

SKRIPSI

PERBANDINGAN ALGORITMA *BACKTRACKING* DENGAN ALGORITMA *HYBRID GENETIC* UNTUK MENYELESAIKAN PERMAINAN CALCUDOKU



MICHAEL ADRIAN

NPM: 2013730039

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2017

UNDERGRADUATE THESIS

**COMPARISON OF THE BACKTRACKING ALGORITHM
AND THE HYBRID GENETIC ALGORITHM TO SOLVE THE
CALCUDOKU PUZZLE**



MICHAEL ADRIAN

NPM: 2013730039

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2017**

LEMBAR PENGESAHAN

PERBANDINGAN ALGORITMA *BACKTRACKING* DENGAN ALGORITMA *HYBRID GENETIC* UNTUK MENYELESAIKAN PERMAINAN CALCUDOKU

MICHAEL ADRIAN

NPM: 2013730039

Bandung, 20 Desember 2017

Menyetujui,

Pembimbing

Dr.rer.nat. Cecilia Esti Nugraheni

Ketua Tim Penguji

Anggota Tim Penguji

Rosa De Lima, M.Kom.

Claudio Franciscus, M.T.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PERBANDINGAN ALGORITMA *BACKTRACKING* DENGAN ALGORITMA *HYBRID GENETIC* UNTUK MENYELESAIKAN PERMAINAN CALCUDOKU

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 20 Desember 2017

Meterai
Rp. 6000

MICHAEL ADRIAN
NPM: 2013730039

ABSTRAK

Calcudoku adalah sebuah permainan teka-teki angka. Tujuan dari teka-teki ini adalah mengisi setiap sel dalam *grid* dengan angka 1 sampai n tanpa pengulangan angka dalam setiap kolomnya dan barisnya untuk *grid* berukuran $n \times n$. *Grid* ini dibagi menjadi sejumlah *cage* dengan setiap *cage* yang jumlah selnya bervariasi. Setiap *cage* dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel. Angka-angka dalam satu *cage* yang sama harus menghasilkan angka tujuan yang telah ditentukan jika dihitung menggunakan operasi matematika yang ditentukan. Angka-angka dalam satu *cage* juga boleh berulang, selama pengulangan tidak terjadi dalam satu kolom atau baris yang sama.

Dua algoritma telah terbukti berhasil dalam menyelesaikan Calcudoku, yaitu algoritma *backtracking* dan algoritma *hybrid genetic*.

Algoritma *backtracking* adalah sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya.

Algoritma *hybrid genetic* dalam kasus ini adalah gabungan dari algoritma *rule based* dan algoritma genetik. Algoritma *rule based* adalah sebuah algoritma berbasis aturan logika untuk menyelesaikan Calcudoku. Algoritma genetik adalah salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam yang mencari solusi dengan menggunakan operator-operator genetik seperti mutasi, kawin silang, dan *elitism*.

Perangkat lunak algoritma *backtracking* dapat menyelesaikan semua permainan yang diujikan. Tetapi pada ukuran *grid* yang besar, algoritma *backtracking* sangat lambat dalam menyelesaikan permainan. Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan karena sifat acak dari algoritma *hybrid genetic* ini. Semakin besar ukuran *grid*, maka kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan semakin besar. Pada ukuran *grid* yang kecil, algoritma *hybrid genetic* cenderung menyelesaikan permainan lebih lambat daripada algoritma *backtracking*. Tetapi pada ukuran *grid* yang besar, algoritma *hybrid genetic* gagal dalam menyelesaikan permainan, sehingga performansinya tidak bisa dibandingkan dengan algoritma *backtracking*. Banyaknya sel yang diisi pada tahap algoritma *rule based* dan nilai untuk parameter-parameter algoritma genetik mempengaruhi kecepatan dan tingkat keberhasilan algoritma *hybrid genetic* dalam menyelesaikan permainan.

Kata-kata kunci: Calcudoku, algoritma *backtracking*, algoritma *hybrid genetic*, algoritma *rule based*, algoritma genetik

ABSTRACT

Calcudoku is a number puzzle game. The goal of this puzzle is to fill each cell in the grid with the numbers from 1 to n without repetitions of the numbers in each column and row for grid with the size of $n \times n$. The grid is divided into a number of cages, with each cage contains a variable number of cells. Each cage is bordered with a thicker line than cell border line. Numbers in the same cage must produce the predetermined target number if calculated using the predetermined mathematical operation. Numbers in a cage can be repeated, as long as the repetitions do not occur in the same column or row.

Two algorithms have been proven to successfully solve Calcudoku. The two algorithms are the backtracking algorithm and the hybrid genetic algorithm.

The backtracking algorithm is a general algorithm with finds a solution by trying one of several choices, if the choice proves to be incorrect, the computation restarts at the point of choice and tries another choice.

In this case, the hybrid genetic algorithm is a combination of the rule based algorithm and the genetic algorithm. Rule based algorithm uses logical rules to solve Calcudoku. Genetic algorithm is a heuristic technique inspired by the process of natural selection, which tries to find a solution by relying on genetic operators such as mutation, crossover, and elitism.

The backtracking algorithm successfully solved all puzzles. But on large grids, the algorithm is very slow in solving the puzzle. There is a chance that the hybrid genetic algorithm failed in solving the puzzle due to the random nature of the algorithm. The larger the grid, the higher the chance that the algorithm will fail in solving the puzzle. On smaller grids, the hybrid genetic algorithm tends to solve the puzzle slower than the backtracking algorithm. But on larger grids, the hybrid genetic algorithm failed to solve the puzzle, so its performance cannot be compared with the performance of the backtracking algorithm. The number of cells filled during the rule based algorithm phase and the values of the genetic algorithm parameters influences the speed and the success rate of the hybrid genetic algorithm in solving the puzzle.

Keywords: Calcudoku, backtracking algorithm, hybrid genetic algorithm, rule based algorithm, genetic algorithm

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

KATA PENGANTAR

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exercitation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius. Claritas est etiam processus dynamicus, qui sequitur mutationem consuetudium lectorum. Mirum est notare quam littera gothica, quam nunc putamus parum claram, anteposuerit litterarum formas humanitatis per seacula quarta decima et quinta decima. Eodem modo typi, qui nunc nobis videntur parum clari, fiant sollemnes in futurum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bandung, Desember 2017

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xxi
DAFTAR TABEL	xxv
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	3
1.3 Tujuan	3
1.4 Batasan Masalah	4
1.5 Metodologi Penelitian	4
1.6 Sistematika Pembahasan	5
2 LANDASAN TEORI	7
2.1 Calcudoku [1] [2]	7
2.2 Algoritma <i>Backtracking</i> [1]	9
2.3 Algoritma <i>Hybrid Genetic</i> [2]	16
2.3.1 Algoritma <i>Rule Based</i>	16
2.3.2 Algoritma Genetik	17
2.3.3 Algoritma <i>Hybrid Genetic</i>	18
3 ANALISIS	23
3.1 Analisis Algoritma <i>Backtracking</i>	23
3.2 Analisis Algoritma <i>Hybrid Genetic</i>	28
3.2.1 Algoritma <i>Rule Based</i>	29
3.2.2 Algoritma Genetik	29
3.3 Perangkat Lunak	48
3.3.1 Diagram <i>Use Case</i> dan Skenario	48
3.3.2 Diagram Kelas	53
3.3.3 Diagram <i>Sequence</i>	54
4 PERANCANGAN	67
4.1 Perancangan Masukan	67
4.2 Perancangan Keluaran	68
4.3 Perancangan Antarmuka	68
4.4 Diagram Kelas	70
4.4.1 Kelas Grid	71
4.4.2 Kelas Cage	75
4.4.3 Kelas Cell	77
4.4.4 Kelas SolverBacktracking	78

4.4.5	Kelas SolverHybridGenetic	79
4.4.6	Kelas SolverRuleBased	80
4.4.7	Kelas SolverGenetic	84
4.4.8	Kelas Chromosome	87
4.4.9	Kelas ChromosomeComparator	88
4.4.10	Kelas Controller	88
4.4.11	Kelas Calcudoku	89
4.4.12	Kelas WindowListener	92
4.4.13	Kelas PuzzleFileFilter	92
4.4.14	Kelas GUI	94
4.4.15	Kelas CellKeyListener	95
4.4.16	Kelas PopupMenuItemListener	96
4.4.17	Kelas CellTextFieldListener	97
4.4.18	Kelas GeneticParameters	98
5	IMPLEMENTASI DAN PENGUJIAN	101
5.1	Lingkungan untuk Pengujian	101
5.2	Implementasi	101
5.3	Pengujian Fungsional	102
5.4	Pengujian Keakuratan	113
5.5	Pengujian Algoritma	116
5.5.1	Pengujian Algoritma <i>Backtracking</i>	116
5.5.2	Pengujian Algoritma <i>Hybrid Genetic</i>	116
6	KESIMPULAN DAN SARAN	125
6.1	Kesimpulan	125
6.2	Saran	125
DAFTAR REFERENSI		127
A ANALISIS ALGORITMA <i>Backtracking</i>		129
B HASIL PENGUJIAN		143
B.1	Algoritma <i>Backtracking</i>	143
B.2	Algoritma <i>Hybrid Genetic</i>	148
C File TEKS SOAL-SOAL PERMAINAN CALCUDOKU UNTUK PENGUJIAN		159
C.1	File Teks Soal-Soal Permainan Calcudoku dengan <i>Grid</i> Berukuran 4×4	160
C.2	File Teks Soal-Soal Permainan Calcudoku dengan <i>Grid</i> Berukuran 5×5	170
C.3	File Teks Soal-Soal Permainan Calcudoku dengan <i>Grid</i> Berukuran 6×6	177
C.4	File Teks Soal-Soal Permainan Calcudoku dengan <i>Grid</i> Berukuran 7×7	188
C.5	File Teks Soal-Soal Permainan Calcudoku dengan <i>Grid</i> Berukuran 8×8	193
C.6	File Teks Soal-Soal Permainan Calcudoku dengan <i>Grid</i> Berukuran 9×9	198
D KODE PROGRAM		201
D.1	Grid.java	201
D.2	Cell.java	206
D.3	Cage.java	206
D.4	SolverBacktracking.java	208
D.5	SolverHybridGenetic.java	209
D.6	SolverRuleBased.java	210
D.7	SolverGenetic.java	226
D.8	Chromosome.java	229

D.9 Controller.java	230
D.10 Calcudoku.java	231
D.11 GUI.java	235
D.12 GeneticParameters.java	241

DAFTAR GAMBAR

1.1	Contoh permainan teka-teki Sudoku dengan solusinya	1
1.2	Contoh permainan teka-teki Calcudoku dengan penjelasan tentang elemen-elemen dari teka-teki ini [2]	2
2.1	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 4 x 4 yang belum diselesaikan. [1]	8
2.2	Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 2.1 [1]	8
2.3	Ilustrasi <i>State space tree</i> yang digunakan dalam algoritma <i>backtracking</i> [1]	11
2.4	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 3 x 3 [1]	12
2.5	Ilustrasi <i>state</i> 3, 4, dan 5 pada sebuah <i>grid</i> teka-teki Calcudoku [1]	13
2.6	Ilustrasi <i>state</i> 19 pada sebuah <i>grid</i> teka-teki Calcudoku [1]	14
2.7	<i>State</i> 25, simpul tujuan, sebagai hasil yang dicapai [1]	14
2.8	<i>State space tree</i> yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 2.4 [1]	15
2.9	Contoh bagaimana cara mendeteksi aturan <i>naked pair</i> [2]	16
2.10	Contoh aturan <i>evil twin</i> [2]	17
2.11	Contoh aturan <i>hidden single</i> [2]	17
2.12	Contoh aturan <i>killer combination</i> untuk <i>cage</i> dengan ukuran 2 sel dengan operasi matematika penjumlahan [2]	17
2.13	Contoh aturan <i>X-wing</i> [2]	18
2.14	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 6 x 6 [2]	19
2.15	Contoh proses kawin silang antara dua kromosom [2]	20
2.16	Contoh proses mutasi [2]	20
2.17	Alur penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma <i>hybrid genetic</i> [2]	21
3.1	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 4 x 4 yang belum diselesaikan, seperti yang digambarkan pada Gambar 2.1. [1]	23
3.2	<i>State</i> 4	24
3.3	<i>State</i> 11	24
3.4	<i>State</i> 12	25
3.5	<i>State</i> 17	25
3.6	<i>State</i> 18	25
3.7	<i>State</i> 19	26
3.8	<i>State</i> 23	26
3.9	<i>State</i> 93	26
3.10	<i>State space tree</i> yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.1	27
3.11	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 6 x 6 yang belum diselesaikan, seperti yang digambarkan pada Gambar 1.2. [2]	28
3.12	Permainan teka-teki Calcudoku setelah diselesaikan dengan algoritma <i>rule based</i>	29
3.13	Kromosom 1 dalam Generasi ke-1	32
3.14	Kromosom 2 dalam Generasi ke-1	32

3.15 Kromosom 3 dalam Generasi ke-1	32
3.16 Kromosom 4 dalam Generasi ke-1	33
3.17 Kromosom 5 dalam Generasi ke-1	33
3.18 Kromosom 6 dalam Generasi ke-1	33
3.19 Kromosom 7 dalam Generasi ke-1	34
3.20 Kromosom 8 dalam Generasi ke-1	34
3.21 Kromosom 9 dalam Generasi ke-1	34
3.22 Kromosom 10 dalam Generasi ke-1	35
3.23 Kromosom 11 dalam Generasi ke-1	35
3.24 Kromosom 12 dalam Generasi ke-1	35
3.25 Kromosom 1 dalam Generasi ke-2	38
3.26 Kromosom 2 dalam Generasi ke-2	38
3.27 Kromosom 3 dalam Generasi ke-2	38
3.28 Kromosom 4 dalam Generasi ke-2	39
3.29 Kromosom 5 dalam Generasi ke-2	39
3.30 Kromosom 6 dalam Generasi ke-2	39
3.31 Kromosom 7 dalam Generasi ke-2	40
3.32 Kromosom 8 dalam Generasi ke-2	40
3.33 Kromosom 9 dalam Generasi ke-2	40
3.34 Kromosom 10 dalam Generasi ke-2	41
3.35 Kromosom 11 dalam Generasi ke-2	41
3.36 Kromosom 12 dalam Generasi ke-2	41
3.37 Kromosom 1 dalam Generasi ke-3	44
3.38 Kromosom 2 dalam Generasi ke-3	44
3.39 Kromosom 3 dalam Generasi ke-3	44
3.40 Kromosom 4 dalam Generasi ke-3	45
3.41 Kromosom 5 dalam Generasi ke-3	45
3.42 Kromosom 6 dalam Generasi ke-3	45
3.43 Kromosom 7 dalam Generasi ke-3	46
3.44 Kromosom 8 dalam Generasi ke-3	46
3.45 Kromosom 9 dalam Generasi ke-3	46
3.46 Kromosom 10 dalam Generasi ke-3	47
3.47 Kromosom 11 dalam Generasi ke-3	47
3.48 Kromosom 12 dalam Generasi ke-3	47
3.49 Diagram <i>use case</i> untuk perangkat lunak permainan teka-teki Calcudoku	49
3.50 Diagram kelas untuk perangkat lunak permainan teka-teki Calcudoku	53
3.51 Diagram <i>sequence</i> saat perangkat lunak dibuka	54
3.52 Diagram <i>sequence</i> saat menu item "Load Puzzle File" dalam menu "File" dipilih (1)	55
3.53 Diagram <i>sequence</i> saat menu item "Load Puzzle File" dalam menu "File" dipilih (2)	56
3.54 Diagram <i>sequence</i> saat file yang ingin dibuka dalam <i>file chooser</i> dipilih (1)	57
3.55 Diagram <i>sequence</i> saat file yang ingin dibuka dalam <i>file chooser</i> dipilih (2)	58
3.56 Diagram <i>sequence</i> saat file permainan yang sudah dipilih dibuka	59
3.57 Diagram <i>sequence</i> saat menu item "Reset Puzzle" dalam menu "File" dipilih	60
3.58 Diagram <i>sequence</i> saat menu item "Close Puzzle File" dalam menu "File" dipilih	61
3.59 Diagram <i>sequence</i> saat menu item "Check Puzzle File" dalam menu "File" dipilih	61
3.60 Diagram <i>sequence</i> saat menu item "Backtracking" dalam menu "Solve" dipilih	62
3.61 Diagram <i>sequence</i> saat menu item "Hybrid Genetic" dalam menu "Solve" dipilih (1)	63
3.62 Diagram <i>sequence</i> saat menu item "Hybrid Genetic" dalam menu "Solve" dipilih (2)	64
3.63 Diagram <i>sequence</i> saat menu item "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih	65

3.64	Diagram sequence saat button "OK" dalam form "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih	65
3.65	Diagram sequence saat button "Cancel" dalam form "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih	66
3.66	Diagram sequence saat perangkat lunak ditutup	66
4.1	Contoh file masukan.	67
4.2	Perancangan GUI sebelum file permainan dibuka.	68
4.3	Perancangan GUI sesudah file permainan dibuka.	68
4.4	Perancangan GUI sesudah permainan berdasarkan file permainan yang dibuka diselesaikan.	69
4.5	Menu File	69
4.6	Menu Solve	70
4.7	Diagram kelas untuk perangkat lunak Calcudoku.	71
4.8	Diagram kelas Grid.	76
4.9	Diagram kelas Cage.	77
4.10	Diagram kelas Cell.	78
4.11	Diagram kelas SolverBacktracking.	79
4.12	Diagram kelas SolverHybridGenetic.	80
4.13	Diagram kelas SolverRuleBased.	85
4.14	Diagram kelas SolverGenetic.	87
4.15	Diagram kelas Chromosome.	88
4.16	Diagram kelas ChromosomeComparator.	88
4.17	Diagram kelas Controller.	90
4.18	Diagram kelas Calcudoku.	93
4.19	Diagram kelas WindowListener.	93
4.20	Diagram kelas PuzzleFileFilter.	93
4.21	Diagram kelas GUI.	96
4.22	Diagram kelas CellKeyListener.	96
4.23	Diagram kelas PopupMenuItemListener.	97
4.24	Diagram kelas CellTextFieldListener.	98
4.25	Diagram kelas GeneticParameters.	99
5.1	Antarmuka perangkat lunak saat pertama kali dibuka	102
5.2	Kotak dialog untuk memilih file permainan yang akan dibuka	103
5.3	Antarmuka perangkat lunak sesudah membuka file permainan yang dipilih	103
5.4	Kotak dialog untuk mengatur nilai dari parameter-parameter algoritma genetik	104
5.5	Antarmuka perangkat lunak setelah permainan berdasarkan file permainan yang telah dibuka diselesaikan	104
5.6	Kotak pesan error "Puzzle file not loaded"	104
5.7	Kotak pemilihan file permainan	105
5.8	Kotak dialog "Are you sure you want to load another puzzle file?"	105
5.9	Pesan error "Invalid puzzle file"	106
5.10	Pesan error "Invalid cages"	106
5.11	Pesan error "Error in loading puzzle file"	107
5.12	Pesan informasi "Congratulations, you have successfully solved the puzzle"	107
5.13	Kotak dialog "Are you sure you want to reset this puzzle?"	107
5.14	Pesan informasi "Row (nomor baris) has duplicate numbers"	108
5.15	Pesan informasi "Column (nomor kolom) has duplicate numbers"	108
5.16	Pesan informasi "Values of cells in the cage do not reach the target number"	108
5.17	Pesan informasi "There are cells with incorrect values in the grid"	109
5.18	Pesan informasi "There are empty cells in the grid"	109

5.19 Pesan informasi "The backtracking algorithm has successfully solved the puzzle"	110
5.20 Pesan informasi "The backtracking algorithm has failed to solve the puzzle"	110
5.21 Pesan informasi "Genetic algorithm parameters have not been set"	110
5.22 Pesan informasi "The hybrid genetic algorithm has successfully solved the puzzle"	111
5.23 Pesan informasi "The hybrid genetic algorithm has failed to solve the puzzle"	111
5.24 Form untuk mengatur nilai untuk parameter-parameter algoritma genetik	112
5.25 Pesan error "Invalid number format"	112
5.26 Kotak dialog "Are you sure you want to close this puzzle file?"	112
5.27 Kotak dialog "Are you sure you want to exit the application?"	113
5.28 File masukan untuk pengujian keakuratan	113
5.29 GUI permainan berdasarkan file masukan yang dapat dilihat pada Gambar 5.28	114
5.30 Solusi untuk permainan berdasarkan file masukan yang dapat dilihat pada Gambar 5.28	115
A.1 Contoh permainan teka-teki Calcudoku dengan ukuran grid 4 x 4 yang belum diselesaikan, seperti yang digambarkan pada Gambar 2.1. [1]	129
A.2 State 4	130
A.3 State 11	130
A.4 State 12	130
A.5 State 17	131
A.6 State 18	131
A.7 State 19	131
A.8 State 23	132
A.9 State 24	132
A.10 State 31	133
A.11 State 32	133
A.12 State 34	133
A.13 State 37	134
A.14 State 47	134
A.15 State 48	135
A.16 State 52	135
A.17 State 53	135
A.18 State 68	136
A.19 State 69	137
A.20 State 71	137
A.21 State 72	137
A.22 State 74	138
A.23 State 75	138
A.24 State 76	138
A.25 State 77	139
A.26 State 78	139
A.27 State 81	139
A.28 State 83	140
A.29 State 85	140
A.30 State 88	140
A.31 State 92	141
A.32 State 93	141

DAFTAR TABEL

3.1	Tabel parameter untuk algoritma genetik yang akan digunakan untuk menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.12	30
3.2	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1	31
3.3	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1	37
3.4	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-3	43
3.5	Skenario me-load file	50
3.6	Skenario memilih salah satu dari dua <i>solver</i> yang disediakan	50
3.7	Skenario me-reset permainan	50
3.8	Skenario meminta perangkat lunak untuk memeriksa permainan	51
3.9	Skenario menutup <i>file</i> masukan	51
3.10	Skenario menyelesaikan permainan dengan usahanya sendiri	52
3.11	Skenario mengatur nilai dari parameter-parameter untuk algoritma genetik	52
5.1	Lingkungan perangkat keras untuk pengujian perangkat lunak	101
5.2	Lingkungan perangkat lunak untuk pengujian perangkat lunak	102
5.3	Tabel jumlah soal permainan Calcudoku berdasarkan ukuran <i>grid</i>	116
5.4	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku	116
5.5	Nilai untuk parameter-parameter algoritma genetik untuk setiap percobaan yang dilakukan	117
5.6	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 1)	117
5.7	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 2)	118
5.8	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 3)	118
5.9	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 4)	118
5.10	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 5)	119
5.11	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 6)	119
5.12	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 7)	119
5.13	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 8)	120
5.14	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 9)	120
5.15	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 10)	120
5.16	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 11)	120
5.17	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 12)	121
5.18	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 13)	121
5.19	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 14)	121
5.20	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 15)	122
5.21	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 16)	122
5.22	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku sebanyak 16 skenario secara keseluruhan	122
B.1	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> 4×4	144
B.2	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> 5×5	145
B.3	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> 6×6	146
B.4	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> 7×7	147
B.5	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> 8×8	147

B.6 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 4×4 (Skenario 1-4)	149
B.7 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 4×4 (Skenario 5-8)	150
B.8 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 4×4 (Skenario 9-12)	151
B.9 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 4×4 (Skenario 13-16)	152
B.10 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 5×5 (Skenario 1-4)	153
B.11 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 5×5 (Skenario 5-8)	154
B.12 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 5×5 (Skenario 9-12)	155
B.13 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> 5×5 (Skenario 13-16)	156
B.14 Daftar jumlah sel yang berhasil diisi oleh algoritma <i>rule based</i> untuk Calcudoku dengan ukuran <i>grid</i> 4×4	157
B.15 Daftar jumlah sel yang berhasil diisi oleh algoritma <i>rule based</i> untuk Calcudoku dengan ukuran <i>grid</i> 5×5	158

BAB 1

PENDAHULUAN

Bab ini membahas tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan dari skripsi ini.

1.1 Latar Belakang

Calcudoku, atau dikenal juga sebagai KenKen, atau Mathdoku, adalah sebuah permainan teka-teki (*puzzle*) angka yang untuk menyelesaiakannya memerlukan perpaduan dari logika dan kemampuan aritmatika yang sederhana. Permainan ini adalah sebuah permainan teka-teki logika yang sederhana, namun, untuk menemukan solusinya cukup rumit, terutama untuk masalah yang lebih susah.

Teka-teki ini mirip dengan Sudoku. Sudoku adalah sebuah permainan teka-teki angka dengan *grid* berukuran n^2 , di mana dalam setiap baris, kolom, dan n^2 area yang berukuran $n \times n$ tidak boleh ada angka yang berulang, dengan n adalah ukuran area. Biasanya, $n = 3$, sehingga *grid* berukuran 9×9 , dan ada 9 area yang berukuran 3×3 . Contoh permainan teka-teki Sudoku dapat dilihat pada Gambar 1.1.

Persamaannya, tujuan dari teka-teki ini adalah mengisi setiap sel (*cell*) dalam (*grid*) dengan angka 1 sampai n tanpa pengulangan angka dalam setiap kolomnya dan barisnya untuk *grid* berukuran $n \times n$, dengan n adalah ukuran *grid*. Tidak ada angka yang boleh muncul lebih dari sekali dalam setiap baris atau kolom dalam *grid*.

Perbedaannya, jika pada Sudoku *grid* berukuran $n \times n$ dibagi menjadi n (*cage*) dengan setiap *cage* terdiri atas n sel, pada Calcudoku *grid* dibagi menjadi sejumlah *cage* yang jumlah selnya bervariasi. Setiap *cage* dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel. Angka-angka dalam satu *cage* yang sama harus menghasilkan angka tujuan yang telah ditentukan jika dihitung menggunakan operasi matematika yang telah ditentukan (penjumlahan, pengurangan, perkalian, atau pembagian). Angka-angka dalam satu *cage* juga boleh berulang, selama pengulangan tidak terjadi dalam satu kolom atau baris yang sama. Jika *cage* hanya berisi satu sel, maka satu-satunya kemungkinan jawaban untuk sel tersebut adalah angka tujuan dari *cage* tersebut. Angka tujuan dan operasi matematika dituliskan di sudut kiri atas *cage*. Pada awalnya, setiap sel dalam setiap *cage* dalam teka-teki ini kosong, belum terisi oleh angka-angka. *Border line* adalah garis pembatas

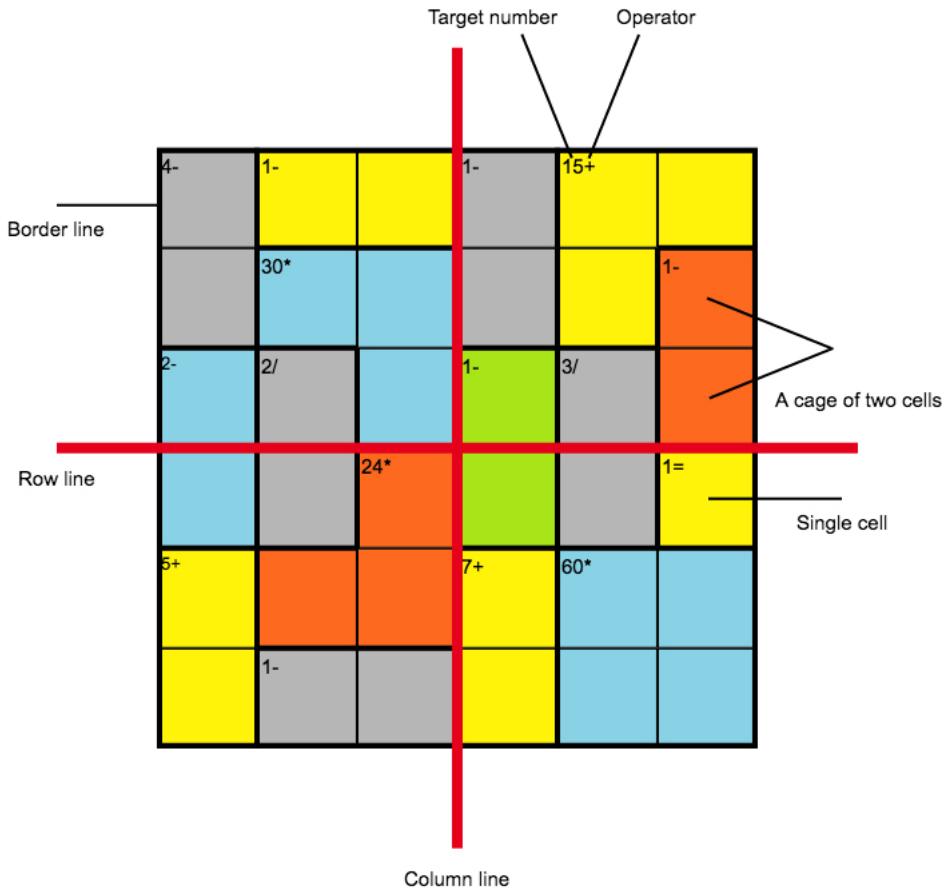
2	5	1	9
8		2	3
3		6	7
	1		6
5	4		1 9
	2		7
9		3	8
2		4	7
1	9	7	6

Unsolved Sudoku

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

Solved Sudoku

Gambar 1.1: Contoh permainan teka-teki Sudoku dengan solusinya



Gambar 1.2: Contoh permainan teka-teki Calcudoku dengan penjelasan elemen-elemen dari teka-teki ini [2]

terluar, *row line* adalah garis pembatas antar baris, dan *column line* adalah garis pembatas antar kolom. Gambar 1.2 menggambarkan contoh sebuah permainan teka-teki Calcudoku [1] [2].

Calcudoku dapat diselesaikan menggunakan beberapa algoritma. Skripsi ini membahas tentang penyelesaian Calcudoku menggunakan algoritma *backtracking* dan algoritma *hybrid genetic*, dan perbandingan performansi *performance* antara kedua algoritma tersebut dalam hal kecepatan dan kesuksesan dalam menyelesaikan Calcudoku.

Algoritma *backtracking* adalah sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya. Untuk bisa melacak kembali langkah-langkah yang telah dipilih, maka algoritma harus secara eksplisit menyimpan jejak dari setiap langkah yang sudah pernah dipilih, atau menggunakan rekursi (*recursion*). Rekursi dipilih karena jauh lebih mudah daripada harus menyimpan jejak setiap langkah yang pernah dipilih, hal ini menyebabkan algoritma ini biasanya berbasis DFS (*Depth First Search*) [1].

Algoritma *rule based* adalah sebuah algoritma berbasis aturan logika untuk menyelesaikan permainan teka-teki Sudoku dan variasinya, termasuk Calcudoku. Beberapa aturan logika yang digunakan dalam algoritma ini adalah *single square rule*, *naked subset rule*, *hidden single rule*, *evil twin rule*, *killer combination*, dan *X-wing*.

Pencarian heuristik adalah sebuah teknik pencarian kecerdasan buatan (*artifical intelligence*) yang menggunakan heuristik dalam langkah-langkahnya. Heuristik adalah semacam aturan tidak tertulis yang mungkin menghasilkan solusi. Heuristik kadang-kadang efektif, tetapi tidak dijamin akan berhasil dalam setiap kasus. Heuristik memerlukan peran penting dalam strategi pencarian karena sifat eksponensial dari kebanyakan masalah. Heuristik membantu mengurangi jumlah alternatif solusi dari angka yang bersifat eksponensial menjadi angka yang bersifat polinomial.

Contoh teknik pencarian heuristik adalah *Generate and Test*, *Hill Climbing*, dan *Best First Search*.

Algoritma genetik adalah salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam. Algoritma ini adalah perpaduan dari bidang biologi dan ilmu komputer. Algoritma ini adalah salah satu dari teknik pencarian heuristik.

Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom. Kromosom ini mengkodekan kemungkinan jawaban untuk sebuah masalah yang diberikan. Kromosom dievaluasi dan diberi *fitness value* berdasarkan seberapa baikkah kromosom dalam menyelesaikan masalah yang diberikan berdasarkan kriteria yang ditentukan oleh pembuat program. Nilai kelayakan ini digunakan sebagai probabilitas kebertahanan hidup kromosom dalam satu siklus reproduksi. Kromosom baru (kromosom anak, *child chromosome*) diproduksi dengan menggabungkan dua (atau lebih) kromosom orang tua (*parent chromosome*). Proses ini dirancang untuk menghasilkan kromosom-kromosom keturunan yang lebih layak, kromosom-kromosom ini menyandikan jawaban yang lebih baik, sampai solusi yang baik dan yang bisa diterima ditemukan.

Algoritma *hybrid genetic* adalah gabungan antara algoritma genetik dan algoritma-algoritma lainnya. Dalam kasus ini, algoritma genetik digabungkan dengan algoritma *rule based*. Algoritma *rule based* akan dijalankan sampai pada titik dimana algoritma tidak bisa menyelesaikan permainan teka-teki Calcudoku. Jika algoritma sudah tidak bisa menyelesaikan permainan, maka algoritma genetik akan mulai dijalankan [2].

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan di atas, dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana cara mengimplementasikan perangkat lunak (*software*) permainan teka-teki Calcudoku?
2. Bagaimana cara mengimplementasikan algoritma *backtracking* untuk menyelesaikan Calcudoku?
3. Bagaimana cara mengimplementasikan algoritma *hybrid genetic* untuk menyelesaikan Calcudoku?
4. Bagaimana perbandingan performansi algoritma *backtracking* dengan algoritma *hybrid genetic* dalam menyelesaikan Calcudoku?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dirumuskan, maka tujuan dari pembuatan skripsi ini adalah:

1. Membuat perangkat lunak solusi permainan teka-teki Calcudoku yang menerima input berupa soal teka-teki dan mampu menyelesaikan soal teka-teki tersebut menggunakan algoritma *backtracking* dan *hybrid genetic*.
2. Membandingkan performansi algoritma *backtracking* dengan algoritma *hybrid genetic* dalam hal kesuksesan dan (jika sukses) kecepatan dalam menyelesaikan Calcudoku.

1.4 Batasan Masalah

Ruang lingkup dari skripsi ini dibatasi oleh batasan-batasan masalah sebagai berikut:

1. Ukuran *grid* untuk permainan teka-teki Calcudoku adalah antara 4×4 sampai dengan 8×8 . Pada awalnya, ukuran *grid* direncanakan akan dibatasi dari 3×3 sampai dengan 9×9 , tetapi karena kurangnya contoh soal teka-teki Calcudoku dengan ukuran 3×3 , dan ada masalah saat pengujian (keluar pesan error "*Memory full*" saat menguji *solver* dengan algoritma *backtracking* pada *grid* yang berukuran 9×9 , maka ukuran *grid* dibatasi dari 4×4 sampai dengan 8×8 .
2. Pada algoritma *rule based*, yang merupakan bagian dari algoritma *hybrid genetic*, aturan-aturan logika yang digunakan dibatasi hanya pada aturan *single square*, *naked single*, *naked double*, *hidden single*, dan *killer combination*.
3. Soal-soal permainan teka-teki Calcudoku yang digunakan dalam pengujian diambil dari sumber-sumber berikut:
 - (a) <https://iota.math.msu.edu/k12-outreach/kenken-puzzles/>
 - (b) <http://thinkmath.edc.org/resource/kenken-puzzles>

1.5 Metodologi Penelitian

Langkah-langkah yang akan dilakukan dalam pembuatan skripsi ini adalah:

1. Studi literatur
 - (a) Melakukan studi literatur tentang permainan teka-teki Calcudoku.
 - (b) Melakukan studi literatur tentang algoritma *backtracking*.
 - (c) Melakukan studi literatur tentang algoritma *rule based* dan algoritma genetik.
2. Analisis, perancangan, dan pengembangan perangkat lunak
 - (a) Melakukan analisis dan menentukan fitur-fitur yang diperlukan dalam perangkat lunak permainan teka-teki Calcudoku.
 - (b) Membuat perangkat lunak Calcudoku dengan fitur-fitur yang telah ditentukan.
 - (c) Mengimplementasikan algoritma *backtracking* untuk Calcudoku.
 - (d) Mengimplementasikan algoritma *hybrid genetic* untuk Calcudoku.
3. Melakukan pengujian dan eksperimen terhadap perangkat lunak Calcudoku yang telah dibuat. Soal-soal permainan teka-teki Calcudoku yang digunakan dalam pengujian diambil dari sumber-sumber berikut:
 - (a) <https://iota.math.msu.edu/k12-outreach/kenken-puzzles/>
 - (b) <http://thinkmath.edc.org/resource/kenken-puzzles>
4. Membandingkan performansi algoritma *backtracking* dengan algoritma *hybrid genetic* dalam menyelesaikan Calcudoku.
5. Membuat kesimpulan berdasarkan hasil pengujian perangkat lunak yang telah dibuat.

1.6 Sistematika Pembahasan

Sistematika pembahasan skripsi ini adalah sebagai berikut:

1. Bab 1 berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi penelitian, dan sistematika pembahasan dari skripsi ini.
2. Bab 2 membahas tentang landasan teori yang digunakan dalam skripsi ini, yaitu tentang permainan teka-teki Calcudoku, algoritma *backtracking* dan algoritma *hybrid genetic*.
3. Bab 3 membahas tentang analisis perangkat lunak Calcudoku dan analisis algoritma *backtracking* dan algoritma *hybrid genetic*.
4. Bab 4 membahas tentang perancangan dan pembuatan perangkat lunak Calcudoku dan algoritma *backtracking* dan algoritma *hybrid genetic* untuk menyelesaikan permainan, perancangan antarmuka (*interface*), input dan output, diagram kelas (*class diagram*), dan diagram aktivitas (*activity diagram*).
5. Bab 5 membahas tentang implementasi dari perangkat lunak Calcudoku dan algoritma *backtracking* dan algoritma *hybrid genetic* yang telah dirancang, implementasi antarmuka, input dan output yang telah dirancang, dan pengujian perangkat lunak Calcudoku dalam hal perbandingan performansi algoritma *backtracking* dan algoritma *hybrid genetic* dalam menyelesaikan permainan.
6. Bab 6 berisi kesimpulan dari pembuatan perangkat lunak Calcudoku dan hasil pengujinya, dan saran untuk penelitian pengembangan perangkat lunak selanjutnya.

BAB 2

LANDASAN TEORI

Bab ini membahas tentang landasan teori yang akan digunakan dalam skripsi ini yang diambil dari dua sumber, yaitu "KenKen Puzzle Solver using Backtracking Algorithm" karya Asanilta Fahda [1] dan "Solving and Modeling Ken-ken Puzzle by Using Hybrid Genetics Algorithm" karya Olivia Johanna, Samuel Lukas, dan Kie Van Ivanký Saputra [2].

2.1 Calcudoku [1] [2]

Sebagai salah satu jenis permainan teka-teki aritmatika dan *grid*, Calcudoku, atau dikenal juga sebagai KenKen, KenDoku, atau Mathdoku, diciptakan pada tahun 2004 oleh seorang guru matematika dari Jepang yang bernama Tetsuya Miyamoto untuk memenuhi tujuannya untuk melatih kemampuan matematika dan logika siswa-siswinya dengan cara yang menyenangkan. Nama KenKen diambil dari kata bahasa Jepang yang berarti kepandaian. Permainan yang mengasah otak ini dengan cepat menyebar ke seluruh Jepang dan Amerika Serikat, menggantikan permainan teka-teki silang di banyak koran. Permainan ini kemudian menjadi sensasi di seluruh dunia setelah munculnya versi *online* dan *mobile* dari permainan teka-teki ini, khususnya menarik untuk pecinta permainan teka-teki angka seperti Sudoku.

Seperti dalam Sudoku, dalam teka-teki ini, pemain diberikan sebuah *grid* dengan ukuran $n \times n$, dengan n biasanya $3 \leq n \leq 9$. *Grid* ini harus diisi dengan angka 1 sampai dengan n sehingga dalam setiap baris setiap angka hanya muncul sekali, dalam setiap kolom setiap angka hanya muncul sekali. Perbedaannya dengan Sudoku adalah, Calcudoku dibagi ke dalam *cage* (sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel, setiap *cage* mempunyai angka tujuan dan operator yang telah ditentukan), dan angka-angka dalam setiap *cage* harus mencapai angka tujuan jika dihitung menggunakan operator yang telah ditentukan. Angka tujuan dan operasi yang telah ditentukan ditulis di sudut kiri atas *cage*. Ada lima kemungkinan operator:

1. $+$, yaitu sebuah operator n -ary yang menandakan penjumlahan.
2. $-$, yaitu sebuah operator biner yang menandakan pengurangan *absolute*.
3. \times , yaitu sebuah operator n -ary yang menandakan perkalian.
4. \div atau $/$, yaitu sebuah operator biner yang menandakan pembagian.
5. $=$, (simbol ini biasanya dihilangkan), yaitu sebuah operator uner yang menandakan persamaan.

Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuran *cage* harus dua sel. Jika operasi matematika yang ditentukan adalah sama dengan, maka ukuran *cage* harus satu sel. Pada beberapa versi dari teka-teki ini, hanya angka tujuan yang diberikan, dan pemain harus menebak operator dari setiap *cage* untuk menyelesaikan teka-tekinya [1] [2].

Untuk menyelesaikan sebuah teka-teki Calcudoku, pemain pertama-tama harus memahami dua permasalahan utama dari teka-teki ini, yaitu:

1. Angka-angka mana yang harus dimasukkan ke dalam sebuah *cage*

3	8+	3-	
7+			
	8+	8+	
			1

Gambar 2.1: Contoh permainan teka-teki dengan ukuran *grid* 4×4 yang belum diselesaikan. [1]

3	2	1	4
1	4	2	3
4	1	3	2
2	3	4	1

Gambar 2.2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 2.1.

2. Dalam urutan apa angka-angka tersebut harus dimasukkan ke dalam sebuah *cage*

Seperti kebanyakan permainan teka-teki angka, cara yang paling mudah untuk menyelesaikan teka-teki ini adalah dengan mengeliminasi angka-angka yang sudah digunakan dan mencoba satu per satu angka yang mungkin (*trial and error*).

Dalam pengisian teka-teki ini ada dua tahapan, yaitu:

1. Mencari *cage* yang hanya berukuran 1 sel, karena *cage* ini tidak menghasilkan pertanyaan angka apa dan urutan apa. Tahap ini adalah tahap yang paling jelas.

Contoh, pada Gambar 2.1, *cage* pada sudut kiri atas dan *cage* pada sudut kanan bawah hanya berukuran 1 sel, dan dapat langsung diisi dengan angka tujuannya.

2. Mencari *cage* yang hanya mempunyai satu kemungkinan kombinasi angka, sehingga masalah angka-angka apa yang harus diisi dalam *cage* tersebut terjawab.

Contoh, *cage* pada sudut kanan atas mempunyai aturan "3-", artinya angka tujuannya adalah 3 dengan menggunakan operasi pengurangan. Satu-satunya pasangan angka dari himpunan $\{1,2,3,4\}$ yang akan menghasilkan angka 3 saat satu angka dikurangkan dari angka yang lainnya adalah $\{1,4\}$. Namun masalahnya adalah urutan angka-angka yang harus dimasukkan. Dalam kasus ini, untungnya, sel pada sudut kanan bawah sudah diisi dengan angka 1, maka angka 1 tidak bisa digunakan lagi pada kolom yang paling kanan. Jadi, dengan menggunakan cara eliminasi, sel pada sudut kanan atas harus diisi dengan angka 4 dan sel di sebelah kirinya, yaitu sel pada baris yang paling atas dan kolom ketiga dari kiri, harus diisi dengan angka 1. Hal ini memberikan solusi untuk sel pada baris yang paling atas dan kolom kedua dari kiri, yaitu angka 2, karena angka 2 adalah angka yang belum pernah dipakai dalam baris tersebut. Proses ini berlanjut sampai semua sel dalam *grid* terisi dan menghasilkan solusi pada Gambar 2.2 [1].

Seiring dengan meningkatnya tingkat kesulitan, langkah berikutnya tidak akan langsung muncul dengan jelas. Kadang-kadang, pemain mencapai titik dimana langkah berikutnya tidak pasti. Pemain harus menebak langkah-langkah berikutnya dan melihat apakah langkah ini akan menghasilkan solusinya. Jika tidak, pemain harus mundur kembali ke titik ketidakpastian tersebut.

Sebuah teka-teki Calcudoku dengan ukuran $n \times n$, dengan n melambangkan jumlah sel dalam satu baris atau kolom, mempunyai n^2 sel dalam sebuah *grid*. Sel yang terletak dalam baris b dan kolom k diberi label $C_{b,k} = bn + k$ dan nilai dari sel tersebut adalah $V(C_{b,k}) \in \{1, 2, \dots, n\}$. Nomor baris b memiliki range $0 \leq b \leq n - 1$. Nomor kolom k memiliki range $0 \leq k \leq n - 1$. Nomor sel C memiliki range $0 \leq C \leq n^2 - 1$. Nomor sel adalah hasil perkalian dari nomor baris tempat sebuah sel berada dikalikan dengan banyaknya sel dalam sebuah baris, lalu dijumlahkan dengan nomor kolom tempat sebuah sel berada. Sebuah *cage*, yang diberi label A_i adalah sebuah himpunan dari sel, yaitu $A_i = \{C_{b,k}\}$. Setiap *cage* terhubung dengan satu operator aritmatika $O_i \in \{+, -, \times, \div, =\}$, artinya operator aritmatika adalah salah satu dari penjumlahan, pengurangan, perkalian, pembagian, dan sama dengan, dan satu angka tujuan $H_i \in N$, artinya angka tujuan adalah sebuah bilangan asli. Tiga aturan dalam mendefinisikan masalah dalam Calcudoku adalah sebagai berikut [2]:

1. $|A_i| = 1 \rightarrow O_i = \phi$, artinya setiap *cage* yang jumlah selnya 1 dengan operasi matematika yang terkait dengan *cage* tersebut memiliki hubungan korespondensi satu ke satu.
2. $O_i \in \{-, \div\} \rightarrow |A_i| = 2$, artinya jika operasi yang digunakan dalam sebuah *cage* adalah pengurangan atau pembagian, maka jumlah sel dalam *cage* tersebut harus 2.
3. $\forall C_{b,k} \rightarrow C_{b,k} \in \exists! A_i$, artinya setiap sel hanya boleh menjadi anggota dari satu dan hanya satu *cage*.

Tujuan dari teka-teki ini adalah untuk mencari nilai $V(C_{b,k})$ dan memenuhi persyaratan berikut [2]:

1. $|A_i| = 1 \wedge C_{b,k} \in A_i \rightarrow V(C_{b,k}) = H_i$, artinya jika sel adalah bagian dari sebuah *cage* yang jumlah selnya 1, maka nilai dari sel tersebut adalah angka tujuan dari *cage* tersebut.
2. $O_i \in \{-\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow |V(C_{a,b}) - V(C_{p,q})| = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah pengurangan, maka nilai absolut dari hasil pengurangan nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
3. $O_i \in \{\div\} \wedge A_i = \{C_{a,b}, C_{p,q}\} \rightarrow V(C_{a,b})/V(C_{p,q}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah pembagian, maka nilai dari hasil pembagian nilai kedua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
4. $O_i \in \{+\} \rightarrow \sum_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah penjumlahan, maka nilai dari hasil penjumlahan dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.
5. $O_i \in \{\times\} \rightarrow \prod_{C_{b,k} \in A_i} V(C_{b,k}) = H_i$, artinya jika sebuah *cage* yang operasi matematikanya adalah perkalian, maka nilai dari hasil perkalian dari nilai semua sel di dalam *cage* tersebut adalah angka tujuan dari *cage* tersebut.

2.2 Algoritma Backtracking [1]

Algoritma *backtracking* adalah sebuah algoritma umum yang mencari solusi dengan mencoba salah satu dari beberapa pilihan, jika pilihan yang dipilih ternyata salah, komputasi dimulai lagi pada titik pilihan dan mencoba pilihan lainnya. Untuk bisa melacak kembali langkah-langkah yang telah dipilih, maka algoritma harus secara eksplisit menyimpan jejak dari setiap langkah yang sudah pernah dipilih, atau menggunakan rekursi (*recursion*). Rekursi dipilih karena jauh lebih mudah daripada harus menyimpan jejak setiap langkah yang pernah dipilih. Hal ini menyebabkan algoritma ini biasanya berbasis DFS (*Depth First Search*).

Algoritma *backtracking* pertama kali diperkenalkan pada tahun 1950 oleh D.H. Lehmer sebagai perbaikan algoritma *brute force*. Algoritma ini lalu dikembangkan lebih lanjut oleh R.J. Walker, S.W. Golomb, dan L.D. Baumert. Algoritma ini terbukti efektif untuk menyelesaikan banyak

permainan logika (misalnya *tic tac toe*, *maze*, catur, dan lain-lain) karena algoritma itu terutama berguna untuk menyelesaikan masalah-masalah *constraint satisfaction*, di mana sekumpulan objek harus memenuhi sejumlah batasan.

Implementasi algoritma *backtracking* memiliki tiga sifat umum, yaitu [1]:

1. *Solution space*

Solusi untuk masalah ini dinyatakan sebagai sebuah vektor X dengan n -tuple:

$$X = (x_1, x_2, \dots, x_n), x_i \in S_i$$

di mana adalah mungkin bahwa:

$$S_1 = S_2 = \dots = S_n$$

2. Fungsi pembangkit X_k (*generating function*)

Fungsi pembangkit X_k dinyatakan sebagai:

$$T(k)$$

di mana $T(k)$ membangkitkan nilai X_k , dari 1 sampai n , yang merupakan komponen dari vektor solusi.

3. Fungsi pembatas (*bounding function*)

Fungsi pembatas dinyatakan sebagai:

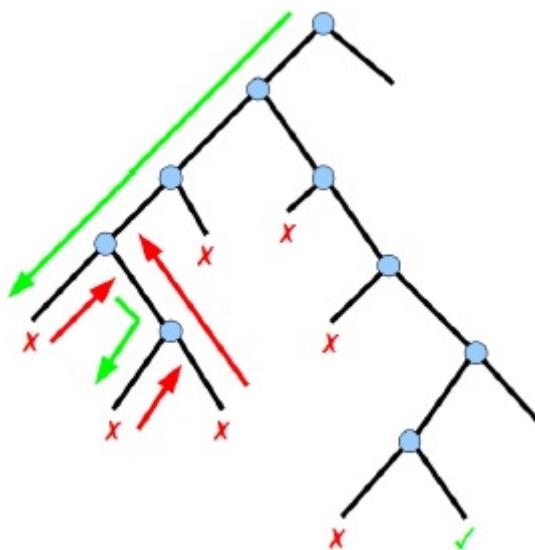
$$B(x_1, x_2, \dots, x_k)$$

di mana B bernilai *true* jika (x_1, x_2, \dots, x_k) mengarah ke solusi. Jika B bernilai *true*, maka nilai $x_k + 1$ akan terus dibangkitkan, dan jika B bernilai *false*, maka (x_1, x_2, \dots, x_k) akan dibuang.

Ruang solusi untuk algoritma *backtracking* disusun dalam sebuah struktur berbentuk pohon (*tree*), di mana setiap simpul (*node*) merepresentasikan keadaan masalah dan sisi (*edge*) diberi label x_i . Jalur dari akar (*root*) ke daun (*leaf*) merepresentasikan sebuah jawaban yang mungkin, dan semua jalur yang dikumpulkan bersama-sama membentuk ruang solusi. Struktur pohon ini disebut sebagai *state space tree*. Gambar 2.3 menggambarkan contoh sebuah *state space tree*.

Langkah-langkah dalam menggunakan *state space tree* untuk mencari solusi adalah [1]:

1. Solusi dicari dengan membangun jalur dari akar ke daun menggunakan algoritma DFS.
2. Simpul yang terbentuk disebut sebagai simpul hidup (*live nodes*).
3. Simpul yang sedang diperluas disebut sebagai *expand nodes* atau *E-nodes*.
4. Setiap kali sebuah *E-node* sedang diperluas, jalur yang dikembangkannya menjadi lebih panjang.
5. Jika jalur yang sedang dikembangkan tidak mengarah ke solusi, maka *E-node* dimatikan dan menjadi simpul mati (*dead node*).
6. Fungsi yang digunakan untuk mematikan *E-node* adalah implementasi dari fungsi pembatas.
7. Simpul mati tidak akan diperluas.
8. Jika jalur yang sedang dibangun berakhir dengan simpul mati, proses akan mundur ke simpul sebelumnya.



Gambar 2.3: Ilustrasi *State space tree* yang digunakan dalam algoritma *backtracking* [1]

9. Simpul sebelumnya terus membangkitkan simpul anak (*child node*) lainnya, yang kemudian menjadi *E-node* baru.
10. Pencarian selesai jika simpul tujuan tercapai.

Setiap simpul di dalam *state space tree* terkait dengan panggilan rekursif. Jika jumlah simpul di dalam pohon $2n$ atau $n!$, maka pada kasus terburuk untuk algoritma *backtracking* ini memiliki kompleksitas waktu $O(p(n)2n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ sebagai polinomial dengan n -derajat menyatakan waktu komputasi untuk setiap simpul.

Ruang solusi untuk sebuah permainan teka-teki Calcudoku dengan *grid* yang berukuran $n \times n$ adalah $X = (x_1, x_2, \dots, x_m)$, $x_i \in \{1, 2, \dots, n\}$, dengan $m = n^2$. Jadi, n adalah jumlah sel dalam satu baris atau kolom, X adalah sebuah himpunan yang merepresentasikan isi dari setiap sel dalam *grid*, dimulai pada sel pada sudut kiri atas, lalu bergerak ke sel di sebelah kanannya dalam baris yang sama, jika sudah mencapai sel yang paling kanan maka bergerak ke sel yang paling kiri pada baris dibawahnya, hingga berakhir pada sel pada sudut kanan bawah, dan S_i adalah sebuah himpunan yang berisi angka-angka dari 1 sampai n .

Fungsi pembangkit membangkitkan sebuah integer secara berurutan dari 1 sampai n sebagai x_k . Fungsi pembatas menggabungkan tiga fungsi pemeriksa pembatas (*constraint checking*), yaitu fungsi pemeriksa kolom (*column checking*), fungsi pemeriksa baris (*row checking*), dan fungsi pemeriksa *grid* (*grid checking*).

Fungsi pemeriksa kolom menghasilkan nilai *true* jika x_k belum ada di dalam kolom dan menghasilkan nilai *false* jika x_k sudah ada di dalam kolom.

Fungsi pemeriksa baris menghasilkan nilai *true* jika x_k belum ada di dalam baris dan menghasilkan nilai *false* jika x_k sudah ada di dalam baris.

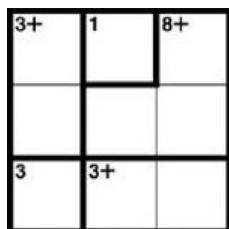
Fungsi pemeriksa *grid* memeriksa operator pada *grid* dan memeriksa berdasarkan operator yang telah ditentukan. Ada 5 operator yang digunakan dalam fungsi ini, yaitu:

1. Operator penjumlahan (+)

Fungsi menghasilkan nilai *true* jika ada sel yang kosong di dalam *cage*, atau hasil penjumlahan semua nilai yang ada di dalam *cage* sama dengan nilai tujuan. Fungsi menghasilkan nilai *false* jika hasil penjumlahan semua nilai yang ada di dalam *cage* tidak sama dengan nilai tujuan.

2. Operator pengurangan (-)

Fungsi menghasilkan nilai *true* jika ada sel yang kosong di dalam *cage*, atau jika hasil pengurangan *absolute* kedua nilai yang ada di dalam *cage* sama dengan nilai tujuan. Fungsi



Gambar 2.4: Contoh permainan teka-teki Calcudoku dengan ukuran $grid\ 3 \times 3$ [1]

menghasilkan nilai *false* jika hasil pengurangan *absolute* kedua nilai yang ada di dalam *cage* tidak sama dengan nilai tujuan.

3. Operator perkalian (\times)

Fungsi menghasilkan nilai *true* jika ada sel yang kosong di dalam *cage*, atau hasil perkalian semua nilai yang ada di dalam *cage* sama dengan nilai tujuan. Fungsi menghasilkan nilai *false* jika hasil perkalian semua nilai yang ada di dalam *cage* tidak sama dengan nilai tujuan.

4. Operator pembagian (\div)

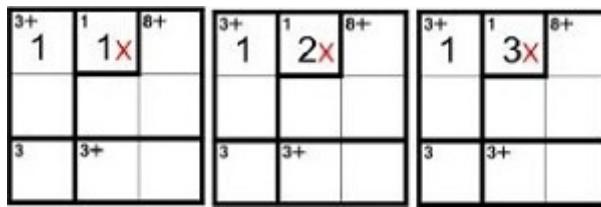
Fungsi menghasilkan nilai *true* jika ada sel yang kosong di dalam *cage*, atau jika hasil pembagian kedua nilai yang ada di dalam *cage* sama dengan nilai tujuan. Fungsi menghasilkan nilai *false* jika hasil pembagian kedua nilai yang ada di dalam *cage* tidak sama dengan nilai tujuan.

5. Operator =

Fungsi akan menghasilkan nilai *true* jika sel dalam *cage* tersebut kosong, atau x_k sama dengan nilai tujuan. Fungsi menghasilkan nilai *false* jika x_k tidak sama dengan nilai tujuan.

State space tree bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan. Untuk mengilustrasikan berkembangnya *state space tree*, teka-teki Calcudoku yang digambarkan pada Gambar 2.4 akan digunakan. Berikut ini adalah tahap-tahap berkembangnya *state space tree* untuk teka-teki tersebut.

1. *State space tree* dimulai dengan *state 1* yang merepresentasikan sebuah *grid* yang kosong.
2. Fungsi pembangkit pertama-tama akan membangkitkan angka 1 sebagai x_1 , yang akan diisi pada sel pertama yang kosong, yaitu sel yang terletak di sudut kiri atas *grid*, atau sel pada kolom ke-1 dan baris ke-1 (*state 2*). Fungsi pembatas akan memeriksa jika langkah ini adalah langkah yang berlaku, dan ternyata langkah ini berlaku.
3. Untuk sel yang kosong berikutnya, yaitu x_2 , atau sel pada kolom ke-2 dan baris ke-1, fungsi pembangkit akan membangkitkan angka 1 (*state 3*), tetapi langkah ini gagal dalam pemeriksaan baris dalam fungsi pembatas karena angka 1 sudah pernah digunakan pada baris tersebut, ini membentuk sebuah simpul mati.
4. Fungsi pembangkit akan mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 4*), tetapi langkah ini gagal dalam pemeriksaan *grid* dalam fungsi pembatas karena angka 2 tidak sama dengan angka tujuan, yaitu angka 1.
5. Fungsi pembangkit akan mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 5*), tetapi langkah ini juga gagal dalam pemeriksaan *grid* dalam fungsi pembatas karena angka 3 tidak sama dengan angka tujuan, yaitu angka 1. Gambar 2.5 menggambarkan *state 3*, *state 4*, dan *state 5* dalam penyelesaian teka-teki Calcudoku ini.



Gambar 2.5: Ilustrasi *state* 3, 4, dan 5 pada sebuah *grid* teka-teki Calcudoku [1]

6. Karena tidak ada solusi yang mungkin, maka mundur ke *state* 1. Fungsi pembangkit akan membangkitkan kemungkinan angka berikutnya sebagai x_1 , yaitu 2, dan ternyata angka 2 berlaku sebagai x_1 (*state* 6), sehingga bisa maju ke x_2 , yaitu sel pada kolom ke-2 dan baris ke-1.
7. Fungsi pembangkit akan membangkitkan angka 1 (*state* 7), dan ini memenuhi syarat yang ditentukan dalam fungsi pembatas, karena angka 1 sama dengan angka tujuan, yaitu angka 1, sehingga bisa maju ke x_3 , yaitu sel pada kolom ke-3 dan baris ke-1.
8. Angka 1 (*state* 8) gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan pada baris tersebut.
9. Angka 2 (*state* 9) juga gagal dalam pemeriksaan baris karena angka 2 sudah pernah digunakan pada baris tersebut.
10. Hal ini menyebabkan hanya tersisa angka 3 sebagai angka yang bisa dimasukkan ke dalam x_3 (*state* 10). Karena *state* 10 ternyata berlaku, baris ke-1 telah selesai diisi, dan bisa maju ke baris ke-2.
11. Langkah berikutnya adalah membuat *state* baru dengan mengisikan angka 1 pada x_4 , yaitu sel pada kolom ke-1 dan baris ke-2 (*state* 11). Ini memenuhi pemeriksaan pembatas, karena $2 + 1 = 3$, sehingga akan maju ke sel berikutnya, yaitu x_5 , atau sel pada kolom ke-2 dan baris ke-2.
12. Angka 1 (*state* 12) jelas tidak bisa digunakan karena gagal dalam pemeriksaan kolom dan pemeriksaan baris; angka 1 sudah pernah digunakan pada kolom dan baris tersebut.
13. Angka 2 (*state* 13) adalah langkah yang berlaku, sehingga bisa maju ke sel berikutnya, yaitu x_6 , atau sel pada kolom ke-3 dan baris ke-2.
14. Langkah berikutnya adalah mengisikan x_6 dengan angka 1 (*state* 14), tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan pada baris tersebut.
15. Langkah berikutnya adalah mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 15), tetapi juga gagal dalam pemeriksaan baris karena angka 2 sudah pernah digunakan pada baris tersebut.
16. Langkah berikutnya adalah mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 16), tetapi juga gagal, kali ini angka 3 gagal dalam pemeriksaan kolom karena angka 3 sudah pernah digunakan pada kolom tersebut.
17. Karena semua kemungkinan angka gagal dalam pemeriksaan baris dan kolom, maka akan mundur ke *state* 11 dan mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 17), dan ternyata angka 3 berlaku sebagai x_5 , sehingga bisa maju ke sel berikutnya, yaitu x_6 .
18. Langkah berikutnya adalah mencoba angka 1 (*state* 18) sebagai x_6 , tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan dalam baris tersebut.

$3+$	1	$8+$
2	1	3
1	3	2
3	$3+$	

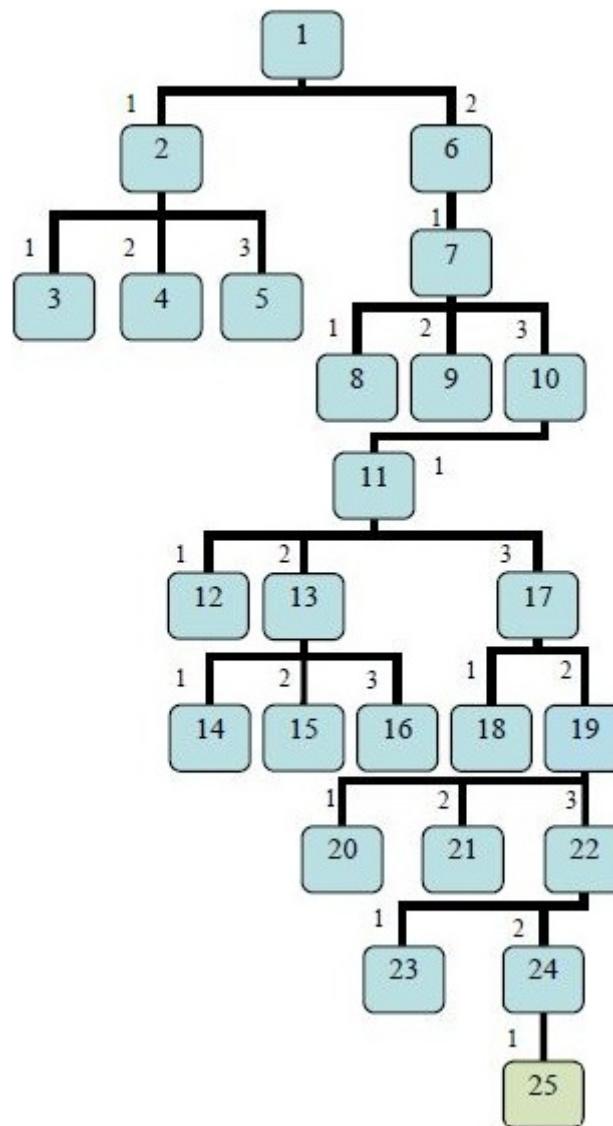
Gambar 2.6: Ilustrasi *state* 19 pada sebuah *grid* teka-teki Calcudoku [1]

$3+$	1	$8+$
2	1	3
1	3	2
3	$3+$	1

Gambar 2.7: *State* 25, simpul tujuan, sebagai hasil yang dicapai [1]

19. Langkah berikutnya adalah mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 19), dan ternyata angka 2 berlaku. Baris ke-2 telah selesai diisi. Gambar 2.6 menggambarkan *state* 19 dalam penyelesaian teka-teki Calcudoku ini.
20. Langkah berikutnya adalah mulai mengisikan sel-sel yang terletak pada baris ke-3, dari kolom yang paling kiri ke kolom yang paling kanan, dimulai dengan mengisikan x_7 , yaitu sel pada kolom ke-1 dan baris ke-3 dengan angka 1 (*state* 20), tetapi gagal dalam pemeriksaan kolom, karena angka 1 sudah pernah digunakan dalam kolom tersebut.
21. Langkah berikutnya adalah mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 21), tetapi juga gagal dalam pemeriksaan kolom, karena angka 2 sudah pernah digunakan dalam kolom tersebut.
22. Langkah berikutnya adalah mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 22), dan ternyata berhasil, sehingga bisa maju ke sel berikutnya, yaitu x_8 , atau sel pada kolom ke-2 dan baris ke-3.
23. Langkah berikutnya adalah mencoba mengisikan angka 1 pada x_8 (*state* 23), tetapi gagal dalam pemeriksaan kolom, karena angka 1 sudah pernah digunakan dalam kolom tersebut.
24. Langkah berikutnya adalah mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 24), dan ternyata berhasil, sehingga bisa maju ke sel berikutnya, yaitu x_9 , atau sel pada kolom ke-3 dan baris ke-3.
25. x_9 adalah sel terakhir, terletak pada sudut kanan bawah *grid*. Langkah berikutnya adalah mencoba mengisikan x_9 dengan angka 1 (*state* 25), dan ternyata berhasil. Algoritma *backtracking* telah selesai mengisikan seluruh sel dalam *grid* dengan benar. Gambar 2.7 menggambarkan *state* 25 dalam penyelesaian teka-teki Calcudoku ini. Algoritma *backtracking* ini mencapai solusinya pada *state* 25, seperti pada *state space tree* yang digambarkan dalam Gambar 2.8. *State space tree* ini telah mencapai simpul tujuannya, yaitu simpul 25, dengan jalur 2-1-3-1-3-2-3-2-1.

Tinggi pohon yang dikembangkan untuk menyelesaikan sebuah teka-teki dengan ukuran $n \times n$ akan memiliki tinggi $n^2 + 1$ saat mencapai simpul tujuannya, dengan jalur dari simpul akar ke simpul tujuan merepresentasikan semua angka yang digunakan untuk mengisi *grid* dari sel pada sudut kiri atas ke sel pada sudut kanan bawah.



Gambar 2.8: *State space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 2.4 [1]

3	9	4	17	6	17	18	27	5
---	---	---	----	---	----	----	----	---

Gambar 2.9: Contoh bagaimana cara mendeteksi aturan *naked pair* [2]

Singkatnya, langkah-langkah dasar dari implementasi algoritma *backtracking* dapat dijelaskan sebagai berikut [1]:

1. Carilah sel pertama atau sel yang kosong di dalam *grid*.
2. Isilah sel dengan sebuah angka dimulai dari 1 sampai n sampai sebuah angka yang berlaku (*valid*) ditemukan atau sampai angka sudah melebihi n .
3. Jika angka untuk sel berlaku, ulangi langkah 1 dan 2.
4. Jika angka untuk sel sudah melebihi n dan tidak ada angka dari 1 sampai n yang berlaku untuk sel tersebut, mundur ke sel sebelumnya dan cobalah kemungkinan angka berikutnya yang berlaku untuk sel tersebut.
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan.

2.3 Algoritma *Hybrid Genetic* [2]

Dalam kasus ini, algoritma *hybrid genetic* adalah gabungan dari algoritma *rule based* dan algoritma genetik. Algoritma *rule based* akan dijalankan sampai pada titik dimana algoritma tidak bisa menyelesaikan permainan teka-teki Calcudoku. Jika algoritma sudah tidak bisa menyelesaikan permainan, maka algoritma genetik akan mulai dijalankan.

2.3.1 Algoritma *Rule Based*

Algoritma *rule based* adalah sebuah algoritma berbasis aturan logika untuk menyelesaikan teka-teki Sudoku dan variasinya, termasuk Calcudoku. Beberapa aturan logika yang digunakan dalam algoritma ini adalah *single square rule*, *naked subset rule*, *hidden single rule*, *evil twin rule*, *killer combination*, dan *X-wing* [2].

Aturan *single square* digunakan jika sebuah *cage* hanya berisi satu sel. Hal ini berarti nilai dari sel tersebut sama dengan angka tujuan yang telah ditentukan.

Aturan *naked subset* digunakan jika ada n sel dalam kolom atau baris yang sama yang mempunyai n kemungkinan nilai yang sama persis untuk mengisikannya, dengan $n \geq 2$. Hal ini berarti sel-sel lainnya dalam baris dan kolom tersebut tidak mungkin diisi dengan nilai yang sama dengan nilai milik n sel tersebut. Gambar 2.9 menunjukkan bagaimana cara kerja aturan ini. Sel-sel pada kolom ke-4 dan ke-6 mempunyai tepat dua kemungkinan nilai (1 atau 7). Ini disebut sebagai *naked pair*. Karena angka 1 dan 7 harus diisi pada sel-sel pada kolom ke-4 dan ke-6, maka angka 1 dan 7 bisa dieliminasi dari sel-sel pada kolom ke-7 dan ke-8.

Aturan *evil twin* digunakan jika sebuah *cage* berisikan dua sel, dan salah satu dari kedua sel sudah terisi, maka sel yang satunya lagi diisi dengan angka yang jika kedua angka dihitung dengan operasi matematika yang ditentukan maka akan menghasilkan angka tujuan yang ditentukan. Aturan ini adalah aturan yang paling mudah. Kenyataannya, aturan ini bisa digeneralisasikan untuk *cage* yang berukuran lebih dari 2 sel. Sel yang belum terisi yang terakhir dalam sebuah area diisi oleh sebuah nilai yang diperlukan untuk mencapai nilai tujuan menggunakan operasi matematika yang telah ditentukan. Contohnya, pada Gambar 2.10, begitu sel di sudut kiri bawah diisi oleh angka 4, maka sel diatasnya harus diisi oleh angka 9.

Aturan *hidden single* digunakan jika sebuah angka hanya bisa diisikan dalam satu sel dalam sebuah baris atau kolom. Aturan ini secara konsep cukup mudah, tetapi kadang-kadang sulit untuk

Gambar 2.10: Contoh aturan *evil twin* [2]Gambar 2.11: Contoh aturan *hidden single* [2]

diamati. Pada Gambar 2.11, nilai-nilai yang mungkin untuk sel yang paling kiri adalah 3, 5, dan 7, tetapi dalam baris ini, angka 7 harus muncul dalam salah satu selnya, dan hanya sel yang paling kiri tersebut yang memiliki kemungkinan nilai 7. Ini disebut sebagai *hidden single*. Sel tersebut harus diisi dengan angka 7.

Aturan *killer combination* adalah aturan yang paling krusial. Aturan ini digunakan jika sebuah *cage* berisikan sel-sel yang berada dalam baris atau kolom yang sama dan operasi yang ditentukan adalah penjumlahan. Kemungkinan angka yang unik untuk aturan *killer combination* berhubungan dengan ukuran *cage*. Contoh, jika sebuah *cage* memiliki dua sel dan angka tujuannya adalah 3, maka kemungkinan angka yang bisa diisikan ke dalam kedua sel tersebut adalah 1 atau 2. Hal ini berarti semua angka lainnya tidak mungkin diisikan ke dalam kedua sel tersebut. Contoh lain, jika sebuah *cage* memiliki tiga sel dan angka tujuannya adalah 24, maka kemungkinan angka yang bisa diisikan ke dalam ketiga sel tersebut adalah 7, 8, atau 9. Gambar 2.12 menampilkan contoh penerapan aturan *killer combination* untuk *cage* dengan ukuran 2 sel. Tabel ini juga bisa diperluas untuk ukuran *cage* lainnya.

Aturan *X-wing* digunakan jika hanya ada dua kemungkinan angka yang bisa diisikan ke dalam dua sel yang berada di dalam dua baris yang berbeda, dan dua kemungkinan angka tersebut juga berada di dalam kolom yang sama maka sel-sel lainnya dalam kolom tersebut tidak mungkin diisi oleh dua kemungkinan angka tersebut, atau jika hanya ada dua kemungkinan angka yang bisa diisikan ke dalam dua sel yang berada di dalam dua kolom yang berbeda, dan dua kemungkinan angka tersebut juga berada di dalam baris yang sama maka sel-sel lainnya dalam baris tersebut tidak mungkin diisi oleh dua kemungkinan angka tersebut. Gambar 2.13 menampilkan contoh penggunaan aturan *X-wing*. Misalnya, jika sel A diisi oleh angka 7, maka angka 7 akan dieliminasi dari sel B dan sel C. Karena sel A dengan sel C dan sel D 'terkunci', maka sel D harus diisi oleh angka 7. Jadi, angka 7 harus diisi pada sel A dan sel D atau pada sel B dan sel C. Angka 7 bisa dieliminasi dari sel-sel yang berwarna hijau.

2.3.2 Algoritma Genetik

Pencarian heuristik adalah sebuah teknik pencarian kecerdasan buatan (*artificial intelligence*) yang menggunakan heuristik dalam langkah-langkahnya. Heuristik adalah semacam aturan tidak tertulis

Cage size	Cage value	Combination
2	3	1/2
2	4	1/3
2	17	8/9
2	16	7/9

Gambar 2.12: Contoh aturan *killer combination* untuk *cage* dengan ukuran 2 sel dengan operasi matematika penjumlahan [2]



Gambar 2.13: Contoh aturan *X-wing* [2]

yang mungkin menghasilkan solusi. Heuristik kadang-kadang efektif, tetapi tidak dijamin akan berhasil. dalam setiap kasus. Heuristik memerlukan peran penting dalam strategi pencarian karena sifat eksponensial dari kebanyakan masalah. Heuristik membantu mengurangi jumlah alternatif solusi dari angka yang bersifat eksponensial menjadi angka yang bersifat polinomial. Contoh teknik pencarian heuristik adalah *Generate and Test*, *Hill Climbing*, dan *Best First Search*.

Algoritma genetik adalah salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam. Algoritma ini adalah perpaduan dari bidang biologi dan ilmu komputer. Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom. Kromosom ini meng-*encode* kemungkinan jawaban untuk sebuah masalah yang diberikan. Kromosom dievaluasi dan diberi *fitness value* berdasarkan seberapa baik kromosom dalam menyelesaikan masalah yang diberikan berdasarkan kriteria yang ditentukan oleh pembuat program. Nilai kelayakan ini digunakan sebagai probabilitas kebertahanan hidup kromosom dalam satu siklus reproduksi. Kromosom baru (kromosom anak, *child chromosome*) diproduksi dengan menggabungkan dua (atau lebih) kromosom orang tua (*parent chromosome*). Proses ini dirancang untuk menghasilkan kromosom-kromosom keturunan yang lebih layak, kromosom-kromosom ini meng-*encode* jawaban yang lebih baik, sampai solusi yang baik dan bisa diterima ditemukan.

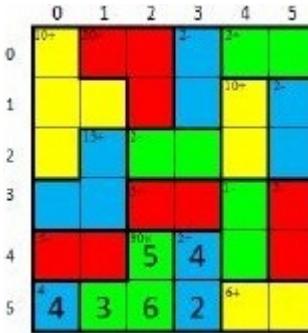
Cara kerja algoritma genetik adalah sebagai berikut [2]:

1. Menentukan populasi kromosom kemungkinan jawaban awal.
2. Membangkitkan populasi kemungkinan jawaban awal secara acak.
3. Mengevaluasi fungsi objektif.
4. Melakukan operasi terhadap kromosom menggunakan operator genetik (reproduksi, kawin silang, dan mutasi).
5. Ulangi langkah 3 dan 4 sampai mencapai kriteria untuk menghentikan algoritma.

Langkah-langkah utama dalam penggunaan algoritma genetik adalah membangkitkan populasi kemungkinan jawaban, mencari fungsi objektif dan fungsi kelayakan, dan penggunaan operator genetik.

