

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
«tahun»**

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
«tahun»**

LEMBAR PENGESAHAN

PERBANDINGAN ALGORITMA BACKTRACKING DENGAN ALGORITMA HYBRID GENETIC UNTUK MENYELESAIKAN PERMAINAN CALCUDOKU

MICHAEL ADRIAN

NPM: 2013730039

Bandung, «tanggal» «bulan» «tahun»

Menyetujui,

Pembimbing Utama

Pembimbing Pendamping

Dr.rer.nat. Cecilia Esti Nugraheni

«pembimbing pendamping/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» «tahun»

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» «tahun»

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	3
1.4 Batasan Masalah	3
1.5 Metodologi Penelitian	3
1.6 Sistematika Pembahasan	4
2 LANDASAN TEORI	5
2.1 Calcudoku [1] [2]	5
2.2 Algoritma <i>Backtracking</i> [1]	7
2.3 Algoritma <i>Hybrid Genetic</i> [2]	14
2.3.1 Algoritma <i>Rule Based</i>	14
2.3.2 Algoritma Genetik	16
2.3.3 Algoritma <i>Hybrid Genetic</i>	16
3 ANALISIS	21
3.1 Analisis Algoritma <i>Backtracking</i>	21
3.2 Analisis Algoritma <i>Hybrid Genetic</i>	35
3.2.1 Algoritma <i>Rule Based</i>	35
3.2.2 Algoritma Genetik	35
3.3 Perangkat Lunak	51
3.3.1 Diagram <i>Use Case</i> dan Skenario	52
3.3.2 Diagram Kelas	55
4 PERANCANGAN	57
4.1 Perancangan Masukan	57
4.2 Perancangan Keluaran	57
4.3 Diagram Kelas	57
4.3.1 Kelas Grid	59
4.3.2 Kelas Cage	63
4.3.3 Kelas Cell	66
4.3.4 Kelas SolverBacktracking	67
4.3.5 Kelas SolverHybridGenetic	68
4.3.6 Kelas SolverRuleBased	68

4.3.7	Kelas SolverGenetic	73
4.3.8	Kelas Chromosome	76
4.3.9	Kelas ChromosomeComparator	77
4.3.10	Kelas Controller	78
4.3.11	Kelas Tester	79
4.4	Diagram <i>Use Case</i>	79
	DAFTAR REFERENSI	81

DAFTAR GAMBAR

1.1	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]	6
2.2	Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 2.1 [1] . .	7
2.3	Ilustrasi <i>State space tree</i> yang digunakan dalam algoritma <i>backtracking</i> [1]	9
2.4	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 3 x 3 [1]	10
2.5	Ilustrasi <i>state</i> 3, 4, dan 5 pada sebuah <i>grid</i> teka-teki Calcudoku [1]	11
2.6	Ilustrasi <i>state</i> 19 pada sebuah <i>grid</i> teka-teki Calcudoku [1]	12
2.7	<i>State</i> 25, simpul tujuan, sebagai hasil yang dicapai [1]	13
2.8	<i>State space tree</i> yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 2.4 [1]	13
2.9	Contoh bagaimana cara mendeteksi aturan <i>naked pair</i> [2]	14
2.10	Contoh aturan <i>evil twin</i> [2]	15
2.11	Contoh aturan <i>hidden single</i> [2]	15
2.12	Contoh aturan <i>killer combination</i> untuk <i>cage</i> dengan ukuran 2 sel dengan operasi matematika penjumlahan [2]	15
2.13	Contoh aturan <i>X-wing</i> [2]	16
2.14	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 6 x 6 [2]	17
2.15	Contoh proses kawin silang antara dua kromosom [2]	18
2.16	Contoh proses mutasi [2]	18
2.17	Alur penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma <i>hybrid genetic</i> [2]	19
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]	21
3.2	<i>State</i> 4	22
3.3	<i>State</i> 11	22
3.4	<i>State</i> 12	22
3.5	<i>State</i> 17	23
3.6	<i>State</i> 18	23
3.7	<i>State</i> 19	23
3.8	<i>State</i> 23	24
3.9	<i>State</i> 24	24
3.10	<i>State</i> 31	25
3.11	<i>State</i> 32	25
3.12	<i>State</i> 34	25
3.13	<i>State</i> 37	26
3.14	<i>State</i> 47	26
3.15	<i>State</i> 48	27
3.16	<i>State</i> 52	27
3.17	<i>State</i> 53	27

