TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IAY40LT

Hans Miikael Uuspõld 120911 IASB

SUDOKU LAHENDAJA

Bakalaureusetöö

Juhendaja
Peeter Ellervee
Professor / Ph.D

Tallinn 2015

Autorideklaratsioon

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Autor: Hans Miikael Uuspõld

27.05.2015

Annotatsioon

Käesoleva lõputöö teemaks oli sudoku lahendaja. Selle töö eesmärgiks oli programmeerida valmis rakendus, mis suudaks ükskõik millise sudoku lahendada ära praktilise ajaga kaasaegsel arvutil. Samuti pidi väljatöötatud programm võimaldama kontrollida sudoku legaalsust. See tähendab, et näha, kas sudoku on lahendatav ja kas sellel on ainult 1 lahendus. Arenduse käigus töötati välja mitmeid algoritme, mis võimaldaksid rakendada sudoku lahendamise loogikat. Lihtsamaks sudokude sisestamiseks ja laadimiseks ning programmi töö kontrollimiseks loodi ka lõputöö käigus programmile graafiline kasutajaliides.

Selleks, et paremini kirjeldada algoritme, on lõputöös välja toodud programmis rakendatud lahendusreeglid ning kuidas nad on rakendatud. Samuti on kirjeldatud viis erinevat meetodit numbri paigutuseks. Lühidalt on kirjeldatud ka sudoku ajalugu ning mis sudoku üldse on.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 6 peatükki, 18 joonist, 2 tabelit.

Abstract

This Bachelor's thesis is about a program implementing sudoku solving rules. The aim of this thesis was to create a program, which would be capable of solving any kind of sudoku puzzle, which is legal, in a practical amount of time on a modern computer. In order to achieve this, several algorithms were created. Two main parts of the algorithm were the implementation of sudoku solving rules and backtracking logic in order to make sure the solver doesn't get stuck, if there aren't sufficient rules implemented in the code to proceed. Additionally, the aim of the thesis was to analyze the effectiveness of the created sudoku solving algorithm.

Each implemented solving rule has been designed in such a manner that other rules are not needed to make a specific rule work. Due to this, it would be possible to add new functions to the sudoku solving program with ease.

The user can access the sudoku solver program functions through a graphical user interface, which allows to control in which manner to solve the given sudoku and also provides an easy way of saving and loading sudoku files.

As part of this thesis, the effectiveness of the solving algorithms were analyzed. In table form there is data about the different solving speeds of different difficulty level sudokus. Also comparisons were made between the solving algorithm with rules implemented and a backtracking algorithm. In addition to being able to solve sudokus, the program can detect whether or not the entered sudoku puzzle was legal or not. A legal sudoku puzzle is such a puzzle, which has only one unique solution and has no repeating numbers in rows, columns or sectors.

The thesis describes the algorithm and the solving rules implemented. In addition, a brief look at the history and nature of sudoku puzzles has been done. The thesis is in Estonian and contains 35 pages of text, 6 chapters, 18 figures, 2 tables.

Sisukord

Jooniste nimekiri

Tabelite nimekiri

1. Sissejuhatus

Lõputöö eesmärgiks oli disainida ja luua tarkvara rakendus, millele oleks võimalik anda sisse sudoku ja lasta programmil see lahendada. Rakenduse töö lõppedes tagastatakse kasutajale lahendatud sudoku ning lahenduskäik.

Sudoku on loogikamõistatus. Sudoku mängimine toimub 81 ruudulisel väljal, mis on ehitatud ülesse üheksast sektorist 3x3 moodustises. Iga sektor sisaldab üheksat ruutu 3x3 formatsioonis. Välja tohib ainult täita numbritega ühest üheksani ning ükski number ei tohi korduda reas, veerus ega sektoris. Õigesti koostatud sudokul leidub ainult üks lahend. Allpool, joonisel 1, on näide sudoku väljast, kus punasega on märgitud sektor.

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

Joonis 1.Sudoku välja näide.

Tihti arvatakse, et sudoku sai alguse Aasia maadest, kuid sudoku juuri võib leida varasemalt läänemaadest, spetsiifilisemalt Šveitsist, kus juba 18. sajandil tuli esimene idee, mis hilisemal ajal inspireeris tänapäevase sudoku arendamise. Leonhard Euleri pani oma leiutisele „Ladina Ruudustik“ sellel ajal aluse. 1979. aastal koostati kaasaegsem sudoku Goward Garns’i poolt, kuid see polnud siiski veel meile tuntud

sudoku. Aastal 1984 ilmus sudoku Jaapanis esmakordselt, seda ajaleht „Monthly Nikoli“ vahendusel. Lõpuks, 1986, tutvustati kahte uuendust mõistatusse, mis tagasid ta populaarsuse – ette ei ole antud kunagi rohkem kui 32 numbrit ja mõistatus muudeti sümmeetriliseks[1].

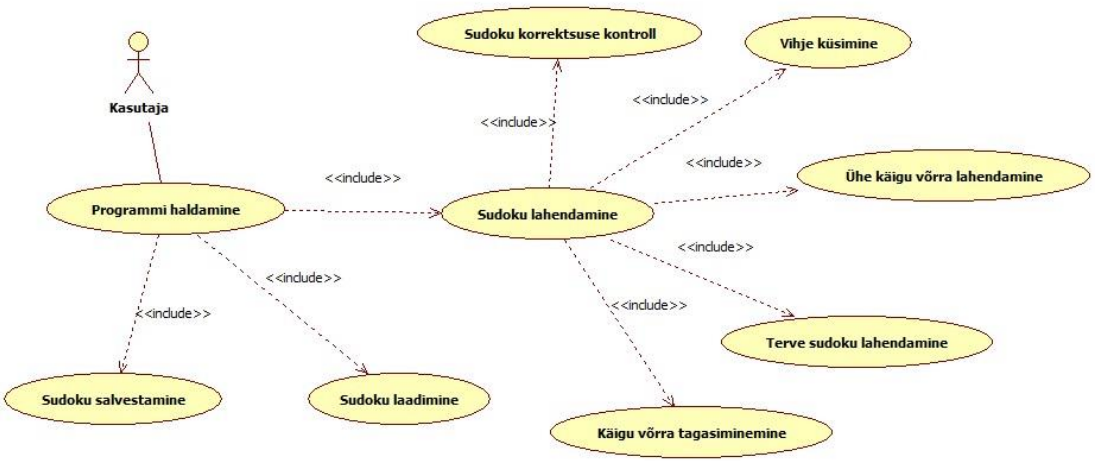
Järgnevalt on välja toodud lõputöö käigus lahendatavad ülesanded:

- Arendada välja rakendus, mis suudab võimalikult kiiresti lahendada etteantud korrektselt koostatud sudoku.
- Kui esineb viga andmetes, peab sudoku suutma seda tuvastada.
- Programm peab võimaldama andma kasutajale vihjeid.
- Programm peab võimaldama lahendada sudokut käigu haaval.
- Rakendus peab olema suuteline kontrollima, et tegu on unikaalse sudokuga.
- Rakendus peab võimaldama andmete salvestamise ja laadimise.

Lõputöö koosneb kahest osast – graafiline kasutajaliides ja sudoku lahendamise loogika algoritm. Esiteks kirjutati lõputöö käigus valmis graafiline kasutajaliides, et oleks võimalik kergelt kontrollida andmete sisestamist ja valikuid, mida soovitakse teha sisestatud andmetega. Liidese arendamine toimus kasutades keelt Java ning Keplar Eclipse keskkonnas.

Teisena töötati lõputöö käigus välja algoritmid, mis rakendaksid loogikat sudoku lahendamiseks. Algoritmide väljatöötamiseks kasutati samuti Java keelt ning arenduskeskkonnaks oli Keplar Eclipse. Algoritmid kontrollivad sisestatud sudoku korrektsust ning sellele järgnevalt hakkavad rakendama loogikat, et lahendada etteantud sudokut. Arenduse käigus töötati välja meetod kontrollimaks kas sisestatud andmed on korrektsed, algoritmid rakendamaks reeglite järgi lahendusloogikat ning tagurdusmeetodi implementatsioon.

Joonisel 2 on välja toodud kasutusjuhud, mis näitavad võimalusi kasutajale.