2.3.3 Algoritma *Hybrid Genetic*

Pencarian *rule based* dimulai dengan mengasumsikan semua nilai sel yang tidak diketahui dengan semua kemungkinan nilai untuk mengisi sel tersebut tanpa melanggar batasan, dengan $P(C_{b,k}) = 1, 2, \dots, n$. Setelah nilai dari satu sel sudah ditentukan, kemungkinan nilai untuk beberapa sel tertentu diperbaharui. Misalnya, penggunaan aturan *naked single* yang dinyatakan dalam persamaan 1 di bawah ini, akan mengakibatkan semua kemungkinan nilai untuk semua sel lain dalam baris yang



Gambar 2.14: Contoh permainan teka-teki Calcudoku dengan ukuran grid 6 x 6 [2]

sama dan dalam kolom yang sama harus diperbarui, seperti dinyatakan dalam persamaan 2 dan 3 di bawah ini. Aturan *naked pair*, salah satu dari aturan jenis *naked subset*, dinyatakan dalam persamaan 4 untuk baris dan persamaan 5 untuk kolom. [2]

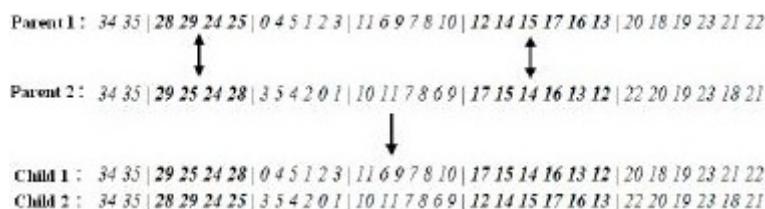
1. $|P(C_{b,k})| = 1 \wedge x \in P(C_{b,k}) \rightarrow V(C_{b,k}) = x$, artinya jika sebuah *cage* berukuran 1 sel, dan x adalah nilai tujuan dari *cage* tersebut, maka nilai dari sel tersebut adalah x .
2. $(V(C_{b,k}) = x) \wedge (\forall a \in \{1, 2, \dots, n\}) \rightarrow P(C_{a,k}) = P(C_{a,k}) - \{x\}$, artinya jika nilai suatu sel pada baris b dan kolom k adalah x , maka x dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada baris b .
3. $(V(C_{b,k}) = x) \wedge (\forall q \in \{1, 2, \dots, n\}) \rightarrow P(C_{b,q}) = P(C_{b,q}) - \{x\}$ artinya jika nilai suatu sel pada baris b dan kolom k adalah x , maka x dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada kolom k .
4. $|P(C_{b,k1})| = |P(C_{b,k2})| = 2 \wedge P(C_{b,k1}) = P(C_{b,k2}) \rightarrow P(C_{b,q}) = P(C_{b,q}) - P(C_{b,k1})$, artinya jika ada dua sel dalam satu baris yang hanya bisa diisi oleh dua kemungkinan angka, maka kedua angka tersebut dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada baris tersebut.
5. $|P(C_{b1,k})| = |P(C_{b2,k})| = 2 \wedge P(C_{b1,k}) = P(C_{b2,k}) \rightarrow P(C_{p,k}) = P(C_{p,k}) - P(C_{b1,k})$, artinya jika ada dua sel dalam satu kolom yang hanya bisa diisi oleh dua kemungkinan angka, maka kedua angka tersebut dihapus dari kemungkinan angka-angka yang bisa digunakan untuk mengisi sel-sel lain pada kolom tersebut.

Algoritma genetik digunakan saat teka-teki masih tidak bisa diselesaikan setelah mengerjakan semua aturan logika secara berulang-ulang. Algoritma ini dimulai dengan meng-*encode* kromosom. Satu kromosom terdiri dari k segmen, dengan $m \leq n$. Satu segmen berisikan sekumpulan gen yang belum diselesaikan yang berada di dalam segmen tersebut. Sebuah segmen merepresentasikan sebuah baris atau kolom. Dalam sebuah kromosom, segmen diurutkan dari baris yang paling atas ke baris yang paling bawah atau dari kolom yang paling kiri ke kolom yang paling kanan. Contoh, salah satu kromosom dari permainan teka-teki Calcudoku pada Gambar 2.14 adalah:

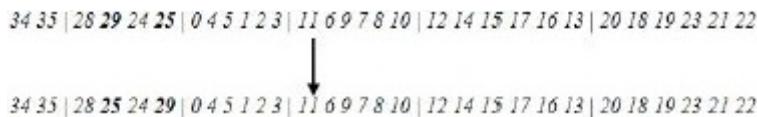
34 35 | 28 29 24 25 | 0 4 5 1 2 3 | 11 6 9 7 8 10 | 12 14 15 17 16 13 | 20 18 19 23 21 22

Setiap segmen dalam contoh kromosom ini merepresentasikan sebuah baris yang belum terselesaikan.

Fungsi objektif, yang direpresentasikan dengan x_j , akan dihitung setelah pembangkitan nilai dari gen pada kromosom sudah dilakukan. Nilai untuk gen ke- j pada sebuah kromosom direpresentasikan dengan w_j . x_j akan bernilai 0 jika belum diselesaikan ($w_j = 0$), dan bernilai 1 jika sudah diselesaikan



Gambar 2.15: Contoh proses kawin silang antara dua kromosom [2]



Gambar 2.16: Contoh proses mutasi [2]

$(w_j \neq 0)$. Untuk kromosom dengan jumlah gen k , fungsi kelayakan, yaitu hasil penjumlahan dari hasil fungsi objektif untuk setiap gen dibagi dengan jumlah gen, dinyatakan dalam persamaan di bawah ini [2]:

$$x_j = \begin{cases} 0 & w_j = 0 \\ 1 & w_j \neq 0 \end{cases}$$

$$\text{fitness} = \frac{\sum_{j=0}^k x_j}{k}$$

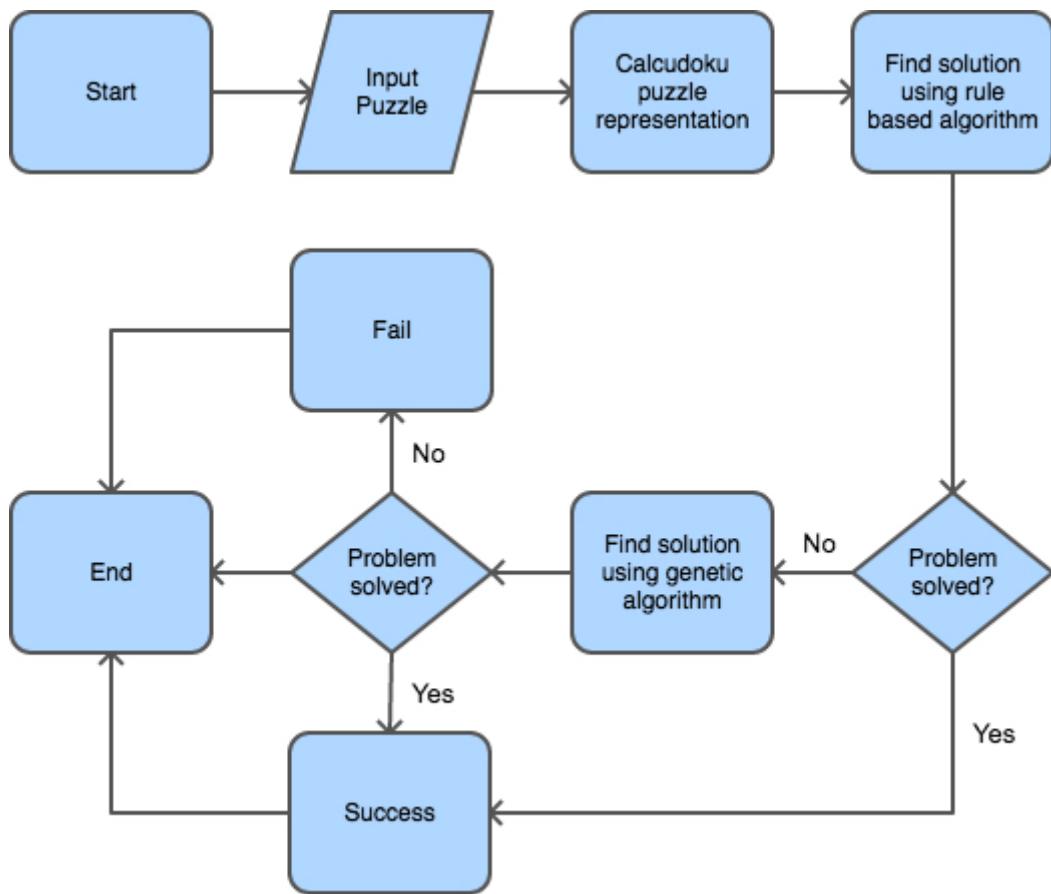
Jadi, solusi dari teka-teki ini adalah mencari kromosom yang nilai kelayakannya 1.

Dalam proses reproduksi kawin silang, dua kromosom, yaitu kromosom orang tua, disilangkan untuk membuat dua kromosom yang baru, yaitu kromosom anak, dengan metodologi kawin silang N -segments. Gambar 2.15 menggambarkan contoh proses kawin silang antara dua kromosom.

Pertukaran mutasi digunakan untuk mendapatkan kemungkinan kromosom yang lain. Mutasi dilakukan di antara gen yang berada dalam segmen yang sama. Gambar 2.16 adalah contoh proses mutasi antara dua gen dalam segmen yang sama.

Cara kerja algoritma *hybrid genetic* adalah sebagai berikut [2]:

1. Masukkan teka-teki yang akan diselesaikan sebagai input. Teka-teki Calcudoku diinputkan oleh pemain dalam bentuk *file*.
2. Representasikan input yang dimasukkan ke dalam format teka-teki Calcudoku. File teka-teki Calcudoku yang telah diinputkan oleh pemain ditampilkan ke layar sebagai teka-teki Calcudoku.
3. Algoritma *hybrid genetic* akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
4. Jika berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* (algoritma *rule based* berhasil mengisi semua sel yang ada di dalam *grid*), maka algoritma selesai.
5. Jika gagal dengan menggunakan algoritma *rule based* (algoritma *rule based* tidak berhasil mengisi semua sel yang ada di dalam *grid*), maka algoritma *hybrid genetic* akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
6. Jika berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik (algoritma genetik berhasil membangkitkan dan menemukan kromosom dengan nilai kelayakan 1), maka algoritma selesai.



Gambar 2.17: Alur penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* [2]

7. Jika gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik (algoritma genetik tidak berhasil membangkitkan dan menemukan kromosom dengan nilai kelayakan 1), artinya algoritma *hybrid genetic* gagal dalam menyelesaikan teka-teki tersebut.

Alur (*flow chart*) penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* dapat dilihat di Gambar 2.17.

BAB 3

ANALISIS

Bab ini membahas tentang analisis cara kerja algoritma *backtracking* dan algoritma *hybrid genetic* untuk menyelesaikan permainan teka-teki Calcudoku, dan analisis kebutuhan perangkat lunak Calcudoku.

3.1 Analisis Algoritma *Backtracking*

Langkah-langkah penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *backtracking* secara umum adalah sebagai berikut:

1. Isilah *grid* mulai dari sel pada sudut kiri atas.
2. Setelah mengisi sebuah sel, isilah sel di sebelah kanannya.
3. Jika sudah mengisi sel yang paling kanan, isilah sel yang paling kiri pada baris berikutnya.
4. Jika semua kemungkinan gagal, mundur ke langkah sebelumnya, dan cobalah kemungkinan berikutnya.
5. Algoritma *backtracking* selesai jika semua sel sudah terisi dengan benar.

Untuk mengilustrasikan cara kerja algoritma *backtracking*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar 3.1 sebagai contoh. *State space tree* seperti yang dijelaskan pada Bab 2.2 akan dibangkitkan.

1. Algoritma *backtracking* dimulai dengan teka-teki yang belum diselesaikan, seperti yang digambarkan pada Gambar 3.1 (*state 1*).
2. Algoritma mengisikan sel pada baris ke-1 dan kolom ke-1 dengan angka 1 (*state 2*), tetapi angka 1 tidak sesuai dengan angka tujuan dari *cage* tersebut.
3. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 3*), tetapi angka 2 juga tidak sesuai dengan angka tujuan dari *cage* tersebut.

3	8+	3-	
7+			
	8+	8+	
			1

Gambar 3.1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan, seperti yang digambarkan pada Gambar 2.1. [1]

3	8+	3-	
7+			
	8+	8+	
			1

Gambar 3.2: *State 4*

3	8+	3-	4
7+			
	8+	8+	
			1

Gambar 3.3: *State 11*

4. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 4*), seperti dapat dilihat pada Gambar 3.2, dan ternyata angka 3 sesuai dengan angka tujuan dari *cage* tersebut, sehingga algoritma dapat maju ke sel berikutnya.
5. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (*state 5*). Algoritma lalu maju ke sel berikutnya.
6. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state 6*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
7. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 7*). Algoritma lalu maju ke sel berikutnya.
8. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state 8*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
9. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 9*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
10. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 10*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
11. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 11*), seperti dapat dilihat pada Gambar 3.3, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.
12. Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 7*). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 12*), seperti dapat dilihat pada Gambar 3.4, tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
13. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 13*). Algoritma lalu maju ke sel berikutnya.

³ 3	⁸⁺ 1	³⁻ 3	
7+			
	⁸⁺	⁸⁺	
			1

Gambar 3.4: *State 12*

³ 3	⁸⁺ 1	³⁻ 4	4
7+			
	⁸⁺	⁸⁺	
			1

Gambar 3.5: *State 17*

14. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state 14*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
15. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 15*), tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.
16. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 16*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
17. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 17*), seperti dapat dilihat pada Gambar 3.5, tetapi angka 4 sudah pernah digunakan dalam baris tersebut.
18. Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-3 dan ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 5*). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 18*), seperti dapat dilihat pada Gambar 3.6. Algoritma lalu maju ke sel berikutnya.
19. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state 19*), seperti dapat dilihat pada Gambar 3.7. Algoritma lalu maju ke sel berikutnya.
20. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state 20*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

³ 3	⁸⁺ 2	³⁻	
7+			
	⁸⁺	⁸⁺	
			1

Gambar 3.6: *State 18*

³ 3	⁸⁺ 2	³⁻ 1	
7+			
	⁸⁺	⁸⁺	
			1

Gambar 3.7: *State 19*

³ 3	⁸⁺ 2	³⁻ 1	4
7+			
	⁸⁺	⁸⁺	
			1

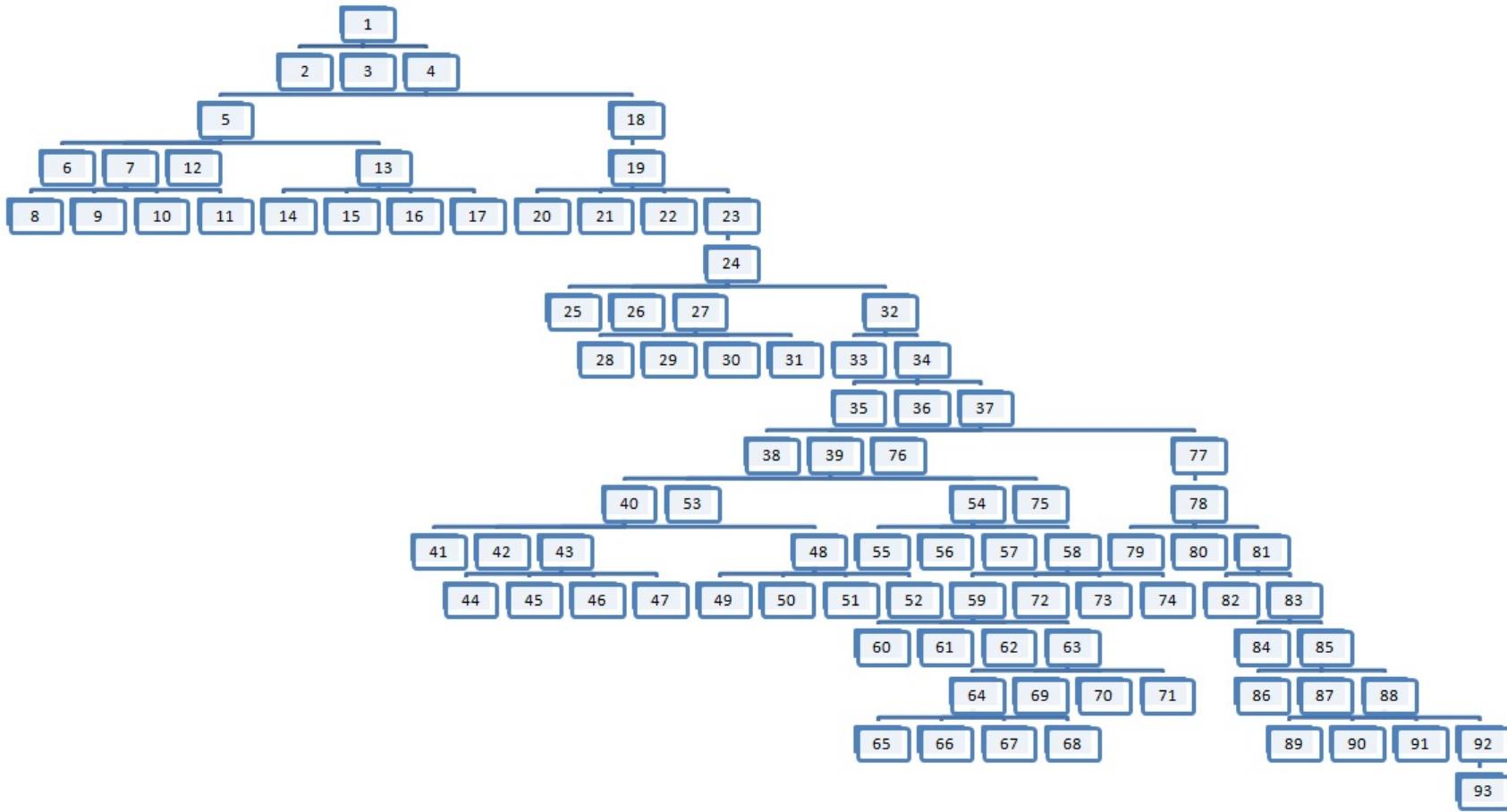
Gambar 3.8: *State 23*

21. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 21*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
22. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 22*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
23. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 23*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.8. Algoritma telah selesai mengisi baris ke-1, sehingga bisa maju ke baris berikutnya.
24. Langkah-langkah di atas diulang untuk mengisi sel-sel pada baris-baris selanjutnya. Algoritma *backtracking* berhasil mengisi semua sel dalam permainan teka-teki Calcudoku ini dengan benar pada *state 93*, seperti dapat dilihat pada Gambar 3.9.

Algoritma ini mencapai solusinya pada state 93, seperti pada *state space tree* yang digambarkan dalam Gambar 3.10. *State space tree* ini telah mencapai simpul tujuannya, yaitu simpul 93, dengan jalur 3-2-1-4-1-4-2-3-4-1-3-2-2-3-4-1. Penjelasan tentang analisis algoritma *backtracking* secara lengkap dapat dilihat di Lampiran A.

³ 3	⁸⁺ 2	³⁻ 1	4
7+ 1	4	2	3
4	⁸⁺ 1	⁸⁺ 3	2
2	3	4	¹ 1

Gambar 3.9: *State 93*



Gambar 3.10: *State space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.1

4-	1-		1-	15+	
	30*				1-
2-	2/		1-	3/	
		24*			1
5+			7+	60*	
	1-				

Gambar 3.11: Contoh permainan teka-teki Calcudoku dengan ukuran *grid* 6 x 6 yang belum diselesaikan, seperti yang digambarkan pada Gambar 1.2. [2]

3.2 Analisis Algoritma *Hybrid Genetic*

Langkah-langkah penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *backtracking* secara umum adalah sebagai berikut:

1. *Solver* akan mencoba menyelesaikan permainan dengan algoritma *rule based*.
 - (a) Algoritma *rule based* dimulai dengan mengaplikasikan aturan logika *single square* dan aturan logika *killer combination*. Kedua aturan logika ini hanya diaplikasikan sekali, yaitu di awal algoritma.
 - (b) Algoritma lalu mengaplikasikan aturan logika *naked subset* dan aturan logika *hidden single*. Kedua aturan logika ini diulang sampai algoritma tidak bisa lagi mengisi sel-sel dalam *grid* atau sampai semua sel dalam *grid* sudah terisi dengan benar.
 - (c) Algoritma *hybrid genetic* selesai jika semua sel dalam *grid* sudah terisi dengan benar.
 - (d) Algoritma genetik akan dimulai jika ada sel-sel di dalam *grid* yang masih kosong.
2. Jika algoritma *rule based* tidak berhasil mengisi semua sel dalam *grid* dengan benar, maka *solver* akan mencoba menyelesaikan permainan dengan algoritma genetik.
 - (a) Algoritma genetik dimulai dengan membangkitkan generasi awal secara acak.
 - (b) Setiap kromosom dalam sebuah generasi dihitung nilai kelayakannya. Nilai kelayakan untuk sebuah kromosom adalah jumlah sel yang sudah diisi dengan benar dibagi dengan jumlah semua sel yang ada di dalam *grid*.
 - (c) Algoritma selesai jika solusi ditemukan. Solusi adalah kromosom dengan nilai kelayakan 1.
 - (d) Jika solusi tidak ditemukan, maka algoritma genetik akan membangkitkan generasi berikutnya, sampai solusi ditemukan. Generasi berikutnya dibangkitkan dari generasi sebelumnya menggunakan operator-operator algoritma genetik, seperti *elitism*, kawin silang, dan mutasi.

Untuk mengilustrasikan cara kerja algoritma *hybrid genetic*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar 3.11 sebagai contoh. Algoritma *hybrid genetic* dimulai dengan mencoba menyelesaikan permainan teka-teki Calcudoku dengan algoritma *rule based*.

4-	1-		1-	15+	
	30*				1-
2-	2/		1-	3/	
		24*			¹ 1
5+			7+	60*	
	1-				

Gambar 3.12: Permainan teka-teki Calcudoku setelah diselesaikan dengan algoritma *rule based*

3.2.1 Algoritma *Rule Based*

Sel pada baris ke-4 dan kolom ke-6 adalah bagian dari sebuah *cage* yang berukuran hanya 1 sel, dan oleh karena itu, angka tujuan dari sel tersebut adalah angka tujuan dari *cage* tersebut (aturan *single square*). Angka tujuan dari *cage* tersebut adalah 1, dan oleh karena itu sel tersebut dapat langsung diisi dengan angka 1, seperti dapat dilihat pada Gambar 3.12.

Sayangnya, algoritma *rule based* gagal dalam mengisi sel-sel lainnya berdasarkan aturan-aturan yang telah didefinisikan setelah beberapa kali percobaan, sehingga algoritma *hybrid genetic* akan mencoba menyelesaikan teka-teki Calcudoku dengan algoritma genetik.

3.2.2 Algoritma Genetik

Dalam contoh ini, parameter-parameter untuk algoritma genetik yang akan digunakan untuk teka-teki Calcudoku ini ditunjukkan pada Tabel 3.1. Dalam kasus ini, parameter ditentukan oleh pembuat program (penulis). Setiap generasi terdiri dari 12 kromosom. $40\% \times 12 \approx 5$ kromosom diambil dari generasi sebelumnya (*elitism*). $50\% \times 12 \approx 6$ kromosom adalah hasil dari pembentukan kromosom-kromosom baru dengan operasi kawin silang, dan $10\% \times 12 \approx 1$ kromosom adalah hasil dari pembentukan kromosom-kromosom baru dengan operasi mutasi. Untuk mengilustrasikan cara kerja algoritma genetik, hanya 3 generasi pertama yang akan dibahas.

Untuk menjaga agar jumlah populasi dalam sebuah generasi tetap konsisten, maka sebaiknya total nilai probabilitas *elitism*, nilai probabilitas kawin silang, dan nilai probabilitas mutasi sama dengan 100%. Jika kurang dari 100%, maka beberapa kromosom dari generasi sebelumnya akan ditambahkan ke generasi berikutnya secara acak. Jika lebih dari 100%, maka beberapa kromosom dari generasi berikutnya akan dibuang secara acak.

Setiap sel mempunyai nilai kelayakan. Nilai kelayakan dari sebuah sel akan bernilai 1 jika nilai dari semua sel yang merupakan bagian dari *cage* yang salah satu selnya adalah sel tersebut menghasilkan nilai tujuan setelah dihitung menggunakan operator yang telah ditentukan dan tidak ada pengulangan angka di dalam baris tersebut maupun kolom tersebut, dan bernilai 0 jika nilai dari semua sel yang merupakan bagian dari *cage* yang salah satu selnya adalah sel tersebut tidak menghasilkan nilai tujuan setelah dihitung menggunakan operator yang telah ditentukan atau ada pengulangan angka di dalam baris tersebut maupun kolom tersebut. Nilai kelayakan sel untuk setiap sel dalam sebuah baris dijumlahkan, lalu dibagi dengan jumlah kolom dalam baris tersebut, dan hasilnya adalah nilai kelayakan baris. Nilai kelayakan baris untuk setiap baris dalam sebuah teka-teki dijumlahkan, lalu dibagi dengan jumlah baris dalam teka-teki tersebut, dan hasilnya adalah nilai kelayakan teka-teki.

Algoritma genetik dimulai dengan membangkitkan kromosom-kromosom baru sebanyak ukuran populasi yang telah ditentukan. Dalam contoh ini, ukuran populasi adalah 12, maka algoritma akan membangkitkan 12 kromosom baru. Ke-12 kromosom awal ini adalah bagian dari generasi pertama.

Tabel 3.1: Tabel parameter untuk algoritma genetik yang akan digunakan untuk menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.12

Parameter	Nilai
Ukuran Populasi	12
Probabilitas <i>Elitism</i>	40%
Probabilitas Kawin Silang	50%
Probabilitas Mutasi	10%

1. Gambar 3.13 menggambarkan Kromosom 1.
2. Gambar 3.14 menggambarkan Kromosom 2.
3. Gambar 3.15 menggambarkan Kromosom 3.
4. Gambar 3.16 menggambarkan Kromosom 4.
5. Gambar 3.17 menggambarkan Kromosom 5.
6. Gambar 3.18 menggambarkan Kromosom 6.
7. Gambar 3.19 menggambarkan Kromosom 7.
8. Gambar 3.20 menggambarkan Kromosom 8.
9. Gambar 3.21 menggambarkan Kromosom 9.
10. Gambar 3.22 menggambarkan Kromosom 10.
11. Gambar 3.23 menggambarkan Kromosom 11.
12. Gambar 3.24 menggambarkan Kromosom 12.

Tabel 3.2: Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1

Nomor Kromosom	Nilai Kelayakan
1	0,3333
2	0,3056
3	0,25
4	0,2222
5	0,4444
6	0,1389
7	0,3889
8	0,25
9	0,1389
10	0,3056
11	0,3889
12	0,5556

Berdasarkan nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1 yang ditampilkan pada Tabel 3.2, 5 kromosom terbaik akan diambil untuk menjadi bagian dari Generasi ke-2. Ke-5 kromosom yang terpilih adalah:

1. Kromosom 12
2. Kromosom 5
3. Kromosom 7
4. Kromosom 11
5. Kromosom 1

⁴⁻ 1	¹⁻ 3	5	¹⁻ 6	¹⁵⁺ 4	2
3	^{30*} 4	2	5	1	¹⁻ 6
²⁻ 2	^{2/} 1	6	¹⁻ 4	^{3/} 3	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	6	1	⁷⁺ 2	^{60*} 5	3
5	¹⁻ 2	3	1	6	4

Gambar 3.13: Kromosom 1 dalam Generasi ke-1

⁴⁻ 3	¹⁻ 2	1	¹⁻ 6	¹⁵⁺ 5	4
1	^{30*} 6	2	5	4	¹⁻ 3
²⁻ 5	^{2/} 3	6	¹⁻ 4	^{3/} 1	2
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	3	⁷⁺ 2	^{60*} 6	5
2	¹⁻ 4	5	1	3	6

Gambar 3.14: Kromosom 2 dalam Generasi ke-1

⁴⁻ 4	¹⁻ 3	6	¹⁻ 2	¹⁵⁺ 1	5
6	^{30*} 5	1	3	2	¹⁻ 4
²⁻ 5	^{2/} 1	4	¹⁻ 6	^{3/} 3	2
3	6	^{24*} 2	5	4	¹⁻ 1
⁵⁺ 1	2	3	⁷⁺ 4	^{60*} 5	6
2	¹⁻ 4	5	1	6	3

Gambar 3.15: Kromosom 3 dalam Generasi ke-1

⁴⁻ 5	¹⁻ 1	4	¹⁻ 6	¹⁵⁺ 2	3
1	^{30*} 2	3	4	5	¹⁻ 6
²⁻ 6	^{2/} 3	5	¹⁻ 2	^{3/} 1	4
2	4	^{24*} 6	5	3	¹⁻ 1
⁵⁺ 4	5	1	⁷⁺ 6	^{60*} 3	2
3	¹⁻ 6	2	1	4	5

Gambar 3.16: Kromosom 4 dalam Generasi ke-1

⁴⁻ 5	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 1	4
1	^{30*} 6	3	5	4	¹⁻ 2
²⁻ 3	^{2/} 2	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	6	⁷⁺ 2	^{60*} 5	3
2	¹⁻ 4	5	1	3	6

Gambar 3.17: Kromosom 5 dalam Generasi ke-1

⁴⁻ 6	¹⁻ 3	5	¹⁻ 2	¹⁵⁺ 1	4
5	^{30*} 1	4	6	2	¹⁻ 3
²⁻ 2	^{2/} 4	6	¹⁻ 1	^{3/} 3	5
3	6	^{24*} 2	5	4	¹⁻ 1
⁵⁺ 1	2	3	⁷⁺ 4	^{60*} 5	6
4	¹⁻ 5	1	3	6	2

Gambar 3.18: Kromosom 6 dalam Generasi ke-1

⁴⁻ 1	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 4	5
3	^{30*} 4	1	5	6	¹⁻ 2
²⁻ 5	^{2/} 2	6	¹⁻ 4	^{3/} 1	3
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	3	1	⁷⁺ 2	^{60*} 5	6
2	¹⁻ 6	5	1	3	4

Gambar 3.19: Kromosom 7 dalam Generasi ke-1

⁴⁻ 3	¹⁻ 1	5	¹⁻ 6	¹⁵⁺ 4	2
2	^{30*} 6	3	5	1	¹⁻ 4
²⁻ 1	^{2/} 2	6	¹⁻ 4	^{3/} 3	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 5	4	1	⁷⁺ 2	^{60*} 6	3
4	¹⁻ 3	2	1	5	6

Gambar 3.20: Kromosom 8 dalam Generasi ke-1

⁴⁻ 4	¹⁻ 6	5	¹⁻ 3	¹⁵⁺ 1	2
3	^{30*} 5	1	6	2	¹⁻ 4
²⁻ 6	^{2/} 1	4	¹⁻ 2	^{3/} 3	5
2	3	^{24*} 6	5	4	¹⁻ 1
⁵⁺ 1	2	3	⁷⁺ 4	^{60*} 5	6
5	¹⁻ 4	2	1	6	3

Gambar 3.21: Kromosom 9 dalam Generasi ke-1

⁴⁻ 5	¹⁻ 1	3	¹⁻ 6	¹⁵⁺ 4	2
3	^{30*} 6	2	5	1	¹⁻ 4
²⁻ 2	^{2/} 3	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 1	4	6	⁷⁺ 2	^{60*} 5	3
4	¹⁻ 2	5	1	3	6

Gambar 3.22: Kromosom 10 dalam Generasi ke-1

⁴⁻ 5	¹⁻ 1	6	¹⁻ 2	¹⁵⁺ 4	3
1	^{30*} 2	3	4	5	¹⁻ 6
²⁻ 4	^{2/} 3	5	¹⁻ 6	^{3/} 1	2
6	4	^{24*} 2	5	3	¹⁻ 1
⁵⁺ 2	5	1	⁷⁺ 3	^{60*} 6	4
3	¹⁻ 6	4	1	2	5

Gambar 3.23: Kromosom 11 dalam Generasi ke-1

⁴⁻ 1	¹⁻ 5	6	¹⁻ 4	¹⁵⁺ 3	2
3	^{30*} 6	1	2	4	¹⁻ 5
²⁻ 4	^{2/} 1	5	¹⁻ 3	^{3/} 2	6
2	3	^{24*} 4	5	6	¹⁻ 1
⁵⁺ 5	4	2	⁷⁺ 6	^{60*} 1	3
6	¹⁻ 2	3	1	5	4

Gambar 3.24: Kromosom 12 dalam Generasi ke-1

Untuk Generasi ke-2, 5 kromosom adalah 5 kromosom terbaik dari Generasi ke-1, 6 kromosom adalah hasil kawin silang dari 2 kromosom dari Generasi ke-1, dan 1 kromosom adalah hasil mutasi dari 1 kromosom dari Generasi ke-1.

1. Gambar 3.25 menggambarkan Kromosom 1, yaitu Kromosom 12 dari Generasi ke-1.
2. Gambar 3.26 menggambarkan Kromosom 2, yaitu Kromosom 5 dari Generasi ke-1.
3. Gambar 3.27 menggambarkan Kromosom 3, yaitu Kromosom 7 dari Generasi ke-1.
4. Gambar 3.28 menggambarkan Kromosom 4, yaitu Kromosom 11 dari Generasi ke-1.
5. Gambar 3.29 menggambarkan Kromosom 5, yaitu Kromosom 1 dari Generasi ke-1.
6. Gambar 3.30 menggambarkan Kromosom 6, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 12 dari Generasi ke-1.
7. Gambar 3.31 menggambarkan Kromosom 7, yaitu hasil kawin silang dari Kromosom 3 dan Kromosom 8 dari Generasi ke-1.
8. Gambar 3.32 menggambarkan Kromosom 8, yaitu hasil kawin silang dari Kromosom 7 dan Kromosom 10 dari Generasi ke-1.
9. Gambar 3.33 menggambarkan Kromosom 9, yaitu hasil kawin silang dari Kromosom 7 dan Kromosom 10 dari Generasi ke-1.
10. Gambar 3.34 menggambarkan Kromosom 10, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 5 dari Generasi ke-1.
11. Gambar 3.35 menggambarkan Kromosom 11, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 5 dari Generasi ke-1.
12. Gambar 3.36 menggambarkan Kromosom 12, yaitu hasil mutasi dari Kromosom 12 dari Generasi ke-1.

Tabel 3.3: Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-2

Nomor Kromosom	Nilai Kelayakan
1	0,5556
2	0,4444
3	0,3889
4	0,3889
5	0,3333
6	0,1944
7	0,1389
8	0,0833
9	0,25
10	0,1944
11	0,1944
12	0,5

Berdasarkan nilai kelayakan untuk kromosom-kromosom pada Generasi ke-2 yang ditampilkan pada Tabel 3.3, 5 kromosom terbaik akan diambil untuk menjadi bagian dari Generasi ke-3. Ke-5 kromosom yang terpilih adalah:

1. Kromosom 1
2. Kromosom 12
3. Kromosom 2
4. Kromosom 3
5. Kromosom 4

1	¹⁻ 5	6	4	¹⁵⁺ 3	2
3	^{30*} 6	1	2	4	¹⁻ 5
²⁻ 4	^{2/} 1	5	¹⁻ 3	^{3/} 2	6
2	3	^{24*} 4	5	6	¹ 1
⁵⁺ 5	4	2	⁷⁺ 6	^{60*} 1	3
6	¹⁻ 2	3	1	5	4

Gambar 3.25: Kromosom 1 dalam Generasi ke-2

⁴⁻ 5	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 1	4
1	^{30*} 6	3	5	4	¹⁻ 2
²⁻ 3	^{2/} 2	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹ 1
⁵⁺ 4	1	6	⁷⁺ 2	^{60*} 5	3
2	¹⁻ 4	5	1	3	6

Gambar 3.26: Kromosom 2 dalam Generasi ke-2

⁴⁻ 1	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 4	5
3	^{30*} 4	1	5	6	¹⁻ 2
²⁻ 5	^{2/} 2	6	¹⁻ 4	^{3/} 1	3
6	5	^{24*} 4	3	2	¹ 1
⁵⁺ 4	1	3	⁷⁺ 2	^{60*} 5	6
2	¹⁻ 6	5	1	3	4

Gambar 3.27: Kromosom 3 dalam Generasi ke-2

⁴⁻ 5	¹⁻ 1	6	¹⁻ 2	¹⁵⁺ 4	3
1	^{30*} 2	3	4	5	¹⁻ 6
²⁻ 4	^{2/} 3	5	¹⁻ 6	^{3/} 1	2
6	4	^{24*} 2	5	3	¹⁻ 1
⁵⁺ 2	5	1	⁷⁺ 3	^{60*} 6	4
3	¹⁻ 6	4	1	2	5

Gambar 3.28: Kromosom 4 dalam Generasi ke-2

⁴⁻ 1	¹⁻ 3	5	¹⁻ 6	¹⁵⁺ 4	2
3	^{30*} 4	2	5	1	¹⁻ 6
²⁻ 2	^{2/} 1	6	¹⁻ 4	^{3/} 3	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	6	1	⁷⁺ 2	^{60*} 5	3
5	¹⁻ 2	3	1	6	4

Gambar 3.29: Kromosom 5 dalam Generasi ke-2

⁴⁻ 3	¹⁻ 2	1	¹⁻ 6	¹⁵⁺ 5	4
1	^{30*} 6	2	5	4	¹⁻ 3
²⁻ 2	^{2/} 3	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 1	4	6	⁷⁺ 2	^{60*} 5	3
2	¹⁻ 4	5	1	3	6

Gambar 3.30: Kromosom 6 dalam Generasi ke-2

⁴⁻ 5	¹⁻ 1	3	¹⁻ 6	¹⁵⁺ 4	2
3	^{30*} 6	2	5	1	¹⁻ 4
²⁻ 5	^{2/} 3	6	¹⁻ 4	^{3/} 1	2
6	5	^{24*} 4	3	2	¹ 1
⁵⁺ 4	1	3	⁷⁺ 2	^{60*} 6	5
4	¹⁻ 2	5	1	3	6

Gambar 3.31: Kromosom 7 dalam Generasi ke-2

⁴⁻ 1	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 4	5
3	^{30*} 4	1	5	6	¹⁻ 2
²⁻ 2	^{2/} 3	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹ 1
⁵⁺ 1	4	6	⁷⁺ 2	^{60*} 5	3
2	¹⁻ 6	1	5	3	4

Gambar 3.32: Kromosom 8 dalam Generasi ke-2

⁴⁻ 5	¹⁻ 1	3	¹⁻ 6	¹⁵⁺ 4	2
3	^{30*} 6	2	5	1	¹⁻ 4
²⁻ 5	^{2/} 2	6	¹⁻ 4	^{3/} 1	3
6	5	^{24*} 4	3	2	¹ 1
⁵⁺ 4	3	1	⁷⁺ 2	^{60*} 5	6
4	¹⁻ 2	5	1	3	6

Gambar 3.33: Kromosom 9 dalam Generasi ke-2

⁴⁻ 3	¹⁻ 2	1	¹⁻ 6	¹⁵⁺ 5	4
1	^{30*} 6	3	5	4	¹⁻ 2
²⁻ 3	^{2/} 2	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	3	⁷⁺ 2	^{60*} 6	5
2	¹⁻ 4	5	1	3	6

Gambar 3.34: Kromosom 10 dalam Generasi ke-2

⁴⁻ 5	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 1	4
1	^{30*} 6	2	5	4	¹⁻ 3
²⁻ 5	^{2/} 3	6	¹⁻ 4	^{3/} 1	2
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	6	⁷⁺ 2	^{60*} 5	3
2	¹⁻ 4	5	1	3	6

Gambar 3.35: Kromosom 11 dalam Generasi ke-2

⁴⁻ 1	¹⁻ 5	6	¹⁻ 4	¹⁵⁺ 3	2
3	^{30*} 6	1	2	4	¹⁻ 5
²⁻ 4	^{2/} 1	5	¹⁻ 3	^{3/} 2	6
2	3	^{24*} 4	5	6	¹⁻ 1
⁵⁺ 5	4	2	⁷⁺ 6	^{60*} 1	3
1	¹⁻ 2	3	6	5	4

Gambar 3.36: Kromosom 12 dalam Generasi ke-2

Untuk Generasi ke-3, 5 kromosom adalah 5 kromosom terbaik dari Generasi ke-2, 6 kromosom adalah hasil kawin silang dari 2 kromosom dari Generasi ke-2, dan 1 kromosom adalah hasil mutasi dari 1 kromosom dari Generasi ke-2.

1. Gambar 3.37 menggambarkan Kromosom 1, yaitu Kromosom 1 dari Generasi ke-2.
2. Gambar 3.38 menggambarkan Kromosom 2, yaitu Kromosom 12 dari Generasi ke-2.
3. Gambar 3.39 menggambarkan Kromosom 3, yaitu Kromosom 2 dari Generasi ke-2.
4. Gambar 3.40 menggambarkan Kromosom 4, yaitu Kromosom 3 dari Generasi ke-2.
5. Gambar 3.41 menggambarkan Kromosom 5, yaitu Kromosom 4 dari Generasi ke-2.
6. Gambar 3.42 menggambarkan Kromosom 6, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 12 dari Generasi ke-2.
7. Gambar 3.43 menggambarkan Kromosom 7, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 12 dari Generasi ke-2.
8. Gambar 3.44 menggambarkan Kromosom 8, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 9 dari Generasi ke-2.
9. Gambar 3.45 menggambarkan Kromosom 9, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 9 dari Generasi ke-2.
10. Gambar 3.46 menggambarkan Kromosom 10, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 6 dari Generasi ke-2.
11. Gambar 3.47 menggambarkan Kromosom 11, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 6 dari Generasi ke-2.
12. Gambar 3.48 menggambarkan Kromosom 12, yaitu hasil mutasi dari Kromosom 2 dari Generasi ke-2.

Tabel 3.4: Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-3

Nomor Kromosom	Nilai Kelayakan
1	0,5556
2	0,5
3	0,4444
4	0,3889
5	0,3889
6	0,2778
7	0,1389
8	0,1389
9	0,1389
10	0,1389
11	0,1944
12	0,3889

Berdasarkan nilai kelayakan untuk kromosom-kromosom pada Generasi ke-3 yang ditampilkan pada Tabel 3.4, 5 kromosom terbaik akan diambil untuk menjadi bagian dari Generasi ke-4. Ke-5 kromosom yang terpilih adalah:

1. Kromosom 1
2. Kromosom 2
3. Kromosom 3
4. Kromosom 4
5. Kromosom 12

Proses ini diulang untuk menghasilkan generasi-generasi berikutnya, sampai algoritma genetik dapat menemukan solusi dari teka-teki Calcudoku tersebut.

⁴⁻ 1	¹⁻ 5	6	¹⁻ 4	¹⁵⁺ 3	2
3	^{30*} 6	1	2	4	¹⁻ 5
²⁻ 4	^{2/} 1	5	¹⁻ 3	^{3/} 2	6
2	3	^{24*} 4	5	6	¹⁻ 1
⁵⁺ 5	4	2	⁷⁺ 6	^{60*} 1	3
6	¹⁻ 2	3	1	5	4

Gambar 3.37: Kromosom 1 dalam Generasi ke-3

⁴⁻ 1	¹⁻ 5	6	4	¹⁵⁺ 3	2
3	^{30*} 6	1	2	4	¹⁻ 5
²⁻ 4	^{2/} 1	5	¹⁻ 3	^{3/} 2	6
2	3	^{24*} 4	5	6	¹⁻ 1
⁵⁺ 5	4	2	⁷⁺ 6	^{60*} 1	3
1	¹⁻ 2	3	6	5	4

Gambar 3.38: Kromosom 2 dalam Generasi ke-3

⁴⁻ 5	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 1	4
1	^{30*} 6	3	5	4	¹⁻ 2
²⁻ 3	^{2/} 2	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	6	⁷⁺ 2	^{60*} 5	3
2	¹⁻ 4	5	1	3	6

Gambar 3.39: Kromosom 3 dalam Generasi ke-3

⁴⁻ 1	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 4	5
3	^{30*} 4	1	5	6	¹⁻ 2
²⁻ 5	^{2/} 2	6	¹⁻ 4	^{3/} 1	3
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	3	⁷⁺ 2	^{60*} 5	6
2	¹⁻ 6	5	1	3	4

Gambar 3.40: Kromosom 4 dalam Generasi ke-3

⁴⁻ 5	¹⁻ 1	6	¹⁻ 2	¹⁵⁺ 4	3
1	^{30*} 2	3	4	5	¹⁻ 6
²⁻ 4	^{2/} 3	5	¹⁻ 6	^{3/} 1	2
6	4	^{24*} 2	5	3	¹⁻ 1
⁵⁺ 2	5	1	⁷⁺ 3	^{60*} 6	4
3	¹⁻ 6	4	1	2	5

Gambar 3.41: Kromosom 5 dalam Generasi ke-3

⁴⁻ 5	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 1	4
3	^{30*} 6	1	2	4	¹⁻ 5
²⁻ 4	^{2/} 1	5	¹⁻ 3	^{3/} 2	6
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	6	⁷⁺ 2	^{60*} 5	3
1	¹⁻ 2	3	6	5	4

Gambar 3.42: Kromosom 6 dalam Generasi ke-3

⁴⁻ 1	¹⁻ 5	6	¹⁻ 4	¹⁵⁺ 3	2
1	^{30*} 6	3	5	4	¹⁻ 2
²⁻ 3	^{2/} 2	1	¹⁻ 4	^{3/} 6	5
2	3	^{24*} 4	5	6	¹⁻ 1
⁵⁺ 5	4	2	⁷⁺ 6	^{60*} 3	1
2	¹⁻ 4	5	1	3	6

Gambar 3.43: Kromosom 7 dalam Generasi ke-3

⁴⁻ 1	¹⁻ 3	5	¹⁻ 6	¹⁵⁺ 4	2
3	^{30*} 4	2	5	1	¹⁻ 6
²⁻ 2	^{2/} 1	6	¹⁻ 4	^{3/} 3	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	3	1	⁷⁺ 2	^{60*} 5	6
4	¹⁻ 2	5	1	3	6

Gambar 3.44: Kromosom 8 dalam Generasi ke-3

⁴⁻ 5	¹⁻ 1	3	¹⁻ 6	¹⁵⁺ 4	2
3	^{30*} 6	2	5	1	¹⁻ 4
²⁻ 5	^{2/} 2	6	¹⁻ 4	^{3/} 1	3
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	6	1	⁷⁺ 2	^{60*} 5	3
5	¹⁻ 2	3	1	6	4

Gambar 3.45: Kromosom 9 dalam Generasi ke-3

⁴⁻ 1	¹⁻ 3	5	¹⁻ 6	¹⁵⁺ 4	2
1	^{30*} 6	2	5	4	¹⁻ 3
²⁻ 2	^{2/} 1	6	¹⁻ 4	^{3/} 3	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 1	4	6	⁷⁺ 2	^{60*} 5	3
2	¹⁻ 4	5	1	3	6

Gambar 3.46: Kromosom 10 dalam Generasi ke-3

⁴⁻ 3	¹⁻ 2	1	¹⁻ 6	¹⁵⁺ 5	4
3	^{30*} 4	2	5	1	¹⁻ 6
²⁻ 2	^{2/} 3	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	6	1	⁷⁺ 2	^{60*} 5	3
5	¹⁻ 2	3	1	6	4

Gambar 3.47: Kromosom 11 dalam Generasi ke-3

⁴⁻ 5	¹⁻ 3	2	¹⁻ 6	¹⁵⁺ 1	4
1	^{30*} 6	3	5	4	¹⁻ 2
²⁻ 3	^{2/} 2	1	¹⁻ 4	^{3/} 6	5
6	5	^{24*} 4	3	2	¹⁻ 1
⁵⁺ 4	1	6	⁷⁺ 2	^{60*} 5	3
1	¹⁻ 4	5	2	3	6

Gambar 3.48: Kromosom 12 dalam Generasi ke-3

3.3 Perangkat Lunak

Berdasarkan landasan teori dan analisis algoritma *backtracking* dan *hybrid genetic* untuk menyelesaikan permainan teka-teki Calcudoku yang telah dilakukan, perangkat lunak Calcudoku akan dibuat. Perangkat lunak ini akan menerima masukan dalam bentuk *file* yang berisi:

1. Ukuran *grid*.
2. Jumlah *cage*.
3. Matriks *cage assignment*, yang merepresentasikan posisi dari setiap *cage* dalam *grid*.
4. Matriks *cage objectives*, yang berisikan angka tujuan dan operasi matematika yang telah ditentukan untuk setiap *cage*.

Umumnya, setiap soal hanya mempunyai satu solusi. Tetapi, ada juga soal yang tidak ada solusinya. Hal ini disebabkan karena kesalahan dalam merancang matriks *cage assignment* dan matriks *cage objectives*. Jika *file* soal yang dibuka oleh perangkat lunak Calcudoku ternyata tidak ada solusinya, maka *solver* akan selalu gagal dalam menyelesaikan permainan.

Perangkat lunak ini akan menghasilkan keluaran berupa antarmuka grafis permainan teka-teki Calcudoku berdasarkan isi *file* yang di-*load* oleh pengguna. Permainan ini dapat diselesaikan oleh pengguna dengan usahanya sendiri, atau menggunakan salah satu dari dua *solver* yang disediakan. Kedua *solver* tersebut yaitu:

1. Algoritma *backtracking*, dan
2. Algoritma *hybrid genetic*.

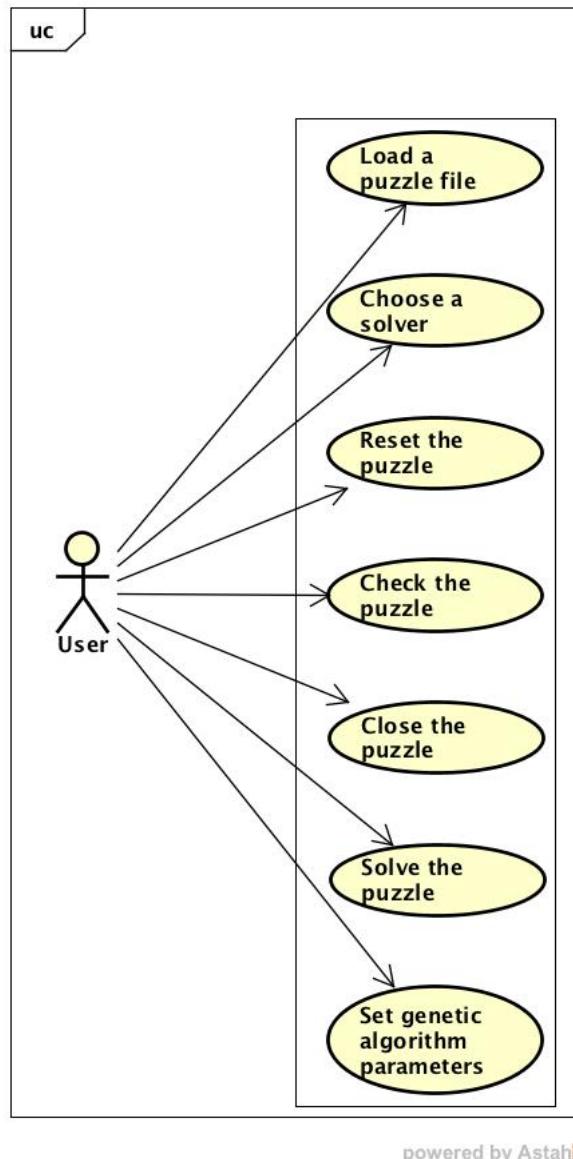
Pengguna dapat me-*load* file masukan untuk memulai permainan, me-*reset* permainan untuk mengulang permainan berdasarkan *file* masukan yang sudah di-*load* dari awal, dan menutup *file* masukan untuk mengakhiri permainan, atau jika ingin me-*load* *file* masukan yang lain. Pengguna juga dapat meminta perangkat lunak untuk memeriksa permainan jika ada masukan yang salah di dalam *grid*, misalnya ada angka yang berulang dalam sebuah baris atau kolom, atau angka-angka dalam sebuah *cage* tidak mencapai angka tujuan yang ditentukan setelah dihitung dengan operasi matematika yang ditentukan. Pemain juga dapat mengatur nilai dari parameter-parameter untuk algoritma genetik.

Kebutuhan-kebutuhan yang diperlukan oleh perangkat lunak ini akan dijelaskan menggunakan diagram *use case*, dan skenario.

3.3.1 Diagram *Use Case* dan Skenario

Diagram *use case* adalah diagram yang menggambarkan interaksi antara sistem (perangkat lunak) dengan pengguna. Berdasarkan analisis perangkat lunak yang telah dilakukan, maka pengguna dapat:

1. Membuka file masukan untuk memulai permainan.
2. Memilih salah satu dari dua *solver* yang disediakan untuk menyelesaikan permainan berdasarkan *file* yang sudah di-*load*.
3. Me-*reset* permainan untuk mengulang permainan berdasarkan *file* masukan yang sudah di-*load* dari awal.
4. Meminta perangkat lunak untuk memeriksa permainan jika ada masukan yang salah di dalam *grid*.



Gambar 3.49: Diagram *use case* untuk perangkat lunak permainan teka-teki Calcudoku

5. Menutup *file* masukan untuk mengakhiri permainan.
6. Menyelesaikan permainan dengan usahanya sendiri.
7. Mengatur nilai dari parameter-parameter untuk algoritma genetik.

Diagram *use case* untuk perangkat lunak permainan teka-teki Calcudoku dapat dilihat pada Gambar 3.49.

Berdasarkan diagram *use case* tersebut, skenario-skenario yang dapat dilakukan oleh pengguna adalah:

1. Membuka file masukan. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.5.
2. Memilih salah satu dari dua *solver* yang disediakan. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.6
3. Me-reset permainan. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.7.

Tabel 3.5: Skenario me-load file

Nama	Membuka file masukan
Aktor	Pengguna
Deskripsi	Memembuka file masukan untuk memulai permainan.
Kondisi Awal	Perangkat lunak belum membuka file masukan.
Kondisi Akhir	Perangkat lunak sudah membuka file masukan .
Skenario Utama	Pengguna masuk ke dalam menu "File", lalu memilih menu item "Load Puzzle File", lalu memilih file masukan yang akan dibuka, dan mengklik tombol "OK". Jika perangkat lunak sudah membuka file masukan, dan ingin membuka file masukan yang baru, akan keluar kotak dialog "Are you sure you want to load another puzzle file?", klik tombol "Yes" untuk membuka file masukan baru, atau klik tombol "No" untuk membatalkan. File masukan yang akan dibuka harus file teks.

Tabel 3.6: Skenario memilih salah satu dari dua solver yang disediakan

Nama	Memilih salah satu dari dua solver yang disediakan
Aktor	Pengguna
Deskripsi	Memilih salah satu dari dua solver yang disediakan untuk menyelesaikan permainan berdasarkan file yang sudah di-load.
Kondisi Awal	Solver belum menyelesaikan permainan.
Kondisi Akhir	Solver berhasil atau gagal dalam menyelesaikan permainan.
Skenario Utama	Pengguna masuk ke dalam menu "Solve", lalu memilih salah satu dari dua solver yang disediakan. Pemain memilih menu item "Backtracking" untuk memilih solver dengan algoritma backtracking, atau menu item "Hybrid Genetic" untuk memilih solver dengan algoritma hybrid genetic.

Tabel 3.7: Skenario me-reset permainan

Nama	Me-reset permainan
Aktor	Pengguna
Deskripsi	Me-reset permainan untuk mengulang permainan berdasarkan file masukan yang sudah di-load dari awal.
Kondisi Awal	Permainan belum di-reset, sel-sel dalam grid mungkin masih berisi angka-angka.
Kondisi Akhir	Permainan sudah di-reset, semua sel-sel dalam grid sudah dalam keadaan kosong.
Skenario Utama	Pengguna masuk ke dalam menu "File", lalu memilih menu item "Reset Puzzle File". Akan keluar kotak dialog "Are you sure you want to reset this puzzle?", klik tombol "Yes untuk me-reset permainan, atau klik tombol "No" untuk membatalkan. Jika perangkat lunak belum me-load file masukan, maka akan keluar pesan error "Puzzle file not loaded".

Tabel 3.8: Skenario meminta perangkat lunak untuk memeriksa permainan

Nama	Meminta perangkat lunak untuk memeriksa permainan
Aktor	Pengguna
Deskripsi	Meminta perangkat lunak untuk memeriksa permainan jika ada masukan yang salah di dalam <i>grid</i> .
Kondisi Awal	Permainan belum diperiksa oleh perangkat lunak.
Kondisi Akhir	Permainan sudah diperiksa oleh perangkat lunak. Pengguna akan diberitahu oleh perangkat lunak jika ada masukan yang salah di dalam <i>grid</i> , misalnya ada angka yang berulang dalam sebuah baris atau kolom, atau angka-angka dalam sebuah <i>cage</i> tidak mencapai angka tujuan yang ditentukan setelah dihitung dengan operasi matematika yang ditentukan.
Skenario Utama	Pengguna masuk ke dalam menu "File", lalu memilih menu item "Check Puzzle File". Jika perangkat lunak belum me-load file masukan, maka akan keluar pesan error "Puzzle file not loaded".

Tabel 3.9: Skenario menutup *file* masukan

Nama	Menutup <i>file</i> masukan
Aktor	Pengguna
Deskripsi	Menutup <i>file</i> masukan untuk mengakhiri permainan.
Kondisi Awal	Perangkat lunak belum menutup <i>file</i> masukan.
Kondisi Akhir	Perangkat lunak sudah menutup <i>file</i> masukan.
Skenario Utama	Pengguna masuk ke dalam menu "File", lalu memilih menu item "Close Puzzle File". Jika perangkat lunak belum me-load file masukan, maka akan keluar pesan error "Puzzle file not loaded".

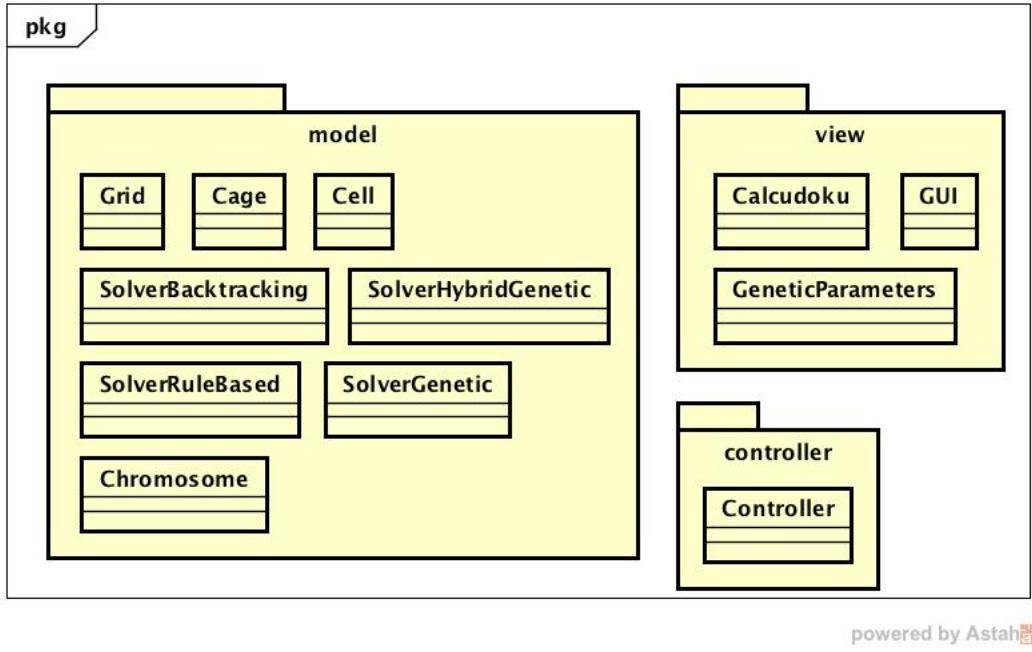
4. Meminta perangkat lunak untuk memeriksa permainan. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.8.
5. Menutup *file* masukan. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.9.
6. Menyelesaikan permainan dengan usahanya sendiri. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.10.
7. Mengatur nilai dari parameter-parameter untuk algoritma genetik. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.11.

Tabel 3.10: Skenario menyelesaikan permainan dengan usahanya sendiri

Nama	Menyelesaikan permainan dengan usahanya sendiri.
Aktor	Pengguna
Deskripsi	Pemain menyelesaikan permainan dengan usahanya sendiri. Pemain mengisikan sel-sel dalam <i>grid</i> dengan angka 1 sampai <i>n</i> , dengan <i>n</i> merupakan ukuran dari <i>grid</i> . Perangkat lunak dapat secara otomatis memeriksa <i>grid</i> jika ada masukan yang salah di dalam <i>grid</i> , misalnya ada angka yang berulang dalam sebuah baris atau kolom, atau angka-angka dalam sebuah <i>cage</i> tidak mencapai angka tujuan yang ditentukan setelah dihitung dengan operasi matematika yang ditentukan.
Kondisi Awal	Semua sel-sel dalam <i>grid</i> dalam keadaan kosong.
Kondisi Akhir	Semua sel-sel dalam <i>grid</i> sudah terisi dengan angka-angka, dengan rincian tidak ada angka yang berulang dalam sebuah baris atau kolom, dan angka-angka dalam setiap <i>cage</i> mencapai angka tujuan yang ditentukan setelah dihitung dengan operasi matematika yang ditentukan.
Skenario Utama	Pemain mengisikan sel-sel dalam <i>grid</i> dengan angka 1 sampai <i>n</i> , dengan <i>n</i> merupakan ukuran dari <i>grid</i> .

Tabel 3.11: Skenario mengatur nilai dari parameter-parameter untuk algoritma genetik

Nama	Mengatur nilai dari parameter-parameter untuk algoritma genetik
Aktor	Pengguna
Deskripsi	Pemain mengatur atau mengubah nilai dari parameter-parameter untuk algoritma genetik dengan mengisi <i>form</i> yang telah disediakan. Pemain menekan tombol "OK" untuk mengatur atau mengubah nilai dari parameter-parameter untuk algoritma genetik.
Kondisi Awal	Parameter-parameter untuk algoritma genetik belum diatur, atau sudah diatur tetapi belum diubah.
Kondisi Akhir	Parameter-parameter untuk algoritma genetik sudah diatur jika belum diatur sebelumnya, atau sudah diubah jadi sudah diatur sebelumnya.
Skenario Utama	Pemain mengisikan <i>form</i> yang telah disediakan, lalu menekan tombol "OK". Nilai untuk parameter-parameter " <i>Generations</i> " dan " <i>Population Size</i> " harus bilangan bulat. Nilai untuk parameter-parameter " <i>Elitism Rate</i> ", " <i>Crossover Rate</i> ", dan " <i>Mutation Rate</i> " harus bilangan desimal.



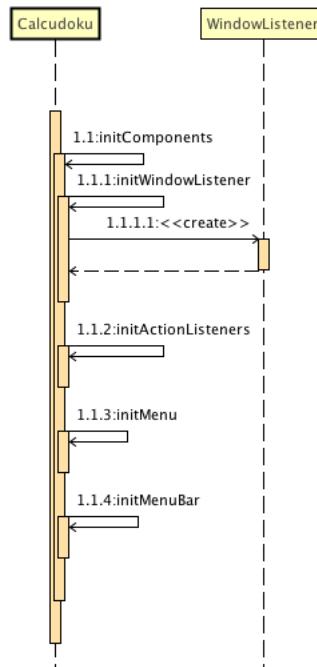
Gambar 3.50: Diagram kelas untuk perangkat lunak permainan teka-teki Calcudoku

3.3.2 Diagram Kelas

Berdasarkan diagram *use case* yang telah dibuat, maka diagram kelas dapat dibuat. Diagram kelas untuk perangkat lunak permainan teka-teki Calcudoku dapat dilihat pada Gambar 3.50.

Berdasarkan diagram kelas tersebut, kelas-kelas yang digunakan dalam perangkat lunak Calcudoku adalah:

1. *Package model*, yaitu *package* yang berisi kelas-kelas yang merepresentasikan permainan teka-teki Calcudoku. *Package* ini terdiri dari 8 kelas, yaitu:
 - (a) Kelas Grid, yaitu kelas yang merepresentasikan sebuah *grid* dalam permainan Calcudoku.
 - (b) Kelas Cell, yaitu kelas yang merepresentasikan sebuah sel dalam *grid*.
 - (c) Kelas Cage, yaitu kelas yang merepresentasikan sebuah *cage* dalam *grid*.
 - (d) Kelas SolverBacktracking, yaitu kelas yang merepresentasikan *solver* untuk permainan Calcudoku dengan algoritma *backtracking*.
 - (e) Kelas SolverHybridGenetic, yaitu kelas yang merepresentasikan *solver* untuk permainan Calcudoku dengan algoritma *hybrid genetic*.
 - (f) Kelas SolverRuleBased, yaitu kelas yang merepresentasikan *solver* untuk permainan Calcudoku dengan algoritma *rule based*, bagian pertama dari algoritma *hybrid genetic*.
 - (g) Kelas SolverGenetic, yaitu kelas yang merepresentasikan *solver* untuk permainan Calcudoku dengan algoritma genetik, bagian kedua dari algoritma *hybrid genetic*.
 - (h) Kelas Chromosome, yaitu kelas yang merepresentasikan sebuah kromosom dalam algoritma genetik.
2. *Package view*, yaitu *package* yang merepresentasikan GUI untuk permainan Calcudoku. *Package* ini terdiri dari 2 kelas, yaitu:
 - (a) Kelas Calcudoku, yaitu kelas yang merepresentasikan *frame* untuk GUI permainan Calcudoku. Kelas ini berisi menu *bar*, dan panel yang merepresentasikan *grid* untuk permainan Calcudoku (kelas GUI).



Gambar 3.51: Diagram *sequence* saat perangkat lunak dibuka

- (b) Kelas GUI, yaitu kelas yang merepresentasikan *grid* untuk permainan Calcudoku.
 - (c) Kelas GeneticParameters, yaitu kelas yang berisi *form* untuk mengatur nilai dari parameter-parameter untuk algoritma genetik.
3. *Package controller*, yaitu penghubung antara kelas-kelas yang ada di dalam *package model* dengan kelas-kelas yang ada di dalam *package view*. *Package* ini terdiri dari 1 kelas, yaitu kelas Controller. Kelas ini menghubungkan kelas-kelas yang ada di dalam *package model* dengan kelas-kelas yang ada di dalam *package view*.

Penjelasan tentang variabel-variabel dan *method-method* yang ada di dalam kelas-kelas di atas akan dijelaskan di dalam bab Perancangan.

3.3.3 Diagram *Sequence*

Diagram *sequence* adalah diagram yang menggambarkan interaksi antar kelas dalam suatu skenario.

Gambar 3.51 menunjukkan diagram *sequence* saat perangkat lunak dibuka.

Gambar 3.52 dan Gambar 3.53 menunjukkan diagram *sequence* saat menu item "Load Puzzle File" dalam menu "File" dipilih.

Gambar 3.54 dan Gambar 3.55 menunjukkan diagram *sequence* saat file permainan yang ingin dibuka dalam file chooser dipilih.

Gambar 3.56 menunjukkan diagram *sequence* saat file permainan yang sudah dipilih dibuka.

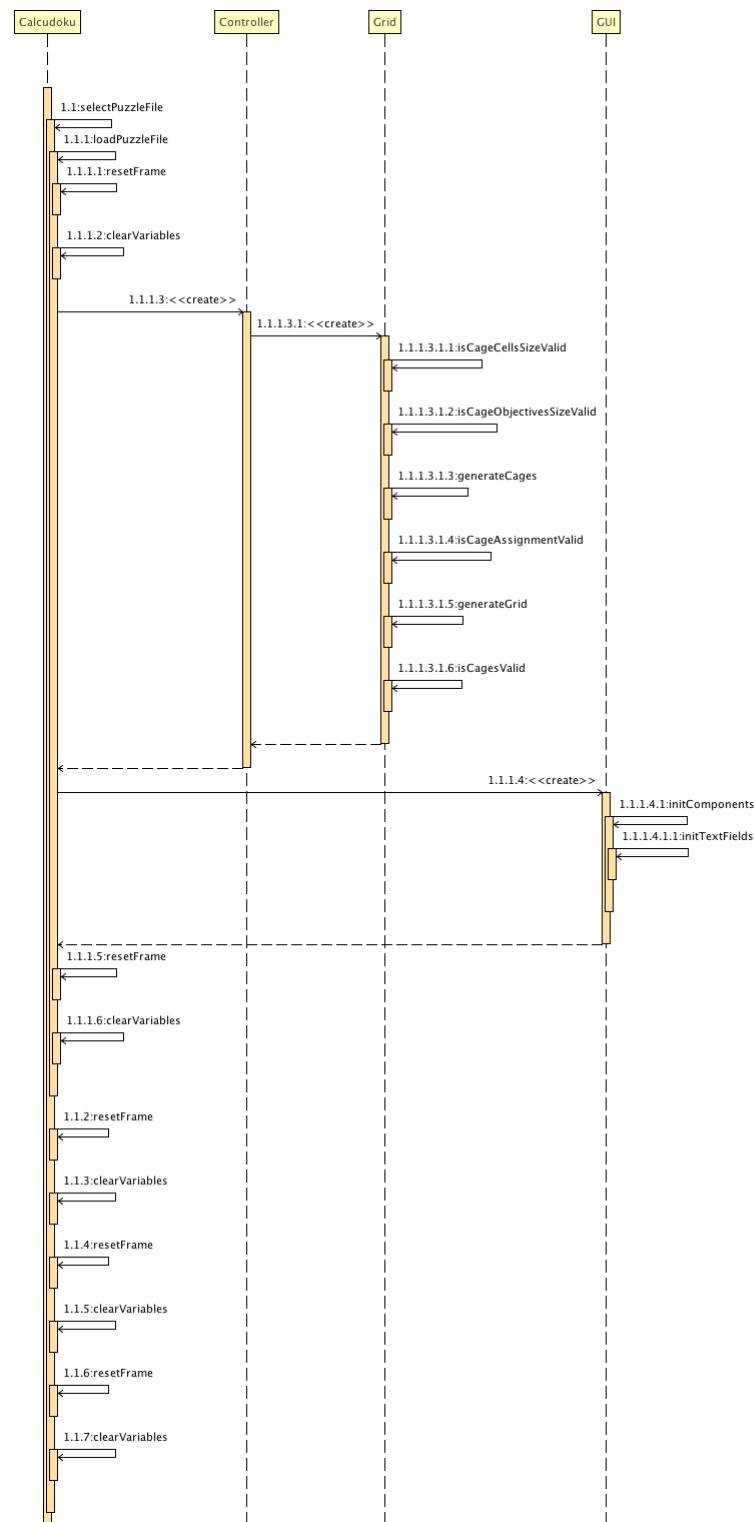
Gambar 3.57 menunjukkan diagram *sequence* saat menu item "Reset Puzzle" dalam menu "File" dipilih.

Gambar 3.58 menunjukkan diagram *sequence* saat menu item "Close Puzzle File" dalam menu "File" dipilih.

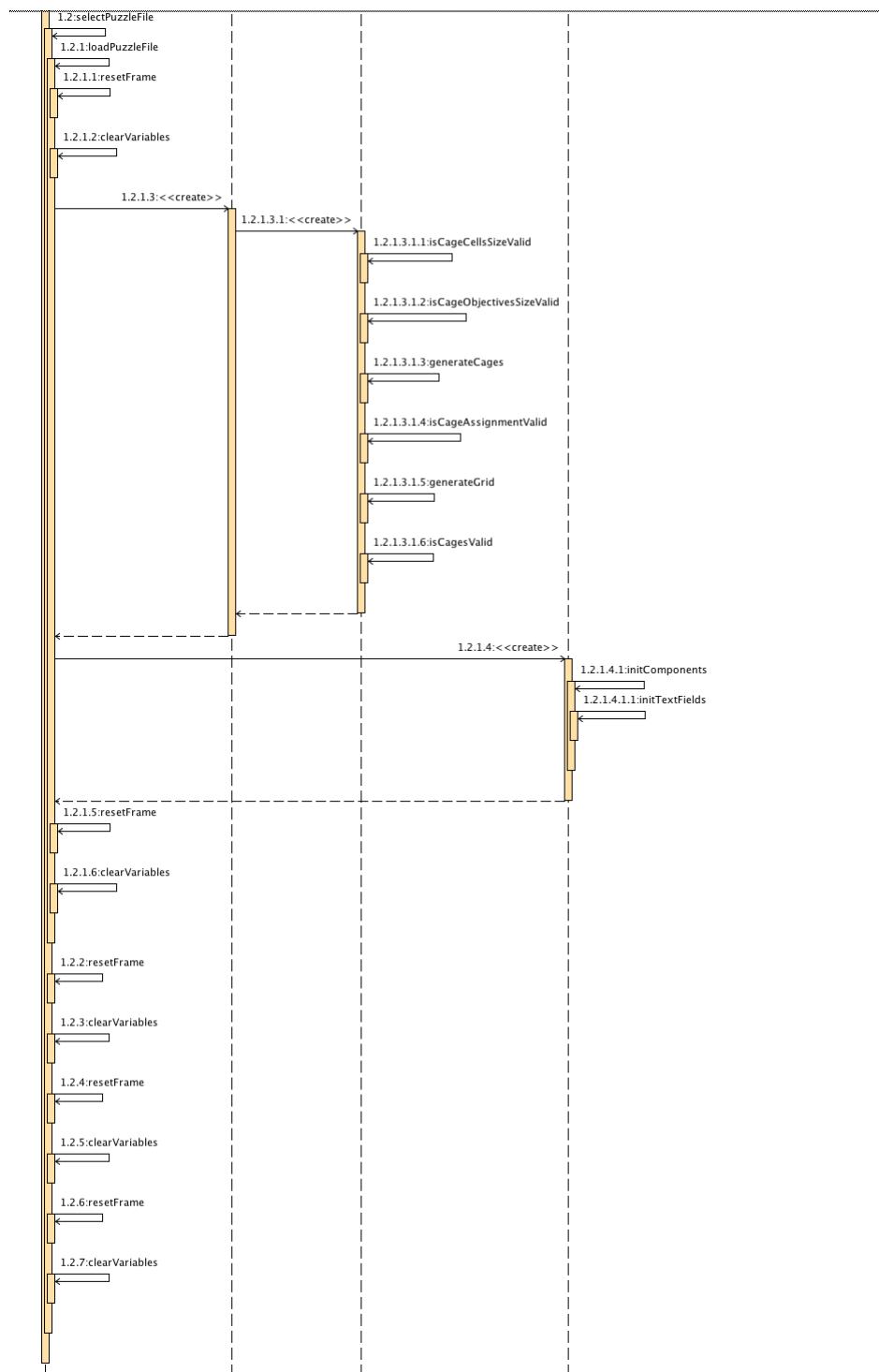
Gambar 3.59 menunjukkan diagram *sequence* saat menu item "Check Puzzle File" dalam menu "File" dipilih.

Gambar 3.60 menunjukkan diagram *sequence* saat menu item "Backtracking" dalam menu "Solve" dipilih.

