

**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, «tanggal» «bulan» 2017

Menyetujui,

Pembimbing Utama

Pembimbing Pendamping

Dr.rer.nat. Cecilia Esti Nugraheni

«pembimbing 2»

Ketua Tim Penguji

Anggota Tim Penguji

«penguji 1»

«penguji 2»

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 «tanggal» «bulan» 2017

Meterai  
Rp. 6000

Michael Adrian  
NPM: 2013730039



## **ABSTRAK**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation 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.

**Kata-kata kunci:** lorem, ipsum



## ABSTRACT

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut labore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation 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.

**Keywords:** lorem, ipsum



*Lorem ipsum dolor sit amet, consectetur adipiscing elit.*



## KATA PENGANTAR

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation 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, consectetuer 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, «bulan» 2017

Penulis



# DAFTAR ISI

<b>KATA PENGANTAR</b>	<b>xv</b>
<b>DAFTAR ISI</b>	<b>xvii</b>
<b>DAFTAR GAMBAR</b>	<b>xx</b>
<b>DAFTAR TABEL</b>	<b>xxiv</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	3
1.3 Tujuan . . . . .	3
1.4 Batasan Masalah . . . . .	3
1.5 Metodologi Penelitian . . . . .	4
1.6 Sistematika Pembahasan . . . . .	4
<b>2 LANDASAN TEORI</b>	<b>7</b>
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 . . . . .	18
2.3.3 Algoritma <i>Hybrid Genetic</i> . . . . .	18
<b>3 ANALISIS</b>	<b>23</b>
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> . . . . .	28
3.2.2 Algoritma Genetik . . . . .	28
3.3 Perangkat Lunak . . . . .	44
3.3.1 Diagram <i>Use Case</i> dan Skenario . . . . .	45
3.3.2 Diagram Kelas . . . . .	48
<b>4 PERANCANGAN</b>	<b>51</b>
4.1 Perancangan Masukan . . . . .	51
4.2 Perancangan Keluaran . . . . .	51
4.3 Perancangan Antarmuka . . . . .	51
4.4 Diagram Kelas . . . . .	54
4.4.1 Kelas Grid . . . . .	57
4.4.2 Kelas Cage . . . . .	60
4.4.3 Kelas Cell . . . . .	63
4.4.4 Kelas SolverBacktracking . . . . .	64
4.4.5 Kelas SolverHybridGenetic . . . . .	65

4.4.6	Kelas SolverRuleBased . . . . .	65
4.4.7	Kelas SolverGenetic . . . . .	71
4.4.8	Kelas Chromosome . . . . .	72
4.4.9	Kelas ChromosomeComparator . . . . .	74
4.4.10	Kelas Controller . . . . .	74
4.4.11	Kelas Calcudoku . . . . .	76
4.4.12	Kelas WindowListener . . . . .	78
4.4.13	Kelas PuzzleFileFilter . . . . .	78
4.4.14	Kelas GUI . . . . .	80
4.4.15	Kelas CellKeyListener . . . . .	81
4.4.16	Kelas PopupMenuItemListener . . . . .	82
4.4.17	Kelas CellTextFieldListener . . . . .	83
4.4.18	Kelas GeneticParameters . . . . .	84
<b>5</b>	<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>87</b>
5.1	Lingkungan untuk Pengujian . . . . .	87
5.2	Implementasi . . . . .	87
5.2.1	Kode Program . . . . .	87
5.2.2	Antarmuka Perangkat Lunak . . . . .	88
5.3	Pengujian Fungsional . . . . .	91
5.4	Pengujian Keakuratan . . . . .	100
5.5	Pengujian Algoritma <i>Backtracking</i> . . . . .	103
5.6	Pengujian dan Eksperimen Algoritma <i>Hybrid Genetic</i> . . . . .	103
5.6.1	Skenario 1 . . . . .	104
5.6.2	Skenario 2 . . . . .	105
5.6.3	Skenario 3 . . . . .	105
5.6.4	Skenario 4 . . . . .	105
5.6.5	Skenario 5 . . . . .	106
5.6.6	Skenario 6 . . . . .	106
5.6.7	Skenario 7 . . . . .	106
5.6.8	Skenario 8 . . . . .	107
5.6.9	Skenario 9 . . . . .	107
5.6.10	Skenario 10 . . . . .	107
5.6.11	Skenario 11 . . . . .	108
5.6.12	Skenario 12 . . . . .	108
5.6.13	Skenario 13 . . . . .	108
5.6.14	Skenario 14 . . . . .	109
5.6.15	Skenario 15 . . . . .	109
5.6.16	Skenario 16 . . . . .	109
5.6.17	Hasil Eksperimen . . . . .	110
<b>6</b>	<b>SIMPULAN DAN SARAN</b>	<b>111</b>
6.1	Simpulan . . . . .	111
6.2	Saran . . . . .	111
<b>DAFTAR REFERENSI</b>	<b>113</b>	
<b>A ANALISIS ALGORITMA <i>Backtracking</i></b>	<b>115</b>	
<b>B HASIL PENGUJIAN</b>	<b>129</b>	
B.1	Algoritma <i>Backtracking</i> . . . . .	129
B.2	Algoritma <i>Hybrid Genetic</i> . . . . .	134

<b>C File TEKS SOAL-SOAL PERMAINAN CALCUDOKU UNTUK PENGUJIAN</b>	<b>143</b>
C.1 <i>File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran <math>4 \times 4</math></i>	143
C.2 <i>File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran <math>5 \times 5</math></i>	151
C.3 <i>File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran <math>6 \times 6</math></i>	157
C.4 <i>File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran <math>7 \times 7</math></i>	168
C.5 <i>File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran <math>8 \times 8</math></i>	174
C.6 <i>File Teks Soal-Soal Permainan Calcudoku dengan Grid Berukuran <math>9 \times 9</math></i>	180
<b>D KODE PROGRAM</b>	<b>183</b>

## 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 . . . . .	24
3.5	<i>State</i> 17 . . . . .	25
3.6	<i>State</i> 18 . . . . .	25
3.7	<i>State</i> 19 . . . . .	25
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 A.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 . . . . .	30
3.14	Kromosom 2 dalam Generasi ke-1 . . . . .	30

3.15 Kromosom 3 dalam Generasi ke-1 . . . . .	30
3.16 Kromosom 4 dalam Generasi ke-1 . . . . .	31
3.17 Kromosom 5 dalam Generasi ke-1 . . . . .	31
3.18 Kromosom 6 dalam Generasi ke-1 . . . . .	31
3.19 Kromosom 7 dalam Generasi ke-1 . . . . .	32
3.20 Kromosom 8 dalam Generasi ke-1 . . . . .	32
3.21 Kromosom 9 dalam Generasi ke-1 . . . . .	32
3.22 Kromosom 10 dalam Generasi ke-1 . . . . .	33
3.23 Kromosom 11 dalam Generasi ke-1 . . . . .	33
3.24 Kromosom 12 dalam Generasi ke-1 . . . . .	33
3.25 Kromosom 1 dalam Generasi ke-2 . . . . .	35
3.26 Kromosom 2 dalam Generasi ke-2 . . . . .	35
3.27 Kromosom 3 dalam Generasi ke-2 . . . . .	35
3.28 Kromosom 4 dalam Generasi ke-2 . . . . .	36
3.29 Kromosom 5 dalam Generasi ke-2 . . . . .	36
3.30 Kromosom 6 dalam Generasi ke-2 . . . . .	36
3.31 Kromosom 7 dalam Generasi ke-2 . . . . .	37
3.32 Kromosom 8 dalam Generasi ke-2 . . . . .	37
3.33 Kromosom 9 dalam Generasi ke-2 . . . . .	37
3.34 Kromosom 10 dalam Generasi ke-2 . . . . .	38
3.35 Kromosom 11 dalam Generasi ke-2 . . . . .	38
3.36 Kromosom 12 dalam Generasi ke-2 . . . . .	38
3.37 Kromosom 1 dalam Generasi ke-3 . . . . .	40
3.38 Kromosom 2 dalam Generasi ke-3 . . . . .	40
3.39 Kromosom 3 dalam Generasi ke-3 . . . . .	40
3.40 Kromosom 4 dalam Generasi ke-3 . . . . .	41
3.41 Kromosom 5 dalam Generasi ke-3 . . . . .	41
3.42 Kromosom 6 dalam Generasi ke-3 . . . . .	41
3.43 Kromosom 7 dalam Generasi ke-3 . . . . .	42
3.44 Kromosom 8 dalam Generasi ke-3 . . . . .	42
3.45 Kromosom 9 dalam Generasi ke-3 . . . . .	42
3.46 Kromosom 10 dalam Generasi ke-3 . . . . .	43
3.47 Kromosom 11 dalam Generasi ke-3 . . . . .	43
3.48 Kromosom 12 dalam Generasi ke-3 . . . . .	43
3.49 Diagram <i>use case</i> untuk perangkat lunak permainan teka-teki Calcudoku . . . . .	46
3.50 Diagram kelas untuk perangkat lunak permainan teka-teki Calcudoku . . . . .	49
4.1 Contoh <i>file</i> masukan . . . . .	52
4.2 Contoh keluaran . . . . .	52
4.3 Perancangan GUI sebelum <i>file</i> permainan dibuka . . . . .	52
4.4 Perancangan GUI sesudah <i>file</i> permainan dibuka . . . . .	53
4.5 Perancangan GUI sesudah permainan berdasarkan <i>file</i> permainan yang dibuka diselesaikan . . . . .	53
4.6 Menu <i>File</i> . . . . .	54
4.7 Menu <i>Solve</i> . . . . .	54
4.8 Diagram kelas untuk perangkat lunak Calcudoku . . . . .	56
4.9 Diagram kelas Grid . . . . .	61
4.10 Diagram kelas Cage . . . . .	62
4.11 Diagram kelas Cell . . . . .	63
4.12 Diagram kelas SolverBacktracking . . . . .	64
4.13 Diagram kelas SolverHybridGenetic . . . . .	66
4.14 Diagram kelas SolverRuleBased . . . . .	70

4.15	Diagram kelas SolverGenetic.	73
4.16	Diagram kelas Chromosome.	73
4.17	Diagram kelas ChromosomeComparator.	74
4.18	Diagram kelas Controller.	75
4.19	Diagram kelas Calcudoku.	79
4.20	Diagram kelas WindowListener.	79
4.21	Diagram kelas PuzzleFileFilter.	79
4.22	Diagram kelas GUI.	82
4.23	Diagram kelas CellKeyListener.	82
4.24	Diagram kelas PopupMenuItemListener.	83
4.25	Diagram kelas CellTextFieldListener.	84
4.26	Diagram kelas GeneticParameters.	85
5.1	Antarmuka perangkat lunak saat pertama kali dibuka	88
5.2	Kotak dialog untuk memilih <i>file</i> permainan yang akan dibuka	89
5.3	Antarmuka perangkat lunak sesudah membuka <i>file</i> permainan yang dipilih	89
5.4	Kotak dialog untuk mengatur nilai dari parameter-parameter algoritma genetik	90
5.5	Antarmuka perangkat lunak setelah permainan berdasarkan <i>file</i> permainan yang telah dibuka diselesaikan	90
5.6	Kotak pesan error " <i>Puzzle file not loaded</i> "	91
5.7	Kotak pemilihan <i>file</i> permainan	92
5.8	Kotak dialog " <i>Are you sure you want to load another puzzle file?</i> "	92
5.9	Pesan error " <i>Invalid puzzle file</i> "	92
5.10	Pesan error " <i>Invalid cages</i> "	93
5.11	Pesan error " <i>Error in loading puzzle file</i> "	93
5.12	Pesan informasi " <i>Congratulations, you have successfully solved the puzzle</i> "	94
5.13	Kotak dialog " <i>Are you sure you want to reset this puzzle?</i> "	94
5.14	Pesan informasi " <i>Row</i> (nomor baris) has duplicate numbers"	95
5.15	Pesan informasi " <i>Column</i> (nomor kolom) has duplicate numbers"	95
5.16	Pesan informasi " <i>Values of cells in the cage do not reach the target number</i> "	95
5.17	Pesan informasi " <i>There are cells with incorrect values in the grid</i> "	96
5.18	Pesan informasi " <i>There are empty cells in the grid</i> "	96
5.19	Pesan informasi " <i>The backtracking algorithm has successfully solved the puzzle</i> "	97
5.20	Pesan informasi " <i>The backtracking algorithm has failed to solve the puzzle</i> "	97
5.21	Pesan informasi " <i>Genetic algorithm parameters have not been set</i> "	98
5.22	Pesan informasi " <i>The hybrid genetic algorithm has successfully solved the puzzle</i> "	98
5.23	Pesan informasi " <i>The hybrid genetic algorithm has failed to solve the puzzle</i> "	99
5.24	<i>Form</i> untuk mengatur nilai untuk parameter-parameter algoritma genetik	99
5.25	Pesan error " <i>Invalid number format</i> "	99
5.26	Kotak dialog " <i>Are you sure you want to close this puzzle file?</i> "	100
5.27	Kotak dialog " <i>Are you sure you want to exit the application?</i> "	100
5.28	<i>File</i> masukan untuk pengujian keakuratan	101
5.29	GUI permainan berdasarkan <i>file</i> masukan yang dapat dilihat pada Gambar 5.28	101
5.30	Solusi untuk permainan berdasarkan <i>file</i> masukan yang dapat dilihat pada Gambar 5.28	103
A.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]	115
A.2	<i>State</i> 4	116
A.3	<i>State</i> 11	116
A.4	<i>State</i> 12	116
A.5	<i>State</i> 17	117
A.6	<i>State</i> 18	117

A.7 <i>State</i> 19 . . . . .	117
A.8 <i>State</i> 23 . . . . .	118
A.9 <i>State</i> 24 . . . . .	118
A.10 <i>State</i> 31 . . . . .	119
A.11 <i>State</i> 32 . . . . .	119
A.12 <i>State</i> 34 . . . . .	119
A.13 <i>State</i> 37 . . . . .	120
A.14 <i>State</i> 47 . . . . .	120
A.15 <i>State</i> 48 . . . . .	121
A.16 <i>State</i> 52 . . . . .	121
A.17 <i>State</i> 53 . . . . .	121
A.18 <i>State</i> 68 . . . . .	122
A.19 <i>State</i> 69 . . . . .	123
A.20 <i>State</i> 71 . . . . .	123
A.21 <i>State</i> 72 . . . . .	123
A.22 <i>State</i> 74 . . . . .	124
A.23 <i>State</i> 75 . . . . .	124
A.24 <i>State</i> 76 . . . . .	124
A.25 <i>State</i> 77 . . . . .	125
A.26 <i>State</i> 78 . . . . .	125
A.27 <i>State</i> 81 . . . . .	125
A.28 <i>State</i> 83 . . . . .	126
A.29 <i>State</i> 85 . . . . .	126
A.30 <i>State</i> 88 . . . . .	126
A.31 <i>State</i> 92 . . . . .	127
A.32 <i>State</i> 93 . . . . .	127

## DAFTAR TABEL

3.1	Tabel parameter untuk algoritma genetik yang akan digunakan untuk menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.12 . . . . .	29
3.2	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1 . . . . .	34
3.3	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1 . . . . .	39
3.4	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-3 . . . . .	44
3.5	Skenario me- <i>load file</i> . . . . .	45
3.6	Skenario memilih salah satu dari dua <i>solver</i> yang disediakan . . . . .	46
3.7	Skenario me- <i>reset</i> permainan . . . . .	47
3.8	Skenario meminta perangkat lunak untuk memeriksa permainan . . . . .	47
3.9	Skenario menutup <i>file</i> masukan . . . . .	48
3.10	Skenario menyelesaikan permainan dengan usahanya sendiri . . . . .	48
3.11	Skenario mengatur nilai dari parameter-parameter untuk algoritma genetik . . . . .	49
5.1	Lingkungan perangkat keras untuk pengujian perangkat lunak . . . . .	87
5.2	Lingkungan perangkat lunak untuk pengujian perangkat lunak . . . . .	88
5.3	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku . . . . .	103
5.4	Nilai untuk parameter-parameter algoritma genetik untuk setiap percobaan yang dilakukan . . . . .	104
5.5	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 1) . . . . .	104
5.6	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 2) . . . . .	105
5.7	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 3) . . . . .	105
5.8	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 4) . . . . .	105
5.9	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 5) . . . . .	106
5.10	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 6) . . . . .	106
5.11	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 7) . . . . .	106
5.12	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 8) . . . . .	107
5.13	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 9) . . . . .	107
5.14	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 10) . . . . .	107
5.15	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 11) . . . . .	108
5.16	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 12) . . . . .	108
5.17	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 13) . . . . .	108
5.18	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 14) . . . . .	109
5.19	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 15) . . . . .	109
5.20	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku (Skenario 16) . . . . .	109
B.1	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> $4 \times 4$ .	130
B.2	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> $5 \times 5$ .	131
B.3	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> $6 \times 6$ .	132
B.4	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> $7 \times 7$ .	133
B.5	Hasil pengujian algoritma <i>backtracking</i> untuk Calcudoku dengan ukuran <i>grid</i> $8 \times 8$ .	133
B.6	Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $4 \times 4$ (Skenario 1-4) . . . . .	135

B.7 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $4 \times 4$ (Skenario 5-8) . . . . .	136
B.8 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $4 \times 4$ (Skenario 9-12) . . . . .	137
B.9 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $4 \times 4$ (Skenario 13-16) . . . . .	138
B.10 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $5 \times 5$ (Skenario 1-4) . . . . .	139
B.11 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $5 \times 5$ (Skenario 5-8) . . . . .	140
B.12 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $5 \times 5$ (Skenario 9-12) . . . . .	141
B.13 Hasil pengujian algoritma <i>hybrid genetic</i> untuk Calcudoku dengan ukuran <i>grid</i> $5 \times 5$ (Skenario 13-16) . . . . .	142



# 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  $9 \times 9$ , dan ada 9 area yang berukuran  $3 \times 3$ . 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,

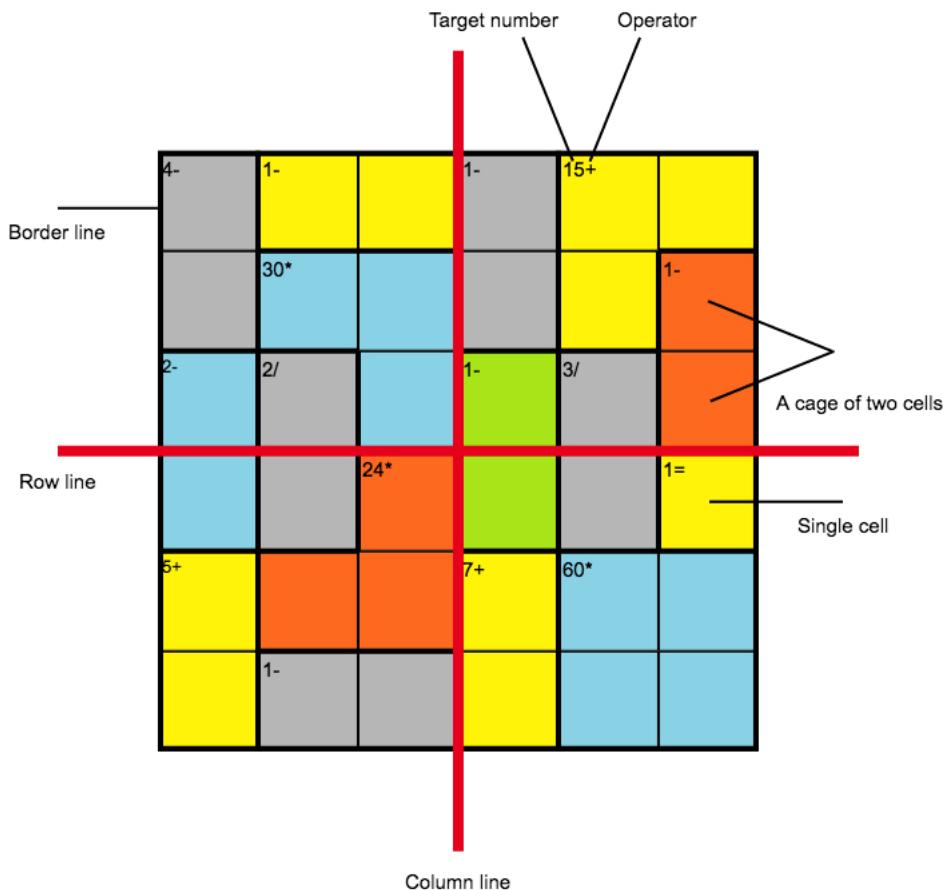
2	5	1	9
8		2	3
3		6	7
	1		6
5	4		1 9
	2		7
9		3	8
2		8	4
1	9	7	6

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]

setiap sel dalam setiap *cage* dalam teka-teki ini kosong, belum terisi oleh angka-angka. *Border line* adalah garis pembatas 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 (*artificial 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 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 kecepatan dan kesuksesan 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  $4 \times 4$  sampai dengan  $8 \times 8$ . Pada awalnya, ukuran *grid* direncanakan akan dibatasi dari  $3 \times 3$  sampai dengan  $9 \times 9$ , tetapi karena kurangnya contoh soal teka-teki Calcudoku dengan ukuran  $3 \times 3$ , dan ada masalah saat pengujian (keluar pesan error "*Memory full*" saat menguji *solver* dengan algoritma *backtracking* pada *grid* yang berukuran  $9 \times 9$ , maka ukuran *grid* dibatasi dari  $4 \times 4$  sampai dengan  $8 \times 8$ .
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*.

## 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.
  - (e) Membandingkan performansi algoritma *backtracking* dengan algoritma *hybrid genetic* dalam menyelesaikan Calcudoku.
3. Melakukan pengujian dan eksperimen terhadap perangkat lunak Calcudoku yang telah dibuat.
4. 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 simpulan 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, mengantikan 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.

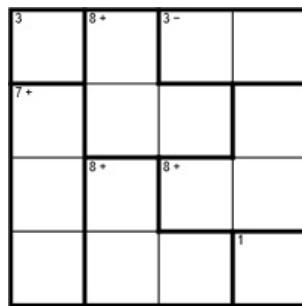
Seperti dalam Sudoku, dalam teka-teki ini, pemain diberikan sebuah *grid* dengan ukuran  $n \times n$ , dengan  $n$  biasanya  $3 \leq n \leq 9$ . *Grid* ini harus diisi dengan angka 1 sampai dengan  $n$  sehingga dalam setiap baris setiap angka hanya muncul sekali, dalam setiap kolom setiap angka hanya 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.  $+$ , sebuah operator  $n$ -ary yang menandakan penjumlahan.
2.  $-$ , sebuah operator biner yang menandakan pengurangan.
3.  $\times$ , sebuah operator  $n$ -ary yang menandakan perkalian.
4.  $\div$  sebuah operator biner yang menandakan pembagian.
5.  $=$ , (simbol ini biasanya dihilangkan), sebuah operator uner yang menandakan persamaan.

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



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

3	8+	3-	
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:

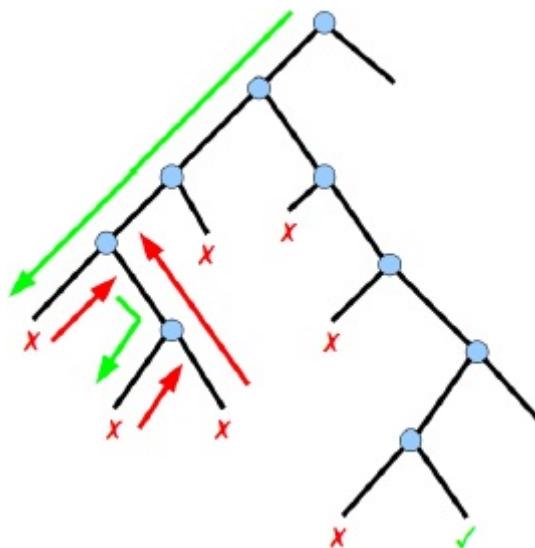
$$B(x_1, x_2, \dots, x_k)$$

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  $n$ .

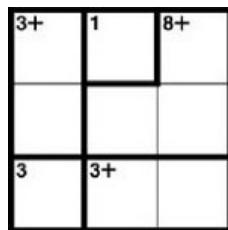
Fungsi pembangkit membangkitkan sebuah integer secara berurutan dari 1 sampai  $n$  sebagai  $x_k$ . Fungsi pembatas menggabungkan tiga fungsi pemeriksa pembatas (*constraint checking*), yaitu fungsi pemeriksa kolom (*column checking*), fungsi pemeriksa baris (*row checking*), dan fungsi pemeriksa *grid* (*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 hasil penjumlahan semua nilai yang ada pada *grid* ditambah dengan  $x_k$  kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika jumlah semua nilai yang ada pada *grid* ditambah  $x_k$  lebih dari nilai tujuan.
2. Operator pengurangan (-), fungsi menghasilkan nilai *true* jika kedua sel dalam *grid* kosong, atau jika ada satu sel yang kosong dan hasil dari  $x_k$  dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan  $x_k$  menghasilkan nilai



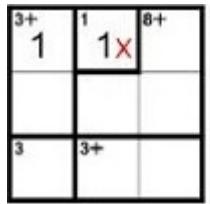
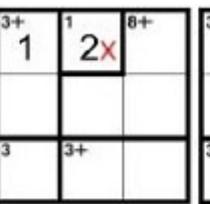
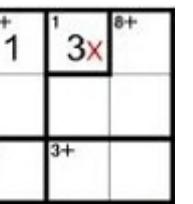
Gambar 2.4: Contoh permainan teka-teki Calcudoku dengan ukuran  $grid$   $3 \times 3$  [1]

tujuan, dan menghasilkan nilai *false* jika ada satu sel kosong dan hasil dari  $x_k$  dikurangi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dikurangi dengan  $x_k$  tidak menghasilkan nilai tujuan.

3. Operator perkalian ( $\times$ ), fungsi menghasilkan nilai *true* jika hasil perkalian dari semua nilai yang ada pada  $grid$  dikali dengan  $x_k$  kurang dari atau sama dengan nilai tujuan, dan menghasilkan nilai *false* jika hasil perkalian dari semua nilai yang ada pada  $grid$  dikali dengan  $x_k$  lebih dari nilai tujuan.
4. Operator pembagian ( $\div$ ), fungsi menghasilkan nilai *true* jika kedua sel dalam  $grid$  kosong, atau jika ada satu sel yang kosong dan hasil dari  $x_k$  dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan  $x_k$  menghasilkan nilai tujuan, dan menghasilkan nilai *false* jika ada satu sel yang kosong dan hasil dari  $x_k$  dibagi dengan nilai dari sel yang lainnya atau hasil dari nilai dari sel yang lainnya dibagi dengan  $x_k$  tidak menghasilkan nilai tujuan.
5. Operator  $=$ , fungsi akan menghasilkan nilai *true* jika  $x_k$  sama dengan nilai tujuan, dan 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 diisikan 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	$3+$	

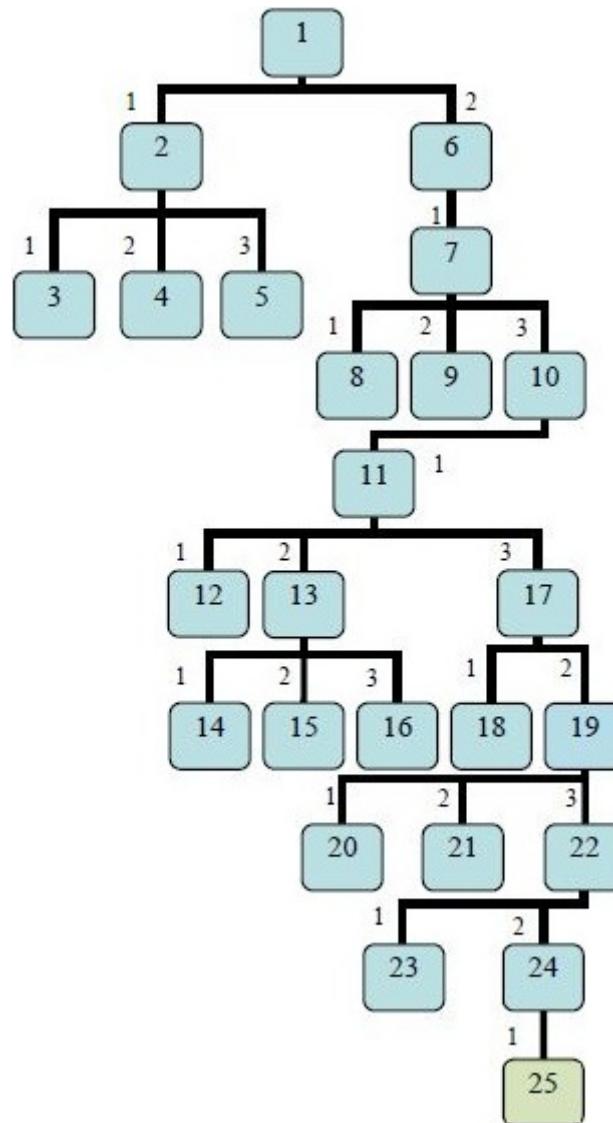
Gambar 2.6: Ilustrasi *state* 19 pada sebuah *grid* teka-teki Calcudoku [1]

$3+$	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 rule* 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.

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]

### 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 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 keberlanjutan 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 yang bisa diterima ditemukan.

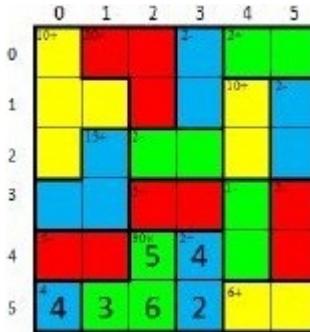
Cara kerja algoritma genetik adalah sebagai berikut [2]:

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  $P(C_{b,k}) =$



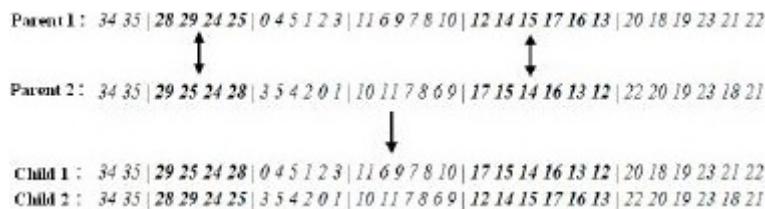
Gambar 2.14: Contoh permainan teka-teki Calcudoku dengan ukuran grid 6 x 6 [2]

1, 2, ...,  $n$ . 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 sama dan dalam kolom yang sama harus diperbaharui, 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]

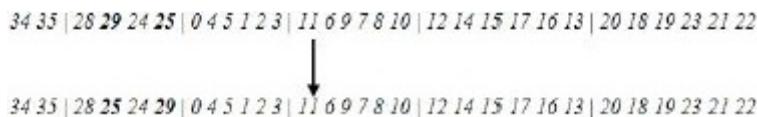
1.  $|P(C_{b,k})| = 1 \wedge x \in P(C_{b,k}) \rightarrow V(C_{b,k}) = x$ , artinya jika sebuah *cage* berukuran 1 sel, dan  $x$  adalah nilai tujuan dari *cage* tersebut, maka nilai dari sel tersebut adalah  $x$ .
2.  $(V(C_{b,k}) = x) \wedge (\forall a \in \{1, 2, \dots, n\}) \rightarrow P(C_{a,k}) = P(C_{a,k}) - \{x\}$ , 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 ( $w_j \neq 0$ ). Untuk kromosom dengan jumlah gen  $k$ , fungsi kelayakan, yaitu hasil



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



Gambar 2.16: Contoh proses mutasi [2]

penjumlahan dari hasil fungsi objektif untuk setiap gen dibagi dengan jumlah gen, dinyatakan dalam persamaan di bawah ini [2]:

$$x_j = \{ 0, w_j = 01, w_j \neq 0 \}$$

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

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

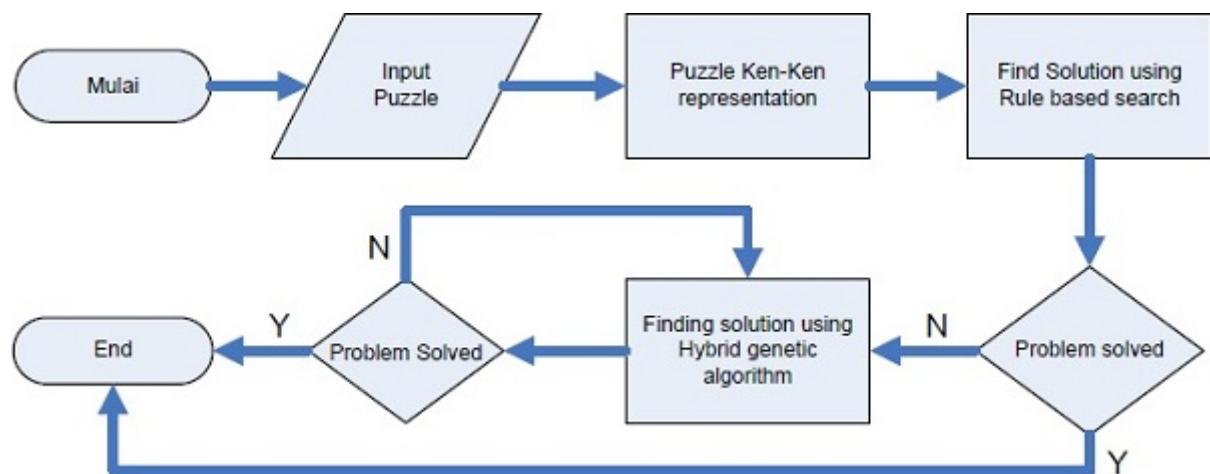
Dalam proses reproduksi kawin silang, dua kromosom, yaitu kromosom orang tua, disilangkan untuk membuat dua kromosom yang baru, yaitu kromosom anak, dengan metodologi kawin silang *N-segments*. Gambar 2.15 menggambarkan contoh proses kawin silang antara dua kromosom.