Joonis 2. Programmi kasutusjuhtude diagramm.

2. Lahendusalgoritmid

Lõputöö käigus oli vaja välja mõelda lahendusalgoritm sudokule, sest jõumeetodiga on sudoku lahendamine ebaeffektiivne. Ebaeffektiivsus tuleneb sellest, et sudokul on ligikaudu $6.67 * 10^{21}$ katte võimalust[2].

Lõputöö käigus sai kasutatud kahte lahendusfunktsiooni. Esimesena üritab programm lahendada sudokut spetsiifiliste reeglite järgi. Kui seda viisi kasutades pole enam võimalik lisada sudokule numbreid, kasutatakse tagurdusmeetodit, et jätkata lahendamist. Tagurdusmeetodist järeldatakse, et kas järgmise ruudu täitmine nõuaks reegli rakendamist, mis pole programmis kirjas või on tegu tupikuga.

2.1 Reeglite järgi lahendamine

See osa programmist üritab järjest rakendada loogikareegleid, et leida etteantud sudokule lahendus. Enamus reegleid võetakse läbi vastavalt raskusastmetele ehk siis alustatakse kõige lihtsamatest reeglitest ja liigutakse aina raskemateni. Selles mõttes sarnaneb programmi töö käsitsilahendamisega. Kõik reeglid on pandud ühte tsükklisse, mida käiakse läbi nii kaua, kuni on reegleid rakendades veel midagi lisatud. Tsükkli käigus kasutatakse viite erinevat meetodit, et numbreid asetada.

1. Üritatakse leida ruut veerus, kuhu sobib ainult üks number.

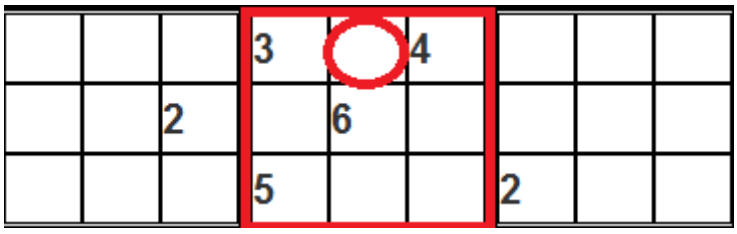
2. Üritatakse leida ruut reas, kuhu sobib ainult üks number.

3. Üritatakse leida ruut sektoris, kuhu sobib ainult üks number.

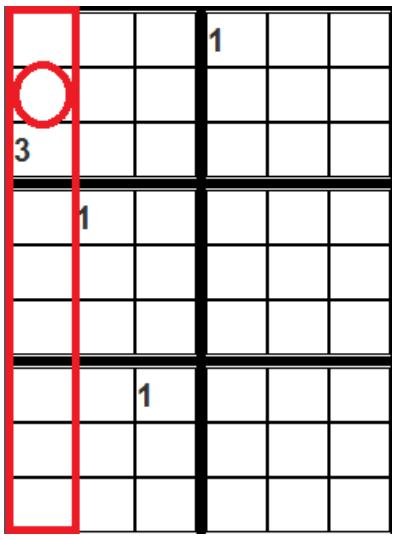
4. Üritatakse leida ruut, kuhu sobib ainult üks number.

5. Üritatakse leida number, mis sobib ainult ühte ruutu.

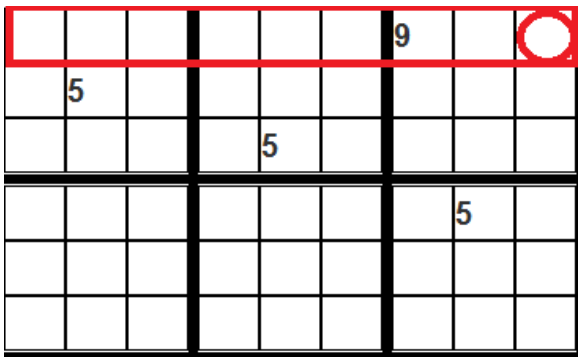
Joonistel 3, 4, 5, 6, 7 on näha näited nendest erinevatest meetoditest.



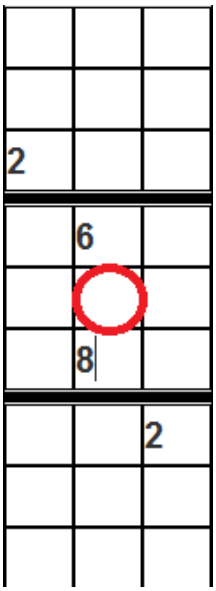
Joonis 3. Ruut sektoris, kuhu sobib ainult üks number.



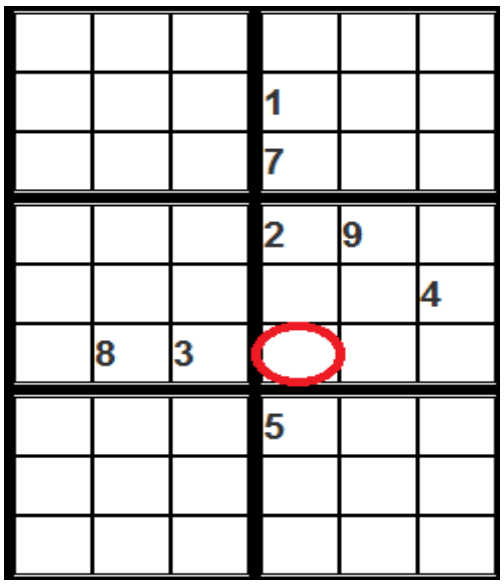
Joonis 4. Ruut veerus, kuhu sobib ainult üks number.



Joonis 5. Ruut reas, kuhu sobib ainult üks number.



Joonis 6. Ruut, kuhu sobib ainult üks number.



Joonis 7. Number, mis sobib ainult ühte ruutu.

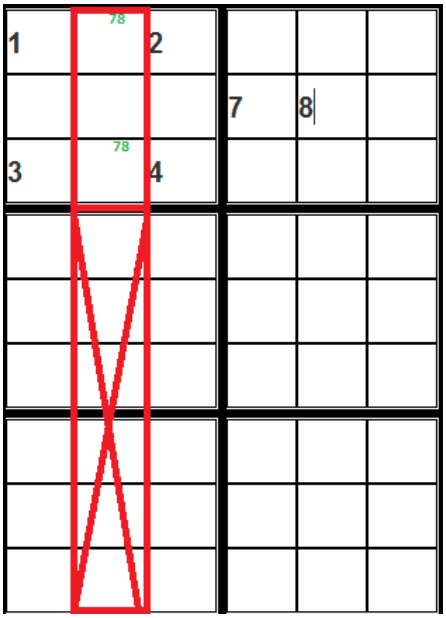
2.2 Rakendatud välistusreeglid

Antud lõputöö käigus rakendati neli välistusreeglit – sektor ja rida/veerg suhtlus, sektor/sektor suhtlus, peidetud hulgad ning nähtavad hulgad. Nende nelja reegliga katetakse mitmed sudoku pusle lahendamise juhud ära. Lisaks nendele on veel teisigi lahendamisloogikaid, kuid neid kasutatakse enamasti vähem, sest nende kasulikus ilmneb alles väga raskete sudokude juures.

2.2.1 Sektor ja rida/veerg suhtlus

Algoritmis uuritakse rida või veergu sektori sees ning kuidas ta suhtleb selle sama rea või veeruga väljaspool sektorit. Kui sektori sees reas või veerus on potensiaalsed kandidaadid numbrile reastatud ühes joones, saab välistada nende potensiaalsete kandidaatide esinemise väljaspool seda sektorit, etteantud reas või veerus.