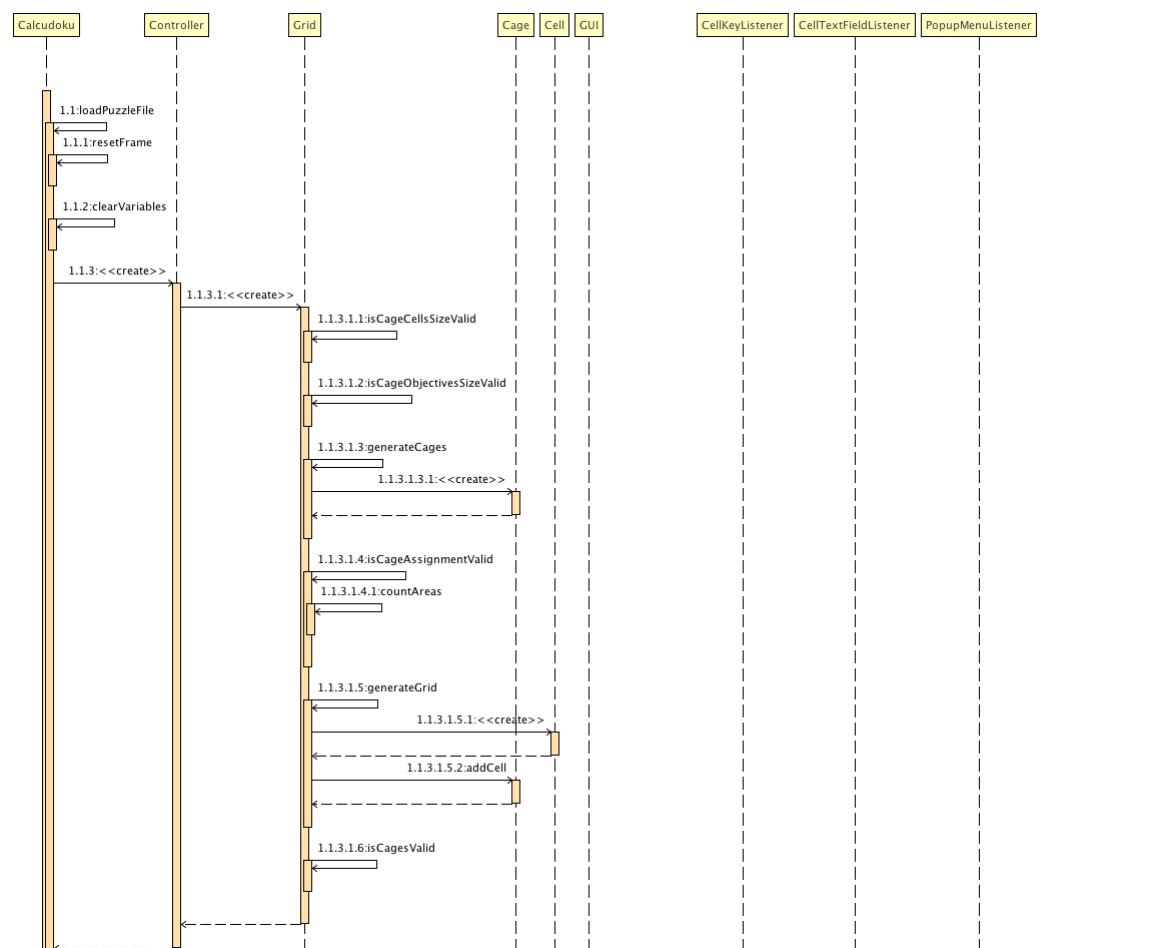
Gambar 3.61 dan Gambar 3.62 menunjukkan diagram *sequence* saat menu item "Hybrid Genetic" dalam menu "Solve" dipilih.



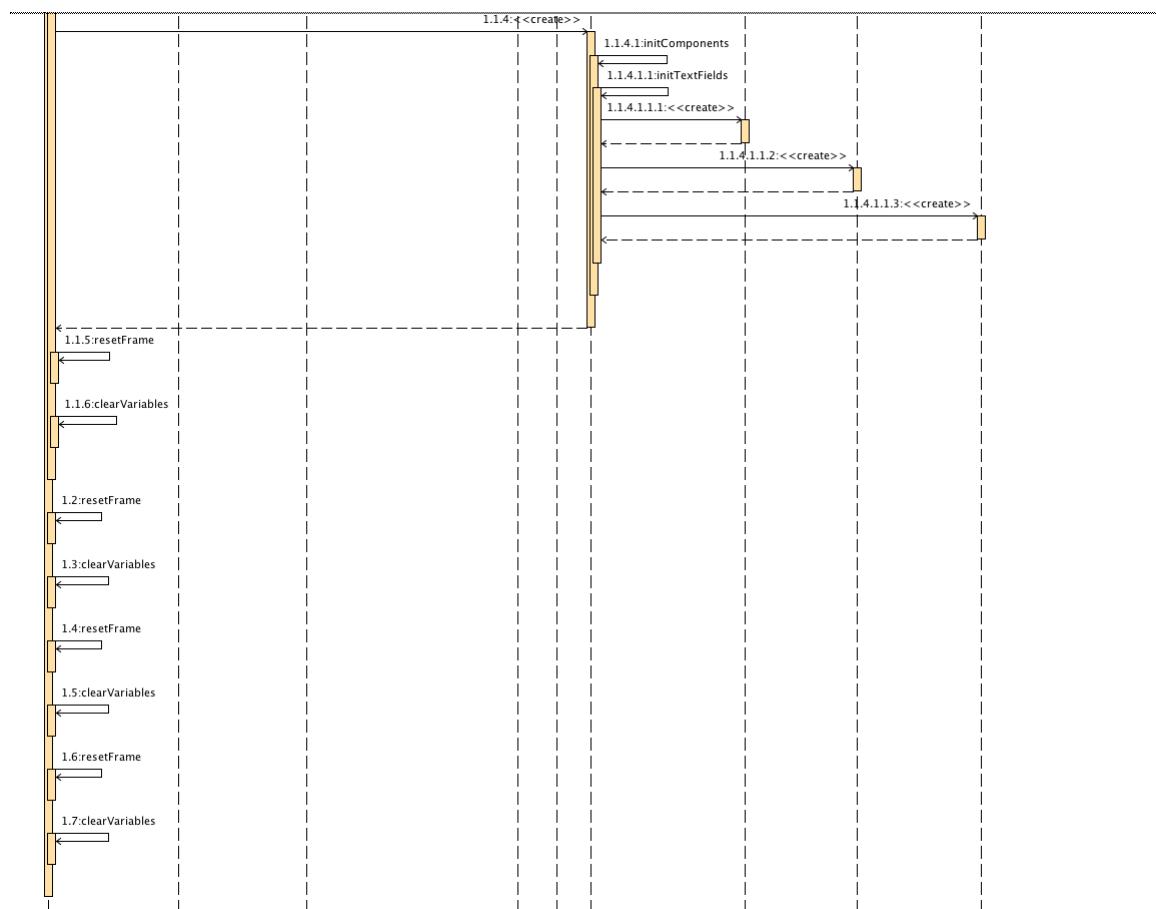
Gambar 3.52: Diagram *sequence* saat menu item "Load Puzzle File" dalam menu "File" dipilih (1)



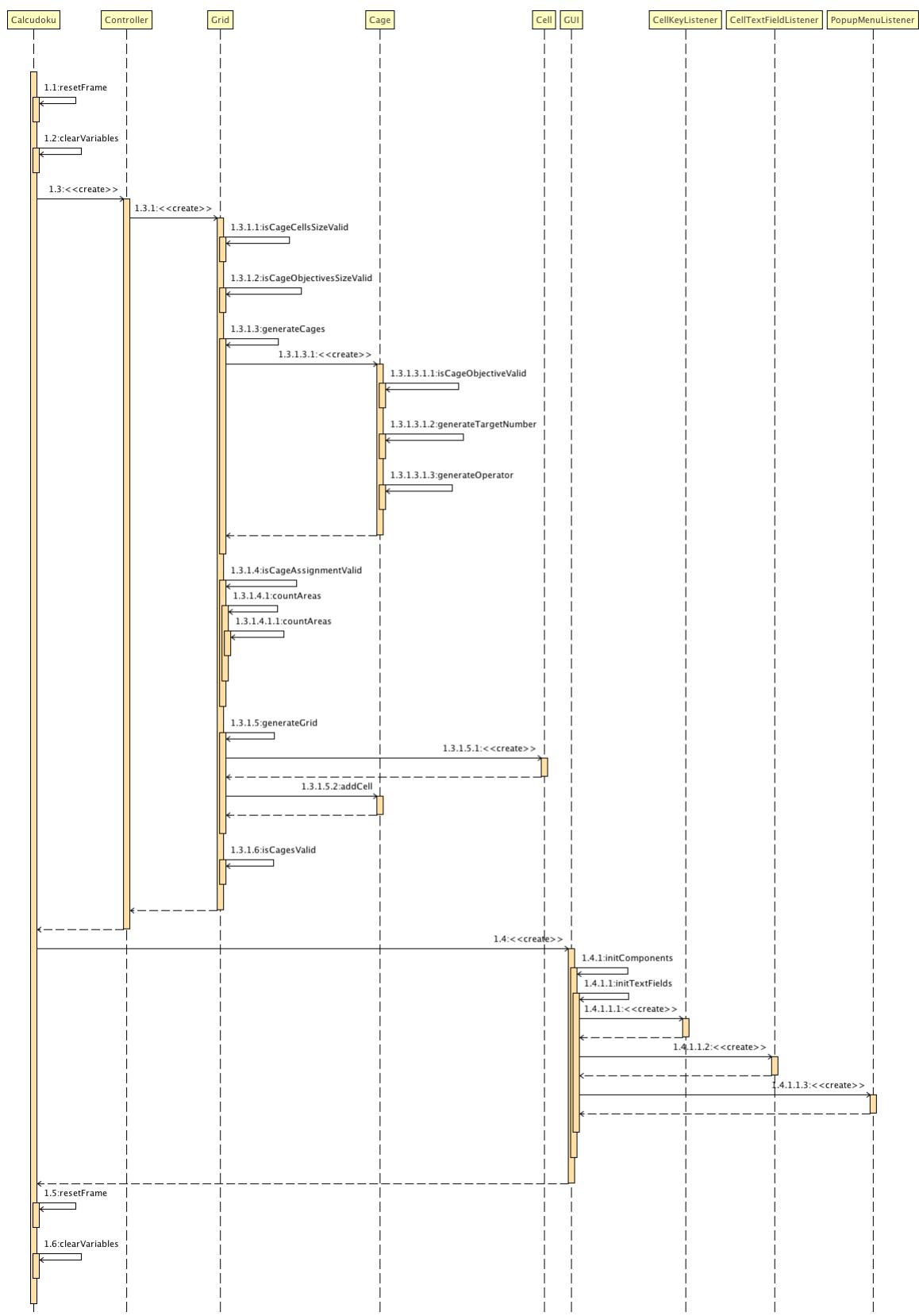
Gambar 3.53: Diagram *sequence* saat menu item "Load Puzzle File" dalam menu "File" dipilih (2)

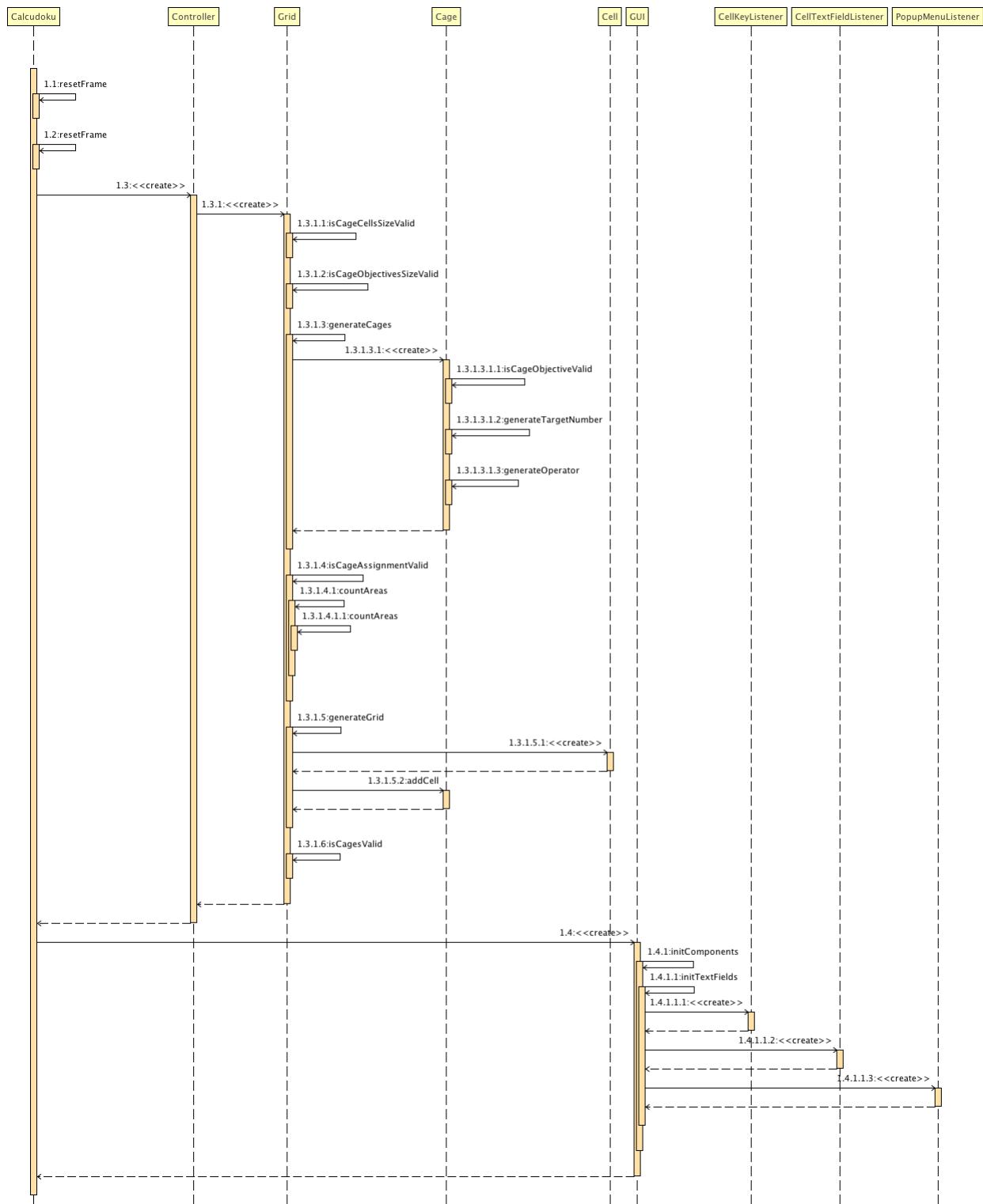


Gambar 3.54: Diagram *sequence* saat file yang ingin dibuka dalam *file chooser* dipilih (1)

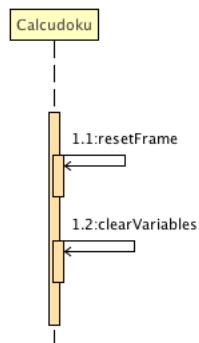


Gambar 3.55: Diagram *sequence* saat *file* yang ingin dibuka dalam *file chooser* dipilih (2)

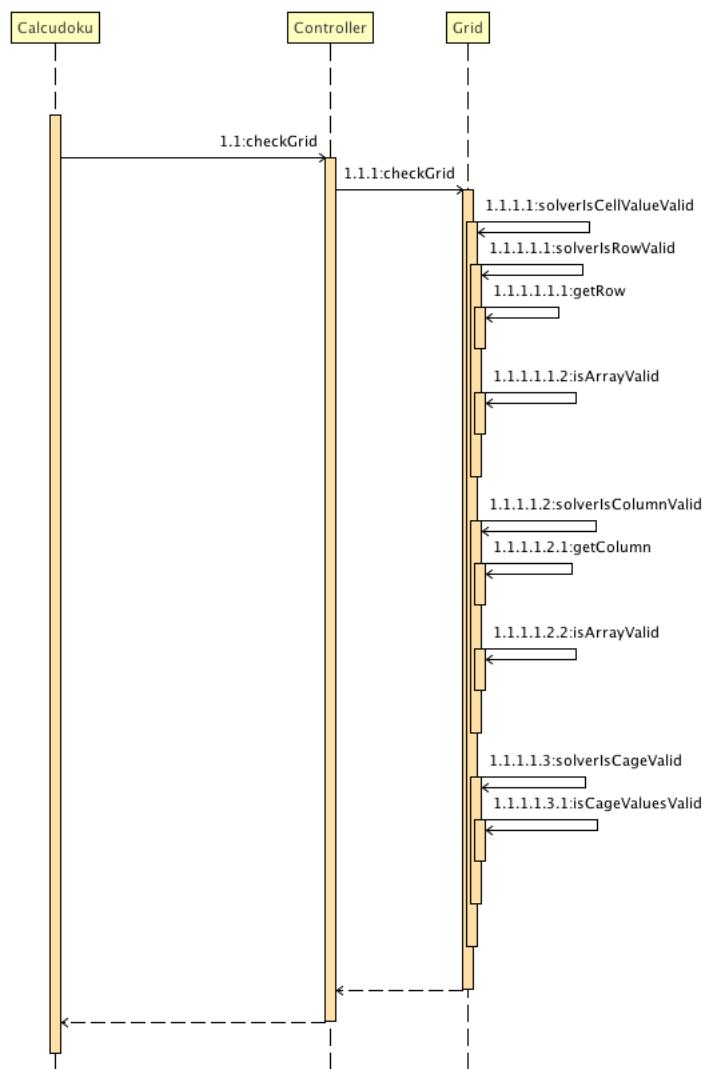
Gambar 3.56: Diagram *sequence* saat file permainan yang sudah dipilih dibuka



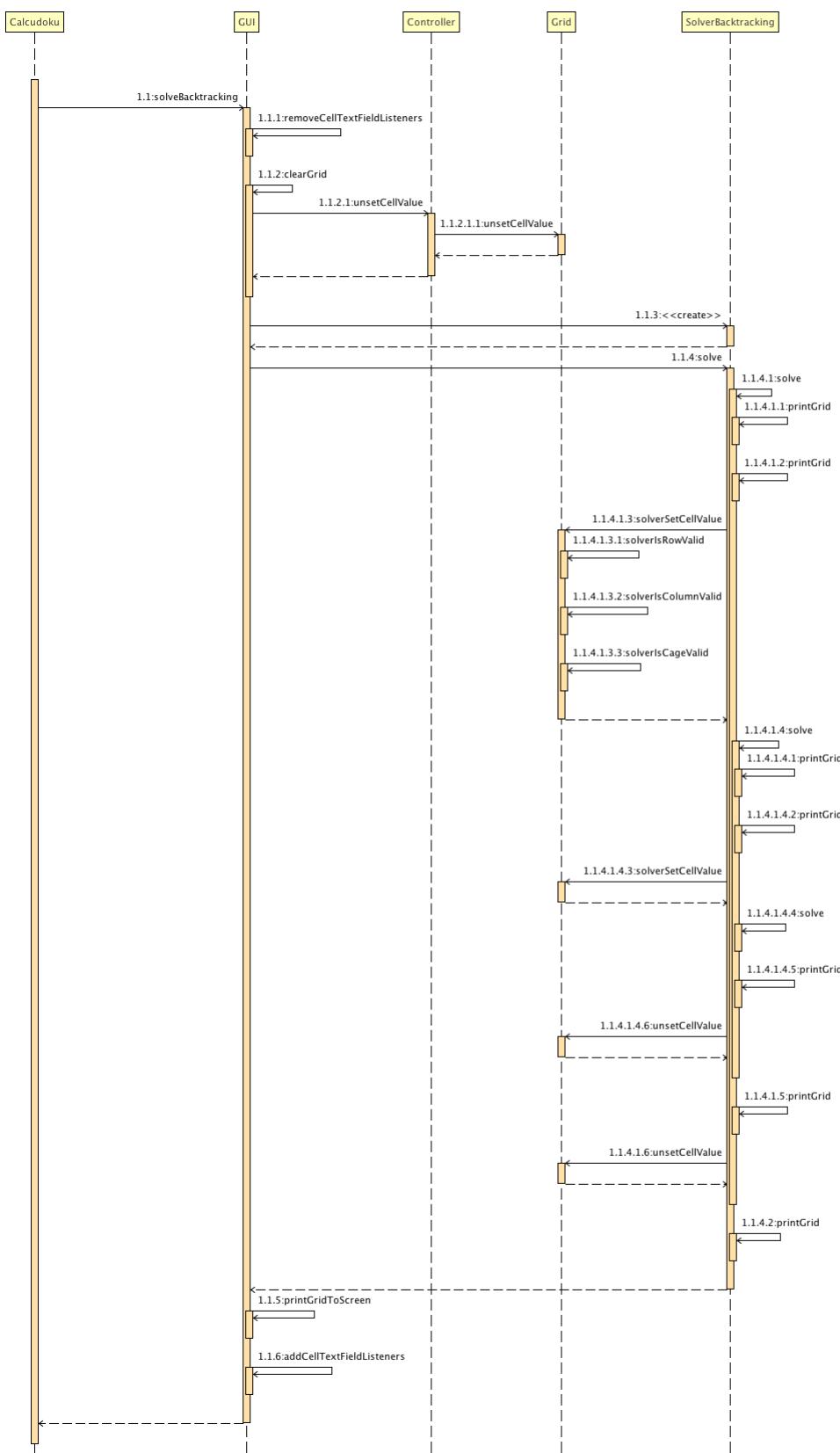
Gambar 3.57: Diagram *sequence* saat menu item "Reset Puzzle" dalam menu "File" dipilih



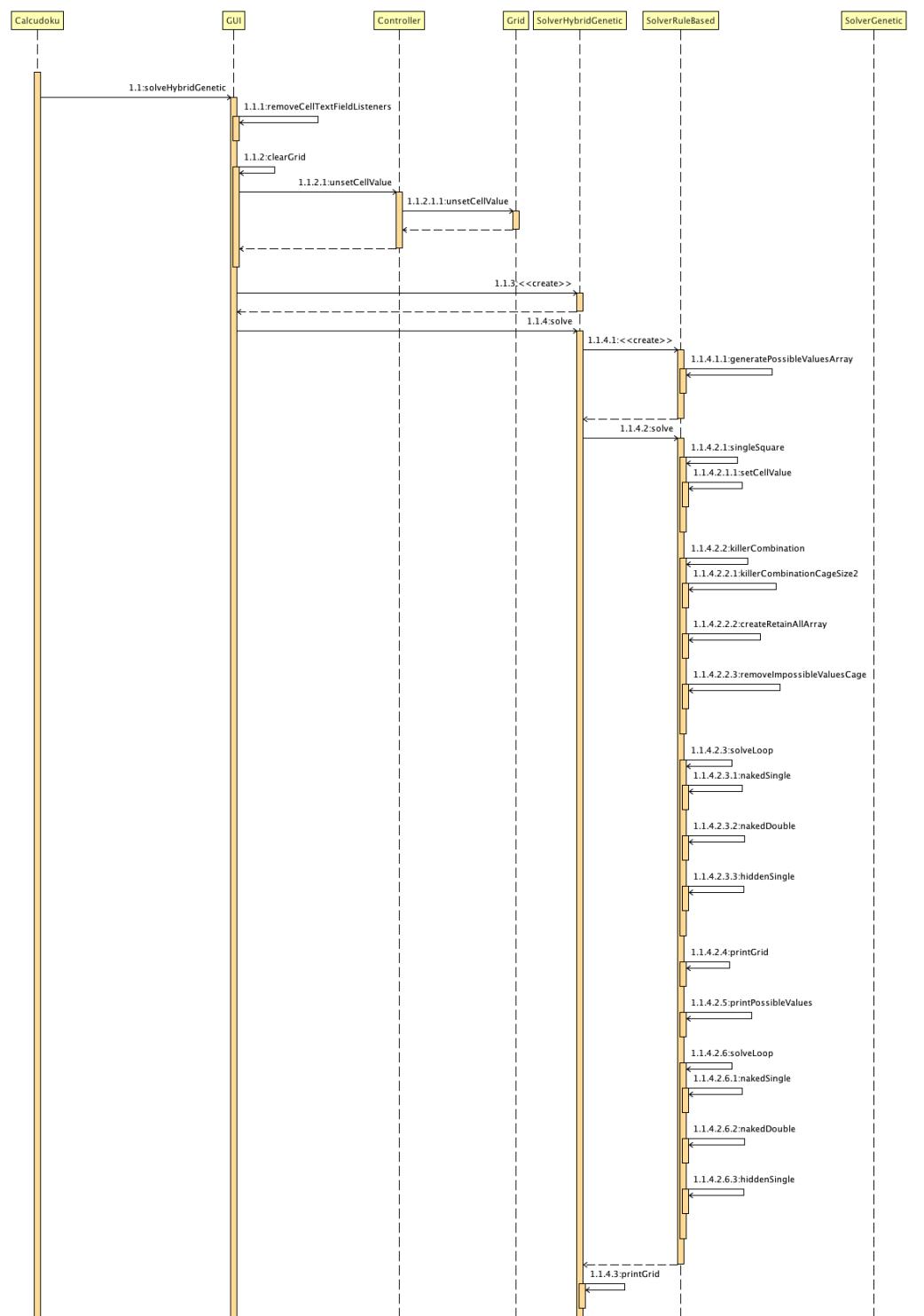
Gambar 3.58: Diagram *sequence* saat menu item "Close Puzzle File" dalam menu "File" dipilih



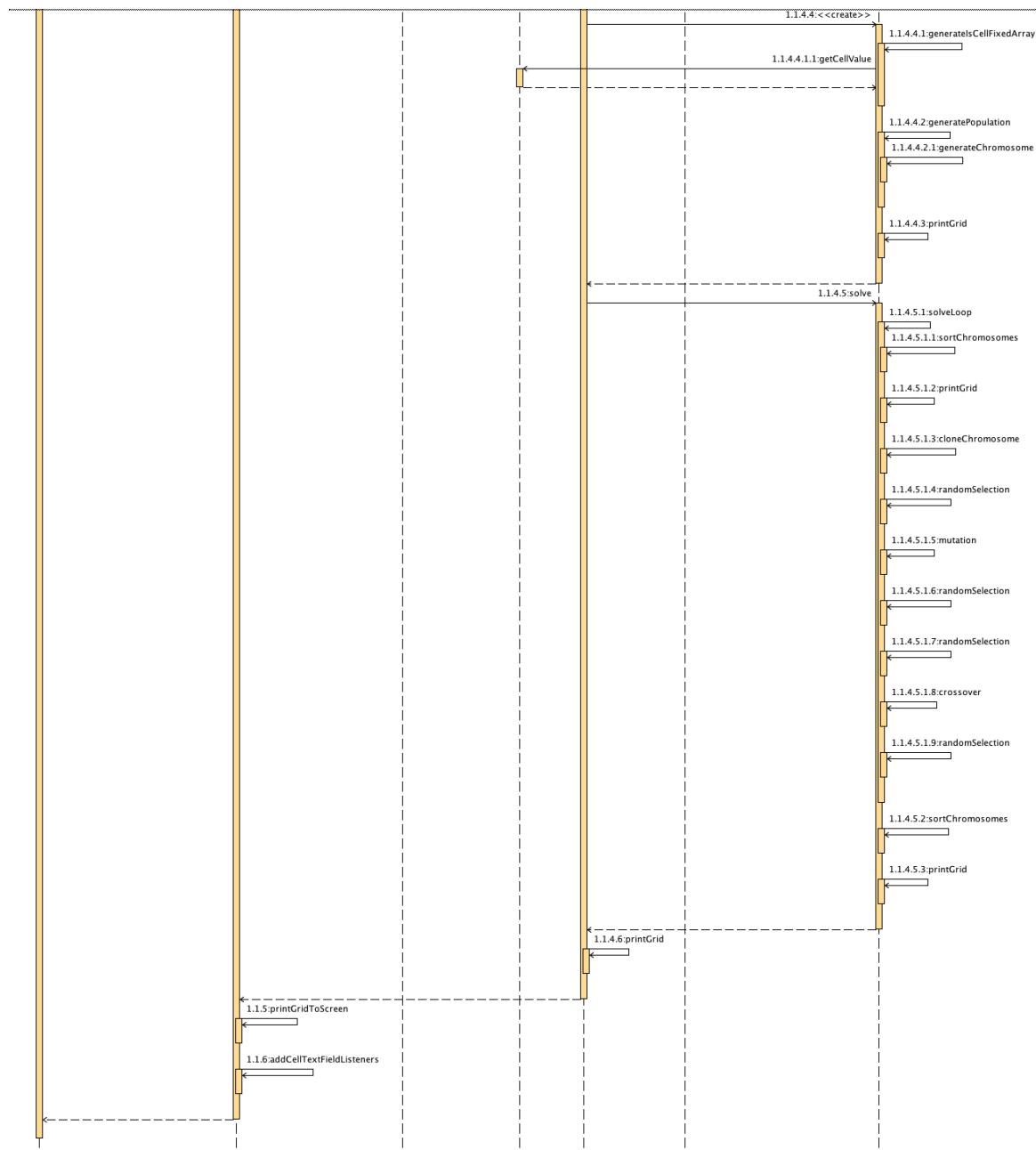
Gambar 3.59: Diagram *sequence* saat menu item "Check Puzzle File" dalam menu "File" dipilih



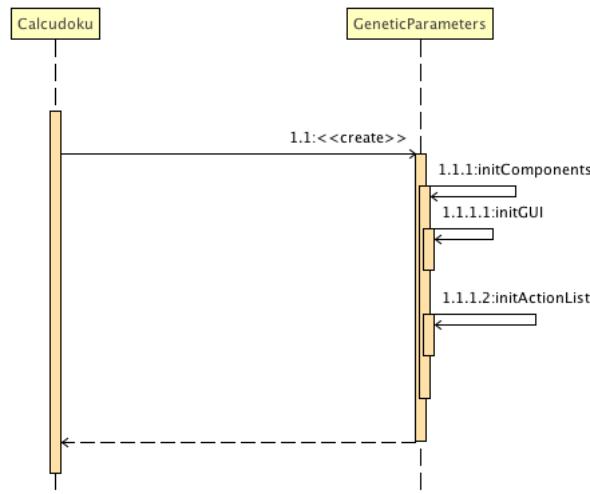
Gambar 3.60: Diagram *sequence* saat menu item "Backtracking" dalam menu "Solve" dipilih



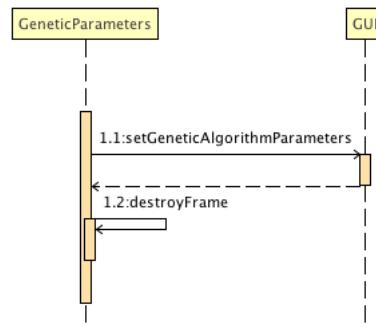
Gambar 3.61: Diagram *sequence* saat menu item "Hybrid Genetic" dalam menu "Solve" dipilih (1)



Gambar 3.62: Diagram *sequence* saat menu item "*Hybrid Genetic*" dalam menu "*Solve*" dipilih (2)



Gambar 3.63: Diagram *sequence* saat menu item "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih



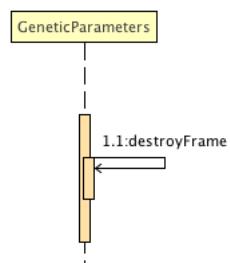
Gambar 3.64: Diagram *sequence* saat button "OK" dalam form "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih

Gambar 3.63 menunjukkan diagram *sequence* saat menu item "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih.

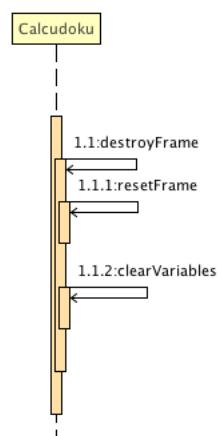
Gambar 3.64 menunjukkan diagram *sequence* saat button "OK" dalam form "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih.

Gambar 3.65 menunjukkan diagram *sequence* saat button "Cancel" dalam form "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih.

Gambar 3.66 menunjukkan diagram *sequence* saat perangkat lunak ditutup.



Gambar 3.65: Diagram *sequence* saat button "Cancel" dalam form "Set Genetic Algorithm Parameters" dalam menu "Solve" dipilih



Gambar 3.66: Diagram *sequence* saat perangkat lunak ditutup

BAB 4

PERANCANGAN

Bab ini membahas tentang perancangan perangkat lunak yang dibuat. Bab ini juga akan membahas tentang perancangan masukan, perancangan keluaran, diagram kelas, diagram *use case*, diagram aktivitas, dan diagram *sequence* untuk perangkat lunak tersebut.

4.1 Perancangan Masukan

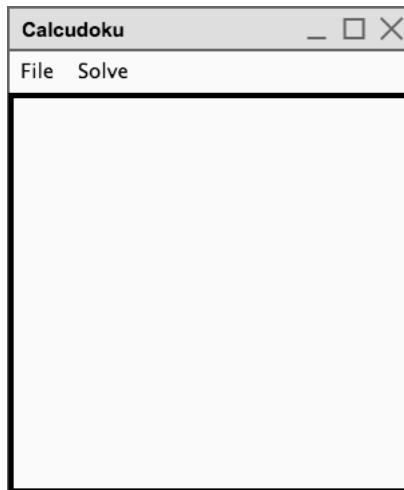
Masukan untuk perangkat lunak permainan teka-teki Calcudoku ini berupa sebuah *file* teks (.txt), seperti yang ditunjukkan pada Gambar 4.1.

Adapun rincian dari *file* teks masukan tersebut adalah sebagai berikut:

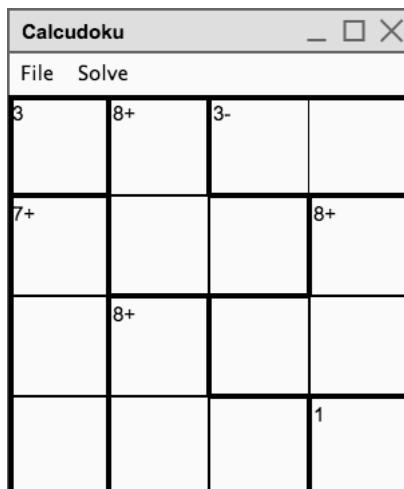
1. Baris pertama berisi ukuran *grid* dan banyaknya *cage* dari teka-teki Calcudoku tersebut. Angka pertama adalah ukuran *grid*, dan angka kedua adalah banyaknya *cage*.
2. Baris kedua sampai ke baris ke- $2 + (n - 1)$, dengan n adalah ukuran *grid*, berisi matriks *cage assignment*. Matriks ini merepresentasikan posisi dari setiap *cage* dalam *grid*. Setiap *cage* direpresentasikan dengan angka yang berbeda. Setiap *cage* dapat mempunyai ukuran (jumlah sel yang terdapat dalam *cage*) yang bervariasi. Setiap sel dalam sebuah *cage* harus berhubungan secara horizontal atau vertikal dengan sel lain dalam *cage* yang sama.
3. Baris ke- $2+n$ dan seterusnya berisi *cage objectives* untuk setiap *cage*. *Cage objectives* berisikan angka tujuan dan operasi matematika yang telah ditentukan. Angka-angka dalam sebuah *cage* harus mencapai angka tujuan jika dihitung menggunakan operasi matematika yang telah ditentukan.

```
4 9
1 2 3 3
1 4 4 5
6 7 7 5
8 8 9 9
7+
2=
2-
3-
2/
1=
6*
3+
7+
```

Gambar 4.1: Contoh *file* masukan.



Gambar 4.2: Perancangan GUI sebelum membuka *file* permainan



Gambar 4.3: Perancangan GUI sesudah membuka *file* permainan

4.2 Perancangan Keluaran

Keluaran untuk perangkat lunak permainan teka-teki Calcudoku ini berupa sebuah matriks yang berisi solusi dari teka-teki Calcudoku yang sudah diselesaikan oleh perangkat lunak ini. Matriks ini langsung ditampilkan ke layar.

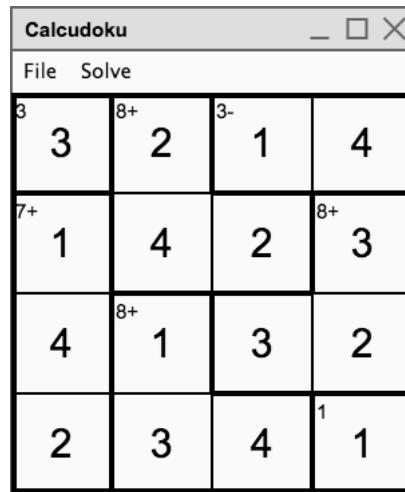
4.3 Perancangan Antarmuka

Antarmuka untuk perangkat lunak ini terdiri dari sebuah *frame* yang berisi sebuah menu *bar* dan GUI dari permainan teka-teki Calcudoku. GUI hanya akan ditampilkan jika perangkat lunak sudah membuka *file* permainan. Jika *file* permainan ditutup, maka GUI juga akan ditutup.

Gambar 4.2 menunjukkan perancangan GUI sebelum *file* permainan dibuka. Gambar 4.3 menunjukkan perancangan GUI sesudah *file* permainan dibuka. Gambar 4.4 menunjukkan perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

Menu *bar* untuk perangkat lunak ini terdiri dari dua menu, yaitu:

1. *File*, yaitu menu yang berisi *item-item* menu yang terkait dengan *file* permainan.
2. *Solve*, yaitu menu yang berisi *item-item* menu yang terkait dengan *solver*.



Gambar 4.4: Perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.



Gambar 4.5: Menu *File*

Menu *File* mempunyai beberapa menu *item*, yaitu:

1. *Load Puzzle File*, yaitu menu *item* untuk membuka *file* permainan.
2. *Reset Puzzle*, yaitu menu *item* untuk me-reset permainan.
3. *Close Puzzle File*, yaitu menu *item* untuk menutup *file* permainan.
4. *Check Puzzle*, yaitu menu *item* untuk memeriksa permainan jika ada masukan yang salah di dalam *grid*.
5. *Exit*, yaitu menu *item* untuk menutup perangkat lunak.

Gambar 4.5 menunjukkan isi dari menu *File*.

Menu *Solve* mempunyai beberapa menu *item*, yaitu:

1. *Backtracking*, yaitu menu *item* untuk menyelesaikan permainan berdasarkan *file* permainan yang dibuka dengan algoritma *backtracking*.
2. *Hybrid Genetic*, yaitu menu *item* untuk menyelesaikan permainan berdasarkan *file* permainan yang dibuka dengan algoritma *hybrid genetic*.
3. *Set Genetic Algorithm Parameters*, yaitu menu *item* untuk mengatur nilai dari parameter-parameter untuk algoritma genetik.

Gambar 4.6 menunjukkan isi dari menu *Solve*.

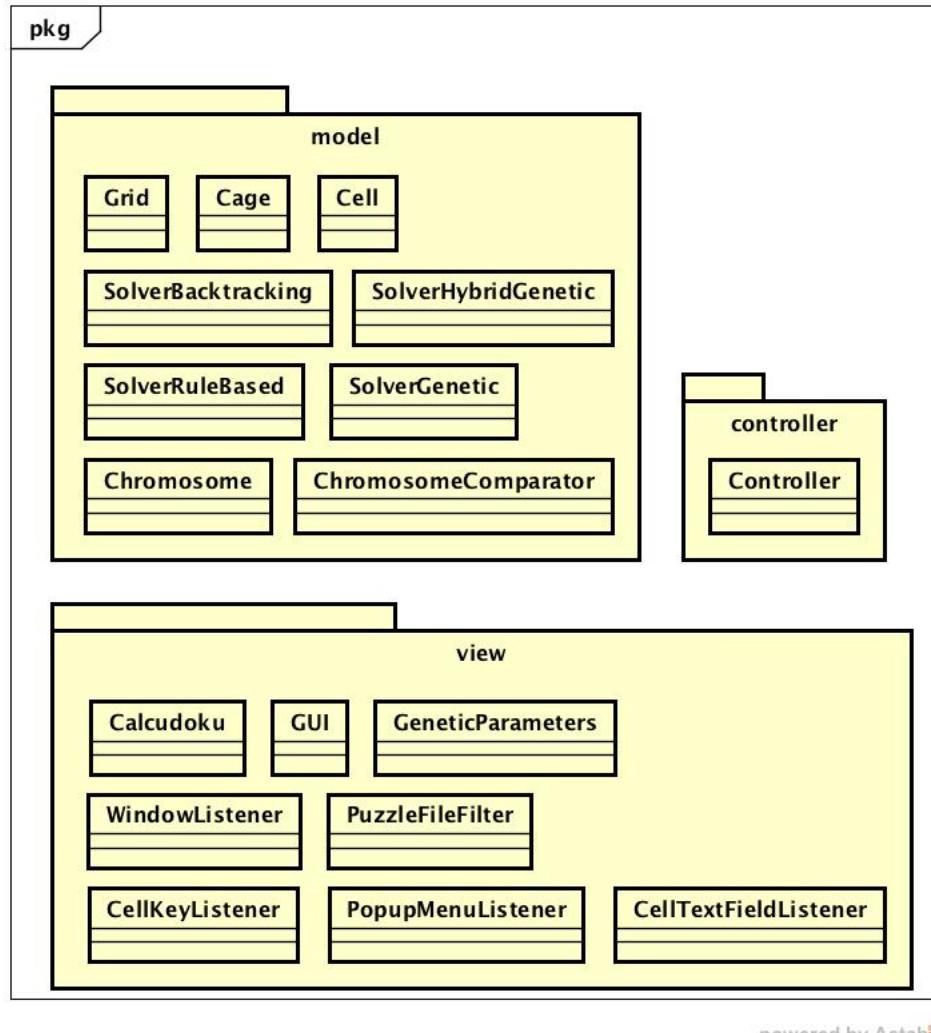


Gambar 4.6: Menu *Solve*

4.4 Diagram Kelas

Perangkat lunak teka-teki Calcudoku ini terdiri dari beberapa kelas, yang dikelompokkan dalam tiga package, yaitu:

1. Model, yaitu *engine* dari perangkat lunak ini. *Package* ini memiliki beberapa kelas, yaitu:
 - (a) Grid, yaitu kelas yang merepresentasikan *grid* dalam teka-teki Calcudoku.
 - (b) Cell, yaitu kelas yang merepresentasikan sel dalam teka-teki Calcudoku.
 - (c) Cage, yaitu kelas yang merepresentasikan *cage* dalam teka-teki Calcudoku.
 - (d) SolverBacktracking, yaitu kelas *solver* untuk teka-teki Calcudoku menggunakan algoritma backtracking.
 - (e) SolverHybridGenetic, yaitu kelas *solver* untuk teka-teki Calcudoku menggunakan algoritma *hybrid genetic*. Algoritma ini akan mencoba menyelesaikan teka-teki Calcudoku menggunakan algoritma *rule based* terlebih dahulu. Algoritma genetik baru akan dijalankan jika algoritma *rule based* gagal dalam menyelesaikan teka-teki Calcudoku.
 - (f) SolverRuleBased, yaitu kelas *solver* untuk teka-teki Calcudoku menggunakan algoritma *rule based*. Dalam algoritma *hybrid genetic*, algoritma akan mencoba menyelesaikan teka-teki Calcudoku menggunakan algoritma *rule based* terlebih dahulu.
 - (g) SolverGenetic, yaitu kelas *solver* untuk teka-teki Calcudoku menggunakan algoritma genetik. Dalam algoritma *hybrid genetic*, algoritma genetik baru akan dijalankan jika algoritma *rule based* gagal dalam menyelesaikan teka-teki Calcudoku.
 - (h) Chromosome, yaitu kelas yang merepresentasikan sebuah kromosom untuk algoritma genetik dalam solver *hybrid genetic*.
 - (i) ChromosomeComparator, yaitu kelas pembanding *custom (custom comparator)* yang berfungsi untuk mengurutkan kromosom berdasarkan nilai kelayakkannya (*fitness value*).
2. View, yaitu GUI dari perangkat lunak ini. *Package* ini memiliki beberapa kelas, yaitu:
 - (a) Calcudoku, yaitu kelas *frame* yang berisi menu *bar* dan instansiasi kelas panel GUI.
 - (b) WindowListener, yaitu kelas *listener* untuk kelas Calcudoku. *Listener* ini berfungsi untuk menambahkan pesan peringatan saat akan menutup perangkat lunak.
 - (c) PuzzleFileFilter, yaitu kelas filter untuk *file chooser*. Filter ini membatasi agar *file chooser* hanya bisa membuka file teks.
 - (d) GUI, yaitu kelas panel yang merepresentasikan GUI dari permainan teka-teki Calcudoku.
 - (e) CellKeyListener, yaitu kelas *listener* untuk kelas GUI. *Listener* ini berfungsi untuk menggerakkan kursor dari sebuah sel ke sel di sebelahnya menggunakan tombol-tombol panah ke kiri, ke atas, ke bawah, dan ke kanan. Listener ini juga berfungsi untuk membatasi agar sel hanya bisa diisi oleh satu angka.
 - (f) PopupMenuListener, yaitu kelas *listener* untuk kelas GUI. *Listener* ini berfungsi untuk mengisi sel dengan angka menggunakan menu *pop up*, sel akan diisi dengan angka yang dipilih.



Gambar 4.7: Diagram kelas untuk perangkat lunak Calcudoku.

- (g) CellTextFieldListener, yaitu kelas *listener* untuk kelas GUI. *Listener* ini berfungsi untuk mengisikan sel dalam kelas Grid dengan angka yang diisikan ke dalam sel dalam GUI.
 - (h) GeneticParameters, yaitu kelas yang berisi *form* untuk mengatur nilai dari parameter-parameter untuk algoritma genetik.
3. Controller, yaitu penghubung antara *package* model dan *package* view. *Package* ini hanya berisi satu kelas, yaitu kelas Controller.

Diagram kelas untuk perangkat lunak ini dapat dilihat pada Gambar 4.7.

Berikut ini adalah rincian dari setiap kelas, dengan setiap atribut dan setiap *method* yang dimilikinya.

4.4.1 Kelas Grid

Kelas Grid mempunyai beberapa atribut, yaitu:

1. size, yaitu ukuran dari matriks *grid*.
2. numberOfCages, yaitu banyaknya *cage* yang terdapat dalam *grid*.

3. cageCells, yaitu sebuah matriks *cage assignment*. Matriks ini merepresentasikan posisi dari setiap *cage* dalam *grid*.
4. cageObjectives, yaitu sebuah *array* yang berisi *cage objectives* untuk setiap *cage*. *Cage objectives* berisikan angka tujuan dan operasi matematika yang telah ditentukan.
5. grid, yaitu representasi dari *grid* dalam teka-teki Calcudoku. *grid* adalah sebuah matriks yang berisi sel-sel. Matriks ini berukuran $n \times n$.
6. cages, yaitu representasi dari sebuah *cage* dalam sebuah *grid*.

Kelas Grid mempunyai beberapa *method*, yaitu:

1. Grid(Integer size, Integer numberOfCages, Integer[][] cageCells, String[] cageObjectives), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa ukuran dari matriks *grid*, banyaknya *cage* yang terdapat dalam *grid*, matriks *cage assignment*, dan array *cage objectives*.
2. countAreas(Integer[][] array), yaitu *method* pembungkus dari *method* countAreas(int[][] array, boolean[][] checked). *Method* ini menerima masukan berupa array *cage assignment* untuk sebuah *cage*, dan menghasilkan keluaran berupa jumlah area dari *cage* tersebut.
3. countAreas(Integer[][] array, boolean[][] checked), yaitu *method* yang menghitung jumlah area dari sebuah *cage* secara rekursif dengan menggunakan algoritma *flood fill*. *Method* ini menerima masukan berupa array *cage assignment* untuk sebuah *cage* dan sebuah array *checked* yang berfungsi untuk menandai sel-sel yang sudah pernah dikunjungi atau belum, dan menghasilkan keluaran berupa jumlah area dari *cage* tersebut.
4. floodFill(int i, int j, Integer[][] array, boolean[][] checked), yaitu implementasi dari algoritma *flood fill* untuk menghitung jumlah area dari sebuah *cage*. *Method* ini menerima masukan berupa posisi baris dan kolom dari sebuah sel, array *cage assignment* untuk sebuah *cage* dan sebuah array *checked* yang berfungsi untuk menandai sel-sel yang sudah pernah dikunjungi atau belum.
5. isCageCellsSizeValid(Integer[][] cageCells), yaitu *method* yang memeriksa apakah ukuran matriks *cage assignment* valid atau tidak. *Method* ini menerima masukan berupa matriks *cage assignment*, dan menghasilkan keluaran apakah matriks tersebut *valid* atau tidak. Matriks tersebut *valid* jika ukuran barisnya dan kolomnya sama dengan variabel *size*.
6. isCageObjectivesSizeValid(String[] cageObjectives), yaitu *method* yang memeriksa apakah ukuran matriks *cage objectives* valid atau tidak. *Method* ini menerima masukan berupa array *cage objectives*, dan menghasilkan keluaran apakah array tersebut *valid* atau tidak. Array tersebut *valid* jika ukuran dari array tersebut sama dengan variabel *numberOfCages*.
7. isCageAssignmentValid(Integer[][] array), yaitu *method* yang memeriksa apakah *cage assignment* untuk sebuah *cage* *valid* atau tidak. *Method* ini menerima masukan berupa matriks *cage assignment* untuk sebuah *cage* dan menghasilkan keluaran apakah matriks tersebut *valid* atau tidak. Matriks tersebut *valid* jika jumlah area dari *cage* tersebut adalah satu.
8. isCagesValid(Cage[] cages), yaitu *method* yang memeriksa apakah setiap *cage* yang ada di dalam *grid* *valid* atau tidak. *Method* ini menerima masukan berupa array *cage*, dan menghasilkan keluaran apakah array tersebut *valid* atau tidak. Array tersebut *valid* jika setiap *cage* dengan operator = hanya berukuran satu sel, setiap *cage* dengan operator + atau \times berukuran minimal dua sel, dan setiap *cage* dengan operator - atau \div berukuran tepat dua sel.

9. generateCages(Cage[] cages), yaitu *method* yang membangkitkan *cage-cage* dalam sebuah *grid*. *Method* ini menerima masukan berupa sebuah array *Cage* yang kosong.
10. generateGrid(Cell[][] grid, Cage[] cages), yaitu *method* yang membangkitkan *grid* dan *cage assignment* dari *grid* tersebut.. *Method* ini menerima masukan berupa sebuah matriks sel yang kosong dan sebuah array *cage* yang kosong.
11. getRow(int rowNumber), yaitu *method* untuk mendapatkan isi dari sebuah baris yang diminta. *Method* ini menerima masukan berupa nomor baris yang diminta dan menghasilkan keluaran berupa isi baris yang diminta.
12. getColumn(int columnNumber), yaitu *method* untuk mendapatkan isi dari sebuah kolom yang diminta. *Method* ini menerima masukan berupa nomor kolom yang diminta dan menghasilkan keluaran berupa isi kolom yang diminta dalam bentuk *ArrayList*.
13. getCageValues(int cageNumber), yaitu *method* untuk mendapatkan isi dari sebuah *cage* yang diminta. *Method* ini menerima masukan berupa nomor *cage* yang diminta dan menghasilkan keluaran berupa isi *cage* yang diminta dalam bentuk *ArrayList*.
14. isArrayValid(ArrayList<Integer> array), yaitu *method* untuk memeriksa apakah sebuah *array valid* atau tidak. *Method* ini menerima masukan berupa *array* yang akan diperiksa dan menghasilkan keluaran apakah *array* tersebut *valid* atau tidak. *Array* tersebut *valid* jika tidak ada angka yang berulang dalam *array* tersebut.
15. isRowValid(int row), yaitu *method* untuk memeriksa apakah sebuah baris *valid* atau tidak. *Method* ini menerima masukan berupa nomor baris yang diminta dan menghasilkan keluaran apakah baris yang diminta tersebut *valid* atau tidak. Baris tersebut *valid* jika tidak ada angka yang berulang dalam baris tersebut.
16. solverIsRowValid(int column), yaitu *method* yang sama dengan *isRowValid*, tetapi *method* ini hanya untuk dipanggil oleh *solver*.
17. isColumnValid(int column), yaitu *method* untuk memeriksa apakah sebuah kolom *valid* atau tidak. *Method* ini menerima masukan berupa nomor kolom yang diminta dan menghasilkan keluaran apakah kolom yang diminta tersebut *valid* atau tidak. Kolom tersebut *valid* jika tidak ada angka yang berulang dalam kolom tersebut.
18. solverIsColumnValid(int column), yaitu *method* yang sama dengan *isColumnValid*, tetapi *method* ini hanya untuk dipanggil oleh *solver*.
19. isCageValuesValid(int cageNumber), yaitu *method* untuk memeriksa apakah nilai dari setiap sel yang berada dalam sebuah *cage valid* atau tidak. *Method* ini menerima masukan berupa nomor *cage* yang diminta dan menghasilkan keluaran apakah nilai dari setiap sel yang berada dalam *cage* yang diminta tersebut *valid* atau tidak. *Cage* tersebut *valid* jika nilai dari setiap sel yang berada di dalam *cage* tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan.
20. isCageValid(int row, int column), yaitu *method* untuk memeriksa apakah sebuah *cage valid* atau tidak. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sebuah sel yang diminta dan menghasilkan keluaran apakah *cage* yang berisi sel tersebut *valid* atau tidak. *Cage* tersebut *valid* jika angka-angka dalam *cage* tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan.
21. solverIsCageValid(int column), yaitu *method* yang sama dengan *isCageValid*, tetapi *method* ini hanya untuk dipanggil oleh *solver*.

22. `isCellValueValid(int row, int column)`, yaitu *method* untuk memeriksa apakah nilai dari sel tersebut *valid* atau tidak. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan diperiksa dan menghasilkan keluaran apakah nilai dari sel tersebut *valid* atau tidak. Nilai dari sebuah sel *valid* jika nilai dari sel tersebut tidak berulang dalam baris dan kolom tempat sel tersebut berada, dan angka-angka dari *cage* yang berisi sel tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan.
23. `solverIsCellValueValid(int column)`, yaitu *method* yang sama dengan `isCellValueValid`, tetapi *method* ini hanya untuk dipanggil oleh solver.
24. `setCellValue(int row, int column, Integer value)`, yaitu *method* untuk mengisi sebuah sel dengan nilai yang telah ditentukan. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan diisi dan nilai dari sel tersebut, dan menghasilkan keluaran apakah nilai dari sel tersebut *valid* atau tidak. Nilai dari sebuah sel *valid* jika nilai dari sel tersebut tidak berulang dalam baris dan kolom tempat sel tersebut berada, dan angka-angka dari *cage* yang berisi sel tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan.
25. `solverSetValue(int row, int column, Integer value)`, yaitu *method* yang sama dengan `setCellValue`, tetapi *method* ini hanya untuk dipanggil oleh solver.
26. `unsetCellValue(int row, int column)`, yaitu *method* untuk menghapus isi dari sebuah sel. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan dihapus isinya.
27. `isWin()`, yaitu *method* yang memeriksa apakah semua sel sudah diisi dengan nilai yang *valid* atau tidak. *Method* ini menghasilkan keluaran apakah semua sel sudah diisi dengan nilai yang *valid* atau tidak. *Method* ini menghasilkan *null* jika ada sel yang belum diisi.
28. `checkGrid()`, yaitu *method* yang memeriksa apakah ada sel yang berisi nilai yang tidak *valid* atau tidak. *Method* ini akan menghasilkan apakah ada sel yang berisi nilai yang tidak *valid* atau tidak. Nilai dari sebuah sel *valid* jika nilai dari sel tersebut tidak berulang dalam baris dan kolom tempat sel tersebut berada, dan angka-angka dari *cage* yang berisi sel tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan. *Method* ini juga memeriksa apakah ada sel yang kosong atau tidak.
29. `isFilled()`, yaitu *method* yang memeriksa apakah semua sel sudah diisi atau tidak. *Method* ini menghasilkan keluaran apakah semua sel sudah diisi atau tidak.
30. `getCellValue(int row, int column)`, yaitu *method* untuk mendapatkan isi dari sebuah sel. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang diminta dan menghasilkan keluaran berupa isi dari sel yang diminta tersebut.
31. `getSize()`, yaitu *method* untuk mendapatkan ukuran dari *grid*. *Method* ini menghasilkan keluaran berupa ukuran dari *grid*.
32. `getNumberOfCages()`, yaitu *method* untuk mendapatkan jumlah *cage* yang ada di dalam *grid*. *Method* ini menghasilkan keluaran berupa jumlah *cage* yang ada di dalam *grid*.
33. `getCageCells()`, yaitu *method* untuk mendapatkan matriks *cage assignment* dari *grid*. *Method* ini menghasilkan keluaran berupa matriks *cage assignment* dari *grid*.
34. `getCageObjectives()`, yaitu *method* untuk mendapatkan *cage objectives* dari setiap *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa sebuah array yang berisi *cage objectives* dari setiap *cage* dalam *grid*.

35. `getGridContents()`, yaitu *method* untuk mendapatkan nilai dari setiap sel *grid*. *Method* ini menghasilkan keluaran berupa sebuah matriks yang berisi nilai dari setiap sel *grid*.
36. `getCages()`, yaitu *method* untuk mendapatkan semua *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa array yang berisi semua *cage* dalam *grid*.
37. `getGame()`, yaitu *method* untuk mendapatkan instansiasi saat ini dari kelas *Grid*. *Method* ini menghasilkan keluaran berupa instansiasi saat ini dari kelas *Grid*.

Diagram kelas Grid dapat dilihat pada Gambar 4.8.

4.4.2 Kelas Cage

Kelas Cage mempunyai beberapa atribut, yaitu:

1. `objective`, yaitu angka tujuan dan operator yang ditentukan untuk *cage* tersebut.
2. `targetNumber`, yaitu angka tujuan dari *cage* tersebut.
3. `operator`, yaitu operator yang ditentukan untuk *cage* tersebut.
4. `cells`, yaitu sebuah array yang berisi sel-sel yang merupakan anggota dari *cage* tersebut.

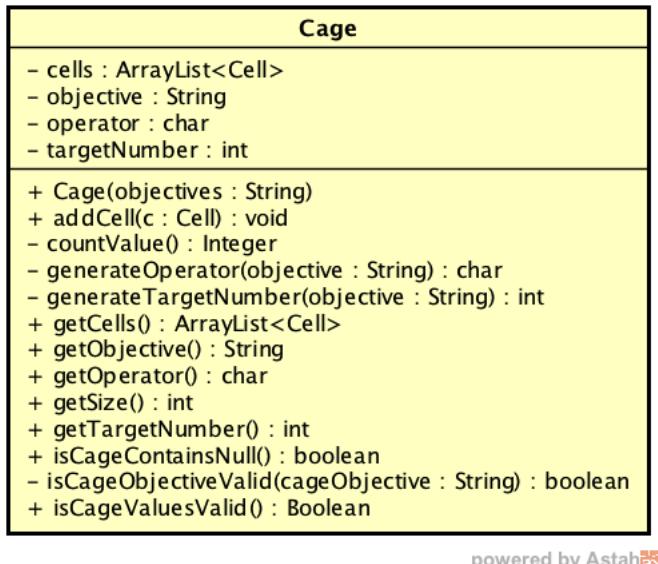
Kelas Cage mempunyai *method-method* berikut:

1. `Cage(String objectives)`, yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *cage objectives* dari *cage* tersebut.
2. `isCageObjectiveValid(String cageObjective)`, yaitu *method* yang memeriksa apakah *cage objective* dari *cage* tersebut *valid* atau tidak. *Method* ini menerima masukan berupa String yang berisi *cage objective* dan menghasilkan keluaran apakah String tersebut valid atau tidak. *Cage objective valid* jika isi dari *cage objective* tersebut adalah satu angka tujuan dari *cage* tersebut dan diikuti oleh satu operator yang telah ditentukan untuk *cage* tersebut.
3. `generateTargetNumber(String objective)`, yaitu *method* yang membangkitkan angka tujuan dari sebuah *cage* dari *cage objective* yang diberikan. *Method* ini menerima masukan berupa String yang berisi *cage objective* dari sebuah *cage* dan menghasilkan keluaran berupa angka tujuan dari *cage* tersebut.
4. `generateOperator(String objective)`, yaitu *method* yang membangkitkan operator yang telah ditentukan untuk sebuah *cage* dari *cage objective* yang diberikan. *Method* ini menerima masukan berupa String yang berisi *cage objective* dari sebuah *cage* dan menghasilkan keluaran berupa operator yang telah ditentukan untuk *cage* tersebut.
5. `addCell(Cell c)`, yaitu *method* untuk menambahkan sebuah sel kedalam sebuah *cage*. *Method* ini menerima masukan berupa sel yang akan dimasukkan ke dalam *cage*.
6. `isCageContainsNull()`, yaitu *method* yang memeriksa apakah sebuah *cage* mempunyai sel yang belum diisi. *Method* ini menghasilkan keluaran apakah *cage* tersebut mempunyai sel yang belum terisi.
7. `isCageValuesValid()`, yaitu *method* yang memeriksa apakah angka-angka dalam sebuah *cage* mencapai angka tujuan dari *cage* tersebut jika dihitung menggunakan operator yang telah ditentukan untuk *cage* tersebut. *Method* ini menghasilkan keluaran apakah angka-angka dalam *cage* tersebut mencapai angka tujuan dari *cage* tersebut jika dihitung menggunakan operator yang telah ditentukan untuk *cage* tersebut. *Method* ini menghasilkan *null* jika ada sel di dalam *cage* yang belum diisi.



powered by Astah

Gambar 4.8: Diagram kelas Grid.



powered by Astah

Gambar 4.9: Diagram kelas Cage.

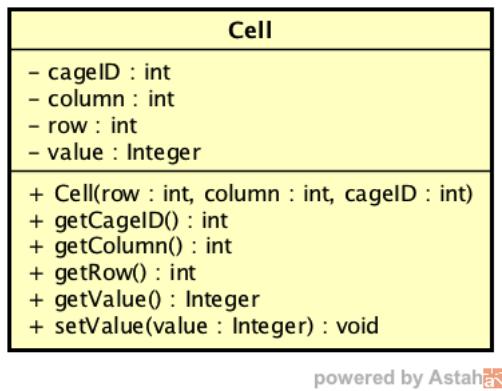
8. `countValue()`, yaitu *method* yang menghitung angka-angka di dalam sebuah *cage* menggunakan operator yang telah ditentukan untuk *cage* tersebut. *Method* ini menghasilkan keluaran hasil perhitungan dari angka-angka di dalam sebuah *cage* menggunakan operator yang telah ditentukan untuk *cage* tersebut. *Method* ini menghasilkan *null* jika ada sel di dalam *cage* yang belum diisi.
9. `getTargetNumber()`, yaitu *method* untuk mendapatkan angka tujuan dari sebuah *cage*. *Method* ini menghasilkan keluaran berupa angka tujuan dari *cage* tersebut.
10. `getOperator()`, yaitu *method* untuk mendapatkan operator yang telah ditentukan untuk sebuah *cage*. *Method* ini menghasilkan keluaran berupa operator yang telah ditentukan untuk *cage* tersebut.
11. `getCells()`, yaitu *method* untuk mendapatkan sel-sel anggota sebuah *cage*. *Method* ini menghasilkan keluaran sebuah `ArrayList` yang berisi sel-sel anggota *cage* tersebut.
12. `getSize()`, yaitu *method* untuk mendapatkan jumlah dari sel-sel anggota sebuah *cage*. *Method* ini menghasilkan keluaran berupa jumlah dari sel-sel anggota *cage* tersebut.

Diagram kelas Cage dapat dilihat pada Gambar 4.9.

4.4.3 Kelas Cell

Kelas Cell mempunyai beberapa atribut, yaitu:

1. row, yaitu posisi baris dari sel tersebut.
2. column, yaitu posisi kolom dari sel tersebut.
3. cageID, yaitu nomor *cage* yang berisi sel tersebut.
4. value, yaitu nilai dari sel tersebut.



Gambar 4.10: Diagram kelas Cell.

Kelas Cell mempunyai beberapa *method*, yaitu:

1. Cell(int row, int column, int cageID), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa nomor baris, dan nomor kolom dari sel tersebut, dan nomor *cage* yang berisi sel tersebut.
2. setValue(Integer value), yaitu *method* untuk mengisi sebuah sel tersebut dengan nilai yang telah ditentukan. *Method* ini menerima masukan berupa nilai yang akan diisikan ke dalam sel tersebut.
3. getValue(), yaitu *method* untuk mendapatkan nilai dari sel tersebut. *Method* ini menghasilkan keluaran berupa nilai dari sel tersebut.
4. getRow(), yaitu *method* untuk mendapatkan nomor baris dari sebuah sel. *Method* ini menghasilkan keluaran berupa nomor baris dari sel tersebut.
5. getColumn(), yaitu *method* untuk mendapatkan nomor kolom dari sebuah sel. *Method* ini menghasilkan keluaran berupa nomor kolom dari sel tersebut.
6. getCageID(), yaitu *method* untuk mendapatkan nomor *cage* yang berisi sebuah sel. *Method* ini menghasilkan keluaran berupa nomor *cage* yang berisi sel tersebut.

Diagram kelas Cell dapat dilihat pada Gambar 4.10.

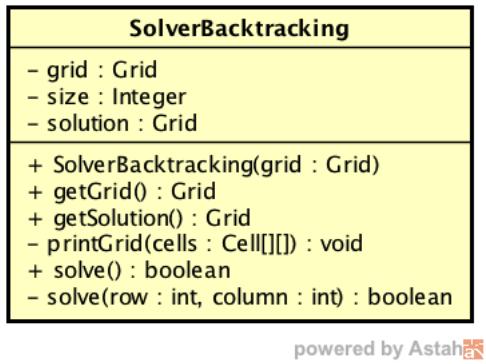
4.4.4 Kelas SolverBacktracking

Kelas SolverBacktracking mempunyai beberapa atribut, yaitu:

1. grid, yaitu *grid* yang akan diselesaikan oleh *solver* dengan algoritma *backtracking*.
2. size, yaitu ukuran dari *grid* yang akan diselesaikan oleh *solver* dengan algoritma *backtracking*.
3. solution, yaitu *grid* yang sudah diselesaikan oleh *solver* dengan algoritma *backtracking*.

Kelas SolverBacktracking mempunyai beberapa *method*, yaitu:

1. SolverBacktracking(Grid grid), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *grid* yang akan diselesaikan oleh *solver* dengan algoritma *backtracking*.



Gambar 4.11: Diagram kelas SolverBacktracking.

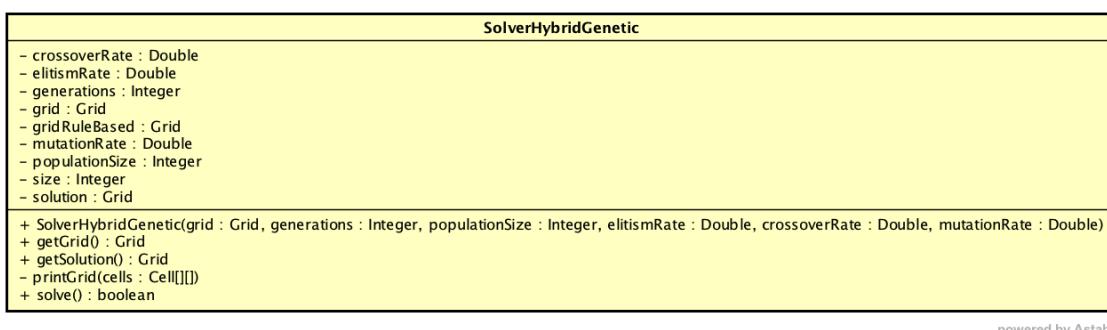
2. `solve()`, yaitu *method* pembungkus dari *method* `solve(int row, int column)`. *Method* ini menghasilkan keluaran apakah *solver* berhasil menyelesaikan teka-teki Calcudoku atau tidak. *Solver* bekerja mulai dari sel pada sudut kiri atas, lalu bergerak ke kanan sampai ke sel yang paling kanan, lalu bergerak ke baris berikutnya sampai ke baris yang paling bawah, selesai pada sel pada sudut kanan bawah.
3. `solve(int row, int column)`, yaitu *method* yang mencoba untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *backtracking*. *Method* ini menerima masukan berupa nomor baris dan nomor kolom yang akan diisi oleh *solver* dan menghasilkan keluaran apakah nilai yang diisi oleh *solver valid* atau tidak. *Solver* akan mulai mengisi sel dari angka 1. Jika berhasil, maka *solver* akan maju ke sel berikutnya. Jika gagal, maka *solver* akan mencoba kemungkinan angka berikutnya. Jika semua kemungkinan angka gagal, maka *solver* akan mundur ke sel sebelumnya dan mencoba kemungkinan angka berikutnya.
4. `getGrid()`, yaitu *method* untuk mendapatkan *grid*. *Method* ini menghasilkan keluaran berupa *grid*.
5. `getSolution()`, yaitu *method* untuk mendapatkan solusi dari *grid* yang sudah diselesaikan oleh *solver*. *Method* ini menghasilkan keluaran berupa solusi dari *grid* tersebut.
6. `printGrid()`, yaitu *method* untuk mencetak isi *grid* ke layar. *Method* ini menerima masukan berupa *grid* yang akan dicetak isinya ke layar.

Diagram kelas SolverBacktracking dapat dilihat pada Gambar 4.11.

4.4.5 Kelas SolverHybridGenetic

Kelas SolverHybridGenetic mempunyai beberapa atribut, yaitu:

1. `grid`, yaitu *grid* yang akan diselesaikan oleh *solver* dengan algoritma *hybrid genetic*.
2. `gridRuleBased`, yaitu *grid* yang telah diselesaikan oleh algoritma *rule based*.
3. `size`, yaitu ukuran dari *grid* yang akan diselesaikan oleh *solver* dengan algoritma *hybrid genetic*.
4. `solution`, yaitu *grid* yang sudah diselesaikan oleh *solver* dengan algoritma *hybrid genetic*.
5. `generations`, yaitu jumlah generasi maksimum yang akan dibangkitkan oleh algoritma genetik.
6. `populationSize`, yaitu jumlah kromosom yang akan dibangkitkan dalam sebuah generasi.
7. `elitismRate`, yaitu parameter tingkat *elitism* dalam algoritma genetik..



Gambar 4.12: Diagram kelas SolverHybridGenetic.

8. crossoverRate, yaitu parameter tingkat kawin silang dalam algoritma genetik.
9. mutationRate, yaitu parameter tingkat mutasi dalam algoritma genetik.

Kelas SolverHybridGenetic mempunyai beberapa *method*, yaitu:

1. SolverHybridGenetic(Grid grid, Integer generations, Integer populationSize, Double elitismRate, Double crossoverRate, Double mutationRate), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *grid* yang akan diselesaikan oleh *solver* dengan algoritma *hybrid genetic*, dan parameter-parameter algoritma genetik, yaitu jumlah generasi, jumlah populasi dalam sebuah generasi, tingkat *elitism*, tingkat kawin silang, dan tingkat mutasi.
2. getGrid(), yaitu *method* untuk mendapatkan *grid* yang akan diselesaikan oleh *solver* dengan algoritma *hybrid genetic*.
3. getSolution(), yaitu *method* untuk mendapatkan *grid* yang sudah diselesaikan oleh *solver* dengan algoritma *hybrid genetic*.
4. solve(), yaitu *method* yang mencoba untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *hybrid genetic*. *Method* ini akan memanggil solver algoritma *rule based*. Jika algoritma *rule based* gagal dalam menyelesaikan teka-teki Calcudoku, maka *method* akan memanggil solver algoritma genetik. *Method* ini menghasilkan keluaran apakah *solver* berhasil menyelesaikan teka-teki Calcudoku atau tidak.
5. printGrid(), yaitu *method* untuk mencetak isi *grid* ke layar. *Method* ini menerima masukan berupa *grid* yang akan dicetak isinya ke layar.

Diagram kelas SolverHybridGenetic dapat dilihat pada Gambar 4.12.

4.4.6 Kelas SolverRuleBased

Kelas SolverRuleBased mempunyai beberapa atribut, yaitu:

1. grid, yaitu *grid* yang akan diselesaikan oleh *solver* dengan algoritma *rule based*.
2. size, yaitu ukuran dari *grid* yang akan diselesaikan oleh *solver* dengan algoritma *rule based*.
3. solution, yaitu *grid* yang sudah diselesaikan oleh *solver* dengan algoritma *rule based*.
4. possibleValues, yaitu sebuah *array* yang menampung semua kemungkinan angka yang mungkin untuk setiap sel yang ada di dalam *grid*

Kelas SolverRuleBased mempunyai *method-method* berikut:

1. SolverRuleBased(Grid grid), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *grid* yang akan diselesaikan oleh *solver* dengan algoritma *rule based*, dan parameter-parameter algoritma genetik, yaitu jumlah generasi, jumlah populasi dalam sebuah generasi, tingkat *elitism*, tingkat kawin silang, dan tingkat mutasi.
2. generatePossibleValuesArray(), yaitu *method* yang membangkitkan kemungkinan angka-angka yang mungkin untuk setiap sel yang ada di dalam *grid*. Angka-angka yang mungkin adalah dari 1 sampai ke ukuran dari *grid*. *Method* ini menghasilkan keluaran berupa array yang menampung semua kemungkinan angka yang mungkin untuk setiap sel yang ada di dalam *grid*.
3. solve(), yaitu *method* yang mencoba untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *rule based*. *Method* ini menghasilkan keluaran apakah *solver* berhasil menyelesaikan teka-teki Calcudoku atau tidak. *Solver* akan mencoba menyelesaikan teka-teki Calcudoku menggunakan aturan-aturan logika, misalnya *single square rule*, *killer combination rule*, *naked subset rule*, *hidden subset rule*, dan *evil twin rule*. Aturan *single square* dan *killer combination* hanya dipakai sekali, dan dilakukan oleh *method* ini, sedangkan aturan *naked subset*, *hidden subset*, dan *evil twin* dapat dipakai berkali-kali, dan dilakukan oleh *method* solveLoop().
4. solveLoop(), yaitu *method* yang mengaplikasikan aturan *naked subset*, *hidden subset*, dan *evil twin* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa *array* kemungkinan angka yang mungkin untuk setiap sel yang ada di dalam *grid*. *Method* ini akan diulang sampai *method* ini tidak bisa mengisi sel-sel yang ada di dalam *grid*.
5. getRowPossibleValues(int rowNumber), yaitu *method* untuk mendapatkan kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam baris yang diminta. *Method* ini menerima masukan berupa nomor baris yang diminta dan menghasilkan keluaran berupa kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam baris yang diminta.
6. getColumnPossibleValues(int rowNumber), yaitu *method* untuk mendapatkan kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam kolom yang diminta. *Method* ini menerima masukan berupa nomor kolom yang diminta dan menghasilkan keluaran berupa kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam kolom yang diminta.
7. singleSquare(), yaitu *method* yang mengaplikasikan aturan *single square* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*.
8. killerCombination(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Dalam perangkat lunak ini aturan ini dibatasi hanya untuk *cage* yang berukuran dua sel.
9. killerCombinationCageSize2(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
10. killerCombinationCageSize2GridSize3(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 3×3 yang sedang diselesaikan oleh algoritma *rule based*.
11. killerCombinationCageSize2GridSize4(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 4×4 yang sedang diselesaikan oleh algoritma *rule based*.

12. killerCombinationCageSize2GridSize5(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 5×5 yang sedang diselesaikan oleh algoritma *rule based*.
13. killerCombinationCageSize2GridSize6(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 6×6 yang sedang diselesaikan oleh algoritma *rule based*.
14. killerCombinationCageSize2GridSize7(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 7×7 yang sedang diselesaikan oleh algoritma *rule based*.
15. killerCombinationCageSize2GridSize8(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 8×8 yang sedang diselesaikan oleh algoritma *rule based*.
16. killerCombinationCageSize2GridSize9(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 9×9 yang sedang diselesaikan oleh algoritma *rule based*.
17. nakedSingle(), yaitu *method* yang mengaplikasikan aturan *naked single* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Aturan ini diaplikasikan untuk baris (*nakedSingleRow()*) dan untuk kolom (*nakedSingleColumn()*).
18. nakedSingleRow(), yaitu *method* yang mengaplikasikan aturan *naked single* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
19. nakedSingleRow(int row), yaitu *method* yang mengaplikasikan aturan *naked single* kepada sebuah baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor baris yang akan diaplikasikan dengan aturan *naked single*.
20. nakedSingleColumn(), yaitu *method* yang mengaplikasikan aturan *naked single* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
21. nakedSingleColumn(int column), yaitu *method* yang mengaplikasikan aturan *naked single* kepada sebuah kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor kolom yang akan diaplikasikan dengan aturan *naked single*.
22. nakedSingle(int row, int column), yaitu *method* yang mengaplikasikan aturan *naked single* kepada sebuah sel yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan diaplikasikan dengan aturan *naked single*.
23. nakedDouble(), yaitu *method* yang mengaplikasikan aturan *naked double* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Aturan ini diaplikasikan untuk baris (*nakedSingleDouble()*) dan untuk kolom (*nakedSingleDouble()*).
24. nakedDoubleRow(), yaitu *method* yang mengaplikasikan aturan *naked double* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
25. nakedDoubleRow(int row), yaitu *method* yang mengaplikasikan aturan *naked double* kepada sebuah baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor baris yang akan diaplikasikan dengan aturan *naked double*.