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*, maka algoritma selesai.
5. Jika gagal dengan menggunakan algoritma *rule based*, 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, maka algoritma selesai.
7. Jika gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, 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.



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



## 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*

Untuk mengilustrasikan cara kerja algoritma *backtracking*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar A.1 sebagai contoh.

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.

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]

<sup>3</sup> 3	<sup>8+</sup>	<sup>3-</sup>	
<sup>7+</sup>			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar 3.2: *State 4*

<sup>3</sup> 3	<sup>8+</sup> 1	<sup>3-</sup> 2	4
<sup>7+</sup>			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar 3.3: *State 11*

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 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.

<sup>3</sup> 3	<sup>8+</sup> 1	<sup>3-</sup> 3	
<sup>7+</sup>			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar 3.4: *State 12*

<sup>3</sup> 3	<sup>8+</sup> 1	<sup>3-</sup> 4	4
7+			
	<sup>8+</sup>	<sup>8+</sup>	
			<sup>1</sup>

Gambar 3.5: *State 17*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup>	
7+			
	<sup>8+</sup>	<sup>8+</sup>	
			<sup>1</sup>

Gambar 3.6: *State 18*

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 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.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	
7+			
	<sup>8+</sup>	<sup>8+</sup>	
			<sup>1</sup>

Gambar 3.7: *State 19*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup>			
	<sup>8+</sup>	<sup>8+</sup>	
			1

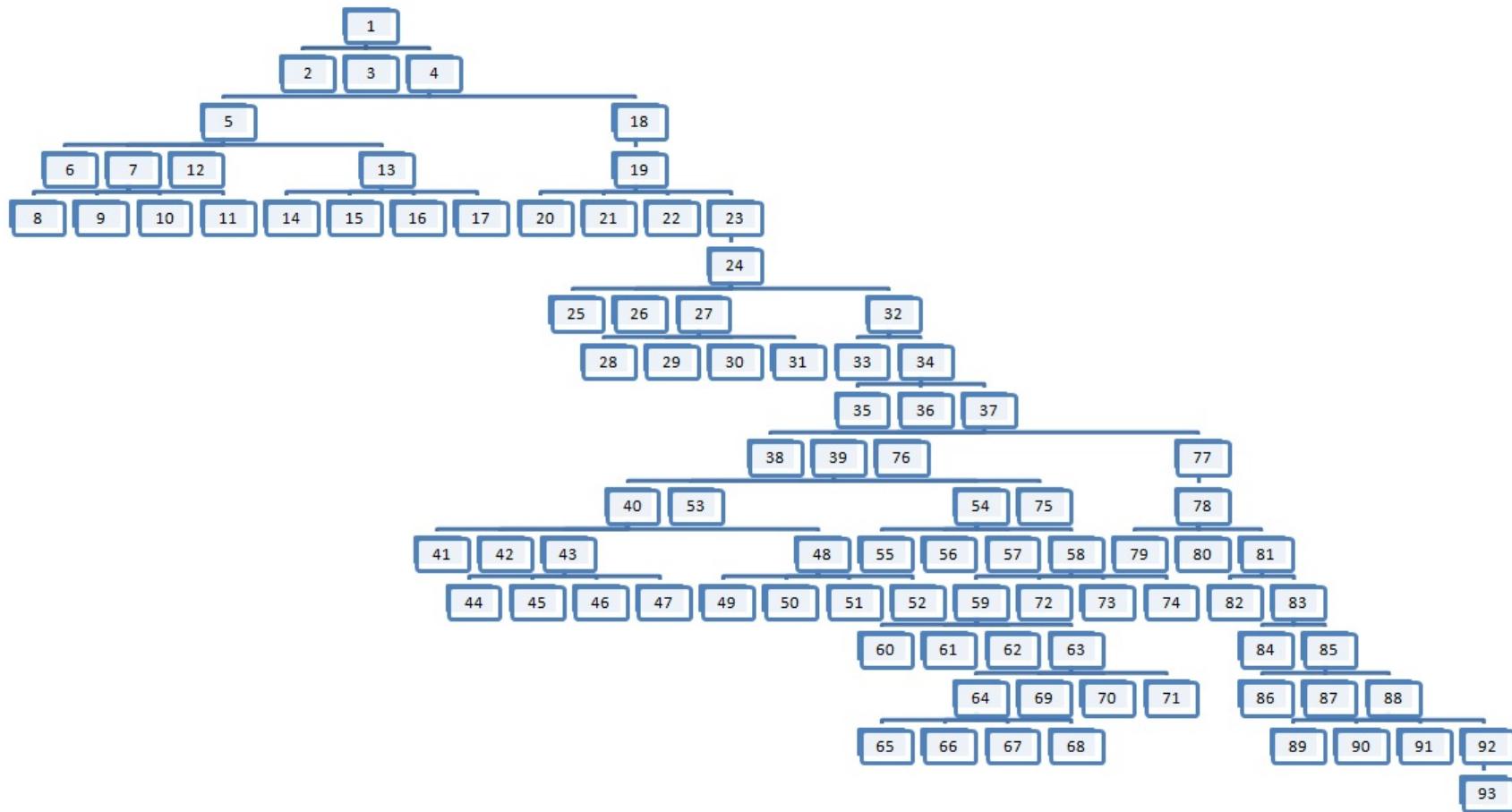
Gambar 3.8: *State 23*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup> 3	2
2	3	4	<sup>1</sup> 1

Gambar 3.9: *State 93*

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 A.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 A.32.

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.



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

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*.

### 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.

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.

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*

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%

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

<sup>4-</sup> 1	<sup>1-</sup> 3	5	<sup>1-</sup> 6	<sup>15+</sup> 4	2
3	<sup>30*</sup> 4	2	5	1	<sup>1-</sup> 6
<sup>2-</sup> 2	<sup>2/-</sup> 1	6	<sup>1-</sup> 4	<sup>3/-</sup> 3	5
6	5	<sup>24*</sup> 4	3	2	<sup>1-</sup> 1
<sup>5+</sup> 4	6	1	<sup>7+</sup> 2	<sup>60*</sup> 5	3
5	<sup>1-</sup> 2	3	1	6	4

Gambar 3.13: Kromosom 1 dalam Generasi ke-1

<sup>4-</sup> 3	<sup>1-</sup> 2	1	<sup>1-</sup> 6	<sup>15+</sup> 5	4
1	<sup>30*</sup> 6	2	5	4	<sup>1-</sup> 3
<sup>2-</sup> 5	<sup>2/-</sup> 3	6	<sup>1-</sup> 4	<sup>3/-</sup> 1	2
6	5	<sup>24*</sup> 4	3	2	<sup>1-</sup> 1
<sup>5+</sup> 4	1	3	<sup>7+</sup> 2	<sup>60*</sup> 6	5
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.14: Kromosom 2 dalam Generasi ke-1

<sup>4-</sup> 4	<sup>1-</sup> 3	6	<sup>1-</sup> 2	<sup>15+</sup> 1	5
6	<sup>30*</sup> 5	1	3	2	<sup>1-</sup> 4
<sup>2-</sup> 5	<sup>2/-</sup> 1	4	<sup>1-</sup> 6	<sup>3/-</sup> 3	2
3	6	<sup>24*</sup> 2	5	4	<sup>1-</sup> 1
<sup>5+</sup> 1	2	3	<sup>7+</sup> 4	<sup>60*</sup> 5	6
2	<sup>1-</sup> 4	5	1	6	3

Gambar 3.15: Kromosom 3 dalam Generasi ke-1

<sup>4-</sup> 5	<sup>1-</sup> 1	4	<sup>1-</sup> 6	<sup>15+</sup> 2	3
1	<sup>30*</sup> 2	3	4	5	<sup>1-</sup> 6
<sup>2-</sup> 6	<sup>2/</sup> 3	5	<sup>1-</sup> 2	<sup>3/</sup> 1	4
2	4	<sup>24*</sup> 6	5	3	<sup>1</sup> 1
<sup>5+</sup> 4	5	1	<sup>7+</sup> 6	<sup>60*</sup> 3	2
3	<sup>1-</sup> 6	2	1	4	5

Gambar 3.16: Kromosom 4 dalam Generasi ke-1

<sup>4-</sup> 5	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 1	4
1	<sup>30*</sup> 6	3	5	4	<sup>1-</sup> 2
<sup>2-</sup> 3	<sup>2/</sup> 2	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.17: Kromosom 5 dalam Generasi ke-1

<sup>4-</sup> 6	<sup>1-</sup> 3	5	<sup>1-</sup> 2	<sup>15+</sup> 1	4
5	<sup>30*</sup> 1	4	6	2	<sup>1-</sup> 3
<sup>2-</sup> 2	<sup>2/</sup> 4	6	<sup>1-</sup> 1	<sup>3/</sup> 3	5
3	6	<sup>24*</sup> 2	5	4	<sup>1</sup> 1
<sup>5+</sup> 1	2	3	<sup>7+</sup> 4	<sup>60*</sup> 5	6
4	<sup>1-</sup> 5	1	3	6	2

Gambar 3.18: Kromosom 6 dalam Generasi ke-1

<sup>4-</sup> 1	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 4	5
3	<sup>30*</sup> 4	1	5	6	<sup>1-</sup> 2
<sup>2-</sup> 5	<sup>2/</sup> 2	6	<sup>1-</sup> 4	<sup>3/</sup> 1	3
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	3	1	<sup>7+</sup> 2	<sup>60*</sup> 5	6
2	<sup>1-</sup> 6	5	1	3	4

Gambar 3.19: Kromosom 7 dalam Generasi ke-1

<sup>4-</sup> 3	<sup>1-</sup> 1	5	<sup>1-</sup> 6	<sup>15+</sup> 4	2
2	<sup>30*</sup> 6	3	5	1	<sup>1-</sup> 4
<sup>2-</sup> 1	<sup>2/</sup> 2	6	<sup>1-</sup> 4	<sup>3/</sup> 3	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 5	4	1	<sup>7+</sup> 2	<sup>60*</sup> 6	3
4	<sup>1-</sup> 3	2	1	5	6

Gambar 3.20: Kromosom 8 dalam Generasi ke-1

<sup>4-</sup> 4	<sup>1-</sup> 6	5	<sup>1-</sup> 3	<sup>15+</sup> 1	2
3	<sup>30*</sup> 5	1	6	2	<sup>1-</sup> 4
<sup>2-</sup> 6	<sup>2/</sup> 1	4	<sup>1-</sup> 2	<sup>3/</sup> 3	5
2	3	<sup>24*</sup> 6	5	4	<sup>1</sup> 1
<sup>5+</sup> 1	2	3	<sup>7+</sup> 4	<sup>60+</sup> 5	6
5	<sup>1-</sup> 4	2	1	6	3

Gambar 3.21: Kromosom 9 dalam Generasi ke-1

<sup>4-</sup> 5	<sup>1-</sup> 1	3	<sup>1-</sup> 6	<sup>15+</sup> 4	2
3	<sup>30*</sup> 6	2	5	1	<sup>1-</sup> 4
<sup>2-</sup> 2	<sup>2/</sup> 3	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1-</sup> 1
<sup>5+</sup> 1	4	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
4	<sup>1-</sup> 2	5	1	3	6

Gambar 3.22: Kromosom 10 dalam Generasi ke-1

<sup>4-</sup> 5	<sup>1-</sup> 1	6	<sup>1-</sup> 2	<sup>15+</sup> 4	3
1	<sup>30*</sup> 2	3	4	5	<sup>1-</sup> 6
<sup>2-</sup> 4	<sup>2/</sup> 3	5	<sup>1-</sup> 6	<sup>3/</sup> 1	2
6	4	<sup>24*</sup> 2	5	3	<sup>1-</sup> 1
<sup>5+</sup> 2	5	1	<sup>7+</sup> 3	<sup>60*</sup> 6	4
3	<sup>1-</sup> 6	4	1	2	5

Gambar 3.23: Kromosom 11 dalam Generasi ke-1

<sup>4-</sup> 1	<sup>1-</sup> 5	6	<sup>1-</sup> 4	<sup>15+</sup> 3	2
3	<sup>30*</sup> 6	1	2	4	<sup>1-</sup> 5
<sup>2-</sup> 4	<sup>2/</sup> 1	5	<sup>1-</sup> 3	<sup>3/</sup> 2	6
2	3	<sup>24*</sup> 4	5	6	<sup>1-</sup> 1
<sup>5+</sup> 5	4	2	<sup>7+</sup> 6	<sup>60*</sup> 1	3
6	<sup>1-</sup> 2	3	1	5	4

Gambar 3.24: Kromosom 12 dalam Generasi ke-1

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 Kromosom 12, Kromosom 5, Kromosom 7, Kromosom 11, dan Kromosom 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.

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

1	<sup>1-</sup> 5	6	4	<sup>15+</sup> 3	2
3	<sup>30*</sup> 6	1	2	4	<sup>1-</sup> 5
<sup>2-</sup> 4	<sup>2/</sup> 1	5	<sup>1-</sup> 3	<sup>3/</sup> 2	6
2	3	<sup>24*</sup> 4	5	6	<sup>1</sup> 1
<sup>5+</sup> 5	4	2	<sup>7+</sup> 6	<sup>60*</sup> 1	3
6	<sup>1-</sup> 2	3	1	5	4

Gambar 3.25: Kromosom 1 dalam Generasi ke-2

<sup>4-</sup> 5	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 1	4
1	<sup>30*</sup> 6	3	5	4	<sup>1-</sup> 2
<sup>2-</sup> 3	<sup>2/</sup> 2	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.26: Kromosom 2 dalam Generasi ke-2

<sup>4-</sup> 1	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 4	5
3	<sup>30*</sup> 4	1	5	6	<sup>1-</sup> 2
<sup>2-</sup> 5	<sup>2/</sup> 2	6	<sup>1-</sup> 4	<sup>3/</sup> 1	3
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	3	<sup>7+</sup> 2	<sup>60*</sup> 5	6
2	<sup>1-</sup> 6	5	1	3	4

Gambar 3.27: Kromosom 3 dalam Generasi ke-2

<sup>4-</sup> 5	<sup>1-</sup> 1	6	<sup>1-</sup> 2	<sup>15+</sup> 4	3
1	<sup>30*</sup> 2	3	4	5	<sup>1-</sup> 6
<sup>2-</sup> 4	<sup>2/</sup> 3	5	<sup>1-</sup> 6	<sup>3/</sup> 1	2
6	4	<sup>24*</sup> 2	5	3	<sup>1</sup> 1
<sup>5+</sup> 2	5	1	<sup>7+</sup> 3	<sup>60*</sup> 6	4
3	<sup>1-</sup> 6	4	1	2	5

Gambar 3.28: Kromosom 4 dalam Generasi ke-2

<sup>4-</sup> 1	<sup>1-</sup> 3	5	<sup>1-</sup> 6	<sup>15+</sup> 4	2
3	<sup>30*</sup> 4	2	5	1	<sup>1-</sup> 6
<sup>2-</sup> 2	<sup>2/</sup> 1	6	<sup>1-</sup> 4	<sup>3/</sup> 3	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	6	1	<sup>7+</sup> 2	<sup>60*</sup> 5	3
5	<sup>1-</sup> 2	3	1	6	4

Gambar 3.29: Kromosom 5 dalam Generasi ke-2

<sup>4-</sup> 3	<sup>1-</sup> 2	1	<sup>1-</sup> 6	<sup>15+</sup> 5	4
1	<sup>30*</sup> 6	2	5	4	<sup>1-</sup> 3
<sup>2-</sup> 2	<sup>2/</sup> 3	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 1	4	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.30: Kromosom 6 dalam Generasi ke-2

<sup>4-</sup> 5	<sup>1-</sup> 1	3	<sup>1-</sup> 6	<sup>15+</sup> 4	2
3	<sup>30*</sup> 6	2	5	1	<sup>1-</sup> 4
<sup>2-</sup> 5	<sup>2/</sup> 3	6	<sup>1-</sup> 4	<sup>3/</sup> 1	2
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	3	<sup>7+</sup> 2	<sup>60*</sup> 6	5
4	<sup>1-</sup> 2	5	1	3	6

Gambar 3.31: Kromosom 7 dalam Generasi ke-2

<sup>4-</sup> 1	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 4	5
3	<sup>30*</sup> 4	1	5	6	<sup>1-</sup> 2
<sup>2-</sup> 2	<sup>2/</sup> 3	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 1	4	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
2	<sup>1-</sup> 6	1	5	3	4

Gambar 3.32: Kromosom 8 dalam Generasi ke-2

<sup>4-</sup> 5	<sup>1-</sup> 1	3	<sup>1-</sup> 6	<sup>15+</sup> 4	2
3	<sup>30*</sup> 6	2	5	1	<sup>1-</sup> 4
<sup>2-</sup> 5	<sup>2/</sup> 2	6	<sup>1-</sup> 4	<sup>3/</sup> 1	3
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	3	1	<sup>7+</sup> 2	<sup>60*</sup> 5	6
4	<sup>1-</sup> 2	5	1	3	6

Gambar 3.33: Kromosom 9 dalam Generasi ke-2

<sup>4-</sup> 3	<sup>1-</sup> 2	1	<sup>1-</sup> 6	<sup>15+</sup> 5	4
1	<sup>30*</sup> 6	3	5	4	<sup>1-</sup> 2
<sup>2-</sup> 3	<sup>2/-</sup> 2	1	<sup>1-</sup> 4	<sup>3/-</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	3	<sup>7+</sup> 2	<sup>60*</sup> 6	5
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.34: Kromosom 10 dalam Generasi ke-2

<sup>4-</sup> 5	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 1	4
1	<sup>30*</sup> 6	2	5	4	<sup>1-</sup> 3
<sup>2-</sup> 5	<sup>2/-</sup> 3	6	<sup>1-</sup> 4	<sup>3/-</sup> 1	2
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.35: Kromosom 11 dalam Generasi ke-2

<sup>4-</sup> 1	<sup>1-</sup> 5	6	<sup>1-</sup> 4	<sup>15+</sup> 3	2
3	<sup>30*</sup> 6	1	2	4	<sup>1-</sup> 5
<sup>2-</sup> 4	<sup>2/-</sup> 1	5	<sup>1-</sup> 3	<sup>3/-</sup> 2	6
2	3	<sup>24*</sup> 4	5	6	<sup>1</sup> 1
<sup>5+</sup> 5	4	2	<sup>7+</sup> 6	<sup>60*</sup> 1	3
1	<sup>1-</sup> 2	3	6	5	4

Gambar 3.36: Kromosom 12 dalam Generasi ke-2

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 Kromosom 1, Kromosom 12, Kromosom 2, Kromosom 3, dan Kromosom 4.

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.

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

<sup>4-</sup> 1	<sup>1-</sup> 5	6	<sup>1-</sup> 4	<sup>15+</sup> 3	2
3	<sup>30*</sup> 6	1	2	4	<sup>1-</sup> 5
<sup>2-</sup> 4	<sup>2/-</sup> 1	5	<sup>1-</sup> 3	<sup>3/-</sup> 2	6
2	3	<sup>24*</sup> 4	5	6	<sup>1</sup> 1
<sup>5+</sup> 5	4	2	<sup>7+</sup> 6	<sup>60*</sup> 1	3
6	<sup>1-</sup> 2	3	1	5	4

Gambar 3.37: Kromosom 1 dalam Generasi ke-3

<sup>4-</sup> 1	<sup>1-</sup> 5	6	4	<sup>15+</sup> 3	2
3	<sup>30*</sup> 6	1	2	4	<sup>1-</sup> 5
<sup>2-</sup> 4	<sup>2/-</sup> 1	5	<sup>1-</sup> 3	<sup>3/-</sup> 2	6
2	3	<sup>24*</sup> 4	5	6	<sup>1</sup> 1
<sup>5+</sup> 5	4	2	<sup>7+</sup> 6	<sup>60*</sup> 1	3
1	<sup>1-</sup> 2	3	6	5	4

Gambar 3.38: Kromosom 2 dalam Generasi ke-3

<sup>4-</sup> 5	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 1	4
1	<sup>30*</sup> 6	3	5	4	<sup>1-</sup> 2
<sup>2-</sup> 3	<sup>2/-</sup> 2	1	<sup>1-</sup> 4	<sup>3/-</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.39: Kromosom 3 dalam Generasi ke-3

<sup>4-</sup> 1	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 4	5
3	<sup>30*</sup> 4	1	5	6	<sup>1-</sup> 2
<sup>2-</sup> 5	<sup>2/</sup> 2	6	<sup>1-</sup> 4	<sup>3/</sup> 1	3
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	3	<sup>7+</sup> 2	<sup>60*</sup> 5	6
2	<sup>1-</sup> 6	5	1	3	4

Gambar 3.40: Kromosom 4 dalam Generasi ke-3

<sup>4-</sup> 5	<sup>1-</sup> 1	6	<sup>1-</sup> 2	<sup>15+</sup> 4	3
1	<sup>30*</sup> 2	3	4	5	<sup>1-</sup> 6
<sup>2-</sup> 4	<sup>2/</sup> 3	5	<sup>1-</sup> 6	<sup>3/</sup> 1	2
6	4	<sup>24*</sup> 2	5	3	<sup>1</sup> 1
<sup>5+</sup> 2	5	1	<sup>7+</sup> 3	<sup>60*</sup> 6	4
3	<sup>1-</sup> 6	4	1	2	5

Gambar 3.41: Kromosom 5 dalam Generasi ke-3

<sup>4-</sup> 5	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 1	4
3	<sup>30*</sup> 6	1	2	4	<sup>1-</sup> 5
<sup>2-</sup> 4	<sup>2/</sup> 1	5	<sup>1-</sup> 3	<sup>3/</sup> 2	6
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
1	<sup>1-</sup> 2	3	6	5	4

Gambar 3.42: Kromosom 6 dalam Generasi ke-3

<sup>4-</sup> 1	<sup>1-</sup> 5	6	<sup>1-</sup> 4	<sup>15+</sup> 3	2
1	<sup>30*</sup> 6	3	5	4	<sup>1-</sup> 2
<sup>2-</sup> 3	<sup>2/</sup> 2	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
2	3	<sup>24*</sup> 4	5	6	<sup>1</sup> 1
<sup>5+</sup> 5	4	2	<sup>7+</sup> 6	<sup>60*</sup> 3	1
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.43: Kromosom 7 dalam Generasi ke-3

<sup>4-</sup> 1	<sup>1-</sup> 3	5	<sup>1-</sup> 6	<sup>15+</sup> 4	2
3	<sup>30*</sup> 4	2	5	1	<sup>1-</sup> 6
<sup>2-</sup> 2	<sup>2/</sup> 1	6	<sup>1-</sup> 4	<sup>3/</sup> 3	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	3	1	<sup>7+</sup> 2	<sup>60*</sup> 5	6
4	<sup>1-</sup> 2	5	1	3	6

Gambar 3.44: Kromosom 8 dalam Generasi ke-3

<sup>4-</sup> 5	<sup>1-</sup> 1	3	<sup>1-</sup> 6	<sup>15+</sup> 4	2
3	<sup>30*</sup> 6	2	5	1	<sup>1-</sup> 4
<sup>2-</sup> 5	<sup>2/</sup> 2	6	<sup>1-</sup> 4	<sup>3/</sup> 1	3
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	6	1	<sup>7+</sup> 2	<sup>60*</sup> 5	3
5	<sup>1-</sup> 2	3	1	6	4

Gambar 3.45: Kromosom 9 dalam Generasi ke-3

<sup>4-</sup> 1	<sup>1-</sup> 3	5	<sup>1-</sup> 6	<sup>15+</sup> 4	2
1	<sup>30*</sup> 6	2	5	4	<sup>1-</sup> 3
<sup>2-</sup> 2	<sup>2/</sup> 1	6	<sup>1-</sup> 4	<sup>3/</sup> 3	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 1	4	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
2	<sup>1-</sup> 4	5	1	3	6

Gambar 3.46: Kromosom 10 dalam Generasi ke-3

<sup>4-</sup> 3	<sup>1-</sup> 2	1	<sup>1-</sup> 6	<sup>15+</sup> 5	4
3	<sup>30*</sup> 4	2	5	1	<sup>1-</sup> 6
<sup>2-</sup> 2	<sup>2/</sup> 3	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	6	1	<sup>7+</sup> 2	<sup>60*</sup> 5	3
5	<sup>1-</sup> 2	3	1	6	4

Gambar 3.47: Kromosom 11 dalam Generasi ke-3

<sup>4-</sup> 5	<sup>1-</sup> 3	2	<sup>1-</sup> 6	<sup>15+</sup> 1	4
1	<sup>30*</sup> 6	3	5	4	<sup>1-</sup> 2
<sup>2-</sup> 3	<sup>2/</sup> 2	1	<sup>1-</sup> 4	<sup>3/</sup> 6	5
6	5	<sup>24*</sup> 4	3	2	<sup>1</sup> 1
<sup>5+</sup> 4	1	6	<sup>7+</sup> 2	<sup>60*</sup> 5	3
1	<sup>1-</sup> 4	5	2	3	6

Gambar 3.48: Kromosom 12 dalam Generasi ke-3

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 Kromosom 1, Kromosom 2, Kromosom 3, Kromosom 4, dan Kromosom 12.

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

### 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*.

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

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.

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 cage*, dan skenario.

### 3.3.1 Diagram *Use Case* dan Skenario

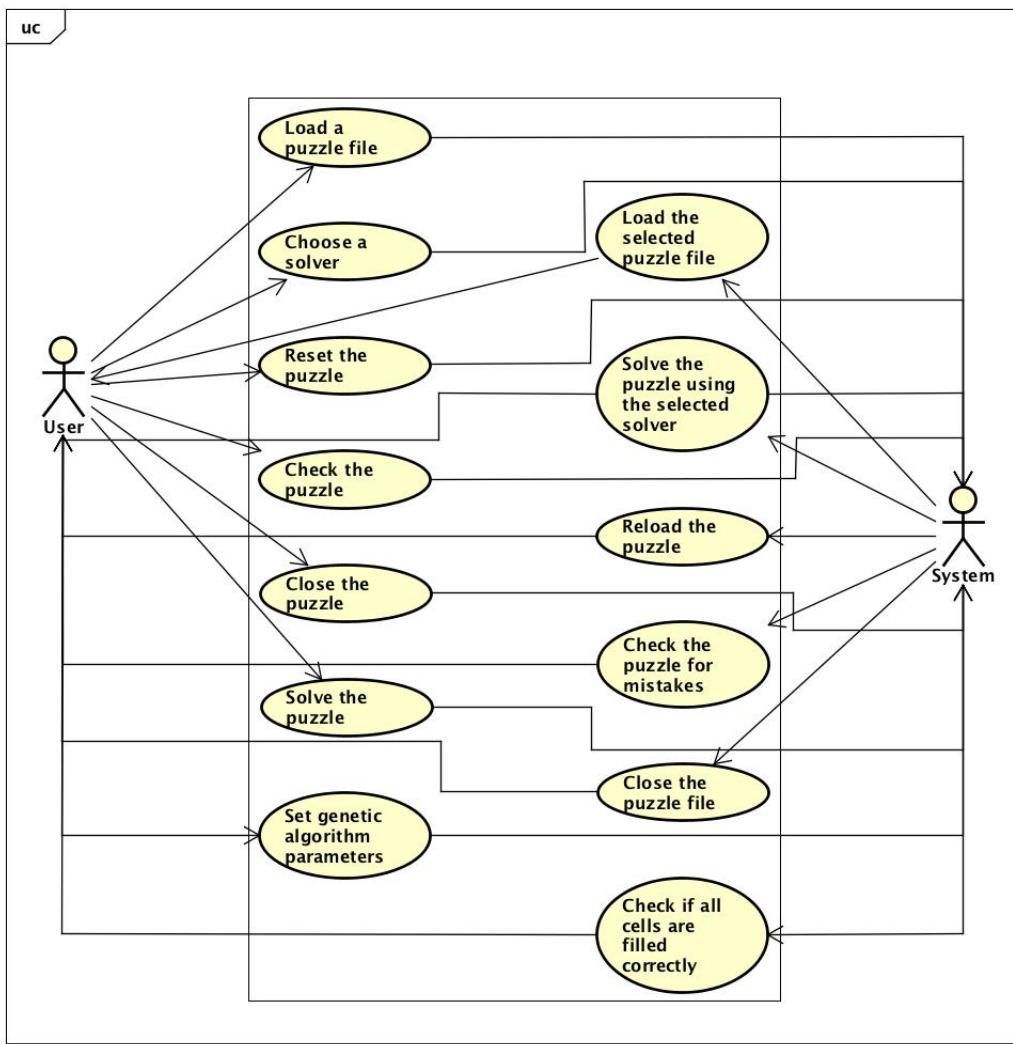
Diagram *use case* adalah diagram yang menggambarkan interaksi antara sistem (perangkat lunak) dengan pengguna. Berdasarkan analisis perangkat lunak yang telah dilakukan, maka pengguna dapat:

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*.
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* yang dapat dilihat pada Gambar 3.49, 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.



powered by Astah

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

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

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

Tabel 3.7: Skenario me-reset permainan

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

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> .

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.