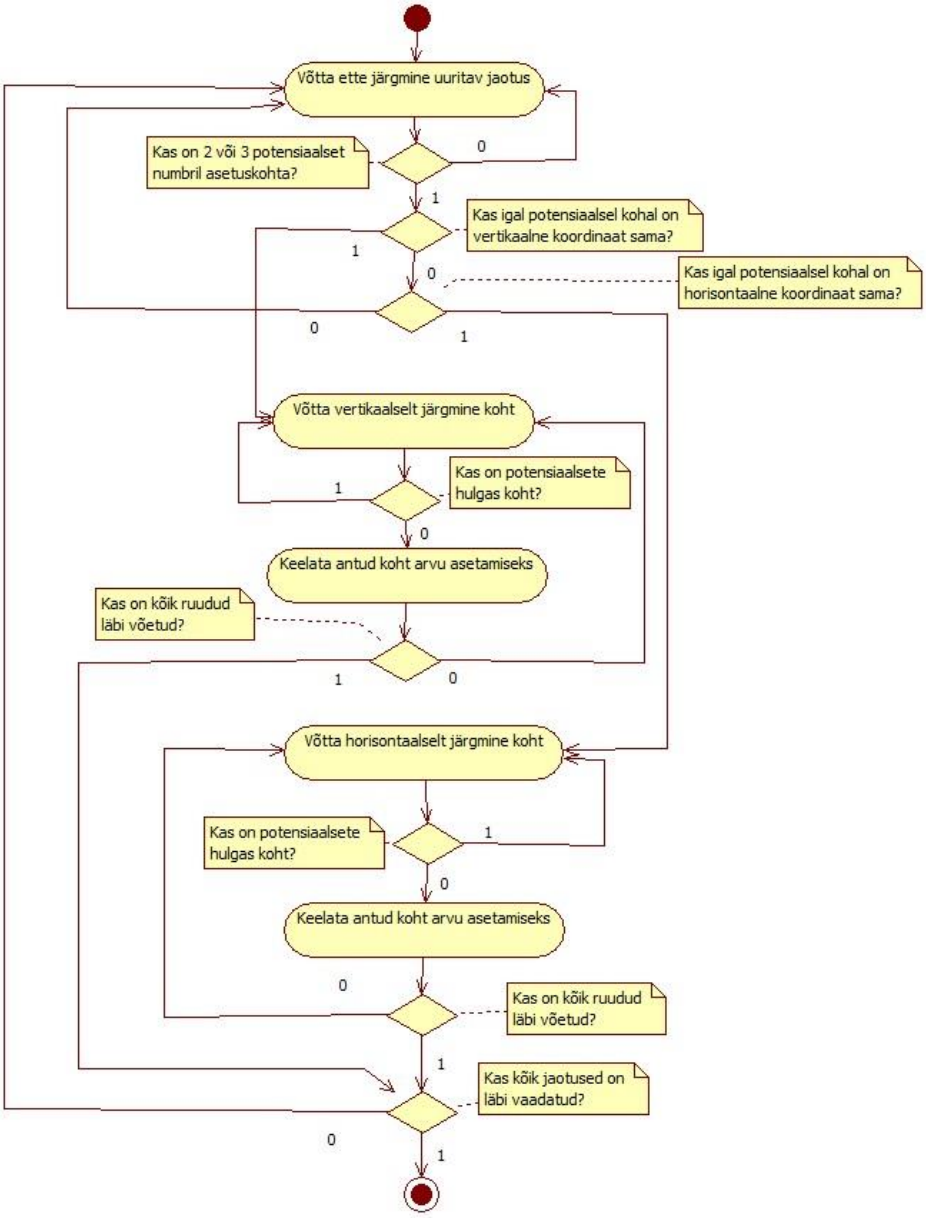
Antud näites, joonis 8, on numbrite 7 ja 8 ainsad võimalikud esinemised sektoris ühes joones. Sellest saab järeldada, et need numbrid ei saa esineda samas veerus väljaspool sektorit ning nõnda saab nad potensiaalsete kandidaatide hulgast välistada antud veerus.



Joonis 8. Sektor ja veerg suhtlus.

Koodis on antud reegel rakendatud joonisel 9 oleva tegevusdiagrammi alusel. Joonisel on jaotise puhul viidatud kas sektorile, reale või veerule, olenevalt, milline jaotise tüüp valiti reegli rakendamiseks. Ruudu puhul on mõeldud sudoku välja ühte üksust, millest väli on ehitatud. Koha puhul on samuti viitatud üksikule ruudule sudoku väljal.

Rakendatud algoritmis on alguses otsitud läbi kõik jaotises asuvate numbrite võimalikud asetuskohad ning siis on üritatud leida kas nad asuvad vertikaalselt samal joonel. Kui tuli vertikaalse osa lõpus välja, et vertikaalselt pole võimalik asetada, siis proovib kood horisontaalset võimalust. Juhul kui kumbki suund ei võimalda asetust, minnakse järgmise jaotuse juurde, et leida võimalusi.



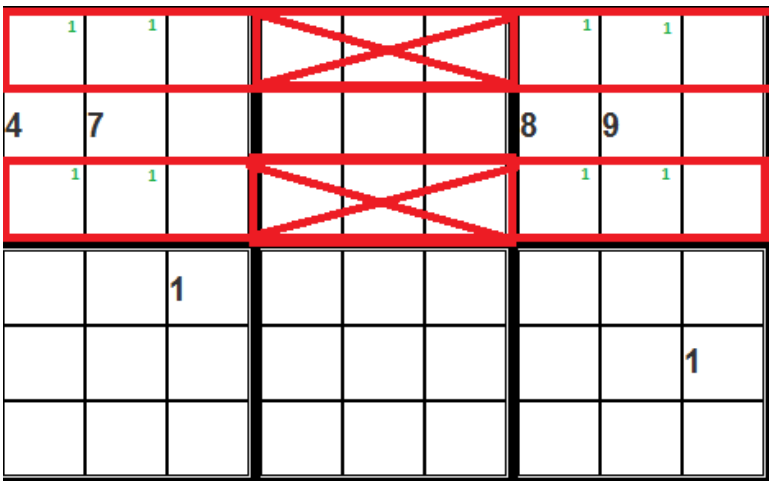
Joonis 9. Sektor ja rida/veerg suhtlus tegevusdiagramm.

2.2.2 Sektor/sektor suhtlus

Sektor/sektor suhtluse korral uuritakse algoritmis kolme erineva sektori potensiaalseid kandidaate kas veerus või reas. Kui leitakse, et kahel sektoril esinevad kandidaadid ainult kahes reas või veerus, mis on identsed kahe sektori puhul, siis saab välistada nende esinemine samades ridades/veergudes kolmandas sektoris.

Antud näites, joonis 10, on number ühe võimalikud potensiaalsed kohad ainult esimese ja kolmanda sektori esimeses ja kolmandas reas. Kuna mõlemal sektoril on number ühe

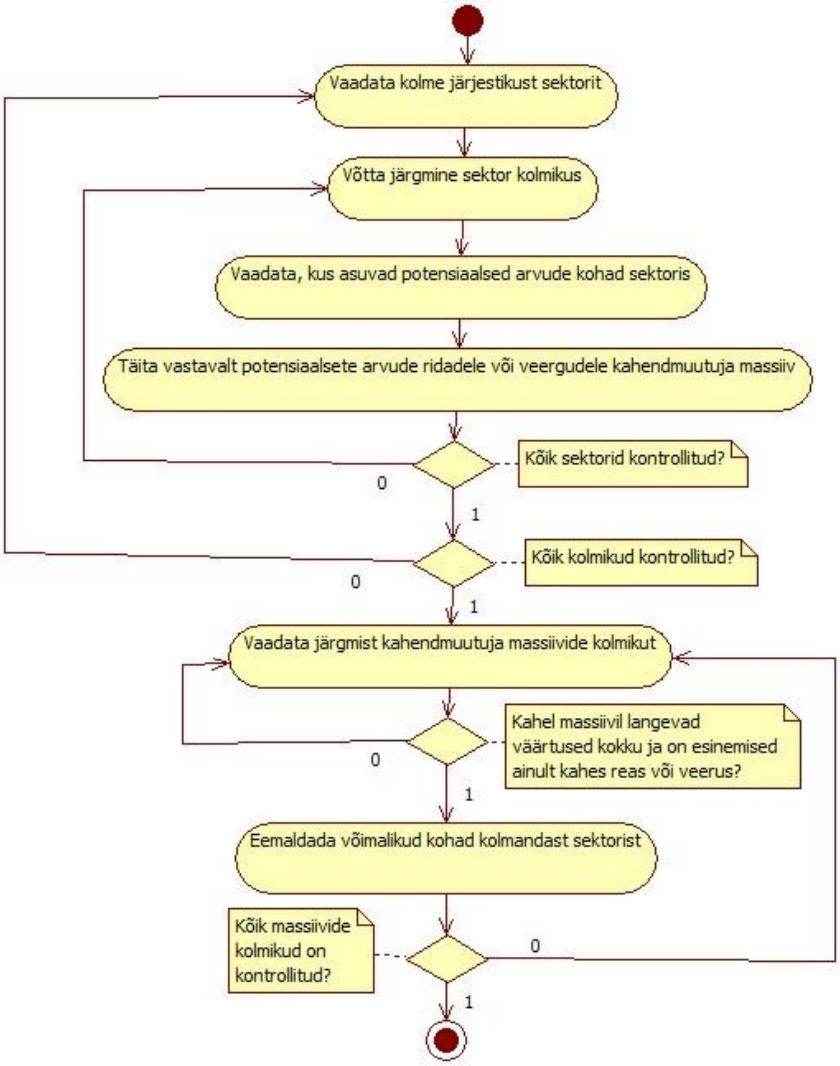
ainsad võimalikud kohad samades ridades, siis saab teise sektori puhul välistada nende esinemine esimeses ja kolmandas reas.