26. `nakedDoubleRow(int row, ArrayList<Integer> doublePossibleValues, ArrayList<Integer> doublePossibleIndexes)`, yaitu *method* yang mengaplikasikan aturan *naked double* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menyimpan nilai-nilai yang disimpan pada *doublePossibleValues* pada kolom-kolom yang nomor kolomnya disimpan pada *array doublePossibleIndexes* dan menghapus nilai-nilai tersebut dari kolom-kolom lainnya. *Method* ini menerima masukan berupa nomor baris yang akan diaplikasikan dengan aturan *naked double*, sebuah *array* yang berisi nilai-nilai, dan sebuah *array* yang berisi nomor-nomor kolom.
27. `nakedDoubleColumn()`, yaitu *method* yang mengaplikasikan aturan *naked double* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
28. `nakedDoubleColumn(int column)`, yaitu *method* yang mengaplikasikan aturan *naked double* kepada sebuah kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor kolom yang akan diaplikasikan dengan aturan *naked double*.
29. `nakedDoubleColumn(int column, ArrayList<Integer> doublePossibleValues, ArrayList<Integer> doublePossibleIndexes)`, yaitu *method* yang mengaplikasikan aturan *naked double* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menyimpan nilai-nilai yang disimpan pada *doublePossibleValues* pada baris-baris yang nomor barisnya disimpan pada *array doublePossibleIndexes* dan menghapus nilai-nilai tersebut dari baris-baris lainnya. *Method* ini menerima masukan berupa nomor kolom yang akan diaplikasikan dengan aturan *naked double*, sebuah *array* yang berisi nilai-nilai, dan sebuah *array* yang berisi nomor-nomor baris.
30. `hiddenSingle()`, yaitu *method* yang mengaplikasikan aturan *hidden single* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Aturan ini diaplikasikan untuk baris (`hiddenSingleRow()`) dan untuk kolom (`hiddenSingleColumn()`).
31. `hiddenSingleRow()`, yaitu *method* yang mengaplikasikan aturan *hidden single* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
32. `hiddenSingleRow(int row)`, yaitu *method* yang mengaplikasikan aturan *hidden single* kepada sebuah baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor baris yang akan diaplikasikan dengan aturan *hidden single*.
33. `hiddenSingleColumn()`, yaitu *method* yang mengaplikasikan aturan *hidden single* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
34. `hiddenSingleColumn(int column)`, yaitu *method* yang mengaplikasikan aturan *hidden single* kepada sebuah kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor kolom yang akan diaplikasikan dengan aturan *hidden single*.
35. `hiddenSingle(int row, int column)`, yaitu *method* yang mengaplikasikan aturan *hidden single* kepada sebuah sel yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan diaplikasikan dengan aturan *hidden single*.
36. `setCellValue(int row, int column, int value)`, yaitu *method* untuk mengisi sebuah sel dengan nilai yang telah ditentukan. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan diisi dan nilai dari sel tersebut.

37. removePossibleValues(int row, int column, int value), yaitu *method* untuk menghapus kemungkinan nilai yang sudah digunakan dalam sebuah sel. *Method* ini menghapus nilai tersebut dari baris yang sama dan kolom yang sama. *Method* ini menerima masukan berupa nomor baris, nomor kolom, dan nilai yang akan dihapus dari sel-sel lain dalam baris dan kolom tempat sel tersebut berada.
38. removeImpossibleValuesCage(Cage cage, ArrayList<Integer> values), yaitu *method* untuk menghabus kemungkinan nilai yang tidak mungkin dari sel-sel di dalam sebuah *cage*. Method ini menerima masukan berupa sebuah *cage* dan sebuah *array* yang berisi nilai-nilai yang mungkin.
39. removeImpossibleValuesCell(int row, int column, ArrayList<Integer> values), yaitu *method* untuk menghabus kemungkinan nilai yang tidak mungkin dari sebuah sel yang diminta. Method ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang diminta, dan sebuah *array* yang berisi nilai-nilai yang mungkin.
40. createRetainAllArray(), yaitu *method* yang menghasilkan *array* yang berisi semua angka dari 1 sampai ukuran dari *grid*. Method ini menghasilkan keluaran berupa *array* yang berisi semua angka dari 1 sampai ukuran dari *grid*.
41. getGridArrayList(), yaitu *method* untuk mendapatkan isi dari *grid* dalam bentuk *ArrayList*. Method ini menghasilkan keluaran berupa isi dari *grid* dalam bentuk *ArrayList*.
42. getGrid, yaitu *method* untuk mendapatkan *grid*. Method ini menghasilkan keluaran berupa *grid*.
43. getSolution, yaitu *method* untuk mendapatkan *grid* yang sudah diselesaikan oleh algoritma *rule based*. Method ini menghasilkan keluaran berupa *grid* yang sudah diselesaikan oleh algoritma *rule based*.
44. printGrid(), yaitu *method* untuk mencetak isi *grid* ke layar.
45. printPossibleValues(), yaitu *method* untuk mencetak kemungkinan angka-angka yang valid untuk setiap sel di dalam *grid* ke layar.

Diagram kelas SolverRuleBased dapat dilihat pada Gambar 4.13.

4.4.7 Kelas SolverGenetic

Kelas SolverGenetic mempunyai atribut-atribut berikut, yaitu:

1. grid, yaitu *grid* yang akan diselesaikan oleh *solver* dengan algoritma genetik.
2. size, yaitu ukuran dari *grid* yang akan diselesaikan oleh *solver* dengan algoritma genetik.
3. isGridFixed, yaitu sebuah matriks yang berisi apakah sel tersebut sudah diisi oleh algoritma *rule based* atau belum. Nilai dari sel yang sudah diisi oleh *rule based* tidak boleh diganti atau dihapus.
4. randomGenerator, yaitu pembangkit angka acak.
5. generations, yaitu jumlah generasi maksimum yang akan dibangkitkan oleh algoritma genetik.
6. populationSize, yaitu jumlah kromosom yang akan dibangkitkan dalam sebuah generasi.
7. elitismRate, yaitu parameter tingkat *elitism* dalam algoritma genetik.
8. crossoverRate, yaitu parameter tingkat kawin silang dalam algoritma genetik.

SolverRuleBased
<pre> - grid : Grid - possibleValues : ArrayList<ArrayList<Integer>> - size : Integer - solution : Grid + SolverRuleBased(grid : Grid) - createRetainAllArray() : ArrayList<Integer> - generatePossibleValuesArray() : ArrayList<Integer> + getColumnPossibleValues() : ArrayList<Integer>[] + getGrid() : Grid + getGridArrayList() : ArrayList<ArrayList<Integer>> + getRowPossibleValues() : ArrayList<Integer> + getSolution() : Grid - hiddenSingle() : void - hiddenSingle(row : int, column : int, value : int) : void - hiddenSingleColumn() : void - hiddenSingleColumn(column : int) : void - hiddenSingleRow() : void - hiddenSingleRow(row : int) : void - killerCombination() : void - killerCombinationCageSize2(cage : Cage) : void - killerCombinationCageSize2GridSize3(cage : Cage) : void - killerCombinationCageSize2GridSize4(cage : Cage) : void - killerCombinationCageSize2GridSize5(cage : Cage) : void - killerCombinationCageSize2GridSize6(cage : Cage) : void - killerCombinationCageSize2GridSize7(cage : Cage) : void - killerCombinationCageSizeGridSize8(cage : Cage) : void - killerCombinationCageSize2GridSize9(cage : Cage) : void - nakedDouble() : void - nakedDoubleColumn() : void - nakedDoubleColumn(column : int) : void - nakedDoubleColumn(column : int, doublePossibleValues : ArrayList<Integer>, doublePossibleIndexes : ArrayList<Integer>) : void - nakedDoubleRow() : void - nakedDoubleRow(row : int) : void - nakedDoubleRow(row : int, doublePossibleValues : ArrayList<Integer>, doublePossibleIndexes : ArrayList<Integer>) : void - nakedSingle() : void - nakedSingle(row : int, column : int) : void - nakedSingleColumn() : void - nakedSingleColumn(column : int) : void - nakedSingleRow() : void - nakedSingleRow(row : int) : void - printGrid() : void - printPossibleValues() : void - removeImpossibleValuesCage(cage : Cage, values : ArrayList<Integer>) : void - removeImpossibleValuesCell(row : int, column : int, values : ArrayList<Integer>) : void - removePossibleValues(row : int, column : int, value : int) : void - setCellValue(row : int, column : int, value : int) : void - singleSquare() : void + solve() : boolean - solveLoop() : ArrayList<ArrayList<Integer>></pre>

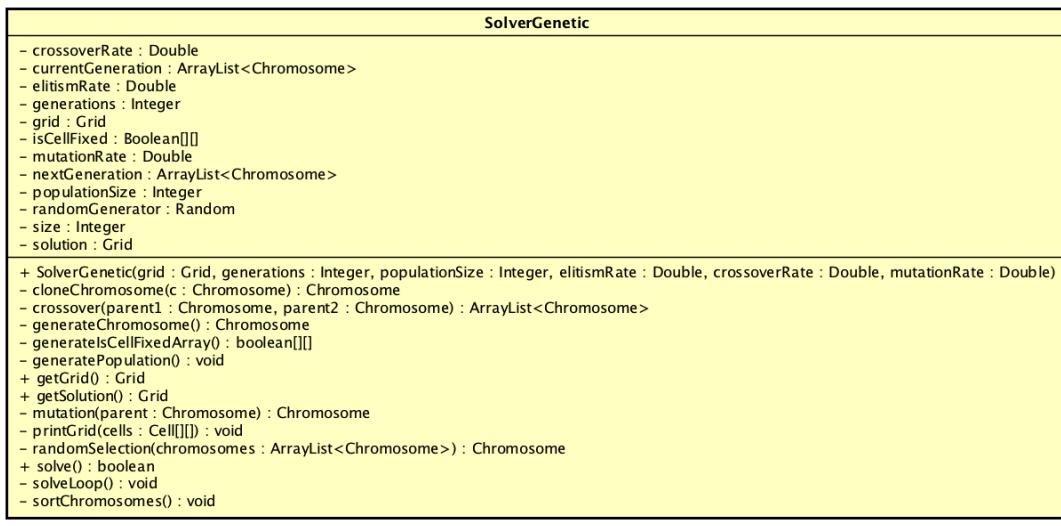
powered by Astah

Gambar 4.13: Diagram kelas SolverRuleBased.

9. mutationRate, yaitu parameter tingkat mutasi dalam algoritma genetik.
10. solution, yaitu *grid* yang sudah diselesaikan oleh *solver* dengan algoritma genetik.
11. currentGeneration, yaitu generasi saat ini dalam algoritma genetik. Algoritma genetik akan membangkitkan generasi baru (nextGeneration), dan generasi baru ini akan menjadi generasi saat ini, dan algoritma akan membangkitkan generasi baru berikutnya.
12. nextGeneration, yaitu generasi berikutnya dalam algoritma genetik.

Kelas SolverGenetic mempunyai *method-method* berikut:

1. SolverGenetic(Grid grid, Integer generations, Integer populationSize, Double elitismRate, Double crossoverRate, Double mutationRate), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *grid* yang akan diselesaikan oleh *solver* dengan algoritma genetik.
2. solve(), yaitu *method* yang mencoba untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma genetik. *Method* ini menghasilkan keluaran apakah *solver* berhasil menyelesaikan teka-teki Calcudoku atau tidak. Algoritma genetik berhasil menyelesaikan teka-teki Calcudoku jika ada kromosom yang nilai kelayakannya 1. *Solver* akan membangkitkan generasi pertama, sedangkan generasi-generasi berikutnya akan dibangkitkan oleh *method* solveLoop().
3. solveLoop(), yaitu *method* yang membangkitkan generasi berikutnya dari generasi sebelumnya menggunakan operator algoritma genetik, yaitu *elitism*, mutasi, dan kawin silang.
4. setParameters(int generations, int populationSize, double elitismRate, double crossoverRate, double mutationRate), yaitu *method* untuk menentukan jumlah generasi maksimum, jumlah kromosom dalam satu generasi, tingkat *elitism*, tingkat kawin silang, dan tingkat mutasi untuk algoritma genetik. Method ini menerima masukan berupa jumlah generasi maksimum, jumlah kromosom dalam satu generasi, tingkat *elitism*, tingkat kawin silang, dan tingkat mutasi untuk algoritma genetik.
5. generateIsCellFixedArray(), yaitu *method* yang membangkitkan matriks yang berisi apakah sel tersebut sudah diisi oleh algoritma *rule based* atau tidak. *Method* ini menghasilkan keluaran berupa sebuah matriks yang berisi apakah sel tersebut sudah diisi oleh algoritma *rule based* atau tidak.
6. generatePopulation(), yaitu *method* yang membangkitkan populasi kromosom awal.
7. generateChromosome(), yaitu *method* yang membangkitkan sebuah kromosom. *Method* ini menghasilkan keluaran berupa sebuah kromosom.
8. sortChromosomes(), yaitu *method* yang mengurutkan kromosom-kromosom dalam generasi saat ini berdasarkan nilai kelayakannya.
9. randomSelection(ArrayList<Chromosome> chromosomes), yaitu *method* untuk memilih sebuah kromosom dari sebuah populasi kromosom secara acak. *Method* ini menerima masukan berupa ArrayList yang berisi sekumpulan kromosom dan menghasilkan keluaran berupa sebuah kromosom yang terpilih. Kromosom untuk proses kawin silang dan proses mutasi dipilih secara acak menggunakan *method* ini.
10. cloneChromosome(Chromosome c), yaitu *method* untuk mengkopi sebuah kromosom. Method ini menerima masukan berupa kromosom yang akan dikopi dan menghasilkan keluaran berupa kromosom baru hasil kopian dari kromosom yang dikopi tersebut.



powered by Astah

Gambar 4.14: Diagram kelas SolverGenetic.

11. crossover(Chromosome parent1, Chromosome parent2), yaitu *method* yang mengaplikasikan operator kawin silang kepada dua kromosom. *Method* ini menerima masukan berupa dua kromosom yang akan dikawinsilangkan dan menghasilkan keluaran berupa sebuah *ArrayList* yang berisi dua kromosom hasil kawin silang.
12. mutation(Chromosome parent), yaitu *method* yang mengaplikasikan operator mutasi kepada sebuah kromosom. *Method* ini menerima masukan berupa kromosom yang akan dimutasi dan menghasilkan keluaran berupa kromosom hasil mutasi.
13. getGrid, yaitu *method* untuk mendapatkan *grid*. Method ini menghasilkan keluaran berupa *grid*.
14. getSolution, yaitu *method* untuk mendapatkan *grid* yang sudah diselesaikan oleh algoritma genetik. *Method* ini menghasilkan keluaran berupa *grid* yang sudah diselesaikan oleh algoritma genetik.
15. printGrid(), yaitu *method* untuk mencetak isi *grid* ke layar.

Diagram kelas SolverGenetic dapat dilihat pada Gambar 4.14.

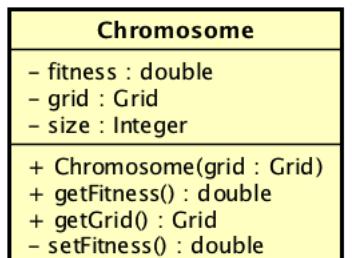
4.4.8 Kelas Chromosome

Kelas Chromosome mempunyai beberapa atribut, yaitu:

1. grid, yaitu sebuah *grid* yang sudah diisi dengan angka-angka secara acak.
2. size, yaitu ukuran dari sebuah *grid*.
3. fitness, yaitu nilai kelayakan dari sebuah *grid*.

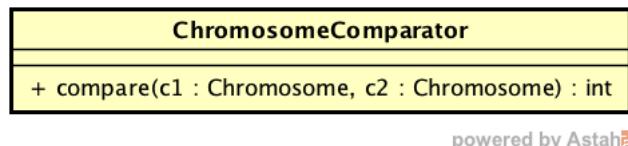
Kelas Chromosome mempunyai beberapa *method*, yaitu:

1. Chromosome(Grid grid), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa sebuah *grid* yang sudah diisi dengan angka-angka secara acak.
2. setFitness(), yaitu *method* yang menghitung nilai kelayakan untuk sebuah *grid*. *Method* ini menghasilkan keluaran berupa nilai kelayakan untuk *grid* tersebut.



powered by Astah

Gambar 4.15: Diagram kelas Chromosome.



powered by Astah

Gambar 4.16: Diagram kelas ChromosomeComparator.

3. `getFitness()`, yaitu *method* untuk mendapatkan nilai kelayakan untuk sebuah *grid*. *Method* ini menghasilkan keluaran berupa nilai kelayakan untuk *grid* tersebut.
4. `getGrid`, yaitu *method* untuk mendapatkan *grid*. *Method* ini menghasilkan keluaran berupa *grid*.

Diagram kelas Chromosome dapat dilihat pada Gambar 4.15.

4.4.9 Kelas ChromosomeComparator

Kelas ChromosomeComparator tidak mempunyai variabel, tetapi kelas ini mempunyai sebuah *method*, yaitu `compare(Chromosome c1, Chromosome c2)`. Fungsi dari *method* ini adalah membandingkan dua buah kromosom berdasarkan nilai kelayakannya. *Method* ini mengeluarkan hasil 1 jika *c1* lebih besar daripada *c2*, -1 jika *c1* lebih kecil daripada *c2*, atau 0 jika *c1* sama dengan *c2*. Diagram kelas ChromosomeComparator dapat dilihat pada Gambar 4.16.

4.4.10 Kelas Controller

Kelas Controller mempunyai beberapa satu atribut, yaitu *g*. *g* adalah representasi dari *grid* dalam teka-teki Calcudoku. *grid* adalah sebuah matriks yang berisi sel-sel. Matriks ini berukuran $n \times n$.

Kelas Controller mempunyai beberapa *method*, yaitu:

1. `Controller(Integer size, Integer numberOfCages, Integer[][] cageCells, String[] cageObjectives)`, yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa ukuran dari matriks *grid*, banyaknya *cage* yang terdapat dalam *grid*, matriks *cage assignment*, dan array *cage objectives*.
2. `setCellValue(int row, int column, Integer value)`, yaitu *method* untuk mengisi sebuah sel dengan nilai yang telah ditentukan. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan diisi dan nilai dari sel tersebut, dan menghasilkan keluaran apakah nilai dari sel tersebut *valid* atau tidak. Nilai dari sebuah sel *valid* jika nilai dari sel tersebut tidak berulang dalam baris dan kolom tempat sel tersebut berada, dan angka-angka dari *cage* yang berisi sel tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan.

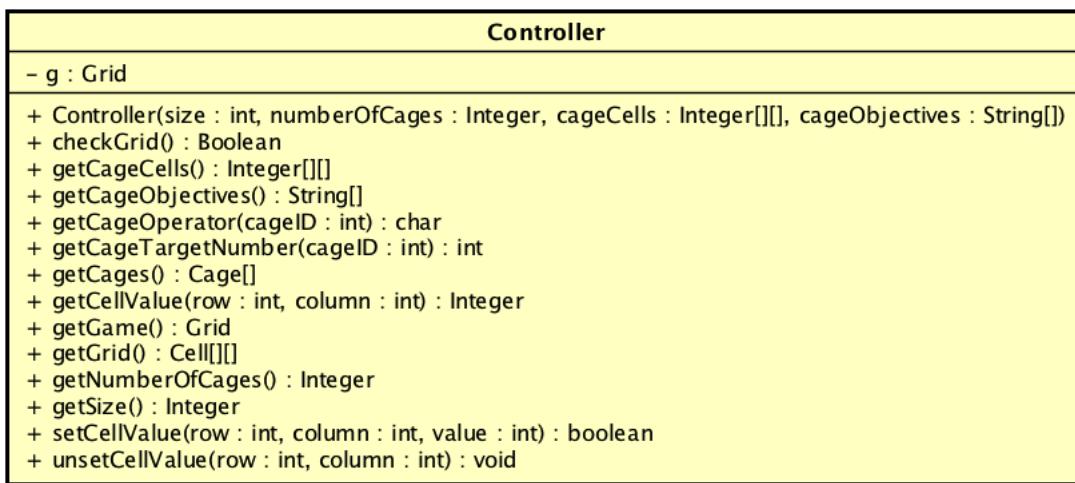
3. unsetCellValue(int row, int column), yaitu *method* untuk menghapus isi dari sebuah sel. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang akan dihapus isinya.
4. checkGrid(), yaitu *method* yang memeriksa apakah ada sel yang berisi nilai yang tidak *valid* atau tidak. *Method* ini akan menghasilkan apakah ada sel yang berisi nilai yang tidak *valid* atau tidak. *Method* ini hanya bekerja jika semua sel yang berada di dalam *grid* sudah diisi. Nilai dari sebuah sel *valid* jika nilai dari sel tersebut tidak berulang dalam baris dan kolom tempat sel tersebut berada, dan angka-angka dari *cage* yang berisi sel tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan.
5. getCellValue(int row, int column), yaitu *method* untuk mendapatkan isi dari sebuah sel. *Method* ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang diminta dan menghasilkan keluaran berupa isi dari sel yang diminta tersebut.
6. getSize(), yaitu *method* untuk mendapatkan ukuran dari *grid*. *Method* ini menghasilkan keluaran berupa ukuran dari *grid*.
7. getNumberOfCages(), yaitu *method* untuk mendapatkan jumlah *cage* yang ada di dalam *grid*. *Method* ini menghasilkan keluaran berupa jumlah *cage* yang ada di dalam *grid*.
8. getCageCells(), yaitu *method* untuk mendapatkan matriks *cage assignment* dari *grid*. *Method* ini menghasilkan keluaran berupa matriks *cage assignment* dari *grid*.
9. getCageObjectives(), yaitu *method* untuk mendapatkan *cage objectives* dari setiap *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa sebuah array yang berisi *cage objectives* dari setiap *cage* dalam *grid*.
10. getGridContents(), yaitu *method* untuk mendapatkan nilai dari setiap sel *grid*. *Method* ini menghasilkan keluaran berupa sebuah matriks yang berisi nilai dari setiap sel *grid*.
11. getCages(), yaitu *method* untuk mendapatkan semua *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa array yang berisi semua *cage* dalam *grid*.
12. getGame(), yaitu *method* untuk mendapatkan instansiasi saat ini dari kelas *Grid*. *Method* ini menghasilkan keluaran berupa instansiasi saat ini dari kelas *Grid*.

Diagram kelas Controller dapat dilihat pada Gambar 4.17.

4.4.11 Kelas Calcudoku

Kelas Calcudoku mempunyai beberapa atribut, yaitu:

1. puzzleFile, yaitu *file* permainan yang sedang dibuka oleh perangkat lunak. *File* ini berbentuk *file* teks.
2. size, yaitu ukuran dari matriks *grid* berdasarkan *file* permainan yang sedang dibuka.
3. numberCages, yaitu banyaknya *cage* yang terdapat dalam *grid* berdasarkan *file* permainan yang sedang dibuka.
4. cageCells, yaitu sebuah matriks *cage assignment* berdasarkan *file* permainan yang sedang dibuka. Matriks ini merepresentasikan posisi dari setiap *cage* dalam *grid*.
5. cageObjectives, yaitu sebuah *array* yang berisi *cage objectives* untuk setiap *cage* berdasarkan *file* permainan yang sedang dibuka. *Cage objectives* berisikan angka tujuan dan operasi matematika yang telah ditentukan.



powered by Astah

Gambar 4.17: Diagram kelas Controller.

6. c, yaitu sebuah instansiasi dari kelas Controller. c berfungsi untuk menghubungkan kelas-kelas yang berada di dalam *package* model dengan kelas-kelas yang berada di dalam *package* view.
7. menuBar, yaitu menu *bar* untuk perangkat lunak ini.
8. menuFile, yaitu menu File di dalam menu *bar*.
9. menuSolve, yaitu menu Solve di dalam menu *bar*.
10. menuItemLoad, yaitu menu *item* Load Puzzle File di dalam menu File.
11. menuItemReset, yaitu menu *item* Reset Puzzle di dalam menu File.
12. menuItemClose, yaitu menu *item* Close Puzzle File di dalam menu File.
13. menuItemCheck, yaitu menu *item* Check Puzzle di dalam menu File.
14. menuItemExit, yaitu menu *item* Exit di dalam menu File.
15. menuItemBacktracking, yaitu menu *item* Backtracking di dalam menu Solve.
16. menuItemHybridGenetic, yaitu menu *item* Hybrid Genetic di dalam menu File.
17. fileChooser, yaitu *file chooser* untuk membuka *file* permainan.
18. gui, yaitu representasi GUI dari *file* permainan yang sedang dibuka.

Kelas Calcudoku mempunyai beberapa *method*, yaitu:

1. Calcudoku(), yaitu konstruktor dari kelas ini. Konstruktor ini berfungsi untuk menginisialisasi *frame* GUI.
2. main(String args[]), yaitu *method main* dari perangkat lunak ini. *Method* ini berfungsi untuk menjalankan perangkat lunak ini.
3. initComponents(), yaitu *method* untuk menginisialisasi komponen-komponen dari *frame* GUI.
4. initWindowListener(), yaitu *method* untuk menginisialisasi *window listener* untuk *frame* GUI.

5. initActionListener(), yaitu *method* untuk menginisialisasi *action listener* untuk setiap menu *item* yang ada di dalam *frame GUI*.
6. initMenu(), yaitu *method* untuk menginisialisasi menu-menu dalam menu *bar* untuk *frame GUI*.
7. initMenuBar(), yaitu *method* untuk menginisialisasi menu *bar* untuk *frame GUI*.
8. menuItemLoadActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Load Puzzle File*" dipilih. *Method* ini akan membuka *file chooser* dimana pengguna memilih *file* permainan untuk dibuka oleh perangkat lunak. Jika perangkat lunak sudah membuka sebuah *file* permainan, maka akan keluar kotak dialog "*Are you sure you want to load another puzzle file?*", jika pengguna memilih "Yes", maka *file* permainan yang baru akan dibuka oleh perangkat lunak.
9. selectPuzzleFile, yaitu *method* yang memeriksa apakah *file* permainan yang akan dibuka *valid* atau tidak. Jika perangkat lunak gagal dalam membuka *file* permainan, maka akan keluar pesan peringatan "*Error in loading puzzle file*". Jika *file* permainan yang dibuka bukan *file* teks, maka akan keluar pesan peringatan "*Invalid puzzle file*". Jika *file* permainan tidak ditemukan, maka akan keluar pesan peringatan "*Puzzle file not found*".
10. menuItemResetActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Reset Puzzle*" dipilih. *Method* ini akan mengeluarkan kotak dialog "*Are you sure you want to reset this puzzle?*", jika pengguna memilih "Yes", maka perangkat lunak akan me-reset permainan. Jika perangkat lunak tidak sedang membuka *file* permainan, maka akan keluar pesan peringatan "*Puzzle file not loaded*".
11. menuItemCheckActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Check Puzzle*" dipilih. *Method* ini akan memeriksa apakah ada sel yang berisi nilai yang tidak *valid* atau tidak. Nilai dari sebuah sel *valid* jika nilai dari sel tersebut tidak berulang dalam baris dan kolom tempat sel tersebut berada, dan angka-angka dari *cage* yang berisi sel tersebut mencapai angka tujuan yang telah ditentukan jika dihitung menggunakan operator yang telah ditentukan. *Method* ini juga memeriksa apakah ada sel yang kosong atau tidak. Jika perangkat lunak tidak sedang membuka *file* permainan, maka akan keluar pesan peringatan "*Puzzle file not loaded*".
12. menuItemCloseActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Close Puzzle File*" dipilih. *Method* ini akan mengeluarkan kotak dialog "*Are you sure you want to close this puzzle file?*", jika pengguna memilih "Yes", maka perangkat lunak akan menutup *file* permainan dan instansiasi kelas GUI berdasarkan *file* permainan yang ditutup. Jika perangkat lunak tidak sedang membuka *file* permainan, maka akan keluar pesan peringatan "*Puzzle file not loaded*".
13. menuItemExitActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Exit*" dipilih. *Method* ini akan mengeluarkan kotak dialog "*Are you sure you want to exit this application*", jika pengguna memilih "Yes", maka perangkat lunak akan ditutup.
14. menuItemBacktrackingActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Backtracking*" dijalankan. *Method* ini akan mencoba menyelesaikan permainan berdasarkan *file* permainan yang dibuka dengan algoritma *backtracking*. Jika perangkat lunak tidak sedang membuka *file* permainan, maka akan keluar pesan peringatan "*Puzzle file not loaded*".
15. menuItemHybridGeneticActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Hybrid Genetic*" dijalankan. *Method* ini akan mencoba menyelesaikan

permainan berdasarkan *file* permainan yang dibuka dengan algoritma *hybrid genetic*. Jika perangkat lunak tidak sedang membuka *file* permainan, maka akan keluar pesan peringatan "*Puzzle file not loaded*".

16. menuItemGeneticParametersActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika menu "*Set Genetic Algorithm Parameters*" dijalankan. *Method* ini akan menampilkan *window* baru dimana pengguna bisa mengatur parameter-parameter untuk algoritma genetik dengan mengisi *form* yang disediakan pada *window* tersebut. Jika perangkat lunak tidak sedang membuka *file* permainan, maka akan keluar pesan peringatan "*Puzzle file not loaded*".
17. loadPuzzleFile(File puzzleFile), yaitu *method* yang menginisialisasi sebuah instansiasi dari kelas GUI berdasarkan *file* permainan yang dibuka. *Method* ini menerima masukan berupa sebuah *file* permainan. Jika perangkat lunak gagal dalam membuka *file* permainan, maka akan keluar pesan peringatan "*Invalid puzzle file*".
18. resetFrame(), yaitu *method* untuk me-reset *frame* setelah menutup *file* permainan. *Method* ini akan menutup instansiasi kelas GUI dan menghapus isi dari variabel *c* dalam *frame*.
19. clearVariables(), yaitu *method* untuk menghapus nilai dari variabel-variabel *puzzleFile*, *size*, *cageCells*, *numberOfCages*, dan *cageObjectives* dalam *frame*.
20. destroyFrame(), yaitu *method* untuk menutup *frame* saat menutup perangkat lunak. *Method* ini juga akan me-reset *frame* dan menghapus nilai dari semua variabel dalam *frame*.

Diagram kelas Calcudoku dapat dilihat pada Gambar 4.18.

4.4.12 Kelas WindowListener

Kelas WindowListener hanya mempunyai satu atribut, yaitu *frame*. *frame* adalah sebuah instansiasi dari kelas Calcudoku. Kelas ini mempunya beberapa *method*, yaitu:

1. WindowListener(Calcudoku frame), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa sebuah instansiasi dari kelas Calcudoku.
2. windowClosing(WindowEvent e), yaitu *method* yang meng-override *method* dari kelas WindowAdapter. *Method* ini berfungsi untuk menutup semua komponen GUI saat perangkat lunak ditutup.
3. windowClosed(WindowEvent e), yaitu *method* yang meng-override *method* dari kelas WindowAdapter. *Method* ini berfungsi untuk menutup perangkat lunak setelah semua komponen GUI ditutup.

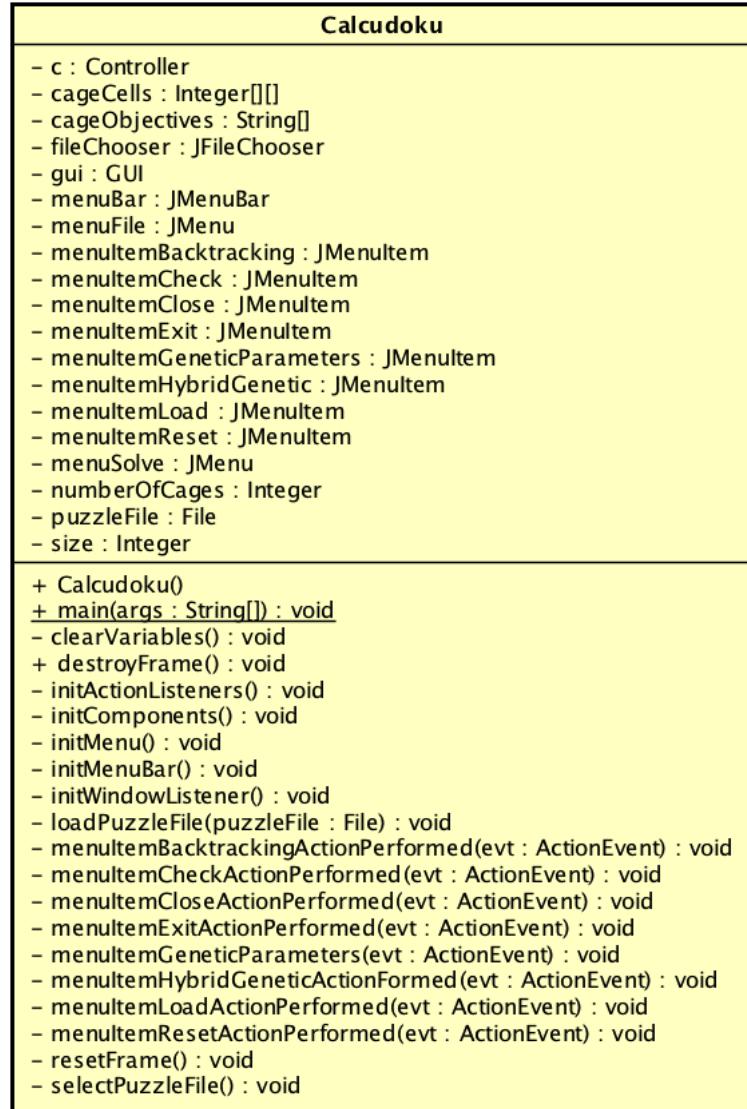
Diagram kelas WindowListener dapat dilihat pada Gambar 4.19.

4.4.13 Kelas PuzzleFileFilter

Kelas PuzzleFileFilter tidak mempunyai atribut, tetapi memiliki beberapa *method*, yaitu:

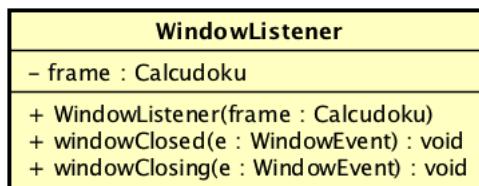
1. accept(File puzzleFile), yaitu *method* yang meng-override *method* dari kelas FileFilter. *Method* ini berfungsi untuk memeriksa apakah *file* yang akan dibuka memenuhi syarat atau tidak. *File* yang memenuhi syarat adalah *file* teks.
2. getDescription(), yaitu *method* yang meng-override *method* dari kelas FileFilter. *Method* ini berfungsi untuk menampilkan deskripsi *file* yang memenuhi syarat.

Diagram kelas PuzzleFileFilter dapat dilihat pada Gambar 4.20.



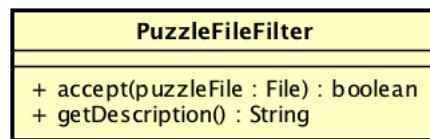
powered by Astah

Gambar 4.18: Diagram kelas Calcudoku.



powered by Astah

Gambar 4.19: Diagram kelas WindowListener.



powered by Astah

Gambar 4.20: Diagram kelas PuzzleFileFilter.

4.4.14 Kelas GUI

Kelas GUI mempunyai beberapa atribut, yaitu:

1. c, yaitu sebuah instansiasi dari kelas Controller. c berfungsi untuk menghubungkan kelas-kelas yang berada di dalam *package* model dengan kelas-kelas yang berada di dalam *package* view.
2. game, yaitu sebuah instansi dari kelas Grid berdasarkan *file* permainan yang sedang dibuka.
3. size, yaitu ukuran dari matriks *grid* berdasarkan *file* permainan yang sedang dibuka.
4. numberOfCages, yaitu banyaknya *cage* yang terdapat dalam *grid* berdasarkan *file* permainan yang sedang dibuka.
5. cageCells, yaitu sebuah matriks *cage assignment* berdasarkan *file* permainan yang sedang dibuka. Matriks ini merepresentasikan posisi dari setiap *cage* dalam *grid*.
6. cageObjectives, yaitu sebuah *array* yang berisi *cage objectives* untuk setiap *cage* berdasarkan *file* permainan yang sedang dibuka. *Cage objectives* berisikan angka tujuan dan operasi matematika yang telah ditentukan.
7. grid, yaitu sebuah *array* yang berisikan semua sel yang ada di dalam *grid*.
8. cages, yaitu sebuah *array* yang berisikan semua *cage* yang ada di dalam *grid*. *textFields*, yaitu sebuah *array* yang berisikan *text field*. Setiap *text field* merepresentasikan sebuah sel yang ada di dalam *grid*. *textFieldCoordinates*, yang berfungsi untuk memetakan *text field* dengan koordinat posisinya. *cellTextFieldListeners*, yaitu sebuah *array* yang berisikan *document listener* untuk semua sel yang ada di dalam *grid*. *font*, yaitu *font* yang digunakan di dalam GUI ini. *cellSize*, yaitu ukuran sebuah sel di dalam GUI ini. *cellBorderWidth*, yaitu ukuran garis pembatas antara dua sel yang berada di dalam sebuah *cage* yang sama. *cageBorderWidth*, yaitu ukuran garis pembatas antara dua sel yang berada di dalam dua *cage* yang berbeda, dan ukuran garis pembatas *grid*.
9. generations, yaitu jumlah generasi maksimum yang akan dibangkitkan oleh algoritma genetik.
10. populationSize, yaitu jumlah kromosom yang akan dibangkitkan dalam sebuah generasi.
11. elitismRate, yaitu parameter tingkat *elitism* dalam algoritma genetik..
12. crossoverRate, yaitu parameter tingkat kawin silang dalam algoritma genetik.
13. mutationRate, yaitu parameter tingkat mutasi dalam algoritma genetik.

Kelas GUI mempunyai beberapa *method*, yaitu:

1. GUI(Controller c), yaitu konstruktor dari kelas ini. Konstruktor ini berfungsi untuk menginisialisasi GUI berdasarkan *file* permainan yang dibuka. *Method* ini menerima masukan berupa sebuah instansiasi dari kelas Controller.
2. getController(), yaitu *method* untuk mendapatkan instansiasi dari kelas Controller. *Method* ini menghasilkan keluaran berupa instansiasi dari kelas Controller.
3. getGridSize(), yaitu *method* untuk mendapatkan ukuran dari *grid*. *Method* ini menghasilkan keluaran berupa ukuran dari *grid*.
4. getNumberOfCages(), yaitu *method* untuk mendapatkan jumlah *cage* yang ada di dalam *grid*. *Method* ini menghasilkan keluaran berupa jumlah *cage* yang ada di dalam *grid*.

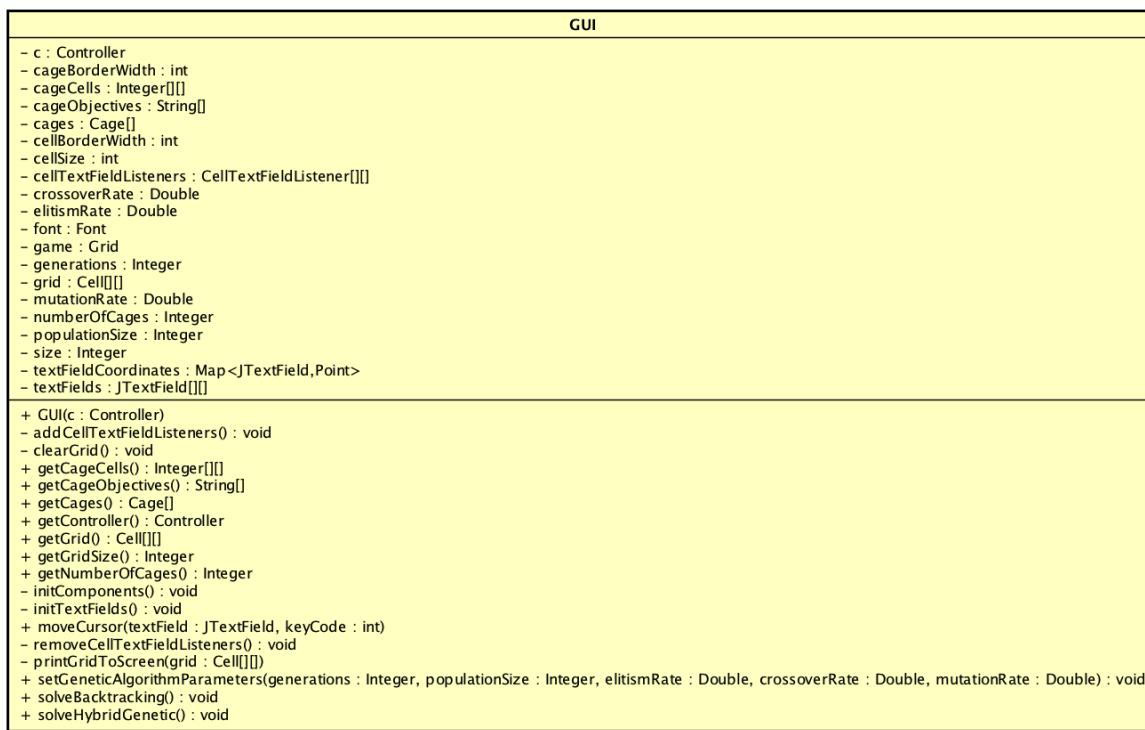
5. `getCageCells()`, yaitu *method* untuk mendapatkan matriks *cage assignment* dari *grid*. *Method* ini menghasilkan keluaran berupa matriks *cage assignment* dari *grid*.
6. `getCageObjectives()`, yaitu *method* untuk mendapatkan *cage objectives* dari setiap *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa sebuah array yang berisi *cage objectives* dari setiap *cage* dalam *grid*.
7. `getGrid()`, yaitu *method* untuk mendapatkan nilai dari setiap sel *grid*. *Method* ini menghasilkan keluaran berupa sebuah matriks yang berisi nilai dari setiap sel *grid*.
8. `getCages()`, yaitu *method* untuk mendapatkan semua *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa array yang berisi semua *cage* dalam *grid*.
9. `setGeneticAlgorithmParameters(Integer generations, Integer populationSize, Double elitismRate, Double crossoverRate, Double mutationRate)`, yaitu *method* untuk mengatur parameter-parameter untuk algoritma genetik. *Method* ini menerima masukan berupa jumlah generasi, jumlah populasi dalam sebuah generasi, tingkat *elitism*, tingkat kawin silang, dan tingkat mutasi.
10. `initTextFields`, yaitu *method* untuk menginisialisasi *text field* yang ada di dalam *grid* pada GUI.
11. `solveBacktracking()`, yaitu *method* untuk memanggil solver untuk menyelesaikan permainan Calcudoku menggunakan algoritma *backtracking*.
12. `solveHybridGenetic()`, yaitu *method* untuk memanggil solver untuk menyelesaikan permainan Calcudoku menggunakan algoritma *hybrid genetic*.
13. `printGridToScreen()`, yaitu *method* untuk mencetak isi *grid* ke GUI.
14. `clearGrid()`, yaitu *method* untuk menghapus isi dari semua sel yang ada di dalam *grid* yang akan diselesaikan oleh *solver* sebelum *solver* mencoba menyelesaikan *grid* tersebut.
15. `addCellTextFieldListeners()`, yaitu *method* yang berfungsi untuk menambahkan *document listener* untuk semua *text field* yang ada pada GUI setelah *solver* berhasil atau gagal dalam menyelesaikan permainan Calcudoku.
16. `removeCellTextFieldListeners()`, yaitu *method* yang berfungsi untuk menghapus *document listener* untuk semua *text field* yang ada pada GUI sebelum *solver* mencoba untuk menyelesaikan permainan Calcudoku, untuk mencegah munculnya pesan peringatan jika ada sel yang kosong atau jika ada angka yang berulang di dalam sebuah baris atau kolom.
17. `moveCursor(JTextField textField)`, yaitu *method* yang berfungsi untuk pindah dari sebuah *text field* ke *text field* di sebelahnya dengan menggunakan tombol-tombol panah. *Method* ini menerima masukan berupa sebuah *text field*.

Diagram kelas GUI dapat dilihat pada Gambar 4.21.

4.4.15 Kelas CellKeyListener

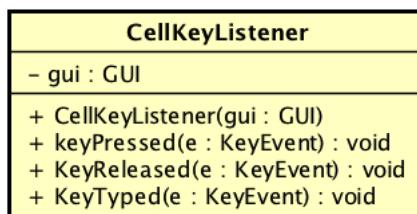
Kelas CellKeyListener hanya mempunyai satu atribut, yaitu `gui`. `gui` adalah sebuah instansiasi dari kelas GUI. Kelas ini mempunya beberapa *method*, yaitu:

1. `CellKeyLIstener(GUI gui)`, yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa sebuah instansiasi dari kelas GUI.



powered by Astah

Gambar 4.21: Diagram kelas GUI.



powered by Astah

Gambar 4.22: Diagram kelas CellKeyListener.

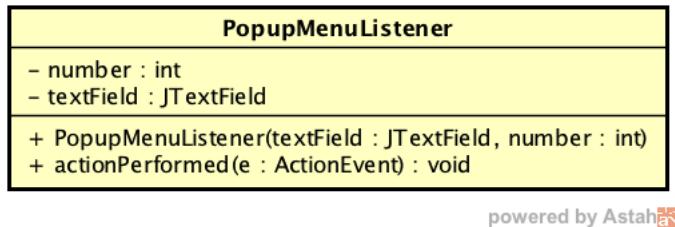
2. `keyPressed(KeyEvent e)`, yaitu *method* yang meng-*override* *method* dari kelas KeyListener. *Method* ini berfungsi untuk pindah dari sebuah *text field* ke *text field* di sebelahnya dengan menggunakan tombol-tombol panah.
3. `keyTyped(KeyEvent e)`, yaitu *method* yang meng-*override* *method* dari kelas KeyListener. *Method* ini berfungsi agar *text field* hanya bisa diisi oleh sebuah angka.
4. `keyReleased(KeyEvent e)`, yaitu *method* yang meng-*override* *method* dari kelas KeyListener. *Method* ini kosong.

Diagram kelas CellKeyListener dapat dilihat pada Gambar 4.22.

4.4.16 Kelas PopupMenuListener

Kelas PopupMenuListener mempunyai beberapa atribut, yaitu:

1. `textField`, yaitu sebuah *text field* yang merepresentasikan sebuah sel yang berada di dalam *grid*.



Gambar 4.23: Diagram kelas PopupMenuItemListener.

2. number, yaitu sebuah angka.

Kelas PopupMenuItemListener mempunyai beberapa *method*, yaitu:

1. PopupMenuItemListener(JTextField textField, int number), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa sebuah *text field* dan sebuah angka.
2. actionPerformed(ActionEvent e), yaitu *method* yang meng-override *method* dari kelas ActionListener. *Method* ini berfungsi untuk mengisi sebuah *text field* dengan angka yang dipilih.

Diagram kelas PopupMenuItemListener dapat dilihat pada Gambar 4.23.

4.4.17 Kelas CellTextFieldListener

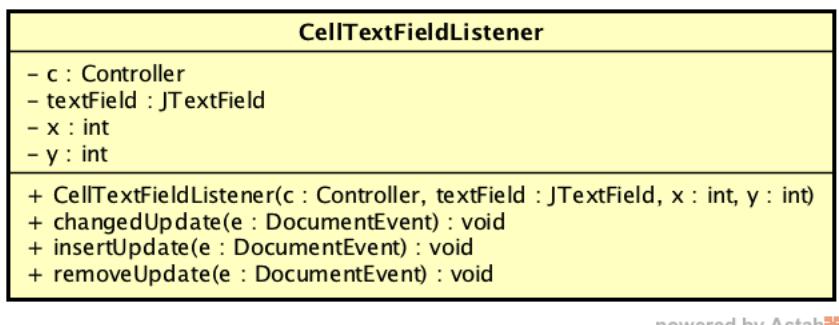
Kelas CellTextFieldListener mempunyai beberapa atribut, yaitu:

1. c, yaitu sebuah instansiasi dari kelas Controller. c berfungsi untuk menghubungkan kelas-kelas yang berada di dalam *package* model dengan kelas-kelas yang berada di dalam *package* view.
2. textField, yaitu sebuah *text field* yang merepresentasikan sebuah sel yang berada di dalam *grid*.
3. x, yaitu koordinat x dari *text field*.
4. y, yaitu koordinat y dari *text field*.

Kelas CellTextFieldListener mempunyai beberapa *method*, yaitu:

1. CellTextFieldListener(Controller c, JTextField textField, int x, int y), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa sebuah instansiasi dari kelas Controller, sebuah *text field*, dan koordinat dari *text field* tersebut.
2. insertUpdate(DocumentEvent e), yaitu *method* yang meng-override *method* dari kelas DocumentEvent. *Method* ini berfungsi untuk mengisi sebuah sel pada kelas Grid dengan angka yang diisikan ke dalam sel pada GUI.
3. removeUpdate(DocumentEvent e), yaitu *method* yang meng-override *method* dari kelas DocumentEvent. *Method* ini berfungsi untuk menghapus isi sebuah sel pada kelas Grid jika angka dalam sel pada GUI dihapus.
4. changeUpdate(DocumentEvent e), yaitu *method* yang meng-override *method* dari kelas DocumentEvent. *Method* ini berfungsi untuk menghapus isi sebuah sel pada kelas Grid dan mengisi sebuah sel pada kelas Grid dengan angka yang baru yang diisikan ke dalam sel pada GUI jika terjadi perubahan angka yang diisikan ke dalam sel pada GUI.

Diagram kelas CellTextFieldListener dapat dilihat pada Gambar 4.24.



powered by Astah

Gambar 4.24: Diagram kelas CellTextFieldListner.

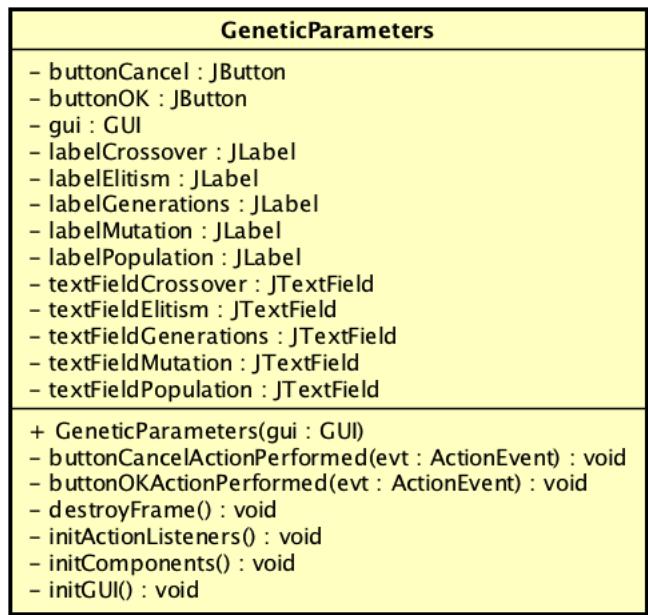
4.4.18 Kelas GeneticParameters

Kelas GeneticParameters mempunyai beberapa atribut, yaitu:

1. gui, yaitu representasi GUI dari *file* permainan yang sedang dibuka.
2. labelGenerations, yaitu label yang bertuliskan "*Generations*".
3. labelPopulation, yaitu label yang bertuliskan "*Population Size*".
4. labelElitism, yaitu label yang bertuliskan "*Elitism Rate*".
5. labelCrossover, yaitu label yang bertuliskan "*Crossover Rate*".
6. labelMutation, yaitu label yang bertuliskan "*Mutation Rate*".
7. textFieldGenerations, yaitu *text field* untuk mengisi nilai jumlah generasi.
8. textFieldPopulation, yaitu *text field* untuk mengisi nilai jumlah populasi dalam sebuah generasi.
9. textFieldElitism, yaitu *text field* untuk mengisi nilai tingkat *elitism*.
10. textFieldCrossover, yaitu *text field* untuk mengisi nilai tingkat kawin silang.
11. textFieldMutation, yaitu *text field* untuk mengisi nilai tingkat mutasi.
12. buttonOK, yaitu tombol untuk mengganti nilai-nilai parameter untuk algoritma genetik dengan nilai-nilai yang dimasukkan ke dalam *form* yang disediakan.
13. buttonCancel, yaitu tombol untuk membatalkan perubahan nilai-nilai parameter untuk algoritma genetik.

Kelas GeneticParameters mempunyai beberapa *method*, yaitu:

1. GeneticParameters(GUI gui), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa sebuah instansiasi dari kelas GUI.
2. initComponents(), yaitu *method* untuk menginisialisasi komponen-komponen dari *form*.
3. initActionListener(), yaitu *method* untuk menginisialisasi *action listener* untuk setiap tombol yang ada di dalam *form*.
4. initGUI(), yaitu *method* untuk menginisialisasi GUI dari *form*.
5. initMenuBar(), yaitu *method* untuk menginisialisasi menu *bar* untuk *frame* GUI.



powered by Astah

Gambar 4.25: Diagram kelas GeneticParameters.

6. buttonOKActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika tombol "OK" ditekan. *Method* ini akan mengganti nilai-nilai parameter untuk algoritma genetik dengan nilai-nilai yang telah dimasukkan ke dalam *form* yang disediakan.
7. buttonCancelActionPerformed(ActionEvent evt), yaitu *method* yang akan dijalankan jika tombol "Cancel" ditekan. *Method* ini akan membatalkan perubahan nilai-nilai parameter untuk algoritma genetik.
8. destroyFrame(), yaitu *method* untuk menutup *form*.

Diagram kelas GeneticParameters dapat dilihat pada Gambar 4.25.

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Bab ini membahas tentang implementasi dan pengujian perangkat lunak berdasarkan rancangan yang sudah dibuat. Ada dua jenis pengujian yang dilakukan, yaitu pengujian fungsional, pengujian keakuratan, dan pengujian eksperimental. Bab ini juga membahas tentang lingkungan yang digunakan untuk pengujian perangkat lunak ini.

5.1 Lingkungan untuk Pengujian

Pengujian perangkat lunak ini dilakukan di Lab Komputasi FTIS Unpar ruang 9018. Semua *file* permainan yang dipakai dalam pengujian ini dapat dilihat di bab Lampiran C.

Ada dua jenis lingkungan untuk pengujian perangkat lunak ini, yaitu:

1. Lingkungan perangkat keras.

Spesifikasi dari lingkungan ini dapat dilihat pada Tabel 5.1.

2. Lingkungan perangkat lunak.

Spesifikasi dari lingkungan ini dapat dilihat pada Tabel 5.2.

5.2 Implementasi

Hasil implementasi dari rancangan perangkat lunak yang sudah dibuat ini terdiri dari dua bagian, yaitu:

1. Kode program

Kode program untuk perangkat lunak ini ditulis dalam bahasa pemrograman Java, berdasarkan dengan rancangan diagram kelas yang sudah dibuat, seperti dapat dilihat pada sub-bab 4.4. Seluruh kode program untuk perangkat lunak ini dapat dilihat di bab Lampiran D

2. Antarmuka perangkat lunak

Antarmuka untuk perangkat lunak ini dirancang berdasarkan rancangan yang sudah dibuat, seperti dapat dilihat pada sub-bab 4.3.

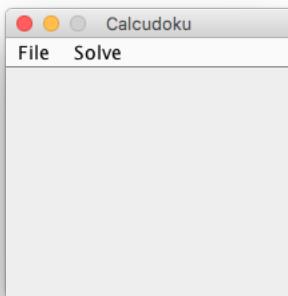
Gambar 5.1 menunjukkan antarmuka perangkat lunak saat pertama kali dibuka, sebelum *file* permainan dibuka. Gambar 5.2 menunjukkan kotak dialog untuk memilih *file* permainan yang

Tabel 5.1: Lingkungan perangkat keras untuk pengujian perangkat lunak

Parameter	Nilai
<i>Processor</i>	Intel Core i3-4130 CPU @ 3.40 GHz
<i>RAM (Random Access Memory)</i>	6.00 GB
<i>VGA (Video Graphics Array)</i>	Intel HD Graphics 4400 dan NVIDIA GeForce GT 630

Tabel 5.2: Lingkungan perangkat lunak untuk pengujian perangkat lunak

Parameter	Nilai
Sistem Operasi	Windows 10
Bahasa Pemrograman	Java
<i>IDE (Integrated Development Environment)</i>	NetBeans IDE 8.2
<i>Library Java</i>	JDK (Java Development Kit) 1.8
<i>JVM (Java Virtual Machine)</i>	Java Version 8 Update 141



Gambar 5.1: Antarmuka perangkat lunak saat pertama kali dibuka

akan dibuka. Gambar 5.3 menunjukkan antarmuka perangkat lunak setelah membuka *file* permainan yang dipilih. Gambar 5.4 menunjukkan kotak dialog untuk mengatur nilai untuk parameter-parameter algoritma genetik. Gambar 5.5 menunjukkan antarmuka perangkat lunak setelah permainan berdasarkan *file* permainan yang telah dibuka diselesaikan.

5.3 Pengujian Fungsional

Pengujian fungsional bertujuan untuk memastikan bahwa perangkat lunak dapat berfungsi sebagaimana mestinya. Skenario-skenario yang dilakukan dalam pengujian fungsional untuk perangkat lunak ini adalah:

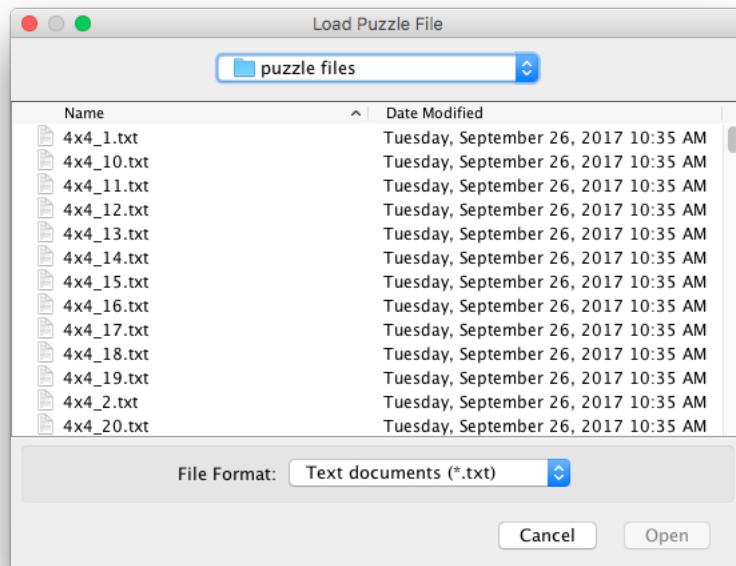
1. Memilih menu *item "Reset Puzzle"*, *"Close Puzzle File"*, dan *"Check Puzzle"* dari menu *"File"* atau menu *item "Backtracking"*, *"Hybrid Genetic"*, dan *"Set Genetic Algorithm Parameters"* dari menu *"Solve"* sebelum membuka *file* permainan.

Jika salah satu dari menu *item* tersebut dipilih sebelum membuka *file* permainan, maka pesan error *"Puzzle file not loaded"* akan muncul, seperti dapat dilihat pada Gambar 5.6.

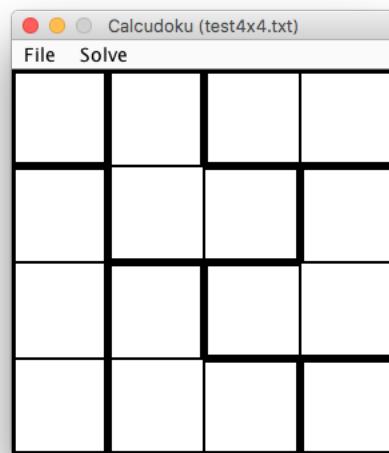
2. Membuka *file* permainan

Pemain dapat membuka *file* permainan dengan memilih menu *item "Load Puzzle File"* dalam menu *"File"*. Jika menu tersebut dipilih maka akan keluar kotak pemilihan *file* permainan, seperti dapat dilihat pada Gambar 5.7. Pilihlah sebuah *file* permainan yang ingin dibuka. Tekan tombol *"OK"* untuk membuka *file* permainan tersebut, atau tekan tombol *"Cancel"* untuk membatalkan proses membuka *file* permainan.

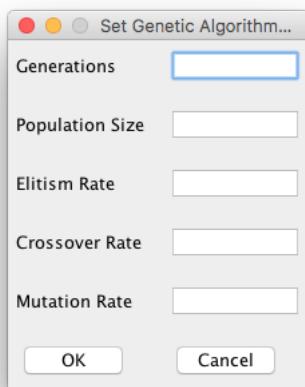
Jika perangkat lunak sudah membuka sebuah *file* permainan, maka akan keluar kotak dialog *"Are you sure you want to load another puzzle file?"*, seperti dapat dilihat pada Gambar 5.8. Tekan tombol *"Yes"* untuk membuka *file* permainan tersebut, atau tekan tombol *"No"* untuk membatalkan proses membuka *file* permainan.



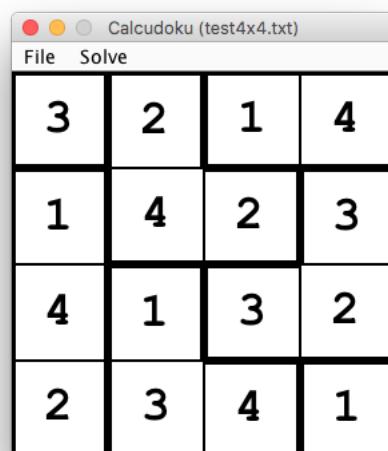
Gambar 5.2: Kotak dialog untuk memilih *file* permainan yang akan dibuka



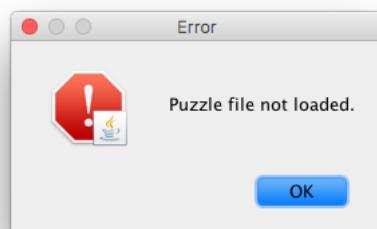
Gambar 5.3: Antarmuka perangkat lunak sesudah membuka *file* permainan yang dipilih



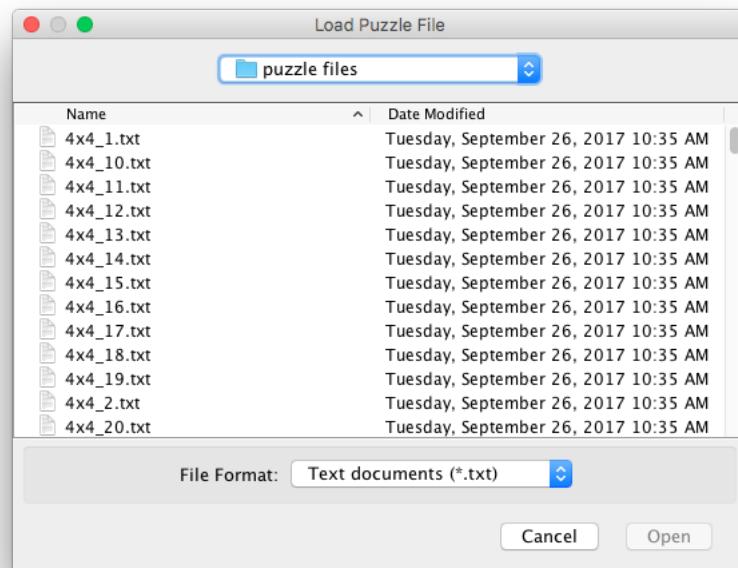
Gambar 5.4: Kotak dialog untuk mengatur nilai dari parameter-parameter algoritma genetik



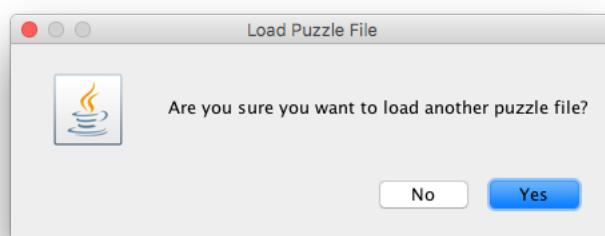
Gambar 5.5: Antarmuka perangkat lunak setelah permainan berdasarkan *file* permainan yang telah dibuka diselesaikan



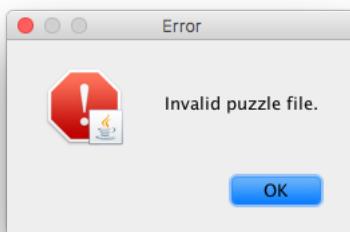
Gambar 5.6: Kotak pesan error "Puzzle file not loaded"



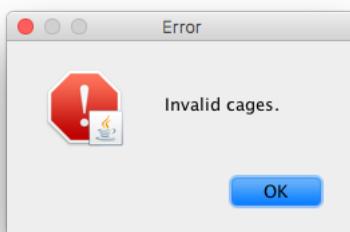
Gambar 5.7: Kotak pemilihan *file* permainan



Gambar 5.8: Kotak dialog "Are you sure you want to load another puzzle file?"



Gambar 5.9: Pesan error "*Invalid puzzle file*"



Gambar 5.10: Pesan error "*Invalid cages*"

Jika *file* permainan yang dipilih berhasil dibuka, maka perangkat lunak akan menampilkan permainan berdasarkan *file* yang dipilih, seperti dapat dilihat pada Gambar 5.3.

File permainan untuk perangkat lunak ini adalah *file* teks (*.txt). Format *file* permainan yang valid dapat dilihat di sub-bab 4.1.

Jika *file* permainan yang dibuka tidak *valid*, misalnya karena *file* permainan yang dibuka bukan *file* teks, atau jika *file* teks yang akan dibuka formatnya tidak *valid*, maka pesan error "*Invalid puzzle file*" akan muncul, seperti dapat dilihat pada Gambar 5.9.

Jika ada kesalahan dalam penentuan operator untuk *cage* yang ada di dalam *grid*, misalnya operator $-$, atau \div untuk *cage* yang tidak berukuran 2 sel, operator $+$, atau \times , untuk *cage* yang berukuran 1 sel, atau operator $=$ untuk *cage* yang berukuran lebih besar dari 1 sel, maka pesan error "*Invalid cages*" akan muncul, seperti dapat dilihat pada Gambar 5.10.

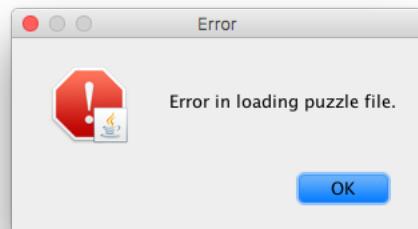
Jika salah satu dari kedua error tersebut terjadi, maka akan keluar pesan error "*Error in loading puzzle file*", seperti dapat dilihat pada Gambar 5.11.

3. Menyelesaikan permainan dengan usahanya sendiri

Jika pemain berhasil menyelesaikan permainan dengan usahanya sendiri, maka kotak informasi "*Congratulations, you have successfully solved the puzzle*" akan muncul, seperti dapat dilihat pada Gambar 5.12.

4. Me-reset permainan

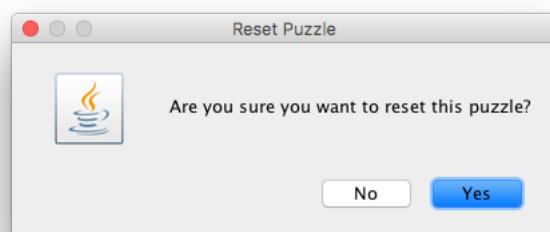
Pemain dapat me-reset permainan dengan memilih menu *item "Reset Puzzle"* dalam menu "*File*". Jika menu *item* ini dipilih, maka akan keluar kotak dialog "*Are you sure you want to reset this puzzle?*", seperti dapat dilihat pada Gambar 5.13. Tekan tombol "*Yes*" untuk me-reset permainan, atau tekan tombol "*No*" untuk membatalkan proses *reset* permainan.



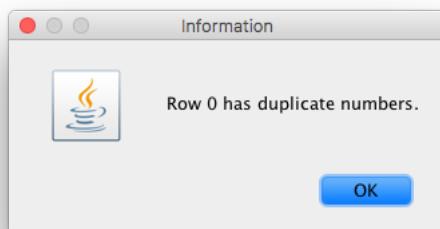
Gambar 5.11: Pesan error "*Error in loading puzzle file*"



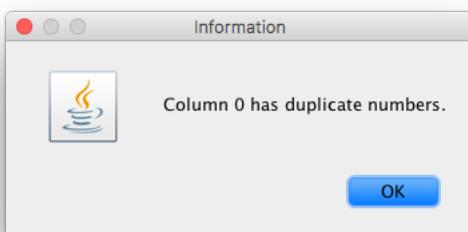
Gambar 5.12: Pesan informasi "*Congratulations, you have successfully solved the puzzle*"



Gambar 5.13: Kotak dialog "*Are you sure you want to reset this puzzle?*"



Gambar 5.14: Pesan informasi "*Row* (nomor baris) *has duplicate numbers*"



Gambar 5.15: Pesan informasi "*Column* (nomor kolom) *has duplicate numbers*"

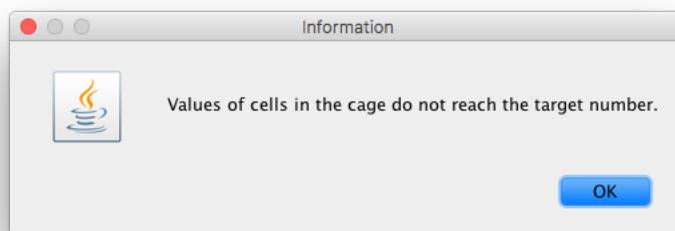
5. Memeriksa permainan jika ada nilai yang salah di dalam *grid*.

Perangkat lunak ini dapat memeriksa jika ada nilai yang salah di dalam *grid* secara otomatis.

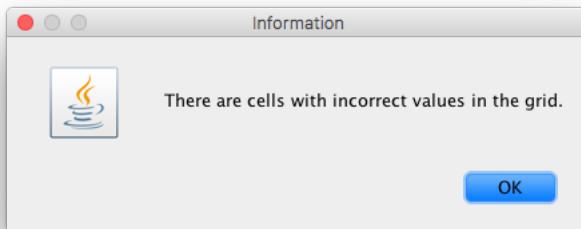
Jika ada nilai yang berulang dalam sebuah baris, maka akan keluar kotak informasi "*Row* (nomor baris) *has duplicate numbers*", seperti dapat dilihat pada Gambar 5.14.

Jika ada nilai yang berulang dalam sebuah kolom, maka akan keluar kotak informasi "*Column* (nomor kolom) *has duplicate numbers*", seperti dapat dilihat pada Gambar 5.15.

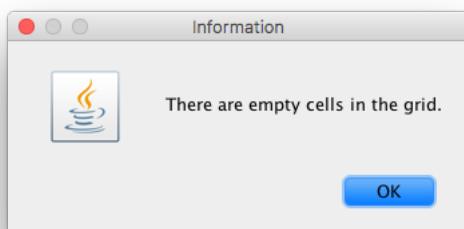
Jika nilai-nilai dalam sebuah *cage* tidak mencapai nilai tujuan yang telah ditentukan jika dihitung dengan operasi matematika yang telah ditentukan, maka akan keluar kotak informasi "*Values of cells in the cage do not reach the target number*", seperti dapat dilihat pada Gambar 5.16.



Gambar 5.16: Pesan informasi "*Values of cells in the cage do not reach the target number*"



Gambar 5.17: Pesan informasi "*There are cells with incorrect values in the grid*"



Gambar 5.18: Pesan informasi "*There are empty cells in the grid*"

Pemain juga dapat meminta perangkat lunak untuk memeriksa permainan jika ada nilai yang salah di dalam *grid* secara manual, dengan memilih menu *item "Check Puzzle"* dalam menu *"File"*.

Jika menu *item* ini dijalankan, dan ternyata ada nilai yang salah di dalam *grid*, maka akan muncul kotak informasi "*There are cells with incorrect values in the grid*", seperti dapat dilihat pada Gambar 5.17.

Jika menu *item* ini dijalankan dan ternyata ada sel yang masih kosong, maka akan muncul kotak informasi "*There are empty cells in the grid*", seperti dapat dilihat pada Gambar 5.18.

6. Menyelesaikan permainan dengan menggunakan algoritma *backtracking*

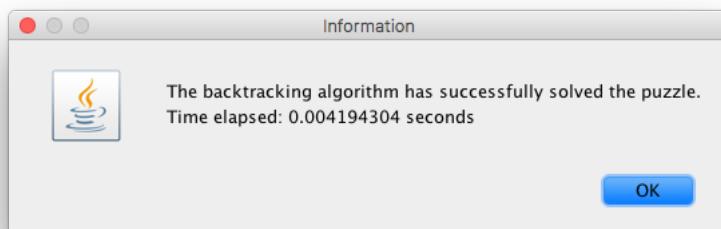
Pemain dapat meminta perangkat lunak untuk menyelesaikan permainan dengan salah satu dari dua *solver* yang disediakan. Salah satu dari kedua *solver* tersebut adalah *solver* dengan algoritma *backtracking*. Untuk menggunakan *solver* ini, pemain memilih menu *item "Backtracking"* dalam menu *"Solve"*.

Jika *solver* ini berhasil dalam menyelesaikan permainan, maka akan muncul kotak informasi "*The backtracking algorithm has successfully solved the puzzle*", dan waktu yang dibutuhkan oleh *solver* untuk menyelesaikan permainan tersebut, seperti dapat dilihat pada Gambar 5.19.

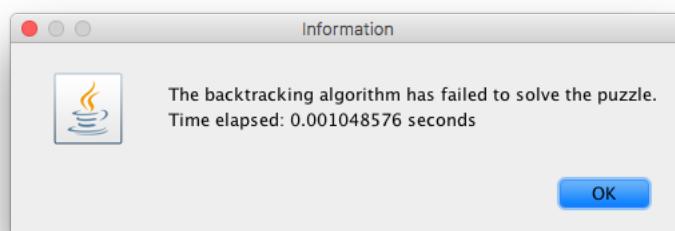
Jika gagal, maka akan muncul kotak informasi "*The backtracking algorithm has failed to solve the puzzle*", dan waktu yang dibutuhkan oleh *solver* untuk menemukan bahwa tidak ada solusi untuk permainan tersebut, seperti dapat dilihat pada Gambar 5.20

7. Menyelesaikan permainan dengan menggunakan algoritma *hybrid genetic*

Pemain dapat meminta perangkat lunak untuk menyelesaikan permainan dengan salah satu dari dua *solver* yang disediakan. Salah satu dari kedua *solver* tersebut adalah *solver* dengan



Gambar 5.19: Pesan informasi "*The backtracking algorithm has successfully solved the puzzle*"

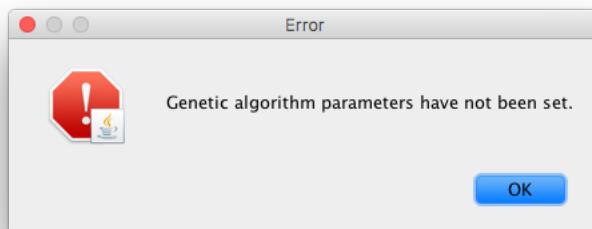


Gambar 5.20: Pesan informasi "*The backtracking algorithm has failed to solve the puzzle*"

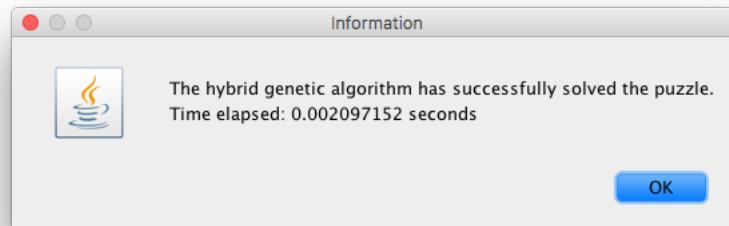
algoritma *hybrid genetic*. Untuk menggunakan *solver* ini, pemain memilih menu item "*Hybrid Genetic*" dalam menu "*Solve*". Jika nilai untuk parameter-parameter algoritma genetik belum ditentukan, maka akan keluar pesan error "*Genetic algorithm parameters have not been set*", seperti dapat dilihat pada Gambar 5.21.

Jika *solver* ini berhasil dalam menyelesaikan permainan, maka akan muncul kotak informasi "*The hybrid genetic algorithm has successfully solved the puzzle*", dan waktu yang dibutuhkan oleh *solver* untuk menyelesaikan permainan tersebut, seperti dapat dilihat pada Gambar 5.22.