### 3.3.2 Diagram Kelas

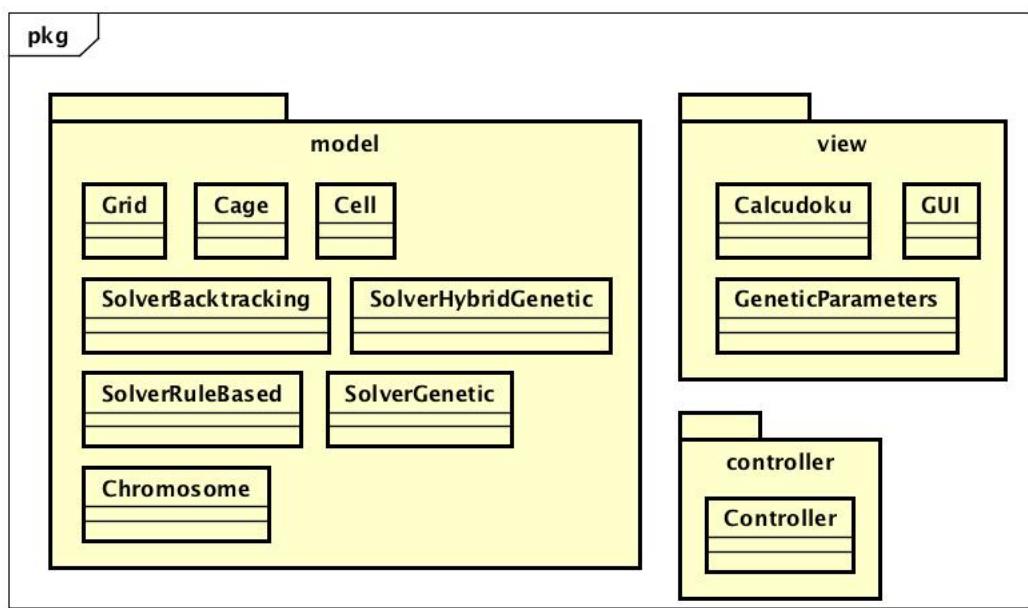
Berdasarkan diagram *use case* yang telah dibuat, maka diagram kelas dapat dibuat. Diagram kelas untuk perangkat lunak permainan teka-teki Calcudoku dapat dilihat pada Gambar 3.50.

Berdasarkan diagram kelas yang dapat dilihat pada Gambar ??, 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.

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".



powered by Astah

Gambar 3.50: Diagram kelas untuk perangkat lunak permainan teka-teki 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).
  - (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.

## 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, 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.

### 4.2 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 program, seperti dapat dilihat pada Gambar 4.2.

### 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 4.3 menunjukkan perancangan GUI sebelum *file* permainan dibuka. Gambar 4.4 menunjukkan perancangan GUI sesudah *file* permainan dibuka. Gambar 4.5 menunjukkan perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

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

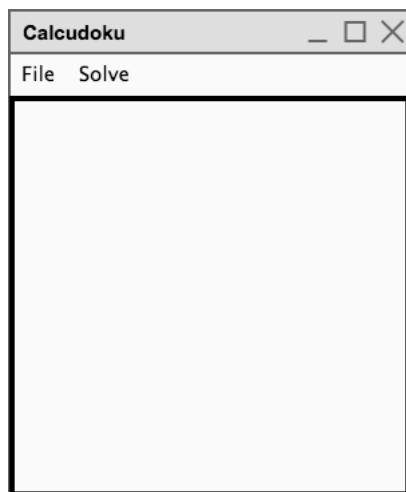
1. *File*, yaitu menu yang berisi *item-item* menu yang terkait dengan *file* permainan.

```
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+
```

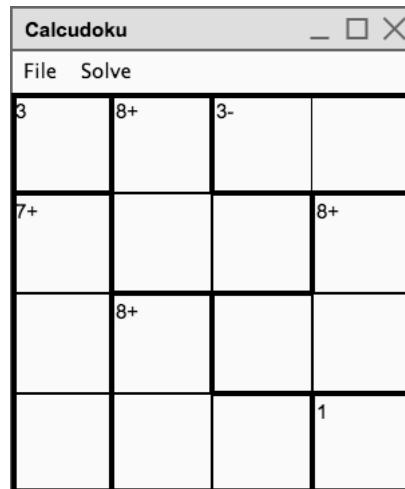
Gambar 4.1: Contoh *file* masukan.

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

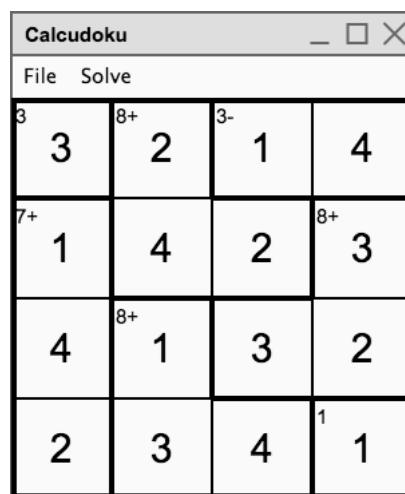
Gambar 4.2: Contoh keluaran.



Gambar 4.3: Perancangan GUI sebelum membuka *file* permainan



Gambar 4.4: Perancangan GUI sesudah membuka *file* permainan



Gambar 4.5: Perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

Gambar 4.6: Menu *File*Gambar 4.7: Menu *Solve*

2. *Solve*, yaitu menu yang berisi *item-item* menu yang terkait dengan *solver*.

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 4.6 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 4.7 menunjukkan isi dari menu *Solve*.

## 4.4 Diagram Kelas

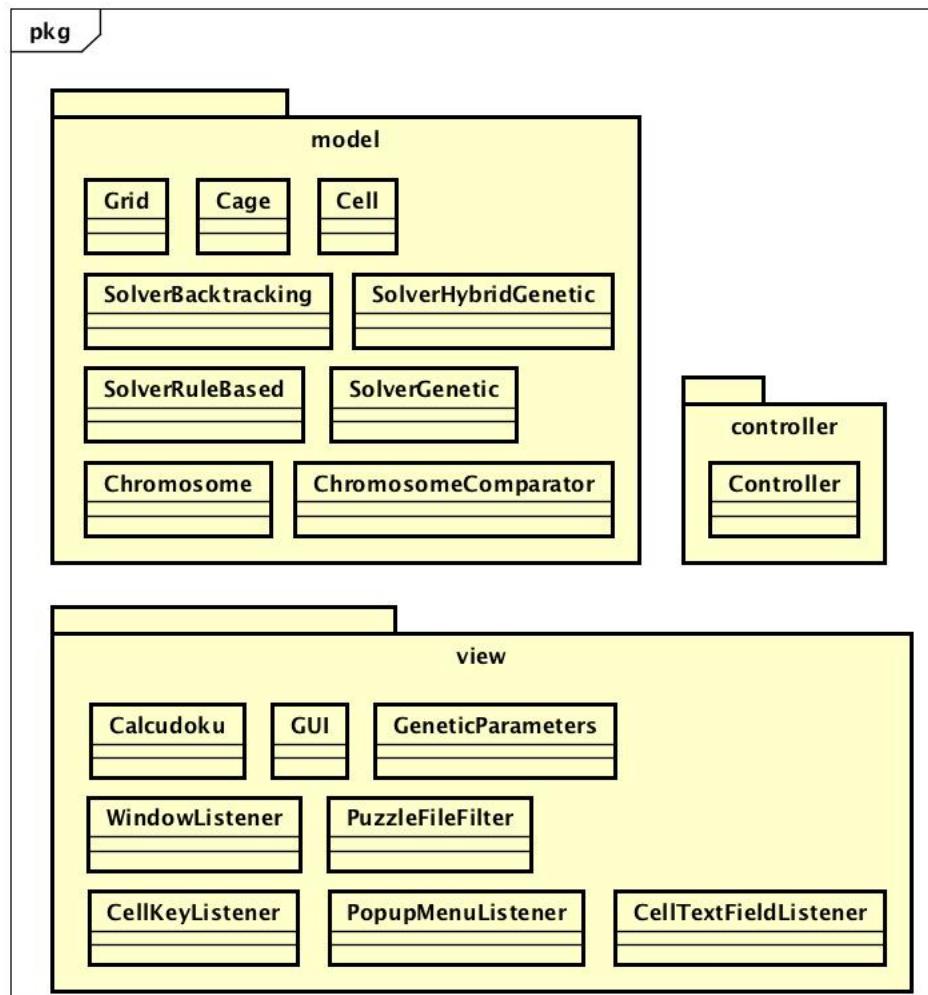
Perangkat lunak teka-teki Calcudoku ini terdiri dari beberapa kelas, yang dikelompokkan dalam tiga package, yaitu:

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*).
  - (j) 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.
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) PopupMenuItemListener, 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.
  - (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 4.8.

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



Gambar 4.8: Diagram kelas untuk perangkat lunak Calcudoku.

#### 4.4.1 Kelas Grid

Kelas Grid mempunyai beberapa atribut, yaitu:

1. size, yaitu ukuran dari matriks *grid*.
2. numberCages, 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 numberCages, 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 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 × berukuran minimal dua sel, dan setiap *cage* dengan operator - atau ÷ 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. `solverSetCellValue(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 4.9.

#### 4.4.2 Kelas Cage

Kelas Cage mempunyai beberapa atribut, yaitu:

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

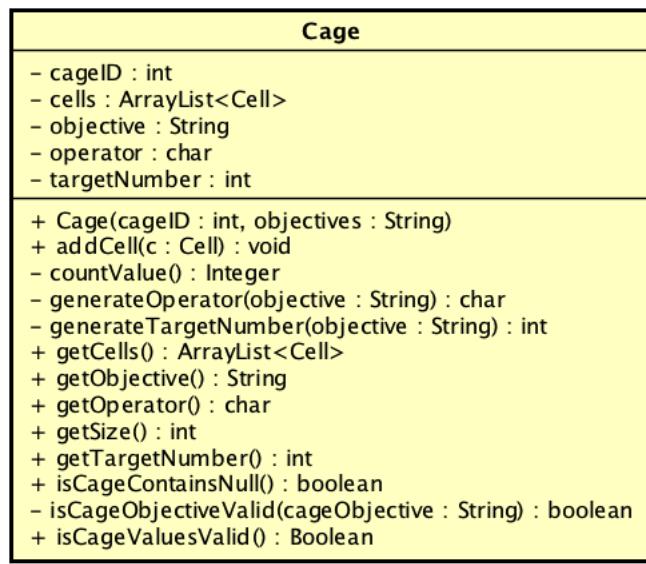
Kelas Cage mempunyai *method-method* berikut:

1. `Cage(int cageID, String objectives)`, yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa nomor dan *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*.



powered by Astah

Gambar 4.9: Diagram kelas Grid.

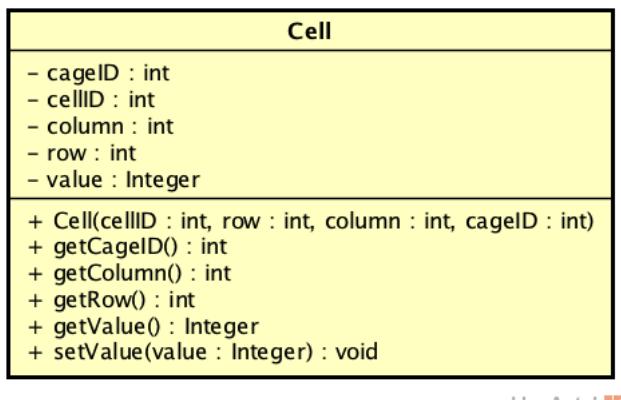


powered by Astah

Gambar 4.10: Diagram kelas 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.
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 4.10.



Gambar 4.11: Diagram kelas Cell.

#### 4.4.3 Kelas Cell

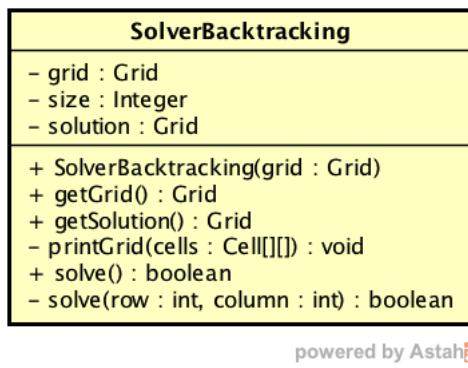
Kelas Cell mempunyai beberapa atribut, yaitu:

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

Kelas Cell mempunyai beberapa *method*, yaitu:

1. Cell(int CellID, int row, int column, int cageID), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa nomor sel, 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 4.11.



Gambar 4.12: Diagram kelas SolverBacktracking.

#### 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*.
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 4.12.

#### 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..
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:

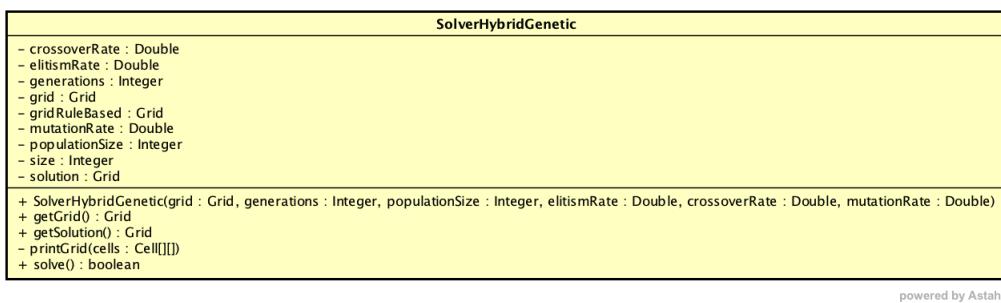
1. SolverHybridGenetic(Grid grid, Integer generations, Integer populationSize, Double elitismRate, Double crossoverRate, Double mutationRate), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *grid* yang akan diselesaikan oleh *solver* dengan algoritma *hybrid genetic*, dan parameter-parameter 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 4.13.

#### 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*.



Gambar 4.13: Diagram kelas SolverHybridGenetic.

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:

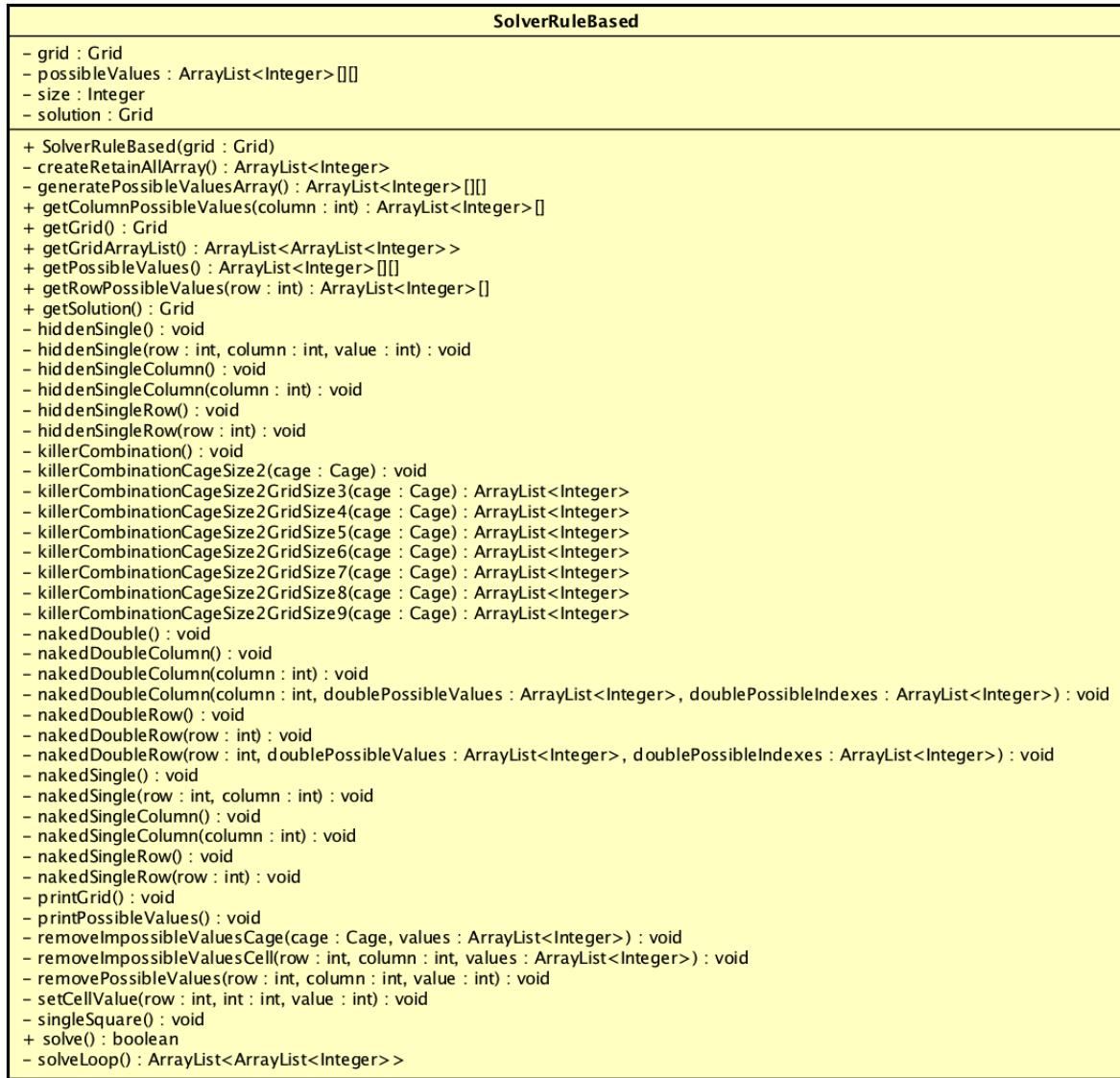
1. SolverRuleBased(Grid grid), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *grid* yang akan diselesaikan oleh *solver* dengan algoritma *rule based*, dan parameter-parameter algoritma genetik, yaitu jumlah generasi, jumlah populasi dalam sebuah generasi, tingkat *elitism*, 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  $3 \times 3$  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  $4 \times 4$  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  $5 \times 5$  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  $6 \times 6$  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  $7 \times 7$  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  $8 \times 8$  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  $9 \times 9$  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 menghapus 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 menghapus 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. `getPossibleValues`, yaitu *method* untuk mendapatkan kemungkinan angka-angka yang mungkin untuk setiap sel di dalam *grid*. *Method* ini menghasilkan keluaran berupa matriks dari *array* yang berisi kemungkinan angka-angka yang mungkin untuk setiap sel di dalam *grid*.
45. `printGrid()`, yaitu *method* untuk mencetak isi *grid* ke layar.
46. `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 4.14.



powered by Astah

Gambar 4.14: Diagram kelas SolverRuleBased.

#### 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.
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.

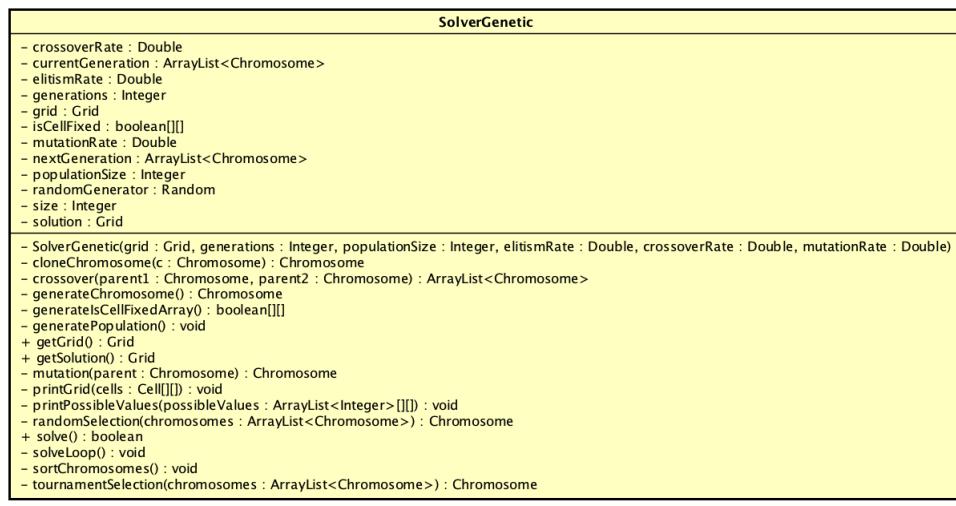
5. generateIsCellFixedArray(), yaitu *method* yang membangkitkan matriks yang berisi apakah sel tersebut sudah diisi oleh algoritma *rule based* atau tidak. *Method* ini menghasilkan keluaran berupa sebuah matriks yang berisi apakah sel tersebut sudah diisi oleh algoritma *rule based* atau tidak.
6. 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.
10. tournamentSelection(ArrayList<Chromosome> chromosomes), yaitu *method* untuk memilih sebuah kromosom dari sebuah populasi kromosom menggunakan metode *tournament selection*. *Method* ini menerima masukan berupa sebuah ArrayList yang berisi sekumpulan kromosom dan menghasilkan keluaran berupa sebuah kromosom yang terpilih.
11. 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.
12. 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.
13. 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 sebuah kromosom hasil mutasi.
14. getGrid, yaitu *method* untuk mendapatkan *grid*. Method ini menghasilkan keluaran berupa *grid*.
15. getSolution, yaitu *method* untuk mendapatkan *grid* yang sudah diselesaikan oleh algoritma genetik. *Method* ini menghasilkan keluaran berupa *grid* yang sudah diselesaikan oleh algoritma genetik.
16. printGrid(), yaitu *method* untuk mencetak isi *grid* ke layar.
17. printPossibleValues(), yaitu *method* untuk mencetak kemungkinan angka-angka yang valid untuk setiap sel di dalam *grid* ke layar. *Method* ini menerima masukan berupa sebuah *array list* yang berisi kemungkinan angka-angka yang valid untuk setiap sel di dalam *grid*.

Diagram kelas SolverGenetic dapat dilihat pada Gambar 4.15.

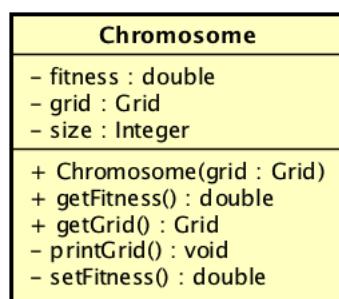
#### 4.4.8 Kelas Chromosome

Kelas Chromosome mempunyai beberapa atribut, yaitu:

1. grid, yaitu sebuah *grid* yang sudah diisi dengan angka-angka secara acak.



Gambar 4.15: Diagram kelas SolverGenetic.



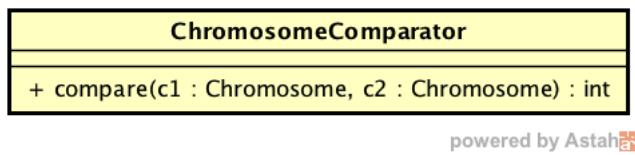
Gambar 4.16: Diagram kelas Chromosome.

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.
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*.
5. printGrid(), yaitu *method* untuk mencetak isi *grid* ke layar.

Diagram kelas Chromosome dapat dilihat pada Gambar 4.16.



Gambar 4.17: Diagram kelas ChromosomeComparator.

#### 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 4.17.

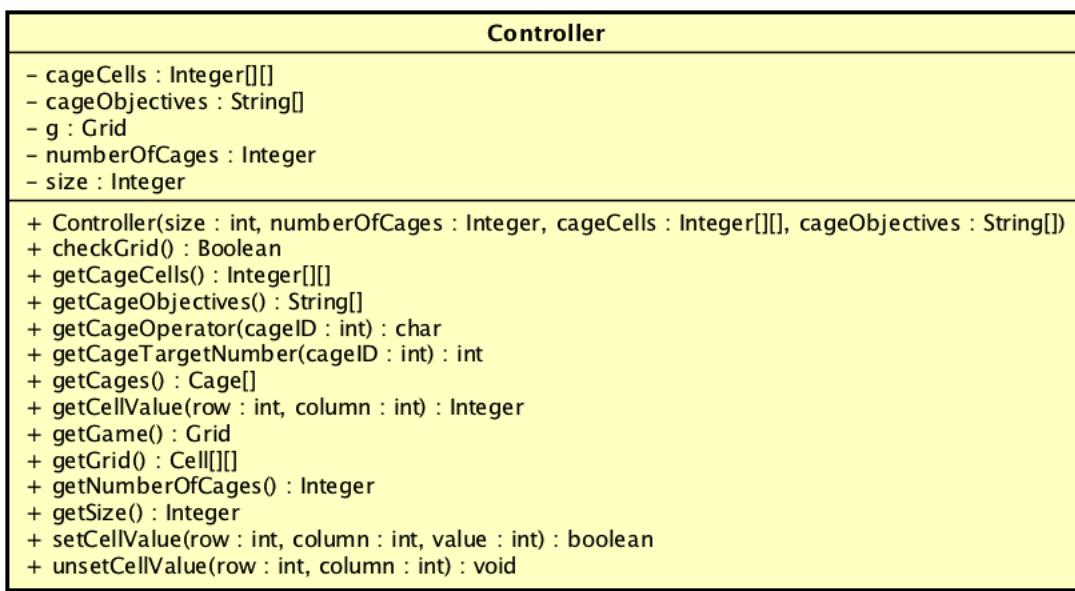
#### 4.4.10 Kelas Controller

Kelas Controller 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. `g`, yaitu 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



powered by Astah

Gambar 4.18: Diagram kelas Controller.

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. `getGames()`, 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 4.18.

#### 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. numberOfCages, 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.
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 *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, numberOfRowsInSection, 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 4.19.

#### 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.

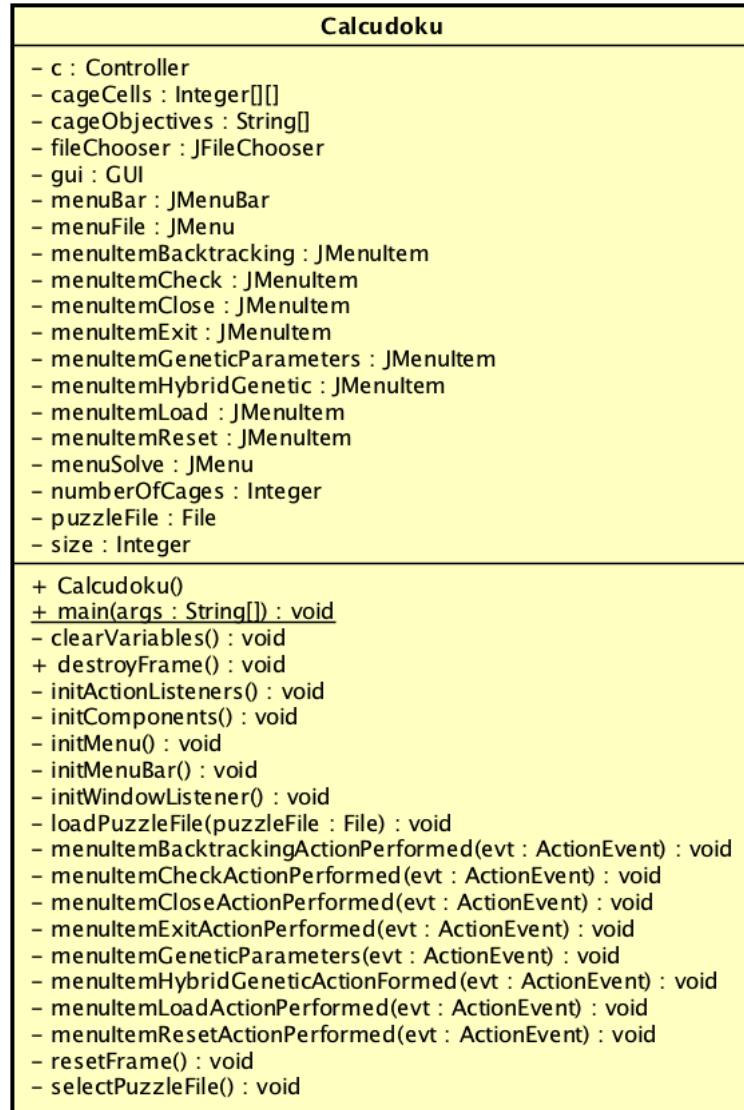
Diagram kelas WindowListener dapat dilihat pada Gambar 4.20.

#### 4.4.13 Kelas PuzzleFileFilter

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

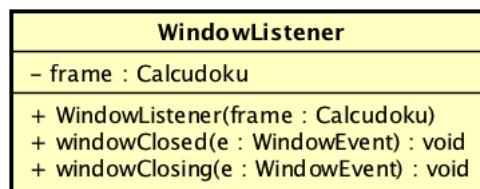
1. accept(File puzzleFile), yaitu *method* yang meng-override *method* dari kelas FileFilter. *Method* ini berfungsi untuk memeriksa apakah *file* yang akan dibuka memenuhi syarat atau tidak. *File* yang memenuhi syarat adalah *file* teks.
2. getDescription(), yaitu *method* yang meng-override *method*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

Diagram kelas PuzzleFileFilter dapat dilihat pada Gambar 4.21.



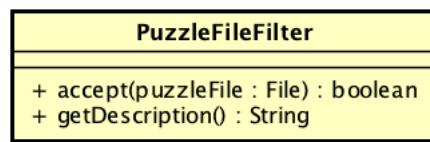
powered by Astah

Gambar 4.19: Diagram kelas Calcudoku.



powered by Astah

Gambar 4.20: Diagram kelas WindowListener.



powered by Astah

Gambar 4.21: 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 [4.22](#).

#### 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.



Gambar 4.22: Diagram kelas GUI.



Gambar 4.23: 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.

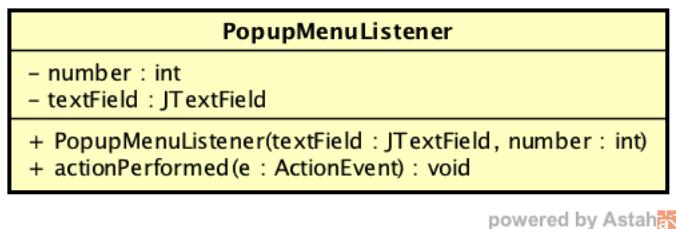
Diagram kelas CellKeyListener dapat dilihat pada Gambar 4.23.

#### 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*.
2. number, yaitu sebuah angka.

Kelas PopupMenuListener mempunyai beberapa *method*, yaitu:



Gambar 4.24: Diagram kelas PopupMenuItemListener.

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 4.24.

#### 4.4.17 Kelas CellTextFieldListener

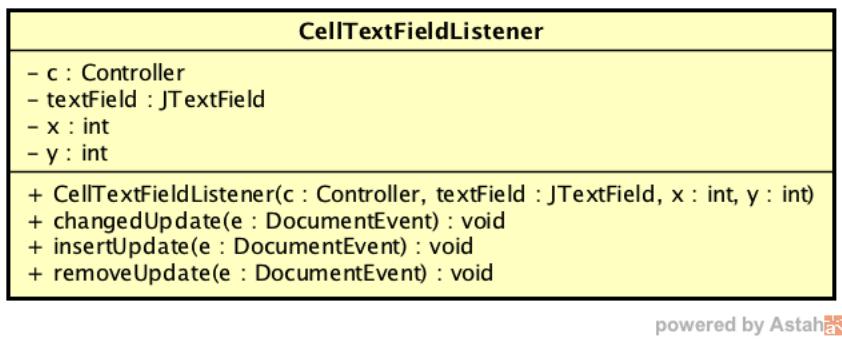
Kelas CellTextFieldListener mempunyai beberapa atribut, yaitu:

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 4.25.



Gambar 4.25: Diagram kelas CellTextFieldListner.

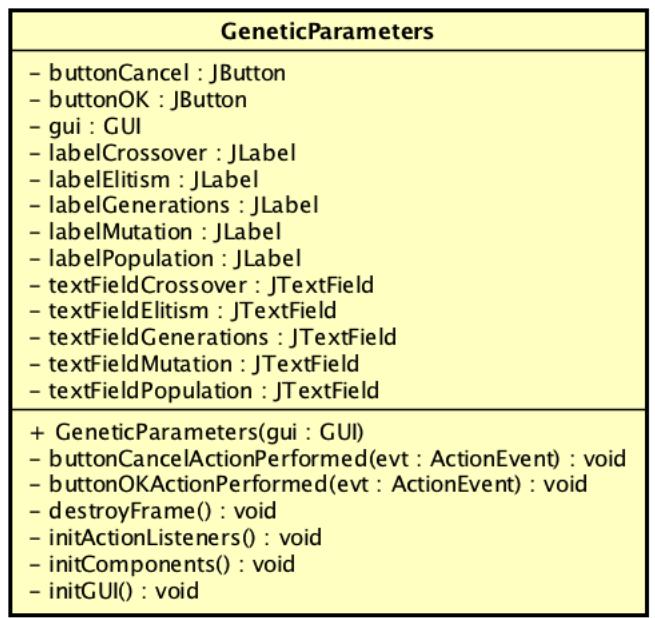
#### 4.4.18 Kelas GeneticParameters

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*.



powered by Astah

Gambar 4.26: Diagram kelas GeneticParameters.