Joonis 10. Sektor/sektor suhtlus.

Koodis on rakendatud antud reegel joonisel 11 asuva tegevusdiagrammi alusel. Kolme järjestikuse sektori puhul mõeldakse kas vertikaalselt või horisontaalselt üksteise järel asuvaid sektoreid. Kahendmuutuja massiivi puhul on mõeldud kolme liikmega massiivi, mille iga liige tähistab sektoris ühte rida või veergu.

Rakendatud algoritmis võetakse vaatluse alla esiteks kolm järjestikust sektorit. Iga sektor kolmikus käiakse ükshaaval läbi. Kui leitakse, et sektoris on potensiaalne koht, kuhu arv asetada, vaadatakse, millise rea või veeru peale jääb antud potensiaalne koht. Selle tuvastamisel määratakse vastavat veergu või rida tähistav kahendmuutuja massiivi elemendi väärtus tõeseks. Igale sektorile on oma vastav kolme liikmeline massiiv ning kuna vaadatakse jaotisena kolme järjestikust sektorit, tekib jaotise ridade või veergude tähistamiseks 3x3 kahemõõtmeline massiiv. Algselt on iga massiivi element määratud vääraks. Antud operatsiooni korratakse iga kolmiku kohta, tehes igale kolmikule vastavad massiivide kogumid. Kuna kokku on kolm kolmikut, siis uurimise lõpuks on moodustunud 3x3x3 kolmemõõtmeline kahendmuutuja massiiv, mida kasutatakse välistamisel. Kohtade välistamise puhul võetakse järjest uurimise alla iga 3x3 osa massiivist. Võrreldakse ühemõõtmelisi massiive, mis tähistavad ridu või veerge, omavahel ning kui leidub kaks identset, siis kasutatakse nende tõese väärtusega elementidele vastavaid ridu või veerge, et välistada erinenud massiivile vastavast sektorist kohti.

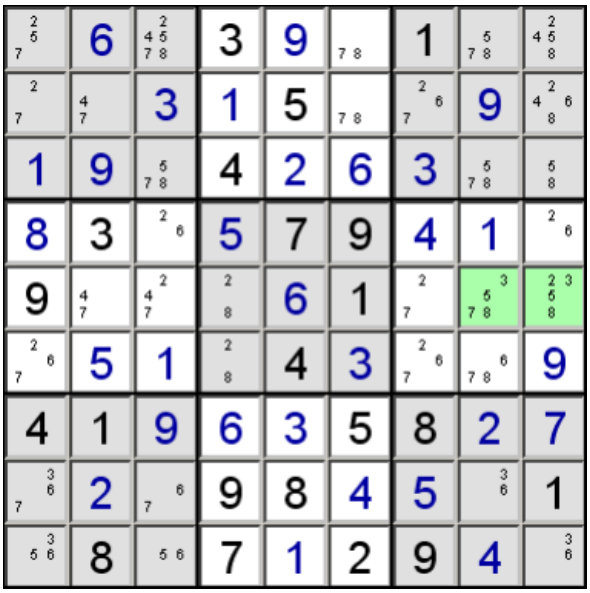


Joonis 11. Sektor/sektor suhtlus tegevusdiagramm.

2.2.3 Peidetud hulk

Antud reegli korral üritatakse algoritmis leida sektoris, reas või veerus kandidaate, mis sobiksid ainult kahte, kolme või nelja ruutu. Nende numbrite esinemised peavad kattuma ruuduti ning kui leitakse numbrid, mis vastavad sellele tingimusele, saab eemaldada nendest erinevate numbrite võimalikud esinemised antud ruutudest.

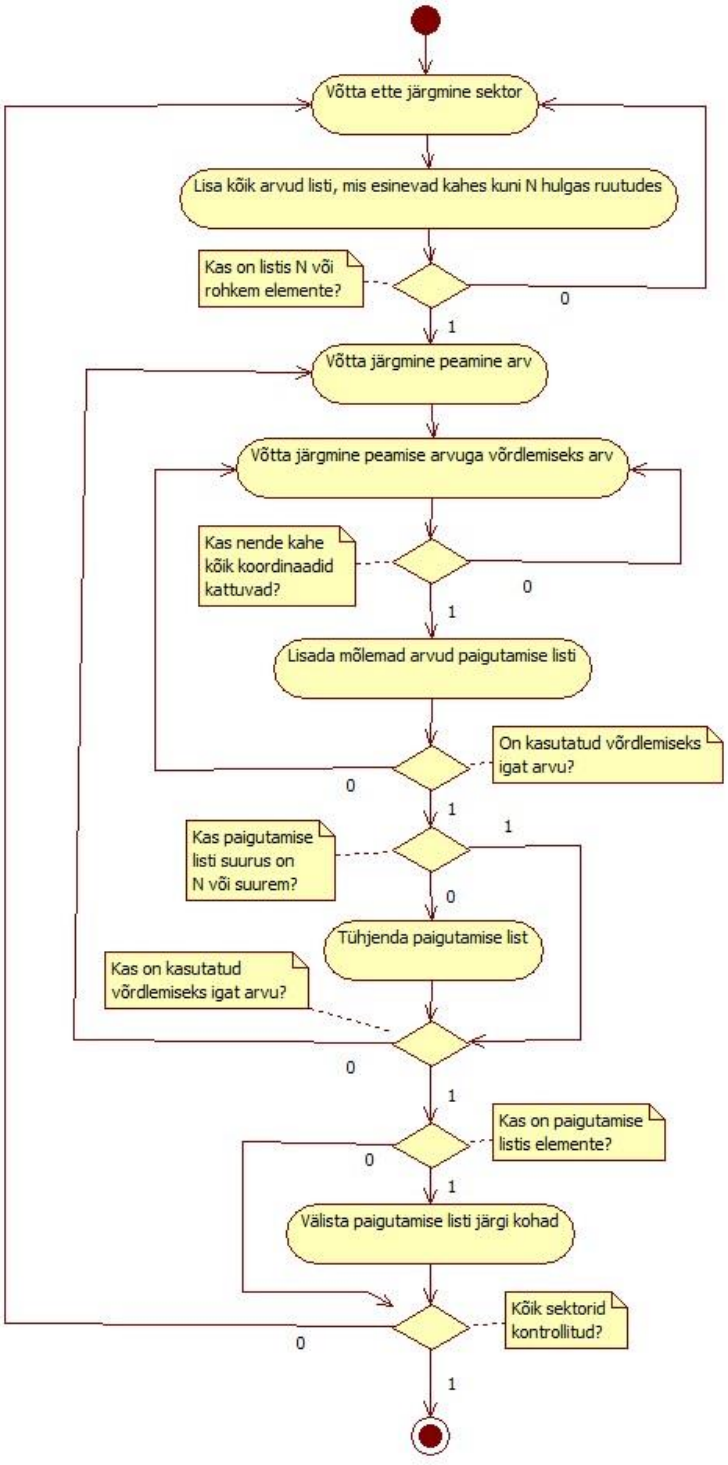
Antud näites, joonis10, on rohelisega märgitud, millistes ruutudes esineb ainult peidetud paar, mille moodustavad numbrid 3 ja 5. Kuna nende võimalikud kohad on ainult antud kahes ruudus, siis see tähendab, et nad ei saa kuskil mujal esineda ning sellepärast saab välistada ülejäänud numbrite võimalikud esinemised nendes kahes ruudus.