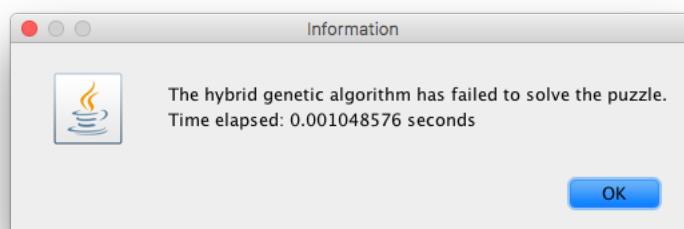
Jika gagal, maka akan muncul kotak informasi "*The hybrid genetic algorithm has failed to solve the puzzle*", dan waktu yang dibutuhkan oleh *solver* untuk menemukan bahwa tidak ada solusi untuk permainan tersebut, seperti dapat dilihat pada Gambar 5.23



Gambar 5.21: Pesan informasi "*Genetic algorithm parameters have not been set*"



Gambar 5.22: Pesan informasi "*The hybrid genetic algorithm has successfully solved the puzzle*"



Gambar 5.23: Pesan informasi "*The hybrid genetic algorithm has failed to solve the puzzle*"

8. Mengatur nilai untuk parameter-parameter algoritma genetik

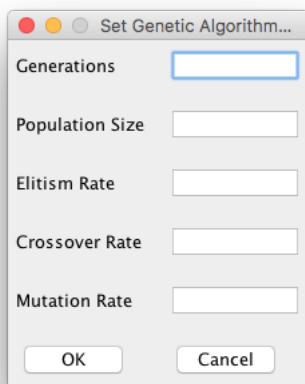
Pemain dapat mengatur sendiri nilai untuk parameter-parameter algoritma genetik dengan memilih menu item "Set Genetic Algorithm Parameters" dalam menu "Solve". Akan muncul sebuah *form* seperti dapat dilihat pada Gambar 5.24.

Isilah *form* tersebut dengan nilai yang diinginkan untuk setiap parameter algoritma genetik. Tekan tombol "OK" untuk menyimpan nilai-nilai telah diisikan ke dalam *form*, atau tekan tombol "Cancel" untuk membatalkan proses mengatur nilai untuk parameter-parameter genetik. Jika angka yang dimasukkan ke dalam *form* tidak *valid*, maka akan muncul pesan error "Invalid number format", seperti dapat dilihat pada Gambar 5.25.

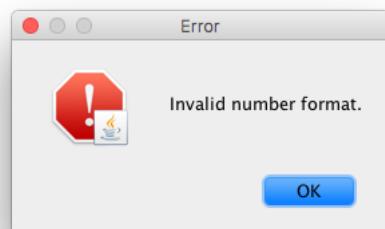
Jika jumlah dari tingkat *elitism*, tingkat mutasi, dan tingkat kawin silang kurang dari 100%, maka beberapa kromosom dari generasi sebelumnya akan diambil secara acak dan dimasukkan ke dalam generasi berikutnya sehingga jumlah kromosom dalam generasi berikutnya tetap sama. Jika jumlah dari tingkat *elitism*, tingkat mutasi, dan tingkat kawin silang lebih dari 100%, maka beberapa kromosom dari generasi berikutnya akan dibuang secara acak sehingga jumlah kromosom dalam generasi berikutnya tetap sama.

9. Menutup *file* permainan

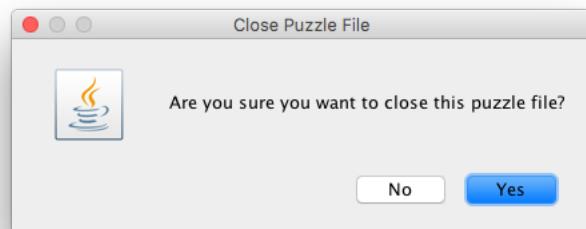
Pemain dapat menutup *file* permainan dengan memilih menu item "Close Puzzle File" dalam menu "File". Jika menu item ini dipilih, maka akan keluar kotak dialog "Are you sure you want to close this puzzle file?", seperti dapat dilihat pada Gambar 5.26. Tekan tombol "Yes" untuk menutup *file* permainan, atau tekan tombol "No" untuk membatalkan proses menutup *file* permainan.



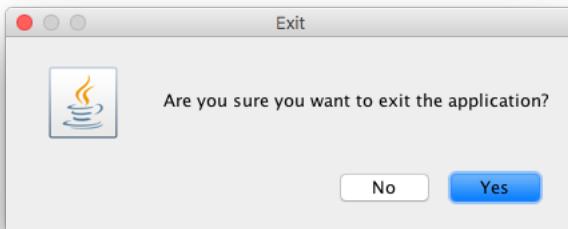
Gambar 5.24: *Form* untuk mengatur nilai untuk parameter-parameter algoritma genetik



Gambar 5.25: Pesan error "Invalid number format"



Gambar 5.26: Kotak dialog "Are you sure you want to close this puzzle file?"



Gambar 5.27: Kotak dialog "Are you sure you want to exit the application?"

```

3 5
1 2 3
1 3 3
4 5 5
3+
1=
8+
3=
3+

```

Gambar 5.28: File masukan untuk pengujian keakuratan

10. Menutup perangkat lunak

Pemain dapat menutup perangkat lunak dengan memilih menu *item* "Exit" dalam menu "File". Jika menu *item* ini dipilih, maka akan keluar kotak dialog "Are you sure you want to exit the application?", seperti dapat dilihat pada Gambar 5.27. Tekan tombol "Yes" untuk menutup perangkat lunak, atau tekan tombol "No" untuk membatalkan proses menutup perangkat lunak.

5.4 Pengujian Keakuratan

Pengujian keakuratan dilakukan untuk memastikan bahwa permainan yang ditampilkan oleh perangkat lunak sesuai dengan *file* permainan yang dibuka. Skenario-skenario yang akan dilakukan dalam pengujian keakuratan untuk perangkat lunak ini adalah:

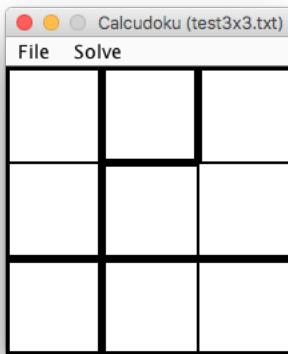
1. Keakuratan dalam menerjemahkan isi *file* masukan menjadi keluaran berupa GUI

Untuk menguji keakuratan perangkat lunak dalam menerjemahkan isi *file* masukan menjadi keluaran berupa GUI, maka perangkat lunak akan diuji dengan membuka sebuah *file* masukan, dan melihat apakah keluarannya, yaitu GUI-nya sesuai dengan *file* masukan yang dibuka.

Dalam kasus ini, perangkat lunak akan membuka *file* masukan yang isinya dapat dilihat pada Gambar 5.28.

Perangkat lunak ini lalu menerjemahkan isi *file* masukan menjadi keluaran berupa GUI yang dapat dilihat pada Gambar 5.29.

Seharusnya, petunjuk, yaitu angka tujuan dan operasi matematika yang ditentukan untuk sebuah *cage*, ditampilkan pada sudut kiri atas dari sel yang paling atas atau yang paling kiri dalam *cage* tersebut. Tetapi, karena dalam perangkat lunak ini sebuah sel adalah sebuah *text field*, dan sebuah *text field* tidak dapat diisi dengan dua teks yang berbeda (petunjuk untuk



Gambar 5.29: GUI permainan berdasarkan file masukan yang dapat dilihat pada Gambar 5.28

sel tersebut dan isi dari sel tersebut), maka petunjuk ditampilkan sebagai *tooltip* yang akan muncul saat sel-sel yang berada di dalam sebuah *cage* di-hover. Setiap sel memiliki *tooltip* yang berisi petunjuk sesuai dengan *cage* tempat sel tersebut berada.

Dalam perangkat lunak ini, setiap sel sudah memiliki *tooltip* sesuai dengan *cage* tempat sel tersebut berada.

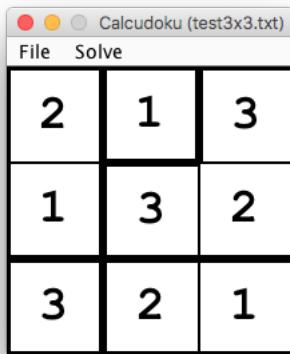
Sel-sel dalam *cage* 1, yaitu sel pada baris ke-1 dan kolom ke-1, dan sel pada baris ke-2 dan kolom ke-1, memiliki *tooltip* "3+". Sel dalam *cage* 2, yaitu sel pada baris ke-1 dan kolom ke-2, memiliki *tooltip* "1=". Sel-sel dalam *cage* 3, yaitu sel pada baris ke-1 dan kolom ke-3, sel pada baris ke-2 dan kolom ke-2, dan sel pada baris ke-2 dan kolom ke-3, memiliki *tooltip* "8+". Sel dalam *cage* 4, yaitu sel pada baris ke-3 dan kolom ke-1, memiliki *tooltip* "3=". Sel-sel dalam *cage* 5, yaitu sel pada baris ke-3 dan kolom ke-2, dan sel pada baris ke-3 dan kolom ke-3, memiliki *tooltip* "3+".

Grid dibatasi oleh garis tebal, setiap *cage* dibatasi oleh garis tebal, dua sel yang berada di dalam dua *cage* yang berbeda dibatasi oleh garis tebal, dan dua sel yang berada di dalam *cage* yang sama dibatasi oleh garis tipis.

Dalam perangkat lunak ini, *grid* dan setiap sel sudah memiliki garis pembatas yang tepat. *Grid* dibatasi oleh garis tebal.

Pada baris ke-1, sel pada kolom ke-1 dan kolom ke-2 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda, sel pada kolom ke-2 dan kolom ke-3 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda. Pada baris ke-2, sel pada kolom ke-1 dan kolom ke-2 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda, sel pada kolom ke-2 dan kolom ke-3 dibatasi oleh garis tipis karena kedua sel tersebut berada dalam dua *cage* yang sama. Pada baris ke-3, sel pada kolom ke-1 dan kolom ke-2 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda, sel pada kolom ke-2 dan kolom ke-3 dibatasi oleh garis tipis karena kedua sel tersebut berada dalam dua *cage* yang sama.

Pada kolom ke-1, sel pada baris ke-1 dan baris ke-2 dibatasi oleh garis tips karena kedua sel tersebut berada dalam dua *cage* yang sama, sel pada baris ke-2 dan baris ke-3 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda. Pada kolom ke-2, sel pada baris ke-1 dan baris ke-2 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda, sel pada baris ke-2 dan baris ke-3 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda. Pada kolom ke-3, sel pada baris ke-1 dan baris ke-2 dibatasi oleh garis tipis karena kedua sel tersebut berada dalam dua



Gambar 5.30: Solusi untuk permainan berdasarkan *file* masukan yang dapat dilihat pada Gambar 5.28

cage yang sama, sel pada baris ke-2 dan baris ke-3 dibatasi oleh garis tebal karena kedua sel tersebut berada dalam dua *cage* yang berbeda.

2. Keakuratan dalam menampilkan pesan informasi pada waktunya.

Untuk menguji keakuratan perangkat lunak dalam menampilkan pesan informasi pada waktunya, maka perangkat lunak harus diuji dengan menyelesaikan sebuah permainan, dan melihat apa kotak informasi ditampilkan pada waktunya atau tidak.

Dalam kasus ini, permainan berdasarkan *file* yang isinya ditampilkan pada Gambar 5.28 akan diselesaikan.

Cage 2, yang hanya berukuran satu sel yang terletak pada baris ke-1 dan kolom ke-2, dan memiliki petunjuk "1=". Isilah sel tersebut dengan angka 2. Karena 2 bukan angka tujuan dari *cage* tersebut, maka pesan informasi "*Values of cells in the cage do not reach the target number*" seperti yang dapat dilihat pada Gambar 5.16 akan muncul. Hapuslah angka 2 dari sel tersebut, dan isilah dengan angka 1, yang merupakan angka tujuan dari *cage* tersebut.

Isilah sel pada baris ke-1 dan kolom ke-1 dengan angka 1. Karena ada angka yang berulang dalam satu baris, maka pesan informasi "*Row has duplicate numbers*" seperti yang dapat dilihat pada Gambar 5.14 akan muncul. Hapuslah angka 1 dari sel tersebut. Sekarang isilah sel pada baris ke-1 dan kolom ke-1 dengan angka 1. Karena ada angka yang berulang dalam satu baris, maka pesan informasi "*Row has duplicate numbers*" seperti yang dapat dilihat pada Gambar 5.14 akan muncul. Hapuslah angka 1 dari sel tersebut.

Isilah sel pada baris ke-2 dan kolom ke-2 dengan angka 1. Karena ada angka yang berulang dalam satu kolom, maka pesan informasi "*Column has duplicate numbers*" seperti yang dapat dilihat pada Gambar 5.15 akan muncul. Hapuslah angka 1 dari sel tersebut.

Selesaikan permainan ini dengan mengisi sel-sel lainnya dengan angka yang benar. Solusi dari permainan ini dapat dilihat pada Gambar 5.30.

Setelah mengisi seluruh sel di dalam *grid* sesuai dengan solusi tersebut, maka pesan informasi "*Congratulations, you have successfully solved the puzzle*" seperti yang dapat dilihat pada Gambar 5.12 akan muncul.

Perangkat lunak ini telah berhasil memunculkan pesan informasi pada waktunya.

Tabel 5.3: Tabel jumlah soal permainan Calcudoku berdasarkan ukuran *grid*

Ukuran Grid	Banyak Soal
4×4	39
5×5	26
6×6	39
7×7	15
8×8	13

Tabel 5.4: Hasil pengujian algoritma *backtracking* untuk Calcudoku

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan
4×4	100%	0.067 detik
5×5	100%	0.701 detik
6×6	100%	13.84 detik
7×7	100%	482.653 detik
8×8	100%	2134.655 detik

5.5 Pengujian Algoritma

Pengujian algoritma dilakukan untuk mengetahui keberhasilan dan kecepatan algoritma dalam menyelesaikan permainan Calcudoku. Ada dua algoritma yang akan diuji, yaitu:

1. Algoritma *backtracking*
2. Algoritma *hybrid genetic*

Terdapat 132 soal yang dipakai dalam pengujian algoritma. Soal-soal tersebut memiliki ukuran *grid* antara 4×4 sampai dengan 8×8 . Jumlah soal berdasarkan ukuran *grid* dapat dilihat pada Tabel 5.3.

Rata-rata tingkat keberhasilan adalah jumlah soal yang berhasil diselesaikan oleh algoritma untuk *grid* yang berukuran $n \times n$ dibagi dengan jumlah semua soal untuk *grid* yang berukuran $n \times n$. Rata-rata kecepatan adalah jumlah dari kecepatan algoritma dalam menyelesaikan permainan untuk *grid* yang berukuran $n \times n$ yang berhasil diselesaikan oleh algoritma dibagi dengan jumlah permainan untuk *grid* yang berukuran $n \times n$ yang berhasil diselesaikan oleh algoritma.

5.5.1 Pengujian Algoritma *Backtracking*

Pengujian algoritma *backtracking* untuk Calcudoku dilakukan untuk mengetahui keberhasilan dan kecepatan algoritma *backtracking* dalam menyelesaikan permainan Calcudoku.

Berdasarkan hasil pengujian yang dapat dilihat pada Tabel 5.4, algoritma *backtracking* dapat menyelesaikan semua permainan yang diujikan. Semakin besar ukuran *grid*, maka waktu yang dibutuhkan oleh algoritma *backtracking* untuk menyelesaikan permainan semakin lama. Pada ukuran *grid* yang besar, algoritma *backtracking* sangat lambat dalam menyelesaikan permainan.

Hasil pengujian selengkapnya dapat dilihat pada Lampiran B.

5.5.2 Pengujian Algoritma *Hybrid Genetic*

Pengujian algoritma *hybrid genetic* untuk Calcudoku dilakukan untuk mengetahui keberhasilan dan kecepatan algoritma *hybrid genetic* dalam menyelesaikan permainan Calcudoku.

Pada algoritma *backtracking* tidak ada parameter yang nilainya dapat diubah, sedangkan pada algoritma *hybrid genetic*, nilai dari parameter-parameter untuk algoritma genetik dapat diubah.

Tabel 5.5: Nilai untuk parameter-parameter algoritma genetik untuk setiap percobaan yang dilakukan

Skenario	Populasi	Generasi	Elitism	Mutasi	Kawin Silang
1	1000	100	10%	10%	80%
2	1000	100	5%	10%	85%
3	1000	100	10%	5%	85%
4	1000	100	5%	5%	90%
5	100	100	10%	10%	80%
6	100	100	5%	10%	85%
7	100	100	10%	5%	85%
8	100	100	5%	5%	90%
9	1000	10	10%	10%	80%
10	1000	10	5%	10%	85%
11	1000	10	10%	5%	85%
12	1000	10	5%	5%	90%
13	100	10	10%	10%	80%
14	100	10	5%	10%	85%
15	100	10	10%	5%	85%
16	100	10	5%	5%	90%

Tabel 5.6: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 1)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4 × 4	100%	3.735 detik	3
5 × 5	42.308%	8.389 detik	9
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

Tidak ada nilai *default* untuk parameter-parameter dari algoritma genetik ini. Nilai dari parameter-parameter tersebut harus diisi sendiri oleh pemain.

Dalam kasus ini, pengujian dilakukan dengan melakukan pengujian keberhasilan dan kecepatan dari *solver* dengan algoritma *hybrid genetic* dengan mengatur nilai dari parameter-parameter dari algoritma genetik dengan nilai yang berbeda-beda untuk setiap percobaan.

Terdapat 16 skenario yang dilakukan untuk percobaan ini. Nilai untuk parameter-parameter untuk algoritma genetik untuk setiap percobaan yang dilakukan dapat dilihat pada Tabel 5.5.

Hasil pengujian dari ke-16 skenario tersebut adalah sebagai berikut:

1. Skenario 1

Hasil pengujian Skenario 1 dapat dilihat pada Tabel 5.6.

2. Skenario 2

Hasil pengujian Skenario 2 dapat dilihat pada Tabel 5.7.

3. Skenario 3

Hasil pengujian Skenario 3 dapat dilihat pada Tabel 5.8.

4. Skenario 4

Hasil pengujian Skenario 4 dapat dilihat pada Tabel 5.9.

Tabel 5.7: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 2)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	100%	4.183 detik	3
5×5	42.308%	9.258 detik	9
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.8: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 3)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	100%	3.924 detik	3
5×5	42.308%	8.806 detik	9
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.9: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 4)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	100%	4.371 detik	3
5×5	42.308%	9.676 detik	9
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.10: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 5)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	56.41%	0.48 detik	5
5×5	19.231%	0.311 detik	14
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.11: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 6)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	56.41%	0.532 detik	5
5×5	19.231%	0.339 detik	14
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

5. Skenario 5

Hasil pengujian Skenario 5 dapat dilihat pada Tabel 5.10.

6. Skenario 6

Hasil pengujian Skenario 6 dapat dilihat pada Tabel 5.11.

7. Skenario 7

Hasil pengujian Skenario 7 dapat dilihat pada Tabel 5.12.

8. Skenario 8

Hasil pengujian Skenario 8 dapat dilihat pada Tabel 5.13.

9. Skenario 9

Hasil pengujian Skenario 9 dapat dilihat pada Tabel 5.14.

10. Skenario 10

Hasil pengujian Skenario 10 dapat dilihat pada Tabel 5.15.

11. Skenario 11

Hasil pengujian Skenario 11 dapat dilihat pada Tabel 5.16.

Tabel 5.12: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 7)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	56.41%	0.505 detik	5
5×5	19.231%	0.325 detik	14
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.13: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 8)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	56.41%	0.557 detik	5
5×5	19.231%	0.352 detik	14
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.14: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 9)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	33.333%	0.457 detik	7
5×5	15.385%	0.487 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.15: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 10)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	33.333%	0.457 detik	7
5×5	15.385%	0.487 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.16: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 11)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	33.333%	0.457 detik	7
5×5	15.385%	0.487 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.17: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 12)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	33.333%	0.457 detik	7
5×5	15.385%	0.487 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.18: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 13)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	23.077%	0.048 detik	9
5×5	15.385%	0.077 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

12. Skenario 12

Hasil pengujian Skenario 12 dapat dilihat pada Tabel 5.17.

13. Skenario 13

Hasil pengujian Skenario 13 dapat dilihat pada Tabel 5.18.

14. Skenario 14

Hasil pengujian Skenario 14 dapat dilihat pada Tabel 5.19.

15. Skenario 15

Hasil pengujian Skenario 15 dapat dilihat pada Tabel 5.20.

16. Skenario 16

Hasil pengujian Skenario 16 dapat dilihat pada Tabel 5.21.

Hasil pengujian algoritma *hybrid genetic* sebanyak 16 skenario secara keseluruhan dapat dilihat pada Tabel 5.22.

Tabel 5.19: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 14)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	23.077%	0.048 detik	9
5×5	15.385%	0.077 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.20: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 15)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	23.077%	0.048 detik	9
5×5	15.385%	0.077 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.21: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 16)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	23.077%	0.048 detik	9
5×5	15.385%	0.077 detik	15
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Tabel 5.22: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku sebanyak 16 skenario secara keseluruhan

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma Rule Based
4×4	53.205%	2.119 detik	6
5×5	23.077%	4.303 detik	13
6×6	0%	-	-
7×7	0%	-	-
8×8	0%	-	-

Berdasarkan hasil pengujian yang telah dilakukan, ada beberapa hal yang dapat dilihat, yaitu:

1. Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan karena sifat acak dari algoritma *hybrid genetic* ini. Semakin besar ukuran *grid*, maka kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan semakin besar. Algoritma *hybrid genetic* ini gagal dalam menyelesaikan permainan Calcudoku dengan ukuran *grid* 6×6 ke atas.
2. Semakin besar ukuran *grid*, maka waktu yang dibutuhkan oleh algoritma *hybrid genetic* untuk menyelesaikan semakin lama. Pada ukuran *grid* yang kecil, algoritma *hybrid genetic* cenderung menyelesaikan permainan lebih lambat daripada algoritma *backtracking*. Tetapi, pada ukuran *grid* yang besar, algoritma *hybrid genetic* mungkin mampu menyelesaikan permainan lebih cepat daripada algoritma *backtracking*, tetapi hal ini tidak dapat dibuktikan karena algoritma *hybrid genetic* gagal dalam menyelesaikan permainan dengan ukuran *grid* yang besar.
3. Semakin banyak sel yang diisi dalam tahap algoritma *rule based*, semakin besar juga kemungkinan algoritma genetik untuk berhasil dalam menyelesaikan permainan dan semakin cepat algoritma genetik dalam menyelesaikan permainan.
4. Nilai untuk parameter-parameter algoritma genetik mempengaruhi kecepatan dan tingkat keberhasilan algoritma *hybrid genetic* dalam menyelesaikan permainan. Semakin besar populasi dalam sebuah generasi sampai ke titik tertentu, dan semakin banyak generasi sampai ke titik tertentu, maka semakin besar juga kemungkinan algoritma *hybrid genetic* berhasil dalam menyelesaikan permainan. Semakin besar tingkat *elitism* dan tingkat mutasi sampai ke titik tertentu, maka semakin cepat juga algoritma *hybrid genetic* dalam menyelesaikan permainan.

Hasil pengujian selengkapnya dapat dilihat pada Lampiran [B](#).

BAB 6

KESIMPULAN DAN SARAN

Bab ini membahas tentang simpulan berdasarkan hasil dari analisis, implementasi, dan pengujian perangkat lunak yang telah dibuat, dan saran-saran untuk penelitian dan pengembangan selanjutnya.

6.1 Kesimpulan

Berdasarkan hasil dari analisis, implementasi, dan pengujian perangkat lunak Calcudoku yang telah dibuat, maka dapat diambil simpulan sebagai berikut:

1. Perangkat lunak permainan teka-teki Calcudoku dengan dua *solver*, yaitu *solver* dengan algoritma *backtracking* dan *solver* dengan algoritma *hybrid genetic*, berhasil dibuat. Perangkat lunak ini menerima input berupa soal teka-teki dan mampu menyelesaikan semua soal tersebut menggunakan algoritma *backtracking*, dan sebagian soal tersebut (ukuran *grid* 4×4 dan 5×5) dengan menggunakan algoritma *hybrid genetic*.
2. Algoritma *backtracking* dapat menyelesaikan semua permainan yang diujikan. Pada ukuran *grid* yang besar, algoritma *backtracking* sangat lambat dalam menyelesaikan permainan.

Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan karena sifat acak dari algoritma *hybrid genetic* ini. Semakin besar ukuran *grid*, maka kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan semakin besar.

Pada ukuran *grid* yang kecil (4×4 dan 5×5), algoritma *hybrid genetic* cenderung menyelesaikan permainan lebih lambat daripada algoritma *backtracking*. Tetapi pada ukuran *grid* yang besar (6×6 ke atas), algoritma *hybrid genetic* gagal dalam menyelesaikan permainan, sehingga performansinya tidak bisa dibandingkan dengan performansi algoritma *backtracking*.

Semakin banyak sel yang diisi dalam tahap algoritma *rule based*, semakin besar juga kemungkinan algoritma genetik untuk berhasil dalam menyelesaikan permainan dan semakin cepat juga algoritma genetik dalam menyelesaikan permainan.

Nilai dari parameter-parameter untuk algoritma genetik mempengaruhi kecepatan dan tingkat keberhasilan algoritma *hybrid genetic* dalam menyelesaikan permainan. Semakin besar populasi dalam sebuah generasi sampai ke titik tertentu, dan semakin banyak generasi sampai ke titik tertentu, maka semakin besar juga kemungkinan algoritma *hybrid genetic* berhasil dalam menyelesaikan permainan. Semakin besar tingkat *elitism* dan tingkat mutasi sampai ke titik tertentu, maka semakin cepat juga algoritma *hybrid genetic* dalam menyelesaikan permainan.

6.2 Saran

Saran-saran yang dapat diberikan untuk mengembangkan penelitian ini adalah:

1. Memperbaiki GUI dari perangkat lunak ini agar petunjuk, yaitu angka tujuan dan operasi matematika yang ditentukan untuk sebuah *cage*, dapat ditampilkan sebagaimana mestinya, yaitu pada di sudut kiri atas sel yang paling atas dan yang paling kiri dalam *cage* tersebut.

2. Menambah aturan-aturan logika untuk algoritma *rule based*, misalnya aturan *naked subset* untuk *cage* yang berukuran lebih besar dari 3 sel, aturan *hidden subset* untuk *cage* yang berukuran lebih besar dari 2 sel, aturan *killer combination* untuk *cage* yang berukuran lebih besar dari 2 sel, dan aturan *evil twin* untuk *cage* yang berukuran minimal 2 sel. Dengan menambah aturan-aturan logika untuk algoritma *rule based*. Diharapkan, dengan menambah aturan-aturan logika untuk algoritma *rule based*, maka tingkat kesuksesan algoritma *hybrid genetic* dalam menyelesaikan permainan Calcudoku dapat meningkat.
3. Memperbaiki algoritma genetik, misalnya proses pemberian nilai kelayakan untuk kromosom, proses pemilihan kromosom untuk kawin silang dan mutasi, proses *elitism*, proses kawin silang, dan proses mutasi, sehingga tingkat kesuksesan algoritma *hybrid genetic* dalam menyelesaikan permainan Calcudoku dapat meningkat.

DAFTAR REFERENSI

- [1] Fahda, A. (2015) Kenken puzzle solver using backtracking algorithm. Makalah IF2211 Strategi Algoritma - Semester II Tahun 2014/2015, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_016.pdf.
- [2] Olivia Johanna, K. V. I. S., Samuel Lukas (2012) Solving and modeling ken-ken puzzle by using hybrid genetics algorithm. *1st International Conference on Engineering and Technology Development (ICETD 2012)*, Bandar Lampung, Lampung, Indonesia, 20-21 Juni, pp. 98–102. Faculty of Engineering and Faculty of Computer Science, Bandar Lampung University.

LAMPIRAN A

ANALISIS ALGORITMA BACKTRACKING

Dalam lampiran ini dijelaskan tentang analisis algoritma *backtracking* seperti dijelaskan pada sub-bab 3.1 secara lengkap.

Untuk mengilustrasikan cara kerja algoritma *backtracking*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar A.1 sebagai contoh. *State space tree* seperti yang dijelaskan pada Bab 2.2 akan dibangkitkan.

1. Algoritma *backtracking* dimulai dengan teka-teki yang belum diselesaikan, seperti yang digambarkan pada Gambar A.1 (*state 1*).
2. Algoritma mengisikan sel pada baris ke-1 dan kolom ke-1 dengan angka 1 (*state 2*), tetapi angka 1 tidak sesuai dengan angka tujuan dari *cage* tersebut.
3. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 3*), tetapi angka 2 juga tidak sesuai dengan angka tujuan dari *cage* tersebut.
4. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 4*), seperti dapat dilihat pada Gambar A.2, dan ternyata angka 3 sesuai dengan angka tujuan dari *cage* tersebut, sehingga algoritma dapat maju ke sel berikutnya.
5. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (*state 5*). Algoritma lalu maju ke sel berikutnya.
6. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state 6*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
7. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 7*). Algoritma lalu maju ke sel berikutnya.
8. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state 8*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.

3	8+	3-	
7+			
	8+	8+	
			1

Gambar A.1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan, seperti yang digambarkan pada Gambar 2.1. [1]

³ 3	⁸⁺	³⁻	
⁷⁺			
	⁸⁺	⁸⁺	
			1

Gambar A.2: *State 4*

³ 3	⁸⁺ 1	³⁻ 2	4
⁷⁺			
	⁸⁺	⁸⁺	
			1

Gambar A.3: *State 11*

9. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 9*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
10. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 10*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
11. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 11*), seperti dapat dilihat pada Gambar A.3, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.
12. Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 7*). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 12*), seperti dapat dilihat pada Gambar A.4, tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
13. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 13*). Algoritma lalu maju ke sel berikutnya.
14. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state 14*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
15. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 15*), tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.

³ 3	⁸⁺ 1	³⁻ 3	
⁷⁺			
	⁸⁺	⁸⁺	
			1

Gambar A.4: *State 12*

³ 3	⁸⁺ 1	³⁻ 4	4
7+			
	⁸⁺	⁸⁺	
			1

Gambar A.5: *State 17*

³ 3	⁸⁺ 2	³⁻	
7+			
	⁸⁺	⁸⁺	
			1

Gambar A.6: *State 18*

16. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 16*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
17. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 17*), seperti dapat dilihat pada Gambar A.5, tetapi angka 4 sudah pernah digunakan dalam baris tersebut.
18. Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-3 dan ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 5*). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 18*), seperti dapat dilihat pada Gambar A.6. Algoritma lalu maju ke sel berikutnya.
19. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state 19*), seperti dapat dilihat pada Gambar A.7. Algoritma lalu maju ke sel berikutnya.
20. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state 20*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
21. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 21*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
22. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 22*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

³ 3	⁸⁺ 2	³⁻ 1	
7+			
	⁸⁺	⁸⁺	
			1

Gambar A.7: *State 19*

3	8+	3-	4
7+			
	8+	8+	
			1

Gambar A.8: *State 23*

3	8+	3-	4
7+ 1			
	8+	8+	
			1

Gambar A.9: *State 24*

23. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 23*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar A.8. Algoritma telah selesai mengisikan baris ke-1, sehingga bisa maju ke baris berikutnya.
24. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-1 dengan angka 1 (*state 24*), seperti dapat dilihat pada Gambar A.9. Algoritma lalu maju ke sel berikutnya.
25. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-2 dengan angka 1 (*state 25*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
26. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 26*), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
27. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 27*). Algoritma lalu maju ke sel berikutnya.
28. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-3 dengan angka 1 (*state 28*), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
29. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 29*), tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.
30. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 30*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
31. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 31*), seperti dapat dilihat pada Gambar A.10, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.
32. Karena semua kemungkinan angka untuk baris ke-2 dan kolom ke-3 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 27*). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 32*), seperti dapat dilihat pada Gambar A.11. Algoritma lalu maju ke sel berikutnya.

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	3	4	
	⁸⁺	⁸⁺	
			1

Gambar A.10: *State 31*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4		
	⁸⁺	⁸⁺	
			1

Gambar A.11: *State 32*

33. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-3 dengan angka 1 (*state 33*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
34. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 34*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar A.12. Algoritma lalu maju ke sel berikutnya.
35. Algoritma lalu mengisikan sel pada baris ke-2 dan kolom ke-4 dengan angka 1 (*state 35*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
36. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 36*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
37. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 37*), seperti dapat dilihat pada Gambar A.13. Algoritma telah selesai mengisikan baris ke-2, sehingga bisa maju ke baris berikutnya.
38. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-1 dengan angka 1 (*state 38*), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
39. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 39*). Algoritma lalu maju ke sel berikutnya.

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	
	⁸⁺	⁸⁺	
			1

Gambar A.12: *State 34*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
	⁸⁺	⁸⁺	
			1

Gambar A.13: *State 37*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 1	⁸⁺ 3	4
			1

Gambar A.14: *State 47*

40. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-2 dengan angka 1 (*state 40*). Algoritma lalu maju ke sel berikutnya.
41. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state 41*), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
42. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 42*), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.
43. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 43*). Algoritma lalu maju ke sel berikutnya.
44. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-4 dengan angka 1 (*state 44*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
45. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 45*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
46. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 46*), tetapi angka 3 sudah pernah digunakan dalam baris dan kolom tersebut.
47. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 47*), seperti dapat dilihat pada Gambar A.14, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.
48. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 43*). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 48*), seperti dapat dilihat pada Gambar A.15. Algoritma lalu maju ke sel berikutnya.
49. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-4 dengan angka 1 (*state 49*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
50. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 50*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 1	⁸⁺ 4	
			1

Gambar A.15: *State 48*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 1	⁸⁺ 4	4
			1

Gambar A.16: *State 52*

51. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 51*), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.
52. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 52*), seperti dapat dilihat pada Gambar A.16, tetapi angka 3 sudah pernah digunakan dalam baris dan kolom tersebut.
53. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 48*). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 53*) seperti dapat dilihat pada Gambar A.17, tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.
54. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 54*). Algoritma lalu maju ke sel berikutnya.
55. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state 55*), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
56. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 56*), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.
57. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 57*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

3	2	1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 2	⁸⁺	
			1

Gambar A.17: *State 53*

3	8+	3-	4
7+ 1	4	2	3
2	8+	8+	1
4	1	4	1

Gambar A.18: *State 68*

58. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 58*). Algoritma lalu maju ke sel berikutnya.
59. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-4 dengan angka 1 (*state 59*). Algoritma telah selesai mengisikan baris ke-2, sehingga bisa maju ke baris berikutnya.
60. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-1 dengan angka 1 (*state 60*), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
61. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 61*), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
62. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 62*), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.
63. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 63*). Algoritma lalu maju ke sel berikutnya.
64. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-2 dengan angka 1 (*state 64*). Algoritma lalu maju ke sel berikutnya.
65. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-3 dengan angka 1 (*state 65*), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
66. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 66*), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
67. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 67*), tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.
68. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 68*), seperti dapat dilihat di Gambar A.18. tetapi angka 4 sudah pernah digunakan dalam baris dan kolom tersebut.
69. Karena semua kemungkinan angka untuk baris ke-4 dan kolom ke-3 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 64*). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 69*), seperti dapat dilihat pada Gambar A.19, tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
70. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 70*), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.
71. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 71*), seperti dapat dilihat di Gambar A.20, tetapi angka 4 sudah pernah digunakan dalam kolom tersebut.

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 3	⁸⁺ 4	1
4	2		¹

Gambar A.19: *State 69*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 3	⁸⁺ 4	1
4	4		¹

Gambar A.20: *State 71*

72. Karena semua kemungkinan angka untuk baris ke-4 dan kolom ke-1 dan ke-2 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 59*). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 72*), seperti dapat dilihat pada Gambar A.21, tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
73. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 73*), tetapi angka 3 sudah pernah digunakan dalam baris dan kolom tersebut.
74. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 74*), seperti dapat dilihat di Gambar A.22, tetapi angka 4 sudah pernah digunakan dalam baris dan kolom tersebut.
75. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-3 dan ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 54*). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 75*), seperti dapat dilihat pada Gambar A.23, tetapi angka 4 sudah pernah digunakan dalam kolom tersebut.
76. Karena semua kemungkinan angka untuk baris ke-3 dan kolom ke-2 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 54*). Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 76*), seperti dapat dilihat pada Gambar A.24, tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 3	⁸⁺ 4	2
			¹

Gambar A.21: *State 72*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 3	⁸⁺ 4	4
			1

Gambar A.22: *State 74*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 4	⁸⁺	
			1

Gambar A.23: *State 75*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
3	⁸⁺	⁸⁺	
			1

Gambar A.24: *State 76*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺	⁸⁺	
			1

Gambar A.25: *State 77*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺ 1	⁸⁺	
			1

Gambar A.26: *State 78*

77. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 77*), seperti dapat dilihat di Gambar A.25. Algoritma lalu maju ke sel berikutnya.
78. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-2 dengan angka 1 (*state 78*), seperti dapat dilihat di Gambar A.26. Algoritma lalu maju ke sel berikutnya.
79. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state 79*), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
80. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 80*), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
81. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 81*), seperti dapat dilihat pada Gambar A.27. Algoritma lalu maju ke sel berikutnya.
82. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state 82*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
83. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 83*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar A.28. Algoritma telah selesai mengisikan baris ke-3, sehingga bisa maju ke baris berikutnya.

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺ 1	⁸⁺ 3	
			1

Gambar A.27: *State 81*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺ 1	⁸⁺ 3	2
			1

Gambar A.28: *State 83*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺ 1	⁸⁺ 3	2
2			1

Gambar A.29: *State 85*

84. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-1 dengan angka 1 (*state 84*), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
85. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 85*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar A.29. Algoritma lalu maju ke sel berikutnya.
86. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-2 dengan angka 1 (*state 86*), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
87. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 87*), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.
88. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 88*), seperti dapat dilihat pada Gambar A.30. Algoritma lalu maju ke sel berikutnya.
89. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-3 dengan angka 1 (*state 89*), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
90. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 90*), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺ 1	⁸⁺ 3	2
2	3		1

Gambar A.30: *State 88*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺ 1	⁸⁺ 3	2
2	3	4	¹ 1

Gambar A.31: *State 92*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4	2	3
4	⁸⁺ 1	⁸⁺ 3	2
2	3	4	¹ 1

Gambar A.32: *State 93*

91. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 91*), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.
92. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 92*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar A.31. Algoritma lalu maju ke sel berikutnya.
93. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-4 dengan angka 1 (*state 93*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar A.32. Algoritma *backtracking* telah selesai mengisi semua sel dalam permainan teka-teki Calcudoku ini dengan benar.

LAMPIRAN B

HASIL PENGUJIAN

Lampiran ini berisi hasil pengujian yang telah dilakukan.

B.1 Algoritma *Backtracking*

Berikut adalah hasil pengujian algoritma *backtracking* untuk Calcudoku.

Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 4 \times 4$ dapat dilihat pada Tabel B.1.

Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 5 \times 5$ dapat dilihat pada Tabel B.2.

Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 6 \times 6$ dapat dilihat pada Tabel B.3.

Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 7 \times 7$ dapat dilihat pada Tabel B.4.

Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 8 \times 8$ dapat dilihat pada Tabel B.5.

Tabel B.1: Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 4 \times 4$

Nomor Soal	Kecepatan
1	0.104 detik
2	0.008 detik
3	0.134 detik
4	0.082 detik
5	0.092 detik
6	0.008 detik
7	0.074 detik
8	0.185 detik
9	0.095 detik
10	0.106 detik
11	0.048 detik
12	0.122 detik
13	0.036 detik
14	0.043 detik
15	0.058 detik
16	0.013 detik
17	0.117 detik
18	0.076 detik
19	0.079 detik
20	0.043 detik
21	0.039 detik
22	0.055 detik
23	0.063 detik
24	0.048 detik
25	0.11 detik
26	0.036 detik
27	0.083 detik
28	0.059 detik
29	0.079 detik
30	0.058 detik
31	0.167 detik
32	0.046 detik
33	0.058 detik
34	0.021 detik
35	0.069 detik
36	0.024 detik
37	0.026 detik
38	0.017 detik
39	0.036 detik

Tabel B.2: Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran *grid* 5×5

Nomor Soal	Kecepatan
1	0.257 detik
2	1.836 detik
3	0.958 detik
4	0.068 detik
5	0.816 detik
6	0.426 detik
7	1.17 detik
8	0.931 detik
9	1.017 detik
10	0.184 detik
11	0.716 detik
12	0.524 detik
13	0.15 detik
14	0.494 detik
15	0.438 detik
16	3.224 detik
17	0.276 detik
18	0.627 detik
19	1.755 detik
20	0.264 detik
21	0.446 detik
22	0.326 detik
23	0.092 detik
24	0.944 detik
25	0.137 detik
26	0.144 detik

Tabel B.3: Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 6 \times 6$

Nomor Soal	Kecepatan
1	6.315 detik
2	3.072 detik
3	2.306 detik
4	1.232 detik
5	0.775 detik
6	4.233 detik
7	2.498 detik
8	0.592 detik
9	5.529 detik
10	2.498 detik
11	0.62 detik
12	3.768 detik
13	22.784 detik
14	19.724 detik
15	0.866 detik
16	5.21 detik
17	2.327 detik
18	2.958 detik
19	5.97 detik
20	6.457 detik
21	4.011 detik
22	3.128 detik
23	243.767 detik
24	0.988 detik
25	0.172 detik
26	3.628 detik
27	8.873 detik
28	5.596 detik
29	1.1 detik
30	4.112 detik
31	1.328 detik
32	2.172 detik
33	5.381 detik
34	1.018 detik
35	72.546 detik
36	14.35 detik
37	14.662 detik
38	2.638 detik
39	50.561 detik

Tabel B.4: Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 7 \times 7$

Nomor Soal	Kecepatan
1	5924.967 detik
2	24.596 detik
3	40.597 detik
4	26.073 detik
5	75.227 detik
6	29.977 detik
7	338.317 detik
8	43.976 detik
9	109.051 detik
10	48.554 detik
11	43.503 detik
12	8.538 detik
13	1990.996 detik
14	64.485 detik
15	270.934 detik

Tabel B.5: Hasil pengujian algoritma *backtracking* untuk Calcudoku dengan ukuran $grid\ 8 \times 8$

Nomor Soal	Kecepatan
1	1417.117 detik
2	4249.97 detik
3	2699.821 detik
4	180.779 detik
5	563.79 detik
6	6068.212 detik
7	1923.112 detik
8	727.159 detik
9	2817.854 detik
10	65.25 detik
11	4800.963 detik
12	1691.002 detik
13	545.492 detik

B.2 Algoritma *Hybrid Genetic*

Berikut adalah hasil pengujian algoritma *hybrid genetic* untuk Calcudoku.

Pada semua skenario, algoritma *hybrid genetic* gagal dalam menyelesaikan permainan dengan ukuran *grid* 6×6 ke atas.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 4×4 untuk Skenario 1 sampai dengan Skenario 4 dapat dilihat pada Tabel B.6.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 4×4 untuk Skenario 5 sampai dengan Skenario 8 dapat dilihat pada Tabel B.7.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 4×4 untuk Skenario 9 sampai dengan Skenario 12 dapat dilihat pada Tabel B.8.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 4×4 untuk Skenario 13 sampai dengan Skenario 16 dapat dilihat pada Tabel B.9.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 5×5 untuk Skenario 1 sampai dengan Skenario 4 dapat dilihat pada Tabel B.10.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 5×5 untuk Skenario 5 sampai dengan Skenario 8 dapat dilihat pada Tabel B.11.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 5×5 untuk Skenario 9 sampai dengan Skenario 12 dapat dilihat pada Tabel B.12.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 5×5 untuk Skenario 13 sampai dengan Skenario 16 dapat dilihat pada Tabel B.13.

Daftar jumlah sel yang berhasil diisi oleh algoritma *rule based* untuk Calcudoku dengan ukuran *grid* 4×4 dapat dilihat pada Tabel B.14.

Daftar jumlah sel yang berhasil diisi oleh algoritma *rule based* untuk Calcudoku dengan ukuran *grid* 5×5 dapat dilihat pada Tabel ??.

Tabel B.6: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran $grid 4 \times 4$ (Skenario 1-4)

Nomor Soal	Kecepatan Skenario 1	Kecepatan Skenario 2	Kecepatan Skenario 3	Kecepatan Skenario 4
1	9.801 detik	11.025 detik	10.413 detik	11.637 detik
2	2.457 detik	2.763 detik	2.457 detik	2.763 detik
3	9.801 detik	11.025 detik	10.413 detik	11.637 detik
4	4.905 detik	5.517 detik	5.211 detik	5.823 detik
5	0.314 detik	0.314 detik	0.314 detik	0.314 detik
6	2.457 detik	2.763 detik	2.457 detik	2.763 detik
7	4.905 detik	5.517 detik	5.211 detik	5.823 detik
8	9.801 detik	11.025 detik	10.413 detik	11.637 detik
9	0.314 detik	0.314 detik	0.314 detik	0.314 detik
10	4.905 detik	5.517 detik	5.211 detik	5.823 detik
11	4.905 detik	5.517 detik	5.211 detik	5.823 detik
12	2.457 detik	2.763 detik	2.457 detik	2.763 detik
13	2.457 detik	2.763 detik	2.457 detik	2.763 detik
14	0.314 detik	0.314 detik	0.314 detik	0.314 detik
15	9.801 detik	11.025 detik	10.413 detik	11.637 detik
16	0.62 detik	0.62 detik	0.62 detik	0.62 detik
17	1.232 detik	1.232 detik	1.232 detik	1.232 detik
18	0.62 detik	0.62 detik	0.62 detik	0.62 detik
19	4.905 detik	5.517 detik	5.211 detik	5.823 detik
20	4.905 detik	5.517 detik	5.211 detik	5.823 detik
21	2.457 detik	2.763 detik	2.457 detik	2.763 detik
22	0.314 detik	0.314 detik	0.314 detik	0.314 detik
23	9.801 detik	11.025 detik	10.413 detik	11.637 detik
24	2.457 detik	2.763 detik	2.457 detik	2.763 detik
25	0.008 detik	0.008 detik	0.008 detik	0.008 detik
26	2.457 detik	2.763 detik	2.457 detik	2.763 detik
27	4.905 detik	5.517 detik	5.211 detik	5.823 detik
28	0.314 detik	0.314 detik	0.314 detik	0.314 detik
29	2.457 detik	2.763 detik	2.457 detik	2.763 detik
30	4.905 detik	5.517 detik	5.211 detik	5.823 detik
31	9.801 detik	11.025 detik	10.413 detik	11.637 detik
32	0.008 detik	0.008 detik	0.008 detik	0.008 detik
33	9.801 detik	11.025 detik	10.413 detik	11.637 detik
34	0.314 detik	0.314 detik	0.314 detik	0.314 detik
35	1.232 detik	1.232 detik	1.232 detik	1.232 detik
36	4.905 detik	5.517 detik	5.211 detik	5.823 detik
37	2.457 detik	2.763 detik	2.457 detik	2.763 detik
38	0.314 detik	0.314 detik	0.314 detik	0.314 detik
39	4.905 detik	5.517 detik	5.211 detik	5.823 detik

Tabel B.7: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 4×4
(Skenario 5-8)

Nomor Soal	Kecepatan Skenario 5	Kecepatan Skenario 6	Kecepatan Skenario 7	Kecepatan Skenario 8
1	Gagal	Gagal	Gagal	Gagal
2	0.999 detik	1.109 detik	1.054 detik	1.164 detik
3	Gagal	Gagal	Gagal	Gagal
4	Gagal	Gagal	Gagal	Gagal
5	0.063 detik	0.063 detik	0.063 detik	0.063 detik
6	0.999 detik	1.109 detik	1.054 detik	1.164 detik
7	Gagal	Gagal	Gagal	Gagal
8	Gagal	Gagal	Gagal	Gagal
9	0.036 detik	0.036 detik	0.036 detik	0.036 detik
10	Gagal	Gagal	Gagal	Gagal
11	Gagal	Gagal	Gagal	Gagal
12	0.999 detik	1.109 detik	1.054 detik	1.164 detik
13	0.999 detik	1.109 detik	1.054 detik	1.164 detik
14	0.036 detik	0.036 detik	0.036 detik	0.036 detik
15	Gagal	Gagal	Gagal	Gagal
16	0.229 detik	0.256 detik	0.229 detik	0.256 detik
17	0.339 detik	0.394 detik	0.366 detik	0.421 detik
18	0.229 detik	0.256 detik	0.229 detik	0.256 detik
19	Gagal	Gagal	Gagal	Gagal
20	Gagal	Gagal	Gagal	Gagal
21	0.999 detik	1.109 detik	1.054 detik	1.164 detik
22	0.063 detik	0.063 detik	0.063 detik	0.063 detik
23	Gagal	Gagal	Gagal	Gagal
24	0.999 detik	1.109 detik	1.054 detik	1.164 detik
25	0.008 detik	0.008 detik	0.008 detik	0.008 detik
26	0.999 detik	1.109 detik	1.054 detik	1.164 detik
27	Gagal	Gagal	Gagal	Gagal
28	0.063 detik	0.063 detik	0.063 detik	0.063 detik
29	0.999 detik	1.109 detik	1.054 detik	1.164 detik
30	Gagal	Gagal	Gagal	Gagal
31	Gagal	Gagal	Gagal	Gagal
32	0.008 detik	0.008 detik	0.008 detik	0.008 detik
33	Gagal	Gagal	Gagal	Gagal
34	0.036 detik	0.036 detik	0.036 detik	0.036 detik
35	0.339 detik	0.394 detik	0.366 detik	0.421 detik
36	Gagal	Gagal	Gagal	Gagal
37	0.999 detik	1.109 detik	1.054 detik	1.164 detik
38	0.118 detik	0.118 detik	0.118 detik	0.118 detik
39	Gagal	Gagal	Gagal	Gagal

Tabel B.8: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 4×4 (Skenario 9-12)

Nomor Soal	Kecepatan Skenario 9	Kecepatan Skenario 10	Kecepatan Skenario 11	Kecepatan Skenario 12
1	Gagal	Gagal	Gagal	Gagal
2	Gagal	Gagal	Gagal	Gagal
3	Gagal	Gagal	Gagal	Gagal
4	Gagal	Gagal	Gagal	Gagal
5	0.314 detik	0.314 detik	0.314 detik	0.314 detik
6	Gagal	Gagal	Gagal	Gagal
7	Gagal	Gagal	Gagal	Gagal
8	Gagal	Gagal	Gagal	Gagal
9	0.314 detik	0.314 detik	0.314 detik	0.314 detik
10	Gagal	Gagal	Gagal	Gagal
11	Gagal	Gagal	Gagal	Gagal
12	Gagal	Gagal	Gagal	Gagal
13	Gagal	Gagal	Gagal	Gagal
14	0.314 detik	0.314 detik	0.314 detik	0.314 detik
15	Gagal	Gagal	Gagal	Gagal
16	0.62 detik	0.62 detik	0.62 detik	0.62 detik
17	1.232 detik	1.232 detik	1.232 detik	1.232 detik
18	0.62 detik	0.62 detik	0.62 detik	0.62 detik
19	Gagal	Gagal	Gagal	Gagal
20	Gagal	Gagal	Gagal	Gagal
21	Gagal	Gagal	Gagal	Gagal
22	0.314 detik	0.314 detik	0.314 detik	0.314 detik
23	Gagal	Gagal	Gagal	Gagal
24	Gagal	Gagal	Gagal	Gagal
25	0.008 detik	0.008 detik	0.008 detik	0.008 detik
26	Gagal	Gagal	Gagal	Gagal
27	Gagal	Gagal	Gagal	Gagal
28	0.314 detik	0.314 detik	0.314 detik	0.314 detik
29	Gagal	Gagal	Gagal	Gagal
30	Gagal	Gagal	Gagal	Gagal
31	Gagal	Gagal	Gagal	Gagal
32	0.008 detik	0.008 detik	0.008 detik	0.008 detik
33	Gagal	Gagal	Gagal	Gagal
34	0.314 detik	0.314 detik	0.314 detik	0.314 detik
35	1.232 detik	1.232 detik	1.232 detik	1.232 detik
36	Gagal	Gagal	Gagal	Gagal
37	Gagal	Gagal	Gagal	Gagal
38	0.314 detik	0.314 detik	0.314 detik	0.314 detik
39	Gagal	Gagal	Gagal	Gagal

Tabel B.9: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid* 4×4 (Skenario 13-16)

Nomor Soal	Kecepatan Skenario 13	Kecepatan Skenario 14	Kecepatan Skenario 15	Kecepatan Skenario 16
1	Gagal	Gagal	Gagal	Gagal
2	Gagal	Gagal	Gagal	Gagal
3	Gagal	Gagal	Gagal	Gagal
4	Gagal	Gagal	Gagal	Gagal
5	0.063 detik	0.063 detik	0.063 detik	0.063 detik
6	Gagal	Gagal	Gagal	Gagal
7	Gagal	Gagal	Gagal	Gagal
8	Gagal	Gagal	Gagal	Gagal
9	0.036 detik	0.036 detik	0.036 detik	0.036 detik
10	Gagal	Gagal	Gagal	Gagal
11	Gagal	Gagal	Gagal	Gagal
12	Gagal	Gagal	Gagal	Gagal
13	Gagal	Gagal	Gagal	Gagal
14	0.036 detik	0.036 detik	0.036 detik	0.036 detik
15	Gagal	Gagal	Gagal	Gagal
16	Gagal	Gagal	Gagal	Gagal
17	Gagal	Gagal	Gagal	Gagal
18	Gagal	Gagal	Gagal	Gagal
19	Gagal	Gagal	Gagal	Gagal
20	Gagal	Gagal	Gagal	Gagal
21	Gagal	Gagal	Gagal	Gagal
22	0.063 detik	0.063 detik	0.063 detik	0.063 detik
23	Gagal	Gagal	Gagal	Gagal
24	Gagal	Gagal	Gagal	Gagal
25	0.008 detik	0.008 detik	0.008 detik	0.008 detik
26	Gagal	Gagal	Gagal	Gagal
27	Gagal	Gagal	Gagal	Gagal
28	0.063 detik	0.063 detik	0.063 detik	0.063 detik
29	Gagal	Gagal	Gagal	Gagal
30	Gagal	Gagal	Gagal	Gagal
31	Gagal	Gagal	Gagal	Gagal
32	0.008 detik	0.008 detik	0.008 detik	0.008 detik
33	Gagal	Gagal	Gagal	Gagal
34	0.036 detik	0.036 detik	0.036 detik	0.036 detik
35	Gagal	Gagal	Gagal	Gagal
36	Gagal	Gagal	Gagal	Gagal
37	Gagal	Gagal	Gagal	Gagal
38	0.118 detik	0.118 detik	0.118 detik	0.118 detik
39	Gagal	Gagal	Gagal	Gagal

Tabel B.10: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran $grid 5 \times 5$ (Skenario 1-4)

Nomor Soal	Kecepatan Skenario 1	Kecepatan Skenario 2	Kecepatan Skenario 3	Kecepatan Skenario 4
1	18.369 detik	19.899 detik	19.134 detik	20.664 detik
2	Gagal	Gagal	Gagal	Gagal
3	0.391 detik	0.391 detik	0.391 detik	0.391 detik
4	Gagal	Gagal	Gagal	Gagal
5	Gagal	Gagal	Gagal	Gagal
6	Gagal	Gagal	Gagal	Gagal
7	12.249 detik	13.779 detik	13.014 detik	14.544 detik
8	Gagal	Gagal	Gagal	Gagal
9	12.249 detik	13.779 detik	13.014 detik	14.544 detik
10	Gagal	Gagal	Gagal	Gagal
11	Gagal	Gagal	Gagal	Gagal
12	Gagal	Gagal	Gagal	Gagal
13	0.391 detik	0.391 detik	0.391 detik	0.391 detik
14	Gagal	Gagal	Gagal	Gagal
15	Gagal	Gagal	Gagal	Gagal
16	Gagal	Gagal	Gagal	Gagal
17	Gagal	Gagal	Gagal	Gagal
18	0.391 detik	0.391 detik	0.391 detik	0.391 detik
19	18.369 detik	19.899 detik	19.134 detik	20.664 detik
20	Gagal	Gagal	Gagal	Gagal
21	4.599 detik	4.981 detik	4.599 detik	4.981 detik
22	0.773 detik	0.773 detik	0.773 detik	0.773 detik
23	Gagal	Gagal	Gagal	Gagal
24	12.249 detik	13.779 detik	13.014 detik	14.544 detik
25	12.249 detik	13.779 detik	13.014 detik	14.544 detik
26	Gagal	Gagal	Gagal	Gagal

Tabel B.11: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran $grid 5 \times 5$ (Skenario 5-8)

Nomor Soal	Kecepatan Skenario 5	Kecepatan Skenario 6	Kecepatan Skenario 7	Kecepatan Skenario 8
1	Gagal	Gagal	Gagal	Gagal
2	Gagal	Gagal	Gagal	Gagal
3	0.043 detik	0.043 detik	0.043 detik	0.043 detik
4	Gagal	Gagal	Gagal	Gagal
5	Gagal	Gagal	Gagal	Gagal
6	Gagal	Gagal	Gagal	Gagal
7	Gagal	Gagal	Gagal	Gagal
8	Gagal	Gagal	Gagal	Gagal
9	Gagal	Gagal	Gagal	Gagal
10	Gagal	Gagal	Gagal	Gagal
11	Gagal	Gagal	Gagal	Gagal
12	Gagal	Gagal	Gagal	Gagal
13	0.077 detik	0.077 detik	0.077 detik	0.077 detik
14	Gagal	Gagal	Gagal	Gagal
15	Gagal	Gagal	Gagal	Gagal
16	Gagal	Gagal	Gagal	Gagal
17	Gagal	Gagal	Gagal	Gagal
18	0.043 detik	0.043 detik	0.043 detik	0.043 detik
19	Gagal	Gagal	Gagal	Gagal
20	Gagal	Gagal	Gagal	Gagal
21	1.247 detik	1.384 detik	1.315 detik	1.453 detik
22	0.146 detik	0.146 detik	0.146 detik	0.146 detik
23	Gagal	Gagal	Gagal	Gagal
24	Gagal	Gagal	Gagal	Gagal
25	Gagal	Gagal	Gagal	Gagal
26	Gagal	Gagal	Gagal	Gagal

Tabel B.12: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran $grid 5 \times 5$ (Skenario 9-12)

Nomor Soal	Kecepatan Skenario 9	Kecepatan Skenario 10	Kecepatan Skenario 11	Kecepatan Skenario 12
1	Gagal	Gagal	Gagal	Gagal
2	Gagal	Gagal	Gagal	Gagal
3	0.391 detik	0.391 detik	0.391 detik	0.391 detik
4	Gagal	Gagal	Gagal	Gagal
5	Gagal	Gagal	Gagal	Gagal
6	Gagal	Gagal	Gagal	Gagal
7	Gagal	Gagal	Gagal	Gagal
8	Gagal	Gagal	Gagal	Gagal
9	Gagal	Gagal	Gagal	Gagal
10	Gagal	Gagal	Gagal	Gagal
11	Gagal	Gagal	Gagal	Gagal
12	Gagal	Gagal	Gagal	Gagal
13	0.391 detik	0.391 detik	0.391 detik	0.391 detik
14	Gagal	Gagal	Gagal	Gagal
15	Gagal	Gagal	Gagal	Gagal
16	Gagal	Gagal	Gagal	Gagal
17	Gagal	Gagal	Gagal	Gagal
18	0.391 detik	0.391 detik	0.391 detik	0.391 detik
19	Gagal	Gagal	Gagal	Gagal
20	Gagal	Gagal	Gagal	Gagal
21	Gagal	Gagal	Gagal	Gagal
22	0.773 detik	0.773 detik	0.773 detik	0.773 detik
23	Gagal	Gagal	Gagal	Gagal
24	Gagal	Gagal	Gagal	Gagal
25	Gagal	Gagal	Gagal	Gagal
26	Gagal	Gagal	Gagal	Gagal

Tabel B.13: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran $grid 5 \times 5$ (Skenario 13-16)

Nomor Soal	Kecepatan Skenario 13	Kecepatan Skenario 14	Kecepatan Skenario 15	Kecepatan Skenario 16
1	Gagal	Gagal	Gagal	Gagal
2	Gagal	Gagal	Gagal	Gagal
3	0.043 detik	0.043 detik	0.043 detik	0.043 detik
4	Gagal	Gagal	Gagal	Gagal
5	Gagal	Gagal	Gagal	Gagal
6	Gagal	Gagal	Gagal	Gagal
7	Gagal	Gagal	Gagal	Gagal
8	Gagal	Gagal	Gagal	Gagal
9	Gagal	Gagal	Gagal	Gagal
10	Gagal	Gagal	Gagal	Gagal
11	Gagal	Gagal	Gagal	Gagal
12	Gagal	Gagal	Gagal	Gagal
13	0.077 detik	0.077 detik	0.077 detik	0.077 detik
14	Gagal	Gagal	Gagal	Gagal
15	Gagal	Gagal	Gagal	Gagal
16	Gagal	Gagal	Gagal	Gagal
17	Gagal	Gagal	Gagal	Gagal
18	0.043 detik	0.043 detik	0.043 detik	0.043 detik
19	Gagal	Gagal	Gagal	Gagal
20	Gagal	Gagal	Gagal	Gagal
21	Gagal	Gagal	Gagal	Gagal
22	0.146 detik	0.146 detik	0.146 detik	0.146 detik
23	Gagal	Gagal	Gagal	Gagal
24	Gagal	Gagal	Gagal	Gagal
25	Gagal	Gagal	Gagal	Gagal
26	Gagal	Gagal	Gagal	Gagal

Tabel B.14: Daftar jumlah sel yang berhasil diisi oleh algoritma *rule based* untuk Calcudoku dengan ukuran *grid* 4×4

Nomor Soal	Jumlah Sel Diisi Algoritma <i>Rule Based</i>
1	0
2	2
3	0
4	1
5	6
6	2
7	1
8	0
9	9
10	1
11	1
12	2
13	2
14	8
15	0
16	4
17	3
18	4
19	1
20	1
21	2
22	6
23	0
24	2
25	16
26	2
27	1
28	6
29	2
30	1
31	0
32	16
33	0
34	8
35	3
36	1
37	2
38	5
39	1

Tabel B.15: Daftar jumlah sel yang berhasil diisi oleh algoritma *rule based* untuk Calcudoku dengan ukuran *grid* 5×5

Nomor Soal	Jumlah Sel Diisi Algoritma <i>Rule Based</i>
1	4
2	0
3	15
4	1
5	1
6	1
7	5
8	1
9	5
10	0
11	0
12	3
13	15
14	2
15	0
16	1
17	0
18	16
19	4
20	2
21	8
22	14
23	1
24	5
25	5
26	1

LAMPIRAN C

FILE TEKS SOAL-SOAL PERMAINAN CALCUDOKU UNTUK PENGUJIAN

Lampiran ini berisi *file* teks dari soal-soal permainan Calcudoku yang dipakai dalam pengujian ini. Soal-soal ini diambil dari sumber-sumber berikut:

1. <https://iota.math.msu.edu/k12-outreach/kenken-puzzles/>
2. <http://thinkmath.edc.org/resource/kenken-puzzles>

C.1 File Teks Soal-Soal Permainan Calcudoku dengan *Grid* Berukuran 4×4

Listing C.1: 4x4_1.txt

```

1| 4 7
2| 1 2 2 3
3| 1 4 5 3
4| 6 4 5 5
5| 6 7 7 7
6| 1-
7| 1-
8| 3-
9| 2/
10| 24*
11| 3-
12| 6+

```

1,-	1,-		3,-
	2, ÷	24, ×	
3,-			
	6, +		

```

6| 24*
7| 3-
8| 9+
9| 2-
10| 6+
11| 2/

```

24, ×		3,-	
	9, +		
	2,-		6, +
2, ÷			

Listing C.2: 4x4_2.txt

```

1| 4 8
2| 1 2 2 3
3| 4 4 5 3
4| 6 4 5 5
5| 6 7 7 8
6| 4=
7| 2/
8| 1-
9| 6+
10| 12*
11| 1-
12| 5+
13| 2=

```

4	2, ÷		1,-
6, +		12, ×	
1,-			
	5, +		2

Listing C.3: 4x4_3.txt

```

1| 4 6
2| 1 1 2 2
3| 1 3 3 3
4| 1 4 4 5
5| 6 6 5 5

```

```

1| 4 7
2| 1 1 2 3
3| 4 4 2 3
4| 5 5 6 3
5| 7 6 6 3
6| 1-
7| 2-
8| 10+
9| 2/
10| 6*
11| 7+
12| 4=

```

1,-		2,-	10, +
2, ÷			
6, ×		7, +	
4			

Listing C.4: 4x4_4.txt

```

1| 4 7
2| 1 2 3 3
3| 1 2 4 4
4| 5 5 4 6
5| 5 7 7 6
6| 6*
7| 3-
8| 2/
9| 48*
10| 7+
11| 1-
12| 4+

```

Listing C.5: 4x4_5.txt

6, ×	3, -	2, ÷	
		48, ×	
7, +			1, -
	4, +		

Listing C.6: 4x4_6.txt

```

1| 4 8
2| 1 2 2 2
3| 1 3 4 5
4| 6 3 4 7
5| 6 8 7 7
6| 2/
7| 24*
8| 2-
9| 2/
10| 3=
11| 1-
12| 4+
13| 4=

```

2, ÷	24, ×		
	2, -	2, ÷	3
1, -			4, +
	4		

Listing C.7: 4x4_7.txt

```

1| 4 7
2| 1 1 2 2
3| 1 3 3 4
4| 5 6 6 7
5| 5 5 7 7
6| 8+
7| 2-
8| 3-
9| 3=
10| 9*
11| 2/
12| 10+

```

8, +		2, -	
	3, -		3
9, ×	2, ÷		10, +

Listing C.8: 4x4_8.txt

```

1| 4 7
2| 1 1 2 2
3| 1 3 4 5
4| 6 3 4 5
5| 6 7 7 7
6| 8*
7| 1-
8| 7+
9| 2/
10| 2-
11| 2-
12| 8*

```

8, ×		1, -	
	7, +	2, ÷	2, -
2, -			
	8, ×		

Listing C.9: 4x4_9.txt

```

1| 4 8
2| 1 2 2 3
3| 1 4 4 3
4| 1 5 6 7
5| 8 5 7 7
6| 24*
7| 1-
8| 3-
9| 7+
10| 2/
11| 1=
12| 18*
13| 1=

```

24, ×	1, -		3, -
	7, +		
	2, ÷	1	18, ×
1			

Listing C.10: 4x4_10.txt

1	4 7
2	1 2 2 3
3	1 4 3 3
4	5 4 6 6
5	5 5 7 6
6	2/
7	1-
8	4*
9	2-
10	8+
11	9+
12	2=

2, ÷	1, -		4, ×
	2, -		
8, +		9, +	
		2	

Listing C.11: 4x4_11.txt

1	4 7
2	1 1 2 2
3	3 4 2 5
4	3 4 6 5
5	7 7 7 7
6	2-
7	24*
8	1-
9	2/
10	5+
11	2=
12	10+

2, -		24, ×	
1, -	2, ÷		5, +
		2	
10, +			

Listing C.12: 4x4_12.txt

1	4 8
2	1 1 2 2
3	3 3 4 4
4	5 4 4 6
5	5 7 8 8
6	2/
7	4+
8	1-
9	12*
10	1-
11	4=
12	1=
13	1-

2, ÷		4, +	
1, -		12, ×	
1, -			4
	1	1, -	

Listing C.13: 4x4_13.txt

1	4 8
2	1 1 2 2
3	3 3 4 5
4	6 6 4 7
5	8 6 4 7
6	2-
7	2-
8	3+
9	9+
10	4=
11	9*
12	2/
13	4=

2, -		2, -	
3, +		9, +	4
9, ×			2, ÷
4			

Listing C.14: 4x4_14.txt

```

1| 4 7
2| 1 2 2 3
3| 1 1 4 3
4| 5 5 4 6
5| 7 7 6 6
6| 24*
7| 2/
8| 2-
9| 3+
10| 4+
11| 9+
12| 3-

```

6+	3-		2÷
	2÷	2-	
			2-
24×			

Listing C.16: 4x4_16.txt

```

1| 4 8
2| 1 2 2 3
3| 1 1 4 4
4| 5 5 6 6
5| 7 7 8 8
6| 12*
7| 4+
8| 4=
9| 5+
10| 3-
11| 5+
12| 7+
13| 2/

```

24×	2÷		2-
		3+	
4+			9+
3-			

Listing C.15: 4x4_15.txt

```

1| 4 7
2| 1 2 2 3
3| 1 4 5 3
4| 1 4 5 6
5| 7 7 7 6
6| 6+
7| 3-
8| 2/
9| 2/
10| 2-
11| 2-
12| 24*

```

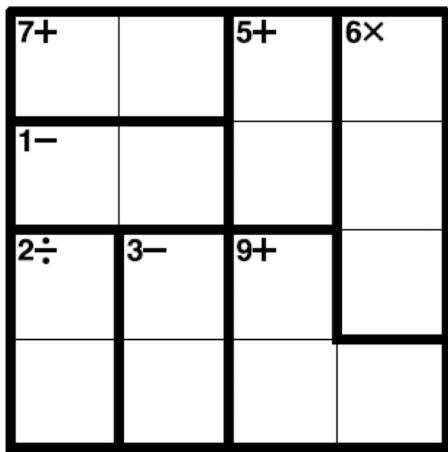
12×	4+		4
		5+	
3-		5+	
7+		2÷	

Listing C.17: 4x4_17.txt

```

1| 4 7
2| 1 1 2 3
3| 4 4 2 3
4| 5 6 7 3
5| 5 6 7 7
6| 7+
7| 5+
8| 6*
9| 1-
10| 2/
11| 3-
12| 9+

```

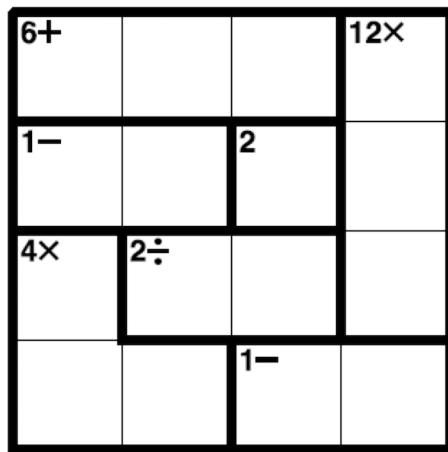


Listing C.18: 4x4_18.txt

```

1| 4 8
2| 1 1 2 3
3| 1 4 5 3
4| 6 4 5 7
5| 6 8 8 7
6| 8*
7| 3=
8| 2-
9| 3-
10| 2/
11| 5+
12| 2-
13| 4+

```

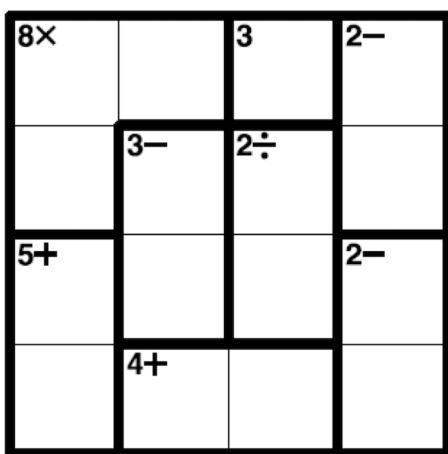


Listing C.20: 4x4_20.txt

```

1| 4 8
2| 1 2 2 3
3| 4 4 5 3
4| 6 6 5 7
5| 6 8 8 7
6| 2=
7| 12*
8| 2-
9| 3-
10| 2/
11| 7+
12| 2/
13| 3+

```

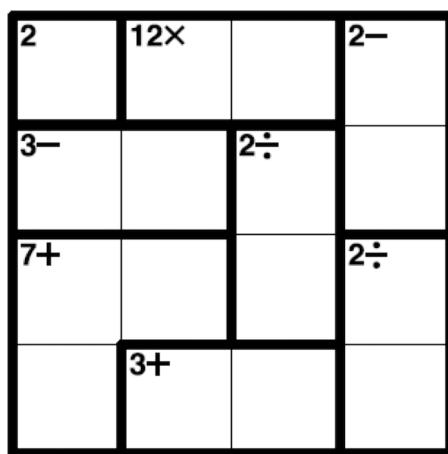


Listing C.19: 4x4_19.txt

```

1| 4 7
2| 1 1 1 2
3| 3 3 4 2
4| 5 6 6 2
5| 5 5 7 7
6| 6+
7| 12*
8| 1-
9| 2=
10| 4*
11| 2/
12| 1-

```

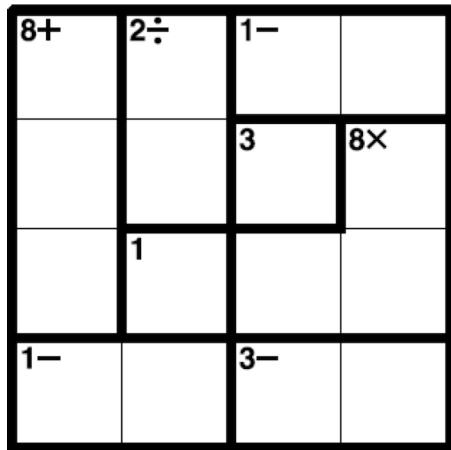


Listing C.21: 4x4_21.txt

```

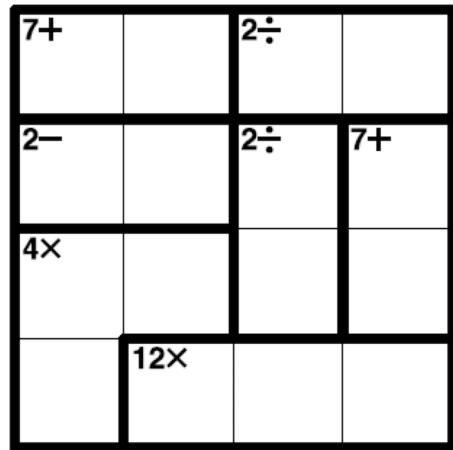
1| 4 8
2| 1 2 3 3
3| 1 2 4 5
4| 1 6 5 5
5| 7 7 8 8
6| 8+
7| 2/
8| 1-
9| 3=
10| 8*
11| 1=
12| 1-
13| 3-

```



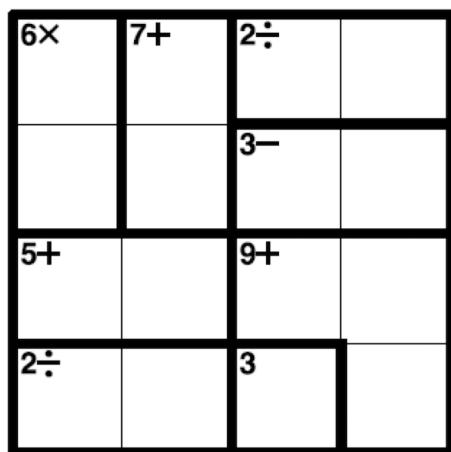
Listing C.22: 4x4_22.txt

1	4	8		
2	1	2	3	3
3	1	2	4	4
4	5	5	6	6
5	7	7	8	6
6	6*			
7	7+			
8	2/			
9	3-			
10	5+			
11	9+			
12	2/			
13	3=			



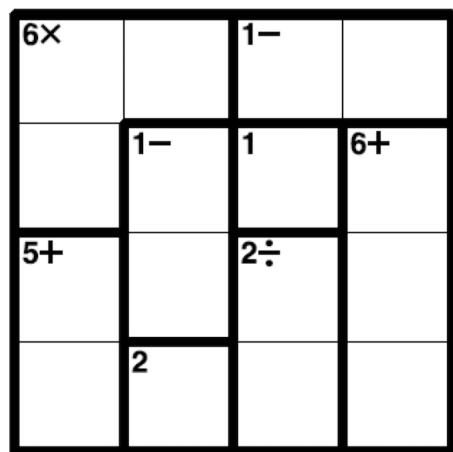
Listing C.24: 4x4_24.txt

1	4	8		
2	1	1	2	2
3	1	3	4	5
4	6	3	7	5
5	6	8	7	5
6	6*			
7	1-			
8	1-			
9	1=			
10	6+			
11	5+			
12	2/			
13	2=			



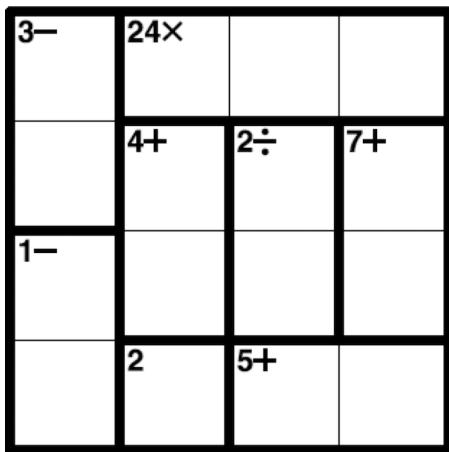
Listing C.23: 4x4_23.txt

1	4	7		
2	1	1	2	2
3	3	3	4	5
4	6	6	4	5
5	6	7	7	7
6	7+			
7	2/			
8	2-			
9	2/			
10	7+			
11	4*			
12	12*			



