moguća, ali najbolje rezultate daje samo u slučaju da je izabran optimalan broj niti.

9. Reference

[1] http://alitarhini.wordpress.com/2012/02/27/parallel-sudoku-solver-algorithm/
[2] http://en.wikipedia.org/wiki/Sudoku_solving_algorithms#Brute-force_algorithm
[3] sudoku.pdf (dauphine.fr)
[4] How to solve sudoku puzzles @ paulspages.co.uk
[5] https://sudokuspoiler.com/sudoku/sudoku9
[6] Miller, Russ and Laurence Boxer. Algorithms Sequential and Parallel: A Unified Approach. Hingham, MA: Charles River Media, 2005. Print.