Joonis 12. Peidetud paar[3].

Koodis on rakendatud antud reegli loogika joonisel 13 asuva tegevusdiagrammi alusel. Jaotise puhul on viidatud üksikule sektorile, reale või veerule. N puhul viidatakse rakendatavale hulga loogikale. Näiteks kui rakendatakse paari loogikat, siis N on võrdne kahega, kui rakendatakse kolmiku loogikat, siis N on võrdne kolmega.

Algoritmi puhul võetakse ette jaotus ning käiakse see läbi, vaadates, kas leidub numbrit, mille puhul on võimalik N või vähem kohti. Kõik need numbrid pannakse listi. Juhul kui on vähem elemente listis, kui on N suurus, tühjendatakse list ja minnakse järgmise jaotuse juurde. Kui on piisav hulk listi elemente, hakatakse elemente omavahel võrdlema ja kui võrreldavate numbrite potensiaalsete asetuskohtade koordinaadid kattuvad, pannakse elemendid uute listi. Juhul kui uues listis on N hulgast väiksem arv elemente, tühjendatakse list. Võrreldakse kõik koordinaadid omavahel ning peale võrdlemise lõppu kasutatakse uut listi, et välistada mittesobivate numbrite esinemist antud leitud ruutudes.



Joonis 13. Peidetud hulga tegevusdiagramm.

2.2.4 Nähtavad hulgad

Nähtava hulkade reeglite puhul otsitakse ruute, kus esineb ainult kaks, kolm või neli kandidaati. Sellega saab välistada nende kandidaatide esinemise väljaspool neid ruute. Antud reegel on väga sarnane peidetud hulkade reeglitega, kuid nende tegevus on

vastupidine. Nähtavate hulkade puhul seisavad kandidaadid üksinda kastides, millest tuleb ka nimi nähtav, sest nad ei ole peidetud teiste kandidaatide keskele. Kuid peidetud hulkade puhul on kandidaadid peidetud teiste keskele ning üritatakse mitte eemaldada teistest ruutudest leitud üksindaseisvaid kandidaate, vaid hoopis üritatakse enda ruutudest eemaldada liigseid kandidaate.

Nähatava kolmiku puhul on kõige lihtsam esinemine kujul 3/3/3, see tähendab, et igas ruudus on kolm identset numbrit. Kuid lisaks sellele juhtumile on veel 3 variatsiooni:

- 3/3/2
- 3/2/2
- 2/2/2

3/X/X, kus X on kas 2 või 3, variatsioonid on rakendatud algoritmis ühe loogikana, kuid 2/2/2 jaoks on tehtud eraldi loogika.

Antud näite, joonis11, puhul moodustavad nähtava paari numbrid 2 ja 3, märgitud rohelisega. Tänu sellele on võimalik eemaldada kahe potensiaalsed esinemised sektoris, mis on tähistatud kollasega ning veerus, mis on tähistatud sinisega.

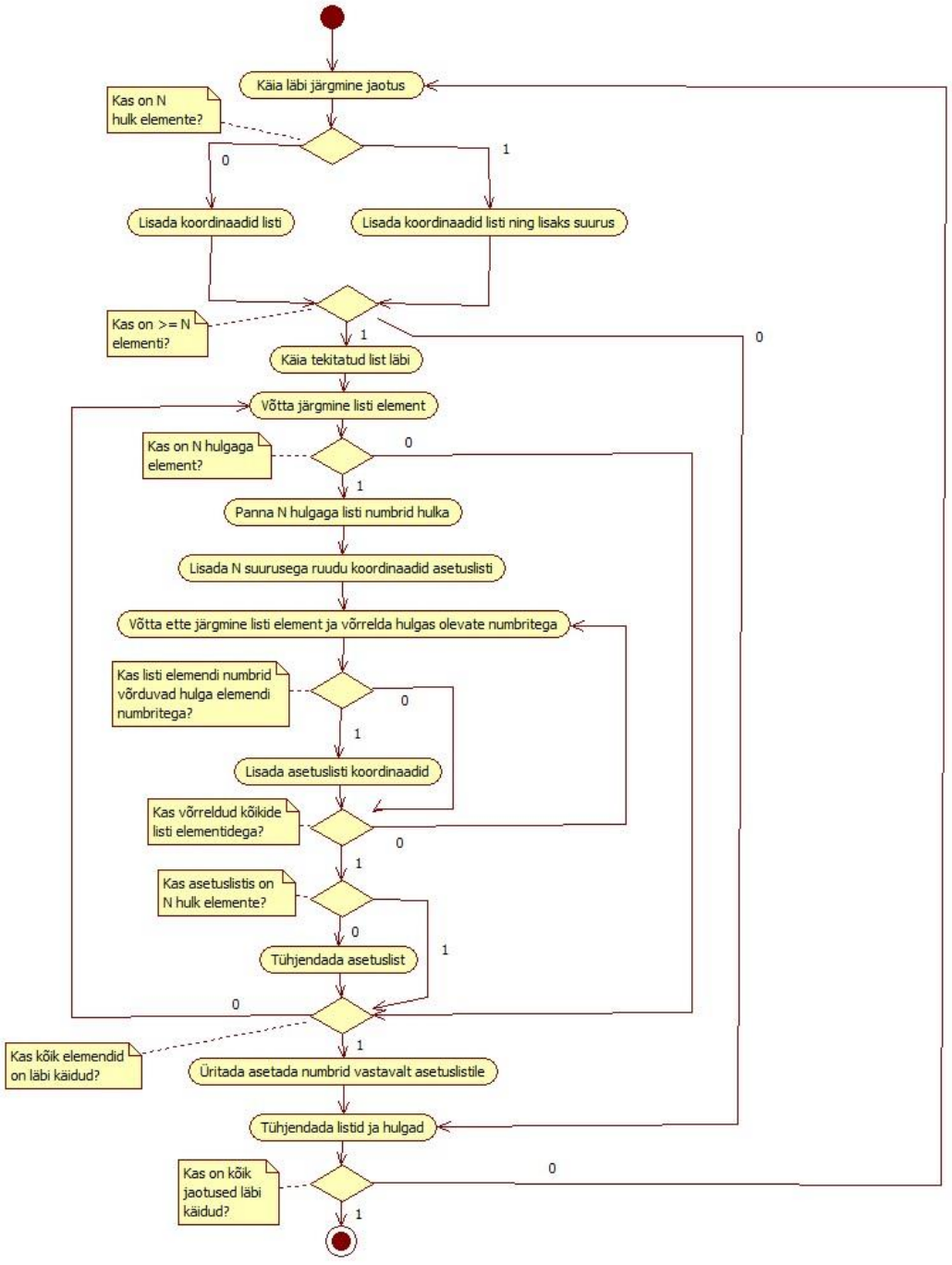
3	1 2 6 8	5	2 6 8	2 4 6	2 4 6 8	7	1 2 4	9
1 2 4	9	1 8	3	7	2 4 8	2 4 5 8	1 2 4 5	6
2 4	7	6 8	2 5 6 8 9	2 4 5 6 9	1	2 4 5 8	3	2 8
6	1 2 5 8	4	2 7 8 9	1 2 9	2 8 9	3	1 2 5 9	1 2 7 8
1 2 7	1 2 3 8	1 3 7 8	4	1 2 3 6 9	5	2 6 8	1 2 6 9	1 2 7 8
1 2 5 7	1 2 3 5 8	9	2 6 7 8	1 2 3 6	2 3 6 8	2 5 6 8	1 2 5 6	4
5 7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

Joonis 14. Nähtav paar[4].

Koodis on antud reegli kohta rakendatud kaks erinevat loogikat. Üks nendest loogikatest katab ära kõik 3/X/X juhtumid, kus X on 2 või 3 ning teine loogika katab ära 2/2/2 juhtumi. Järgnevalt on esimesena toodud välja 3/X/X variatsooni loogikat rakendav tegevusdiagramm. Jaotuse puhul on viidatud üksikule sektorile, reale või

veerule. N puhul on viidatud hulgale, mida rakendatakse. Nähtava paari puhul on N võrdne kahega, nähtava kolmiku puhul N on võrdne kolmega.