Listing C.25: 4x4_25.txt

1	4	8		
2	1	2	2	2
3	1	3	4	5
4	6	3	4	5
5	6	7	8	8
6	3-			
7	24*			
8	4+			
9	2/			
10	7+			
11	1-			
12	2=			
13	5+			

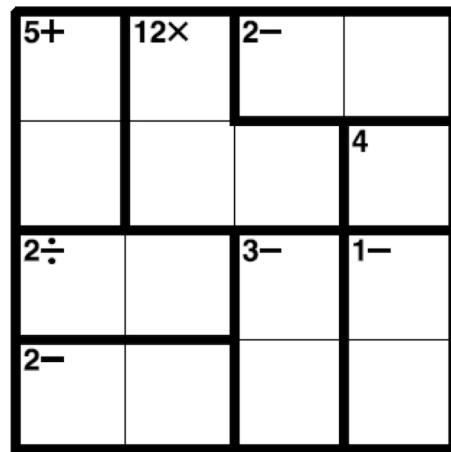


Listing C.26: 4x4_26.txt

```

1| 4 7
2| 1 1 2 2
3| 3 1 4 4
4| 3 5 4 4
5| 6 6 6 7
6| 6*
7| 2/
8| 1-
9| 9+
10| 4=
11| 8*
12| 3=

```

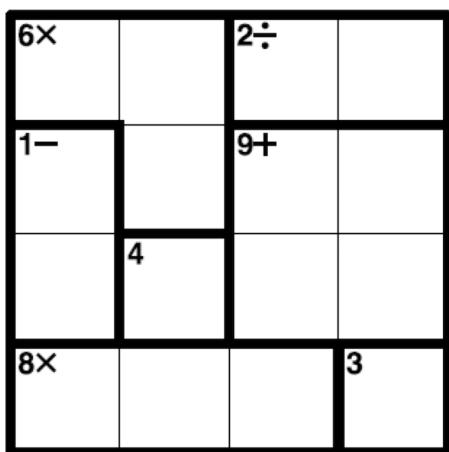


Listing C.28: 4x4_28.txt

```

1| 4 9
2| 1 2 3 4
3| 1 2 5 4
4| 6 7 5 8
5| 6 7 9 9
6| 7+
7| 1-
8| 1=
9| 2/
10| 2/
11| 2/
12| 5+
13| 3=
14| 12*

```

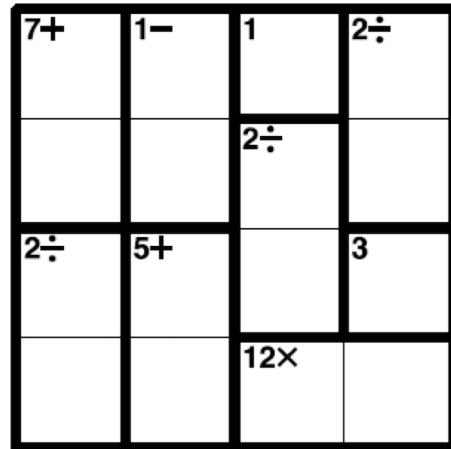


Listing C.27: 4x4_27.txt

```

1| 4 8
2| 1 2 3 3
3| 1 2 2 4
4| 5 5 6 7
5| 8 8 6 7
6| 5+
7| 12*
8| 2-
9| 4=
10| 2/
11| 3-
12| 1-
13| 2-

```

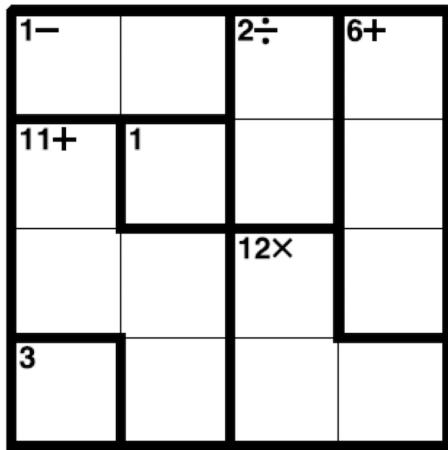


Listing C.29: 4x4_29.txt

```

1| 4 7
2| 1 1 2 3
3| 4 5 2 3
4| 4 4 6 3
5| 7 4 6 6
6| 1-
7| 2/
8| 6+
9| 11+
10| 1=
11| 12*
12| 3=

```

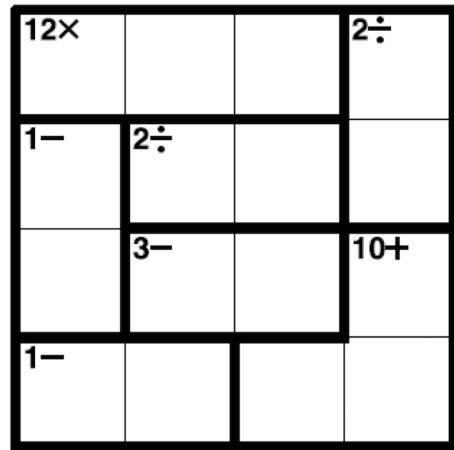


Listing C.30: 4x4_30.txt

```

1| 4 8
2| 1 2 2 3
3| 1 4 4 5
4| 6 6 5 5
5| 7 7 8 8
6| 2-
7| 1-
8| 1=
9| 4+
10| 48*
11| 3+
12| 1-
13| 2/

```

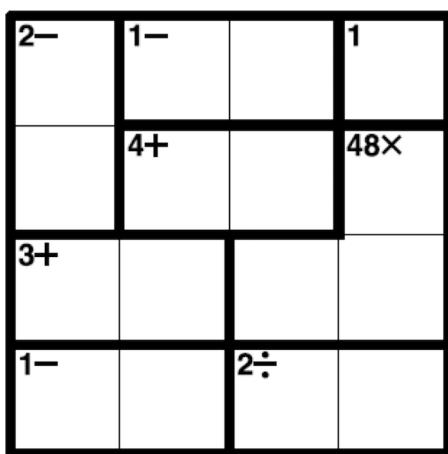


Listing C.32: 4x4_32.txt

```

1| 4 8
2| 1 2 2 3
3| 1 4 4 3
4| 5 6 6 7
5| 5 8 6 7
6| 3-
7| 2/
8| 7+
9| 6*
10| 1-
11| 9+
12| 1-
13| 3=

```

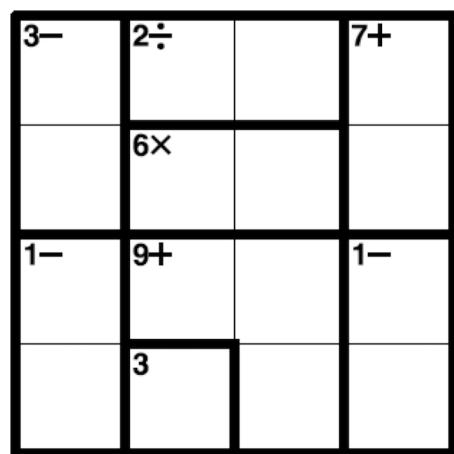


Listing C.31: 4x4_31.txt

```

1| 4 7
2| 1 1 1 2
3| 3 4 4 2
4| 3 5 5 6
5| 7 7 6 6
6| 12*
7| 2/
8| 1-
9| 2/
10| 3-
11| 10+
12| 1-

```

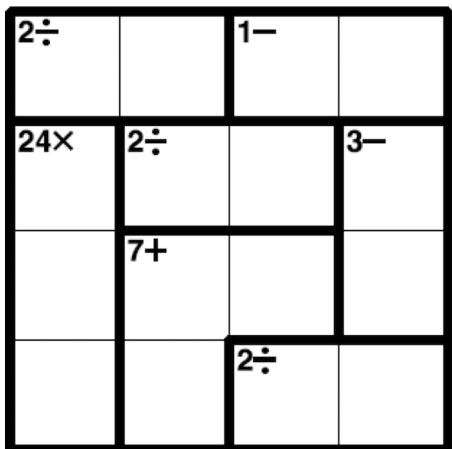


Listing C.33: 4x4_33.txt

```

1| 4 7
2| 1 1 2 2
3| 3 4 4 5
4| 3 6 6 5
5| 3 6 7 7
6| 2/
7| 1-
8| 24*
9| 2/
10| 3-
11| 7+
12| 2/

```

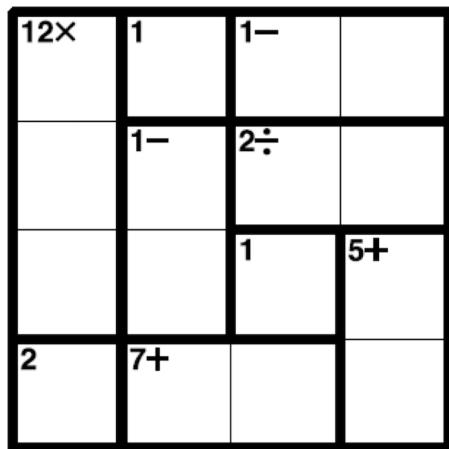


Listing C.34: 4x4_34.txt

```

1| 4 8
2| 1 1 2 3
3| 4 4 2 3
4| 5 6 6 7
5| 5 5 8 8
6| 3+
7| 3-
8| 1-
9| 1-
10| 48*
11| 2-
12| 2=
13| 2/

```

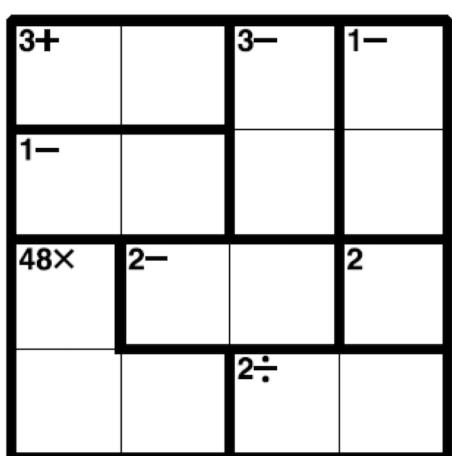


Listing C.36: 4x4_36.txt

```

1| 4 7
2| 1 2 2 3
3| 1 1 3 3
4| 4 5 3 6
5| 4 5 7 7
6| 2*
7| 7+
8| 10+
9| 1-
10| 5+
11| 4=
12| 2/

```

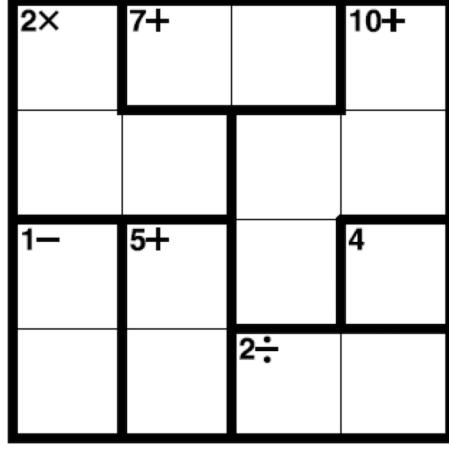


Listing C.35: 4x4_35.txt

```

1| 4 9
2| 1 2 3 3
3| 1 4 5 5
4| 1 4 6 7
5| 8 9 9 7
6| 12*
7| 1=
8| 1-
9| 1-
10| 2/
11| 1=
12| 5+
13| 2=
14| 7+

```

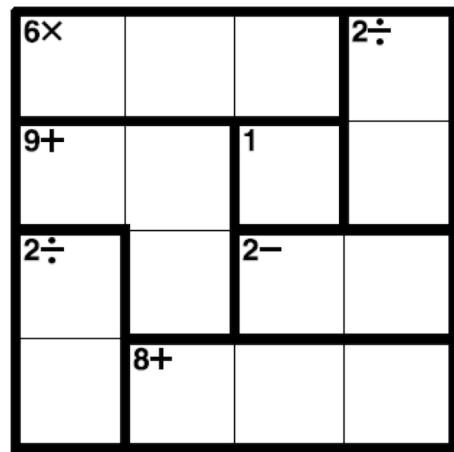
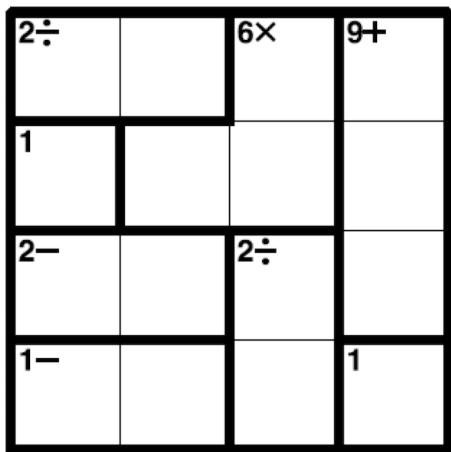


Listing C.37: 4x4_37.txt

```

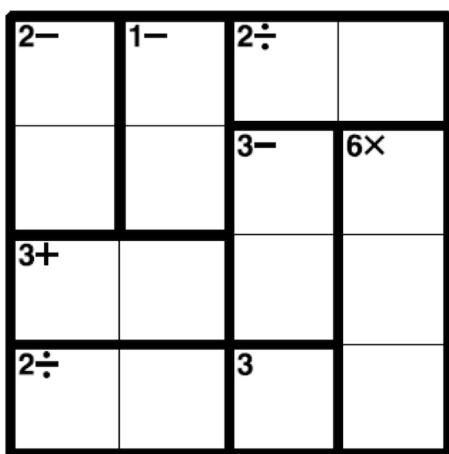
1| 4 8
2| 1 1 2 3
3| 4 2 2 3
4| 5 5 6 3
5| 7 7 6 8
6| 2/
7| 6*
8| 9+
9| 1=
10| 2-
11| 2/
12| 1-
13| 1=

```



Listing C.38: 4x4_38.txt

1	4	8		
2	1	2	3	3
3	1	2	4	5
4	6	6	4	5
5	7	7	8	5
6	2-			
7	1-			
8	2/			
9	3-			
10	6*			
11	3+			
12	2/			
13	3=			



Listing C.39: 4x4_39.txt

1	4	7		
2	1	1	1	2
3	3	3	4	2
4	5	3	6	6
5	5	7	7	7
6	6*			
7	2/			
8	9+			
9	1=			
10	2/			
11	2-			
12	8+			

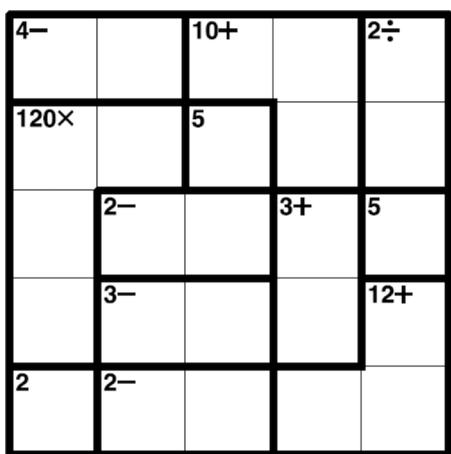
C.2 File Teks Soal-Soal Permainan Calcudoku dengan *Grid* Berukuran 5×5

Listing C.40: 5x5_1.txt

```

1| 5 12
2| 1 1 2 2 3
3| 4 4 5 2 3
4| 4 6 6 7 8
5| 4 9 9 7 10
6| 11 12 12 10 10
7| 4-
8| 10+
9| 2/
10| 120*
11| 5=
12| 2-
13| 3+
14| 5=
15| 3-
16| 12+
17| 2=
18| 2-

```

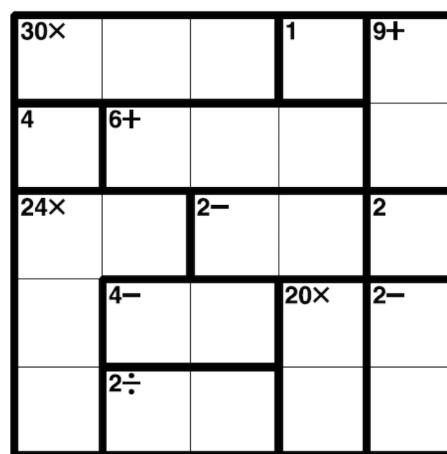


Listing C.42: 5x5_3.txt

```

1| 5 12
2| 1 1 1 2 3
3| 4 5 5 5 3
4| 6 6 7 7 8
5| 6 9 9 10 11
6| 6 12 12 10 11
7| 30*
8| 1=
9| 9+
10| 4=
11| 6+
12| 24*
13| 2-
14| 2=
15| 4-
16| 20*
17| 2-
18| 2/

```

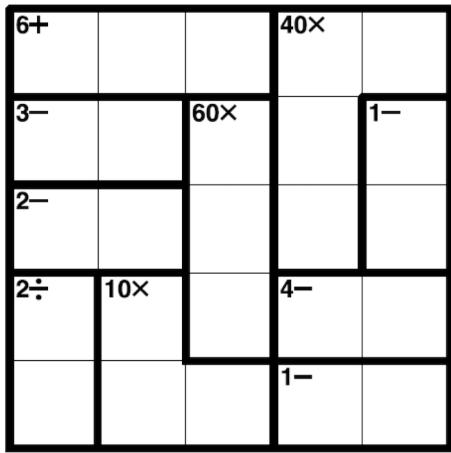


Listing C.41: 5x5_2.txt

```

1| 5 10
2| 1 1 1 2 2
3| 3 3 4 2 5
4| 6 6 4 2 5
5| 7 8 4 9 9
6| 7 8 8 10 10
7| 6+
8| 40*
9| 3-
10| 60*
11| 1-
12| 2-
13| 2/
14| 10*
15| 4-
16| 1-

```

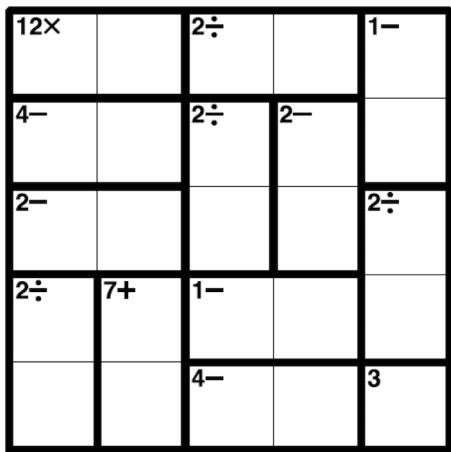


Listing C.43: 5x5_4.txt

```

1| 5 13
2| 1 1 2 2 3
3| 4 4 5 6 3
4| 7 7 5 6 8
5| 9 10 11 11 8
6| 9 10 12 12 13
7| 12*
8| 2/
9| 1-
10| 4-
11| 2/
12| 2-
13| 2-
14| 2/
15| 2/
16| 7+
17| 1-
18| 4-
19| 3=

```

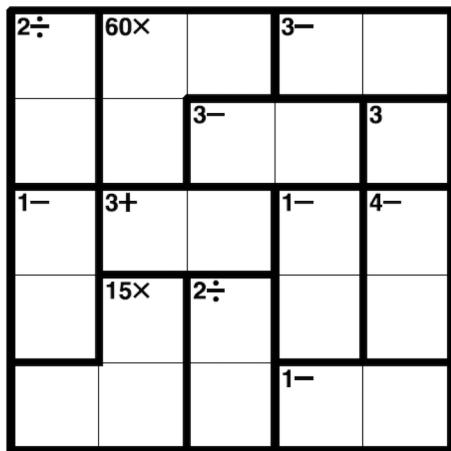


Listing C.44: 5x5_5.txt

```

1| 5 12
2| 1 2 2 3 3
3| 1 2 4 4 5
4| 6 7 7 8 9
5| 6 10 11 8 9
6| 10 10 11 12 12
7| 2/
8| 60*
9| 3-
10| 3-
11| 3=
12| 1-
13| 3+
14| 1-
15| 4-
16| 15*
17| 2/
18| 1-

```

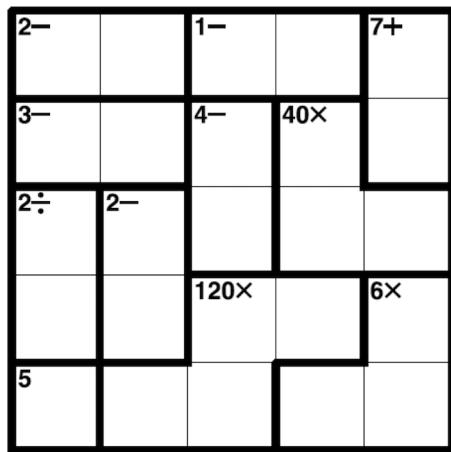


Listing C.45: 5x5_6.txt

```

1| 5 11
2| 1 1 2 2 3
3| 4 4 5 6 3
4| 7 8 5 6 6
5| 7 8 9 9 10
6| 11 9 9 10 10
7| 2-
8| 1-
9| 7+
10| 3-
11| 4-
12| 40*
13| 2/
14| 2-
15| 120*
16| 6*
17| 5=

```

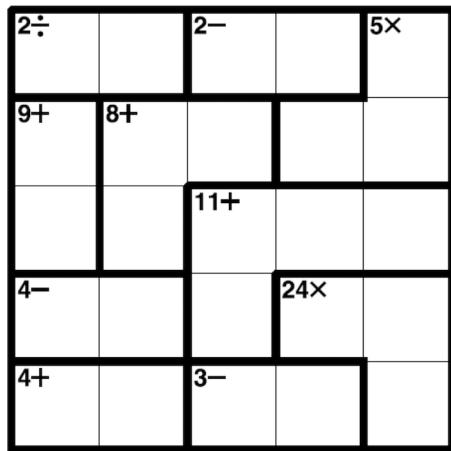


Listing C.46: 5x5_7.txt

```

1| 5 10
2| 1 1 2 2 3
3| 4 5 5 3 3
4| 4 5 6 6 6
5| 7 7 6 8 8
6| 9 9 10 10 8
7| 2/
8| 2-
9| 5*
10| 9+
11| 8+
12| 11+
13| 4-
14| 24*
15| 4+
16| 3-

```

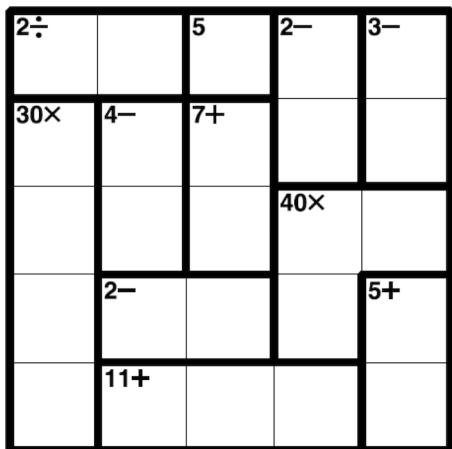


Listing C.47: 5x5_8.txt

```

1| 5 11
2| 1 1 2 3 4
3| 5 6 7 3 4
4| 5 6 7 8 8
5| 5 9 9 8 10
6| 5 11 11 11 10
7| 2/
8| 5=
9| 2-
10| 3-
11| 30*
12| 4-
13| 7+
14| 40*
15| 2-
16| 5+
17| 11+

```

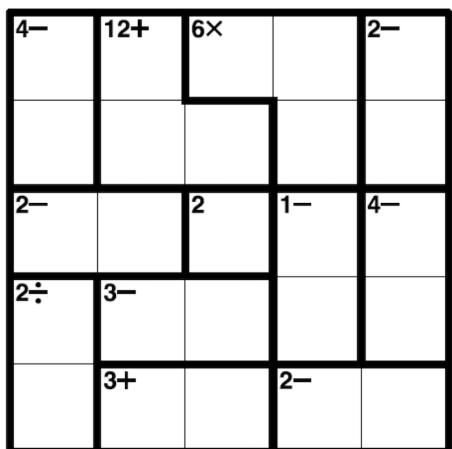


Listing C.48: 5x5_9.txt

```

1| 5 12
2| 1 2 3 3 4
3| 1 2 2 3 4
4| 5 5 6 7 8
5| 9 10 10 7 8
6| 9 11 11 12 12
7| 4-
8| 12+
9| 6*
10| 2-
11| 2-
12| 2=
13| 1-
14| 4-
15| 2/
16| 3-
17| 3+
18| 2-

```

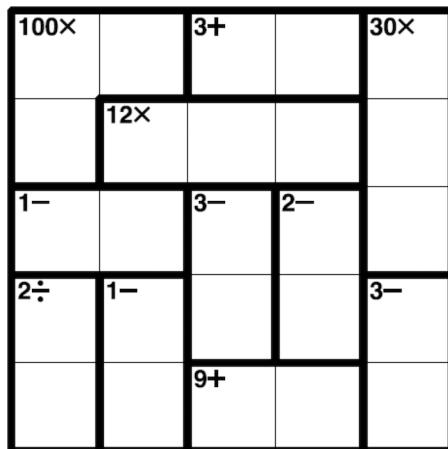


Listing C.49: 5x5_10.txt

```

1| 5 11
2| 1 1 2 2 3
3| 1 4 4 4 3
4| 5 5 6 7 3
5| 8 9 6 7 10
6| 8 9 11 11 10
7| 100*
8| 3+
9| 30*
10| 12*
11| 1-
12| 3-
13| 2-
14| 2/
15| 1-
16| 3-
17| 9+

```

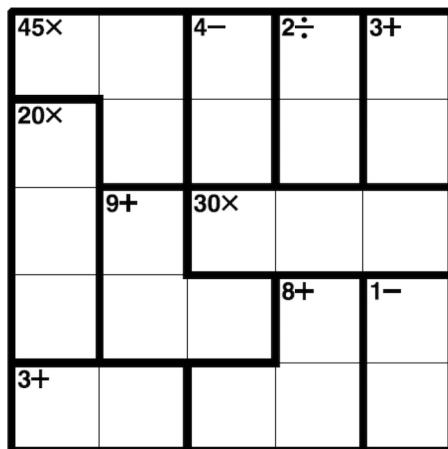


Listing C.50: 5x5_11.txt

```

1| 5 10
2| 1 1 2 3 4
3| 5 1 2 3 4
4| 5 6 7 7 7
5| 5 6 6 8 9
6| 10 10 8 8 9
7| 45*
8| 4-
9| 2/
10| 3+
11| 20*
12| 9+
13| 30*
14| 8+
15| 1-
16| 3+

```

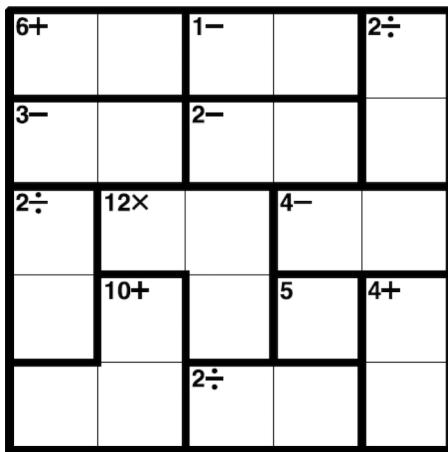


Listing C.51: 5x5_12.txt

```

1| 5 12
2| 1 1 2 2 3
3| 4 4 5 5 3
4| 6 7 7 8 8
5| 6 9 7 10 11
6| 9 9 12 12 11
7| 6+
8| 1-
9| 2/
10| 3-
11| 2-
12| 2/
13| 12*
14| 4-
15| 10+
16| 5=
17| 4+
18| 2/

```

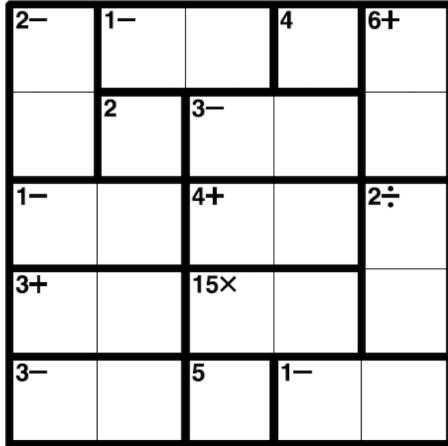


Listing C.52: 5x5_13.txt

```

1| 5 14
2| 1 2 2 3 4
3| 1 5 6 6 4
4| 7 7 8 8 9
5| 10 10 11 11 9
6| 12 12 13 14 14
7| 2-
8| 1-
9| 4=
10| 6+
11| 2=
12| 3-
13| 1-
14| 4+
15| 2/
16| 3+
17| 15*
18| 3-
19| 5=
20| 1-

```

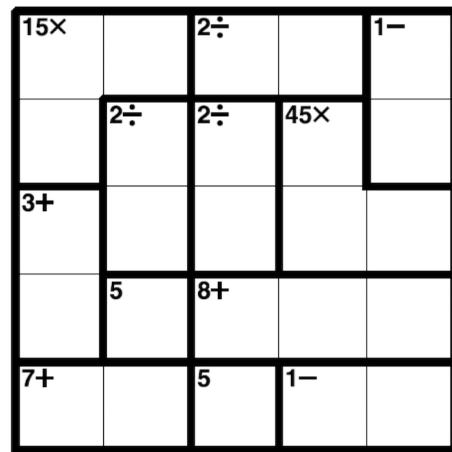


Listing C.53: 5x5_14.txt

```

1| 5 12
2| 1 1 2 2 3
3| 1 4 5 6 3
4| 7 4 5 6 6
5| 7 8 9 9 9
6| 10 10 11 12 12
7| 15*
8| 2/
9| 1-
10| 2/
11| 2/
12| 45*
13| 3+
14| 5=
15| 8+
16| 7+
17| 5=
18| 1-

```

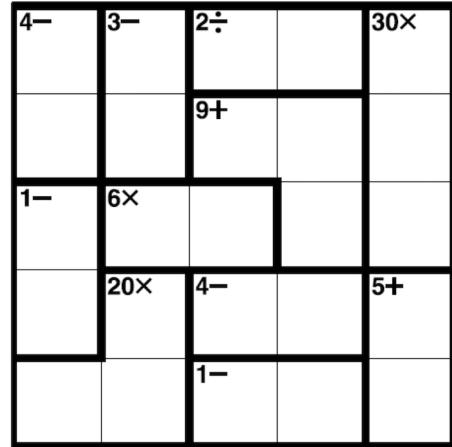


Listing C.54: 5x5_15.txt

```

1| 5 11
2| 1 2 3 3 4
3| 1 2 5 5 4
4| 6 7 7 5 4
5| 6 8 9 9 10
6| 8 8 11 11 10
7| 4-
8| 3-
9| 2/
10| 30*
11| 9+
12| 1-
13| 6*
14| 20*
15| 4-
16| 5+
17| 1-

```

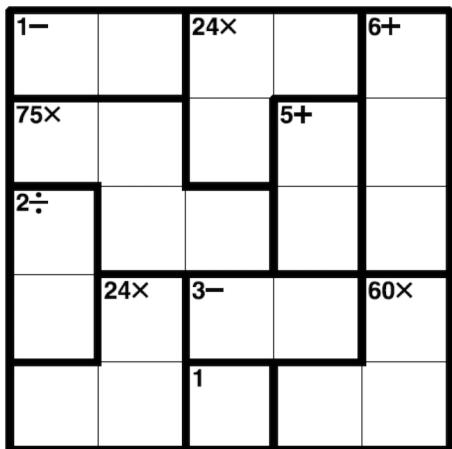


Listing C.55: 5x5_16.txt

```

1| 5 10
2| 1 1 2 2 3
3| 4 4 2 5 3
4| 6 4 4 5 3
5| 6 7 8 8 9
6| 7 7 10 9 9
7| 1-
8| 24*
9| 6+
10| 75*
11| 5+
12| 2/
13| 24*
14| 3-
15| 60*
16| 1=

```

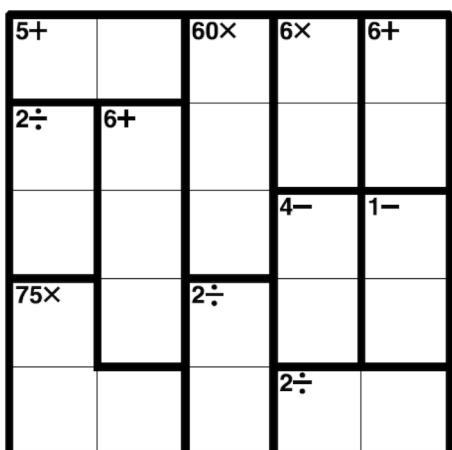


Listing C.56: 5x5_17.txt

```

1| 5 11
2| 1 1 2 3 4
3| 5 6 2 3 4
4| 5 6 2 7 8
5| 9 6 10 7 8
6| 9 9 10 11 11
7| 5+
8| 60*
9| 6*
10| 6+
11| 2/
12| 6+
13| 4-
14| 1-
15| 75*
16| 2/
17| 2/

```

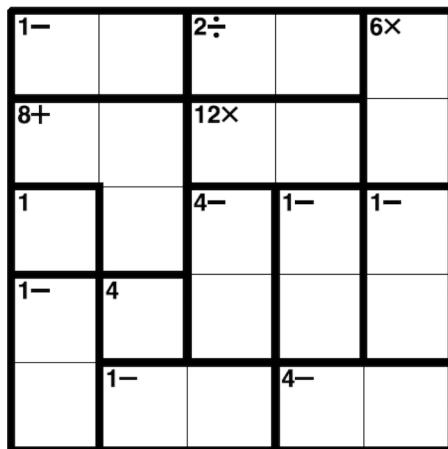


Listing C.57: 5x5_18.txt

```

1| 5 13
2| 1 1 2 2 3
3| 4 4 5 5 3
4| 6 4 7 8 9
5| 10 11 7 8 9
6| 10 12 12 13 13
7| 1-
8| 2/
9| 6*
10| 8+
11| 12*
12| 1=
13| 4-
14| 1-
15| 1-
16| 1-
17| 4=
18| 1-
19| 4-

```

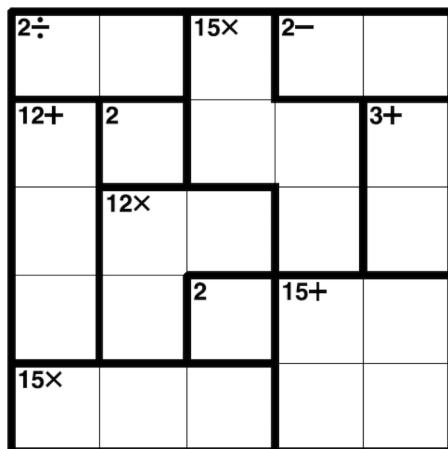


Listing C.58: 5x5_19.txt

```

1| 5 10
2| 1 1 2 3 3
3| 4 5 2 2 6
4| 4 7 7 2 6
5| 4 7 8 9 9
6| 10 10 10 9 9
7| 2/
8| 15*
9| 2-
10| 12+
11| 2=
12| 3+
13| 12*
14| 2=
15| 15+
16| 15*

```



Listing C.59: 5x5_20.txt

```

1| 5 12
2| 1 1 2 2 3
3| 4 5 5 6 3
4| 4 7 7 6 8
5| 4 9 8 8 8
6| 10 10 11 11 12
7| 3-
8| 5+
9| 1-
10| 12+
11| 15*
12| 2-
13| 2/
14| 40*
15| 1=
16| 5+
17| 1-
18| 5=

```

3-		5+		1-
12+	15×		2-	
	2÷			40×
	1			
5+		1-		5

Listing C.60: 5x5_21.txt

```

1| 5 13
2| 1 2 2 3 4
3| 5 5 6 3 4
4| 5 7 6 8 9
5| 10 7 11 8 9
6| 10 12 12 13 9
7| 4=
8| 4-
9| 1-
10| 2/
11| 12*
12| 2-
13| 3-
14| 3+
15| 12+
16| 4-
17| 4=
18| 1-
19| 5=

```

4	4-		1-	2÷
12×		2-		
	3-		3+	12+
4-		4		
	1-		5	

Listing C.61: 5x5_22.txt

```

1| 5 13
2| 1 2 3 4 4
3| 1 5 3 6 6
4| 7 5 3 8 8
5| 9 10 10 11 12
6| 9 13 13 11 12
7| 20*
8| 3=
9| 8*
10| 2/
11| 10*
12| 4+
13| 3=
14| 1-
15| 1-
16| 9+
17| 10*
18| 1-
19| 2-

```

20×	3	8×	2÷	
	10×		4+	
3			1-	
1-	9+		10×	1-
	2-			

Listing C.62: 5x5_23.txt

```

1| 5 11
2| 1 2 3 3 4
3| 1 2 2 5 5
4| 1 2 6 6 7
5| 8 8 9 9 7
6| 10 10 10 11 11
7| 15*
8| 96*
9| 7+
10| 3=
11| 4-
12| 2/
13| 2/
14| 3-
15| 2-
16| 10+
17| 5+

```

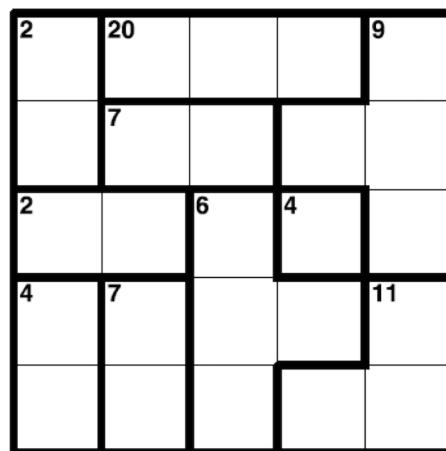
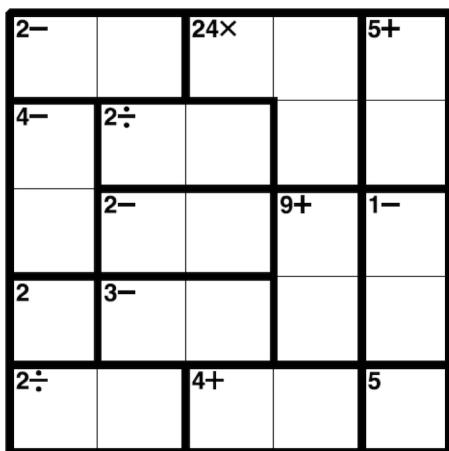
15×	96×	7+		3
			4-	
		2÷		2÷
3-		2-		
10+		5+		

Listing C.63: 5x5_24.txt

```

1| 5 13
2| 1 1 2 2 3
3| 4 5 5 2 3
4| 4 6 6 7 8
5| 9 10 10 7 8
6| 11 11 12 12 13
7| 2-
8| 24*
9| 5+
10| 4-
11| 2/
12| 2-
13| 9+
14| 1-
15| 2=
16| 3-
17| 2/
18| 4+
19| 5=

```

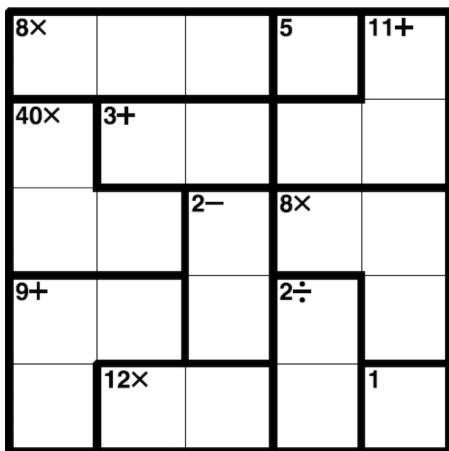


Listing C.64: 5x5_25.txt

```

1| 5 11
2| 1 1 1 2 3
3| 4 5 5 3 3
4| 4 4 6 7 7
5| 8 8 6 9 7
6| 8 10 10 9 11
7| 8*
8| 5=
9| 11+
10| 49*
11| 3+
12| 2-
13| 8*
14| 9+
15| 2/
16| 12*
17| 1=

```



Listing C.65: 5x5_26.txt

```

1| 5 10
2| 1 2 2 2 3
3| 1 4 4 3 3
4| 5 5 6 7 3
5| 8 9 6 6 10
6| 8 9 6 10 10
7| 2/
8| 20*
9| 9+
10| 7+
11| 2-
12| 6*
13| 4=
14| 4-
15| 7+
16| 11+

```

C.3 File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran 6×6

Listing C.66: 6x6_1.txt

```

1| 6 15
2| 1 1 2 3 4 4
3| 1 5 2 3 6 4
4| 5 5 7 7 6 8
5| 9 9 10 10 6 11
6| 12 12 13 10 11 11
7| 12 13 13 14 15 15
8| 30*
9| 3+
10| 11+
11| 16*
12| 13+
13| 36*
14| 1-
15| 3=
16| 2/
17| 9*
18| 50*
19| 8+
20| 15+
21| 2=
22| 5-

```

2, ÷		5	2, -	6, ×	
8, +	3, ÷			4	
	4	3, ÷		3, ÷	3, -
2, ÷		5, -	24, ×		
4, -					
2	5, +		11, +		

30, ×		3, +	11, +	16, ×	
	13, +			36, ×	
		1, -			3
2, ÷		9, ×			50, ×
8, +		15, +			
			2	5, -	

Listing C.67: 6x6_2.txt

```

1| 6 19
2| 1 1 2 3 4 4
3| 5 6 6 3 7 4
4| 5 8 9 9 10 11
5| 12 12 13 14 10 11
6| 15 15 13 14 14 16
7| 17 18 18 19 19 16
8| 2/
9| 5=
10| 2-
11| 6*
12| 8+
13| 3/
14| 4=
15| 4=
16| 3/
17| 3/
18| 3-
19| 2/
20| 5-
21| 24*
22| 4-
23| 12*
24| 2=
25| 5+
26| 11+

```

```

1| 6 19
2| 1 1 2 3 4 5
3| 6 6 2 3 4 7
4| 8 9 10 10 10 7
5| 8 11 11 12 12 13
6| 14 15 16 17 17 13
7| 15 15 16 18 19 19
8| 2/
9| 2-
10| 1-
11| 1-
12| 5=
13| 2/
14| 3+
15| 5-
16| 4=
17| 15*
18| 3-
19| 2/
20| 1-
21| 4=
22| 75*
23| 5-
24| 2/
25| 4=
26| 3/

```

2, ÷		2, -	1, -	1, -	5
2, ÷					3, +
5, -	4	15, ×			
	3, -		2, ÷		1, -
4	75, ×	5, -	2, ÷		
			4	3, ÷	

Listing C.69: 6x6_4.txt

```

1| 6 18
2| 1 2 3 3 4 5
3| 1 6 7 3 4 5
4| 8 6 7 9 10 11
5| 8 12 12 9 11 11
6| 13 14 14 15 15 15
7| 16 16 14 17 17 18
8| 11+

```

9 | 6=
 10 4*
 11 12*
 12 8+
 13 1-
 14 3-
 15 1-
 16 11+
 17 2=
 18 4*
 19 2/
 20 4=
 21 9+
 22 36*
 23 2/
 24 1-
 25 6=

11, +	6	4, ×		12, ×	8, +
	1, -	3, -			
1 -			11, +	2	4, ×
	2, ÷				
4	9, +		36, ×		
2, ÷		1, -		6	

Listing C.70: 6x6_5.txt

1 | 6 18
 2 1 2 2 3 3 3
 3 4 2 5 6 7 3
 4 4 8 9 6 7 10
 5 11 8 9 12 13 10
 6 11 14 15 12 13 13
 7 16 16 15 17 17 18
 8 5=
 9 4*
 10 216*
 11 3+
 12 4=
 13 6+
 14 2-
 15 1-
 16 15*
 17 3/
 18 2/
 19 7+
 20 8+
 21 6=
 22 5-
 23 1-
 24 7+
 25 1=

5	4, ×		216, ×		
3, +		4	6, +	2, -	
	1, -	15, ×			3, ÷
2, ÷			7, +	8, +	
1, -			7, +		1

Listing C.71: 6x6_6.txt

1 | 6 18
 2 1 1 2 2 3 4
 3 5 1 6 2 7 4
 4 5 8 6 9 7 10
 5 11 8 12 12 13 10
 6 14 14 14 12 13 15
 7 16 16 17 18 18 15
 8 24*
 9 2*
 10 5=
 11 11+
 12 3/
 13 7+
 14 2-
 15 5-
 16 5=
 17 2/
 18 6=
 19 12+
 20 1-
 21 13+
 22 3/
 23 20*
 24 2=
 25 5-

24, ×		2, ×		5	11, +
3, ÷		7, +		2, -	
	5, -		5		2, ÷
6		12, +		1, -	
13, +					3, ÷
20, ×		2	5, -		

Listing C.72: 6x6_7.txt

1 | 6 19
 2 1 2 2 2 3 3
 3 4 5 5 6 7 3
 4 4 8 9 9 7 10
 5 11 11 12 13 14 10
 6 15 15 12 13 14 16
 7 17 17 18 19 19 16
 8 1=
 9 72*
 10 11+
 11 20*
 12 1-
 13 1=
 14 2/
 15 6=
 16 7+
 17 3/
 18 7+
 19 6+
 20 2/
 21 9+
 22 6*
 23 3/
 24 2-
 25 4=
 26 5-

1	72 ×			11 +	
20 ×	1 -		1	2 ÷	
	6	7 +			3 ÷
7 +		6 +	2 ÷	9 +	
6 ×					3 ÷
2 -		4	5 -		

Listing C.73: 6x6_8.txt

1| 6 16
 2| 1 1 2 3 4 5
 3| 6 6 2 3 4 5
 4| 7 8 3 9 9
 5| 7 7 8 10 10 9
 6| 11 12 13 14 14 15
 7| 11 12 13 16 16 15
 8| 3/
 9| 9+
 10| 15+
 11| 7+
 12| 2/
 13| 1-
 14| 12+
 15| 3*
 16| 16+
 17| 1-
 18| 1-
 19| 30*
 20| 3/
 21| 1-
 22| 2-
 23| 2/

3 ÷		9 +	15 +	7 +	2 ÷
1 -					
12 +	3 ×			16 +	
			1 -		
1 -	30 ×	3 ÷	1 -		2 -
			2 ÷		

Listing C.74: 6x6_9.txt

1| 6 16
 2| 1 1 2 2 3 3
 3| 1 1 4 5 6 7
 4| 8 9 4 5 6 10
 5| 8 9 9 11 12 10
 6| 13 14 14 11 12 10
 7| 13 13 15 15 16 16
 8| 14+
 9| 1-
 10| 2-
 11| 2-
 12| 1-
 13| 3/
 14| 1=
 15| 2/
 16| 4*
 17| 40*
 18| 3/
 19| 11+
 20| 80*

21| 2-
 22| 5-
 23| 1-

14 +		1 -		2 -	
		2 -	1 -	3 ÷	1
2 ÷	4 ×				40 ×
			3 ÷	11 +	
80 ×	2 -				
		5 -		1 -	

Listing C.75: 6x6_10.txt

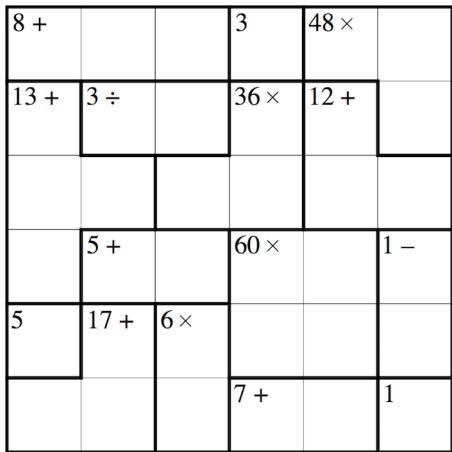
1| 6 17
 2| 1 1 2 2 3 3
 3| 1 1 4 5 6 3
 4| 7 4 4 8 6 6
 5| 9 9 10 8 11 11
 6| 12 13 10 14 15 16
 7| 12 13 17 14 15 16
 8| 14+
 9| 30*
 10| 3*
 11| 60*
 12| 6=
 13| 12+
 14| 2=
 15| 2/
 16| 30*
 17| 3+
 18| 1-
 19| 5-
 20| 5+
 21| 1-
 22| 1-
 23| 3-
 24| 3=

14 +		30 ×		3 ×	
		60 ×	6	12 +	
2			2 ÷		
30 ×		3 +		1 -	
5 -	5 +		1 -	1 -	3 -
		3			

Listing C.76: 6x6_11.txt

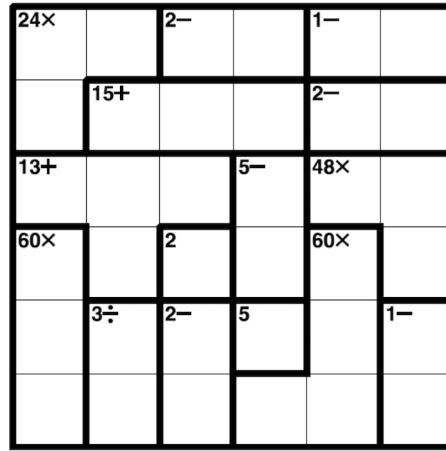
1| 6 15
 2| 1 1 1 2 3 3
 3| 4 5 5 6 7 3
 4| 4 4 6 6 7 7
 5| 4 8 8 9 9 10
 6| 11 12 13 9 9 10
 7| 12 12 13 14 14 15
 8| 8+
 9| 3=
 10| 48*
 11| 13+
 12| 3/
 13| 36*
 14| 12+

15	5+
16	60*
17	1-
18	5=
19	17+
20	6*
21	7+
22	1=



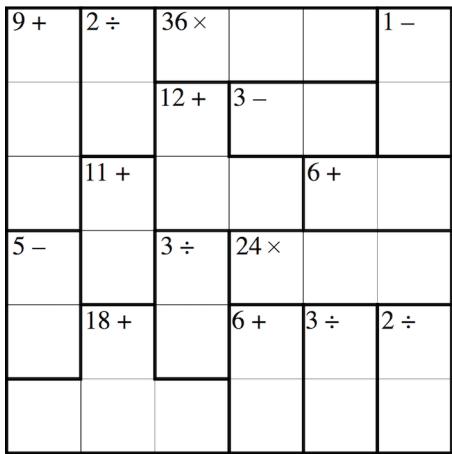
Listing C.77: 6x6_12.txt

12	2-
13	13+
14	5-
15	48*
16	60*
17	2=
18	60*
19	3/
20	2-
21	5=
22	1-



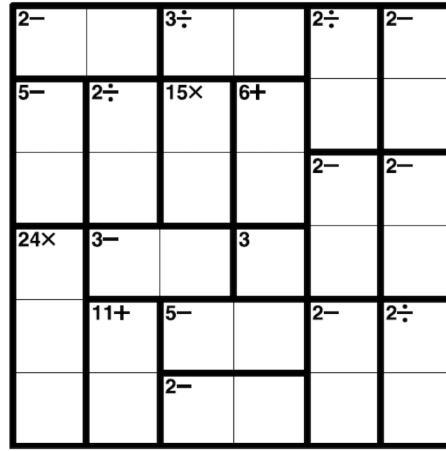
Listing C.79: 6x6_14.txt

1	6 15
2	1 2 3 3 3 4
3	1 2 5 6 6 4
4	1 7 5 5 8 8
5	9 7 10 11 11 11
6	9 12 10 13 14 15
7	12 12 12 13 14 15
8	9+
9	2/
10	36*
11	1-
12	12+
13	3-
14	11+
15	6+
16	5-
17	3/
18	24*
19	18+
20	6+
21	3/
22	2/



Listing C.78: 6x6_13.txt

1	6 18
2	1 1 2 2 3 4
3	5 6 7 8 3 4
4	5 6 7 8 9 10
5	11 12 12 13 9 10
6	11 14 15 15 16 17
7	11 14 18 18 16 17
8	2-
9	3/
10	2/
11	2-
12	5-
13	2/
14	15*
15	6+
16	2-
17	2-
18	24*
19	3-
20	3=
21	11+
22	5-
23	2-
24	2/
25	2-

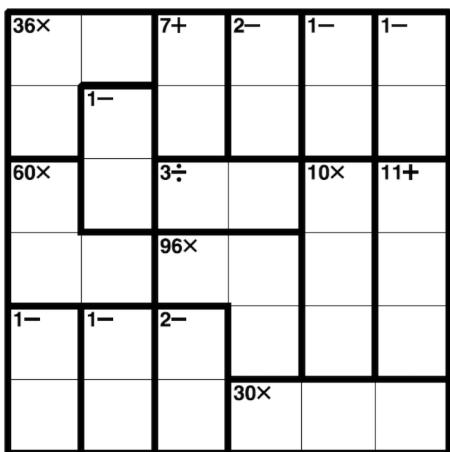


Listing C.80: 6x6_15.txt

1	6 15
2	1 1 2 2 3 3
3	1 4 4 4 5 5
4	6 6 6 7 8 8
5	9 6 10 7 11 8
6	9 12 13 14 11 15
7	9 12 13 11 11 15
8	24*
9	2-
10	1-
11	15+

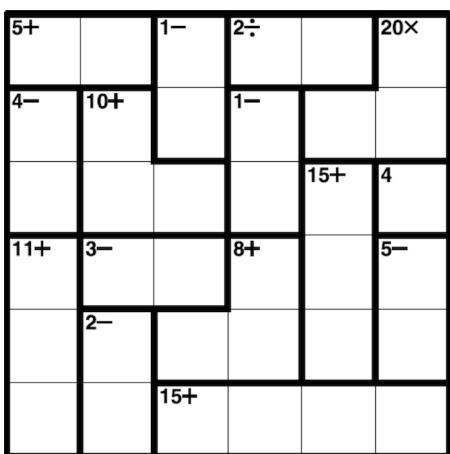
1	6 15
2	1 1 2 3 4 5
3	1 6 2 3 4 5
4	7 6 8 8 9 10
5	7 7 11 11 9 10

6	12	13	14	11	9	10
7	12	13	14	15	15	15
8	36*					
9	7+					
10	2-					
11	1-					
12	1-					
13	1-					
14	60*					
15	3/					
16	10*					
17	11+					
18	96*					
19	1-					
20	1-					
21	2-					
22	30*					



Listing C.81: 6x6_16.txt

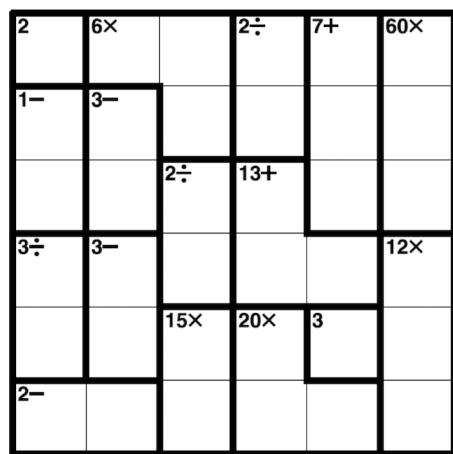
1	6	15				
2	1	1	2	3	3	4
3	5	6	2	7	4	4
4	5	6	7	8	9	
5	10	11	11	12	8	13
6	10	14	12	12	8	13
7	10	14	15	15	15	15
8	5+					
9	1-					
10	2/					
11	20*					
12	4-					
13	10+					
14	1-					
15	15+					
16	4=					
17	11+					
18	3-					
19	8+					
20	5-					
21	2-					
22	15+					



Listing C.82: 6x6_17.txt

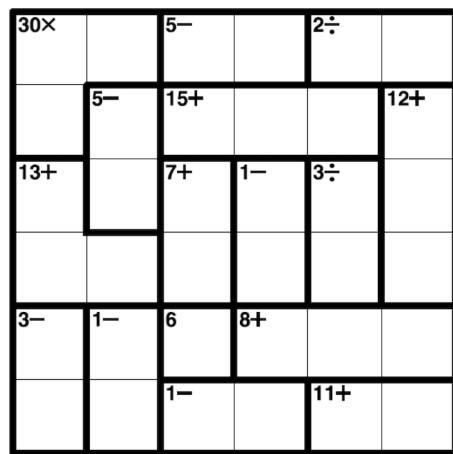
1	6	16				
2	1	2	3	4	5	

3	6	7	2	3	4	5
4	6	7	8	9	4	5
5	10	11	8	9	9	12
6	10	11	13	14	15	12
7	16	16	13	14	14	12
8	2=					
9	6*					
10	2/					
11	7+					
12	60*					
13	1-					
14	3-					
15	2/					
16	13+					
17	3/					
18	3-					
19	12*					
20	15*					
21	20*					
22	3=					
23	2-					



Listing C.83: 6x6_18.txt

1	6	16				
2	1	1	2	2	3	3
3	1	4	5	5	5	6
4	7	4	8	9	10	6
5	7	7	8	9	10	6
6	11	12	13	14	14	14
7	11	12	15	15	16	16
8	30*					
9	5-					
10	2/					
11	5-					
12	15+					
13	12+					
14	13+					
15	7+					
16	1-					
17	3/					
18	3-					
19	1-					
20	6=					
21	8+					
22	1-					
23	11+					

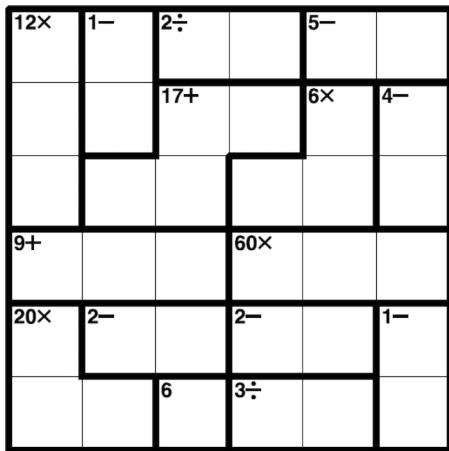
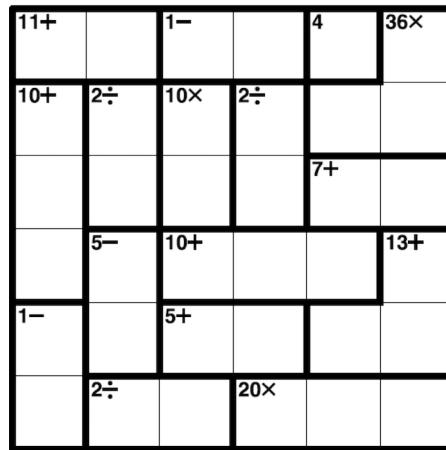


Listing C.84: 6x6_19.txt

```

1| 6 15
2| 1 2 3 3 4 4
3| 1 2 5 5 6 7
4| 1 5 5 6 6 7
5| 8 8 8 9 9 9
6| 10 11 11 12 12 13
7| 10 10 14 15 15 13
8| 12*
9| 1-
10| 2/
11| 5-
12| 17+
13| 6*
14| 4-
15| 9+
16| 60*
17| 20*
18| 2-
19| 2-
20| 1-
21| 6=
22| 3/

```



Listing C.85: 6x6_20.txt

```

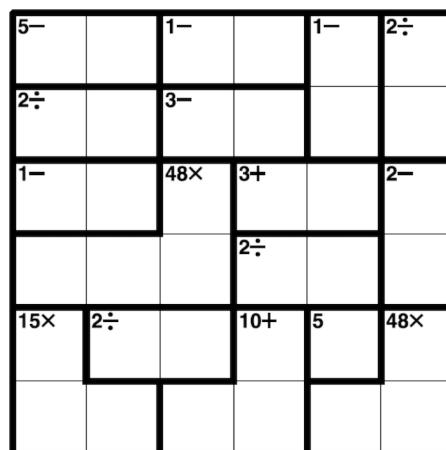
1| 6 16
2| 1 1 2 2 3 4
3| 5 6 7 8 4 4
4| 5 6 7 8 9 9
5| 5 10 11 11 11 12
6| 13 10 14 14 12 12
7| 13 15 15 16 16 16
8| 11+
9| 1-
10| 4=
11| 36*
12| 10+
13| 2/
14| 10*
15| 2/
16| 7+
17| 5-
18| 10+
19| 13+
20| 1-
21| 5+
22| 2/
23| 20*

```

```

1| 6 16
2| 1 1 2 2 3 4
3| 5 5 6 6 3 4
4| 7 7 8 9 9 10
5| 8 8 8 11 11 10
6| 12 13 13 14 15 16
7| 12 12 14 14 16 16
8| 5-
9| 1-
10| 1-
11| 2/
12| 2/
13| 3-
14| 1-
15| 48*
16| 3+
17| 2-
18| 2/
19| 15*
20| 2/
21| 10+
22| 5=
23| 48*

```



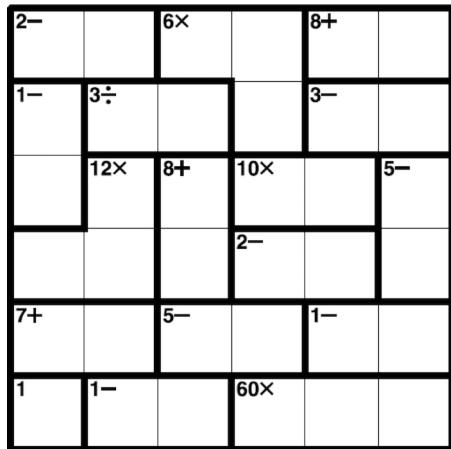
Listing C.87: 6x6_22.txt

```

1| 6 17
2| 1 1 2 2 3 3
3| 4 5 5 2 6 6
4| 4 7 8 9 9 10
5| 7 7 8 11 11 10
6| 12 12 13 13 14 14
7| 15 16 16 17 17 17
8| 2-
9| 6*
10| 8+
11| 1-
12| 3/
13| 3-
14| 12*
15| 8+
16| 10*
17| 5-
18| 2-
19| 7+
20| 5-

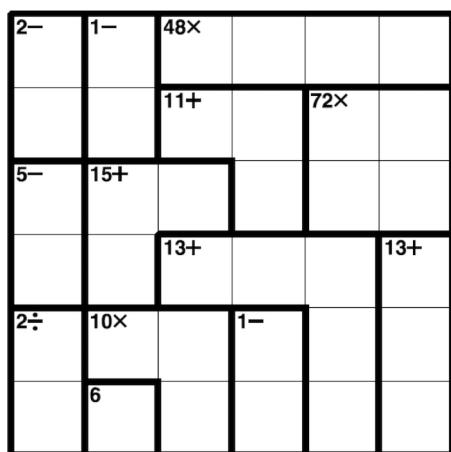
```

21 1-
22 1-
23 1-
24 60*



Listing C.88: 6x6_23.txt

1 6 13
2 1 2 3 3 3 3
3 1 2 4 4 5 5
4 6 7 7 4 5 5
5 6 7 8 8 9
6 10 11 11 12 8 9
7 10 13 11 12 8 9
8 2-
9 1-
10 48*
11 11+
12 72*
13 5-
14 15+
15 13+
16 13+
17 2/
18 10*
19 1-
20 6=



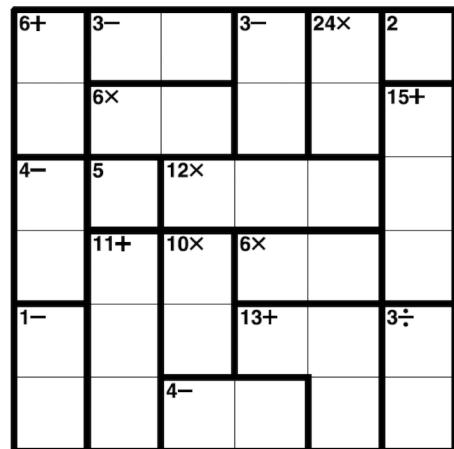
Listing C.89: 6x6_24.txt

1 6 17
2 1 2 2 3 4 5
3 1 6 6 3 4 7
4 8 9 10 10 10 7
5 8 11 12 13 13 7
6 14 11 12 15 15 16
7 14 11 17 17 15 16
8 6+
9 3-
10 3-
11 24*
12 2=

13 6*
14 15+
15 4-
16 5=

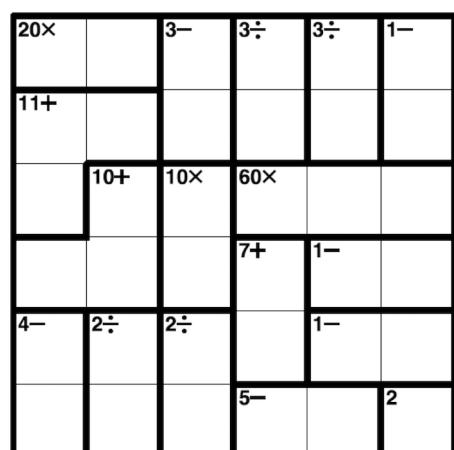
17 12*

18 11+
19 10*
20 6*
21 1-
22 13+
23 3/
24 4-



Listing C.90: 6x6_25.txt

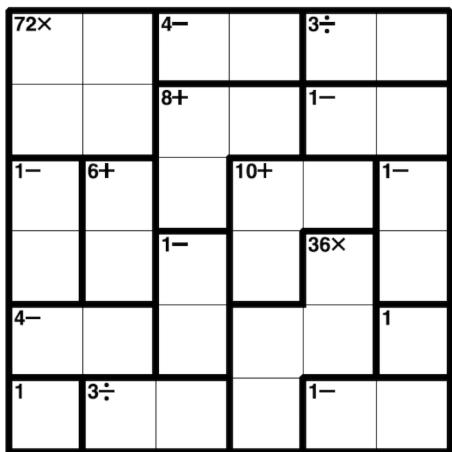
1 6 17
2 1 1 2 3 4 5
3 6 6 2 3 4 5
4 6 7 8 9 9 9
5 7 7 8 10 11 11
6 12 13 14 10 15 15
7 12 13 14 16 16 17
8 20*
9 3-
10 3/
11 3/
12 1-
13 11+
14 10+
15 10*
16 60*
17 7+
18 1-
19 4-
20 2/
21 2/
22 1-
23 5-
24 2=



Listing C.91: 6x6_26.txt

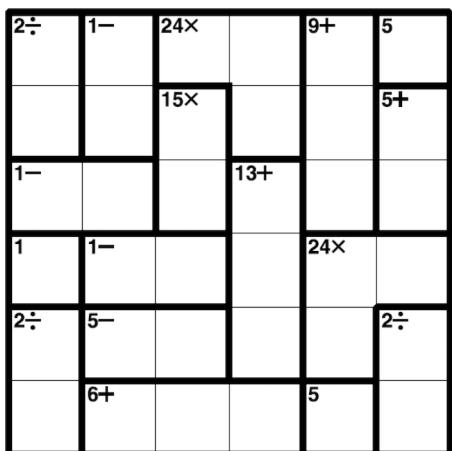
1 6 16
2 1 1 2 2 3 3
3 1 1 4 4 5 5
4 6 7 4 8 8 9
5 6 7 10 8 11 9
6 12 12 10 11 11 13
7 14 15 15 11 16 16
8 72*
9 4-
10 3/

11	8+
12	1-
13	1-
14	6+
15	10+
16	1-
17	1-
18	36*
19	4-
20	1=
21	1=
22	3/
23	1-



Listing C.92: 6x6_27.txt

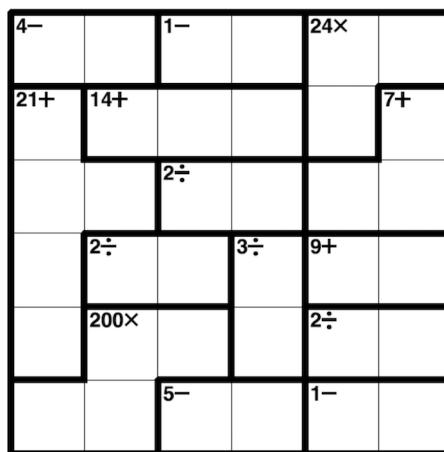
1	6 17
2	1 2 3 3 4 5
3	1 2 6 3 4 7
4	8 8 6 9 4 7
5	10 11 11 9 12 12
6	13 14 14 9 12 15
7	13 16 16 16 17 15
8	2/
9	1-
10	24*
11	9+
12	5=
13	15*
14	5+
15	1-
16	13+
17	1=
18	1-
19	24*
20	2/
21	5-
22	2/
23	6+
24	5=



Listing C.93: 6x6_28.txt

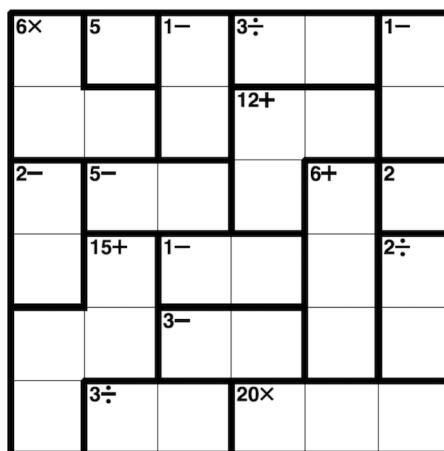
1	6 14
2	1 1 2 2 3 3
3	4 5 5 5 3 6
4	4 4 7 7 6 6

5	4 8 8 9 10 10
6	4 11 11 9 12 12
7	11 11 13 13 14 14
8	4-
9	1-
10	24*
11	21+
12	14+
13	7+
14	2/
15	2/
16	3/
17	9+
18	200*
19	2/
20	5-
21	1-



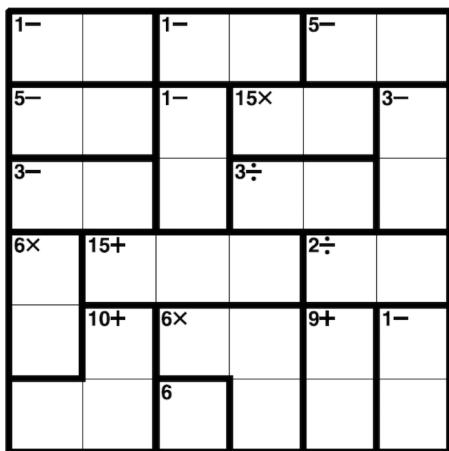
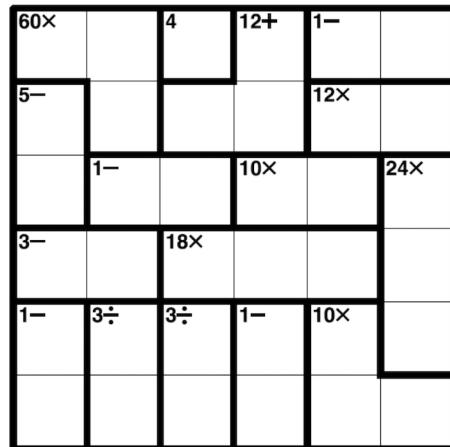
Listing C.94: 6x6_29.txt

1	6 16
2	1 2 3 4 4 5
3	1 1 3 6 6 5
4	7 8 8 6 9 10
5	7 11 12 12 9 13
6	11 11 14 14 9 13
7	11 15 15 16 16 16
8	6*
9	5=
10	1-
11	3/
12	1-
13	12+
14	2-
15	5-
16	6+
17	2=
18	15+
19	1-
20	2/
21	3-
22	3/
23	20*



Listing C.95: 6x6_30.txt

2	1	1	2	2	3	3
3	4	4	5	6	6	7
4	8	8	5	9	9	7
5	10	11	11	11	12	12
6	10	13	14	14	15	16
7	13	13	17	14	15	16
8	1-					
9	1-					
10	5-					
11	5-					
12	1-					
13	15*					
14	3-					
15	3-					
16	3/					
17	6*					
18	15+					
19	2/					
20	10+					
21	6*					
22	9+					
23	1-					
24	6=					

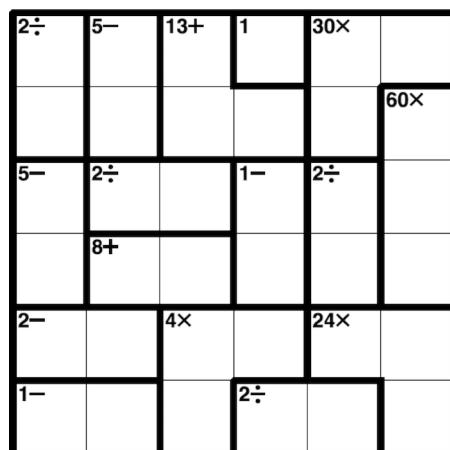


Listing C.96: 6x6_31.txt

1	6	16				
2	1	1	2	3	4	4
3	5	1	3	3	6	6
4	5	7	7	8	8	9
5	10	10	11	11	11	9
6	12	13	14	15	16	9
7	12	13	14	15	16	16
8	60*					
9	4=					
10	12+					
11	1-					
12	5-					
13	12*					
14	1-					
15	10*					
16	24*					
17	3-					
18	18*					
19	1-					
20	3/					
21	3/					
22	1-					
23	10*					

1	6	16				
2	1	2	3	4	5	5
3	1	2	3	3	5	6
4	7	8	8	9	10	6
5	7	11	11	9	10	6
6	12	12	13	13	14	14
7	15	15	13	16	16	14
8	2/					
9	5-					
10	13+					
11	1=					
12	30*					
13	60*					
14	5-					
15	2/					
16	1-					
17	2/					
18	8+					
19	2-					
20	4*					
21	24*					
22	1-					
23	2/					

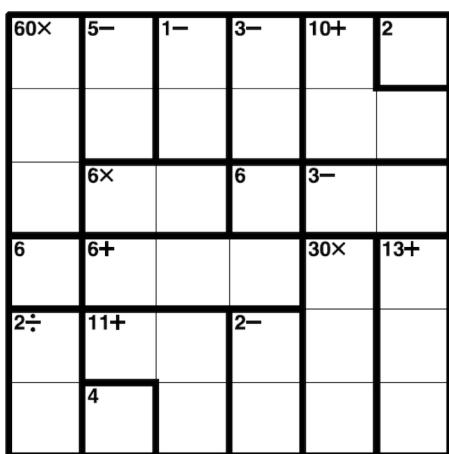
Listing C.97: 6x6_32.txt



Listing C.98: 6x6_33.txt

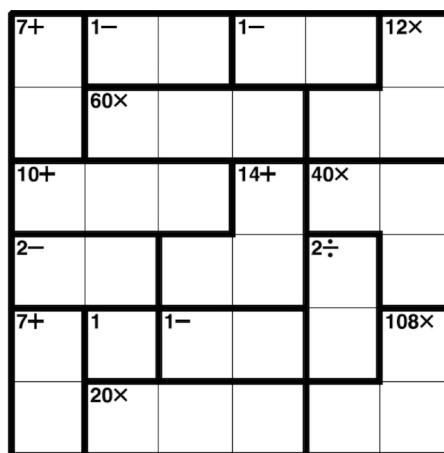
1	6	17				
2	1	2	3	4	5	6
3	1	2	3	4	5	5
4	1	7	7	8	9	9
5	10	11	11	11	12	13
6	14	15	15	16	12	13
7	14	17	15	16	12	13
8	60*					
9	5-					
10	1-					
11	3-					
12	10+					
13	2=					
14	6*					
15	6=					
16	3-					
17	6=					
18	6+					
19	30*					
20	13+					

21 2/
22 11+
23 2-
24 4=



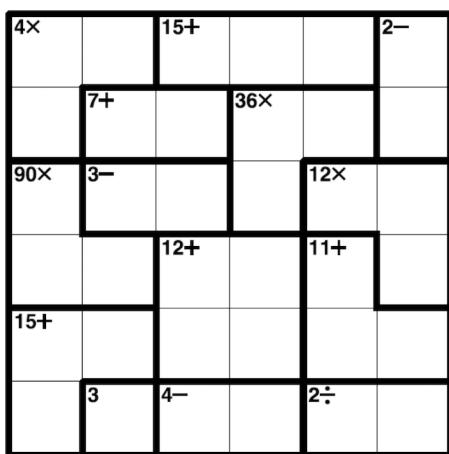
Listing C.99: 6x6_34.txt

17 2/
18 7+
19 1=
20 1-
21 108*
22 20*



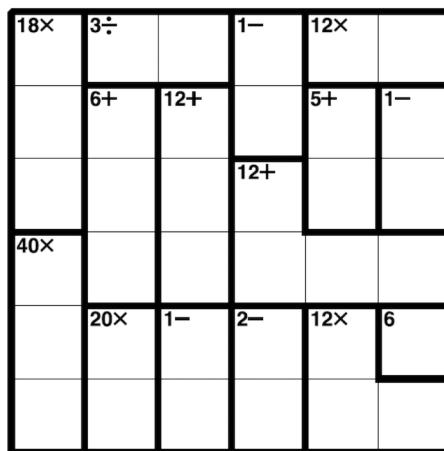
Listing C.101: 6x6_36.txt

1 6 14
2 1 1 2 2 2 3
3 1 4 4 5 5 3
4 6 7 7 5 8 8
5 6 6 9 9 10 8
6 11 11 9 9 10 10
7 11 12 13 13 14 14
8 4*
9 15+
10 2-
11 7+
12 36*
13 90*
14 3-
15 12*
16 12+
17 11+
18 15+
19 3=
20 4-
21 2/