5. `initMenuBar()`, yaitu *method* untuk menginisialisasi menu *bar* untuk *frame* GUI.
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 4.26.



# 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.

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

1. Lingkungan perangkat keras, yaitu lingkungan yang digunakan untuk pengujian perangkat lunak ini memiliki spesifikasi berikut:
2. Lingkungan perangkat lunak, yaitu lingkungan yang digunakan untuk pengujian perangkat lunak ini memiliki spesifikasi berikut:

### 5.2 Implementasi

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

1. Kode program
2. Antarmuka perangkat lunak

Kedua bagian tersebut akan dijelaskan lebih lanjut di bawah ini.

#### 5.2.1 Kode Program

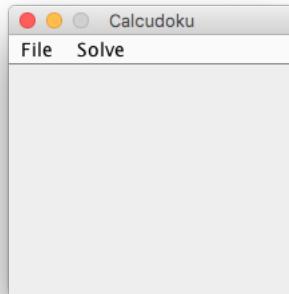
Kode program untuk perangkat lunak ini ditulis dalam bahasa pemrograman Java, berdasarkan dengan rancangan diagram kelas yang sudah dibuat, seperti dapat dilihat pada sub-bab 4.4. Seluruh kode program untuk perangkat lunak ini dapat dilihat di bab Lampiran.

Tabel 5.1: Lingkungan perangkat keras untuk pengujian perangkat lunak

Parameter	Nilai
<i>Processor</i>	Lorem Ipsum
<i>RAM (Random Access Memory)</i>	Lorem Ipsum
<i>VGA (Video Graphics Array)</i>	Lorem Ipsum

Tabel 5.2: Lingkungan perangkat lunak untuk pengujian perangkat lunak

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

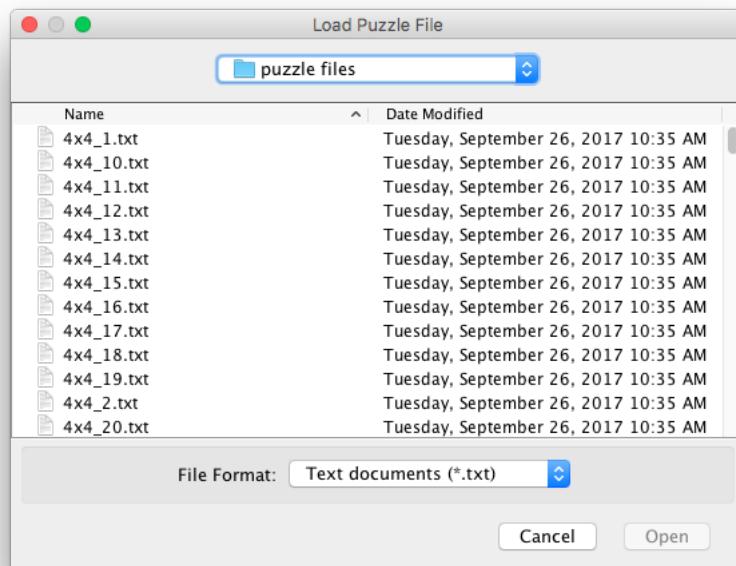


Gambar 5.1: Antarmuka perangkat lunak saat pertama kali dibuka

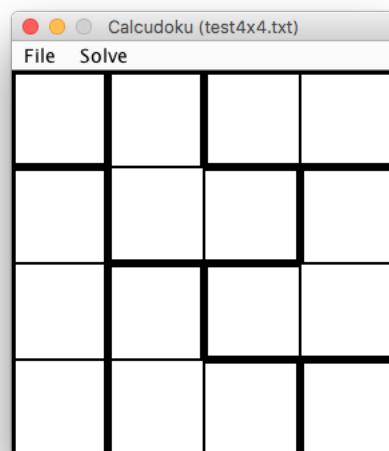
### 5.2.2 Antarmuka Perangkat Lunak

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

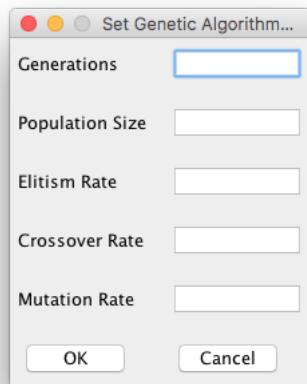
Gambar 5.1 menunjukkan antarmuka perangkat lunak saat pertama kali dibuka, sebelum *file* permainan dibuka. Gambar 5.2 menunjukkan kotak dialog untuk memilih *file* permainan yang akan dibuka. Gambar 5.3 menunjukkan antarmuka perangkat lunak setelah membuka *file* permainan yang dipilih. Gambar 5.4 menunjukkan 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.



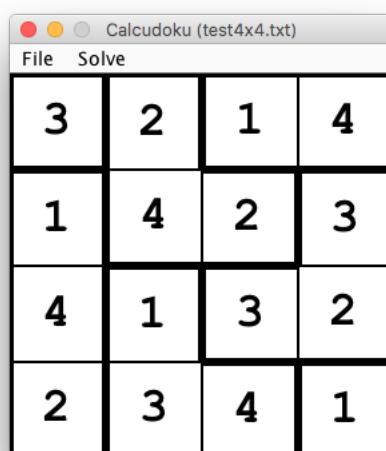
Gambar 5.2: Kotak dialog untuk memilih *file* permainan yang akan dibuka



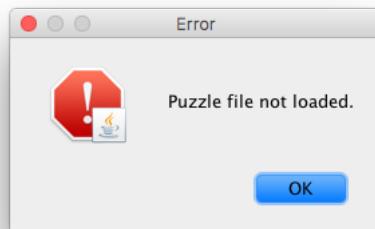
Gambar 5.3: Antarmuka perangkat lunak sesudah membuka *file* permainan yang dipilih



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*"

### 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.

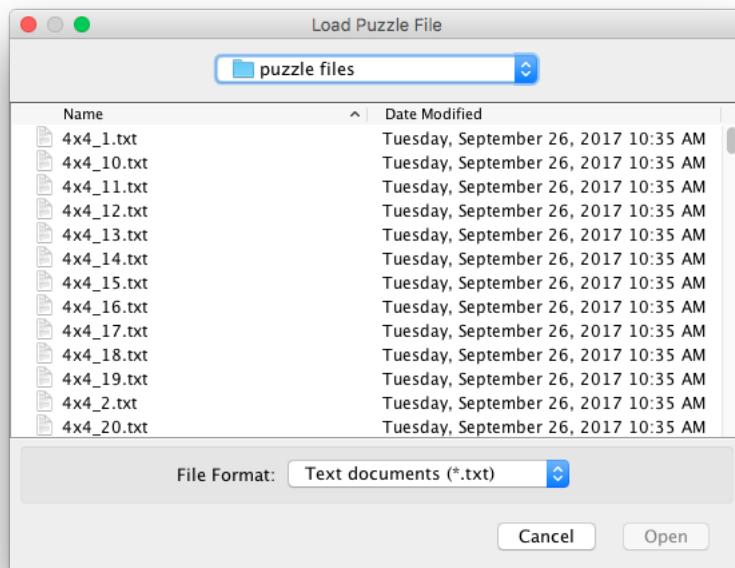
Jika *file* permainan yang dipilih berhasil dibuka, maka perangkat lunak akan menampilkan permainan berdasarkan *file* yang dipilih, seperti dapat dilihat pada Gambar 5.3.

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

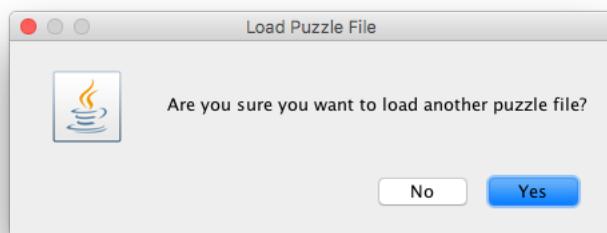
Jika *file* permainan yang dibuka tidak *valid*, misalnya karena *file* permainan yang dibuka bukan *file* teks, atau jika *file* teks yang akan dibuka formatnya tidak *valid*, maka pesan error "*Invalid puzzle file*" akan muncul, seperti dapat dilihat pada Gambar 5.9.

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.

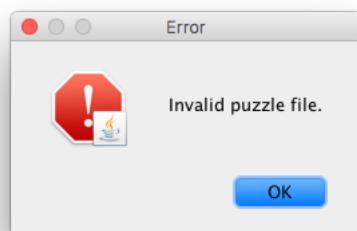
Jika salah satu dari kedua error tersebut terjadi, maka akan keluar pesan error "*Error in loading puzzle file*", seperti dapat dilihat pada Gambar 5.11.



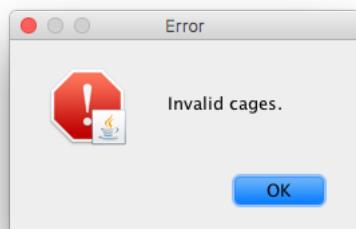
Gambar 5.7: Kotak pemilihan *file* permainan



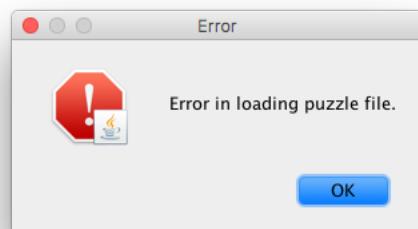
Gambar 5.8: Kotak dialog "Are you sure you want to load another puzzle file?"



Gambar 5.9: Pesan error "Invalid puzzle file"



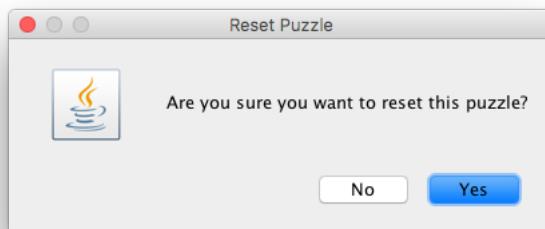
Gambar 5.10: Pesan error "*Invalid cages*"



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?*"

### 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.

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

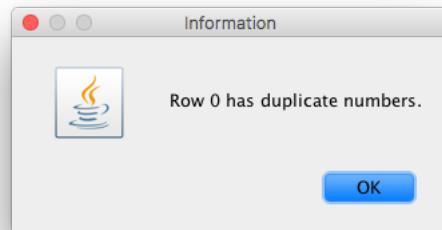
Perangkat lunak ini dapat memeriksa jika ada nilai yang salah di dalam *grid* secara otomatis.

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.

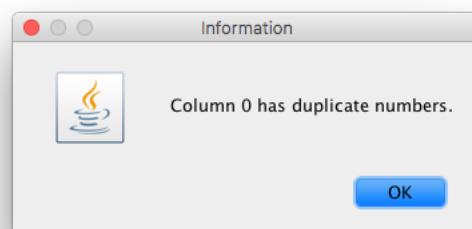
Jika ada nilai yang berulang dalam sebuah kolom, maka akan keluar kotak informasi "*Column* (nomor kolom) *has duplicate numbers*", seperti dapat dilihat pada Gambar 5.15.

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.

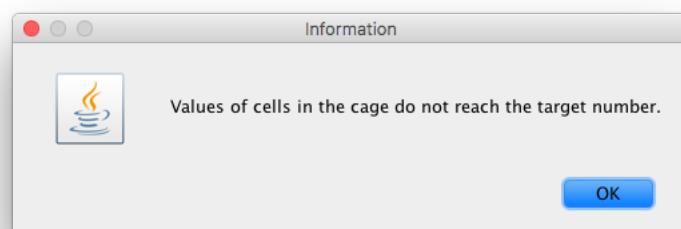
Pemain juga dapat meminta perangkat lunak untuk memeriksa permainan jika ada nilai yang salah di dalam *grid* secara manual, dengan memilih menu item "Check Puzzle" dalam menu "File".



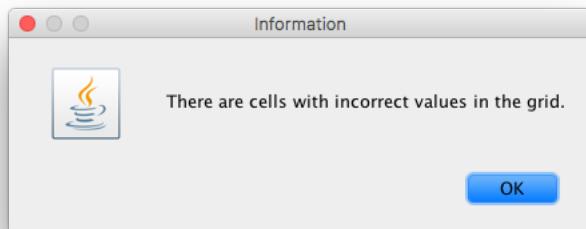
Gambar 5.14: Pesan informasi "*Row* (nomor baris) *has duplicate numbers*"



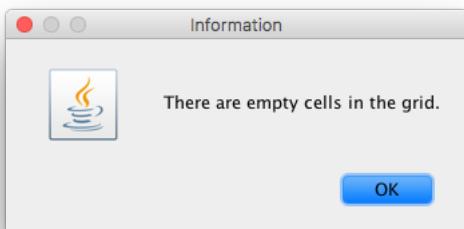
Gambar 5.15: Pesan informasi "*Column* (nomor kolom) *has duplicate numbers*"



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*"

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.

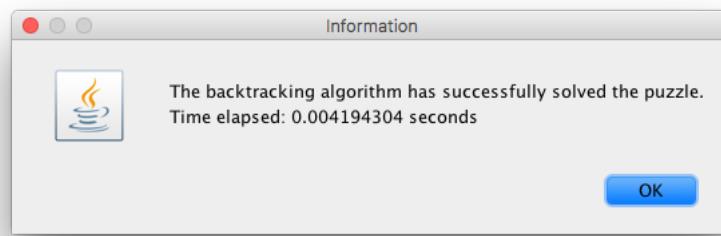
Jika menu *item* ini dijalankan dan ternyata ada sel yang masih kosong, maka akan muncul kotak informasi "*There are empty cells in the grid*", seperti dapat dilihat pada Gambar 5.18.

#### 6. Menyelesaikan permainan dengan menggunakan algoritma *backtracking*

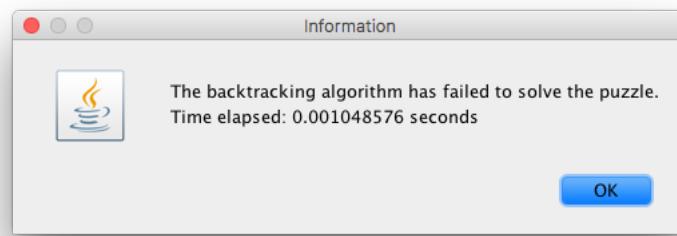
Pemain dapat meminta perangkat lunak untuk menyelesaikan permainan dengan salah satu dari dua *solver* yang disediakan. Salah satu dari kedua *solver* tersebut adalah *solver* dengan algoritma *backtracking*. Untuk menggunakan *solver* ini, pemain memilih menu *item* "*Backtracking*" dalam menu "*Solve*".

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.

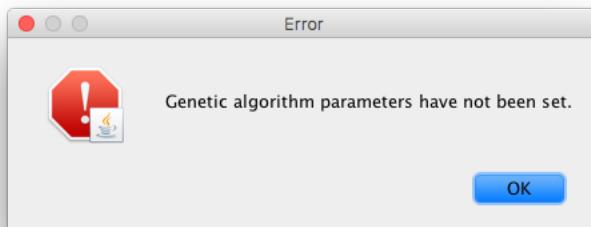
Jika gagal, maka akan muncul kotak informasi "*The backtracking algorithm has failed to solve the puzzle*", dan waktu yang dibutuhkan oleh *solver* untuk menemukan bahwa tidak ada solusi untuk permainan tersebut, seperti dapat dilihat pada Gambar 5.20



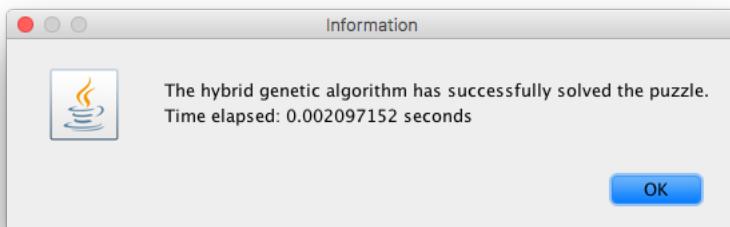
Gambar 5.19: Pesan informasi "*The backtracking algorithm has successfully solved the puzzle*"



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



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*"

#### 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 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.

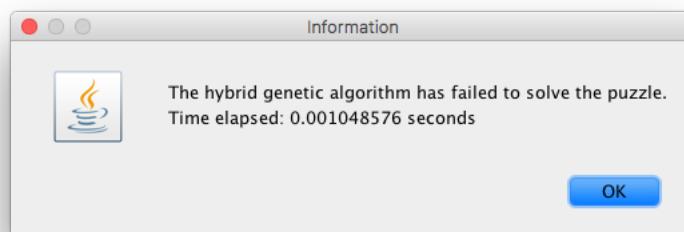
Jika *solver* ini berhasil dalam menyelesaikan permainan, maka akan muncul kotak informasi "*The hybrid genetic algorithm has successfully solved the puzzle*", dan waktu yang dibutuhkan oleh *solver* untuk menyelesaikan permainan tersebut, seperti dapat dilihat pada Gambar 5.22.

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

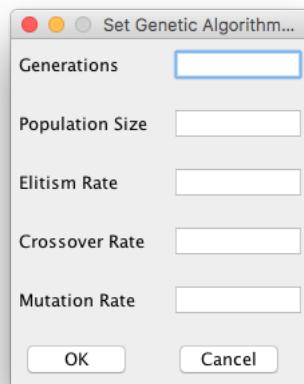
#### 8. Mengatur nilai untuk parameter-parameter algoritma genetik

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.

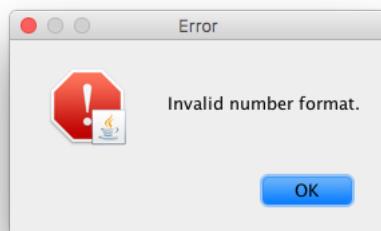
Isilah *form* tersebut dengan nilai yang diinginkan untuk setiap parameter algoritma genetik. Tekan tombol "*OK*" untuk menyimpan nilai-nilai telah diisikan ke dalam *form*, atau tekan tombol "*Cancel*" untuk membatalkan proses mengatur nilai untuk parameter-parameter genetik. Jika angka yang dimasukkan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



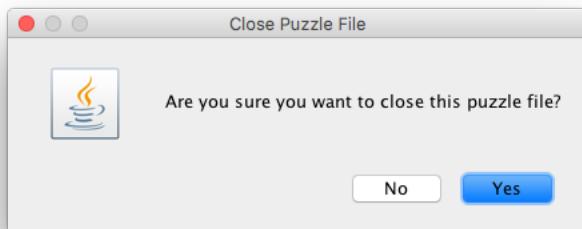
Gambar 5.23: Pesan informasi "*The hybrid genetic algorithm has failed to solve the puzzle*"



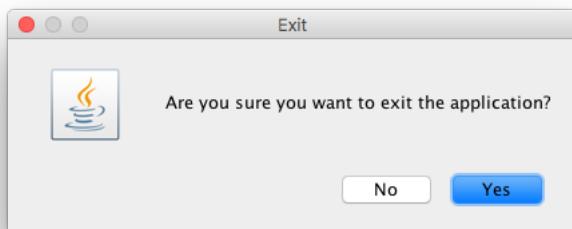
Gambar 5.24: *Form* untuk mengatur nilai untuk parameter-parameter algoritma genetik



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?"

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.

#### 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

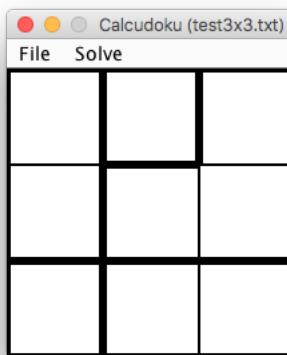
Pengujian keakuratan dilakukan untuk memastikan bahwa permainan yang ditampilkan oleh perangkat lunak sesuai dengan *file* permainan yang dibuka. Skenario-skenario yang akan dilakukan

```

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

```

Gambar 5.28: *File* masukan untuk pengujian keakuratan



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

dalam pengujian keakuratan untuk perangkat lunak ini adalah:

- 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.

Pada GUI ini, petunjuk, yaitu angka tujuan dan operasi matematika yang ditentukan untuk sebuah *cage*, 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 *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.

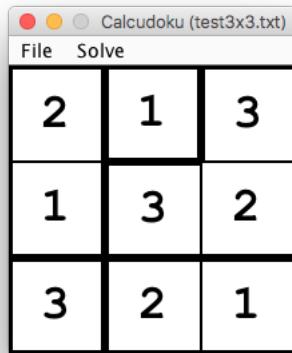
Dalam kasus ini, permainan berdasarkan *file* yang isinya ditampilkan pada Gambar 5.28 akan diselesaikan.

*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.



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

Tabel 5.3: Hasil pengujian algoritma *backtracking* untuk Calcudoku

Ukuran Grid	Tingkat Keberhasilan	Kecepatan
$4 \times 4$	100%	0.067 detik
$5 \times 5$	100%	0.701 detik
$6 \times 6$	100%	13.84 detik
$7 \times 7$	100%	482.653 detik
$8 \times 8$	100%	2134.655 detik

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.

## 5.5 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 5.3, 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.

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

Tabel 5.4: Nilai untuk parameter-parameter algoritma genetik untuk setiap percobaan yang dilakukan

Skenario	Populasi	Generasi	<i>Elitism</i>	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 5.5: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 1)

Ukuran <i>Grid</i>	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
4 × 4	100%	3.579 detik	3
5 × 5	42%	8.389 detik	9
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

Dalam kasus ini, pengujian eksperimental 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 5.4.

### 5.6.1 Skenario 1

Hasil pengujian Skenario 1 dapat dilihat pada Tabel 5.5.

Tabel 5.6: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 2)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	100%	4.002 detik	3
$5 \times 5$	42%	9.258 detik	9
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

Tabel 5.7: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 3)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	100%	3.751 detik	3
$5 \times 5$	42%	8.806 detik	9
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

### 5.6.2 Skenario 2

Hasil pengujian Skenario 2 dapat dilihat pada Tabel 5.6.

### 5.6.3 Skenario 3

Hasil pengujian Skenario 3 dapat dilihat pada Tabel 5.7.

### 5.6.4 Skenario 4

Hasil pengujian Skenario 4 dapat dilihat pada Tabel 5.8.

Tabel 5.8: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 4)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	100%	4.175 detik	3
$5 \times 5$	42%	9.676 detik	9
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

Tabel 5.9: 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	62%	0.498 detik	5
5 × 5	19%	0.311 detik	14
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

Tabel 5.10: 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	62%	0.553 detik	5
5 × 5	19%	0.339 detik	14
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

### 5.6.5 Skenario 5

Hasil pengujian Skenario 5 dapat dilihat pada Tabel 5.9.

### 5.6.6 Skenario 6

Hasil pengujian Skenario 6 dapat dilihat pada Tabel 5.9.

### 5.6.7 Skenario 7

Hasil pengujian Skenario 7 dapat dilihat pada Tabel 5.11.

Tabel 5.11: 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	62%	0.524 detik	5
5 × 5	19%	0.325 detik	14
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

Tabel 5.12: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 8)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	62%	0.579 detik	5
$5 \times 5$	19%	0.352 detik	14
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

Tabel 5.13: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 9)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	36%	0.511 detik	7
$5 \times 5$	15%	0.487 detik	15
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

### 5.6.8 Skenario 8

Hasil pengujian Skenario 8 dapat dilihat pada Tabel 5.12.

### 5.6.9 Skenario 9

Hasil pengujian Skenario 9 dapat dilihat pada Tabel 5.13.

### 5.6.10 Skenario 10

Hasil pengujian Skenario 10 dapat dilihat pada Tabel 5.14.

Tabel 5.14: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 10)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	36%	0.511 detik	7
$5 \times 5$	15%	0.487 detik	15
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

Tabel 5.15: 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	36%	0.511 detik	7
5 × 5	15%	0.487 detik	15
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

Tabel 5.16: 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	36%	0.511 detik	7
5 × 5	15%	0.487 detik	15
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

### 5.6.11 Skenario 11

Hasil pengujian Skenario 11 dapat dilihat pada Tabel 5.15.

### 5.6.12 Skenario 12

Hasil pengujian Skenario 12 dapat dilihat pada Tabel 5.16.

### 5.6.13 Skenario 13

Hasil pengujian Skenario 13 dapat dilihat pada Tabel 5.17.

Tabel 5.17: 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	34%	0.054 detik	9
5 × 5	15%	0.077 detik	15
6 × 6	0%	-	-
7 × 7	0%	-	-
8 × 8	0%	-	-

Tabel 5.18: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 14)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	34%	0.054 detik	9
$5 \times 5$	15%	0.077 detik	15
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

Tabel 5.19: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 15)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	34%	0.054 detik	9
$5 \times 5$	15%	0.077 detik	15
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

#### 5.6.14 Skenario 14

Hasil pengujian Skenario 14 dapat dilihat pada Tabel 5.18.

#### 5.6.15 Skenario 15

Hasil pengujian Skenario 15 dapat dilihat pada Tabel 5.19.

#### 5.6.16 Skenario 16

Hasil pengujian Skenario 16 dapat dilihat pada Tabel 5.20.

Tabel 5.20: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 16)

Ukuran Grid	Rata-Rata Tingkat Keberhasilan	Rata-Rata Kecepatan	Rata-Rata Jumlah Sel Diisi Algoritma <i>Rule Based</i>
$4 \times 4$	34%	0.054 detik	9
$5 \times 5$	15%	0.077 detik	15
$6 \times 6$	0%	-	-
$7 \times 7$	0%	-	-
$8 \times 8$	0%	-	-

### 5.6.17 Hasil Eksperimen

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

1. Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan Calcudoku 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*  $6 \times 6$  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

### SIMPULAN 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 Simpulan

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

1. Algoritma *backtracking* dapat menyelesaikan semua permainan yang diujikan. Pada ukuran *grid* yang besar, algoritma *backtracking* sangat lambat dalam menyelesaikan permainan.
2. Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan Calcudoku karena sifat acak dari algoritma *hybrid genetic* ini. Semakin besar ukuran *grid*, maka kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan semakin besar.
3. 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.
4. 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.
5. 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.

#### 6.2 Saran

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

1. 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.

2. 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](http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_016.pdf).
- [2] Johanna, O., Lukas, S., dan Saputra, K. V. I. (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 subbab 3.1 secara lengkap.

Untuk mengilustrasikan cara kerja algoritma *backtracking*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar A.1 sebagai contoh.

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.
9. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 9*), tetapi angka 2 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]

<sup>3</sup> 3	<sup>8+</sup>	<sup>3-</sup>	
<sup>7+</sup>			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.2: *State 4*

<sup>3</sup> 3	<sup>8+</sup> 1	<sup>3-</sup> 2	4
<sup>7+</sup>			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.3: *State 11*

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.
16. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 16*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.

<sup>3</sup> 3	<sup>8+</sup> 1	<sup>3-</sup> 3	
<sup>7+</sup>			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.4: *State 12*

<sup>3</sup> 3	<sup>8+</sup> 1	<sup>3-</sup> 4	4
7+			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.5: *State 17*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup>	
7+			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.6: *State 18*

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.
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

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	
7+			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.7: *State 19*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
7+			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.8: *State 23*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
7+ 1			
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.9: *State 24*

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.
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.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	3	4	
	8+	8+	
			1

Gambar A.10: *State 31*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4		
	8+	8+	
			1

Gambar A.11: *State 32*

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.
40. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-2 dengan angka 1 (*state 40*). Algoritma lalu maju ke sel berikutnya.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	
	8+	8+	
			1

Gambar A.12: *State 34*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.13: *State 37*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 1	<sup>8+</sup> 3	4
			1

Gambar A.14: *State 47*

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.
51. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 51*), tetapi angka 3 sudah pernah digunakan dalam kolom tersebut.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 1	<sup>8+</sup> 4	
			<sup>1</sup>

Gambar A.15: *State 48*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 1	<sup>8+</sup> 4	4
			<sup>1</sup>

Gambar A.16: *State 52*

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.
58. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 58*). Algoritma lalu maju ke sel berikutnya.

3	2	1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 2	<sup>8+</sup>	
			<sup>1</sup>

Gambar A.17: *State 53*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 3	<sup>8+</sup> 4	1
4	1	4	<sup>1</sup>

Gambar A.18: *State 68*

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.
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.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 3	<sup>8+</sup> 4	1
4	2		<sup>1</sup>

Gambar A.19: *State 69*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 3	<sup>8+</sup> 4	1
4	4		<sup>1</sup>

Gambar A.20: *State 71*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 3	<sup>8+</sup> 4	2
			<sup>1</sup>

Gambar A.21: *State 72*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 3	<sup>8+</sup> 4	4
			1

Gambar A.22: *State 74*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
2	<sup>8+</sup> 4	<sup>8+</sup>	
			1

Gambar A.23: *State 75*

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.
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.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
3	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.24: *State 76*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup>	<sup>8+</sup>	
			1

Gambar A.25: *State 77*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup>	
			1

Gambar A.26: *State 78*

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.
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.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup> 3	
			1

Gambar A.27: *State 81*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup> 3	2
			1

Gambar A.28: *State 83*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup> 3	2
2			1

Gambar A.29: *State 85*

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.
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.

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup> 3	2
2	3		1

Gambar A.30: *State 88*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup> 3	2
2	3	4	<sup>1</sup> 1

Gambar A.31: *State 92*

<sup>3</sup> 3	<sup>8+</sup> 2	<sup>3-</sup> 1	4
<sup>7+</sup> 1	4	2	3
4	<sup>8+</sup> 1	<sup>8+</sup> 3	2
2	3	4	<sup>1</sup> 1

Gambar A.32: *State 93*

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	Waktu
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  $grid 5 \times 5$ 

Nomor Soal	Waktu
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	Waktu
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	Waktu
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	Waktu
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*  $6 \times 6$  ke atas.

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $4 \times 4$  untuk Skenario 1 sampai dengan Skenario 4 dapat dilihat pada Tabel [B.6](#).

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $4 \times 4$  untuk Skenario 5 sampai dengan Skenario 8 dapat dilihat pada Tabel [B.7](#).

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $4 \times 4$  untuk Skenario 9 sampai dengan Skenario 12 dapat dilihat pada Tabel [B.8](#).

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $4 \times 4$  untuk Skenario 13 sampai dengan Skenario 16 dapat dilihat pada Tabel [B.9](#).

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $5 \times 5$  untuk Skenario 1 sampai dengan Skenario 4 dapat dilihat pada Tabel [B.10](#).

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $5 \times 5$  untuk Skenario 5 sampai dengan Skenario 8 dapat dilihat pada Tabel [B.11](#).

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $5 \times 5$  untuk Skenario 9 sampai dengan Skenario 12 dapat dilihat pada Tabel [B.12](#).

Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku dengan ukuran *grid*  $5 \times 5$  untuk Skenario 13 sampai dengan Skenario 16 dapat dilihat pada Tabel [B.13](#).