Algoritmis käiakse läbi jaotuse kõik ruudud ning leitakse kõik ruudud, millede suurus on N või väiksem ning lisatakse listi. Kui hulk on võrdne N-ga, siis lisatakse kolmas element antud ruudu listi, et kergesti pärast ülesse leida. Kui on piisav hulk elemente listis, käiakse see läbi. Võrreldakse uuritava ruudu potensiaalseid numbreid kombinatsiooni numbritega. Kombinatsioon on numbrite kolmik, mille põhjal rakendatakse loogikat. Juhul, kui leitakse, et kõik numbrid sisalduvad kombinatsioonis, pannakse ruut koos võrreldava ruuduga listi. Kui võrdlemise lõpuks on leitud N hulk elemente, siis jäetakse list alles, muidu kustutatakse. Hiljemalt kasutatakse listi, et välistada kohti.

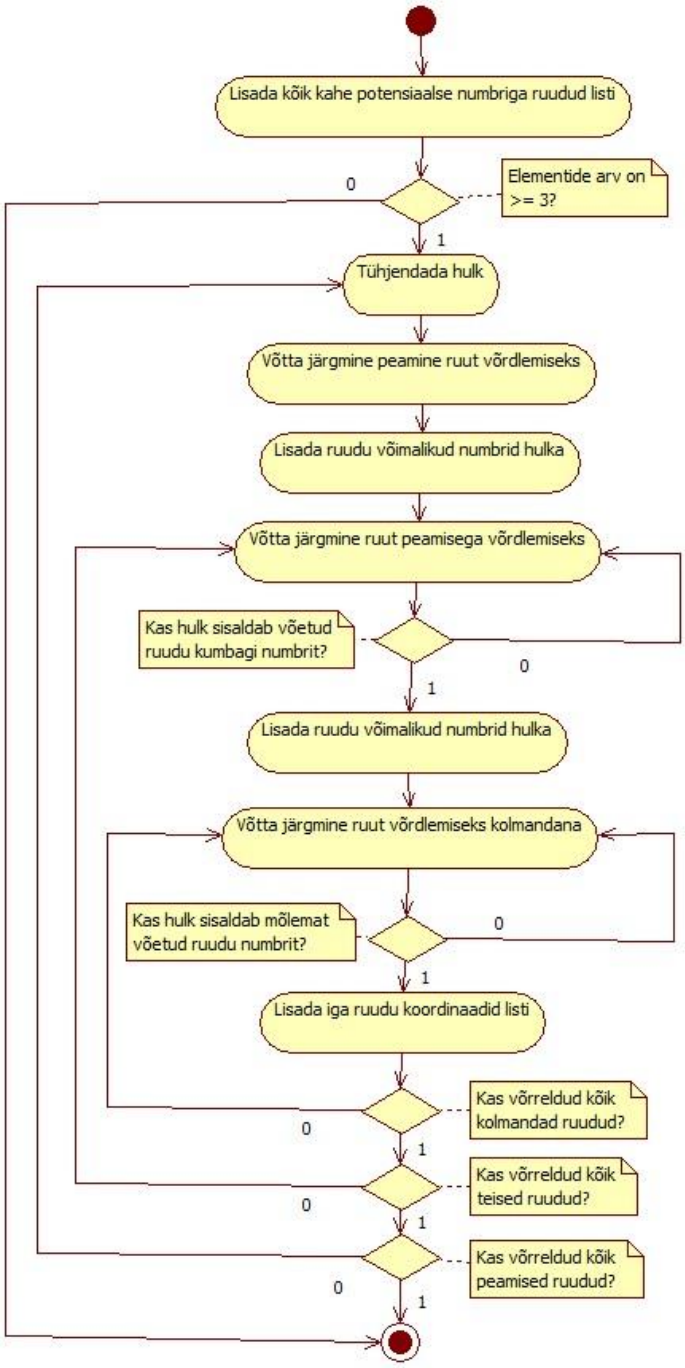


Joonis 15. Nähtavate hulkade tegevusdiagramm.

Järgnevalt on välja toodud 2/2/2 loogikat rakendava osa tegevusdiagramm joonisel 16. Antud reegli puhul võetakse uurimise alla kõik kahe liikmega ruudud. Nad pannakse listi, et pärast oleks kergem neid üksteisega võrrelda. Reegli toimimise jaoks on vaja kolme ruutu. Ükski ruut ei sisalda täielikku hulka vajaminevatest arvudest ehk siis vajaminev kombinatsioon on 3 numbrit aga ühes ruudus pole rohkem kui 2 numbrit. Kuid igal ruudul on mõlema teise ruuduga üks ühine number. Ehk siis kui võetaks

esimesed kaks ruutu, saaks nende kahe potensiaalsetest numbritest kokku panna vajamineva kolmiku kombinatsiooni.

Antud algoritmis võetakse esimene ruut, lisatakse hulka, mis hakkab endas hoidma vajaminevat numbrite kolmikut, siis üritatakse leida teine arv, millel oleks esimese ruudu arvudega vähemalt üks kokkulangev number. Taolise ruudu leidmisel lisatakse leitud ruudu arvud samuti hulka. Kui oli leitud õige arv, siis sellega saab kokku vajamineva kolmese hulga. Kolmanda ruudu leidmiseks peavad kolmanda koha mõlemad arvud sisalduma hulgas. Kui leitakse taoline ruut, lisatakse kõik kolm uude listi, mida kasutatakse hiljemalt kohtade välistamiseks.



Joonis 16. Nähtava paari 2/2/2 tegevusdiagramm.