Listing C.100: 6x6_35.txt

1 6 15
2 1 2 2 3 4 4
3 1 5 6 3 7 8
4 1 5 6 9 7 8
5 10 5 6 9 9 9
6 10 11 12 13 14 15
7 10 11 12 13 14 14
8 18*
9 3/
10 1-
11 12*
12 6+
13 12+
14 5+
15 1-
16 12+
17 40*
18 20*
19 1-
20 2-
21 12*
22 6=

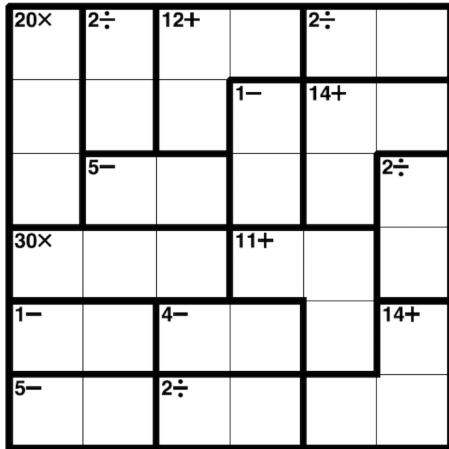


Listing C.102: 6x6_37.txt

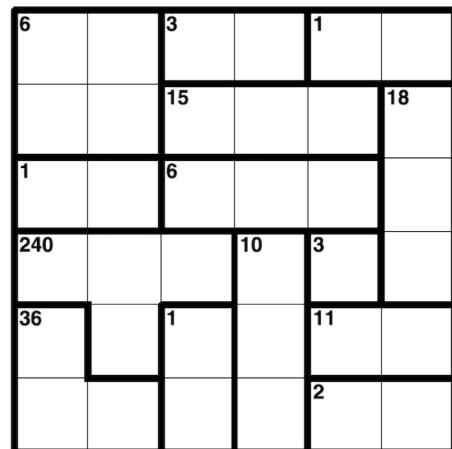
1 6 15
2 1 2 2 3 3 4
3 1 5 5 5 4 4
4 6 6 6 7 8 8
5 9 9 7 7 10 8
6 11 12 13 13 10 14
7 11 15 15 15 14 14
8 7+
9 1-
10 1-
11 12*
12 60*
13 10+
14 14+
15 40*
16 2-

1 6 15
2 1 2 3 3 4 4
3 1 2 3 5 6 6
4 1 7 7 5 6 8
5 9 9 9 10 10 8
6 11 11 12 12 10 13
7 14 14 15 15 13 13
8 20*
9 2/
10 12+
11 2/
12 1-
13 14+

14	5-
15	2/
16	30*
17	11+
18	1-
19	4-
20	14+
21	5-
22	2/

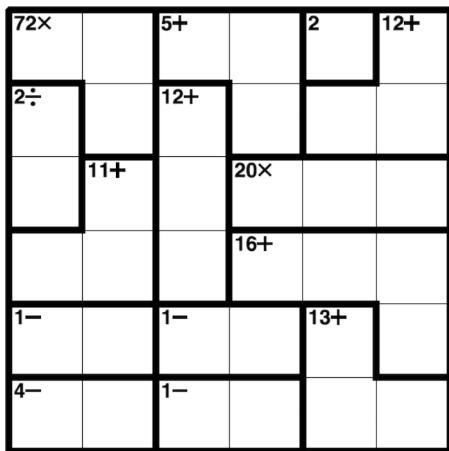


12	18*
13	1-
14	6+
15	240*
16	10+
17	3=
18	36*
19	1-
20	11+
21	2/



Listing C.103: 6x6_38.txt

1	6 14
2	1 1 2 2 3 4
3	5 1 6 2 4 4
4	5 7 6 8 8 8
5	7 7 6 9 9 9
6	10 10 11 11 12 9
7	13 13 14 14 12 12
8	72*
9	5+
10	2=
11	12+
12	2/
13	12+
14	11+
15	20*
16	16+
17	1-
18	1-
19	13+
20	4-
21	1-



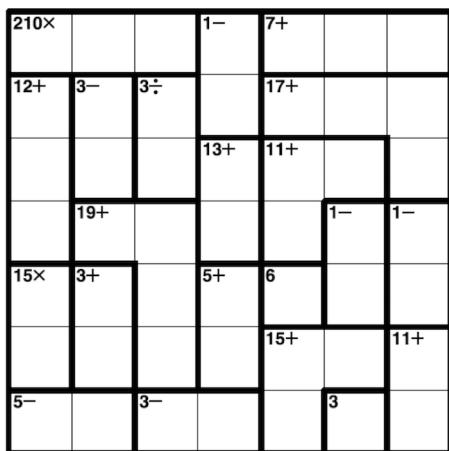
Listing C.104: 6x6_39.txt

1	6 14
2	1 1 2 2 3 3
3	1 1 4 4 4 5
4	6 6 7 7 7 5
5	8 8 8 9 10 5
6	11 8 12 9 13 13
7	11 11 12 9 14 14
8	6*
9	3/
10	1-
11	15+

C.4 File Teks Soal-Soal Permainan Calcudoku dengan *Grid* Berukuran 7×7

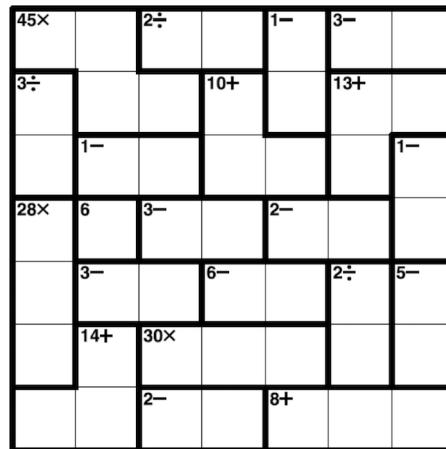
Listing C.105: 7x7_1.txt

1	7	21
2	1	1 1 2 3 3 3
3	4	5 6 2 7 7 7
4	4	5 6 8 9 9 7
5	4	10 10 8 9 11 12
6	13	14 10 15 16 11 12
7	13	14 10 15 17 17 18
8	19	19 20 20 17 21 18
9	21*	
10	1-	
11	7+	
12	12+	
13	3-	
14	3/	
15	17+	
16	13+	
17	11+	
18	19+	
19	1-	
20	1-	
21	15*	
22	3+	
23	5+	
24	6=	
25	15+	
26	11+	
27	5-	
28	3-	
29	3=	



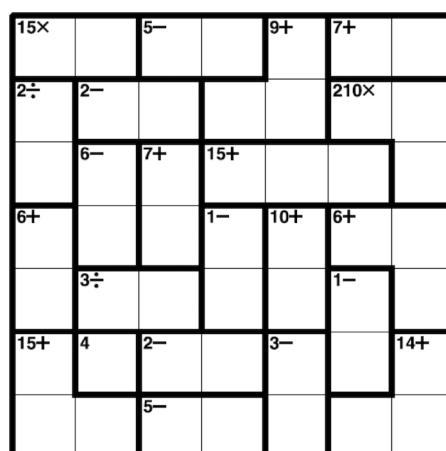
Listing C.106: 7x7_2.txt

1	7	21
2	1	1 2 2 3 4 4
3	5	1 1 6 3 7 7
4	5	8 8 6 6 7 9
5	10	11 12 12 13 13 9
6	10	14 14 15 15 16 17
7	10	18 19 19 19 16 17
8	18	18 20 20 21 21 21
9	45*	
10	2/	
11	1-	
12	3-	
13	3/	
14	10+	
15	13+	
16	1-	
17	1-	
18	28*	
19	6=	
20	3-	
21	2-	
22	3-	
23	6-	
24	2/	
25	5-	
26	14+	
27	30*	
28	2-	
29	8+	



Listing C.107: 7x7_3.txt

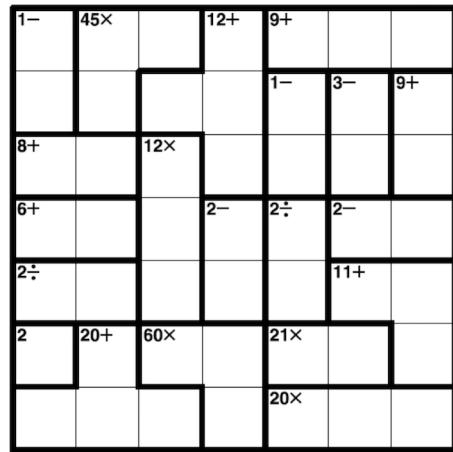
1	7	22					
2	1	1	2	2	3	4	4
3	5	6	6	3	3	7	7
4	5	8	9	10	10	10	7
5	11	8	9	12	13	14	14
6	11	15	15	12	13	16	14
7	17	18	19	19	20	16	21
8	17	17	22	22	20	21	21
9	15	*					
10	5-						
11	9+						
12	7+						
13	2/						
14	2-						
15	210*						
16	6-						
17	7+						
18	15+						
19	6+						
20	1-						
21	10+						
22	6+						
23	3/						
24	1-						
25	15+						
26	4=						
27	2-						
28	3-						
29	14+						
30	5-						



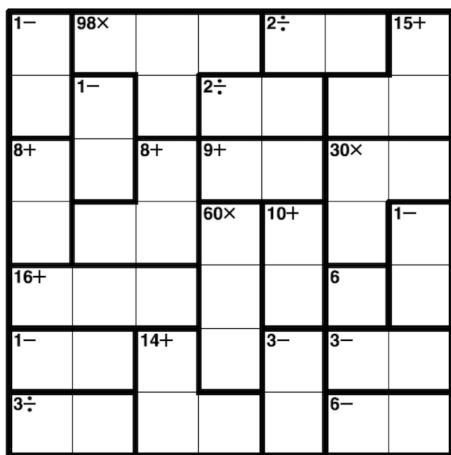
Listing C.108: 7x7 4.txt

1	7	21					
2	1	2	2	2	3	3	4
3	1	5	2	6	6	4	4
4	7	5	8	9	9	10	10

5	7 8 8 11 12 10 13
6	14 14 14 11 12 15 13
7	16 16 17 11 18 19 19
8	20 20 17 17 18 21 21
9	1-
10	98*
11	2/
12	15+
13	1-
14	2/
15	8+
16	8+
17	9+
18	30*
19	60*
20	10+
21	1-
22	16+
23	6=
24	1-
25	14+
26	3-
27	3-
28	3/
29	6-



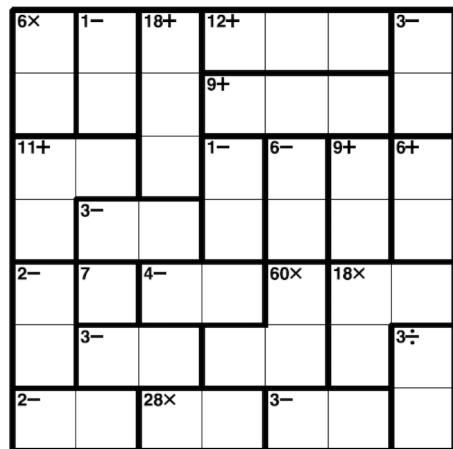
Listing C.110: 7x7_6.txt



Listing C.109: 7x7_5.txt

1	7 20
2	1 2 2 3 4 4 4
3	1 2 3 3 5 6 7
4	8 8 9 3 5 6 7
5	10 10 9 11 12 13 13
6	14 14 9 11 12 15 15
7	16 17 18 18 19 19 15
8	17 17 17 18 20 20 20
9	1-
10	45*
11	12+
12	9+
13	1-
14	3-
15	9+
16	8+
17	12*
18	6+
19	2-
20	2/
21	2-
22	2/
23	11+
24	2=
25	20+
26	60*
27	21*
28	20*

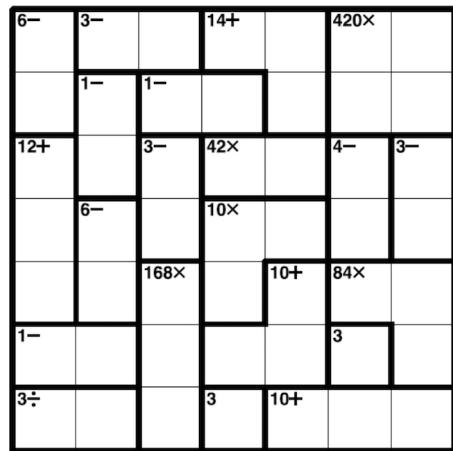
1	7 22
2	1 2 3 4 4 4 5
3	1 2 3 6 6 6 5
4	7 7 3 8 9 10 11
5	7 12 12 8 9 10 11
6	13 14 15 15 16 17 17
7	13 18 18 16 16 17 19
8	20 20 21 21 22 22 19
9	6*
10	1-
11	18+
12	12+
13	3-
14	9+
15	11+
16	1-
17	6-
18	9+
19	6+
20	3-
21	2-
22	7=
23	4-
24	60*
25	18*
26	3-
27	3/
28	2-
29	28*
30	3-



Listing C.111: 7x7_7.txt

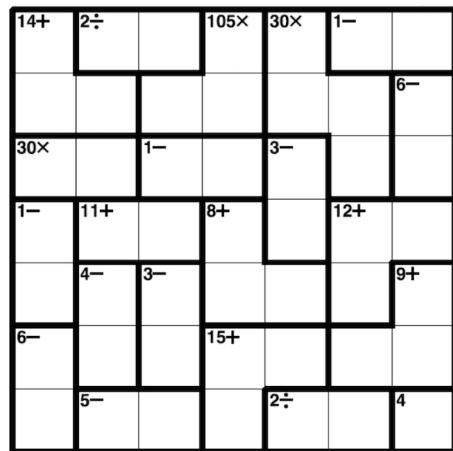
1	7 21
2	1 2 2 3 3 4 4
3	1 5 6 6 3 4 4
4	7 5 8 9 9 10 11
5	7 12 8 13 13 10 11
6	7 12 14 13 15 16 16
7	17 17 14 15 15 18 16
8	19 19 14 20 21 21 21
9	6-
10	3-
11	14+
12	420*
13	1-

14 1-
15 12+
16 3-
17 42*
18 4-
19 3-
20 6-
21 10*
22 168*
23 10+
24 84*
25 1-
26 3=



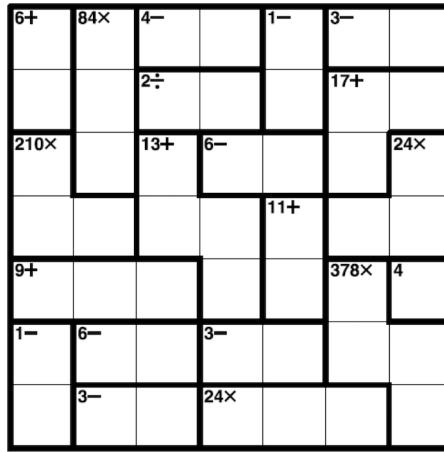
Listing C.112: 7x7_8.txt

1 7 21
2 1 2 2 3 4 5 5
3 1 1 3 3 4 4 6
4 7 7 8 8 9 4 6
5 10 11 11 12 9 13 13
6 10 14 15 12 12 13 16
7 17 14 15 18 18 16 16
8 17 19 19 18 20 20 21
9 14+
10 2/
11 105*
12 30*
13 1-
14 6-
15 30*
16 1-
17 3-
18 1-
19 11+
20 8+
21 12+
22 4-
23 3-
24 9+
25 6-
26 15+
27 5-
28 2/
29 4=



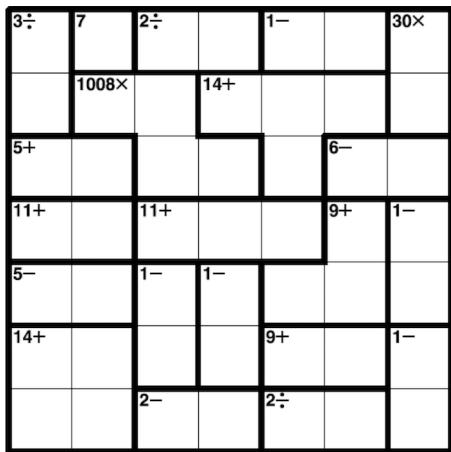
Listing C.113: 7x7_9.txt

1 7 20
2 1 2 3 3 4 5 5
3 1 2 6 6 4 7 7
4 8 2 9 10 10 7 11
5 8 8 9 9 12 11 11
6 13 13 13 9 12 14 15
7 16 17 17 18 18 14 14
8 16 19 19 20 20 20 14
9 6+
10 84*
11 4-
12 1-
13 3-
14 2/
15 17+
16 210*
17 13+
18 6-
19 24*
20 11+
21 9+
22 378*
23 4=



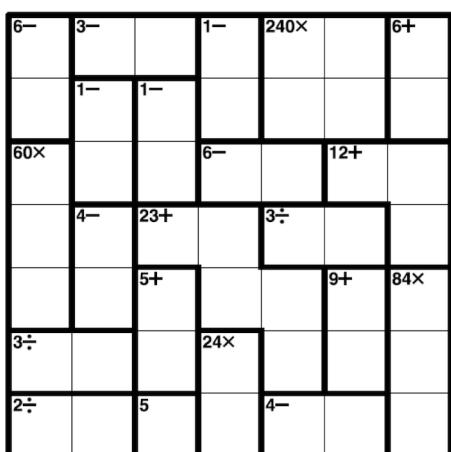
Listing C.114: 7x7_10.txt

1 7 21
2 1 2 3 3 4 4 5
3 1 6 6 7 7 7 5
4 8 8 6 6 7 9 9
5 10 10 11 11 11 12 13
6 14 14 15 16 12 12 13
7 17 17 15 16 18 18 19
8 17 17 20 20 21 21 19
9 3/
10 7=



Listing C.115: 7x7_11.txt

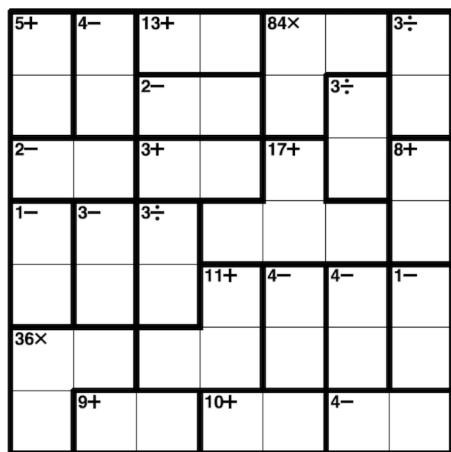
1 7 21
 2 1 2 2 3 4 4 5
 3 1 6 7 3 4 4 5
 4 8 6 7 9 9 10 10
 5 8 11 12 12 13 13 10
 6 8 11 14 12 12 15 16
 7 17 17 14 18 12 15 16
 8 19 19 20 18 21 21 16
 9 6-
 10 3-
 11 1-
 12 240*
 13 6+
 14 1-
 15 1-
 16 60*
 17 6-
 18 12+
 19 4-
 20 23+
 21 3/
 22 5+
 23 9+
 24 84*
 25 3/
 26 24*
 27 2/
 28 5=
 29 4-



Listing C.116: 7x7_12.txt

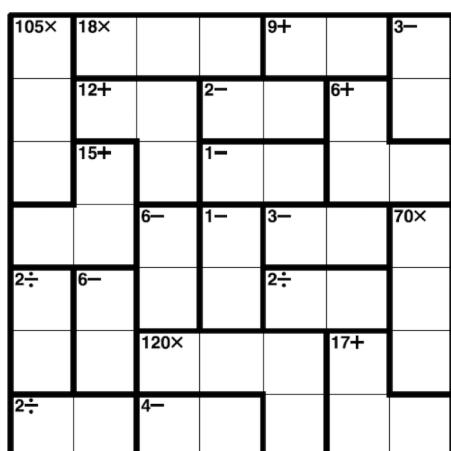
1 7 22
 2 1 2 3 3 4 4 5
 3 1 2 6 6 4 7 5
 4 8 8 9 9 10 7 11
 5 12 13 14 10 10 10 11
 6 12 13 14 15 16 17 18
 7 19 19 15 15 16 17 18
 8 19 20 20 21 21 22 22
 9 5+
 10 4-
 11 13+
 12 84*
 13 3/
 14 2-

15 3/
 16 2-
 17 3+
 18 17+
 19 8+
 20 1-
 21 3-
 22 3/
 23 11+
 24 4-
 25 4-
 26 1-
 27 36*
 28 9+
 29 10+
 30 4-



Listing C.117: 7x7_13.txt

1 7 20
 2 1 2 2 2 3 3 4
 3 1 5 5 6 6 7 4
 4 1 8 5 9 9 7 7
 5 8 8 10 11 12 12 13
 6 14 15 10 11 16 16 13
 7 14 15 17 17 17 18 13
 8 19 19 20 20 17 18 18
 9 105*
 10 18*
 11 9+
 12 3-
 13 12+
 14 2-
 15 6+
 16 15+
 17 1-
 18 6-
 19 1-
 20 3-
 21 70*
 22 2/
 23 6-
 24 2/
 25 120*
 26 17+
 27 2/
 28 4-

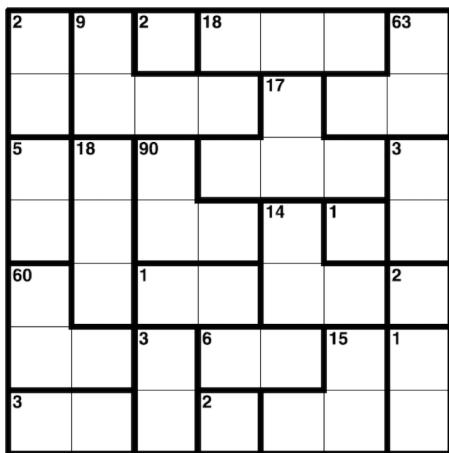
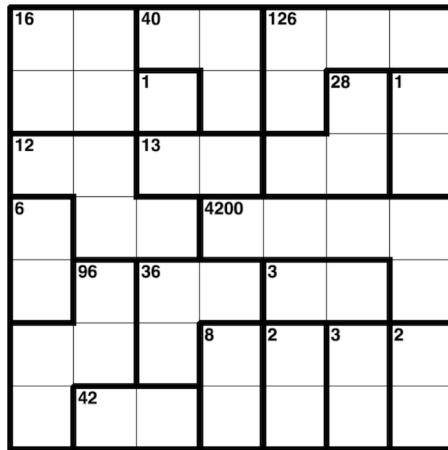


Listing C.118: 7x7_14.txt

```

1 7 21
2 1 2 3 4 4 4 5
3 1 2 2 2 6 5 5
4 7 8 9 6 6 6 10
5 7 8 9 9 11 12 10
6 13 8 14 14 11 11 15
7 13 13 16 17 17 18 19
8 20 20 16 21 18 18 19
9 2-
10 9+
11 2=
12 18+
13 63*
14 17+
15 5-
16 18+
17 96*
18 3-
19 14*
20 1=
21 60*
22 1-
23 2=
24 3-
25 6+
26 15+
27 1-
28 3/
29 2=

```



Listing C.119: 7x7_15.txt

```

1 7 18
2 1 1 2 2 3 3 3
3 1 1 4 2 3 5 6
4 7 7 8 8 5 5 6
5 9 7 7 10 10 10 10
6 9 11 12 12 13 13 10
7 11 11 12 14 15 16 17
8 11 18 18 14 15 16 17
9 16+
10 40*
11 126*
12 1=
13 28*
14 1-
15 12+
16 13+
17 6-
18 4200*
19 96*
20 36*
21 3/
22 8+
23 2-
24 3-
25 2/
26 42*

```

C.5 File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran 8×8

Listing C.120: 8x8_1.txt

```

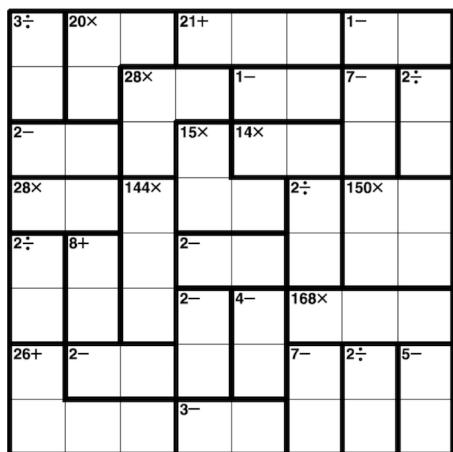
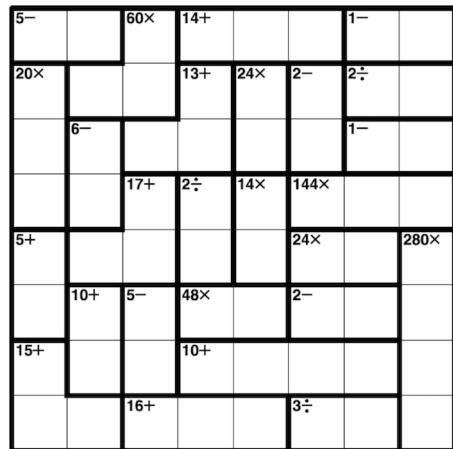
1| 8 27
2| 1 2 2 3 3 3 4 4
3| 1 2 5 5 6 6 7 8
4| 9 9 5 10 11 11 7 8
5| 12 12 13 10 10 14 15 15
6| 16 17 13 18 18 14 15 15
7| 16 17 13 19 20 21 21 21
8| 22 23 23 19 20 24 25 26
9| 22 22 22 27 27 24 25 26
10| 3/
11| 20*
12| 21+
13| 1-
14| 28*
15| 1-
16| 7-
17| 2/
18| 2-
19| 15*
20| 14*
21| 28*
22| 144*
23| 2/
24| 150*
25| 2/
26| 8+
27| 2-
28| 2-
29| 4-
30| 168*
31| 26+
32| 2-
33| 7-
34| 2/
35| 5-
36| 3-

```

```

24| 144*
25| 5+
26| 24*
27| 280*
28| 10+
29| 5-
30| 48*
31| 2-
32| 15+
33| 10+
34| 16+
35| 3/

```



Listing C.121: 8x8_2.txt

```

1| 8 26
2| 1 1 2 3 3 3 4 4
3| 5 2 2 6 7 8 9 9
4| 5 10 6 6 7 8 11 11
5| 5 10 12 13 14 15 15 15
6| 16 12 12 13 14 17 17 18
7| 16 19 20 21 21 22 22 18
8| 23 19 20 24 24 24 24 18
9| 23 23 25 25 25 26 26 18
10| 5-
11| 60*
12| 14+
13| 1-
14| 20*
15| 13+
16| 24*
17| 2-
18| 2/
19| 6-
20| 1-
21| 17+
22| 2/
23| 14*

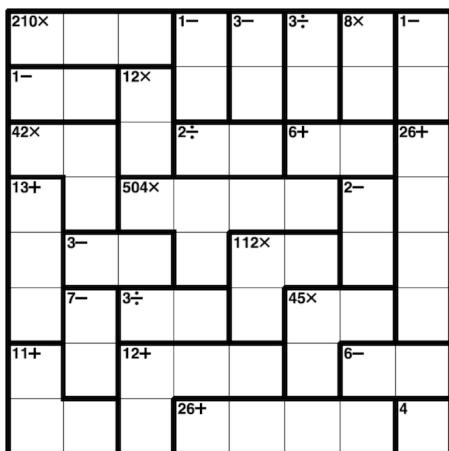
```

```

1| 8 25
2| 1 1 1 2 3 4 5 6
3| 7 7 8 2 3 4 5 6
4| 9 9 8 10 10 11 11 12
5| 13 9 14 14 14 14 15 12
6| 13 16 16 14 17 17 15 12
7| 13 18 19 19 17 20 20 12
8| 21 18 22 22 22 20 23 23
9| 21 21 22 24 24 24 24 25
10| 210*
11| 1-
12| 3-
13| 3/
14| 8*
15| 1-
16| 1-
17| 12*
18| 42*
19| 2/
20| 6+
21| 26+
22| 13+
23| 504*
24| 2-
25| 3-
26| 112*
27| 7-
28| 3/
29| 45*
30| 11+
31| 12+
32| 6-
33| 26+
34| 4=

```

Listing C.122: 8x8_3.txt

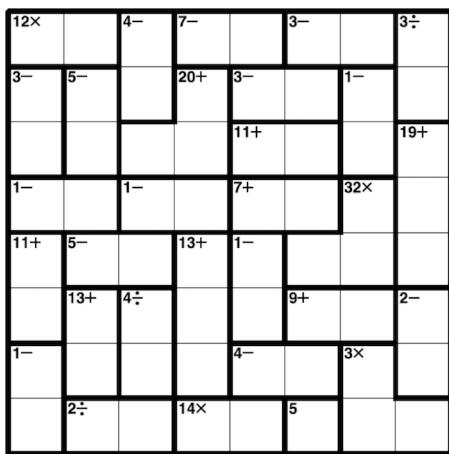


Listing C.123: 8x8_4.txt

```

1| 8 30
2| 1 1 2 3 3 4 4 5
3| 6 7 2 8 9 9 10 5
4| 6 7 8 8 11 11 10 12
5| 13 13 14 14 15 15 16 12
6| 17 18 18 19 20 16 16 12
7| 17 21 22 19 20 23 23 24
8| 25 21 22 19 26 26 27 24
9| 25 28 28 29 29 30 27 27
10| 12*
11| 4-
12| 7-
13| 3-
14| 3/
15| 3-
16| 5-
17| 20+
18| 3-
19| 1-
20| 11+
21| 19+
22| 1-
23| 1-
24| 7+
25| 32*
26| 11+
27| 5-
28| 13+
29| 1-
30| 13+
31| 4/
32| 9+
33| 2-
34| 1-
35| 4-
36| 3*
37| 2/
38| 14*
39| 5=

```



Listing C.124: 8x8_5.txt

```

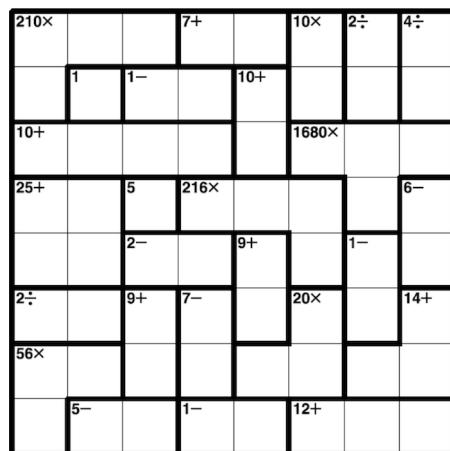
1| 8 26
2| 1 1 1 2 2 3 4 5
3| 1 6 7 7 8 3 4 5
4| 9 9 9 9 8 10 10 10

```

```

5| 11 11 12 13 13 13 10 14
6| 11 11 15 15 16 13 17 14
7| 18 18 19 20 16 21 17 22
8| 23 23 19 20 21 21 22 22
9| 23 24 24 25 25 26 26 26
10| 210*
11| 7+
12| 10*
13| 2/
14| 4/
15| 1=
16| 1-
17| 10+
18| 10+
19| 1680*
20| 25+
21| 5=
22| 216*
23| 6-
24| 2-
25| 9+
26| 1-
27| 2/
28| 9+
29| 7-
30| 20*
31| 14+
32| 56*
33| 5-
34| 1-
35| 12+

```

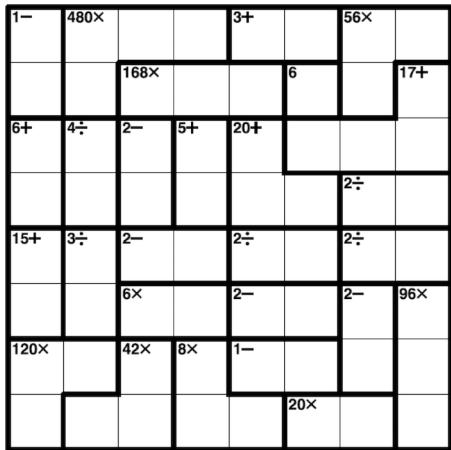


Listing C.125: 8x8_6.txt

```

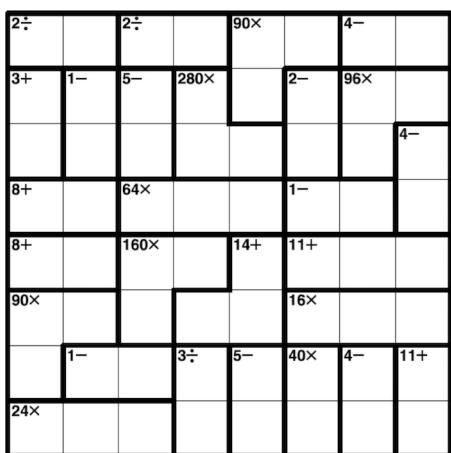
1| 8 27
2| 1 2 2 2 3 3 4 4
3| 1 2 5 5 5 6 4 7
4| 8 9 10 11 12 7 7 7
5| 8 9 10 11 12 12 13 13
6| 14 15 16 16 17 17 18 18
7| 14 15 19 19 20 20 21 22
8| 23 23 24 25 26 26 21 22
9| 23 24 24 25 25 27 27 22
10| 1-
11| 480*
12| 3+
13| 56*
14| 168*
15| 6=
16| 17+
17| 6+
18| 4/
19| 2-
20| 5+
21| 20+
22| 2/
23| 15+
24| 3/
25| 2-
26| 2/
27| 2/
28| 6*
29| 2-
30| 2-
31| 96*
32| 120*
33| 42*
34| 8*
35| 1-
36| 20*

```



Listing C.126: 8x8_7.txt

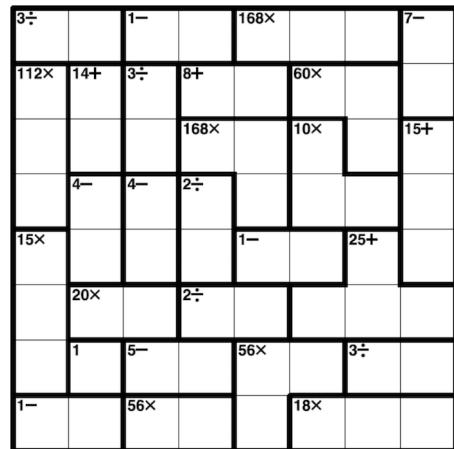
1	8	27						
2	1	1	2	2	3	3	4	4
3	5	6	7	8	3	9	10	10
4	5	6	7	8	9	10	11	
5	12	12	13	13	13	14	14	11
6	15	15	16	16	17	18	18	18
7	19	19	16	17	17	20	20	20
8	19	21	21	22	23	24	25	26
9	27	27	27	22	23	24	25	26
10	2/							
11	2/							
12	90*							
13	4-							
14	3+							
15	1-							
16	5-							
17	280*							
18	2-							
19	96*							
20	4-							
21	8+							
22	64*							
23	1-							
24	8+							
25	160*							
26	14+							
27	11+							
28	90*							
29	16*							
30	1-							
31	3/							
32	5-							
33	40*							
34	4-							
35	11+							
36	24*							



Listing C.127: 8x8_8.txt

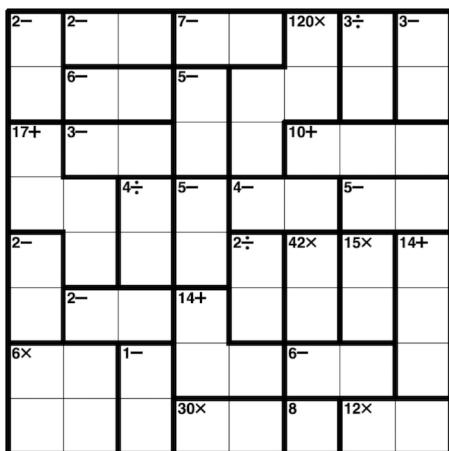
1	8	27						
2	1	1	2	2	3	3	3	4
3	5	6	7	8	8	9	9	4
4	5	6	7	10	10	11	9	12
5	5	13	14	15	10	11	11	12
6	16	13	14	15	17	17	18	12
7	16	19	19	20	20	18	18	18

8	16	21	22	22	23	23	24	24
9	25	25	26	26	23	27	27	27
10	3/							
11	1-							
12	168*							
13	7-							
14	112*							
15	14+							
16	3/							
17	8+							
18	60*							
19	168*							
20	10*							
21	15+							
22	4-							
23	4-							
24	2/							
25	15*							
26	1-							
27	25+							
28	20*							
29	2/							
30	1=							
31	5-							
32	56*							
33	3/							
34	1-							
35	56*							
36	18*							



Listing C.128: 8x8_9.txt

1	8	28						
2	1	2	2	3	3	4	5	6
3	1	7	7	8	4	4	5	6
4	9	10	10	8	4	11	11	11
5	9	9	12	13	14	14	15	15
6	16	9	12	13	17	18	19	20
7	16	21	21	22	17	18	19	20
8	23	23	24	22	22	25	25	20
9	23	23	24	26	26	27	28	28
10	2-							
11	2-							
12	7-							
13	120*							
14	3/							
15	3-							
16	6-							
17	5-							
18	17+							
19	3-							
20	10+							
21	4/							
22	5-							
23	4-							
24	5-							
25	2-							
26	2/							
27	42*							
28	15*							
29	14+							
30	2-							
31	14+							
32	6*							
33	1-							
34	6-							
35	30*							
36	8=							
37	12*							

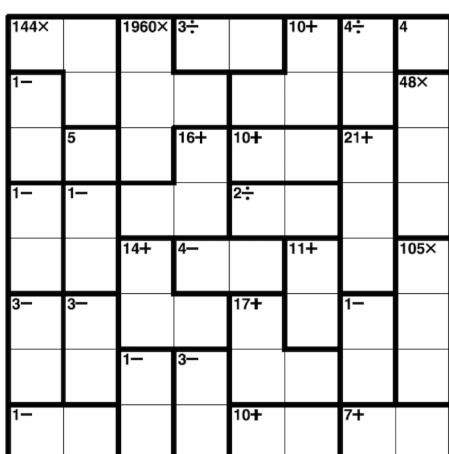


Listing C.129: 8x8_10.txt

```

1| 8 28
2| 1 1 2 3 3 4 5 6
3| 7 1 2 2 4 5 8
4| 7 9 2 10 11 11 12 8
5| 13 14 10 10 15 15 12 8
6| 13 14 16 17 17 18 12 19
7| 20 21 16 16 22 18 23 19
8| 20 21 24 25 22 22 23 19
9| 26 26 24 25 27 27 28 28
10| 144*
11| 1960*
12| 3/
13| 10+
14| 4/
15| 4=
16| 1-
17| 48*
18| 5=
19| 16+
20| 18+
21| 21+
22| 1-
23| 1-
24| 2/
25| 14+
26| 4-
27| 11+
28| 165*
29| 3-
30| 3-
31| 17+
32| 1-
33| 1-
34| 3-
35| 1-
36| 10+
37| 7+

```



Listing C.130: 8x8_11.txt

```

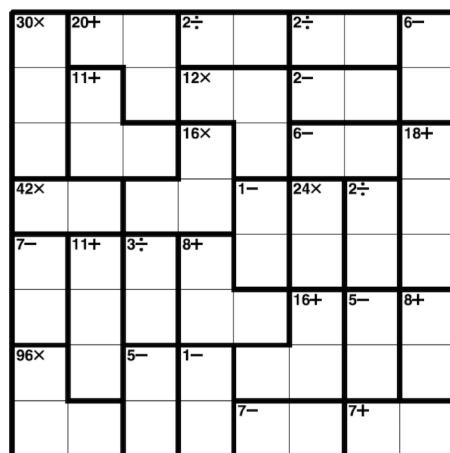
1| 8 27
2| 1 2 2 3 3 4 4 5
3| 1 6 2 7 7 8 8 5
4| 1 6 6 9 7 10 10 11
5| 12 12 9 9 13 14 15 11
6| 16 17 18 19 13 14 15 11

```

```

7| 16 17 18 19 19 20 21 22
8| 23 17 24 25 20 20 21 22
9| 23 23 24 25 26 26 27 27
10| 30*
11| 20+
12| 2/
13| 2/
14| 6-
15| 11+
16| 12*
17| 2-
18| 16*
19| 6-
20| 18+
21| 42*
22| 1-
23| 24*
24| 2/
25| 7-
26| 11+
27| 3/
28| 8+
29| 16+
30| 5-
31| 8+
32| 96*
33| 5-
34| 1-
35| 7-
36| 7+

```

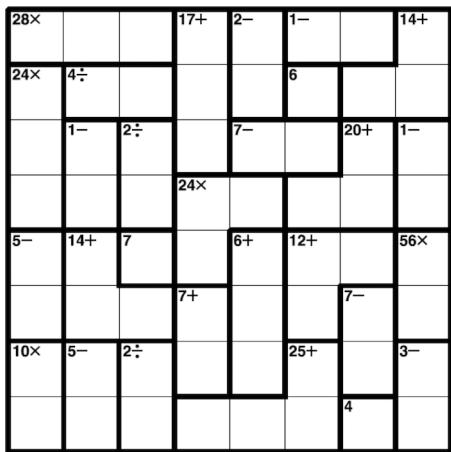


Listing C.131: 8x8_12.txt

```

1| 8 28
2| 1 1 1 2 3 4 4 5
3| 6 7 7 2 3 8 5 5
4| 6 9 10 2 11 11 12 13
5| 6 9 10 14 14 12 12 13
6| 15 16 17 14 18 19 19 20
7| 15 16 16 21 18 19 22 20
8| 23 24 25 21 18 26 22 27
9| 23 24 25 26 26 26 28 27
10| 28*
11| 17+
12| 2-
13| 1-
14| 14+
15| 24*
16| 4/
17| 6=
18| 1-
19| 2/
20| 7-
21| 20+
22| 1-
23| 24*
24| 5-
25| 14+
26| 7=
27| 6+
28| 12+
29| 56*
30| 7+
31| 7-
32| 10*
33| 5-
34| 2/
35| 25+
36| 3-
37| 4=

```

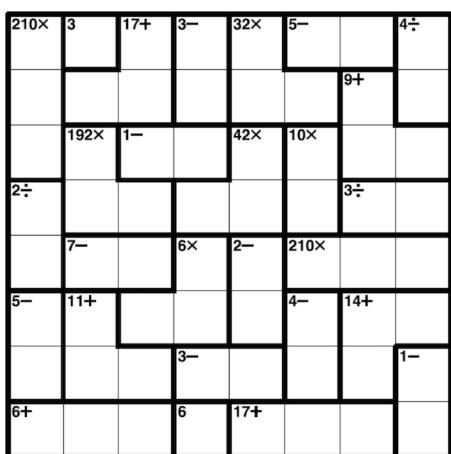


Listing C.132: 8x8_13.txt

```

1| 8 27
2| 1 2 3 4 5 6 6 7
3| 1 3 3 4 5 5 8 7
4| 1 9 10 10 11 12 8 8
5| 13 9 11 11 12 14 14
6| 13 15 15 16 17 18 18 18
7| 19 20 16 16 17 21 22 22
8| 19 20 20 23 23 21 22 24
9| 25 25 25 26 27 27 27 24
10| 210*
11| 3=
12| 17+
13| 3-
14| 32*
15| 5-
16| 4/
17| 9+
18| 192*
19| 1-
20| 42*
21| 10*
22| 2/
23| 3/
24| 7-
25| 6*
26| 2-
27| 210*
28| 5-
29| 11+
30| 4-
31| 14+
32| 3-
33| 1-
34| 6+
35| 6=
36| 17+

```



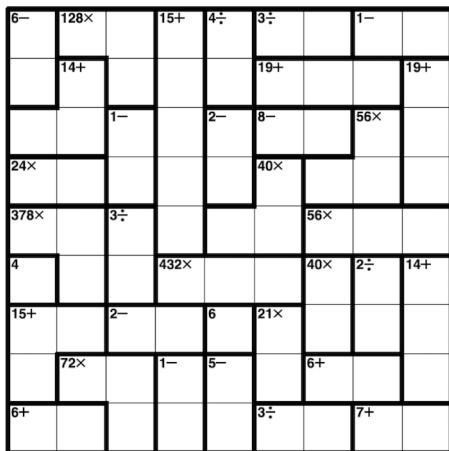
C.6 File Teks Soal-Soal Permainan Calcudoku dengan *Grid* Berukuran 9×9

Listing C.133: 9x9_1.txt

```

1| 9 34
2| 1 2 2 3 4 5 5 6 6
3| 1 7 2 3 4 8 8 8 9
4| 7 7 10 3 11 12 12 13 9
5| 14 14 10 3 11 15 13 13 9
6| 16 16 17 3 15 15 18 18 18
7| 19 16 17 20 20 20 21 22 23
8| 24 24 25 25 26 27 21 22 23
9| 24 28 28 29 30 27 31 31 23
10| 32 32 28 29 30 33 33 34 34
11| 6-
12| 128*
13| 15+
14| 4/
15| 3/
16| 1-
17| 14+
18| 19+
19| 19+
20| 1-
21| 2-
22| 8-
23| 56*
24| 24*
25| 40*
26| 378*
27| 3/
28| 56*
29| 4=
30| 432*
31| 46*
32| 2/
33| 14+
34| 15+
35| 2-
36| 6=
37| 21*
38| 72*
39| 1-
40| 5-
41| 6+
42| 6+
43| 3/
44| 7+

```



Listing C.134: 9x9_2.txt

```

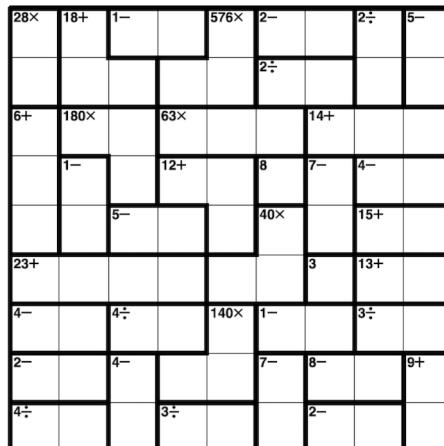
1| 9 36
2| 1 2 3 3 4 5 5 6 7
3| 1 2 2 4 4 8 8 6 7
4| 9 10 10 11 11 11 12 12 12
5| 9 13 10 14 14 15 16 17 17
6| 9 13 18 18 14 19 16 20 20
7| 21 21 21 21 19 19 22 23 23
8| 24 24 25 25 26 27 27 28 28
9| 29 29 30 26 26 31 32 32 33
10| 34 34 30 35 35 31 36 36 33
11| 28*
12| 18+
13| 1-
14| 576*
15| 2-

```

```

16| 2/
17| 5-
18| 2/
19| 6+
20| 180*
21| 63*
22| 14+
23| 1-
24| 12+
25| 8-
26| 7-
27| 4-
28| 5-
29| 40*
30| 15+
31| 23+
32| 3=
33| 13+
34| 4-
35| 4/
36| 140*
37| 1-
38| 3/
39| 2-
40| 4-
41| 7-
42| 8-
43| 9+
44| 4/
45| 3/
46| 2-

```



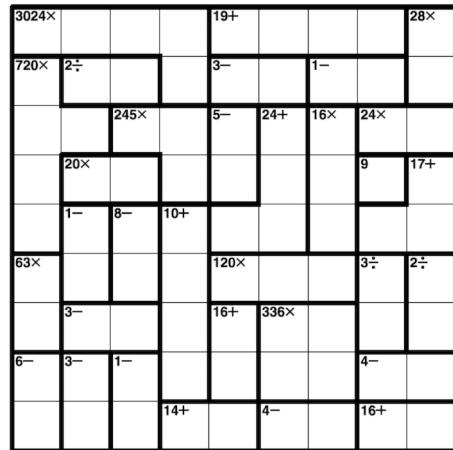
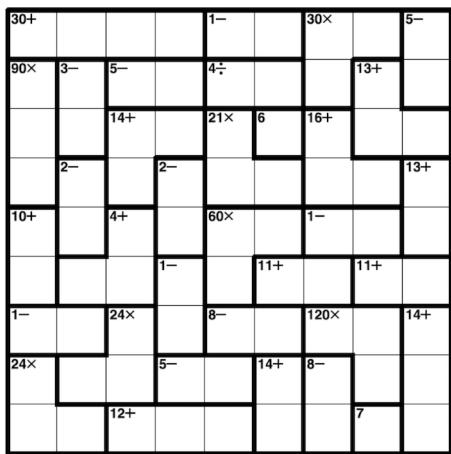
Listing C.135: 9x9_3.txt

```

1| 9 34
2| 1 1 1 1 2 2 3 3 4
3| 5 6 7 7 8 8 3 9 4
4| 5 6 10 10 11 12 13 9 9
5| 5 14 10 15 11 11 13 13 16
6| 17 14 18 15 19 19 20 20 16
7| 17 18 18 21 19 22 22 23 23
8| 24 24 25 21 26 26 27 27 28
9| 29 25 25 30 30 31 32 27 28
10| 29 29 33 33 33 31 32 34 28
11| 30+
12| 1-
13| 30*
14| 5-
15| 90*
16| 3-
17| 5-
18| 4/
19| 13+
20| 14+
21| 21*
22| 6=
23| 16+
24| 2-
25| 2-
26| 13+
27| 10+
28| 4+
29| 60*
30| 1-

```

31 1-
 32 11+
 33 11+
 34 1-
 35 24*
 36 8-
 37 120*x
 38 14+
 39 24*
 40 5-
 41 14+
 42 8-
 43 12+
 44 7=



Listing C.136: 9x9_4.txt

```

1| 9 32
2| 1 1 1 1 2 2 2 2 3
3| 4 5 5 1 6 6 7 7 3
4| 4 4 8 8 9 10 11 12 12
5| 4 13 13 8 9 10 11 14 15
6| 4 16 17 18 10 10 11 15 15
7| 19 16 17 18 20 20 20 21 22
8| 19 23 23 18 24 25 25 21 22
9| 26 27 28 18 24 25 25 29 29
10| 26 27 28 30 30 31 31 32 32
11| 3024*
12| 19+
13| 28*
14| 720*
15| 2/
16| 3-
17| 1-
18| 245*
19| 5-
20| 24+
21| 16*
22| 24*
23| 20*
24| 9=
25| 17+
26| 1-
27| 8-
28| 10+
29| 63*
30| 120*x
31| 3/
32| 2/
33| 3-
34| 16+
35| 336*
36| 6-
37| 3-
38| 1-
39| 4-
40| 14+
41| 4-
42| 16+

```


LAMPIRAN D

KODE PROGRAM

Lampiran ini berisi kode program dari perangkat lunak Calcudoku ini.

D.1 Grid.java

Listing D.1: Grid.java

```
1 package model;
2
3 import java.util.ArrayList;
4 import java.util.Objects;
5 import javax.swing.JOptionPane;
6
7 public class Grid
8 {
9
10    private final Integer size;
11    private final Integer numberOfCages;
12    private final Integer[][] cageCells;
13    private final String[] cageObjectives;
14    private final Cell[][] grid;
15    private final Cage[] cages;
16
17    public Grid(Integer size, Integer numberOfCages, Integer[][] cageCells,
18                String[] cageObjectives)
19    {
20        this.size = size;
21        this.numberOfCages = numberOfCages;
22        if (isCageCellsSizeValid(cageCells))
23        {
24            this.cageCells = cageCells;
25        }
26        else
27        {
28            JOptionPane.showMessageDialog(null, "Invalid array size.", "Error",
29                                         JOptionPane.ERROR_MESSAGE);
29            throw new IllegalStateException("Invalid array size.");
30        }
31        if (isCageObjectivesSizeValid(cageObjectives))
32        {
33            this.cageObjectives = cageObjectives;
34        }
35        else
36        {
37            JOptionPane.showMessageDialog(null, "Invalid array size.", "Error",
38                                         JOptionPane.ERROR_MESSAGE);
39            throw new IllegalStateException("Invalid array size.");
40        }
41        this.grid = new Cell[size][size];
42        this.cages = new Cage[numberOfCages];
43        generateCages(cages);
44        for (int i = 0; i < cages.length; i++)
45        {
46            Integer[][] array = new Integer[size][size];
47            for (int j = 0; j < cageCells.length; j++)
48            {
49                for (int k = 0; k < cageCells[j].length; k++)
50                {
51                    if (cageCells[j][k] == i + 1)
52                    {
53                        array[j][k] = 1;
54                    }
55                    else
56                    {
57                        array[j][k] = 0;
58                    }
59                }
60            }
61            if (!isCageAssignmentValid(array))
62            {
63                JOptionPane.showMessageDialog(null, "Invalid cage assignment.",
64                                         "Error", JOptionPane.ERROR_MESSAGE);
65                throw new IllegalStateException("Invalid cage assignment.");
66            }
67        }
68    }
69}
```

```

67        }
68    }
69    generateGrid(grid, cages);
70    if (!isCagesValid(cages))
71    {
72        JOptionPane.showMessageDialog(null, "Invalid cages.", "Error",
73                                     JOptionPane.ERROR_MESSAGE);
74        throw new IllegalStateException("Invalid cages.");
75    }
76}
77
78 private int countAreas(Integer [][] array)
79{
80    boolean [][] checked = new boolean [size] [size];
81    for (int i = 0; i < size; i++)
82    {
83        for (int j = 0; j < size; j++)
84        {
85            checked [i] [j] = false;
86        }
87    }
88    return countAreas(array, checked);
89}
90
91 private int countAreas(Integer [][] array, boolean [][] checked)
92{
93    int areas = 0;
94    for (int i = 0; i < array.length; i++)
95    {
96        for (int j = 0; j < array.length; j++)
97        {
98            if (checked [i] [j])
99            {
100                continue;
101            }
102            if (array [i] [j] == 0)
103            {
104                checked [i] [j] = true;
105                continue;
106            }
107            areas++;
108            floodFill(i, j, array, checked);
109        }
110    }
111    return areas;
112}
113
114 private void floodFill(int i, int j, Integer [][] array,
115                      boolean [][] checked)
116{
117    if (array [i] [j] == 0 || checked [i] [j])
118    {
119        return;
120    }
121    checked [i] [j] = true;
122    if (j < array.length - 1)
123    {
124        floodFill(i, j + 1, array, checked);
125    }
126    if (i < array.length - 1)
127    {
128        floodFill(i + 1, j, array, checked);
129    }
130    if (j > 0)
131    {
132        floodFill(i, j - 1, array, checked);
133    }
134    if (i > 0)
135    {
136        floodFill(i - 1, j, array, checked);
137    }
138}
139
140 private boolean isCageCellsSizeValid(Integer [][] cageCells)
141{
142    if (cageCells.length == size)
143    {
144        for (int i = 0; i < size; i++)
145        {
146            if (cageCells [i].length != size)
147            {
148                return false;
149            }
150        }
151    }
152    else
153    {
154        return false;
155    }
156    return true;
157}
158
159 private boolean isCageObjectivesSizeValid(String [] cageObjectives)
160{
161    return cageCells.length == size;
162}
163
164 private boolean isCageAssignmentValid(Integer [][] array)
165{

```

```

166     return countAreas(array) == 1;
167 }
168
169 private boolean isCagesValid(Cage[] cages)
170 {
171     for (Cage c : cages)
172     {
173         if (c.getOperator() == '=' && c.getSize() != 1)
174         {
175             return false;
176         }
177         if ((c.getOperator() == '-' || c.getOperator() == '/'))
178             && c.getSize() != 2)
179         {
180             return false;
181         }
182         if ((c.getOperator() == '+' || c.getOperator() == '*')
183             && c.getSize() < 2)
184         {
185             return false;
186         }
187     }
188     return true;
189 }
190
191 private void generateCages(Cage[] cages)
192 {
193     for (int i = 0; i < cages.length; i++)
194     {
195         cages[i] = new Cage(cageObjectives[i]);
196     }
197 }
198
199 private void generateGrid(Cell[][] grid, Cage[] cages)
200 {
201     for (int i = 0; i < cageCells.length; i++)
202     {
203         for (int j = 0; j < cageCells[i].length; j++)
204         {
205             grid[i][j] = new Cell(i, j, (cageCells[i][j] - 1));
206             cages[cageCells[i][j] - 1].addCell(grid[i][j]);
207         }
208     }
209 }
210
211 public ArrayList<Integer> getRow(int rowNumber)
212 {
213     ArrayList<Integer> row = new ArrayList<>();
214     for (int i = 0; i < grid.length; i++)
215     {
216         if (grid[rowNumber][i].getValue() != null)
217         {
218             row.add(grid[rowNumber][i].getValue());
219         }
220     }
221     return row;
222 }
223
224 public ArrayList<Integer> getColumn(int columnNumber)
225 {
226     ArrayList<Integer> column = new ArrayList<>();
227     for (Cell[] row : grid) {
228         if (row[columnNumber].getValue() != null) {
229             column.add(row[columnNumber].getValue());
230         }
231     }
232     return column;
233 }
234
235 public ArrayList<Integer> getCageValues(int cageNumber)
236 {
237     ArrayList<Integer> cage = new ArrayList<>();
238     for (int i = 0; i < cages[cageNumber].getSize(); i++)
239     {
240         if (cages[cageNumber].getCells().get(i).getValue() != null)
241         {
242             cage.add(cages[cageNumber].getCells().get(i).getValue());
243         }
244     }
245     return cage;
246 }
247
248 private boolean isArrayValid(ArrayList<Integer> array)
249 {
250     for (int i = 0; i < array.size(); i++)
251     {
252         for (int j = 0; j < array.size(); j++)
253         {
254             if (Objects.equals(array.get(i), array.get(j)) && (i != j))
255             {
256                 return false;
257             }
258         }
259     }
260     return true;
261 }
262
263 private boolean isRowValid(int row)
264 {

```

```

265     ArrayList<Integer> array = getRow(row);
266     if (!isArrayValid(array))
267     {
268         JOptionPane.showMessageDialog(null,
269             "Row " + row + " has duplicate numbers.", "Information", JOptionPane.INFORMATION_MESSAGE);
270         return false;
271     }
272     else
273     {
274         return true;
275     }
276 }
277
278 private boolean solverIsRowValid(int row)
279 {
280     ArrayList<Integer> array = getRow(row);
281     return isArrayValid(array);
282 }
283
284 private boolean isColumnValid(int column)
285 {
286     ArrayList<Integer> array = getColumn(column);
287     if (!isArrayValid(array))
288     {
289         JOptionPane.showMessageDialog(null,
290             "Column " + column + " has duplicate numbers.", "Information", JOptionPane.INFORMATION_MESSAGE);
291         return false;
292     }
293     else
294     {
295         return true;
296     }
297 }
298
299 }
300
301 private boolean solverIsColumnValid(int column)
302 {
303     ArrayList<Integer> array = getColumn(column);
304     return isArrayValid(array);
305 }
306
307 private Boolean isCageValuesValid(int cageNumber)
308 {
309     if (cages[cageNumber].isCageValuesValid() == null)
310     {
311         return true;
312     }
313     else
314     {
315         return cages[cageNumber].isCageValuesValid();
316     }
317 }
318
319 private boolean isCageValid(int row, int column)
320 {
321     if (isCageValuesValid(
322         getGridContents()[row][column].get CageID()) == false)
323     {
324         JOptionPane.showMessageDialog(null,
325             "Values of cells in the cage do not reach the target number.", "Information", JOptionPane.INFORMATION_MESSAGE);
326         return false;
327     }
328     else
329     {
330         return true;
331     }
332 }
333
334
335 private boolean solverIsCageValid(int row, int column)
336 {
337     return isCageValuesValid(
338         getGridContents()[row][column].get CageID()) == true;
339 }
340
341 public boolean isCellValueValid(int row, int column)
342 {
343     return (isRowValid(row) && isColumnValid(column)
344             && isCageValid(row, column));
345 }
346
347 public boolean solverIsCellValueValid(int row, int column)
348 {
349     return (solverIsRowValid(row) && solverIsColumnValid(column)
350             && solverIsCageValid(row, column));
351 }
352
353 public boolean setCellValue(int row, int column, Integer value)
354 {
355     if (value > 0 && value <= size)
356     {
357         getGridContents()[row][column].setValue(value);
358         isWin();
359         return isCellValueValid(row, column);
360     }
361     else
362     {
363         JOptionPane.showMessageDialog(null,

```

```

364             "Cell value must be between 1 and " + size + ".",
365             "Information", JOptionPane.INFORMATION_MESSAGE);
366         return false;
367     }
368 }
369
370 public boolean solverSetCellValue(int row, int column, Integer value)
371 {
372     if (value > 0 && value <= size)
373     {
374         getGridContents()[row][column].setValue(value);
375         return (solverIsRowValid(row) && solverIsColumnValid(column)
376                 && solverIsCageValid(row, column));
377     }
378     else
379     {
380         return false;
381     }
382 }
383
384 public void unsetCellValue(int row, int column)
385 {
386     getGridContents()[row][column].setValue(null);
387 }
388
389 public Boolean isWin()
390 {
391     for (int i = 0; i < size; i++)
392     {
393         for (int j = 0; j < size; j++)
394         {
395             if (getGridContents()[i][j].getValue() == null)
396             {
397                 return null;
398             }
399             if (isCellValueValid(i, j) == false)
400             {
401                 return false;
402             }
403         }
404     }
405     JOptionPane.showMessageDialog(null,
406         "Congratulations, you have successfully solved the puzzle!",
407         "Information", JOptionPane.INFORMATION_MESSAGE);
408     return true;
409 }
410
411 public Boolean checkGrid()
412 {
413     for (int i = 0; i < size; i++)
414     {
415         for (int j = 0; j < size; j++)
416         {
417             if (getGridContents()[i][j].getValue() == null)
418             {
419                 JOptionPane.showMessageDialog(null,
420                     "There are empty cells in the grid.",
421                     "Information", JOptionPane.INFORMATION_MESSAGE);
422                 return null;
423             }
424             if (solverIsCellValueValid(i, j) == false)
425             {
426                 JOptionPane.showMessageDialog(null,
427                     "There are cells with incorrect values in the"
428                     + " grid.", "Information",
429                     JOptionPane.INFORMATION_MESSAGE);
430                 return false;
431             }
432         }
433     }
434     JOptionPane.showMessageDialog(null,
435         "Congratulations, you have successfully solved the puzzle!",
436         "Information", JOptionPane.INFORMATION_MESSAGE);
437     return true;
438 }
439
440 public boolean isFilled()
441 {
442     for (int i = 0; i < size; i++)
443     {
444         for (int j = 0; j < size; j++)
445         {
446             if (getGridContents()[i][j].getValue() == null)
447             {
448                 return false;
449             }
450         }
451     }
452     return true;
453 }
454
455 public Integer getCellValue(int row, int column)
456 {
457     return getGridContents()[row][column].getValue();
458 }
459
460 public Integer getSize()
461 {
462     return size;
463 }

```

```

463 }
464
465     public Integer getNumberOfCages()
466     {
467         return numberOfCages;
468     }
469
470     public Integer[][][] getCageCells()
471     {
472         return cageCells;
473     }
474
475     public String[] getCageObjectives()
476     {
477         return cageObjectives;
478     }
479
480     public Cell[][][] getGridContents()
481     {
482         return grid;
483     }
484
485     public Cage[] getCages()
486     {
487         return cages;
488     }
489
490     public Grid getGame()
491     {
492         return this;
493     }
494 }
495 }
```

D.2 Cell.java

Listing D.2: Cell.java

```

1 package model;
2
3 public class Cell
4 {
5
6     private final int row;
7     private final int column;
8     private final int cageID;
9     private Integer value;
10
11    public Cell(int row, int column, int cageID)
12    {
13        this.row = row;
14        this.column = column;
15        this.cageID = cageID;
16    }
17
18    public void setValue(Integer value)
19    {
20        this.value = value;
21    }
22
23    public Integer getValue()
24    {
25        return value;
26    }
27
28    public int getRow()
29    {
30        return row;
31    }
32
33    public int getColumn()
34    {
35        return column;
36    }
37
38    public int getCageID()
39    {
40        return cageID;
41    }
42 }
43 }
```

D.3 Cage.java

Listing D.3: Cage.java

```

1 package model;
2
3 import java.util.ArrayList;
4 import javax.swing.JOptionPane;
```

```

6  public class Cage
7  {
8
9      private final String objective;
10     private final int targetNumber;
11     private final char operator;
12     private final ArrayList<Cell> cells;
13
14     public Cage(String objective)
15     {
16         if (isCageObjectiveValid(objective))
17         {
18             this.objective = objective;
19         }
20         else
21         {
22             JOptionPane.showMessageDialog(null, "Invalid cage objectives.", "Error", JOptionPane.ERROR_MESSAGE);
23             throw new IllegalStateException("Invalid cage objectives.");
24         }
25         this.targetNumber = generateTargetNumber(this.objective);
26         this.operator = generateOperator(this.objective);
27         this.cells = new ArrayList<>();
28     }
29
30     private boolean isCageObjectiveValid(String cageObjective)
31     {
32         return cageObjective.matches("\\d+[+-/=]");
33     }
34
35     private int generateTargetNumber(String objective)
36     {
37         return Integer.parseInt(objective.substring(0,
38                                         objective.length() - 1));
39     }
40
41     private char generateOperator(String objective)
42     {
43         return objective.charAt(objective.length() - 1);
44     }
45
46     public void addCell(Cell c)
47     {
48         cells.add(c);
49     }
50
51     public boolean isCageContainsNull()
52     {
53         for (int i = 0; i < cells.size(); i++)
54         {
55             if (cells.get(i).getValue() == null)
56             {
57                 return true;
58             }
59         }
60         return false;
61     }
62
63     public Boolean isCageValuesValid()
64     {
65         if (countValue() == null)
66         {
67             return null;
68         }
69         else
70         {
71             return (countValue() == getTargetNumber());
72         }
73     }
74
75     private Integer countValue()
76     {
77         Integer value = null;
78         if (isCageContainsNull() == true)
79         {
80             value = null;
81         }
82         else
83         {
84             switch (getOperator())
85             {
86                 case '+':
87                     value = 0;
88                     for (int i = 0; i < cells.size(); i++)
89                     {
90                         value += cells.get(i).getValue();
91                     }
92                     break;
93                 case '-':
94                     if (cells.get(0).getValue() > cells.get(1).getValue())
95                     {
96                         value = cells.get(0).getValue()
97                             - cells.get(1).getValue();
98                     }
99                     else if (cells.get(1).getValue() > cells.get(0).getValue())
100                     {
101                         value = cells.get(1).getValue()
102                             - cells.get(0).getValue();
103                     }
104             }
105         }
106     }

```

```

105         else
106             {
107                 value = 0;
108             }
109         break;
110     case '*':
111         value = 1;
112         for (int i = 0; i < cells.size(); i++)
113         {
114             value *= cells.get(i).getValue();
115         }
116         break;
117     case '/':
118         if (cells.get(0).getValue() > cells.get(1).getValue())
119         {
120             if (cells.get(0).getValue()
121                 % cells.get(1).getValue() == 0)
122             {
123                 value = cells.get(0).getValue()
124                     / cells.get(1).getValue();
125             }
126             else
127             {
128                 value = 0;
129             }
130         }
131         else if (cells.get(1).getValue() > cells.get(0).getValue())
132         {
133             if (cells.get(1).getValue()
134                 % cells.get(0).getValue() == 0)
135             {
136                 value = cells.get(1).getValue()
137                     / cells.get(0).getValue();
138             }
139             else
140             {
141                 value = 0;
142             }
143         }
144         else
145         {
146             value = 0;
147         }
148         break;
149     case '=':
150         value = cells.get(0).getValue();
151         break;
152     default :
153         value = null;
154     }
155 }
156 return value;
157 }
158
159 public String getObjective()
160 {
161     return objective;
162 }
163
164 public int getTargetNumber()
165 {
166     return targetNumber;
167 }
168
169 public char getOperator()
170 {
171     return operator;
172 }
173
174 public ArrayList<Cell> getCells()
175 {
176     return cells;
177 }
178
179 public int getSize()
180 {
181     return cells.size();
182 }
183 }
184 }
```

D.4 SolverBacktracking.java

Listing D.4: SolverBacktracking.java

```

1 package model;
2
3 public class SolverBacktracking
4 {
5
6     private final Grid grid;
7     private final Integer size;
8     private Grid solution;
9
10    public SolverBacktracking(Grid grid)
```

```

11     {
12         this.grid = grid;
13         this.size = grid.getSize();
14     }
15
16     public boolean solve()
17     {
18         if (solve(0, 0) == true)
19         {
20             this.solution = grid;
21             printGrid(solution.getGridContents());
22             return true;
23         }
24         else
25         {
26             return false;
27         }
28     }
29
30     private boolean solve(int row, int column)
31     {
32         if (column >= size)
33         {
34             column = 0;
35             row++;
36             if (row >= size)
37             {
38                 printGrid(grid.getGridContents());
39                 return true;
40             }
41         }
42         for (int value = 1; value <= size; value++)
43         {
44             printGrid(grid.getGridContents());
45             if (grid.solverSetCellValue(row, column, value))
46             {
47                 if (solve(row, column + 1))
48                 {
49                     return true;
50                 }
51             }
52         }
53         printGrid(grid.getGridContents());
54         grid.unsetCellValue(row, column);
55         return false;
56     }
57
58     public Grid getGrid()
59     {
60         return grid;
61     }
62
63     public Grid getSolution()
64     {
65         return solution;
66     }
67
68     private void printGrid(Cell[][] cells)
69     {
70         for (int i = 0; i < size; i++)
71         {
72             for (int j = 0; j < size; j++)
73             {
74                 System.out.print(cells[i][j].getValue() + " ");
75             }
76             System.out.println(" ");
77         }
78         System.out.println(" ");
79     }
80 }

```

D.5 SolverHybridGenetic.java

Listing D.5: SolverHybridGenetic.java

```

20|     this.grid = grid;
21|     this.size = grid.getSize();
22|     this.generations = generations;
23|     this.populationSize = populationSize;
24|     this.elitismRate = elitismRate;
25|     this.crossoverRate = crossoverRate;
26|     this.mutationRate = mutationRate;
27}
28
29 public boolean solve()
30{
31    SolverRuleBased srb = new SolverRuleBased(grid);
32    boolean isFilled = srb.solve();
33    this.gridRuleBased = srb.getGrid();
34    if (isFilled)
35    {
36        this.solution = srb.getSolution();
37        printGrid(solution.getGridContents());
38        return true;
39    }
40    else
41    {
42        SolverGenetic sg = new SolverGenetic(gridRuleBased, generations,
43                                              populationSize, elitismRate, crossoverRate, mutationRate);
44        if (sg.solve() == true)
45        {
46            this.solution = sg.getSolution();
47            printGrid(solution.getGridContents());
48            return true;
49        }
50    }
51    return false;
52}
53
54 public Grid getGrid()
55{
56    return grid;
57}
58
59 public Grid getSolution()
60{
61    return solution;
62}
63
64 private void printGrid(Cell[][] cells)
65{
66    for (int i = 0; i < size; i++)
67    {
68        for (int j = 0; j < size; j++)
69        {
70            System.out.print(cells[i][j].getValue() + " ");
71        }
72        System.out.println(" ");
73    }
74    System.out.println(" ");
75}
76}
77}

```

D.6 SolverRuleBased.java

Listing D.6: SolverRuleBased.java

```

1 package model;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import javax.swing.JOptionPane;
6
7 public class SolverRuleBased
8{
9
10    private final Grid grid;
11    private final Integer size;
12    private Grid solution;
13    private ArrayList<Integer>[][] possibleValues;
14
15    public SolverRuleBased(Grid grid)
16    {
17        this.grid = grid;
18        this.size = grid.getSize();
19        this.possibleValues = generatePossibleValuesArray();
20    }
21
22    @SuppressWarnings("unchecked")
23    private ArrayList<Integer>[][] generatePossibleValuesArray()
24    {
25        possibleValues = new ArrayList[size][size];
26        for (int i = 0; i < size; i++)
27        {
28            for (int j = 0; j < size; j++)
29            {
30                possibleValues[i][j] = new ArrayList<Integer>();
31            }
32        }
33    }
34}

```

```

33|     for (int i = 0; i < size; i++)
34|     {
35|         for (int j = 0; j < size; j++)
36|         {
37|             for (int k = 1; k <= size; k++)
38|             {
39|                 possibleValues[i][j].add(k);
40|             }
41|         }
42|     }
43|     return possibleValues;
44|
45|
46| public boolean solve()
47| {
48|     singleSquare();
49|     killerCombination();
50|     ArrayList<ArrayList<Integer>> currentGridArrayList
51|     = getGridArrayList();
52|     ArrayList<ArrayList<Integer>> newGridArrayList = solveLoop();
53|     while (!currentGridArrayList.equals(newGridArrayList))
54|     {
55|         printGrid();
56|         printPossibleValues();
57|         currentGridArrayList = newGridArrayList;
58|         newGridArrayList = solveLoop();
59|     }
60|     if (grid.isFilled())
61|     {
62|         this.solution = grid;
63|         return true;
64|     }
65|     else
66|     {
67|         return false;
68|     }
69| }
70|
71| private ArrayList<ArrayList<Integer>> solveLoop()
72| {
73|     nakedSingle();
74|     nakedDouble();
75|     hiddenSingle();
76|     return getGridArrayList();
77| }
78|
79| @SuppressWarnings("unchecked")
80| public ArrayList<Integer>[] getRowPossibleValues(int row)
81| {
82|     ArrayList<Integer>[] array;
83|     array = new ArrayList[size];
84|     for (int i = 0; i < size; i++)
85|     {
86|         array[i] = possibleValues[row][i];
87|     }
88|     return array;
89| }
90|
91| @SuppressWarnings("unchecked")
92| public ArrayList<Integer>[] getColumnPossibleValues(int column)
93| {
94|     ArrayList<Integer>[] array;
95|     array = new ArrayList[size];
96|     for (int i = 0; i < size; i++)
97|     {
98|         array[i] = possibleValues[i][column];
99|     }
100|    return array;
101| }
102|
103| private void singleSquare()
104| {
105|     for (Cage c : grid.getCages())
106|     {
107|         if (c.getSize() == 1)
108|         {
109|             setCellValue(c.getCells().get(0).getRow(),
110|                          c.getCells().get(0).getColumn(), c.getTargetNumber());
111|         }
112|     }
113| }
114|
115| private void killerCombination()
116| {
117|     int cageSize;
118|     ArrayList<Integer> array = new ArrayList<Integer>();
119|     for (Cage c : grid.getCages())
120|     {
121|         cageSize = c.getSize();
122|         switch (cageSize)
123|         {
124|             case 2:
125|                 killerCombinationCageSize2(c);
126|                 break;
127|             default:
128|                 array = createRetainAllArray();
129|                 removeImpossibleValuesCage(c, array);
130|                 break;
131|         }
132|     }
133| }

```

```

132     }
133 }
134
135 private void killerCombinationCageSize2(Cage cage)
136 {
137     int gridSize = size;
138     ArrayList<Integer> array = new ArrayList<Integer>();
139     switch (gridSize)
140     {
141         case 3 :
142             array = killerCombinationCageSize2GridSize3(cage);
143             break;
144         case 4 :
145             array = killerCombinationCageSize2GridSize4(cage);
146             break;
147         case 5 :
148             array = killerCombinationCageSize2GridSize5(cage);
149             break;
150         case 6 :
151             array = killerCombinationCageSize2GridSize6(cage);
152             break;
153         case 7 :
154             array = killerCombinationCageSize2GridSize7(cage);
155             break;
156         case 8 :
157             array = killerCombinationCageSize2GridSize8(cage);
158             break;
159         case 9 :
160             array = killerCombinationCageSize2GridSize9(cage);
161             break;
162         default :
163             JOptionPane.showMessageDialog(null, "Invalid grid size.", "Error", JOptionPane.ERROR_MESSAGE);
164             throw new IllegalStateException("Invalid grid size.");
165     }
166     removeImpossibleValuesCage(cage, array);
167 }
168
169 private ArrayList<Integer> killerCombinationCageSize2GridSize3(Cage cage)
170 {
171     char cageOperator = cage.getOperator();
172     int cageTargetNumber = cage.getTargetNumber();
173     ArrayList<Integer> array = new ArrayList<Integer>();
174     switch (cageOperator)
175     {
176         case '+':
177             switch (cageTargetNumber)
178             {
179                 case 3 :
180                     array.add(1);
181                     array.add(2);
182                     break;
183                 case 4 :
184                     array.add(1);
185                     array.add(3);
186                     break;
187                 case 5 :
188                     array.add(2);
189                     array.add(3);
190                     break;
191                 default :
192                     array = createRetainAllArray();
193                     break;
194             }
195             break;
196         case '-':
197             switch (cageTargetNumber)
198             {
199                 case 2 :
200                     array.add(1);
201                     array.add(3);
202                     break;
203                 default :
204                     array = createRetainAllArray();
205                     break;
206             }
207             break;
208         case '*':
209             switch (cageTargetNumber)
210             {
211                 case 2 :
212                     array.add(1);
213                     array.add(2);
214                     break;
215                 case 3 :
216                     array.add(1);
217                     array.add(3);
218                     break;
219                 case 6 :
220                     array.add(2);
221                     array.add(3);
222                     break;
223                 default :
224                     array = createRetainAllArray();
225                     break;
226             }
227             break;
228         case '/':
229             switch (cageTargetNumber)
230             {