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

Nomor Soal	Waktu Skenario 1	Waktu Skenario 2	Waktu Skenario 3	Waktu 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	0.62 detik	0.62 detik	0.62 detik	0.62 detik
6	2.457 detik	2.763 detik	2.457 detik	2.763 detik
7	2.457 detik	2.763 detik	2.457 detik	2.763 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	1.232 detik	1.232 detik	1.232 detik	1.232 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	1.232 detik	1.232 detik	1.232 detik	1.232 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	2.457 detik	2.763 detik	2.457 detik	2.763 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	0.314 detik	0.314 detik	0.314 detik	0.314 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*  $4 \times 4$   
(Skenario 5-8)

Nomor Soal	Waktu Skenario 5	Waktu Skenario 6	Waktu Skenario 7	Waktu 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	0.229 detik	0.256 detik	0.229 detik	0.256 detik
6	0.999 detik	1.109 detik	1.054 detik	1.164 detik
7	0.999 detik	1.109 detik	1.054 detik	1.164 detik
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	0.339 detik	0.394 detik	0.366 detik	0.421 detik
14	0.036 detik	0.036 detik	0.036 detik	0.036 detik
15	Gagal	Gagal	Gagal	Gagal
16	0.339 detik	0.394 detik	0.366 detik	0.421 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	0.999 detik	1.109 detik	1.054 detik	1.164 detik
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	0.118 detik	0.118 detik	0.118 detik	0.118 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  $grid 4 \times 4$  (Skenario 9-12)

Nomor Soal	Waktu Skenario 9	Waktu Skenario 10	Waktu Skenario 11	Waktu Skenario 12
1	Gagal	Gagal	Gagal	Gagal
2	Gagal	Gagal	Gagal	Gagal
3	Gagal	Gagal	Gagal	Gagal
4	Gagal	Gagal	Gagal	Gagal
5	0.62 detik	0.62 detik	0.62 detik	0.62 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	1.232 detik	1.232 detik	1.232 detik	1.232 detik
14	0.314 detik	0.314 detik	0.314 detik	0.314 detik
15	Gagal	Gagal	Gagal	Gagal
16	1.232 detik	1.232 detik	1.232 detik	1.232 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	0.314 detik	0.314 detik	0.314 detik	0.314 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*  $4 \times 4$   
(Skenario 5-8)

Nomor Soal	Waktu Skenario 13	Waktu Skenario 14	Waktu Skenario 15	Waktu Skenario 16
1	Gagal	Gagal	Gagal	Gagal
2	Gagal	Gagal	Gagal	Gagal
3	Gagal	Gagal	Gagal	Gagal
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	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	0.118 detik	0.118 detik	0.118 detik	0.118 detik
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	Waktu Skenario 1	Waktu Skenario 2	Waktu Skenario 3	Waktu 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	Waktu Skenario 5	Waktu Skenario 6	Waktu Skenario 7	Waktu 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	Waktu Skenario 9	Waktu Skenario 10	Waktu Skenario 11	Waktu 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	Waktu Skenario 13	Waktu Skenario 14	Waktu Skenario 15	Waktu 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

## 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 $4 \times 4$

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+
```

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=
```

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

Listing C.4: 4x4\_4.txt

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=			

Listing C.5: 4x4\_5.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.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=			

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+			

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*			

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=			

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=

```

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+

```

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-

```

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=

```

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-

```

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*

```

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/			

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.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.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.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.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.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.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.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.28: 4x4\_28.txt

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

Listing C.29: 4x4\_29.txt

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

Listing C.31: 4x4\_31.txt

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

Listing C.32: 4x4\_32.txt

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

Listing C.33: 4x4\_33.txt

1	4	7	
2	1	1	2
3	3	4	4
4	3	6	6
5	3	6	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.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.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.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 $5 \times 5$

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.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.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.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=

```

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=

```

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-

```

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+

```

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| 40*
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 $6 \times 6$

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-

```

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+

```

Listing C.68: 6x6\_3.txt

```

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/

```

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=

```

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=

```

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-

```

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-

```

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/

```

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-

```

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=

```

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

```

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 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+
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 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 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 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 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 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*

```

Listing C.86: 6x6\_21.txt

```

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 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*
24|

```

Listing C.95: 6x6\_30.txt

```

1| 6 17
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=
25|

```

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*
24|

```

Listing C.97: 6x6\_32.txt

```

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|

```

19	2-
20	4*
21	24*
22	1-
23	2/

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

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

Listing C.101: 6x6\_36.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 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/

```

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

```

#### C.4 File Teks Soal-Soal Permainan Calcudoku dengan *Grid* Berukuran $7 \times 7$

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| 210*
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 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.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*

```

Listing C.110: 7x7\_6.txt

```

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=
27| 3/
28| 3=
29| 10+

```

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=
24| 1-
25| 6-
26| 3-
27| 3-
28| 24*

```

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-
11| 2/
12| 1-
13| 30*
14| 1008*
15| 14+
16| 5-
17| 6-
18| 11+
19| 11+
20| 9+
21| 1-
22| 5-
23| 1-
24| 1-
25| 14+
26| 9+
27| 1-
28| 2-
29| 2/

```

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	90*
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 $8 \times 8$

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-

```

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*
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.122: 8x8\_3.txt

```

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.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 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 18
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 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 10+
21 | 21+

```

22	1-
23	1-
24	2/
25	14+
26	4-
27	11+
28	105*
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 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 $9 \times 9$

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| 40*
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 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*
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*
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 permainan Calcudoku ini.

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 /**
8 * 
9 * @author michaeladrian39
10 */
11 public class Grid
12 {
13
14     private final Integer size;
15     private final Integer numberOfCages;
16     private final Integer[][] cageCells;
17     private final String[] cageObjectives;
18     private final Cell[][] grid;
19     private final Cage[] cages;
20
21     public Grid(Integer size, Integer numberOfCages, Integer[][] cageCells,
22                 String[] cageObjectives)
23     {
24         this.size = size;
25         this.numberOfCages = numberOfCages;
26         if (isCageCellsSizeValid(cageCells))
27         {
28             this.cageCells = cageCells;
29         }
30         else
31         {
32             JOptionPane.showMessageDialog(null, "Invalid array size.", "Error",
33                                         JOptionPane.ERROR_MESSAGE);
34             throw new IllegalStateException("Invalid array size.");
35         }
36         if (isCageObjectivesSizeValid(cageObjectives))
37         {
38             this.cageObjectives = cageObjectives;
39         }
40         else
41         {
42             JOptionPane.showMessageDialog(null, "Invalid array size.", "Error",
43                                         JOptionPane.ERROR_MESSAGE);
44             throw new IllegalStateException("Invalid array size.");
45         }
46         this.grid = new Cell[size][size];
47         this.cages = new Cage[numberOfCages];
48         generateCages(cages);
49         for (int i = 0; i < cages.length; i++)
50         {
51             Integer[][] array = new Integer[size][size];
52             for (int j = 0; j < cageCells.length; j++)
53             {
54                 for (int k = 0; k < cageCells[j].length; k++)
55                 {
56                     if (cageCells[j][k] == i + 1)
57                     {
58                         array[j][k] = 1;
59                     }
60                     else
61                     {
62                         array[j][k] = 0;
63                     }
64                 }
65             }
66             if (!isCageAssignmentValid(array))
67             {
68                 JOptionPane.showMessageDialog(null, "Invalid cage assignment.",
69                                         "Error", JOptionPane.ERROR_MESSAGE);
69                 throw new IllegalStateException("Invalid cage assignment.");
70             }
71 }
```

```

72     }
73     generateGrid(grid, cages);
74     if (!isCagesValid(cages))
75     {
76         JOptionPane.showMessageDialog(null, "Invalid cages.", "Error",
77             JOptionPane.ERROR_MESSAGE);
78         throw new IllegalStateException("Invalid cages.");
79     }
80 }
81
82 private int countAreas(Integer [][] array)
83 {
84     boolean [][] checked = new boolean [size] [size];
85     for (int i = 0; i < size; i++)
86     {
87         for (int j = 0; j < size; j++)
88         {
89             checked [i] [j] = false;
90         }
91     }
92     return countAreas(array, checked);
93 }
94
95 private int countAreas(Integer [][] array, boolean [][] checked)
96 {
97     int areas = 0;
98     for (int i = 0; i < array.length; i++)
99     {
100         for (int j = 0; j < array.length; j++)
101         {
102             if (checked [i] [j])
103             {
104                 continue;
105             }
106             if (array [i] [j] == 0)
107             {
108                 checked [i] [j] = true;
109                 continue;
110             }
111             areas++;
112             floodFill(i, j, array, checked);
113         }
114     }
115     return areas;
116 }
117
118 private void floodFill(int i, int j, Integer [][] array,
119                      boolean [][] checked)
120 {
121     if (array [i] [j] == 0 || checked [i] [j])
122     {
123         return;
124     }
125     checked [i] [j] = true;
126     if (j < array.length - 1)
127     {
128         floodFill(i, j + 1, array, checked);
129     }
130     if (i < array.length - 1)
131     {
132         floodFill(i + 1, j, array, checked);
133     }
134     if (j > 0)
135     {
136         floodFill(i, j - 1, array, checked);
137     }
138     if (i > 0)
139     {
140         floodFill(i - 1, j, array, checked);
141     }
142 }
143
144 private boolean isCageCellsSizeValid(Integer [][] cageCells)
145 {
146     if (cageCells.length == size)
147     {
148         for (int i = 0; i < size; i++)
149         {
150             if (cageCells [i].length != size)
151             {
152                 return false;
153             }
154         }
155     }
156     else
157     {
158         return false;
159     }
160     return true;
161 }
162
163 private boolean isCageObjectivesSizeValid(String [] cageObjectives)
164 {
165     return cageCells.length == size;
166 }
167
168 private boolean isCageAssignmentValid(Integer [][] array)
169 {
170     return countAreas(array) == 1;

```

```

171     }
172
173     private boolean isCagesValid(Cage[] cages)
174     {
175         for (Cage c : cages)
176         {
177             if (c.getOperator() == '=' && c.getSize() != 1)
178             {
179                 return false;
180             }
181             if ((c.getOperator() == '-' || c.getOperator() == '/'))
182                 && c.getSize() != 2)
183             {
184                 return false;
185             }
186             if ((c.getOperator() == '+' || c.getOperator() == '*'))
187                 && c.getSize() < 2)
188             {
189                 return false;
190             }
191         }
192         return true;
193     }
194
195     private void generateCages(Cage[] cages)
196     {
197         for (int i = 0; i < cages.length; i++)
198         {
199             cages[i] = new Cage(i, cageObjectives[i]);
200         }
201     }
202
203     private void generateGrid(Cell[][] grid, Cage[] cages)
204     {
205         for (int i = 0; i < cageCells.length; i++)
206         {
207             for (int j = 0; j < cageCells[i].length; j++)
208             {
209                 grid[i][j] = new Cell((i * size) + j, i, j,
210                     (cageCells[i][j] - 1));
211                 cages[cageCells[i][j] - 1].addCell(grid[i][j]);
212             }
213         }
214     }
215
216     public ArrayList<Integer> getRow(int rowNumber)
217     {
218         ArrayList<Integer> row = new ArrayList<>();
219         for (int i = 0; i < grid.length; i++)
220         {
221             if (grid[rowNumber][i].getValue() != null)
222             {
223                 row.add(grid[rowNumber][i].getValue());
224             }
225         }
226         return row;
227     }
228
229     public ArrayList<Integer> getColumn(int columnNumber)
230     {
231         ArrayList<Integer> column = new ArrayList<>();
232         for (Cell[] row : grid) {
233             if (row[columnNumber].getValue() != null) {
234                 column.add(row[columnNumber].getValue());
235             }
236         }
237         return column;
238     }
239
240     public ArrayList<Integer> getCageValues(int cageNumber)
241     {
242         ArrayList<Integer> cage = new ArrayList<>();
243         for (int i = 0; i < cages[cageNumber].getSize(); i++)
244         {
245             if (cages[cageNumber].getCells().get(i).getValue() != null)
246             {
247                 cage.add(cages[cageNumber].getCells().get(i).getValue());
248             }
249         }
250         return cage;
251     }
252
253     private boolean isArrayValid(ArrayList<Integer> array)
254     {
255         for (int i = 0; i < array.size(); i++)
256         {
257             for (int j = 0; j < array.size(); j++)
258             {
259                 if (Objects.equals(array.get(i), array.get(j)) && (i != j))
260                 {
261                     return false;
262                 }
263             }
264         }
265         return true;
266     }
267
268     private boolean isRowValid(int row)
269     {

```

```

270     ArrayList<Integer> array = getRow(row);
271     if (!isArrayValid(array))
272     {
273         JOptionPane.showMessageDialog(null,
274             "Row " + row + " has duplicate numbers.",
275             "Information", JOptionPane.INFORMATION_MESSAGE);
276         return false;
277     }
278     else
279     {
280         return true;
281     }
282 }
283
284 private boolean solverIsRowValid(int row)
285 {
286     ArrayList<Integer> array = getRow(row);
287     return isArrayValid(array);
288 }
289
290 private boolean isColumnValid(int column)
291 {
292     ArrayList<Integer> array = getColumn(column);
293     if (!isArrayValid(array))
294     {
295         JOptionPane.showMessageDialog(null,
296             "Column " + column + " has duplicate numbers.",
297             "Information", JOptionPane.INFORMATION_MESSAGE);
298         return false;
299     }
300     else
301     {
302         return true;
303     }
304 }
305
306 private boolean solverIsColumnValid(int column)
307 {
308     ArrayList<Integer> array = getColumn(column);
309     return isArrayValid(array);
310 }
311
312 private Boolean isCageValuesValid(int cageNumber)
313 {
314     if (cages[cageNumber].isCageValuesValid() == null)
315     {
316         return true;
317     }
318     else
319     {
320         return cages[cageNumber].isCageValuesValid();
321     }
322 }
323
324 private boolean isCageValid(int row, int column)
325 {
326     ArrayList<Integer> cage =
327         getCageValues(getGridContents()[row][column].get CageID());
328     if (isCageValuesValid(
329         getGridContents()[row][column].get CageID()) == false)
330     {
331         JOptionPane.showMessageDialog(null,
332             "Values of cells in the cage do not reach the target number.",
333             "Information", JOptionPane.INFORMATION_MESSAGE);
334         return false;
335     }
336     else
337     {
338         return true;
339     }
340 }
341
342 private boolean solverIsCageValid(int row, int column)
343 {
344     ArrayList<Integer> cage =
345         getCageValues(getGridContents()[row][column].get CageID());
346     return isCageValuesValid(
347         getGridContents()[row][column].get CageID()) == true;
348 }
349
350 public boolean isCellValueValid(int row, int column)
351 {
352     return (isRowValid(row) && isColumnValid(column)
353             && isCageValid(row, column));
354 }
355
356 public boolean solverIsCellValueValid(int row, int column)
357 {
358     return (solverIsRowValid(row) && solverIsColumnValid(column)
359             && solverIsCageValid(row, column));
360 }
361
362 public boolean setCellValue(int row, int column, Integer value)
363 {
364     if (value > 0 && value <= size)
365     {
366         getGridContents()[row][column].setValue(value);
367         isWin();
368         return isCellValueValid(row, column);
369     }
370 }

```

```

369     }
370   else
371   {
372     JOptionPane.showMessageDialog(null,
373       "Cell value must be between 1 and " + size + ".",
374       "Information", JOptionPane.INFORMATION_MESSAGE);
375     return false;
376   }
377 }
378
379 public boolean solverSetCellValue(int row, int column, Integer value)
380 {
381   if (value > 0 && value <= size)
382   {
383     getGridContents()[row][column].setValue(value);
384     return (solverIsRowValid(row) && solverIsColumnValid(column)
385       && solverIsCageValid(row, column));
386   }
387   else
388   {
389     return false;
390   }
391 }
392
393 public void unsetCellValue(int row, int column)
394 {
395   getGridContents()[row][column].setValue(null);
396 }
397
398 public Boolean isWin()
399 {
400   for (int i = 0; i < size; i++)
401   {
402     for (int j = 0; j < size; j++)
403     {
404       if (getGridContents()[i][j].getValue() == null)
405       {
406         return null;
407       }
408       if (isCellValueValid(i, j) == false)
409       {
410         return false;
411       }
412     }
413   }
414   JOptionPane.showMessageDialog(null,
415     "Congratulations, you have successfully solved the puzzle!",
416     "Information", JOptionPane.INFORMATION_MESSAGE);
417   return true;
418 }
419
420 public Boolean checkGrid()
421 {
422   for (int i = 0; i < size; i++)
423   {
424     for (int j = 0; j < size; j++)
425     {
426       if (getGridContents()[i][j].getValue() == null)
427       {
428         JOptionPane.showMessageDialog(null,
429           "There are empty cells in the grid.",
430           "Information", JOptionPane.INFORMATION_MESSAGE);
431         return null;
432       }
433       if (solverIsCellValueValid(i, j) == false)
434       {
435         JOptionPane.showMessageDialog(null,
436           "There are cells with incorrect values in the"
437           + "grid.", "Information",
438           JOptionPane.INFORMATION_MESSAGE);
439         return false;
440       }
441     }
442   }
443   JOptionPane.showMessageDialog(null,
444     "Congratulations, you have successfully solved the puzzle!",
445     "Information", JOptionPane.INFORMATION_MESSAGE);
446   return true;
447 }
448
449 public boolean isFilled()
450 {
451   for (int i = 0; i < size; i++)
452   {
453     for (int j = 0; j < size; j++)
454     {
455       if (getGridContents()[i][j].getValue() == null)
456       {
457         return false;
458       }
459     }
460   }
461   return true;
462 }
463
464 public Integer getCellValue(int row, int column)
465 {
466   return getGridContents()[row][column].getValue();
467 }

```

```

468
469     public Integer getSize()
470     {
471         return size;
472     }
473
474     public Integer getNumberOfCages()
475     {
476         return numberOfCages;
477     }
478
479     public Integer [][] getCageCells()
480     {
481         return cageCells;
482     }
483
484     public String [] getCageObjectives()
485     {
486         return cageObjectives;
487     }
488
489     public Cell [][] getGridContents()
490     {
491         return grid;
492     }
493
494     public Cage [] getCages()
495     {
496         return cages;
497     }
498
499     public Grid getGame()
500     {
501         return this;
502     }
503
504 }
```

Listing D.2: Cell.java

```

1 package model;
2
3 /**
4 * 
5 * @author michaeladrian39
6 */
7 public class Cell
8 {
9
10    private final int cellID;
11    private final int row;
12    private final int column;
13    private final int cageID;
14    private Integer value;
15
16    public Cell(int cellID, int row, int column, int cageID)
17    {
18        this.cellID = cellID;
19        this.row = row;
20        this.column = column;
21        this.cageID = cageID;
22    }
23
24    public void setValue(Integer value)
25    {
26        this.value = value;
27    }
28
29    public Integer getValue()
30    {
31        return value;
32    }
33
34    public int getRow()
35    {
36        return row;
37    }
38
39    public int getColumn()
40    {
41        return column;
42    }
43
44    public int getCageID()
45    {
46        return cageID;
47    }
48 }
```

Listing D.3: Cage.java

```

1 package model;
2
3 import java.util.ArrayList;
4 import javax.swing.JOptionPane;
```

```

5 /**
6 *
7 * @author michaeladrian39
8 */
9
10 public class Cage
11 {
12     private final int cageID;
13     private final String objective;
14     private final int targetNumber;
15     private final char operator;
16     private final ArrayList<Cell> cells;
17
18     public Cage(int cageID, String objective)
19     {
20         this.cageID = cageID;
21         if (isCageObjectiveValid(objective))
22         {
23             this.objective = objective;
24         }
25         else
26         {
27             JOptionPane.showMessageDialog(null, "Invalid cage objectives."
28                                         "Error", JOptionPane.ERROR_MESSAGE);
29             throw new IllegalStateException("Invalid cage objectives.");
30         }
31         this.targetNumber = generateTargetNumber(this.objective);
32         this.operator = generateOperator(this.objective);
33         this.cells = new ArrayList<>();
34     }
35
36     private boolean isCageObjectiveValid(String cageObjective)
37     {
38         return cageObjective.matches("\\d+[*+-/=]");
39     }
40
41     private int generateTargetNumber(String objective)
42     {
43         return Integer.parseInt(objective.substring(0,
44                                     objective.length() - 1));
45     }
46
47     private char generateOperator(String objective)
48     {
49         return objective.charAt(objective.length() - 1);
50     }
51
52     public void addCell(Cell c)
53     {
54         cells.add(c);
55     }
56
57     public boolean isCageContainsNull()
58     {
59         for (int i = 0; i < cells.size(); i++)
60         {
61             if (cells.get(i).getValue() == null)
62             {
63                 return true;
64             }
65         }
66         return false;
67     }
68
69     public Boolean isCageValuesValid()
70     {
71         if (countValue() == null)
72         {
73             return null;
74         }
75         else
76         {
77             return (countValue() == getTargetNumber());
78         }
79     }
80
81     private Integer countValue()
82     {
83         Integer value = null;
84         if (isCageContainsNull() == true)
85         {
86             value = null;
87         }
88         else
89         {
90             switch (getOperator())
91             {
92                 case '+':
93                     value = 0;
94                     for (int i = 0; i < cells.size(); i++)
95                     {
96                         value += cells.get(i).getValue();
97                     }
98                     break;
99                 case '-':
100                     if (cells.get(0).getValue() > cells.get(1).getValue())
101                     {
102                         value = cells.get(0).getValue();
103                     }
104                     break;
105             }
106         }
107     }

```

```

104             - cells.get(1).getValue();
105         }
106         else if (cells.get(1).getValue() > cells.get(0).getValue())
107     {
108             value = cells.get(1).getValue()
109             - cells.get(0).getValue();
110         }
111         else
112     {
113             value = 0;
114         }
115         break;
116     case '*':
117         value = 1;
118         for (int i = 0; i < cells.size(); i++)
119     {
120             value *= cells.get(i).getValue();
121         }
122         break;
123     case '/':
124         if (cells.get(0).getValue() > cells.get(1).getValue())
125     {
126             if (cells.get(0).getValue()
127                 % cells.get(1).getValue() == 0)
128             {
129                 value = cells.get(0).getValue()
130                 / cells.get(1).getValue();
131             }
132             else
133             {
134                 value = 0;
135             }
136         }
137         else if (cells.get(1).getValue() > cells.get(0).getValue())
138     {
139             if (cells.get(1).getValue()
140                 % cells.get(0).getValue() == 0)
141             {
142                 value = cells.get(1).getValue()
143                 / cells.get(0).getValue();
144             }
145             else
146             {
147                 value = 0;
148             }
149         }
150         else
151     {
152             value = 0;
153         }
154         break;
155     case '=':
156         value = cells.get(0).getValue();
157         break;
158     default :
159         value = null;
160     }
161 }
162 return value;
163 }
164 public String getObjective()
165 {
166     return objective;
167 }
168
169 public int getTargetNumber()
170 {
171     return targetNumber;
172 }
173
174 public char getOperator()
175 {
176     return operator;
177 }
178
179 public ArrayList<Cell> getCells()
180 {
181     return cells;
182 }
183
184 public int getSize()
185 {
186     return cells.size();
187 }
188
189 }
190 }
```

Listing D.4: SolverBacktracking.java

```

1 package model;
2
3 /**
4 *
5 * @author apple
6 */
7 public class SolverBacktracking
8 {
```

```

9
10    private final Grid grid;
11    private final Integer size;
12    private Grid solution;
13
14    public SolverBacktracking(Grid grid)
15    {
16        this.grid = grid;
17        this.size = grid.getSize();
18    }
19
20    public boolean solve()
21    {
22        if (solve(0, 0) == true)
23        {
24            this.solution = grid;
25            printGrid(solution.getGridContents());
26            return true;
27        }
28        else
29        {
30            return false;
31        }
32    }
33
34    private boolean solve(int row, int column)
35    {
36        if (column >= size)
37        {
38            column = 0;
39            row++;
40            if (row >= size)
41            {
42                printGrid(grid.getGridContents());
43                return true;
44            }
45        }
46        for (int value = 1; value <= size; value++)
47        {
48            printGrid(grid.getGridContents());
49            if (grid.solverSetCellValue(row, column, value))
50            {
51                if (solve(row, column + 1))
52                {
53                    return true;
54                }
55            }
56        }
57        printGrid(grid.getGridContents());
58        grid.unsetCellValue(row, column);
59        return false;
60    }
61
62    public Grid getGrid()
63    {
64        return grid;
65    }
66
67    public Grid getSolution()
68    {
69        return solution;
70    }
71
72    private void printGrid(Cell[][] cells)
73    {
74        for (int i = 0; i < size; i++)
75        {
76            for (int j = 0; j < size; j++)
77            {
78                System.out.print(cells[i][j].getValue() + " ");
79            }
80            System.out.println("");
81        }
82        System.out.println("");
83    }
84}
85

```

Listing D.5: SolverHybridGenetic.java