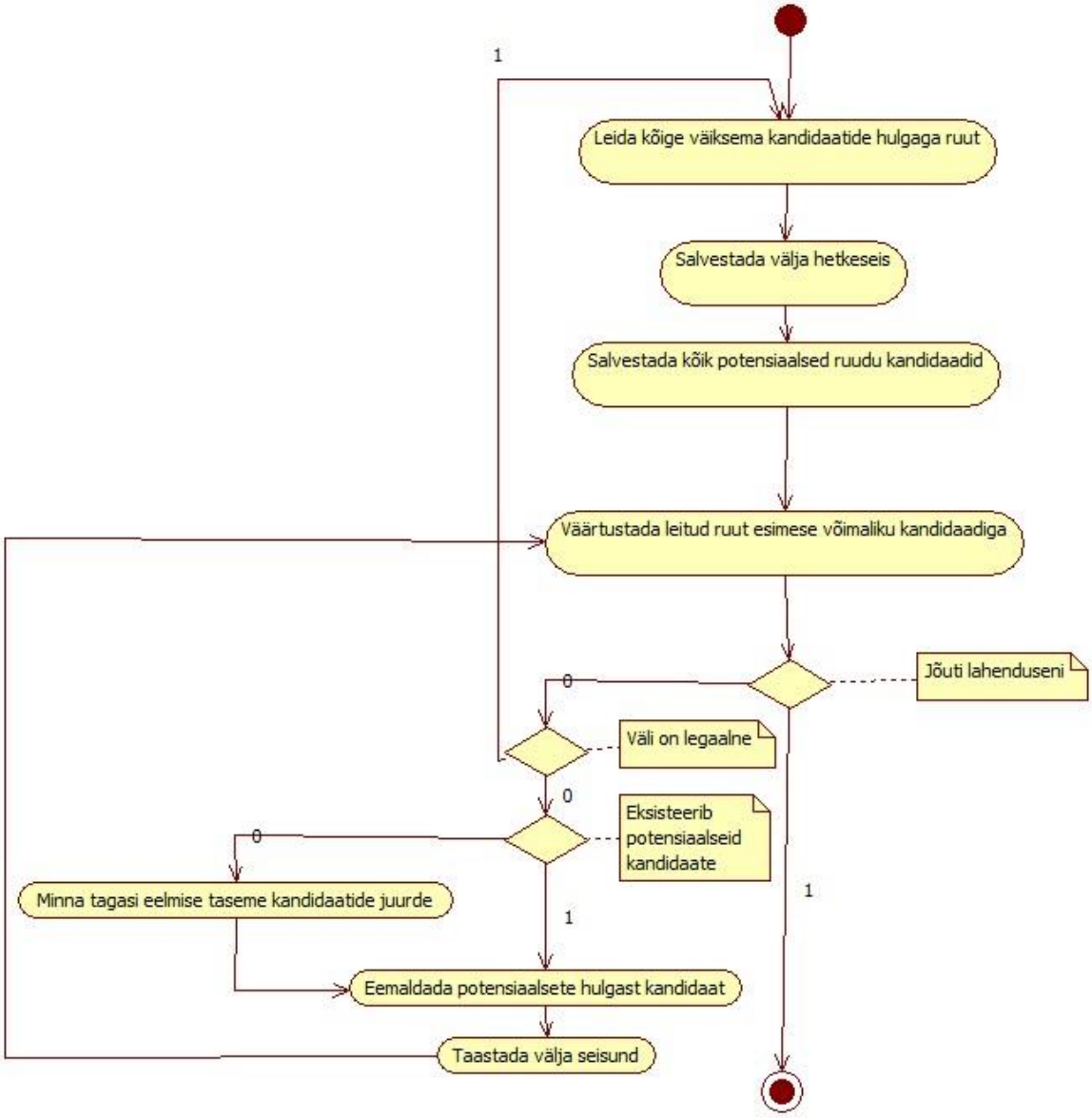
2.3 Tagurdamine

Tagurdamine on üks toore jõuga lahendamise meetodeid. Sellega on võimalik alati leida lahendus, kui see peaks eksisteerima. Tavaliselt kasutatakse taolist lähenemist arvutuslike probleemide lahendamiseks, milledele on seatud teatud piirangud. Kuid kui ülesandel on suur hulk võimalikke lahendusi, siis võib tagurdamise kasutamine võtta

ebapraktiliselt kaua aega. Nagu on ka hiljemalt näha võtab sudoku puhul tagurdamine seda rohkem aega, mida vähem on vihjeid.

Kui reeglite järgi lahendamisega ei jõuta lahendini, siis üritab programm kasutada tagurdamist. Tagurdamise käigus otsitakse ülesse kõige väiksemate potensiaalsete numbritega ruut, sest taolist lähenemist kasutades on vaja tagurdamist kõige vähem sooritada. Samuti on tõenäosus kõrgem, et lisatakse õige number. Leitakse esimene taoline ruut ning proovitakse sinna erinevaid arve. Peale arvu asetamist üritatakse lahendada sudokut jälle loogika abil. Kuna on valitud proovimiseks ruut, kus on kõige vähem kandidaate, siis töötab tagurdamine kiiremini, kui juhul, kus on valitud ruut rohkemate kandidaatidega.

Koodis on rakendatud tagurdamine kahes erinevas kohas – lahendamisel ja unikaalse lahendi otsimisel. Kummalgi nendest juhtudest ei ürita algoritm leida kõiki võimalikke lahendeid, sest antud programmi töö jaoks ei ole see vajalik. Lahendamise puhul, juhul kui on vaja välja kutsuda tagurdamist, üritab kood leida ainult esimese võimaliku lahenduse. Unikaalasuse kontrollis otsib algoritm kuni 2 lahendust ning peale seda katkestab oma töö, sest on juba tõestatud, et sudoku ei oli unikaalne.



Joonis 17. Tagurdusalgoritmi otsustusdiagramm

3. Tarkvaraline realisatsioon

3.1 Programmi struktuur

Programmi valmistamisel loodi mitmeid erinevaid meetodeid. Programmi põhitsükkel omab ise vähe funktsionaalsust. Tema peamine ülesanne on kutsuda välja erinevaid funktsioone, mis rakendavad erinevaid vajalikke loogikaid.

Programmi põhiliseks funktsiooniks on runIt, mis rakendab kogu lahendamise loogikat. Sellega kaasneb ka ükshaaval lahendamine ja vihje küsimine. Funktsiooni checkForLegalAndUnique abil kontrollitakse kas sudokul on unikaalne lahend.

Järgnevalt on kirjeldatud programmi põhilisi meetodeid.

1. runIt()
   Ta peamine ülesanne on jooksutada tsüklit, mis kutsub välja sudoku lahendamise loogikat nii kaua, kuni on võimalik lisada numbreid ning peale seda kutsuda välja tagurdamist, kui pole sudoku veel lahendatud.
2. singlesCheck()
   Kontrollib, kas leidub üksikuid peidetud või nähtavaid, mida saaks panna kirja.
3. placementRules()
   Üritab panna sudokuväljale paika numbreid, kui on selleks võimalus. Käib läbi järjest read, veerud ning sektorid.
4. insertablePos()
   Rakendab sudokule kõige tavalisemat välistamise loogikat.
5. singlesBoxColumn()
   Rakendab sektor/veerg või rea loogikat.
6. hiddenGroupsFull()
   Rakendab peidetud gruppide loogikat, koos tema variatsioonidega.
7. nakedGroups()
   Rakendab nähtavate gruppide loogikat, koos tema variatsioonidega.
8. boxLineReduction()
   Rakendab sektor/sektor loogikat.
9. backTrackingAdd()

Liigub järgmisele tagurdamise tasemele, kui eelmisega ei olnud võimalik lahendust leida.

10. checkForLegalAndUnique
    Kontrollib, kas on tegu unikaalse lahendusega sudokuga ning kas on andmed sisestatud korrektselt.