```

```

231|
232|     {
233|         case 2 :
234|             array.add(1);
235|             array.add(2);
236|             break;
237|         case 3 :
238|             array.add(1);
239|             array.add(3);
240|             break;
241|         default :
242|             array = createRetainAllArray();
243|             break;
244|     }
245|     break;
246| default :
247|     JOptionPane.showMessageDialog(null,
248|         "Invalid operator.", "Error",
249|         JOptionPane.ERROR_MESSAGE);
250|     throw new IllegalStateException("Invalid operator.");
251|
252| }
253|
254| private ArrayList<Integer> killerCombinationCageSize2GridSize4(Cage cage)
255| {
256|     char cageOperator = cage.getOperator();
257|     int cageTargetNumber = cage.getTargetNumber();
258|     ArrayList<Integer> array = new ArrayList<Integer>();
259|     switch (cageOperator)
260|     {
261|         case '+':
262|             switch (cageTargetNumber)
263|             {
264|                 case 3 :
265|                     array.add(1);
266|                     array.add(2);
267|                     break;
268|                 case 4 :
269|                     array.add(1);
270|                     array.add(3);
271|                     break;
272|                 case 6 :
273|                     array.add(2);
274|                     array.add(4);
275|                     break;
276|                 case 7 :
277|                     array.add(3);
278|                     array.add(4);
279|                     break;
280|                 default :
281|                     array = createRetainAllArray();
282|                     break;
283|             }
284|             break;
285|         case '-':
286|             switch (cageTargetNumber)
287|             {
288|                 case 3 :
289|                     array.add(1);
290|                     array.add(4);
291|                     break;
292|                 default :
293|                     array = createRetainAllArray();
294|                     break;
295|             }
296|             break;
297|         case '*':
298|             switch (cageTargetNumber)
299|             {
300|                 case 2 :
301|                     array.add(1);
302|                     array.add(2);
303|                     break;
304|                 case 3 :
305|                     array.add(1);
306|                     array.add(3);
307|                     break;
308|                 case 4 :
309|                     array.add(1);
310|                     array.add(4);
311|                     break;
312|                 case 6 :
313|                     array.add(2);
314|                     array.add(3);
315|                     break;
316|                 case 8 :
317|                     array.add(2);
318|                     array.add(4);
319|                     break;
320|                 case 12 :
321|                     array.add(3);
322|                     array.add(4);
323|                     break;
324|                 default :
325|                     array = createRetainAllArray();
326|                     break;
327|             }
328|             break;
329|         case '/':
330|     }
331| }

```



```

429             array.add(3);
430             array.add(4);
431             break;
432         case 15 :
433             array.add(3);
434             array.add(5);
435             break;
436         case 20 :
437             array.add(4);
438             array.add(5);
439             break;
440         default :
441             array = createRetainAllArray();
442             break;
443     }
444     break;
445     case '/':
446     switch (cageTargetNumber)
447     {
448         case 3 :
449             array.add(1);
450             array.add(3);
451             break;
452         case 4 :
453             array.add(1);
454             array.add(4);
455             break;
456         case 5 :
457             array.add(1);
458             array.add(5);
459             break;
460         default :
461             array = createRetainAllArray();
462             break;
463     }
464     break;
465     default :
466         JOptionPane.showMessageDialog(null,
467             "Invalid operator.", "Error",
468             JOptionPane.ERROR_MESSAGE);
469         throw new IllegalStateException("Invalid operator.");
470     }
471     return array;
472 }
473
474 private ArrayList<Integer> killerCombinationCageSize2GridSize6(Cage cage)
475 {
476     char cageOperator = cage.getOperator();
477     int cageTargetNumber = cage.getTargetNumber();
478     ArrayList<Integer> array = new ArrayList<Integer>();
479     switch (cageOperator)
480     {
481         case '+':
482             switch (cageTargetNumber)
483             {
484                 case 3 :
485                     array.add(1);
486                     array.add(2);
487                     break;
488                 case 4 :
489                     array.add(1);
490                     array.add(3);
491                     break;
492                 case 10 :
493                     array.add(4);
494                     array.add(6);
495                     break;
496                 case 11 :
497                     array.add(5);
498                     array.add(6);
499                     break;
500                 default :
501                     array = createRetainAllArray();
502                     break;
503             }
504             break;
505         case '-':
506             switch (cageTargetNumber)
507             {
508                 case 5 :
509                     array.add(1);
510                     array.add(6);
511                     break;
512                 default :
513                     array = createRetainAllArray();
514                     break;
515             }
516             break;
517         case '*':
518             switch (cageTargetNumber)
519             {
520                 case 2 :
521                     array.add(1);
522                     array.add(2);
523                     break;
524                 case 3 :
525                     array.add(1);
526                     array.add(3);
527                     break;

```

```

528     case 4 :
529         array.add(1);
530         array.add(4);
531         break;
532     case 5 :
533         array.add(1);
534         array.add(5);
535         break;
536     case 8 :
537         array.add(2);
538         array.add(4);
539         break;
540     case 10 :
541         array.add(2);
542         array.add(5);
543         break;
544     case 15 :
545         array.add(3);
546         array.add(5);
547         break;
548     case 18 :
549         array.add(3);
550         array.add(6);
551         break;
552     case 20 :
553         array.add(4);
554         array.add(5);
555         break;
556     case 24 :
557         array.add(4);
558         array.add(6);
559         break;
560     case 30 :
561         array.add(5);
562         array.add(6);
563         break;
564     default :
565         array = createRetainAllArray();
566         break;
567     }
568     break;
569 case '/':
570     switch (cageTargetNumber)
571     {
572         case 4 :
573             array.add(1);
574             array.add(4);
575             break;
576         case 5 :
577             array.add(1);
578             array.add(5);
579             break;
580         case 6 :
581             array.add(1);
582             array.add(6);
583             break;
584         default :
585             array = createRetainAllArray();
586             break;
587     }
588     break;
589 default :
590     JOptionPane.showMessageDialog(null,
591         "Invalid operator.", "Error",
592         JOptionPane.ERROR_MESSAGE);
593     throw new IllegalStateException("Invalid operator.");
594 }
595 return array;
596 }
597
598 private ArrayList<Integer> killerCombinationCageSize2GridSize7(Cage cage)
599 {
600     char cageOperator = cage.getOperator();
601     int cageTargetNumber = cage.getTargetNumber();
602     ArrayList<Integer> array = new ArrayList<Integer>();
603     switch (cageOperator)
604     {
605         case '+':
606             switch (cageTargetNumber)
607             {
608                 case 3 :
609                     array.add(1);
610                     array.add(2);
611                     break;
612                 case 4 :
613                     array.add(1);
614                     array.add(3);
615                     break;
616                 case 12 :
617                     array.add(5);
618                     array.add(7);
619                     break;
620                 case 13 :
621                     array.add(6);
622                     array.add(7);
623                     break;
624                 default :
625                     array = createRetainAllArray();
626                     break;
627             }
628         }
629     }
630 }
```

```
627 }  
628     break;  
629 case "6":  
630     switch (cageTargetNumber)  
631     {  
632         case 6 :  
633             array.add(1);  
634             array.add(7);  
635             break;  
636         default :  
637             array = createRetainAllArray();  
638             break;  
639     }  
640     break;  
641 case "7":  
642     switch (cageTargetNumber)  
643     {  
644         case 2 :  
645             array.add(1);  
646             array.add(2);  
647             break;  
648         case 3 :  
649             array.add(1);  
650             array.add(3);  
651             break;  
652         case 4 :  
653             array.add(1);  
654             array.add(4);  
655             break;  
656         case 5 :  
657             array.add(1);  
658             array.add(5);  
659             break;  
660         case 7 :  
661             array.add(1);  
662             array.add(7);  
663             break;  
664         case 8 :  
665             array.add(2);  
666             array.add(4);  
667             break;  
668         case 10 :  
669             array.add(2);  
670             array.add(5);  
671             break;  
672         case 14 :  
673             array.add(2);  
674             array.add(7);  
675             break;  
676         case 15 :  
677             array.add(3);  
678             array.add(5);  
679             break;  
680         case 18 :  
681             array.add(3);  
682             array.add(6);  
683             break;  
684         case 20 :  
685             array.add(4);  
686             array.add(5);  
687             break;  
688         case 21 :  
689             array.add(3);  
690             array.add(7);  
691             break;  
692         case 24 :  
693             array.add(4);  
694             array.add(6);  
695             break;  
696         case 28 :  
697             array.add(4);  
698             array.add(7);  
699             break;  
700         case 30 :  
701             array.add(5);  
702             array.add(6);  
703             break;  
704         case 35 :  
705             array.add(5);  
706             array.add(7);  
707             break;  
708         case 42 :  
709             array.add(6);  
710             array.add(7);  
711             break;  
712         default :  
713             array = createRetainAllArray();  
714             break;  
715     }  
716     break;  
717 case "/":  
718     switch (cageTargetNumber)  
719     {  
720         case 4 :  
721             array.add(1);  
722             array.add(4);  
723             break;  
724         case 5 :  
725             array.add(1);
```



```

825             array.add(3);
826             array.add(5);
827             break;
828         case 16 :
829             array.add(2);
830             array.add(8);
831             break;
832         case 18 :
833             array.add(3);
834             array.add(6);
835             break;
836         case 20 :
837             array.add(4);
838             array.add(5);
839             break;
840         case 21 :
841             array.add(3);
842             array.add(7);
843             break;
844         case 28 :
845             array.add(4);
846             array.add(7);
847             break;
848         case 30 :
849             array.add(5);
850             array.add(6);
851             break;
852         case 32 :
853             array.add(4);
854             array.add(8);
855             break;
856         case 35 :
857             array.add(5);
858             array.add(7);
859             break;
860         case 40 :
861             array.add(5);
862             array.add(8);
863             break;
864         case 42 :
865             array.add(6);
866             array.add(7);
867             break;
868         case 48 :
869             array.add(6);
870             array.add(8);
871             break;
872         case 56 :
873             array.add(7);
874             array.add(8);
875             break;
876     default :
877         array = createRetainAllArray();
878         break;
879     }
880     break;
881 case '/':
882     switch (cageTargetNumber)
883     {
884         case 5 :
885             array.add(1);
886             array.add(5);
887             break;
888         case 6 :
889             array.add(1);
890             array.add(6);
891             break;
892         case 7 :
893             array.add(1);
894             array.add(7);
895             break;
896         case 8 :
897             array.add(1);
898             array.add(8);
899             break;
900     default :
901         array = createRetainAllArray();
902         break;
903     }
904     break;
905 default :
906     JOptionPane.showMessageDialog(null,
907         "Invalid operator.", "Error",
908         JOptionPane.ERROR_MESSAGE);
909     throw new IllegalStateException("Invalid operator.");
910 }
911 return array;
912 }
913
914 private ArrayList<Integer> killerCombinationCageSize2GridSize9(Cage cage)
915 {
916     char cageOperator = cage.getOperator();
917     int cageTargetNumber = cage.getTargetNumber();
918     ArrayList<Integer> array = new ArrayList<Integer>();
919     switch (cageOperator)
920     {
921         case '+':
922             switch (cageTargetNumber)
923             {

```

```
924|         case 3 :
925|             array.add(1);
926|             array.add(2);
927|             break;
928|         case 4 :
929|             array.add(1);
930|             array.add(3);
931|             break;
932|         case 16 :
933|             array.add(7);
934|             array.add(9);
935|             break;
936|         case 17 :
937|             array.add(8);
938|             array.add(9);
939|             break;
940|         default :
941|             array = createRetainAllArray();
942|             break;
943|     }
944|     break;
945| case '1' :
946|     switch (cageTargetNumber)
947|     {
948|         case 8 :
949|             array.add(1);
950|             array.add(9);
951|             break;
952|         default :
953|             array = createRetainAllArray();
954|             break;
955|     }
956|     break;
957| case '2' :
958|     switch (cageTargetNumber)
959|     {
960|         case 2 :
961|             array.add(1);
962|             array.add(2);
963|             break;
964|         case 3 :
965|             array.add(1);
966|             array.add(3);
967|             break;
968|         case 4 :
969|             array.add(1);
970|             array.add(4);
971|             break;
972|         case 5 :
973|             array.add(1);
974|             array.add(5);
975|             break;
976|         case 7 :
977|             array.add(1);
978|             array.add(7);
979|             break;
980|         case 9 :
981|             array.add(1);
982|             array.add(9);
983|             break;
984|         case 10 :
985|             array.add(2);
986|             array.add(5);
987|             break;
988|         case 14 :
989|             array.add(2);
990|             array.add(7);
991|             break;
992|         case 15 :
993|             array.add(3);
994|             array.add(5);
995|             break;
996|         case 16 :
997|             array.add(2);
998|             array.add(8);
999|             break;
1000|        case 20 :
1001|            array.add(4);
1002|            array.add(5);
1003|            break;
1004|        case 21 :
1005|            array.add(3);
1006|            array.add(7);
1007|            break;
1008|        case 27 :
1009|            array.add(3);
1010|            array.add(9);
1011|            break;
1012|        case 28 :
1013|            array.add(4);
1014|            array.add(7);
1015|            break;
1016|        case 30 :
1017|            array.add(5);
1018|            array.add(6);
1019|            break;
1020|        case 32 :
1021|            array.add(4);
1022|            array.add(8);
```

```
1023         break;
1024     case 35 :
1025         array.add(5);
1026         array.add(7);
1027         break;
1028     case 36 :
1029         array.add(4);
1030         array.add(9);
1031         break;
1032     case 40 :
1033         array.add(5);
1034         array.add(8);
1035         break;
1036     case 42 :
1037         array.add(6);
1038         array.add(7);
1039         break;
1040     case 45 :
1041         array.add(5);
1042         array.add(9);
1043         break;
1044     case 48 :
1045         array.add(6);
1046         array.add(8);
1047         break;
1048     case 54 :
1049         array.add(6);
1050         array.add(9);
1051         break;
1052     case 56 :
1053         array.add(7);
1054         array.add(8);
1055         break;
1056     case 63 :
1057         array.add(7);
1058         array.add(9);
1059         break;
1060     case 72 :
1061         array.add(8);
1062         array.add(9);
1063         break;
1064     default :
1065         array = createRetainAllArray();
1066         break;
1067     }
1068     break;
1069 case '/':
1070     switch (cageTargetNumber)
1071     {
1072         case 5 :
1073             array.add(1);
1074             array.add(5);
1075             break;
1076         case 6 :
1077             array.add(1);
1078             array.add(6);
1079             break;
1080         case 7 :
1081             array.add(1);
1082             array.add(7);
1083             break;
1084         case 8 :
1085             array.add(1);
1086             array.add(8);
1087             break;
1088         case 9 :
1089             array.add(1);
1090             array.add(9);
1091             break;
1092         default :
1093             array = createRetainAllArray();
1094             break;
1095     }
1096     break;
1097 default :
1098     JOptionPane.showMessageDialog(null,
1099         "Invalid operator.", "Error",
1100         JOptionPane.ERROR_MESSAGE);
1101     throw new IllegalStateException("Invalid operator.");
1102 }
1103 return array;
1104 }
1105
1106 private void nakedSingle()
1107 {
1108     nakedSingleRow();
1109     nakedSingleColumn();
1110 }
1111
1112 private void nakedSingleRow()
1113 {
1114     for (int i = 0; i < size; i++)
1115     {
1116         nakedSingleRow(i);
1117     }
1118 }
1119
1120 private void nakedSingleRow(int row)
1121 {
```



```

1221             }
1222         }
1223         nakedDoubleRow(row, doublePossibleValues,
1224                         doublePossibleIndexes);
1225     }
1226 }
1227 }
1228 }
1229
1230 private void nakedDoubleRow(int row,
1231     ArrayList<Integer> doublePossibleValues,
1232     ArrayList<Integer> doublePossibleIndexes)
1233 {
1234     for (int i = 0; i < size; i++)
1235     {
1236         if (!doublePossibleIndexes.contains(i))
1237         {
1238             possibleValues[row][i].removeAll(doublePossibleValues);
1239         }
1240     }
1241 }
1242
1243 private void nakedDoubleColumn()
1244 {
1245     for (int i = 0; i < size; i++)
1246     {
1247         nakedDoubleColumn(i);
1248     }
1249 }
1250
1251 private void nakedDoubleColumn(int column)
1252 {
1253     ArrayList<Integer>[] columnPossibleValues =
1254         getColumnPossibleValues(column);
1255     ArrayList<Integer> row2PossibleIndexes = new ArrayList<Integer>();
1256     ArrayList<ArrayList<Integer>> row2PossibleValues = new ArrayList<>();
1257     ArrayList<ArrayList<Integer>> uniquePossibleValues = new ArrayList<>();
1258     ArrayList<Integer> uniquePossibleValuesFrequency =
1259         new ArrayList<Integer>();
1260     for (int i = 0; i < columnPossibleValues.length; i++)
1261     {
1262         if (columnPossibleValues[i].size() == 2)
1263         {
1264             row2PossibleIndexes.add(i);
1265             row2PossibleValues.add(columnPossibleValues[i]);
1266         }
1267     }
1268     if (row2PossibleIndexes.size() >= 2
1269          && row2PossibleValues.size() >= 2)
1270     {
1271         for (int i = 0; i < row2PossibleValues.size(); i++)
1272         {
1273             if (!uniquePossibleValues.contains(
1274                 row2PossibleValues.get(i)))
1275             {
1276                 uniquePossibleValues.add(row2PossibleValues.get(i));
1277             }
1278         }
1279         for (int i = 0; i < uniquePossibleValues.size(); i++)
1280         {
1281             uniquePossibleValuesFrequency.add(Collections.frequency(
1282                 row2PossibleValues, uniquePossibleValues.get(i)));
1283         }
1284         for (int i = 0; i < uniquePossibleValuesFrequency.size(); i++)
1285         {
1286             if (uniquePossibleValuesFrequency.get(i) == 2)
1287             {
1288                 ArrayList<Integer> doublePossibleValues =
1289                     uniquePossibleValues.get(i);
1290                 ArrayList<Integer> doublePossibleIndexes =
1291                     new ArrayList<Integer>();
1292                 for (int j = 0; j < row2PossibleValues.size(); j++)
1293                 {
1294                     if (row2PossibleValues.get(j).equals(
1295                         uniquePossibleValues.get(i)))
1296                     {
1297                         doublePossibleIndexes.add(
1298                             row2PossibleIndexes.get(j));
1299                     }
1300                 }
1301                 nakedDoubleColumn(column, doublePossibleValues,
1302                                 doublePossibleIndexes);
1303             }
1304         }
1305     }
1306 }
1307
1308 private void nakedDoubleColumn(int column,
1309     ArrayList<Integer> doublePossibleValues,
1310     ArrayList<Integer> doublePossibleIndexes)
1311 {
1312     for (int i = 0; i < size; i++)
1313     {
1314         if (!doublePossibleIndexes.contains(i))
1315         {
1316             possibleValues[i][column].removeAll(doublePossibleValues);
1317         }
1318     }
1319 }
```

```

1320
1321     private void hiddenSingle()
1322     {
1323         hiddenSingleRow();
1324         hiddenSingleColumn();
1325     }
1326
1327     private void hiddenSingleRow()
1328     {
1329         for (int i = 0; i < size; i++)
1330         {
1331             hiddenSingleRow(i);
1332         }
1333     }
1334
1335     private void hiddenSingleRow(int row)
1336     {
1337         ArrayList<Integer>[] rowPossibleValues = getRowPossibleValues(row);
1338         int[] possibleValuesFrequency = new int[size];
1339         ArrayList<Integer> columnValues = new ArrayList<Integer>();
1340         ArrayList<Integer> columnIndexes = new ArrayList<Integer>();
1341         for (ArrayList<Integer> rowPossibleValue : rowPossibleValues)
1342         {
1343             for (int i = 1; i <= possibleValuesFrequency.length; i++)
1344             {
1345                 if (rowPossibleValue.contains(i))
1346                 {
1347                     possibleValuesFrequency[i - 1]++;
1348                 }
1349             }
1350         }
1351         for (int i = 0; i < possibleValuesFrequency.length; i++)
1352         {
1353             if (possibleValuesFrequency[i] == 1)
1354             {
1355                 columnValues.add(i + 1);
1356             }
1357         }
1358         for (int i = 0; i < columnValues.size(); i++)
1359         {
1360             for (int j = 0; j < rowPossibleValues.length; j++)
1361             {
1362                 if (rowPossibleValues[j].contains(columnValues.get(i)))
1363                 {
1364                     columnIndexes.add(j);
1365                 }
1366             }
1367         }
1368         for (int i = 0; i < columnValues.size(); i++)
1369         {
1370             if (rowPossibleValues[columnIndexes.get(i)].size() >= 2)
1371             {
1372                 hiddenSingle(row, columnIndexes.get(i), columnValues.get(i));
1373             }
1374         }
1375     }
1376
1377     private void hiddenSingleColumn()
1378     {
1379         for (int i = 0; i < size; i++)
1380         {
1381             hiddenSingleColumn(i);
1382         }
1383     }
1384
1385     private void hiddenSingleColumn(int column)
1386     {
1387         ArrayList<Integer>[] columnPossibleValues
1388             = getColumnPossibleValues(column);
1389         int[] possibleValuesFrequency = new int[size];
1390         ArrayList<Integer> rowValues = new ArrayList<Integer>();
1391         ArrayList<Integer> rowIndexes = new ArrayList<Integer>();
1392         for (ArrayList<Integer> columnPossibleValue : columnPossibleValues)
1393         {
1394             for (int i = 1; i <= possibleValuesFrequency.length; i++)
1395             {
1396                 if (columnPossibleValue.contains(i))
1397                 {
1398                     possibleValuesFrequency[i - 1]++;
1399                 }
1400             }
1401         }
1402         for (int i = 0; i < possibleValuesFrequency.length; i++)
1403         {
1404             if (possibleValuesFrequency[i] == 1)
1405             {
1406                 rowValues.add(i + 1);
1407             }
1408         }
1409         for (int i = 0; i < rowValues.size(); i++)
1410         {
1411             for (int j = 0; j < columnPossibleValues.length; j++)
1412             {
1413                 if (columnPossibleValues[j].contains(rowValues.get(i)))
1414                 {
1415                     rowIndexes.add(j);
1416                 }
1417             }
1418         }

```

```

1419     for (int i = 0; i < rowValues.size(); i++)
1420     {
1421         if (columnPossibleValues[rowIndexes.get(i)].size() >= 2)
1422         {
1423             hiddenSingle(rowIndexes.get(i), column, rowValues.get(i));
1424         }
1425     }
1426
1427     private void hiddenSingle(int row, int column, int value)
1428     {
1429         setCellValue(row, column, value);
1430     }
1431
1432     private void setCellValue(int row, int column, int value)
1433     {
1434         grid.solverSetCellValue(row, column, value);
1435         removePossibleValues(row, column, value);
1436     }
1437
1438     private void removePossibleValues(int row, int column, int value)
1439     {
1440         possibleValues[row][column].clear();
1441         for (int i = 0; i < size; i++)
1442         {
1443             if (possibleValues[i][column].contains(value))
1444             {
1445                 possibleValues[i][column].remove(
1446                     possibleValues[i][column].indexOf(value));
1447             }
1448         }
1449         for (int i = 0; i < size; i++)
1450         {
1451             if (possibleValues[row][i].contains(value))
1452             {
1453                 possibleValues[row][i].remove(
1454                     possibleValues[row][i].indexOf(value));
1455             }
1456         }
1457     }
1458
1459
1460     private void removeImpossibleValuesCage(Cage cage,
1461                                         ArrayList<Integer> values)
1462     {
1463         for (int i = 0; i < cage.getSize(); i++)
1464         {
1465             removeImpossibleValuesCell(cage.getCells().get(i).getRow(),
1466                                         cage.getCells().get(i).getColumn(), values);
1467         }
1468     }
1469
1470     private void removeImpossibleValuesCell(int row, int column,
1471                                         ArrayList<Integer> values)
1472     {
1473         possibleValues[row][column].retainAll(values);
1474     }
1475
1476     private ArrayList<Integer> createRetainAllArray()
1477     {
1478         ArrayList<Integer> array = new ArrayList<Integer>();
1479         for (int i = 1; i <= size; i++)
1480         {
1481             array.add(i);
1482         }
1483         return array;
1484     }
1485
1486     public ArrayList<ArrayList<Integer>> getGridArrayList()
1487     {
1488         ArrayList<ArrayList<Integer>> gridArrayList = new ArrayList<>();
1489         for (int i = 0; i < size; i++)
1490         {
1491             ArrayList<Integer> gridArrayListRow = new ArrayList<Integer>();
1492             for (int j = 0; j < size; j++)
1493             {
1494                 gridArrayListRow.add(grid.getCellValue(i, j));
1495             }
1496             gridArrayList.add(gridArrayListRow);
1497         }
1498         return gridArrayList;
1499     }
1500
1501     public Grid getGrid()
1502     {
1503         return grid;
1504     }
1505
1506     public Grid getSolution()
1507     {
1508         return solution;
1509     }
1510
1511     private void printGrid()
1512     {
1513         Cell[][] cells = grid.getGridContents();
1514         for (int i = 0; i < size; i++)
1515         {
1516             for (int j = 0; j < size; j++)
1517             {

```

```

1518         System.out.print(cells[i][j].getValue() + " ");
1519     }
1520     System.out.println(" ");
1521   }
1522 System.out.println(" ");
1523 }
1524
1525 private void printPossibleValues()
1526 {
1527   for (int i = 0; i < possibleValues.length; i++)
1528   {
1529     for (int j = 0; j < possibleValues[i].length; j++)
1530     {
1531       System.out.println("Row" + i + " , Column" + j);
1532       for (int k = 0; k < possibleValues[i][j].size(); k++)
1533       {
1534         System.out.print(possibleValues[i][j].get(k) + " ");
1535       }
1536       System.out.println(" ");
1537     }
1538   }
1539 System.out.println(" ");
1540 }
1541 }
1542 }
1543 }
```

D.7 SolverGenetic.java

Listing D.7: SolverGenetic.java

```

1 package model;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.Random;
6
7 public class SolverGenetic
8 {
9
10    private final Grid grid;
11    private final Integer size;
12    private final boolean[][] isCellFixed;
13    private final Random randomGenerator;
14    private final Integer generations;
15    private final Integer populationSize;
16    private final Double elitismRate;
17    private final Double mutationRate;
18    private final Double crossoverRate;
19    private Grid solution;
20    private ArrayList<Chromosome> currentGeneration =
21        new ArrayList<Chromosome>();
22    private ArrayList<Chromosome> nextGeneration = new ArrayList<Chromosome>();
23
24    public SolverGenetic(Grid grid, Integer generations,
25        Integer populationSize, double elitismRate, double crossoverRate,
26        double mutationRate)
27    {
28        this.grid = grid;
29        this.size = grid.getSize();
30        this.generations = generations;
31        this.populationSize = populationSize;
32        this.elitismRate = elitismRate;
33        this.crossoverRate = crossoverRate;
34        this.mutationRate = mutationRate;
35        this.isCellFixed = generateIsCellFixedArray();
36        this.randomGenerator = new Random();
37        generatePopulation();
38        for (int i = 0; i < currentGeneration.size(); i++)
39        {
40            printGrid(currentGeneration.get(i).getGrid().getGridContents());
41            System.out.println(currentGeneration.get(i).getFitness());
42        }
43    }
44
45    public boolean solve()
46    {
47        for (int i = 0; i < generations; i++)
48        {
49            solveLoop();
50            sortChromosomes();
51            for (int j = 0; j < populationSize; j++)
52            {
53                printGrid(
54                    currentGeneration.get(j).getGrid().getGridContents());
55                System.out.println(currentGeneration.get(j).getFitness());
56                if (currentGeneration.get(j).getFitness() == 1.0)
57                {
58                    this.solution = currentGeneration.get(j).getGrid();
59                    return true;
60                }
61            }
62        }
63    }
64 }
```

```

65    private void solveLoop()
66    {
67        int elitismNumber = (int) Math.round(populationSize * elitismRate);
68        int mutationNumber = (int) Math.round(populationSize * mutationRate);
69        int crossoverNumber
70            = (int) Math.round((populationSize * crossoverRate) / 2);
71        sortChromosomes();
72        for (int i = 0; i < populationSize; i++)
73        {
74            printGrid(currentGeneration.get(i).getGrid().getGridContents());
75            System.out.println(currentGeneration.get(i).getFitness());
76        }
77        for (int i = 0; i < elitismNumber; i++)
78        {
79            if (!nextGeneration.contains(currentGeneration.get(i)))
80            {
81                nextGeneration.add(cloneChromosome(currentGeneration.get(i)));
82            }
83        }
84        for (int i = 0; i < mutationNumber; i++)
85        {
86            Chromosome parent = randomSelection(currentGeneration);
87            nextGeneration.add(mutation(parent));
88        }
89        for (int i = 0; i < crossoverNumber; i++)
90        {
91            nextGeneration.addAll(crossover(randomSelection(currentGeneration),
92                                            randomSelection(currentGeneration)));
93        }
94        if (nextGeneration.size() < populationSize)
95        {
96            while (nextGeneration.size() < populationSize)
97            {
98                nextGeneration.add(randomSelection(currentGeneration));
99            }
100       }
101      if (nextGeneration.size() > populationSize)
102      {
103          int extraChromosomes = nextGeneration.size() - populationSize;
104          for (int i = 0; i < extraChromosomes; i++)
105          {
106              int index = randomGenerator.nextInt(nextGeneration.size());
107              nextGeneration.remove(index);
108          }
109      }
110      currentGeneration = nextGeneration;
111      nextGeneration = new ArrayList<Chromosome>();
112    }
113
114    private boolean[][] generateIsCellFixedArray()
115    {
116        boolean[][] array = new boolean[size][size];
117        for (int i = 0; i < size; i++)
118        {
119            for (int j = 0; j < size; j++)
120            {
121                array[i][j] = grid.getCellValue(i, j) != null;
122            }
123        }
124        return array;
125    }
126
127    private void generatePopulation()
128    {
129        for (int i = 0; i < populationSize; i++)
130        {
131            currentGeneration.add(generateChromosome());
132        }
133    }
134
135    private Chromosome generateChromosome()
136    {
137        Grid chromosomeGrid = new Grid(size, grid.getNumberOfCages(),
138                                       grid.getCageCells(), grid.getCageObjectives());
139        for (int i = 0; i < size; i++)
140        {
141            for (int j = 0; j < size; j++)
142            {
143                if (grid.getCellValue(i, j) != null)
144                {
145                    chromosomeGrid.solverSetCellValue(i, j,
146                        grid.getCellValue(i, j));
147                }
148            }
149        }
150        for (int i = 0; i < size; i++)
151        {
152            for (int j = 0; j < size; j++)
153            {
154                if (!grid.getRow(i).contains(j + 1))
155                {
156                    int index = randomGenerator.nextInt(size);
157                    while (chromosomeGrid.getCellValue(i, index) != null)
158                    {
159                        index = randomGenerator.nextInt(size);
160                    }
161                    chromosomeGrid.solverSetCellValue(i, index, j + 1);
162                }
163            }

```

```

164        }
165    }
166    Chromosome c = new Chromosome(chromosomeGrid);
167    return c;
168 }
169
170 private void sortChromosomes()
171 {
172     Collections.sort(currentGeneration,
173                     new ChromosomeComparator().reversed());
174 }
175
176 private Chromosome randomSelection(ArrayList<Chromosome> chromosomes)
177 {
178     int randomIndex = randomGenerator.nextInt(chromosomes.size());
179     Chromosome c = cloneChromosome(chromosomes.get(randomIndex));
180     return c;
181 }
182
183 private Chromosome cloneChromosome(Chromosome c)
184 {
185     Grid gridClone = new Grid(size, grid.getNumberOfCages(),
186                               grid.getCageCells(), grid.getCageObjectives());
187     for (int i = 0; i < size; i++)
188     {
189         for (int j = 0; j < size; j++)
190         {
191             gridClone.solverSetCellValue(i, j,
192                                         c.getGrid().getCellValue(i, j));
193         }
194     }
195     Chromosome chromosomeClone = new Chromosome(gridClone);
196     return chromosomeClone;
197 }
198
199 private ArrayList<Chromosome> crossover(Chromosome parent1,
200                                            Chromosome parent2)
201 {
202     while (parent1 == parent2 || parent1.equals(parent2))
203     {
204         parent1 = randomSelection(currentGeneration);
205         parent2 = randomSelection(currentGeneration);
206     }
207     Grid childGrid1 = new Grid(size, grid.getNumberOfCages(),
208                               grid.getCageCells(), grid.getCageObjectives());
209     Grid childGrid2 = new Grid(size, grid.getNumberOfCages(),
210                               grid.getCageCells(), grid.getCageObjectives());
211     for (int i = 0; i < size; i++)
212     {
213         int randomIndex = randomGenerator.nextInt(2);
214         for (int j = 0; j < size; j++)
215         {
216             if (randomIndex == 0)
217             {
218                 childGrid1.solverSetCellValue(i, j,
219                                             parent1.getGrid().getCellValue(i, j));
220                 childGrid2.solverSetCellValue(i, j,
221                                             parent2.getGrid().getCellValue(i, j));
222             }
223             else
224             {
225                 childGrid1.solverSetCellValue(i, j,
226                                             parent2.getGrid().getCellValue(i, j));
227                 childGrid2.solverSetCellValue(i, j,
228                                             parent1.getGrid().getCellValue(i, j));
229             }
230         }
231     }
232     ArrayList<Chromosome> childChromosomes = new ArrayList<Chromosome>();
233     Chromosome child1 = new Chromosome(childGrid1);
234     Chromosome child2 = new Chromosome(childGrid2);
235     childChromosomes.add(child1);
236     childChromosomes.add(child2);
237     return childChromosomes;
238 }
239
240 private Chromosome mutation(Chromosome parent)
241 {
242     Grid childGrid = new Grid(size, grid.getNumberOfCages(),
243                               grid.getCageCells(), grid.getCageObjectives());
244     int randomRow = randomGenerator.nextInt(size);
245     int randomColumn1 = randomGenerator.nextInt(size);
246     int randomColumn2 = randomGenerator.nextInt(size);
247     while (isCellFixed[randomRow][randomColumn1] == true
248            || isCellFixed[randomRow][randomColumn2] == true
249            || randomColumn1 == randomColumn2)
250     {
251         randomRow = randomGenerator.nextInt(size);
252         randomColumn1 = randomGenerator.nextInt(size);
253         randomColumn2 = randomGenerator.nextInt(size);
254     }
255     for (int i = 0; i < size; i++)
256     {
257         if (i == randomRow)
258         {
259             childGrid.solverSetCellValue(i, randomColumn1,
260                                         parent.getGrid().getCellValue(i, randomColumn2));
261             childGrid.solverSetCellValue(i, randomColumn2,
262                                         parent.getGrid().getCellValue(i, randomColumn1));
263         }
264     }
265 }

```

```

263     }
264     for (int j = 0; j < size; j++)
265     {
266         if (childGrid.getCellValue(i, j) == null)
267         {
268             childGrid.solverSetCellValue(i, j,
269                 parent.getGrid().getCellValue(i, j));
270         }
271     }
272     Chromosome child = new Chromosome(childGrid);
273     return child;
274 }
275
276 public Grid getGrid()
277 {
278     return grid;
279 }
280
281 public Grid getSolution()
282 {
283     return solution;
284 }
285
286 private void printGrid(Cell[][] cells)
287 {
288     for (int i = 0; i < size; i++)
289     {
290         for (int j = 0; j < size; j++)
291         {
292             System.out.print(cells[i][j].getValue() + " ");
293         }
294         System.out.println(" ");
295     }
296     System.out.println(" ");
297 }
298
299 }
300 }
```

D.8 Chromosome.java

Listing D.8: Chromosome.java

```

1 package model;
2
3 import java.util.Comparator;
4
5 public class Chromosome
6 {
7
8     private final Grid grid;
9     private final int size;
10    private final double fitness;
11
12    public Chromosome(Grid grid)
13    {
14        this.grid = grid;
15        this.size = grid.getSize();
16        this.fitness = setFitness();
17    }
18
19    private double setFitness()
20    {
21        double numberOfValidCells = 0;
22        double numberOfCells = size * size;
23        for (int i = 0; i < size; i++)
24        {
25            for (int j = 0; j < size; j++)
26            {
27                if (grid.solverIsCellValueValid(i, j) == true)
28                {
29                    numberOfValidCells++;
30                }
31            }
32        }
33        double value = numberOfValidCells / numberOfCells;
34        return value;
35    }
36
37    public double getFitness()
38    {
39        return fitness;
40    }
41
42    public Grid getGrid()
43    {
44        return grid;
45    }
46
47 }
48
49 class ChromosomeComparator implements Comparator<Chromosome>
50 {
51     @Override
```

```

53|     public int compare(Chromosome c1, Chromosome c2)
54|     {
55|         if (c1.getFitness() - c2.getFitness() > 0)
56|         {
57|             return 1;
58|         }
59|         else if (c1.getFitness() - c2.getFitness() < 0)
60|         {
61|             return -1;
62|         }
63|         else
64|         {
65|             return 0;
66|         }
67|     }
68|
69}

```

D.9 Controller.java

Listing D.9: Controller.java

```

1 package controller;
2
3 import model.Grid;
4 import model.Cage;
5 import model.Cell;
6
7 public class Controller
8 {
9
10    private Grid g;
11
12    public Controller(Integer size, Integer numberOfWorks,
13                      Integer [][] cageCells, String [] cageObjectives)
14    {
15        g = new Grid(size, numberOfWorks, cageCells, cageObjectives);
16    }
17
18    public boolean setCellValue(int row, int column, int value)
19    {
20        return g.setCellValue(row, column, value);
21    }
22
23    public void unsetCellValue(int row, int column)
24    {
25        g.unsetCellValue(row, column);
26    }
27
28    public Boolean checkGrid()
29    {
30        return g.checkGrid();
31    }
32
33    public Integer getCellValue(int row, int column)
34    {
35        return g.setCellValue(row, column);
36    }
37
38    public Integer getSize()
39    {
40        return g.getSize();
41    }
42
43    public Integer getNumberOfWorks()
44    {
45        return g.getNumberOfWorks();
46    }
47
48    public Integer [][] getCageCells()
49    {
50        return g.getCageCells();
51    }
52
53    public String [] getCageObjectives()
54    {
55        return g.getCageObjectives();
56    }
57
58    public int getCageTargetNumber(int cageID)
59    {
60        return g.getCages () [cageID].getTargetNumber ();
61    }
62
63    public char getCageOperator(int cageID)
64    {
65        return g.getCages () [cageID].getOperator ();
66    }
67
68    public Cell [][] getGrid()
69    {
70        return g.getGridContents ();
71    }
72
73    public Cage [] getCages()

```

```

74|     {
75|         return g.getCages();
76|     }
77|
78|     public Grid getGame()
79|     {
80|         return g.getGame();
81|     }
82|
83}

```

D.10 Calcudoku.java

Listing D.10: Calcudoku.java

```

1 package view;
2
3 import controller.Controller;
4 import java.awt.Dimension;
5 import java.awt.EventQueue;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.WindowAdapter;
8 import java.awt.event.WindowEvent;
9 import java.io.File;
10 import java.io.FileNotFoundException;
11 import java.util.NoSuchElementException;
12 import java.util.Scanner;
13 import javax.swing.JFileChooser;
14 import javax.swing.JFrame;
15 import javax.swing.JMenu;
16 import javax.swing.JMenuBar;
17 import javax.swing.JMenuItem;
18 import javax.swing.JOptionPane;
19 import javax.swing.filechooser.FileFilter;
20
21 public class Calcudoku extends JFrame
22 {
23
24     private File puzzleFile;
25     private Integer size;
26     private Integer numberOfCages;
27     private Integer[][] cageCells;
28     private String[] cageObjectives;
29     private Controller c;
30     private final JMenuBar menuBar;
31     private final JMenu menuFile;
32     private final JMenu menuSolve;
33     private final JMenuItem menuItemLoad;
34     private final JMenuItem menuItemReset;
35     private final JMenuItem menuItemClose;
36     private final JMenuItem menuItemCheck;
37     private final JMenuItem menuItemExit;
38     private final JMenuItem menuItemBacktracking;
39     private final JMenuItem menuItemHybridGenetic;
40     private final JMenuItem menuItemGeneticParameters;
41     private final JFileChooser fileChooser;
42     private GUI gui;
43
44     public Calcudoku()
45     {
46         this.menuBar = new JMenuBar();
47         this.menuFile = new JMenu();
48         this.menuSolve = new JMenu();
49         this.menuItemLoad = new JMenuItem();
50         this.menuItemReset = new JMenuItem();
51         this.menuItemClose = new JMenuItem();
52         this.menuItemCheck = new JMenuItem();
53         this.menuItemExit = new JMenuItem();
54         this.menuItemBacktracking = new JMenuItem();
55         this.menuItemHybridGenetic = new JMenuItem();
56         this.menuItemGeneticParameters = new JMenuItem();
57         this.fileChooser = new JFileChooser();
58         initComponents();
59     }
60
61     public static void main(String args[])
62     {
63         EventQueue.invokeLater(() ->
64         {
65             new Calcudoku().setVisible(true);
66         });
67     }
68
69     private void initComponents()
70     {
71         this.setTitle("Calcudoku");
72         this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
73         this.setMinimumSize(new Dimension(216, 216));
74         this.setLocationRelativeTo(null);
75         this.setResizable(false);
76         initWindowListener();
77         initActionListeners();
78         initMenu();
79         initMenuBar();
80         this.validate();
81     }

```



```

180         clearVariables();
181         JOptionPane.showMessageDialog(null,
182             "Error_in_loading_puzzle_file .", "Error",
183             JOptionPane.ERROR_MESSAGE);
184         throw new IllegalStateException("Error_in_loading_"
185             + "puzzle_file .");
186     }
187 }
188 else
189 {
190     resetFrame();
191     clearVariables();
192     JOptionPane.showMessageDialog(null, "Invalid_puzzle_file .",
193         "Error", JOptionPane.ERROR_MESSAGE);
194     throw new IllegalStateException("Invalid_puzzle_file .");
195 }
196
197 catch (FileNotFoundException fnfe)
198 {
199     resetFrame();
200     clearVariables();
201     JOptionPane.showMessageDialog(null, "Puzzle_file_not_found .",
202         "Error", JOptionPane.ERROR_MESSAGE);
203     throw new IllegalStateException("Puzzle_file_not_found .");
204 }
205
206
207 private void menuItemResetActionPerformed(ActionEvent evt)
208 {
209     if (c == null || gui == null)
210     {
211         resetFrame();
212         JOptionPane.showMessageDialog(null, "Puzzle_file_not_loaded .",
213             "Error", JOptionPane.ERROR_MESSAGE);
214         throw new IllegalStateException("Puzzle_file_not_loaded .");
215     }
216     else
217     {
218         if (JOptionPane.showConfirmDialog(null,
219             "Are you sure you want to reset this puzzle?", "Reset_Puzzle",
220             JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
221         {
222             resetFrame();
223             this.c = new Controller(size, numberOfCages, cageCells,
224                 cageObjectives);
225             this.gui = new GUI(c);
226             this.getContentPane().add(gui);
227             this.validate();
228             this.revalidate();
229             this.pack();
230             this.setLocationRelativeTo(null);
231             this.setTitle("Calcudoku(" + puzzleFile.getName() + ")");
232         }
233     }
234 }
235
236 private void menuItemCheckActionPerformed(ActionEvent evt)
237 {
238     if (c == null || gui == null)
239     {
240         JOptionPane.showMessageDialog(null, "Puzzle_file_not_loaded .",
241             "Error", JOptionPane.ERROR_MESSAGE);
242         throw new IllegalStateException("Puzzle_file_not_loaded .");
243     }
244     else
245     {
246         c.checkGrid();
247     }
248 }
249
250 private void menuItemCloseActionPerformed(ActionEvent evt)
251 {
252     if (c == null || gui == null)
253     {
254         JOptionPane.showMessageDialog(null, "Puzzle_file_not_loaded .",
255             "Error", JOptionPane.ERROR_MESSAGE);
256         throw new IllegalStateException("Puzzle_file_not_loaded .");
257     }
258     else
259     {
260         if (JOptionPane.showConfirmDialog(null,
261             "Are you sure you want to close this puzzle_file ?",
262             "Close_Puzzle_File", JOptionPane.YES_NO_OPTION)
263             == JOptionPane.YES_OPTION)
264         {
265             resetFrame();
266             clearVariables();
267         }
268     }
269 }
270
271 private void menuItemExitActionPerformed(ActionEvent evt)
272 {
273     if (JOptionPane.showConfirmDialog(null,
274         "Are you sure you want to exit this application?", "Exit",
275         JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
276     {
277         destroyFrame();
278     }
}

```

```

279 }
280
281     private void menuItemBacktrackingActionPerformed(ActionEvent evt)
282     {
283         if (c == null || gui == null)
284         {
285             JOptionPane.showMessageDialog(null, "Puzzle file not loaded.", "Error", JOptionPane.ERROR_MESSAGE);
286             throw new IllegalStateException("Puzzle file not loaded.");
287         }
288     }
289     else
290     {
291         gui.solveBacktracking();
292     }
293 }
294
295     private void menuItemHybridGeneticActionPerformed(ActionEvent evt)
296     {
297         if (c == null || gui == null)
298         {
299             JOptionPane.showMessageDialog(null, "Puzzle file not loaded.", "Error", JOptionPane.ERROR_MESSAGE);
300             throw new IllegalStateException("Puzzle file not loaded.");
301         }
302     }
303     else
304     {
305         gui.solveHybridGenetic();
306     }
307 }
308
309     private void menuItemGeneticParametersActionPerformed(ActionEvent evt)
310     {
311         if (c == null || gui == null)
312         {
313             JOptionPane.showMessageDialog(null, "Puzzle file not loaded.", "Error", JOptionPane.ERROR_MESSAGE);
314             throw new IllegalStateException("Puzzle file not loaded.");
315         }
316     }
317     else
318     {
319         GeneticParameters gp = new GeneticParameters(gui);
320         gp.setVisible(true);
321     }
322 }
323
324     private void loadPuzzleFile(File puzzleFile) throws FileNotFoundException
325     {
326         resetFrame();
327         clearVariables();
328         try
329         {
330             try (Scanner sc = new Scanner(puzzleFile))
331             {
332                 this.size = sc.nextInt();
333                 this.cageCells = new Integer[size][size];
334                 this.numberOfCages = sc.nextInt();
335                 this.cageObjectives = new String[numberOfCages];
336                 for (int i = 0; i < size; i++)
337                 {
338                     for (int j = 0; j < size; j++)
339                     {
340                         this.cageCells[i][j] = sc.nextInt();
341                     }
342                 }
343                 for (int i = 0; i < numberOfCages; i++)
344                 {
345                     this.cageObjectives[i] = sc.next();
346                 }
347                 if (sc.hasNext())
348                 {
349                     JOptionPane.showMessageDialog(null, "Invalid puzzle file.", "Error", JOptionPane.ERROR_MESSAGE);
350                     throw new IllegalStateException("Invalid puzzle file.");
351                 }
352                 sc.close();
353             }
354             this.c = new Controller(size, numberOfCages, cageCells,
355                                     cageObjectives);
356             this.gui = new GUI(c);
357             this.getContentPane().add(gui);
358             this.validate();
359             this.revalidate();
360             this.pack();
361             this.setLocationRelativeTo(null);
362             this.puzzleFile = puzzleFile;
363             this.setTitle("Calcudoku (" + puzzleFile.getName() + ")");
364         }
365         catch (NoSuchElementException nsee)
366         {
367             resetFrame();
368             clearVariables();
369             JOptionPane.showMessageDialog(null, "Invalid puzzle file.", "Error", JOptionPane.ERROR_MESSAGE);
370             throw new IllegalStateException("Invalid puzzle file.");
371         }
372     }
373
374     private void resetFrame()
375     {
376
377     }

```

```

378     this.getContentPane().removeAll();
379     this.c = null;
380     this.setTitle("Calcudoku");
381     this.validate();
382     this.revalidate();
383     this.pack();
384     this.setLocationRelativeTo(null);
385   }
386
387   private void clearVariables()
388   {
389     this.puzzleFile = null;
390     this.size = null;
391     this.cageCells = null;
392     this.numberOfCages = null;
393     this.cageObjectives = null;
394   }
395
396   public void destroyFrame()
397   {
398     resetFrame();
399     clearVariables();
400     this.dispose();
401   }
402 }
403
404 class WindowListener extends WindowAdapter
405 {
406
407   private final Calcudoku frame;
408
409   public WindowListener(Calcudoku frame)
410   {
411     this.frame = frame;
412   }
413
414   @Override
415   public void windowClosing(WindowEvent e)
416   {
417     if (JOptionPane.showConfirmDialog(null,
418         "Are you sure you want to exit the application?", "Exit",
419         JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
420     {
421       frame.destroyFrame();
422     }
423   }
424
425   @Override
426   public void windowClosed(WindowEvent e)
427   {
428     System.exit(0);
429   }
430 }
431
432 }
433
434 class PuzzleFileFilter extends FileFilter
435 {
436
437   @Override
438   public boolean accept(File puzzleFile)
439   {
440     return puzzleFile.isDirectory()
441        || puzzleFile.getAbsolutePath().endsWith(".txt");
442   }
443
444   @Override
445   public String getDescription()
446   {
447     return "Text documents (*.txt)";
448   }
449 }
450 }
```

D.11 GUI.java

Listing D.11: GUI.java

```

1 package view;
2
3 import controller.Controller;
4 import java.awt.Color;
5 import java.awt.Dimension;
6 import java.awt.Font;
7 import java.awt.GridLayout;
8 import java.awt.Point;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.awt.event.KeyEvent;
12 import java.awt.event.KeyListener;
13 import java.util.HashMap;
14 import java.util.Map;
15 import java.util.Objects;
16 import javax.swing.BorderFactory;
17 import javax.swing.JMenuItem;
```

```

18| import javax.swing.JOptionPane;
19| import javax.swing.JPanel;
20| import javax.swing.JPopupMenu;
21| import javax.swing.JTextField;
22| import javax.swing.event.DocumentEvent;
23| import javax.swing.event.DocumentListener;
24| import model.Cage;
25| import model.Cell;
26| import model.Grid;
27| import model.SolverBacktracking;
28| import model.SolverHybridGenetic;
29|
30| public class GUI extends JPanel
31{
32|
33|     private final Controller c;
34|     private final Grid game;
35|     private final Integer size;
36|     private final Integer numberOfcages;
37|     private final Integer[][] cageCells;
38|     private final String[] cageObjectives;
39|     private final Cell[][] grid;
40|     private final Cage[] cages;
41|     private final JTextField[][] textFields;
42|     private final Map<JTextField, Point> textFieldCoordinates;
43|     private final CellTextFieldListener[][] cellTextFieldListeners;
44|     private final Font font;
45|     private final int cellSize;
46|     private final int cellBorderWidth;
47|     private final int cageBorderWidth;
48|     private Integer generations;
49|     private Integer populationSize;
50|     private Double elitismRate;
51|     private Double crossoverRate;
52|     private Double mutationRate;
53|
54|     public GUI(Controller c)
55{
56        this.c = c;
57        this.game = c.getGame();
58        this.size = c.getSize();
59        this.cageCells = c.getCageCells();
60        this.numberOfCages = c.getNumberOfCages();
61        this.cageObjectives = c.getCageObjectives();
62        this.grid = c.getGrid();
63        this.cages = c.getCages();
64        this.textFields = new JTextField[size][size];
65        this.textFieldCoordinates = new HashMap<>();
66        this.cellTextFieldListeners = new CellTextFieldListener[size][size];
67        this.font = new Font("Courier New", Font.CENTER_BASELINE, 36);
68        this.cellSize = 72;
69        this.cellBorderWidth = 1;
70        this.cageBorderWidth = 3;
71        initComponents();
72    }
73|
74|     private void initComponents()
75{
76        this.setPreferredSize(new Dimension(cellSize * size, cellSize * size));
77        this.setLayout(new GridLayout(size, size));
78        initTextFields();
79    }
80|
81|     public Controller getController()
82{
83        return c;
84    }
85|
86|     public Integer getGridSize()
87{
88        return size;
89    }
90|
91|     public Integer getNumberOfCages()
92{
93        return numberOfCages;
94    }
95|
96|     public Integer[][] getCageCells()
97{
98        return cageCells;
99    }
100|
101|    public String[] getCageObjectives()
102{
103        return cageObjectives;
104    }
105|
106|    public Cell[][] getGrid()
107{
108        return grid;
109    }
110|
111|    public Cage[] getCages()
112{
113        return cages;
114    }
115|
116|    public void setGeneticAlgorithmParameters(Integer generations,

```

```

117         Integer populationSize , Double elitismRate , Double crossoverRate ,
118         Double mutationRate )
119     {
120         this .generations = generations ;
121         this .populationSize = populationSize ;
122         this .elitismRate = elitismRate ;
123         this .crossoverRate = crossoverRate ;
124         this .mutationRate = mutationRate ;
125     }
126
127     private void initTextFields ()
128     {
129         for ( int y = 0; y < size ; y++)
130         {
131             for ( int x = 0; x < size ; x++)
132             {
133                 JTextField textField = new JTextField () ;
134                 textField.addKeyListener (new CellKeyListener (this )) ;
135                 cellTextFieldListeners [y] [x] = new CellTextFieldListener (c ,
136                     textField , y , x) ;
137                 textField.getDocument () .addDocumentListener (
138                     cellTextFieldListeners [y] [x]) ;
139                 textFieldCoordinates .put (textField , new Point (x , y)) ;
140                 textFields [y] [x] = textField ;
141             }
142         }
143         for ( int y = 0; y < size ; y++)
144         {
145             for ( int x = 0; x < size ; x++)
146             {
147                 JTextField textField = textFields [y] [x] ;
148                 int cageID ;
149                 int topBorderWidth ;
150                 int leftBorderWidth ;
151                 int bottomBorderWidth ;
152                 int rightBorderWidth ;
153                 if (y == 0)
154                 {
155                     topBorderWidth = cageBorderWidth ;
156                     if (Objects .equals (cageCells [y] [x] , cageCells [y + 1] [x]))
157                     {
158                         bottomBorderWidth = cellBorderWidth ;
159                     }
160                     else
161                     {
162                         bottomBorderWidth = cageBorderWidth ;
163                     }
164                 }
165                 else if (y == size - 1)
166                 {
167                     bottomBorderWidth = cageBorderWidth ;
168                     if (Objects .equals (cageCells [y] [x] , cageCells [y - 1] [x]))
169                     {
170                         topBorderWidth = cellBorderWidth ;
171                     }
172                     else
173                     {
174                         topBorderWidth = cageBorderWidth ;
175                     }
176                 }
177                 else
178                 {
179                     if (Objects .equals (cageCells [y] [x] , cageCells [y + 1] [x]))
180                     {
181                         bottomBorderWidth = cellBorderWidth ;
182                     }
183                     else
184                     {
185                         bottomBorderWidth = cageBorderWidth ;
186                     }
187                     if (Objects .equals (cageCells [y] [x] , cageCells [y - 1] [x]))
188                     {
189                         topBorderWidth = cellBorderWidth ;
190                     }
191                     else
192                     {
193                         topBorderWidth = cageBorderWidth ;
194                     }
195                 }
196                 if (x == 0)
197                 {
198                     leftBorderWidth = cageBorderWidth ;
199                     if (Objects .equals (cageCells [y] [x] , cageCells [y] [x + 1]))
200                     {
201                         rightBorderWidth = cellBorderWidth ;
202                     }
203                     else
204                     {
205                         rightBorderWidth = cageBorderWidth ;
206                     }
207                 }
208                 else if (x == size - 1)
209                 {
210                     rightBorderWidth = cageBorderWidth ;
211                     if (Objects .equals (cageCells [y] [x] , cageCells [y] [x - 1]))
212                     {
213                         leftBorderWidth = cellBorderWidth ;
214                     }
215                 }

```

```

216         {
217             leftBorderWidth = cageBorderWidth;
218         }
219     }
220     else
221     {
222         if (Objects.equals(cageCells[y][x], cageCells[y][x + 1]))
223         {
224             rightBorderWidth = cellBorderWidth;
225         }
226         else
227         {
228             rightBorderWidth = cageBorderWidth;
229         }
230         if (Objects.equals(cageCells[y][x], cageCells[y][x - 1]))
231         {
232             leftBorderWidth = cellBorderWidth;
233         }
234         else
235         {
236             leftBorderWidth = cageBorderWidth;
237         }
238     }
239     cageID = grid[y][x].getCageID();
240     textField.setToolTipText(cages[cageID].getObjective());
241     textField.setBorder(BorderFactory.createMatteBorder(
242         topBorderWidth, leftBorderWidth, bottomBorderWidth,
243         rightBorderWidth, Color.BLACK));
244     textField.setFont(font);
245     textField.setHorizontalAlignment(JTextField.CENTER);
246     textField.setPreferredSize(new Dimension(cellSize, cellSize));
247     JPopupMenu popupMenu = new JPopupMenu();
248     for (int i = 1; i <= size; i++)
249     {
250         JMenuItem menuItem = new JMenuItem(" " + i);
251         popupMenu.add(menuItem);
252         menuItem.addActionListener(new PopupMenuItemListener(textField,
253             i));
254     }
255     textField.add(popupMenu);
256     textField.setComponentPopupMenu(popupMenu);
257 }
258 }
259 for (int y = 0; y < size; y++)
260 {
261     for (int x = 0; x < size; x++)
262     {
263         this.add(textFields[y][x]);
264     }
265 }
266
267
268 public void solveBacktracking()
269 {
270     removeCellTextFieldListeners();
271     clearGrid();
272     Cell[][][] solution;
273     float startTime = System.nanoTime();
274     float endTime;
275     float duration;
276     SolverBacktracking sb = new SolverBacktracking(game);
277     if (sb.solve() == true)
278     {
279         solution = sb.getSolution().getGridContents();
280         printGridToScreen(solution);
281         endTime = System.nanoTime();
282         duration = (endTime - startTime) / 1000000000;
283         System.out.println("The backtracking algorithm has successfully "
284             + "solved the puzzle." + "\nTime elapsed: " + duration
285             + " seconds");
286         JOptionPane.showMessageDialog(null,
287             "The backtracking algorithm has successfully solved the "
288             + "puzzle." + "\nTime elapsed: " + duration
289             + " seconds", "Information",
290             JOptionPane.INFORMATION_MESSAGE);
291     }
292     else
293     {
294         endTime = System.nanoTime();
295         duration = (endTime - startTime) / 1000000000;
296         System.out.println("The backtracking algorithm has failed to solve "
297             + "the puzzle." + "\nTime elapsed: " + duration
298             + " seconds");
299         JOptionPane.showMessageDialog(null,
300             "The backtracking algorithm has failed to solve the "
301             + "puzzle." + "\nTime elapsed: " + duration
302             + " seconds", "Information",
303             JOptionPane.INFORMATION_MESSAGE);
304     }
305     addCellTextFieldListeners();
306 }
307
308 public void solveHybridGenetic()
309 {
310     if (generations == null || populationSize == null
311         || elitismRate == null || crossoverRate == null
312         || mutationRate == null)
313     {
314         JOptionPane.showMessageDialog(null,

```

```

315             "Genetic\u00a5algorithm\u00a5parameters\u00a5have\u00a5not\u00a5been\u00a5set.\u00a5",
316             "Error", JOptionPane.ERROR_MESSAGE);
317     throw new IllegalStateException(
318         "Genetic\u00a5algorithm\u00a5parameters\u00a5have\u00a5not\u00a5been\u00a5set.\u00a5");
319 }
320 else
321 {
322     removeCellTextFieldListeners();
323     clearGrid();
324     Cell [][] solution;
325     float startTime = System.nanoTime();
326     float endTime;
327     float duration;
328     SolverHybridGenetic shg = new SolverHybridGenetic(game, generations,
329             populationSize, elitismRate, crossoverRate, mutationRate);
330     if (shg.solve() == true)
331     {
332         solution = shg.getSolution().getGridContents();
333         printGridToScreen(solution);
334         endTime = System.nanoTime();
335         duration = (endTime - startTime) / 1000000000;
336         System.out.println("The\u00a5hybrid\u00a5genetic\u00a5algorithm\u00a5has\u00a5successfully\u00a5
337             + "solved\u00a5the\u00a5puzzle.\u00a5" + "\nTime\u00a5elapsed:\u00a5" + duration
338             + "\u00a5seconds");
339         JOptionPane.showMessageDialog(null,
340             "The\u00a5hybrid\u00a5genetic\u00a5algorithm\u00a5has\u00a5successfully\u00a5solved\u00a5the\u00a5
341             + "puzzle.\u00a5" + "\nTime\u00a5elapsed:\u00a5" + duration
342             + "\u00a5seconds", "Information",
343             JOptionPane.INFORMATION_MESSAGE);
344     }
345     else
346     {
347         endTime = System.nanoTime();
348         duration = (endTime - startTime) / 1000000000;
349         System.out.println("The\u00a5hybrid\u00a5genetic\u00a5algorithm\u00a5has\u00a5failed\u00a5to\u00a5
350             + "solve\u00a5the\u00a5puzzle.\u00a5" + "\nTime\u00a5elapsed:\u00a5" + duration
351             + "\u00a5seconds");
352         JOptionPane.showMessageDialog(null,
353             "The\u00a5hybrid\u00a5genetic\u00a5algorithm\u00a5has\u00a5failed\u00a5to\u00a5solve\u00a5the\u00a5 +
354             "puzzle.\u00a5" + "\nTime\u00a5elapsed:\u00a5" + duration
355             + "\u00a5seconds", "Information",
356             JOptionPane.INFORMATION_MESSAGE);
357     }
358     addCellTextFieldListeners();
359 }
360 }
361
362 private void printGridToScreen (Cell [][] grid)
363 {
364     for (int x = 0; x < size; x++)
365     {
366         for (int y = 0; y < size; y++)
367         {
368             String value = Integer.toString(grid [x] [y].getValue ());
369             textFields [x] [y].setText (value);
370         }
371     }
372 }
373
374 private void clearGrid ()
375 {
376     for (int i = 0; i < size; i++)
377     {
378         for (int j = 0; j < size; j++)
379         {
380             c.unsetCellValue(i, j);
381         }
382     }
383 }
384
385 private void addCellTextFieldListeners ()
386 {
387     for (int x = 0; x < size; x++)
388     {
389         for (int y = 0; y < size; y++)
390         {
391             textFields [x] [y].getDocument ().addDocumentListener (
392                 cellTextFieldListeners [x] [y]);
393         }
394     }
395 }
396
397 private void removeCellTextFieldListeners ()
398 {
399     for (int x = 0; x < size; x++)
400     {
401         for (int y = 0; y < size; y++)
402         {
403             textFields [x] [y].getDocument ().removeDocumentListener (
404                 cellTextFieldListeners [x] [y]);
405         }
406     }
407 }
408
409 public void moveCursor (JTextField textField, int keyCode)
410 {
411     Point coordinates = textFieldCoordinates.get (textField);
412     switch (keyCode)
413     {

```

```

414     case KeyEvent.VK_LEFT:
415         if (coordinates.x > 0)
416         {
417             textFields [ coordinates.y ][ coordinates.x - 1].requestFocus ();
418         }
419         break;
420     case KeyEvent.VK_KP_LEFT:
421         if (coordinates.x > 0)
422         {
423             textFields [ coordinates.y ][ coordinates.x - 1].requestFocus ();
424         }
425         break;
426     case KeyEvent.VK_UP:
427         if (coordinates.y > 0)
428         {
429             textFields [ coordinates.y - 1][ coordinates.x ].requestFocus ();
430         }
431         break;
432     case KeyEvent.VK_KP_UP:
433         if (coordinates.y > 0)
434         {
435             textFields [ coordinates.y - 1][ coordinates.x ].requestFocus ();
436         }
437         break;
438     case KeyEvent.VK_RIGHT:
439         if (coordinates.x < size - 1)
440         {
441             textFields [ coordinates.y ][ coordinates.x + 1].requestFocus ();
442         }
443         break;
444     case KeyEvent.VK_KP_RIGHT:
445         if (coordinates.x < size - 1)
446         {
447             textFields [ coordinates.y ][ coordinates.x + 1].requestFocus ();
448         }
449         break;
450     case KeyEvent.VK_DOWN:
451         if (coordinates.y < size - 1)
452         {
453             textFields [ coordinates.y + 1][ coordinates.x ].requestFocus ();
454         }
455         break;
456     case KeyEvent.VK_KP_DOWN:
457         if (coordinates.y < size - 1)
458         {
459             textFields [ coordinates.y + 1][ coordinates.x ].requestFocus ();
460         }
461         break;
462     }
463 }
464
465 }
466
467 class CellKeyListener implements KeyListener
468 {
469
470     private final GUI gui;
471
472     CellKeyListener(GUI gui)
473     {
474         this.gui = gui;
475     }
476
477     @Override
478     public void keyPressed(KeyEvent e)
479     {
480         int keyCode = e.getKeyCode();
481         JTextField textField = (JTextField) e.getSource();
482         switch (keyCode)
483         {
484             case KeyEvent.VK_LEFT :
485                 e.consume();
486                 gui.moveCursor(textField , KeyEvent.VK_LEFT);
487                 break;
488             case KeyEvent.VK_UP :
489                 e.consume();
490                 gui.moveCursor(textField , KeyEvent.VK_UP);
491                 break;
492             case KeyEvent.VK_RIGHT :
493                 e.consume();
494                 gui.moveCursor(textField , KeyEvent.VK_RIGHT);
495                 break;
496             case KeyEvent.VK_DOWN :
497                 e.consume();
498                 gui.moveCursor(textField , KeyEvent.VK_DOWN);
499                 break;
500             case KeyEvent.VK_KP_LEFT :
501                 e.consume();
502                 gui.moveCursor(textField , KeyEvent.VK_KP_LEFT);
503                 break;
504             case KeyEvent.VK_KP_UP :
505                 e.consume();
506                 gui.moveCursor(textField , KeyEvent.VK_KP_UP);
507                 break;
508             case KeyEvent.VK_KP_RIGHT :
509                 e.consume();
510                 gui.moveCursor(textField , KeyEvent.VK_KP_RIGHT);
511                 break;
512             case KeyEvent.VK_KP_DOWN :

```

```

513             e.consume();
514             gui.moveCursor(textField, KeyEvent.VK_KP_DOWN);
515         }
516     }
517 }
518
519 @Override
520 public void keyTyped(KeyEvent e)
521 {
522     JTextField textField = (JTextField) e.getSource();
523     char c = e.getKeyChar();
524     if (!Character.isDigit(c))
525     {
526         e.consume();
527     }
528     String gridSize = ":" + gui.getGridSize();
529     int gridSizeDigits = gridSize.length();
530     if (textField.getText().length() >= gridSizeDigits)
531     {
532         e.consume();
533     }
534 }
535
536 @Override
537 public void keyReleased(KeyEvent e)
538 {
539 }
540
541 }
542 }
543
544 class PopupMenuItemListener implements ActionListener
545 {
546
547     private final JTextField textField;
548     private final int number;
549
550     PopupMenuItemListener(JTextField textField, int number)
551     {
552         this.textField = textField;
553         this.number = number;
554     }
555
556     @Override
557     public void actionPerformed(ActionEvent e)
558     {
559         textField.setText(number + ":");
560     }
561 }
562 }
563
564 class CellTextFieldListener implements DocumentListener
565 {
566
567     private final Controller c;
568     private final JTextField textField;
569     private final int x;
570     private final int y;
571
572     public CellTextFieldListener(Controller c, JTextField textField, int x,
573                                 int y)
574     {
575         this.c = c;
576         this.textField = textField;
577         this.x = x;
578         this.y = y;
579     }
580
581     @Override
582     public void insertUpdate(DocumentEvent e)
583     {
584         Integer value = Integer.parseInt(textField.getText());
585         c.setCellValue(x, y, value);
586     }
587
588     @Override
589     public void removeUpdate(DocumentEvent e)
590     {
591         c.unsetCellValue(x, y);
592     }
593
594     @Override
595     public void changedUpdate(DocumentEvent e)
596     {
597         Integer value = Integer.parseInt(textField.getText());
598         c.unsetCellValue(x, y);
599         c.setCellValue(x, y, value);
600     }
601 }
602 }
```

D.12 GeneticParameters.java

Listing D.12: GeneticParameters.java

```
1| package view;
```

```

2
3 import java.awt.event.ActionEvent;
4 import javax.swing.GroupLayout;
5 import javax.swing.JButton;
6 import javax.swing.JFrame;
7 import javax.swing.JLabel;
8 import javax.swing.JOptionPane;
9 import javax.swing.JTextField;
10 import javax.swing.SwingConstants;
11 import javax.swing.WindowConstants;
12
13 public class GeneticParameters extends JFrame
14 {
15
16     private final GUI gui;
17     private final JLabel labelGenerations;
18     private final JLabel labelPopulation;
19     private final JLabel labelElitism;
20     private final JLabel labelCrossover;
21     private final JLabel labelMutation;
22     private final JTextField textFieldGenerations;
23     private final JTextField textFieldPopulation;
24     private final JTextField textFieldElitism;
25     private final JTextField textFieldCrossover;
26     private final JTextField textFieldMutation;
27     private final JButton buttonOK;
28     private final JButton buttonCancel;
29
30     /**
31      * Creates new form Test
32     */
33     public GeneticParameters(GUI gui)
34     {
35         this.gui = gui;
36         labelGenerations = new JLabel();
37         labelPopulation = new JLabel();
38         labelElitism = new JLabel();
39         labelCrossover = new JLabel();
40         labelMutation = new JLabel();
41         textFieldGenerations = new JTextField();
42         textFieldPopulation = new JTextField();
43         textFieldElitism = new JTextField();
44         textFieldCrossover = new JTextField();
45         textFieldMutation = new JTextField();
46         buttonOK = new JButton();
47         buttonCancel = new JButton();
48         initComponents();
49         this.setVisible(true);
50     }
51
52     private void initComponents()
53     {
54         this.setTitle("Set Genetic Algorithm Parameters");
55         this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
56         this.setLocationRelativeTo(null);
57         this.setResizable(false);
58         labelGenerations.setText("Generations");
59         labelPopulation.setText("Population Size");
60         labelElitism.setText("Elitism Rate");
61         labelCrossover.setText("Crossover Rate");
62         labelMutation.setText("Mutation Rate");
63         buttonOK.setText("OK");
64         buttonCancel.setText("Cancel");
65         initGUI();
66         initActionListeners();
67     }
68
69     private void initActionListeners()
70     {
71         this.buttonOK.addActionListener(this::buttonOKActionPerformed);
72         this.buttonCancel.addActionListener(this::buttonCancelActionPerformed);
73     }
74
75     private void initGUI()
76     {
77         GroupLayout layout = new GroupLayout(this.getContentPane());
78         this.getContentPane().setLayout(layout);
79         layout.setHorizontalGroup(
80             layout.createParallelGroup(GroupLayout.Alignment.LEADING)
81                 .addGroup(layout.createSequentialGroup()
82                     .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
83                         .addGroup(layout.createSequentialGroup()
84                             .addComponent(labelGenerations)
85                             .addComponent(labelPopulation)
86                             .addComponent(labelElitism)
87                             .addComponent(labelCrossover)
88                             .addComponent(labelMutation)
89                             .addComponent(buttonOK))
90                         .addGroup(layout.createSequentialGroup()
91                             .addGap(18, 18, 18)
92                             .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
93                                 .addComponent(textFieldGenerations, GroupLayout.PREFERRED_SIZE,
94                                     100, GroupLayout.PREFERRED_SIZE)
95                                 .addComponent(textFieldPopulation, GroupLayout.PREFERRED_SIZE,
96                                     100, GroupLayout.PREFERRED_SIZE)))
97                     .addGap(18, 18, 18)
98                 )
99             )
100        );

```

```

101                                     GroupLayout.PREFERRED_SIZE)
102                                     .addComponent(textFieldElitism,
103                                         GroupLayout.PREFERRED_SIZE,
104                                         100,
105                                         GroupLayout.PREFERRED_SIZE)
106                                     .addComponent(textFieldCrossover,
107                                         GroupLayout.PREFERRED_SIZE,
108                                         100,
109                                         GroupLayout.PREFERRED_SIZE)
110                                     .addComponent(textFieldMutation,
111                                         GroupLayout.PREFERRED_SIZE,
112                                         100,
113                                         GroupLayout.PREFERRED_SIZE)
114                                     .addComponent(buttonCancel))
115                                     .addContainerGap(GroupLayout.DEFAULT_SIZE,
116                                         Short.MAX_VALUE)));
117 layout.linkSize(SwingConstants.HORIZONTAL, buttonOK, buttonCancel);
118 layout.setVerticalGroup(
119     layout.createParallelGroup(GroupLayout.Alignment.LEADING)
120         .addGroup(layout.createSequentialGroup()
121             .addContainerGap())
122             .addGroup(layout.createParallelGroup(
123                 GroupLayout.Alignment.BASELINE)
124                 .addComponent(textFieldGenerations,
125                     GroupLayout.PREFERRED_SIZE,
126                     GroupLayout.DEFAULT_SIZE,
127                     GroupLayout.PREFERRED_SIZE)
128                 .addComponent(labelGenerations)))
129             .addGap(18, 18, 18)
130             .addGroup(layout.createParallelGroup(
131                 GroupLayout.Alignment.BASELINE)
132                 .addComponent(labelPopulation)
133                 .addComponent(textFieldPopulation,
134                     GroupLayout.PREFERRED_SIZE,
135                     GroupLayout.DEFAULT_SIZE,
136                     GroupLayout.PREFERRED_SIZE))
137             .addGap(18, 18, 18)
138             .addGroup(layout.createParallelGroup(
139                 GroupLayout.Alignment.BASELINE)
140                 .addComponent(labelElitism)
141                 .addComponent(textFieldElitism,
142                     GroupLayout.PREFERRED_SIZE,
143                     GroupLayout.DEFAULT_SIZE,
144                     GroupLayout.PREFERRED_SIZE))
145             .addGap(18, 18, 18)
146             .addGroup(layout.createParallelGroup(
147                 GroupLayout.Alignment.BASELINE)
148                 .addComponent(labelCrossover)
149                 .addComponent(textFieldCrossover,
150                     GroupLayout.PREFERRED_SIZE,
151                     GroupLayout.DEFAULT_SIZE,
152                     GroupLayout.PREFERRED_SIZE))
153             .addGap(18, 18, 18)
154             .addGroup(layout.createParallelGroup(
155                 GroupLayout.Alignment.BASELINE)
156                 .addComponent(labelMutation)
157                 .addComponent(textFieldMutation,
158                     GroupLayout.PREFERRED_SIZE,
159                     GroupLayout.DEFAULT_SIZE,
160                     GroupLayout.PREFERRED_SIZE))
161             .addGap(18, 18, 18)
162             .addGroup(layout.createParallelGroup(
163                 GroupLayout.Alignment.BASELINE)
164                 .addComponent(buttonOK)
165                 .addComponent(buttonCancel)))
166             .addContainerGap(GroupLayout.DEFAULT_SIZE,
167                                         Short.MAX_VALUE)));
168     this.validate();
169     this.revalidate();
170     this.pack();
171     this.setLocationRelativeTo(null);
172 }
173
174 private void buttonOKActionPerformed(ActionEvent evt)
175 {
176     try
177     {
178         Integer generations = Integer.parseInt(textFieldGenerations.getText());
179         Integer population = Integer.parseInt(textFieldPopulation.getText());
180         Double elitism = Double.parseDouble(textFieldElitism.getText());
181         Double crossover = Double.parseDouble(textFieldCrossover.getText());
182         Double mutation = Double.parseDouble(textFieldMutation.getText());
183         gui.setGeneticAlgorithmParameters(generations, population, elitism,
184             crossover, mutation);
185         destroyFrame();
186     }
187     catch (NumberFormatException nfe)
188     {
189         JOptionPane.showMessageDialog(null, "Invalid number format.",
190             "Error", JOptionPane.ERROR_MESSAGE);
191         throw new IllegalStateException("Invalid number format.");
192     }
193 }
194
195 private void buttonCancelActionPerformed(ActionEvent evt)
196 {
197     destroyFrame();
198 }
199

```

```
200|     private void destroyFrame()
201| {
202|     this.getContentPane().removeAll();
203|     this.validate();
204|     this.revalidate();
205|     this.pack();
206|     this.setLocationRelativeTo(null);
207|     this.dispose();
208| }
209|
210| }
```