```

1 package model;
2
3 /**
4 * 
5 * @author michaeladrian39
6 */
7 public class SolverHybridGenetic
8 {
9
10    private final Grid grid;
11    private Grid gridRuleBased;
12    private final Integer size;
13    private Grid solution;
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
20 public SolverHybridGenetic(Grid grid, Integer generations,
21     Integer populationSize, Double elitismRate, Double crossoverRate,
22     Double mutationRate)
23 {
24     this.grid = grid;
25     this.size = grid.getSize();
26     this.generations = generations;
27     this.populationSize = populationSize;
28     this.elitismRate = elitismRate;
29     this.crossoverRate = crossoverRate;
30     this.mutationRate = mutationRate;
31 }
32
33 public boolean solve()
34 {
35     SolverRuleBased srb = new SolverRuleBased(grid);
36     boolean isFilled = srb.solve();
37     this.gridRuleBased = srb.getGrid();
38     if (isFilled)
39     {
40         this.solution = srb.getSolution();
41         printGrid(solution.getGridContents());
42         return true;
43     }
44     else
45     {
46         SolverGenetic sg = new SolverGenetic(gridRuleBased, generations,
47             populationSize, elitismRate, crossoverRate, mutationRate);
48         if (sg.solve() == true)
49         {
50             this.solution = sg.getSolution();
51             printGrid(solution.getGridContents());
52             return true;
53         }
54     }
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 }
81 }
```

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 /**
8 * 
9 * @author michaeladrian39
10 */
11 public class SolverRuleBased
12 {
13
14     private final Grid grid;
15     private final Integer size;
16     private Grid solution;
17     private ArrayList<Integer>[][] possibleValues;
18
19     public SolverRuleBased(Grid grid)
20     {
21         this.grid = grid;
22         this.size = grid.getSize();
23         this.possibleValues = generatePossibleValuesArray();
24     }
25
26     private ArrayList<Integer>[][] generatePossibleValuesArray()
27     {
28         possibleValues = new ArrayList[size][size];
29         for (int i = 0; i < size; i++)
30         {
31             for (int j = 0; j < size; j++)
32             {
```

```

33         possibleValues[i][j] = new ArrayList<Integer>();
34     }
35 }
36 for (int i = 0; i < size; i++)
37 {
38     for (int j = 0; j < size; j++)
39     {
40         for (int k = 1; k <= size; k++)
41         {
42             possibleValues[i][j].add(k);
43         }
44     }
45 }
46 return possibleValues;
47 }
48
49 public boolean solve()
50 {
51     singleSquare();
52     killerCombination();
53     ArrayList<ArrayList<Integer>> currentGridArrayList
54     = getGridArrayList();
55     ArrayList<ArrayList<Integer>> newGridArrayList = solveLoop();
56     while (!currentGridArrayList.equals(newGridArrayList))
57     {
58         printGrid();
59         printPossibleValues();
60         currentGridArrayList = newGridArrayList;
61         newGridArrayList = solveLoop();
62     }
63     if (grid.isFilled())
64     {
65         this.solution = grid;
66         return true;
67     }
68     else
69     {
70         return false;
71     }
72 }
73
74 private ArrayList<ArrayList<Integer>> solveLoop()
75 {
76     nakedSingle();
77     nakedDouble();
78     hiddenSingle();
79     return getGridArrayList();
80 }
81
82 public ArrayList<Integer>[] getRowPossibleValues(int row)
83 {
84     ArrayList<Integer>[] array;
85     array = new ArrayList[size];
86     for (int i = 0; i < size; i++)
87     {
88         array[i] = possibleValues[row][i];
89     }
90     return array;
91 }
92
93 public ArrayList<Integer>[] getColumnPossibleValues(int column)
94 {
95     ArrayList<Integer>[] array;
96     array = new ArrayList[size];
97     for (int i = 0; i < size; i++)
98     {
99         array[i] = possibleValues[i][column];
100    }
101   return array;
102 }
103
104 private void singleSquare()
105 {
106     for (Cage c : grid.getCages())
107     {
108         if (c.getSize() == 1)
109         {
110             setCellValue(c.getCells().get(0).getRow(),
111                         c.getCells().get(0).getColumn(), c.getTargetNumber());
112         }
113     }
114 }
115
116 private void killerCombination()
117 {
118     int cageSize;
119     ArrayList<Integer> array = new ArrayList();
120     for (Cage c : grid.getCages())
121     {
122         cageSize = c.getSize();
123         switch (cageSize)
124         {
125             case 2:
126                 killerCombinationCageSize2(c);
127                 break;
128             default:
129                 array = createRetainAllArray();
130                 removeImpossibleValuesCage(c, array);
131                 break;
132         }
133     }
134 }
135
136 private void killerCombinationCageSize2(Cage c)
137 {
138     int cageSize;
139     ArrayList<Integer> array = new ArrayList();
140     for (Cell cell : c.getCells())
141     {
142         cageSize = cell.getRow();
143         if (cageSize == 1)
144         {
145             array.add(cell.getValue());
146         }
147     }
148     if (array.size() == 2)
149     {
150         for (Cell cell : c.getCells())
151         {
152             if (array.contains(cell.getValue()))
153             {
154                 cell.setValue(0);
155             }
156         }
157     }
158 }
159
160 private ArrayList<Integer> createRetainAllArray()
161 {
162     ArrayList<Integer> array = new ArrayList();
163     for (int i = 1; i <= size; i++)
164     {
165         array.add(i);
166     }
167     return array;
168 }
169
170 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
171 {
172     for (Cell cell : c.getCells())
173     {
174         if (array.contains(cell.getValue()))
175         {
176             cell.setValue(0);
177         }
178     }
179 }
180
181 private void printGrid()
182 {
183     for (int i = 0; i < size; i++)
184     {
185         for (int j = 0; j < size; j++)
186         {
187             System.out.print(grid[i][j]);
188         }
189         System.out.println();
190     }
191 }
192
193 private void printPossibleValues()
194 {
195     for (int i = 0; i < size; i++)
196     {
197         for (int j = 0; j < size; j++)
198         {
199             System.out.print(possibleValues[i][j]);
200         }
201         System.out.println();
202     }
203 }
204
205 private void setCellValue(int row, int column, int value)
206 {
207     grid[row][column] = value;
208 }
209
210 private void nakedSingle()
211 {
212     for (int i = 0; i < size; i++)
213     {
214         for (int j = 0; j < size; j++)
215         {
216             if (grid[i][j] == 0)
217             {
218                 ArrayList<Integer> possibleValues = possibleValues[i][j];
219                 if (possibleValues.size() == 1)
220                 {
221                     grid[i][j] = possibleValues.get(0);
222                 }
223             }
224         }
225     }
226 }
227
228 private void nakedDouble()
229 {
230     for (int i = 0; i < size; i++)
231     {
232         for (int j = 0; j < size; j++)
233         {
234             if (grid[i][j] == 0)
235             {
236                 ArrayList<Integer> possibleValues = possibleValues[i][j];
237                 if (possibleValues.size() == 2)
238                 {
239                     for (int k = 0; k < size; k++)
240                     {
241                         if (grid[i][k] == 0)
242                         {
243                             ArrayList<Integer> kPossibleValues = possibleValues[k];
244                             if (kPossibleValues.contains(possibleValues.get(0)))
245                             {
246                                 grid[i][k] = possibleValues.get(1);
247                             }
248                             else if (kPossibleValues.contains(possibleValues.get(1)))
249                             {
250                                 grid[i][k] = possibleValues.get(0);
251                             }
252                         }
253                     }
254                 }
255             }
256         }
257     }
258 }
259
260 private void hiddenSingle()
261 {
262     for (int i = 0; i < size; i++)
263     {
264         for (int j = 0; j < size; j++)
265         {
266             if (grid[i][j] == 0)
267             {
268                 ArrayList<Integer> possibleValues = possibleValues[i][j];
269                 if (possibleValues.size() == 1)
270                 {
271                     grid[i][j] = possibleValues.get(0);
272                 }
273             }
274         }
275     }
276 }
277
278 private void solveLoop()
279 {
280     ArrayList<ArrayList<Integer>> currentGridArrayList
281     = getGridArrayList();
282     ArrayList<ArrayList<Integer>> newGridArrayList = solveLoop();
283     while (!currentGridArrayList.equals(newGridArrayList))
284     {
285         printGrid();
286         printPossibleValues();
287         currentGridArrayList = newGridArrayList;
288         newGridArrayList = solveLoop();
289     }
290 }
291
292 private void killerCombination()
293 {
294     for (Cage c : grid.getCages())
295     {
296         if (c.getSize() == 1)
297         {
298             setCellValue(c.getCells().get(0).getRow(),
299                         c.getCells().get(0).getColumn(), c.getTargetNumber());
300         }
301     }
302 }
303
304 private void killerCombinationCageSize2(Cage c)
305 {
306     int cageSize;
307     ArrayList<Integer> array = new ArrayList();
308     for (Cell cell : c.getCells())
309     {
310         cageSize = cell.getRow();
311         if (cageSize == 1)
312         {
313             array.add(cell.getValue());
314         }
315     }
316     if (array.size() == 2)
317     {
318         for (Cell cell : c.getCells())
319         {
320             if (array.contains(cell.getValue()))
321             {
322                 cell.setValue(0);
323             }
324         }
325     }
326 }
327
328 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
329 {
330     for (Cell cell : c.getCells())
331     {
332         if (array.contains(cell.getValue()))
333         {
334             cell.setValue(0);
335         }
336     }
337 }
338
339 private void printGrid()
340 {
341     for (int i = 0; i < size; i++)
342     {
343         for (int j = 0; j < size; j++)
344         {
345             System.out.print(grid[i][j]);
346         }
347         System.out.println();
348     }
349 }
350
351 private void printPossibleValues()
352 {
353     for (int i = 0; i < size; i++)
354     {
355         for (int j = 0; j < size; j++)
356         {
357             System.out.print(possibleValues[i][j]);
358         }
359         System.out.println();
360     }
361 }
362
363 private void setCellValue(int row, int column, int value)
364 {
365     grid[row][column] = value;
366 }
367
368 private void nakedSingle()
369 {
370     for (int i = 0; i < size; i++)
371     {
372         for (int j = 0; j < size; j++)
373         {
374             if (grid[i][j] == 0)
375             {
376                 ArrayList<Integer> possibleValues = possibleValues[i][j];
377                 if (possibleValues.size() == 1)
378                 {
379                     grid[i][j] = possibleValues.get(0);
380                 }
381             }
382         }
383     }
384 }
385
386 private void nakedDouble()
387 {
388     for (int i = 0; i < size; i++)
389     {
390         for (int j = 0; j < size; j++)
391         {
392             if (grid[i][j] == 0)
393             {
394                 ArrayList<Integer> possibleValues = possibleValues[i][j];
395                 if (possibleValues.size() == 2)
396                 {
397                     for (int k = 0; k < size; k++)
398                     {
399                         if (grid[i][k] == 0)
400                         {
401                             ArrayList<Integer> kPossibleValues = possibleValues[k];
402                             if (kPossibleValues.contains(possibleValues.get(0)))
403                             {
404                                 grid[i][k] = possibleValues.get(1);
405                             }
406                             else if (kPossibleValues.contains(possibleValues.get(1)))
407                             {
408                                 grid[i][k] = possibleValues.get(0);
409                             }
410                         }
411                     }
412                 }
413             }
414         }
415     }
416 }
417
418 private void hiddenSingle()
419 {
420     for (int i = 0; i < size; i++)
421     {
422         for (int j = 0; j < size; j++)
423         {
424             if (grid[i][j] == 0)
425             {
426                 ArrayList<Integer> possibleValues = possibleValues[i][j];
427                 if (possibleValues.size() == 1)
428                 {
429                     grid[i][j] = possibleValues.get(0);
430                 }
431             }
432         }
433     }
434 }
435
436 private void killerCombination()
437 {
438     for (Cage c : grid.getCages())
439     {
440         if (c.getSize() == 1)
441         {
442             setCellValue(c.getCells().get(0).getRow(),
443                         c.getCells().get(0).getColumn(), c.getTargetNumber());
444         }
445     }
446 }
447
448 private void killerCombinationCageSize2(Cage c)
449 {
450     int cageSize;
451     ArrayList<Integer> array = new ArrayList();
452     for (Cell cell : c.getCells())
453     {
454         cageSize = cell.getRow();
455         if (cageSize == 1)
456         {
457             array.add(cell.getValue());
458         }
459     }
460     if (array.size() == 2)
461     {
462         for (Cell cell : c.getCells())
463         {
464             if (array.contains(cell.getValue()))
465             {
466                 cell.setValue(0);
467             }
468         }
469     }
470 }
471
472 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
473 {
474     for (Cell cell : c.getCells())
475     {
476         if (array.contains(cell.getValue()))
477         {
478             cell.setValue(0);
479         }
480     }
481 }
482
483 private void printGrid()
484 {
485     for (int i = 0; i < size; i++)
486     {
487         for (int j = 0; j < size; j++)
488         {
489             System.out.print(grid[i][j]);
490         }
491         System.out.println();
492     }
493 }
494
495 private void printPossibleValues()
496 {
497     for (int i = 0; i < size; i++)
498     {
499         for (int j = 0; j < size; j++)
500         {
501             System.out.print(possibleValues[i][j]);
502         }
503         System.out.println();
504     }
505 }
506
507 private void setCellValue(int row, int column, int value)
508 {
509     grid[row][column] = value;
510 }
511
512 private void nakedSingle()
513 {
514     for (int i = 0; i < size; i++)
515     {
516         for (int j = 0; j < size; j++)
517         {
518             if (grid[i][j] == 0)
519             {
520                 ArrayList<Integer> possibleValues = possibleValues[i][j];
521                 if (possibleValues.size() == 1)
522                 {
523                     grid[i][j] = possibleValues.get(0);
524                 }
525             }
526         }
527     }
528 }
529
530 private void nakedDouble()
531 {
532     for (int i = 0; i < size; i++)
533     {
534         for (int j = 0; j < size; j++)
535         {
536             if (grid[i][j] == 0)
537             {
538                 ArrayList<Integer> possibleValues = possibleValues[i][j];
539                 if (possibleValues.size() == 2)
540                 {
541                     for (int k = 0; k < size; k++)
542                     {
543                         if (grid[i][k] == 0)
544                         {
545                             ArrayList<Integer> kPossibleValues = possibleValues[k];
546                             if (kPossibleValues.contains(possibleValues.get(0)))
547                             {
548                                 grid[i][k] = possibleValues.get(1);
549                             }
550                             else if (kPossibleValues.contains(possibleValues.get(1)))
551                             {
552                                 grid[i][k] = possibleValues.get(0);
553                             }
554                         }
555                     }
556                 }
557             }
558         }
559     }
560 }
561
562 private void hiddenSingle()
563 {
564     for (int i = 0; i < size; i++)
565     {
566         for (int j = 0; j < size; j++)
567         {
568             if (grid[i][j] == 0)
569             {
570                 ArrayList<Integer> possibleValues = possibleValues[i][j];
571                 if (possibleValues.size() == 1)
572                 {
573                     grid[i][j] = possibleValues.get(0);
574                 }
575             }
576         }
577     }
578 }
579
580 private void killerCombination()
581 {
582     for (Cage c : grid.getCages())
583     {
584         if (c.getSize() == 1)
585         {
586             setCellValue(c.getCells().get(0).getRow(),
587                         c.getCells().get(0).getColumn(), c.getTargetNumber());
588         }
589     }
590 }
591
592 private void killerCombinationCageSize2(Cage c)
593 {
594     int cageSize;
595     ArrayList<Integer> array = new ArrayList();
596     for (Cell cell : c.getCells())
597     {
598         cageSize = cell.getRow();
599         if (cageSize == 1)
600         {
601             array.add(cell.getValue());
602         }
603     }
604     if (array.size() == 2)
605     {
606         for (Cell cell : c.getCells())
607         {
608             if (array.contains(cell.getValue()))
609             {
610                 cell.setValue(0);
611             }
612         }
613     }
614 }
615
616 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
617 {
618     for (Cell cell : c.getCells())
619     {
620         if (array.contains(cell.getValue()))
621         {
622             cell.setValue(0);
623         }
624     }
625 }
626
627 private void printGrid()
628 {
629     for (int i = 0; i < size; i++)
630     {
631         for (int j = 0; j < size; j++)
632         {
633             System.out.print(grid[i][j]);
634         }
635         System.out.println();
636     }
637 }
638
639 private void printPossibleValues()
640 {
641     for (int i = 0; i < size; i++)
642     {
643         for (int j = 0; j < size; j++)
644         {
645             System.out.print(possibleValues[i][j]);
646         }
647         System.out.println();
648     }
649 }
650
651 private void setCellValue(int row, int column, int value)
652 {
653     grid[row][column] = value;
654 }
655
656 private void nakedSingle()
657 {
658     for (int i = 0; i < size; i++)
659     {
660         for (int j = 0; j < size; j++)
661         {
662             if (grid[i][j] == 0)
663             {
664                 ArrayList<Integer> possibleValues = possibleValues[i][j];
665                 if (possibleValues.size() == 1)
666                 {
667                     grid[i][j] = possibleValues.get(0);
668                 }
669             }
670         }
671     }
672 }
673
674 private void nakedDouble()
675 {
676     for (int i = 0; i < size; i++)
677     {
678         for (int j = 0; j < size; j++)
679         {
680             if (grid[i][j] == 0)
681             {
682                 ArrayList<Integer> possibleValues = possibleValues[i][j];
683                 if (possibleValues.size() == 2)
684                 {
685                     for (int k = 0; k < size; k++)
686                     {
687                         if (grid[i][k] == 0)
688                         {
689                             ArrayList<Integer> kPossibleValues = possibleValues[k];
690                             if (kPossibleValues.contains(possibleValues.get(0)))
691                             {
692                                 grid[i][k] = possibleValues.get(1);
693                             }
694                             else if (kPossibleValues.contains(possibleValues.get(1)))
695                             {
696                                 grid[i][k] = possibleValues.get(0);
697                             }
698                         }
699                     }
700                 }
701             }
702         }
703     }
704 }
705
706 private void hiddenSingle()
707 {
708     for (int i = 0; i < size; i++)
709     {
710         for (int j = 0; j < size; j++)
711         {
712             if (grid[i][j] == 0)
713             {
714                 ArrayList<Integer> possibleValues = possibleValues[i][j];
715                 if (possibleValues.size() == 1)
716                 {
717                     grid[i][j] = possibleValues.get(0);
718                 }
719             }
720         }
721     }
722 }
723
724 private void killerCombination()
725 {
726     for (Cage c : grid.getCages())
727     {
728         if (c.getSize() == 1)
729         {
730             setCellValue(c.getCells().get(0).getRow(),
731                         c.getCells().get(0).getColumn(), c.getTargetNumber());
732         }
733     }
734 }
735
736 private void killerCombinationCageSize2(Cage c)
737 {
738     int cageSize;
739     ArrayList<Integer> array = new ArrayList();
740     for (Cell cell : c.getCells())
741     {
742         cageSize = cell.getRow();
743         if (cageSize == 1)
744         {
745             array.add(cell.getValue());
746         }
747     }
748     if (array.size() == 2)
749     {
750         for (Cell cell : c.getCells())
751         {
752             if (array.contains(cell.getValue()))
753             {
754                 cell.setValue(0);
755             }
756         }
757     }
758 }
759
760 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
761 {
762     for (Cell cell : c.getCells())
763     {
764         if (array.contains(cell.getValue()))
765         {
766             cell.setValue(0);
767         }
768     }
769 }
770
771 private void printGrid()
772 {
773     for (int i = 0; i < size; i++)
774     {
775         for (int j = 0; j < size; j++)
776         {
777             System.out.print(grid[i][j]);
778         }
779         System.out.println();
780     }
781 }
782
783 private void printPossibleValues()
784 {
785     for (int i = 0; i < size; i++)
786     {
787         for (int j = 0; j < size; j++)
788         {
789             System.out.print(possibleValues[i][j]);
790         }
791         System.out.println();
792     }
793 }
794
795 private void setCellValue(int row, int column, int value)
796 {
797     grid[row][column] = value;
798 }
799
800 private void nakedSingle()
801 {
802     for (int i = 0; i < size; i++)
803     {
804         for (int j = 0; j < size; j++)
805         {
806             if (grid[i][j] == 0)
807             {
808                 ArrayList<Integer> possibleValues = possibleValues[i][j];
809                 if (possibleValues.size() == 1)
810                 {
811                     grid[i][j] = possibleValues.get(0);
812                 }
813             }
814         }
815     }
816 }
817
818 private void nakedDouble()
819 {
820     for (int i = 0; i < size; i++)
821     {
822         for (int j = 0; j < size; j++)
823         {
824             if (grid[i][j] == 0)
825             {
826                 ArrayList<Integer> possibleValues = possibleValues[i][j];
827                 if (possibleValues.size() == 2)
828                 {
829                     for (int k = 0; k < size; k++)
830                     {
831                         if (grid[i][k] == 0)
832                         {
833                             ArrayList<Integer> kPossibleValues = possibleValues[k];
834                             if (kPossibleValues.contains(possibleValues.get(0)))
835                             {
836                                 grid[i][k] = possibleValues.get(1);
837                             }
838                             else if (kPossibleValues.contains(possibleValues.get(1)))
839                             {
840                                 grid[i][k] = possibleValues.get(0);
841                             }
842                         }
843                     }
844                 }
845             }
846         }
847     }
848 }
849
850 private void hiddenSingle()
851 {
852     for (int i = 0; i < size; i++)
853     {
854         for (int j = 0; j < size; j++)
855         {
856             if (grid[i][j] == 0)
857             {
858                 ArrayList<Integer> possibleValues = possibleValues[i][j];
859                 if (possibleValues.size() == 1)
860                 {
861                     grid[i][j] = possibleValues.get(0);
862                 }
863             }
864         }
865     }
866 }
867
868 private void killerCombination()
869 {
870     for (Cage c : grid.getCages())
871     {
872         if (c.getSize() == 1)
873         {
874             setCellValue(c.getCells().get(0).getRow(),
875                         c.getCells().get(0).getColumn(), c.getTargetNumber());
876         }
877     }
878 }
879
880 private void killerCombinationCageSize2(Cage c)
881 {
882     int cageSize;
883     ArrayList<Integer> array = new ArrayList();
884     for (Cell cell : c.getCells())
885     {
886         cageSize = cell.getRow();
887         if (cageSize == 1)
888         {
889             array.add(cell.getValue());
890         }
891     }
892     if (array.size() == 2)
893     {
894         for (Cell cell : c.getCells())
895         {
896             if (array.contains(cell.getValue()))
897             {
898                 cell.setValue(0);
899             }
900         }
901     }
902 }
903
904 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
905 {
906     for (Cell cell : c.getCells())
907     {
908         if (array.contains(cell.getValue()))
909         {
910             cell.setValue(0);
911         }
912     }
913 }
914
915 private void printGrid()
916 {
917     for (int i = 0; i < size; i++)
918     {
919         for (int j = 0; j < size; j++)
920         {
921             System.out.print(grid[i][j]);
922         }
923         System.out.println();
924     }
925 }
926
927 private void printPossibleValues()
928 {
929     for (int i = 0; i < size; i++)
930     {
931         for (int j = 0; j < size; j++)
932         {
933             System.out.print(possibleValues[i][j]);
934         }
935         System.out.println();
936     }
937 }
938
939 private void setCellValue(int row, int column, int value)
940 {
941     grid[row][column] = value;
942 }
943
944 private void nakedSingle()
945 {
946     for (int i = 0; i < size; i++)
947     {
948         for (int j = 0; j < size; j++)
949         {
950             if (grid[i][j] == 0)
951             {
952                 ArrayList<Integer> possibleValues = possibleValues[i][j];
953                 if (possibleValues.size() == 1)
954                 {
955                     grid[i][j] = possibleValues.get(0);
956                 }
957             }
958         }
959     }
960 }
961
962 private void nakedDouble()
963 {
964     for (int i = 0; i < size; i++)
965     {
966         for (int j = 0; j < size; j++)
967         {
968             if (grid[i][j] == 0)
969             {
970                 ArrayList<Integer> possibleValues = possibleValues[i][j];
971                 if (possibleValues.size() == 2)
972                 {
973                     for (int k = 0; k < size; k++)
974                     {
975                         if (grid[i][k] == 0)
976                         {
977                             ArrayList<Integer> kPossibleValues = possibleValues[k];
978                             if (kPossibleValues.contains(possibleValues.get(0)))
979                             {
980                                 grid[i][k] = possibleValues.get(1);
981                             }
982                             else if (kPossibleValues.contains(possibleValues.get(1)))
983                             {
984                                 grid[i][k] = possibleValues.get(0);
985                             }
986                         }
987                     }
988                 }
989             }
990         }
991     }
992 }
993
994 private void hiddenSingle()
995 {
996     for (int i = 0; i < size; i++)
997     {
998         for (int j = 0; j < size; j++)
999         {
1000            if (grid[i][j] == 0)
1001            {
1002                ArrayList<Integer> possibleValues = possibleValues[i][j];
1003                if (possibleValues.size() == 1)
1004                {
1005                    grid[i][j] = possibleValues.get(0);
1006                }
1007            }
1008        }
1009    }
1010 }
1011
1012 private void killerCombination()
1013 {
1014     for (Cage c : grid.getCages())
1015     {
1016         if (c.getSize() == 1)
1017         {
1018             setCellValue(c.getCells().get(0).getRow(),
1019                         c.getCells().get(0).getColumn(), c.getTargetNumber());
1020         }
1021     }
1022 }
1023
1024 private void killerCombinationCageSize2(Cage c)
1025 {
1026     int cageSize;
1027     ArrayList<Integer> array = new ArrayList();
1028     for (Cell cell : c.getCells())
1029     {
1030         cageSize = cell.getRow();
1031         if (cageSize == 1)
1032         {
1033             array.add(cell.getValue());
1034         }
1035     }
1036     if (array.size() == 2)
1037     {
1038         for (Cell cell : c.getCells())
1039         {
1040             if (array.contains(cell.getValue()))
1041             {
1042                 cell.setValue(0);
1043             }
1044         }
1045     }
1046 }
1047
1048 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
1049 {
1050     for (Cell cell : c.getCells())
1051     {
1052         if (array.contains(cell.getValue()))
1053         {
1054             cell.setValue(0);
1055         }
1056     }
1057 }
1058
1059 private void printGrid()
1060 {
1061     for (int i = 0; i < size; i++)
1062     {
1063         for (int j = 0; j < size; j++)
1064         {
1065             System.out.print(grid[i][j]);
1066         }
1067         System.out.println();
1068     }
1069 }
1070
1071 private void printPossibleValues()
1072 {
1073     for (int i = 0; i < size; i++)
1074     {
1075         for (int j = 0; j < size; j++)
1076         {
1077             System.out.print(possibleValues[i][j]);
1078         }
1079         System.out.println();
1080     }
1081 }
1082
1083 private void setCellValue(int row, int column, int value)
1084 {
1085     grid[row][column] = value;
1086 }
1087
1088 private void nakedSingle()
1089 {
1090     for (int i = 0; i < size; i++)
1091     {
1092         for (int j = 0; j < size; j++)
1093         {
1094             if (grid[i][j] == 0)
1095             {
1096                 ArrayList<Integer> possibleValues = possibleValues[i][j];
1097                 if (possibleValues.size() == 1)
1098                 {
1099                     grid[i][j] = possibleValues.get(0);
1100                }
1101            }
1102        }
1103    }
1104 }
1105
1106 private void nakedDouble()
1107 {
1108     for (int i = 0; i < size; i++)
1109     {
1110         for (int j = 0; j < size; j++)
1111         {
1112             if (grid[i][j] == 0)
1113             {
1114                 ArrayList<Integer> possibleValues = possibleValues[i][j];
1115                 if (possibleValues.size() == 2)
1116                 {
1117                     for (int k = 0; k < size; k++)
1118                     {
1119                         if (grid[i][k] == 0)
1120                         {
1121                             ArrayList<Integer> kPossibleValues = possibleValues[k];
1122                             if (kPossibleValues.contains(possibleValues.get(0)))
1123                             {
1124                                 grid[i][k] = possibleValues.get(1);
1125                             }
1126                             else if (kPossibleValues.contains(possibleValues.get(1)))
1127                             {
1128                                 grid[i][k] = possibleValues.get(0);
1129                             }
1130                         }
1131                     }
1132                 }
1133             }
1134         }
1135     }
1136 }
1137
1138 private void hiddenSingle()
1139 {
1140     for (int i = 0; i < size; i++)
1141     {
1142         for (int j = 0; j < size; j++)
1143         {
1144             if (grid[i][j] == 0)
1145             {
1146                 ArrayList<Integer> possibleValues = possibleValues[i][j];
1147                 if (possibleValues.size() == 1)
1148                 {
1149                     grid[i][j] = possibleValues.get(0);
1150                 }
1151             }
1152         }
1153     }
1154 }
1155
1156 private void killerCombination()
1157 {
1158     for (Cage c : grid.getCages())
1159     {
1160         if (c.getSize() == 1)
1161         {
1162             setCellValue(c.getCells().get(0).getRow(),
1163                         c.getCells().get(0).getColumn(), c.getTargetNumber());
1164         }
1165     }
1166 }
1167
1168 private void killerCombinationCageSize2(Cage c)
1169 {
1170     int cageSize;
1171     ArrayList<Integer> array = new ArrayList();
1172     for (Cell cell : c.getCells())
1173     {
1174         cageSize = cell.getRow();
1175         if (cageSize == 1)
1176         {
1177             array.add(cell.getValue());
1178         }
1179     }
1180     if (array.size() == 2)
1181     {
1182         for (Cell cell : c.getCells())
1183         {
1184             if (array.contains(cell.getValue()))
1185             {
1186                 cell.setValue(0);
1187             }
1188         }
1189     }
1190 }
1191
1192 private void removeImpossibleValuesCage(Cage c, ArrayList<Integer> array)
1193 {
1194     for (Cell cell : c.getCells())
1195     {
1196         if (array.contains(cell.getValue()))
1197         {
1198             cell.setValue(0);
1199         }
1200     }
1201 }
1202
1203 private void printGrid()
1204 {
1205     for (int i = 0; i < size; i++)
1206     {
1207         for (int j = 0; j < size; j++)
1208         {
1209             System.out.print(grid[i][j]);
1210         }
1211         System.out.println();
1212     }
1213 }
1214
1215 private void printPossibleValues()
1216 {
1217     for (int i = 0; i < size; i++)
1218     {
1219         for (int j = 0; j < size; j++)
1220         {
1221             System.out.print(possibleValues[i][j]);
1222         }
1223         System.out.println();
1224     }
1225 }
1226
1227 private void setCellValue(int row, int column, int value)
1228 {
1229     grid[row][column] = value;
1230 }
1231
1232 private void nakedSingle()
1233 {
1234     for (int i = 0; i < size; i++)
1235     {
1236         for (int j = 0; j < size; j++)
1237         {
1238             if (grid[i][j] == 0)
1239             {
1240                 ArrayList<Integer> possibleValues = possibleValues[i][j];
1241                 if (possibleValues.size() == 1)
1242                 {
1243                     grid[i][j] = possibleValues.get(0);
1244                 }
1245             }
1246         }
1247     }
1248 }
1249
1250 private void nakedDouble()
1251 {
1252     for (int i = 0; i < size; i++)
1253     {
1254         for (int j = 0; j < size; j++)
1255         {
1256             if (grid[i][j] == 0)
1257             {
1258                 ArrayList<Integer> possibleValues = possibleValues[i][j];
1259                 if (possibleValues.size() == 2)
1260                 {
1261                     for (int k = 0; k < size; k++)
1262                     {
1263                         if (grid[i][k] == 0)
1264                         {
1265                             ArrayList<Integer> kPossibleValues = possibleValues[k];
1266                             if (kPossibleValues.contains(possibleValues.get(0)))
1267                             {
1268                                 grid[i][k] = possibleValues.get(1);
1269                             }
1270                             else if (kPossibleValues.contains(possibleValues.get(1)))
1271                             {
1272                                 grid[i][k] = possibleValues.get(0);
1273                             }
1274                         }
1275                     }
1276                 }
1277             }
1278         }
1279     }
1280 }

```

```

132     }
133 }
134 }
135 }
136 private void killerCombinationCageSize2(Cage cage)
137 {
138     int gridSize = size;
139     char cageOperator = cage.getOperator();
140     int cageTargetNumber = cage.getTargetNumber();
141     ArrayList<Integer> array = new ArrayList();
142     switch (gridSize)
143     {
144         case 3 :
145             array = killerCombinationCageSize2GridSize3(cage);
146             break;
147         case 4 :
148             array = killerCombinationCageSize2GridSize4(cage);
149             break;
150         case 5 :
151             array = killerCombinationCageSize2GridSize5(cage);
152             break;
153         case 6 :
154             array = killerCombinationCageSize2GridSize6(cage);
155             break;
156         case 7 :
157             array = killerCombinationCageSize2GridSize7(cage);
158             break;
159         case 8 :
160             array = killerCombinationCageSize2GridSize8(cage);
161             break;
162         case 9 :
163             array = killerCombinationCageSize2GridSize9(cage);
164             break;
165         default :
166             JOptionPane.showMessageDialog(null, "Invalid grid size .",
167                                         "Error", JOptionPane.ERROR_MESSAGE);
168             throw new IllegalStateException("Invalid grid size .");
169     }
170     removeImpossibleValuesCage(cage, array);
171 }
172
173 private ArrayList<Integer> killerCombinationCageSize2GridSize3(Cage cage)
174 {
175     char cageOperator = cage.getOperator();
176     int cageTargetNumber = cage.getTargetNumber();
177     ArrayList<Integer> array = new ArrayList();
178     switch (cageOperator)
179     {
180         case '+' :
181             switch (cageTargetNumber)
182             {
183                 case 3 :
184                     array.add(1);
185                     array.add(2);
186                     break;
187                 case 4 :
188                     array.add(1);
189                     array.add(3);
190                     break;
191                 case 5 :
192                     array.add(2);
193                     array.add(3);
194                     break;
195                 default :
196                     array = createRetainAllArray();
197                     break;
198             }
199             break;
200         case '-' :
201             switch (cageTargetNumber)
202             {
203                 case 2 :
204                     array.add(1);
205                     array.add(3);
206                     break;
207                 default :
208                     array = createRetainAllArray();
209                     break;
210             }
211             break;
212         case '*' :
213             switch (cageTargetNumber)
214             {
215                 case 2 :
216                     array.add(1);
217                     array.add(2);
218                     break;
219                 case 3 :
220                     array.add(1);
221                     array.add(3);
222                     break;
223                 case 6 :
224                     array.add(2);
225                     array.add(3);
226                     break;
227                 default :
228                     array = createRetainAllArray();
229                     break;
230             }

```

```

231         break;
232     case '/':
233         switch (cageTargetNumber)
234     {
235         case 2:
236             array.add(1);
237             array.add(2);
238             break;
239         case 3:
240             array.add(1);
241             array.add(3);
242             break;
243         default:
244             array = createRetainAllArray();
245             break;
246     }
247     break;
248     default:
249         JOptionPane.showMessageDialog(null,
250             "Invalid operator.", "Error",
251             JOptionPane.ERROR_MESSAGE);
252         throw new IllegalStateException("Invalid operator.");
253     }
254     return array;
255 }
256
257 private ArrayList<Integer> killerCombinationCageSize2GridSize4(Cage cage)
258 {
259     char cageOperator = cage.getOperator();
260     int cageTargetNumber = cage.getTargetNumber();
261     ArrayList<Integer> array = new ArrayList();
262     switch (cageOperator)
263     {
264         case '+':
265             switch (cageTargetNumber)
266             {
267                 case 3:
268                     array.add(1);
269                     array.add(2);
270                     break;
271                 case 4:
272                     array.add(1);
273                     array.add(3);
274                     break;
275                 case 6:
276                     array.add(2);
277                     array.add(4);
278                     break;
279                 case 7:
280                     array.add(3);
281                     array.add(4);
282                     break;
283                 default:
284                     array = createRetainAllArray();
285                     break;
286             }
287             break;
288         case '-':
289             switch (cageTargetNumber)
290             {
291                 case 3:
292                     array.add(1);
293                     array.add(4);
294                     break;
295                 default:
296                     array = createRetainAllArray();
297                     break;
298             }
299             break;
300         case '*':
301             switch (cageTargetNumber)
302             {
303                 case 2:
304                     array.add(1);
305                     array.add(2);
306                     break;
307                 case 3:
308                     array.add(1);
309                     array.add(3);
310                     break;
311                 case 4:
312                     array.add(1);
313                     array.add(4);
314                     break;
315                 case 6:
316                     array.add(2);
317                     array.add(3);
318                     break;
319                 case 8:
320                     array.add(2);
321                     array.add(4);
322                     break;
323                 case 12:
324                     array.add(3);
325                     array.add(4);
326                     break;
327                 default:
328                     array = createRetainAllArray();
329                     break;
330             }
331     }
332 }
```

```

330 }
331     }
332     case '/':
333         switch (cageTargetNumber)
334     {
335         case 3:
336             array.add(1);
337             array.add(3);
338             break;
339         case 4:
340             array.add(1);
341             array.add(4);
342             break;
343         default:
344             array = createRetainAllArray();
345             break;
346     }
347     break;
348 default:
349     JOptionPane.showMessageDialog(null,
350         "Invalid operator.", "Error",
351         JOptionPane.ERROR_MESSAGE);
352     throw new IllegalStateException("Invalid operator.");
353 }
354 return array;
355 }

356 private ArrayList<Integer> killerCombinationCageSize2GridSize5(Cage cage)
357 {
358     char cageOperator = cage.getOperator();
359     int cageTargetNumber = cage.getTargetNumber();
360     ArrayList<Integer> array = new ArrayList();
361     switch (cageOperator)
362     {
363         case '+':
364             switch (cageTargetNumber)
365             {
366                 case 3:
367                     array.add(1);
368                     array.add(2);
369                     break;
370                 case 4:
371                     array.add(1);
372                     array.add(3);
373                     break;
374                 case 8:
375                     array.add(3);
376                     array.add(5);
377                     break;
378                 case 9:
379                     array.add(4);
380                     array.add(5);
381                     break;
382                 default:
383                     array = createRetainAllArray();
384                     break;
385             }
386             break;
387         case '-':
388             switch (cageTargetNumber)
389             {
390                 case 4:
391                     array.add(1);
392                     array.add(5);
393                     break;
394                 default:
395                     array = createRetainAllArray();
396                     break;
397             }
398             break;
399         case '*':
400             switch (cageTargetNumber)
401             {
402                 case 2:
403                     array.add(1);
404                     array.add(2);
405                     break;
406                 case 3:
407                     array.add(1);
408                     array.add(3);
409                     break;
410                 case 4:
411                     array.add(1);
412                     array.add(4);
413                     break;
414                 case 5:
415                     array.add(1);
416                     array.add(5);
417                     break;
418                 case 6:
419                     array.add(2);
420                     array.add(3);
421                     break;
422                 case 8:
423                     array.add(2);
424                     array.add(4);
425                     break;
426                 case 10:
427                     array.add(2);
428             }