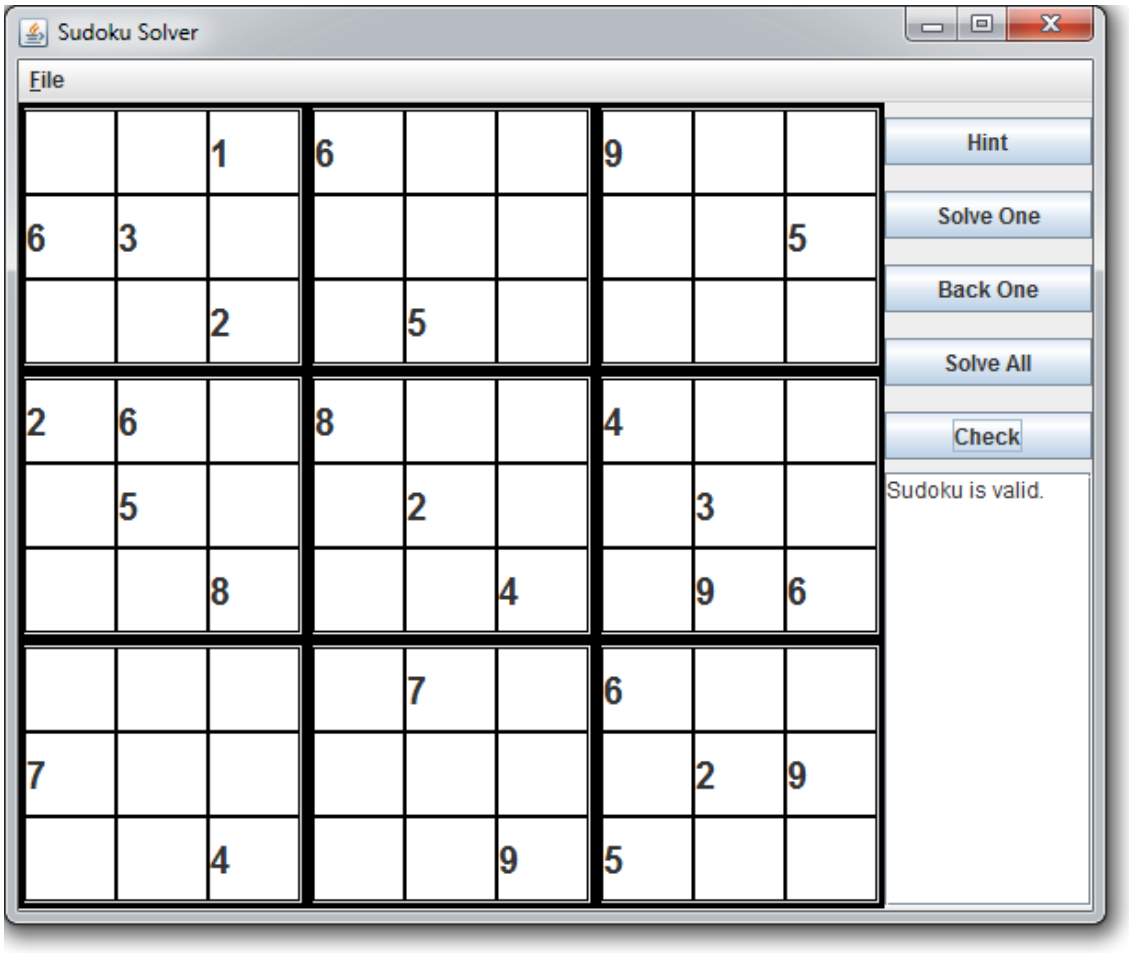
3.2 Failiformaat

Antud sudoku programmi realisatsioonis on iga sudoku salvestatud eraldi faili. Niimoodi on kergem leida ülesse spetsiifilisi sudokusid, kuid samas, kuna igal sudokul on enda fail, siis kui kasutaja salvestab suurel hulgal sudokusi, võib muutuda faile hoidev kaust korratuks. Sudokuväli on salvestatud faili kujul, kus iga ruuduväärtus on omaette real. Tühjad kohad on esindatud nullidena. Failid, kuhu sudokud salvestatakse, on .sud laiendusega.

Lõik faili formaadist:

```
...
6
0
0
0
0
0
7
0
3
0
1
0
0
...
```

4. Graafiline kasutajaliides



Joonis 18. Graafiline kasutajaliides

Rakendus kooseb erinevatest paneelidest. Ülaosas asub paneel, kust on võimalik andmeid kas salvestada või laadida. Menüü sisaldab selle jaoks vastavalt elemente „Save“ ja „Load“. Tegevuste sooritamisel rakendatakse failifilter, mis näitab ainult .sud pikendusega faile. Elementi valides tuleb ette aken, millega on võimalik lehitseda kataloogide vahel ja leida soovitud fail laadimise puhul. Faili valiku kinnitades laaditakse sudoku väljale. Salvestamise valikul on võimalik samamoodi otsida soovitud kataloog ning sinna salvestada .sud pikendusega fail ise valitud nimega.

Lahendusväli on tehtud üheksast klassist, 3x3 moodustises. Sellest omakorda iga klass koosneb üheksast ruudust, asetatud 3x3 formatsiooni. Väljal on võimalik näha sudoku hetkeseisu.

Paremal asuv paneel koosneb viiest nupust ja teksti alast. Viis nupu valikut on „Hint“, „Solve One“, „Back One“, „Solve All“, „Check“. Nupp „Hint“ lubab kasutajal sellele vajutades näha, millisesse ruutu oleks järgmisena võimalik lahendusloogika järgi asetada number. „Solve One“ annab võimaluse lahendada etteantud sudokut sammhaaval. Iga vajutusega lahendatakse sudoku üks ruut. „Back One“ puhul on võimalik liigutada välja lahendamise progressiooni ühe käigu võrra tagasi. Iga vajutusega võetakse lahendamist ühe ruudu võrra tagasi. „Solve All“ lahendab terve sudoku järjest ning väljastab lahendusväljale lahendatud kombinatsiooni. „Check“ nuppuga saab kontrollida, kas sisestatud sudoku andmed oli korrektsed ning kas antud sudoku variandil leidub ainult üks lahend. Nuppude all on samal paneelil tekstiväli, kuhu väljastatakse, millisele ruudule lisati number ning millise reegli põhjal. Põhimõtteliselt näidatakse selles tekstiväljas algoritmi lahenduskäiku.

5. Analüüs

Töö käigus sai testitud sudoku lahendamise algoritmi effektiivsust erineva tasemete sudokude peal. Testimiseks valitud erinevad sudokud võeti internetis olevast kogumikust[5]. Lisaks on tabel võrdlemaks lahendamiskiirust sudokude puhul, mis on vähimate võimalike vihjete arvuga[6], [7]. On tõestatud, et sudoku ei saa olla vähema, kui 17 vihjega ja olla siiski unikaalse lahendiga[8].

Selgus, et peamiselt ei sõltunud sudoku raskusastmest lahendamise aeg. Kuid siiski ekstreemse raskuse puhul oli näha lahendamisaja tõusu. See juhtus arvatavasti sellepärast, et sudoku oli ehitatud ülesse lahendamiseks reeglite järgi, mida polnud koodis realiseeritud ning sellest tulenevalt pöörduti rohkem tagurdamise poole. Lisaks oli näha ka arvestatavaid erinevusi sama raskusega sudokude lahendamisel. Selle peamine põhjus võis olla millises järjekorras rakendati lahendusreegleid. Reeglite rakendamise järjekorrast sõltub mitu korda korratakse tsükklit.

Teisest tabelist selgus, et mida väiksem hulk vihjeid on algselt antud sudoku lahendamiseks, seda aeglasemalt tegutseb tagurdamise loogika. Antud tabeli tagurdamise ajad leiti kasutades Internetist võetud loogikat[8].

Seega saab järeldada, et sudoku lahendmisaeg sõltub ennekõike tema struktuurist. Alljärgnevalt on toodud välja tabelid, mille põhjal on otsused tehtud.

Tabel 1: Erinevate sudokude lahendamiseks kulunud ajad.

Number	Raskusaste	Vihjete arv	Algoritmi aeg sekundites
5	Kerge	34	0.021
10	Kerge	34	0.024
15	Kerge	34	0.029
20	Kerge	34	0.032
25	Kerge	34	0.023
30	Keskmine	32	0.040
35	Keskmine	32	0.048
40	Keskmine	32	0.039
45	Keskmine	32	0.030
50	Keskmine	32	0.024
55	Raske	26	0.040
60	Raske	26	0.022
65	Raske	26	0.031
70	Raske	26	0.038
75	Raske	26	0.029
80	Väga raske	24	0.044
85	Väga raske	24	0.041
90	Väga raske	24	0.035
95	Väga raske	24	0.030
100	Väga raske	24	0.032
105	Ekstreemne	24	0.062
110	Ekstreemne	24	0.062
115	Ekstreemne	24	0.063
120	Ekstreemne	24	0.057
125	Ekstreemne	24	0.054

Tabel 2: Tagurdamise ja algoritmi võrdlus.

Variant	Vihjete arv	Algoritmi aeg sekundites	Tagurdamise aeg sekundites
Allikas 6	17	0.038	6.16
Allikas 7	17	0.059	2.82

6. Kokkuvõte

Lõputöö käigus sai välja arendatud lihtne tarkvara, millega on võimalik lahendada etteantud sudoku mõistatus. Rakendus võiks leida kasutust inimeste seas, kes lahendavad sudokusi ning soovivad kontrollida enda lahendust või kontrollida, kas on mingi loogika rakendamine jäänud kahe silma vahele. Tulevikus oleks võimalik lisada rakendusse lisaks veel erinevate reeglite realisatsioone. Rakendusest on eemaldatud arenduse käigus mitmed ebastabiilsused ning programm töötab ilma varasemalt eksisteerinud tõrgeteta. Seega võib töö eesmärke lugeda õnnestunuks.

Kasutatud kirjandus

1. [WWW]http://www.vint.ee/sudoku-reeglid-ja-info/ (16.05.2015)
2. [WWW]http://www.afjarvis.staff.shef.ac.uk/sudoku/ (16.05.2015)
3. [WWW]http://www.sadmansoftware.com/sudoku/hiddensubset.php (16.05.2015)
4. [WWW]http://www.sadmansoftware.com/sudoku/nakedsubset.php (16.05.2015)
5. [WWW]http://www.websudoku.com/ (16.05.2015)
6. [WWW]http://en.wikipedia.org/wiki/File:Sudoku_puzzle_hard_for_brute_force.jpg (16.05.2015)
7. [WWW]http://theconversation.com/good-at-sudoku-heres-some-youll-never-complete-5234 (16.05.2015)
8. [WWW]http://www.technologyreview.com/view/426554/mathematicians-solve-minimum-sudoku-problem/ (16.05.2015)
9. [WWW] http://www.geeksforgeeks.org/backtracking-set-7-suduku/ (16.05.2015)