```



```

528         array.add(1);
529         array.add(3);
530         break;
531     case 4 :
532         array.add(1);
533         array.add(4);
534         break;
535     case 5 :
536         array.add(1);
537         array.add(5);
538         break;
539     case 8 :
540         array.add(2);
541         array.add(4);
542         break;
543     case 10 :
544         array.add(2);
545         array.add(5);
546         break;
547     case 15 :
548         array.add(3);
549         array.add(5);
550         break;
551     case 18 :
552         array.add(3);
553         array.add(6);
554         break;
555     case 20 :
556         array.add(4);
557         array.add(5);
558         break;
559     case 24 :
560         array.add(4);
561         array.add(6);
562         break;
563     case 30 :
564         array.add(5);
565         array.add(6);
566         break;
567     default :
568         array = createRetainAllArray();
569         break;
570     }
571     break;
572 case '/':
573     switch (cageTargetNumber)
574     {
575         case 4 :
576             array.add(1);
577             array.add(4);
578             break;
579         case 5 :
580             array.add(1);
581             array.add(5);
582             break;
583         case 6 :
584             array.add(1);
585             array.add(6);
586             break;
587         default :
588             array = createRetainAllArray();
589             break;
590     }
591     break;
592 default :
593     JOptionPane.showMessageDialog(null,
594         "Invalid operator.", "Error",
595         JOptionPane.ERROR_MESSAGE);
596     throw new IllegalStateException("Invalid operator.");
597 }
598 return array;
599 }
600
private ArrayList<Integer> killerCombinationCageSize2GridSize7(Cage cage)
{
    char cageOperator = cage.getOperator();
    int cageTargetNumber = cage.getTargetNumber();
    ArrayList<Integer> array = new ArrayList();
    switch (cageOperator)
    {
        case '+':
            switch (cageTargetNumber)
            {
                case 3 :
                    array.add(1);
                    array.add(2);
                    break;
                case 4 :
                    array.add(1);
                    array.add(3);
                    break;
                case 12 :
                    array.add(5);
                    array.add(7);
                    break;
                case 13 :
                    array.add(6);
                    array.add(7);
                    break;
            }
        }
    }
}

```

```

627     default :
628         array = createRetainAllArray();
629         break;
630     }
631     break;
632 case '-':
633     switch (cageTargetNumber)
634     {
635         case 6 :
636             array.add(1);
637             array.add(7);
638             break;
639         default :
640             array = createRetainAllArray();
641             break;
642     }
643     break;
644 case '*':
645     switch (cageTargetNumber)
646     {
647         case 2 :
648             array.add(1);
649             array.add(2);
650             break;
651         case 3 :
652             array.add(1);
653             array.add(3);
654             break;
655         case 4 :
656             array.add(1);
657             array.add(4);
658             break;
659         case 5 :
660             array.add(1);
661             array.add(5);
662             break;
663         case 7 :
664             array.add(1);
665             array.add(7);
666             break;
667         case 8 :
668             array.add(2);
669             array.add(4);
670             break;
671         case 10 :
672             array.add(2);
673             array.add(5);
674             break;
675         case 14 :
676             array.add(2);
677             array.add(7);
678             break;
679         case 15 :
680             array.add(3);
681             array.add(5);
682             break;
683         case 18 :
684             array.add(3);
685             array.add(6);
686             break;
687         case 20 :
688             array.add(4);
689             array.add(5);
690             break;
691         case 21 :
692             array.add(3);
693             array.add(7);
694             break;
695         case 24 :
696             array.add(4);
697             array.add(6);
698             break;
699         case 28 :
700             array.add(4);
701             array.add(7);
702             break;
703         case 30 :
704             array.add(5);
705             array.add(6);
706             break;
707         case 35 :
708             array.add(5);
709             array.add(7);
710             break;
711         case 42 :
712             array.add(6);
713             array.add(7);
714             break;
715         default :
716             array = createRetainAllArray();
717             break;
718     }
719     break;
720 case '/':
721     switch (cageTargetNumber)
722     {
723         case 4 :
724             array.add(1);
725             array.add(4);

```

```

726
727         break;
728     case 5 :
729         array.add(1);
730         array.add(5);
731         break;
732     case 6 :
733         array.add(1);
734         array.add(6);
735         break;
736     case 7 :
737         array.add(1);
738         array.add(7);
739         break;
740     default :
741         array = createRetainAllArray();
742         break;
743     }
744     break;
745     default :
746         JOptionPane.showMessageDialog(null,
747             "Invalid operator.", "Error",
748             JOptionPane.ERROR_MESSAGE);
749         throw new IllegalStateException("Invalid operator.");
750     }
751 }
752
753 private ArrayList<Integer> killerCombinationCageSize2GridSize8(Cage cage)
754 {
755     char cageOperator = cage.getOperator();
756     int cageTargetNumber = cage.getTargetNumber();
757     ArrayList<Integer> array = new ArrayList();
758     switch (cageOperator)
759     {
760         case '+':
761             switch (cageTargetNumber)
762             {
763                 case 3:
764                     array.add(1);
765                     array.add(2);
766                     break;
767                 case 4:
768                     array.add(1);
769                     array.add(3);
770                     break;
771                 case 14:
772                     array.add(6);
773                     array.add(8);
774                     break;
775                 case 15:
776                     array.add(7);
777                     array.add(8);
778                     break;
779                 default:
780                     array = createRetainAllArray();
781                     break;
782             }
783             break;
784         case '-':
785             switch (cageTargetNumber)
786             {
787                 case 7:
788                     array.add(1);
789                     array.add(8);
790                     break;
791                 default:
792                     array = createRetainAllArray();
793                     break;
794             }
795             break;
796         case '*':
797             switch (cageTargetNumber)
798             {
799                 case 2:
800                     array.add(1);
801                     array.add(2);
802                     break;
803                 case 3:
804                     array.add(1);
805                     array.add(3);
806                     break;
807                 case 4:
808                     array.add(1);
809                     array.add(4);
810                     break;
811                 case 5:
812                     array.add(1);
813                     array.add(5);
814                     break;
815                 case 7:
816                     array.add(1);
817                     array.add(7);
818                     break;
819                 case 10:
820                     array.add(2);
821                     array.add(5);
822                     break;
823                 case 14:
824                     array.add(2);

```

```

825|             array.add(7);
826|         break;
827|     case 15 :
828|         array.add(3);
829|         array.add(5);
830|         break;
831|     case 16 :
832|         array.add(2);
833|         array.add(8);
834|         break;
835|     case 18 :
836|         array.add(3);
837|         array.add(6);
838|         break;
839|     case 20 :
840|         array.add(4);
841|         array.add(5);
842|         break;
843|     case 21 :
844|         array.add(3);
845|         array.add(7);
846|         break;
847|     case 28 :
848|         array.add(4);
849|         array.add(7);
850|         break;
851|     case 30 :
852|         array.add(5);
853|         array.add(6);
854|         break;
855|     case 32 :
856|         array.add(4);
857|         array.add(8);
858|         break;
859|     case 35 :
860|         array.add(5);
861|         array.add(7);
862|         break;
863|     case 40 :
864|         array.add(5);
865|         array.add(8);
866|         break;
867|     case 42 :
868|         array.add(6);
869|         array.add(7);
870|         break;
871|     case 48 :
872|         array.add(6);
873|         array.add(8);
874|         break;
875|     case 56 :
876|         array.add(7);
877|         array.add(8);
878|         break;
879|     default :
880|         array = createRetainAllArray();
881|         break;
882|     }
883|     break;
884| case '/':
885|     switch (cageTargetNumber)
886|     {
887|         case 5 :
888|             array.add(1);
889|             array.add(5);
890|             break;
891|         case 6 :
892|             array.add(1);
893|             array.add(6);
894|             break;
895|         case 7 :
896|             array.add(1);
897|             array.add(7);
898|             break;
899|         case 8 :
900|             array.add(1);
901|             array.add(8);
902|             break;
903|         default :
904|             array = createRetainAllArray();
905|             break;
906|     }
907|     break;
908|     default :
909|         JOptionPane.showMessageDialog(null,
910|             "Invalid operator.", "Error",
911|             JOptionPane.ERROR_MESSAGE);
912|         throw new IllegalStateException("Invalid operator.");
913|     }
914|     return array;
915| }
916|
917| private ArrayList<Integer> killerCombinationCageSize2GridSize9(Cage cage)
918| {
919|     char cageOperator = cage.getOperator();
920|     int cageTargetNumber = cage.getTargetNumber();
921|     ArrayList<Integer> array = new ArrayList();
922|     switch (cageOperator)
923|     {

```

```

924     case '+' :
925         switch (cageTargetNumber)
926         {
927             case 3 :
928                 array.add(1);
929                 array.add(2);
930                 break;
931             case 4 :
932                 array.add(1);
933                 array.add(3);
934                 break;
935             case 16 :
936                 array.add(7);
937                 array.add(9);
938                 break;
939             case 17 :
940                 array.add(8);
941                 array.add(9);
942                 break;
943             default :
944                 array = createRetainAllArray();
945                 break;
946         }
947         break;
948     case '-' :
949         switch (cageTargetNumber)
950         {
951             case 8 :
952                 array.add(1);
953                 array.add(9);
954                 break;
955             default :
956                 array = createRetainAllArray();
957                 break;
958         }
959         break;
960     case '*' :
961         switch (cageTargetNumber)
962         {
963             case 2 :
964                 array.add(1);
965                 array.add(2);
966                 break;
967             case 3 :
968                 array.add(1);
969                 array.add(3);
970                 break;
971             case 4 :
972                 array.add(1);
973                 array.add(4);
974                 break;
975             case 5 :
976                 array.add(1);
977                 array.add(5);
978                 break;
979             case 7 :
980                 array.add(1);
981                 array.add(7);
982                 break;
983             case 9 :
984                 array.add(1);
985                 array.add(9);
986                 break;
987             case 10 :
988                 array.add(2);
989                 array.add(5);
990                 break;
991             case 14 :
992                 array.add(2);
993                 array.add(7);
994                 break;
995             case 15 :
996                 array.add(3);
997                 array.add(5);
998                 break;
999             case 16 :
1000                 array.add(2);
1001                 array.add(8);
1002                 break;
1003             case 20 :
1004                 array.add(4);
1005                 array.add(5);
1006                 break;
1007             case 21 :
1008                 array.add(3);
1009                 array.add(7);
1010                 break;
1011             case 27 :
1012                 array.add(3);
1013                 array.add(9);
1014                 break;
1015             case 28 :
1016                 array.add(4);
1017                 array.add(7);
1018                 break;
1019             case 30 :
1020                 array.add(5);
1021                 array.add(6);
1022                 break;

```

```

1023         case 32 :
1024             array.add(4);
1025             array.add(8);
1026             break;
1027         case 35 :
1028             array.add(5);
1029             array.add(7);
1030             break;
1031         case 36 :
1032             array.add(4);
1033             array.add(9);
1034             break;
1035         case 40 :
1036             array.add(5);
1037             array.add(8);
1038             break;
1039         case 42 :
1040             array.add(6);
1041             array.add(7);
1042             break;
1043         case 45 :
1044             array.add(5);
1045             array.add(9);
1046             break;
1047         case 48 :
1048             array.add(6);
1049             array.add(8);
1050             break;
1051         case 54 :
1052             array.add(6);
1053             array.add(9);
1054             break;
1055         case 56 :
1056             array.add(7);
1057             array.add(8);
1058             break;
1059         case 63 :
1060             array.add(7);
1061             array.add(9);
1062             break;
1063         case 72 :
1064             array.add(8);
1065             array.add(9);
1066             break;
1067         default :
1068             array = createRetainAllArray();
1069             break;
1070     }
1071     break;
1072     case '/' :
1073     switch (cageTargetNumber)
1074     {
1075         case 5 :
1076             array.add(1);
1077             array.add(5);
1078             break;
1079         case 6 :
1080             array.add(1);
1081             array.add(6);
1082             break;
1083         case 7 :
1084             array.add(1);
1085             array.add(7);
1086             break;
1087         case 8 :
1088             array.add(1);
1089             array.add(8);
1090             break;
1091         case 9 :
1092             array.add(1);
1093             array.add(9);
1094             break;
1095         default :
1096             array = createRetainAllArray();
1097             break;
1098     }
1099     break;
1100     default :
1101         JOptionPane.showMessageDialog(null,
1102             "Invalid operator.", "Error",
1103             JOptionPane.ERROR_MESSAGE);
1104         throw new IllegalStateException("Invalid operator.");
1105     }
1106     return array;
1107 }
1108
1109 private void nakedSingle()
1110 {
1111     nakedSingleRow();
1112     nakedSingleColumn();
1113 }
1114
1115 private void nakedSingleRow()
1116 {
1117     for (int i = 0; i < size; i++)
1118     {
1119         nakedSingleRow(i);
1120     }
1121 }
```



```

1221                     column2PossibleIndexes.get(j));
1222                 }
1223             }
1224         nakedDoubleRow(row, doublePossibleValues,
1225                         doublePossibleIndexes);
1226     }
1227 }
1228 }
1229 }
1230
1231 private void nakedDoubleRow(int row,
1232     ArrayList<Integer> doublePossibleValues,
1233     ArrayList<Integer> doublePossibleIndexes)
1234 {
1235     for (int i = 0; i < size; i++)
1236     {
1237         if (!doublePossibleIndexes.contains(i))
1238         {
1239             possibleValues[row][i].removeAll(doublePossibleValues);
1240         }
1241     }
1242 }
1243
1244 private void nakedDoubleColumn()
1245 {
1246     for (int i = 0; i < size; i++)
1247     {
1248         nakedDoubleColumn(i);
1249     }
1250 }
1251
1252 private void nakedDoubleColumn(int column)
1253 {
1254     ArrayList<Integer>[] columnPossibleValues =
1255         getColumnPossibleValues(column);
1256     ArrayList<Integer> row2PossibleIndexes = new ArrayList();
1257     ArrayList<ArrayList<Integer>> row2PossibleValues = new ArrayList();
1258     ArrayList<ArrayList<Integer>> uniquePossibleValues = new ArrayList();
1259     ArrayList<Integer> uniquePossibleValuesFrequency = new ArrayList();
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();
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();
1340         ArrayList<Integer> columnIndexes = new ArrayList();
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();
1391         ArrayList<Integer> rowIndexes = new ArrayList();
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
1428 private void hiddenSingle(int row, int column, int value)
1429 {
1430     setCellValue(row, column, value);
1431 }
1432
1433 private void setCellValue(int row, int column, int value)
1434 {
1435     grid.solverSetCellValue(row, column, value);
1436     removePossibleValues(row, column, value);
1437 }
1438
1439 private void removePossibleValues(int row, int column, int value)
1440 {
1441     possibleValues[row][column].clear();
1442     for (int i = 0; i < size; i++)
1443     {
1444         if (possibleValues[i][column].contains(value))
1445         {
1446             possibleValues[i][column].remove(
1447                 possibleValues[i][column].indexOf(value));
1448         }
1449     }
1450     for (int i = 0; i < size; i++)
1451     {
1452         if (possibleValues[row][i].contains(value))
1453         {
1454             possibleValues[row][i].remove(
1455                 possibleValues[row][i].indexOf(value));
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();
1479     for (int i = 1; i <= size; i++)
1480     {
1481         array.add(i);
1482     }
1483     return array;
1484 }
1485
1486 private 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();
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 ArrayList<Integer>[][] getPossibleValues()
1512 {
1513     return possibleValues;
1514 }
1515
1516 private void printGrid()
1517 {

```

```

1518     Cell [][] cells = grid.getGridContents();
1519     for (int i = 0; i < size; i++)
1520     {
1521         for (int j = 0; j < size; j++)
1522         {
1523             System.out.print(cells[i][j].getValue() + " ");
1524         }
1525         System.out.println("");
1526     }
1527     System.out.println("");
1528 }
1529
1530 private void printPossibleValues()
1531 {
1532     for (int i = 0; i < possibleValues.length; i++)
1533     {
1534         for (int j = 0; j < possibleValues[i].length; j++)
1535         {
1536             System.out.println("Row " + i + ", Column " + j);
1537             for (int k = 0; k < possibleValues[i][j].size(); k++)
1538             {
1539                 System.out.print(possibleValues[i][j].get(k) + " ");
1540             }
1541             System.out.println("");
1542         }
1543         System.out.println("");
1544     }
1545     System.out.println("");
1546 }
1547
1548 }
```

Listing D.7: SolverGenetic.java

```

1 package model;
2
3 import java.util.ArrayList;
4 import javax.swing.JOptionPane;
5
6 /**
7 *
8 * @author michaeladrian39
9 */
10 public class Cage
11 {
12
13     private final int cageID;
14     private final String objective;
15     private final int targetNumber;
16     private final char operator;
17     private final ArrayList<Cell> cells;
18
19     public Cage(int cageID, String objective)
20     {
21         this.cageID = cageID;
22         if (isCageObjectiveValid(objective))
23         {
24             this.objective = objective;
25         }
26         else
27         {
28             JOptionPane.showMessageDialog(null, "Invalid cage objectives.",
29                                         "Error", JOptionPane.ERROR_MESSAGE);
30             throw new IllegalStateException("Invalid cage objectives.");
31         }
32         this.targetNumber = generateTargetNumber(this.objective);
33         this.operator = generateOperator(this.objective);
34         this.cells = new ArrayList<>();
35     }
36
37     private boolean isCageObjectiveValid(String cageObjective)
38     {
39         return cageObjective.matches("\\d+[+-/=]");
40     }
41
42     private int generateTargetNumber(String objective)
43     {
44         return Integer.parseInt(objective.substring(0,
45             objective.length() - 1));
46     }
47
48     private char generateOperator(String objective)
49     {
50         return objective.charAt(objective.length() - 1);
51     }
52
53     public void addCell(Cell c)
54     {
55         cells.add(c);
56     }
57
58     public boolean isCageContainsNull()
59     {
60         for (int i = 0; i < cells.size(); i++)
61         {
62             if (cells.get(i).getValue() == null)
63             {
64                 return true;
65             }
66         }
67     }
68 }
```

```

65        }
66    }
67    return false;
68 }
69
70 public Boolean isCageValuesValid()
71 {
72     if (countValue() == null)
73     {
74         return null;
75     }
76     else
77     {
78         return (countValue() == getTargetNumber());
79     }
80 }
81
82 private Integer countValue()
83 {
84     Integer value = null;
85     if (isCageContainsNull() == true)
86     {
87         value = null;
88     }
89     else
90     {
91         switch (getOperator())
92         {
93             case '+':
94                 value = 0;
95                 for (int i = 0; i < cells.size(); i++)
96                 {
97                     value += cells.get(i).getValue();
98                 }
99                 break;
100            case '-':
101                if (cells.get(0).getValue() > cells.get(1).getValue())
102                {
103                    value = cells.get(0).getValue()
104                        - cells.get(1).getValue();
105                }
106                else if (cells.get(1).getValue() > cells.get(0).getValue())
107                {
108                    value = cells.get(1).getValue()
109                        - cells.get(0).getValue();
110                }
111            else
112            {
113                value = 0;
114            }
115            break;
116            case '*':
117                value = 1;
118                for (int i = 0; i < cells.size(); i++)
119                {
120                    value *= cells.get(i).getValue();
121                }
122            break;
123            case '/':
124                if (cells.get(0).getValue() > cells.get(1).getValue())
125                {
126                    if (cells.get(0).getValue()
127                        % cells.get(1).getValue() == 0)
128                    {
129                        value = cells.get(0).getValue()
130                            / cells.get(1).getValue();
131                    }
132                }
133                else
134                {
135                    value = 0;
136                }
137            else if (cells.get(1).getValue() > cells.get(0).getValue())
138            {
139                if (cells.get(1).getValue()
140                    % cells.get(0).getValue() == 0)
141                {
142                    value = cells.get(1).getValue()
143                        / cells.get(0).getValue();
144                }
145            }
146            else
147            {
148                value = 0;
149            }
150        }
151    }
152    value = 0;
153    break;
154    case '=':
155        value = cells.get(0).getValue();
156        break;
157    default :
158        value = null;
159    }
160 }
161 return value;
162 }
163

```

```

164
165     public String getObjective()
166     {
167         return objective;
168     }
169
170     public int getTargetNumber()
171     {
172         return targetNumber;
173     }
174
175     public char getOperator()
176     {
177         return operator;
178     }
179
180     public ArrayList<Cell> getCells()
181     {
182         return cells;
183     }
184
185     public int getSize()
186     {
187         return cells.size();
188     }
189 }
190 }
```

Listing D.8: Chromosome.java

```

1 package model;
2
3 import java.util.Comparator;
4
5 /**
6 * 
7 * @author michaeladrian39
8 */
9 public class Chromosome
10 {
11
12     private final Grid grid;
13     private final int size;
14     private final double fitness;
15
16     public Chromosome(Grid grid)
17     {
18         this.grid = grid;
19         this.size = grid.getSize();
20         this.fitness = setFitness();
21     }
22
23     private double setFitness()
24     {
25         double numberValidCells = 0;
26         double numberCells = size * size;
27         for (int i = 0; i < size; i++)
28         {
29             for (int j = 0; j < size; j++)
30             {
31                 if (grid.solverIsCellValueValid(i, j) == true)
32                 {
33                     numberValidCells++;
34                 }
35             }
36         }
37         double value = numberValidCells / numberCells;
38         return value;
39     }
40
41     public double getFitness()
42     {
43         return fitness;
44     }
45
46     public Grid getGrid()
47     {
48         return grid;
49     }
50
51     private void printGrid()
52     {
53         Cell [][] cells = grid.getGridContents();
54         for (int i = 0; i < size; i++)
55         {
56             for (int j = 0; j < size; j++)
57             {
58                 System.out.print(cells[i][j].getValue() + " ");
59             }
60             System.out.println("");
61         }
62         System.out.println("");
63     }
64
65 }
66
67 class ChromosomeComparator implements Comparator<Chromosome>
68 {
```

```

69
70     @Override
71     public int compare(Chromosome c1, Chromosome c2)
72     {
73         if (c1.getFitness() - c2.getFitness() > 0)
74         {
75             return 1;
76         }
77         else if (c1.getFitness() - c2.getFitness() < 0)
78         {
79             return -1;
80         }
81         else
82         {
83             return 0;
84         }
85     }
86 }
87 }
```

Listing D.9: Controller.java

```

1 package controller;
2
3 import model.Grid;
4 import model.Cage;
5 import model.Cell;
6
7 /**
8 * 
9 * @author michaeladrian39
10 */
11 public class Controller
12 {
13
14     private Integer size;
15     private Integer numberOfCages;
16     private Integer [][] cageCells;
17     private String [] cageObjectives;
18     private Grid g;
19
20     public Controller(Integer size, Integer numberOfCages,
21                     Integer [][] cageCells, String [] cageObjectives)
22     {
23         this.size = size;
24         this.numberOfCages = numberOfCages;
25         this.cageCells = cageCells;
26         this.cageObjectives = cageObjectives;
27         g = new Grid(size, numberOfCages, cageCells, cageObjectives);
28     }
29
30     public boolean setCellValue(int row, int column, int value)
31     {
32         return g.setCellValue(row, column, value);
33     }
34
35     public void unsetCellValue(int row, int column)
36     {
37         g.unsetCellValue(row, column);
38     }
39
40     public Boolean checkGrid()
41     {
42         return g.checkGrid();
43     }
44
45     public Integer getCellValue(int row, int column)
46     {
47         return g.getCellValue(row, column);
48     }
49
50     public Integer getSize()
51     {
52         return g.getSize();
53     }
54
55     public Integer getNumberOfCages()
56     {
57         return g.getNumberOfCages();
58     }
59
60     public Integer [][] getCageCells()
61     {
62         return g.getCageCells();
63     }
64
65     public String [] getCageObjectives()
66     {
67         return g.getCageObjectives();
68     }
69
70     public int getCageTargetNumber(int cageID)
71     {
72         return g.getCages () [cageID].getTargetNumber();
73     }
74
75     public char getCageOperator(int cageID)
76     {
```

```

77     return g.getCages()[cageID].getOperator();
78 }
79
80 public Cell[][] getGrid()
81 {
82     return g.getGridContents();
83 }
84
85 public Cage[] getCages()
86 {
87     return g.getCages();
88 }
89
90 public Grid getGame()
91 {
92     return g.getGame();
93 }
94
95 }

```

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 /**
22 *
23 * @author michaeladrian39
24 */
25 public class Calcudoku extends JFrame
26 {
27
28     private File puzzleFile;
29     private Integer size;
30     private Integer numberCages;
31     private Integer[][] cageCells;
32     private String[] cageObjectives;
33     private Controller c;
34     private final JMenuBar menuBar;
35     private final JMenu menuFile;
36     private final JMenu menuSolve;
37     private final JMenuItem menuItemLoad;
38     private final JMenuItem menuItemReset;
39     private final JMenuItem menuItemClose;
40     private final JMenuItem menuItemCheck;
41     private final JMenuItem menuItemExit;
42     private final JMenuItem menuItemBacktracking;
43     private final JMenuItem menuItemHybridGenetic;
44     private final JMenuItem menuItemGeneticParameters;
45     private final JFileChooser fileChooser;
46     private GUI gui;
47
48     public Calcudoku()
49     {
50         this.menuBar = new JMenuBar();
51         this.menuFile = new JMenu();
52         this.menuSolve = new JMenu();
53         this.menuItemLoad = new JMenuItem();
54         this.menuItemReset = new JMenuItem();
55         this.menuItemClose = new JMenuItem();
56         this.menuItemCheck = new JMenuItem();
57         this.menuItemExit = new JMenuItem();
58         this.menuItemBacktracking = new JMenuItem();
59         this.menuItemHybridGenetic = new JMenuItem();
60         this.menuItemGeneticParameters = new JMenuItem();
61         this.fileChooser = new JFileChooser();
62         initComponents();
63     }
64
65     public static void main(String args[])
66     {
67         EventQueue.invokeLater(() ->
68         {
69             new Calcudoku().setVisible(true);
70         });
71     }
72
73     private void initComponents()
74     {
75         this.setTitle("Calcudoku");
76         this.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

```

```

77     this.setMinimumSize(new Dimension(216, 216));
78     this.setLocationRelativeTo(null);
79     this.setResizable(false);
80     initWindowListener();
81     initActionListeners();
82     initMenu();
83     initMenuBar();
84     this.validate();
85     this.revalidate();
86     this.pack();
87 }
88
89 private void initWindowListener()
90 {
91     this.addWindowListener(new WindowListener(this));
92 }
93
94 private void initActionListeners()
95 {
96     this.fileChooser.setFileFilter(new PuzzleFileFilter());
97     this.menuItemLoad.addActionListener(this::menuItemLoadActionPerformed);
98     this.menuItemReset.addActionListener(
99         this::menuItemResetActionPerformed);
100    this.menuItemCheck.addActionListener(
101        this::menuItemCheckActionPerformed);
102    this.menuItemClose.addActionListener(
103        this::menuItemCloseActionPerformed);
104    this.menuItemExit.addActionListener(this::menuItemExitActionPerformed);
105    this.menuItemBacktracking.addActionListener(
106        this::menuItemBacktrackingActionPerformed);
107    this.menuItemHybridGenetic.addActionListener(
108        this::menuItemHybridGeneticActionPerformed);
109    this.menuItemGeneticParameters.addActionListener(
110        this::menuItemGeneticParametersActionPerformed);
111 }
112
113 private void initMenu()
114 {
115     File directory = new File(System.getProperty("user.dir"));
116     this.menuFile.setText("File");
117     this.menuSolve.setText("Solve");
118     this.menuItemLoad.setText("Load\u2022Puzzle\u2022File");
119     this.menuItemReset.setText("Reset\u2022Puzzle");
120     this.menuItemClose.setText("Close\u2022Puzzle\u2022File");
121     this.menuItemCheck.setText("Check\u2022Puzzle");
122     this.menuItemExit.setText("Exit");
123     this.menuItemBacktracking.setText("Backtracking");
124     this.menuItemHybridGenetic.setText("Hybrid\u2022Genetic");
125     this.menuItemGeneticParameters.setText(
126         "Set\u2022Genetic\u2022Algorithm\u2022Parameters");
127     this.fileChooser.setDialogTitle("Load\u2022Puzzle\u2022File");
128     fileChooser.setCurrentDirectory(directory);
129 }
130
131 private void initMenuBar()
132 {
133     this.menuFile.add(menuItemLoad);
134     this.menuFile.add(menuItemReset);
135     this.menuFile.add(menuItemClose);
136     this.menuFile.addSeparator();
137     this.menuFile.add(menuItemCheck);
138     this.menuFile.addSeparator();
139     this.menuFile.add(menuItemExit);
140     this.menuBar.add(menuFile);
141     this.menuSolve.add(menuItemBacktracking);
142     this.menuSolve.add(menuItemHybridGenetic);
143     this.menuSolve.addSeparator();
144     this.menuSolve.add(menuItemGeneticParameters);
145     this.menuBar.add(menuSolve);
146     this.setJMenuBar(menuBar);
147 }
148
149 private void menuItemLoadActionPerformed(ActionEvent evt)
150 {
151     if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
152     {
153         if (c != null && gui != null)
154         {
155             if ( JOptionPane.showConfirmDialog(null,
156                 "Are\u2022you\u2022sure\u2022you\u2022want\u2022to\u2022load\u2022another\u2022puzzle\u2022file?",
157                 "Load\u2022Puzzle\u2022File", JOptionPane.YES_NO_OPTION)
158                 == JOptionPane.YES_OPTION)
159             {
160                 selectPuzzleFile();
161             }
162         }
163         else
164         {
165             selectPuzzleFile();
166         }
167     }
168 }
169
170 private void selectPuzzleFile()
171 {
172     this.puzzleFile = fileChooser.getSelectedFile();
173     try
174     {
175         if (puzzleFile.getAbsolutePath().endsWith(".txt"))

```

```

176
177     {
178         try
179         {
180             loadPuzzleFile(puzzleFile);
181         }
182         catch (IllegalStateException ise)
183         {
184             resetFrame();
185             clearVariables();
186             JOptionPane.showMessageDialog(null,
187                 "Error in loading puzzle file .", "Error",
188                 JOptionPane.ERROR_MESSAGE);
189             throw new IllegalStateException("Error in loading "
190                 + "puzzle file .");
191         }
192     }
193     else
194     {
195         resetFrame();
196         clearVariables();
197         JOptionPane.showMessageDialog(null, "Invalid puzzle file .",
198             "Error", JOptionPane.ERROR_MESSAGE);
199         throw new IllegalStateException("Invalid puzzle file .");
200     }
201 }
202 catch (FileNotFoundException fnfe)
203 {
204     resetFrame();
205     clearVariables();
206     JOptionPane.showMessageDialog(null, "Puzzle file not found .",
207         "Error", JOptionPane.ERROR_MESSAGE);
208     throw new IllegalStateException("Puzzle file not found .");
209 }
210
211 private void menuItemResetActionPerformed(ActionEvent evt)
212 {
213     if (c == null || gui == null)
214     {
215         resetFrame();
216         JOptionPane.showMessageDialog(null, "Puzzle file not loaded .",
217             "Error", JOptionPane.ERROR_MESSAGE);
218         throw new IllegalStateException("Puzzle file not loaded .");
219     }
220     else
221     {
222         if (JOptionPane.showConfirmDialog(null,
223             "Are you sure you want to reset this puzzle?", "Reset Puzzle",
224             JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
225         {
226             resetFrame();
227             this.c = new Controller(size, numberOfCages, cageCells,
228                 cageObjectives);
229             this.gui = new GUI(c);
230             this.getContentPane().add(gui);
231             this.validate();
232             this.revalidate();
233             this.pack();
234             this.setLocationRelativeTo(null);
235             this.setTitle("Calcudoku(" + puzzleFile.getName() + ")");
236         }
237     }
238 }
239
240 private void menuItemCheckActionPerformed(ActionEvent evt)
241 {
242     if (c == null || gui == null)
243     {
244         JOptionPane.showMessageDialog(null, "Puzzle file not loaded .",
245             "Error", JOptionPane.ERROR_MESSAGE);
246         throw new IllegalStateException("Puzzle file not loaded .");
247     }
248     else
249     {
250         c.checkGrid();
251     }
252 }
253
254 private void menuItemCloseActionPerformed(ActionEvent evt)
255 {
256     if (c == null || gui == null)
257     {
258         JOptionPane.showMessageDialog(null, "Puzzle file not loaded .",
259             "Error", JOptionPane.ERROR_MESSAGE);
260         throw new IllegalStateException("Puzzle file not loaded .");
261     }
262     else
263     {
264         if (JOptionPane.showConfirmDialog(null,
265             "Are you sure you want to close this puzzle file ?",
266             "Close Puzzle File", JOptionPane.YES_NO_OPTION)
267             == JOptionPane.YES_OPTION)
268         {
269             resetFrame();
270             clearVariables();
271         }
272     }
273 }
274

```

```

275  private void menuItemExitActionPerformed(ActionEvent evt)
276  {
277      if (JOptionPane.showConfirmDialog(null,
278          "Are you sure you want to exit this application?", "Exit",
279          JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
280      {
281          destroyFrame();
282      }
283  }
284
285  private void menuItemBacktrackingActionPerformed(ActionEvent evt)
286  {
287      if (c == null || gui == null)
288      {
289          JOptionPane.showMessageDialog(null, "Puzzle file not loaded.",
290              "Error", JOptionPane.ERROR_MESSAGE);
291          throw new IllegalStateException("Puzzle file not loaded.");
292      }
293      else
294      {
295          gui.solveBacktracking();
296      }
297  }
298
299  private void menuItemHybridGeneticActionPerformed(ActionEvent evt)
300  {
301      if (c == null || gui == null)
302      {
303          JOptionPane.showMessageDialog(null, "Puzzle file not loaded.",
304              "Error", JOptionPane.ERROR_MESSAGE);
305          throw new IllegalStateException("Puzzle file not loaded.");
306      }
307      else
308      {
309          gui.solveHybridGenetic();
310      }
311  }
312
313  private void menuItemGeneticParametersActionPerformed(ActionEvent evt)
314  {
315      if (c == null || gui == null)
316      {
317          JOptionPane.showMessageDialog(null, "Puzzle file not loaded.",
318              "Error", JOptionPane.ERROR_MESSAGE);
319          throw new IllegalStateException("Puzzle file not loaded.");
320      }
321      else
322      {
323          GeneticParameters gp = new GeneticParameters(gui);
324          gp.setVisible(true);
325      }
326  }
327
328  private void loadPuzzleFile(File puzzleFile) throws FileNotFoundException
329  {
330      resetFrame();
331      clearVariables();
332      try
333      {
334          try (Scanner sc = new Scanner(puzzleFile))
335          {
336              this.size = sc.nextInt();
337              this.cageCells = new Integer[size][size];
338              this.numberOfCages = sc.nextInt();
339              this.cageObjectives = new String[numberOfCages];
340              for (int i = 0; i < size; i++)
341              {
342                  for (int j = 0; j < size; j++)
343                  {
344                      this.cageCells[i][j] = sc.nextInt();
345                  }
346              }
347              for (int i = 0; i < numberOfCages; i++)
348              {
349                  this.cageObjectives[i] = sc.next();
350              }
351              if (sc.hasNext())
352              {
353                  JOptionPane.showMessageDialog(null, "Invalid puzzle file.",
354                      "Error", JOptionPane.ERROR_MESSAGE);
355                  throw new IllegalStateException("Invalid puzzle file.");
356              }
357              sc.close();
358          }
359          this.c = new Controller(size, numberOfCages, cageCells,
360              cageObjectives);
361          this.gui = new GUI(c);
362          this.getContentPane().add(gui);
363          this.validate();
364          this.revalidate();
365          this.pack();
366          this.setLocationRelativeTo(null);
367          this.puzzleFile = puzzleFile;
368          this.setTitle("Calcudoku (" + puzzleFile.getName() + ")");
369      }
370      catch (NoSuchElementException nsee)
371      {
372          resetFrame();
373          clearVariables();

```

```

374         JOptionPane.showMessageDialog(null, "Invalid puzzle file.",
375             "Error", JOptionPane.ERROR_MESSAGE);
376         throw new IllegalStateException("Invalid puzzle file.");
377     }
378 }
379
380 private void resetFrame()
381 {
382     this.getContentPane().removeAll();
383     this.c = null;
384     this.setTitle("Calcudoku");
385     this.validate();
386     this.revalidate();
387     this.pack();
388     this.setLocationRelativeTo(null);
389 }
390
391 private void clearVariables()
392 {
393     this.puzzleFile = null;
394     this.size = null;
395     this.cageCells = null;
396     this.numberOfCages = null;
397     this.cageObjectives = null;
398 }
399
400 public void destroyFrame()
401 {
402     resetFrame();
403     clearVariables();
404     this.dispose();
405 }
406
407 }
408
409 class WindowListener extends WindowAdapter
410 {
411     private final Calcudoku frame;
412
413     public WindowListener(Calcudoku frame)
414     {
415         this.frame = frame;
416     }
417
418     @Override
419     public void windowClosing(WindowEvent e)
420     {
421         if (JOptionPane.showConfirmDialog(null,
422             "Are you sure you want to exit the application?", "Exit",
423             JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION)
424         {
425             frame.destroyFrame();
426         }
427     }
428
429     @Override
430     public void windowClosed(WindowEvent e)
431     {
432         System.exit(0);
433     }
434
435 }
436
437
438 class PuzzleFileFilter extends FileFilter
439 {
440
441     @Override
442     public boolean accept(File puzzleFile)
443     {
444         return puzzleFile.isDirectory()
445             || puzzleFile.getAbsolutePath().endsWith(".txt");
446     }
447
448     @Override
449     public String getDescription()
450     {
451         return "Text documents (*.txt)";
452     }
453 }
454 }
```

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| /**
31|  * @author michaeladrian39
32|  */
33| */
34| public class GUI extends JPanel
35| {
36|
37|     private final Controller c;
38|     private final Grid game;
39|     private final Integer size;
40|     private final Integer numberOfCages;
41|     private final Integer[][] cageCells;
42|     private final String[] cageObjectives;
43|     private final Cell[][] grid;
44|     private final Cage[] cages;
45|     private final JTextField[][] textFields;
46|     private final Map<JTextField, Point> textFieldCoordinates;
47|     private final CellTextFieldListener[][] cellTextFieldListeners;
48|     private final Font font;
49|     private final int cellSize;
50|     private final int cellBorderWidth;
51|     private final int cageBorderWidth;
52|     private Integer generations;
53|     private Integer populationSize;
54|     private Double elitismRate;
55|     private Double crossoverRate;
56|     private Double mutationRate;
57|
58|     public GUI(Controller c)
59|     {
60|         this.c = c;
61|         this.game = c.getGame();
62|         this.size = c.getSize();
63|         this.cageCells = c.getCageCells();
64|         this.numberOfCages = c.getNumberOfCages();
65|         this.cageObjectives = c.getCageObjectives();
66|         this.grid = c.getGrid();
67|         this.cages = c.getCages();
68|         this.textFields = new JTextField[size][size];
69|         this.textFieldCoordinates = new HashMap<>();
70|         this.cellTextFieldListeners = new CellTextFieldListener[size][size];
71|         this.font = new Font("Courier-New", Font.CENTER_BASELINE, 36);
72|         this.cellSize = 72;
73|         this.cellBorderWidth = 1;
74|         this.cageBorderWidth = 3;
75|         initComponents();
76|     }
77|
78|     private void initComponents()
79|     {
80|         this.setPreferredSize(new Dimension(cellSize * size, cellSize * size));
81|         this.setLayout(new GridLayout(size, size));
82|         initTextFields();
83|     }
84|
85|     public Controller getController()
86|     {
87|         return c;
88|     }
89|
90|     public Integer getGridSize()
91|     {
92|         return size;
93|     }
94|
95|     public Integer getNumberOfCages()
96|     {
97|         return numberOfCages;
98|     }
99|
100|    public Integer[][] getCageCells()
101|    {
102|        return cageCells;
103|    }
104|
105|    public String[] getCageObjectives()
106|    {
107|        return cageObjectives;
108|    }
109|
110|    public Cell[][] getGrid()
111|    {
112|        return grid;
113|    }

```

```

114
115     public Cage[] getCages()
116     {
117         return cages;
118     }
119
120     public void setGeneticAlgorithmParameters(Integer generations,
121         Integer populationSize, Double elitismRate, Double crossoverRate,
122         Double mutationRate)
123     {
124         this.generations = generations;
125         this.populationSize = populationSize;
126         this.elitismRate = elitismRate;
127         this.crossoverRate = crossoverRate;
128         this.mutationRate = mutationRate;
129     }
130
131     private void initTextFields()
132     {
133         for (int y = 0; y < size; y++)
134         {
135             for (int x = 0; x < size; x++)
136             {
137                 JTextField textField = new JTextField();
138                 textField.addKeyListener(new CellKeyListener(this));
139                 cellTextFieldListeners[y][x] = new CellTextFieldListener(c,
140                     textField, y, x);
141                 textField.getDocument().addDocumentListener(
142                     cellTextFieldListeners[y][x]);
143                 textFieldCoordinates.put(textField, new Point(x, y));
144                 textFields[y][x] = textField;
145             }
146         }
147         for (int y = 0; y < size; y++)
148         {
149             for (int x = 0; x < size; x++)
150             {
151                 JTextField textField = textFields[y][x];
152                 int cageID;
153                 int topBorderWidth;
154                 int leftBorderWidth;
155                 int bottomBorderWidth;
156                 int rightBorderWidth;
157                 if (y == 0)
158                 {
159                     topBorderWidth = cageBorderWidth;
160                     if (Objects.equals(cageCells[y][x], cageCells[y + 1][x]))
161                     {
162                         bottomBorderWidth = cellBorderWidth;
163                     }
164                     else
165                     {
166                         bottomBorderWidth = cageBorderWidth;
167                     }
168                 }
169                 else if (y == size - 1)
170                 {
171                     bottomBorderWidth = cageBorderWidth;
172                     if (Objects.equals(cageCells[y][x], cageCells[y - 1][x]))
173                     {
174                         topBorderWidth = cellBorderWidth;
175                     }
176                     else
177                     {
178                         topBorderWidth = cageBorderWidth;
179                     }
180                 }
181                 else
182                 {
183                     if (Objects.equals(cageCells[y][x], cageCells[y + 1][x]))
184                     {
185                         bottomBorderWidth = cellBorderWidth;
186                     }
187                     else
188                     {
189                         bottomBorderWidth = cageBorderWidth;
190                     }
191                     if (Objects.equals(cageCells[y][x], cageCells[y - 1][x]))
192                     {
193                         topBorderWidth = cellBorderWidth;
194                     }
195                     else
196                     {
197                         topBorderWidth = cageBorderWidth;
198                     }
199                 }
200                 if (x == 0)
201                 {
202                     leftBorderWidth = cageBorderWidth;
203                     if (Objects.equals(cageCells[y][x], cageCells[y][x + 1]))
204                     {
205                         rightBorderWidth = cellBorderWidth;
206                     }
207                     else
208                     {
209                         rightBorderWidth = cageBorderWidth;
210                     }
211                 }
212             else if (x == size - 1)

```

```

213     {
214         rightBorderWidth = cageBorderWidth;
215         if (Objects.equals(cageCells[y][x], cageCells[y][x - 1]))
216         {
217             leftBorderWidth = cellBorderWidth;
218         }
219         else
220         {
221             leftBorderWidth = cageBorderWidth;
222         }
223     }
224     else
225     {
226         if (Objects.equals(cageCells[y][x], cageCells[y][x + 1]))
227         {
228             rightBorderWidth = cellBorderWidth;
229         }
230         else
231         {
232             rightBorderWidth = cageBorderWidth;
233         }
234         if (Objects.equals(cageCells[y][x], cageCells[y][x - 1]))
235         {
236             leftBorderWidth = cellBorderWidth;
237         }
238         else
239         {
240             leftBorderWidth = cageBorderWidth;
241         }
242     }
243     cageID = grid[y][x].getCageID();
244     textField.setToolTipText(cages[cageID].getObjective());
245     textField.setBorder(BorderFactory.createMatteBorder(
246         topBorderWidth, leftBorderWidth, bottomBorderWidth,
247         rightBorderWidth, Color.BLACK));
248     textField.setFont(font);
249     textField.setHorizontalAlignment(JTextField.CENTER);
250     textField.setPreferredSize(new Dimension(cellSize, cellSize));
251     JPopupMenu popupMenu = new JPopupMenu();
252     for (int i = 1; i <= size; i++)
253     {
254         JMenuItem menuItem = new JMenuItem("" + i);
255         menuItem.addActionListener(new PopupMenuItemListener(textField,
256             i));
257     }
258     textField.add(popupMenu);
259     textField.setComponentPopupMenu(popupMenu);
260 }
261 }
262 for (int y = 0; y < size; y++)
263 {
264     for (int x = 0; x < size; x++)
265     {
266         this.add(textFields[y][x]);
267     }
268 }
269 }
270 }
271
272 public void solveBacktracking()
273 {
274     removeCellTextFieldListeners();
275     clearGrid();
276     Cell[][] solution;
277     float startTime = System.nanoTime();
278     float endTime;
279     float duration;
280     SolverBacktracking sb = new SolverBacktracking(game);
281     if (sb.solve() == true)
282     {
283         solution = sb.getSolution().getGridContents();
284         printGridToScreen(solution);
285         endTime = System.nanoTime();
286         duration = (endTime - startTime) / 1000000000;
287         System.out.println("The backtracking algorithm has successfully "
288             + "solved the puzzle." + "\nTime elapsed: " + duration
289             + " seconds");
290         JOptionPane.showMessageDialog(null,
291             "The backtracking algorithm has successfully solved the "
292             + "puzzle." + "\nTime elapsed: " + duration
293             + " seconds", "Information",
294             JOptionPane.INFORMATION_MESSAGE);
295     }
296     else
297     {
298         endTime = System.nanoTime();
299         duration = (endTime - startTime) / 1000000000;
300         System.out.println("The backtracking algorithm has failed to solve "
301             + "the puzzle." + "\nTime elapsed: " + duration
302             + " seconds");
303         JOptionPane.showMessageDialog(null,
304             "The backtracking algorithm has failed to solve the "
305             + "puzzle." + "\nTime elapsed: " + duration
306             + " seconds", "Information",
307             JOptionPane.INFORMATION_MESSAGE);
308     }
309     addCellTextFieldListeners();
310 }
311

```

```

312     public void solveHybridGenetic()
313     {
314         if (generations == null || populationSize == null
315             || elitismRate == null || crossoverRate == null
316             || mutationRate == null)
317         {
318             JOptionPane.showMessageDialog(null,
319                 "Genetic\u00d7algorithm\u00d7parameters\u00d7have\u00d7not\u00d7been\u00d7set.\",
320                 "Error", JOptionPane.ERROR_MESSAGE);
321             throw new IllegalStateException(
322                 "Genetic\u00d7algorithm\u00d7parameters\u00d7have\u00d7not\u00d7been\u00d7set.");
323         }
324     else
325     {
326         removeCellTextFieldListeners();
327         clearGrid();
328         Cell[][] solution;
329         float startTime = System.nanoTime();
330         float endTime;
331         float duration;
332         SolverHybridGenetic shg = new SolverHybridGenetic(game, generations,
333                 populationSize, elitismRate, crossoverRate, mutationRate);
334         if (shg.solve() == true)
335         {
336             solution = shg.getSolution().getGridContents();
337             printGridToScreen(solution);
338             endTime = System.nanoTime();
339             duration = (endTime - startTime) / 1000000000;
340             System.out.println("The\u00d7hybrid\u00d7genetic\u00d7algorithm\u00d7has\u00d7successfully\u00d7
341                 + "solved\u00d7the\u00d7puzzle." + "\nTime\u00d7elapsed:\u00d7" + duration
342                 + "\u00d7seconds");
343             JOptionPane.showMessageDialog(null,
344                 "The\u00d7hybrid\u00d7genetic\u00d7algorithm\u00d7has\u00d7successfully\u00d7solved\u00d7the\u00d7
345                 + "puzzle." + "\nTime\u00d7elapsed:\u00d7" + duration
346                 + "\u00d7seconds", "Information",
347                 JOptionPane.INFORMATION_MESSAGE);
348         }
349     else
350     {
351         endTime = System.nanoTime();
352         duration = (endTime - startTime) / 1000000000;
353         System.out.println("The\u00d7hybrid\u00d7genetic\u00d7algorithm\u00d7has\u00d7failed\u00d7to\u00d7
354                 + "solve\u00d7the\u00d7puzzle." + "\nTime\u00d7elapsed:\u00d7" + duration
355                 + "\u00d7seconds");
356         JOptionPane.showMessageDialog(null,
357                 "The\u00d7hybrid\u00d7genetic\u00d7algorithm\u00d7has\u00d7failed\u00d7to\u00d7solve\u00d7the\u00d7 +
358                 "puzzle." + "\nTime\u00d7elapsed:\u00d7" + duration
359                 + "\u00d7seconds", "Information",
360                 JOptionPane.INFORMATION_MESSAGE);
361     }
362     addCellTextFieldListeners();
363   }
364 }
365
366 private void printGridToScreen(Cell[][] grid)
367 {
368     for (int x = 0; x < size; x++)
369     {
370         for (int y = 0; y < size; y++)
371         {
372             String value = Integer.toString(grid[x][y].getValue());
373             textFields[x][y].setText(value);
374         }
375     }
376 }
377
378 private void clearGrid()
379 {
380     for (int i = 0; i < size; i++)
381     {
382         for (int j = 0; j < size; j++)
383         {
384             c.unsetCellValue(i, j);
385         }
386     }
387 }
388
389 private void addCellTextFieldListeners()
390 {
391     for (int x = 0; x < size; x++)
392     {
393         for (int y = 0; y < size; y++)
394         {
395             textFields[x][y].getDocument().addDocumentListener(
396                 cellTextFieldListeners[x][y]);
397         }
398     }
399 }
400
401 private void removeCellTextFieldListeners()
402 {
403     for (int x = 0; x < size; x++)
404     {
405         for (int y = 0; y < size; y++)
406         {
407             textFields[x][y].getDocument().removeDocumentListener(
408                 cellTextFieldListeners[x][y]);
409         }
410     }
}

```

```

411     }
412 
413     public void moveCursor(JTextField textField, int keyCode)
414     {
415         Point coordinates = textFieldCoordinates.get(textField);
416         switch (keyCode)
417         {
418             case KeyEvent.VK_LEFT:
419                 if (coordinates.x > 0)
420                 {
421                     textFields[coordinates.y][coordinates.x - 1].requestFocus();
422                 }
423                 break;
424             case KeyEvent.VK_KP_LEFT:
425                 if (coordinates.x > 0)
426                 {
427                     textFields[coordinates.y][coordinates.x - 1].requestFocus();
428                 }
429                 break;
430             case KeyEvent.VK_UP:
431                 if (coordinates.y > 0)
432                 {
433                     textFields[coordinates.y - 1][coordinates.x].requestFocus();
434                 }
435                 break;
436             case KeyEvent.VK_KP_UP:
437                 if (coordinates.y > 0)
438                 {
439                     textFields[coordinates.y - 1][coordinates.x].requestFocus();
440                 }
441                 break;
442             case KeyEvent.VK_RIGHT:
443                 if (coordinates.x < size - 1)
444                 {
445                     textFields[coordinates.y][coordinates.x + 1].requestFocus();
446                 }
447                 break;
448             case KeyEvent.VK_KP_RIGHT:
449                 if (coordinates.x < size - 1)
450                 {
451                     textFields[coordinates.y][coordinates.x + 1].requestFocus();
452                 }
453                 break;
454             case KeyEvent.VK_DOWN:
455                 if (coordinates.y < size - 1)
456                 {
457                     textFields[coordinates.y + 1][coordinates.x].requestFocus();
458                 }
459                 break;
460             case KeyEvent.VK_KP_DOWN:
461                 if (coordinates.y < size - 1)
462                 {
463                     textFields[coordinates.y + 1][coordinates.x].requestFocus();
464                 }
465                 break;
466         }
467     }
468 }
469 }
470 
471 class CellKeyListener implements KeyListener
472 {
473 
474     private final GUI gui;
475 
476     CellKeyListener(GUI gui)
477     {
478         this.gui = gui;
479     }
480 
481     @Override
482     public void keyPressed(KeyEvent e)
483     {
484         int keyCode = e.getKeyCode();
485         JTextField textField = (JTextField) e.getSource();
486         switch (keyCode)
487         {
488             case KeyEvent.VK_LEFT :
489                 e.consume();
490                 gui.moveCursor(textField, KeyEvent.VK_LEFT);
491                 break;
492             case KeyEvent.VK_UP :
493                 e.consume();
494                 gui.moveCursor(textField, KeyEvent.VK_UP);
495                 break;
496             case KeyEvent.VK_RIGHT :
497                 e.consume();
498                 gui.moveCursor(textField, KeyEvent.VK_RIGHT);
499                 break;
500             case KeyEvent.VK_DOWN :
501                 e.consume();
502                 gui.moveCursor(textField, KeyEvent.VK_DOWN);
503                 break;
504             case KeyEvent.VK_KP_LEFT :
505                 e.consume();
506                 gui.moveCursor(textField, KeyEvent.VK_KP_LEFT);
507                 break;
508             case KeyEvent.VK_KP_UP :
509                 e.consume();

```

```

510         gui.moveCursor(textField, KeyEvent.VK_KP_UP);
511         break;
512     case KeyEvent.VK_KP_RIGHT :
513         e.consume();
514         gui.moveCursor(textField, KeyEvent.VK_KP_RIGHT);
515         break;
516     case KeyEvent.VK_KP_DOWN :
517         e.consume();
518         gui.moveCursor(textField, KeyEvent.VK_KP_DOWN);
519         break;
520     }
521 }
522
523 @Override
524 public void keyTyped(KeyEvent e)
525 {
526     JTextField textField = (JTextField) e.getSource();
527     char c = e.getKeyChar();
528     if (!Character.isDigit(c))
529     {
530         e.consume();
531     }
532     String gridSize = "" + gui.getGridSize();
533     int gridSizeDigits = gridSize.length();
534     if (textField.getText().length() >= gridSizeDigits)
535     {
536         e.consume();
537     }
538 }
539
540 @Override
541 public void keyReleased(KeyEvent e)
542 {
543 }
544
545 }
546 }
547
548 class PopupMenuItemListener implements ActionListener
549 {
550
551     private final JTextField textField;
552     private final int number;
553
554     PopupMenuItemListener(JTextField textField, int number)
555     {
556         this.textField = textField;
557         this.number = number;
558     }
559
560     @Override
561     public void actionPerformed(ActionEvent e)
562     {
563         textField.setText(number + " ");
564     }
565 }
566
567
568 class CellTextFieldListener implements DocumentListener
569 {
570
571     private final Controller c;
572     private final JTextField textField;
573     private final int x;
574     private final int y;
575
576     public CellTextFieldListener(Controller c, JTextField textField, int x,
577                                 int y)
578     {
579         this.c = c;
580         this.textField = textField;
581         this.x = x;
582         this.y = y;
583     }
584
585     @Override
586     public void insertUpdate(DocumentEvent e)
587     {
588         Integer value = Integer.parseInt(textField.getText());
589         c.setCellValue(x, y, value);
590     }
591
592     @Override
593     public void removeUpdate(DocumentEvent e)
594     {
595         c.unsetCellValue(x, y);
596     }
597
598     @Override
599     public void changedUpdate(DocumentEvent e)
600     {
601         Integer value = Integer.parseInt(textField.getText());
602         c.unsetCellValue(x, y);
603         c.setCellValue(x, y, value);
604     }
605 }
606 }
```

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 /**
14 * 
15 * @author michaeladrian39
16 */
17 public class GeneticParameters extends JFrame
18 {
19
20     private final GUI gui;
21     private final JLabel labelGenerations;
22     private final JLabel labelPopulation;
23     private final JLabel labelElitism;
24     private final JLabel labelCrossover;
25     private final JLabel labelMutation;
26     private final JTextField textFieldGenerations;
27     private final JTextField textFieldPopulation;
28     private final JTextField textFieldElitism;
29     private final JTextField textFieldCrossover;
30     private final JTextField textFieldMutation;
31     private final JButton buttonOK;
32     private final JButton buttonCancel;
33
34 /**
35 * Creates new form Test
36 */
37 public GeneticParameters(GUI gui)
38 {
39     this.gui = gui;
40     labelGenerations = new JLabel();
41     labelPopulation = new JLabel();
42     labelElitism = new JLabel();
43     labelCrossover = new JLabel();
44     labelMutation = new JLabel();
45     textFieldGenerations = new JTextField();
46     textFieldPopulation = new JTextField();
47     textFieldElitism = new JTextField();
48     textFieldCrossover = new JTextField();
49     textFieldMutation = new JTextField();
50     buttonOK = new JButton();
51     buttonCancel = new JButton();
52     initComponents();
53     this.setVisible(true);
54 }
55
56 private void initComponents()
57 {
58     this.setTitle("Set_Genetic_Algorithm_Parameters");
59     this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
60     this.setLocationRelativeTo(null);
61     this.setResizable(false);
62     labelGenerations.setText("Generations");
63     labelPopulation.setText("Population_Size");
64     labelElitism.setText("Elitism_Rate");
65     labelCrossover.setText("Crossover_Rate");
66     labelMutation.setText("Mutation_Rate");
67     buttonOK.setText("OK");
68     buttonCancel.setText("Cancel");
69     initGUI();
70     initActionListeners();
71 }
72
73 private void initActionListeners()
74 {
75     this.buttonOK.addActionListener(this::buttonOKActionPerformed);
76     this.buttonCancel.addActionListener(this::buttonCancelActionPerformed);
77 }
78
79 private void initGUI()
80 {
81     GroupLayout layout = new GroupLayout(this.getContentPane());
82     this.getContentPane().setLayout(layout);
83     layout.setHorizontalGroup(
84         layout.createParallelGroup(GroupLayout.Alignment.LEADING)
85             .addGroup(layout.createSequentialGroup()
86                 .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
87                     .addGroup(layout.createSequentialGroup()
88                         .addComponent(labelGenerations)
89                         .addComponent(labelPopulation)
90                         .addComponent(labelElitism)
91                         .addComponent(labelCrossover)
92                         .addComponent(labelMutation)
93                         .addComponent(buttonOK))
94                     .addGap(18, 18, 18)
95                     .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
96                         .addGroup(layout.createSequentialGroup()
97                             .addComponent(textFieldGenerations)
98                             .addComponent(textFieldPopulation)
99                             .addComponent(textFieldElitism)
100                            .addComponent(textFieldCrossover)
101                            .addComponent(textFieldMutation)))
102                         .addGap(18, 18, 18)
103                         .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
104                             .addComponent(buttonOK)
105                             .addComponent(buttonCancel)))))));
106     }
107 }
```

```

98         .addComponent(textFieldGenerations,
99                         GroupLayout.PREFERRED_SIZE,
100                        100,
101                         GroupLayout.PREFERRED_SIZE)
102         .addComponent(textFieldPopulation,
103                         GroupLayout.PREFERRED_SIZE,
104                        100,
105                         GroupLayout.PREFERRED_SIZE)
106         .addComponent(textFieldElitism,
107                         GroupLayout.PREFERRED_SIZE,
108                        100,
109                         GroupLayout.PREFERRED_SIZE)
110         .addComponent(textFieldCrossover,
111                         GroupLayout.PREFERRED_SIZE,
112                        100,
113                         GroupLayout.PREFERRED_SIZE)
114         .addComponent(textFieldMutation,
115                         GroupLayout.PREFERRED_SIZE,
116                        100,
117                         GroupLayout.PREFERRED_SIZE)
118         .addComponent(buttonCancel))
119     .addContainerGap(GroupLayout.DEFAULT_SIZE,
120                      Short.MAX_VALUE)));
121 layout.linkSize(SwingConstants.HORIZONTAL, buttonOK, buttonCancel);
122 layout.setVerticalGroup(
123     layout.createParallelGroup(GroupLayout.Alignment.LEADING)
124         .addGroup(layout.createSequentialGroup()
125             .addContainerGap()
126             .addGroup(layout.createParallelGroup(
127                 GroupLayout.Alignment.BASELINE)
128                 .addComponent(textFieldGenerations,
129                             GroupLayout.PREFERRED_SIZE,
130                             GroupLayout.DEFAULT_SIZE,
131                             GroupLayout.PREFERRED_SIZE)
132                 .addComponent(labelGenerations))
133             .addGap(18, 18, 18)
134             .addGroup(layout.createParallelGroup(
135                 GroupLayout.Alignment.BASELINE)
136                 .addComponent(labelPopulation)
137                 .addComponent(textFieldPopulation,
138                             GroupLayout.PREFERRED_SIZE,
139                             GroupLayout.DEFAULT_SIZE,
140                             GroupLayout.PREFERRED_SIZE))
141             .addGap(18, 18, 18)
142             .addGroup(layout.createParallelGroup(
143                 GroupLayout.Alignment.BASELINE)
144                 .addComponent(labelElitism)
145                 .addComponent(textFieldElitism,
146                             GroupLayout.PREFERRED_SIZE,
147                             GroupLayout.DEFAULT_SIZE,
148                             GroupLayout.PREFERRED_SIZE))
149             .addGap(18, 18, 18)
150             .addGroup(layout.createParallelGroup(
151                 GroupLayout.Alignment.BASELINE)
152                 .addComponent(labelCrossover)
153                 .addComponent(textFieldCrossover,
154                             GroupLayout.PREFERRED_SIZE,
155                             GroupLayout.DEFAULT_SIZE,
156                             GroupLayout.PREFERRED_SIZE))
157             .addGap(18, 18, 18)
158             .addGroup(layout.createParallelGroup(
159                 GroupLayout.Alignment.BASELINE)
160                 .addComponent(labelMutation)
161                 .addComponent(textFieldMutation,
162                             GroupLayout.PREFERRED_SIZE,
163                             GroupLayout.DEFAULT_SIZE,
164                             GroupLayout.PREFERRED_SIZE))
165             .addGap(18, 18, 18)
166             .addGroup(layout.createParallelGroup(
167                 GroupLayout.Alignment.BASELINE)
168                 .addComponent(buttonOK)
169                 .addComponent(buttonCancel))
170             .addContainerGap(GroupLayout.DEFAULT_SIZE,
171                             Short.MAX_VALUE)));
172     this.validate();
173     this.revalidate();
174     this.pack();
175     this.setLocationRelativeTo(null);
176 }
177
178 private void buttonOKActionPerformed(ActionEvent evt)
179 {
180     try
181     {
182         Integer generations = Integer.parseInt(textFieldGenerations.getText());
183         Integer population = Integer.parseInt(textFieldPopulation.getText());
184         Double elitism = Double.parseDouble(textFieldElitism.getText());
185         Double crossover = Double.parseDouble(textFieldCrossover.getText());
186         Double mutation = Double.parseDouble(textFieldMutation.getText());
187         gui.setGeneticAlgorithmParameters(generations, population, elitism,
188                                         crossover, mutation);
189         destroyFrame();
190     }
191     catch (NumberFormatException nfe)
192     {
193         JOptionPane.showMessageDialog(null, "Invalid number format.", "Error",
194                                     JOptionPane.ERROR_MESSAGE);
195         throw new IllegalStateException("Invalid number format.");
196     }
}

```

```
197    }
198    private void buttonCancelActionPerformed(ActionEvent evt)
199    {
200        destroyFrame();
201    }
202
203    private void destroyFrame()
204    {
205        this.getContentPane().removeAll();
206        this.validate();
207        this.revalidate();
208        this.pack();
209        this.setLocationRelativeTo(null);
210        this.dispose();
211    }
212}
213}
214}
```