3.18	<i>State 68</i>	28
3.19	<i>State 69</i>	29
3.20	<i>State 71</i>	29
3.21	<i>State 72</i>	29
3.22	<i>State 74</i>	30
3.23	<i>State 75</i>	30
3.24	<i>State 76</i>	30
3.25	<i>State 77</i>	31
3.26	<i>State 78</i>	31
3.27	<i>State 81</i>	31
3.28	<i>State 83</i>	32
3.29	<i>State 85</i>	32
3.30	<i>State 88</i>	32
3.31	<i>State 92</i>	33
3.32	<i>State 93</i>	33
3.33	<i>State space tree</i> yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.1	34
3.34	Contoh permainan teka-teki Calcudoku dengan ukuran <i>grid</i> 6 x 6 yang belum diselesaikan, seperti yang digambarkan pada Gambar 1.1. [2]	35
3.35	Permainan teka-teki Calcudoku setelah diselesaikan dengan algoritma <i>rule based</i>	36
3.36	Kromosom 1 dalam Generasi ke-1	37
3.37	Kromosom 2 dalam Generasi ke-1	37
3.38	Kromosom 3 dalam Generasi ke-1	37
3.39	Kromosom 4 dalam Generasi ke-1	38
3.40	Kromosom 5 dalam Generasi ke-1	38
3.41	Kromosom 6 dalam Generasi ke-1	38
3.42	Kromosom 7 dalam Generasi ke-1	39
3.43	Kromosom 8 dalam Generasi ke-1	39
3.44	Kromosom 9 dalam Generasi ke-1	39
3.45	Kromosom 10 dalam Generasi ke-1	40
3.46	Kromosom 11 dalam Generasi ke-1	40
3.47	Kromosom 12 dalam Generasi ke-1	40
3.48	Kromosom 1 dalam Generasi ke-2	42
3.49	Kromosom 2 dalam Generasi ke-2	42
3.50	Kromosom 3 dalam Generasi ke-2	42
3.51	Kromosom 4 dalam Generasi ke-2	43
3.52	Kromosom 5 dalam Generasi ke-2	43
3.53	Kromosom 6 dalam Generasi ke-2	43
3.54	Kromosom 7 dalam Generasi ke-2	44
3.55	Kromosom 8 dalam Generasi ke-2	44
3.56	Kromosom 9 dalam Generasi ke-2	44
3.57	Kromosom 10 dalam Generasi ke-2	45
3.58	Kromosom 11 dalam Generasi ke-2	45
3.59	Kromosom 12 dalam Generasi ke-2	45
3.60	Kromosom 1 dalam Generasi ke-3	47
3.61	Kromosom 2 dalam Generasi ke-3	47
3.62	Kromosom 3 dalam Generasi ke-3	47
3.63	Kromosom 4 dalam Generasi ke-3	48
3.64	Kromosom 5 dalam Generasi ke-3	48
3.65	Kromosom 6 dalam Generasi ke-3	48
3.66	Kromosom 7 dalam Generasi ke-3	49

3.67 Kromosom 8 dalam Generasi ke-3	49
3.68 Kromosom 9 dalam Generasi ke-3	49
3.69 Kromosom 10 dalam Generasi ke-3	50
3.70 Kromosom 11 dalam Generasi ke-3	50
3.71 Kromosom 12 dalam Generasi ke-3	50
3.72 Diagram <i>use case</i> untuk perangkat lunak permainan teka-teki Calcudoku	53
3.73 Diagram kelas untuk perangkat lunak permainan teka-teki Calcudoku	56
4.1 Contoh <i>file</i> masukan.	58
4.2 Contoh keluaran.	58
4.3 Diagram kelas untuk perangkat lunak Calcudoku.	60
4.4 Diagram kelas Grid.	64
4.5 Diagram kelas Cage.	66
4.6 Diagram kelas Cell.	67
4.7 Diagram kelas SolverBacktracking.	68
4.8 Diagram kelas SolverHybridGenetic.	69
4.9 Diagram kelas SolverRuleBased.	74
4.10 Diagram kelas SolverGenetic.	76
4.11 Diagram kelas Chromosome.	77
4.12 Diagram kelas ChromosomeComparator.	77
4.13 Diagram kelas Controller.	79
4.14 Diagram kelas Tester.	79
4.15 Diagram <i>use case</i> untuk perangkat lunak Calcudoku.	80

DAFTAR TABEL

3.1	Tabel parameter untuk algoritma genetik yang akan digunakan untuk menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.35	36
3.2	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1	41
3.3	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-1	46
3.4	Tabel nilai kelayakan untuk kromosom-kromosom pada Generasi ke-3	51
3.5	Skenario me- <i>load file</i>	52
3.6	Skenario memilih salah satu dari dua <i>solver</i> yang disediakan	53
3.7	Skenario me- <i>reset</i> permainan	54
3.8	Skenario meminta perangkat lunak untuk memeriksa permainan	54
3.9	Skenario menutup <i>file</i> masukan	54
3.10	Skenario menyelesaikan permainan dengan usahanya sendiri	55

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

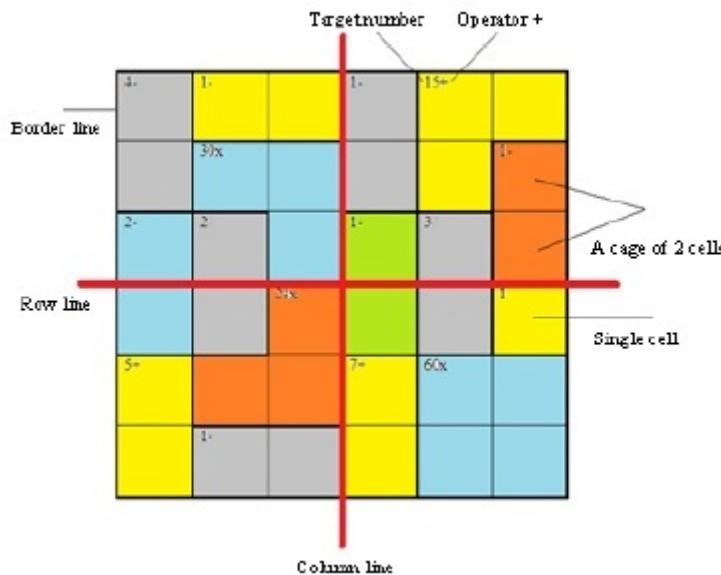
Calcudoku, atau dikenal juga sebagai KenKen, atau Mathdoku, adalah sebuah permainan teka-teki (*puzzle*) angka yang untuk menyelesaiannya memerlukan perpaduan dari logika dan kemampuan aritmatika yang sederhana. Permainan ini adalah sebuah permainan teka-teki logika yang sederhana, namun, untuk menemukan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

Teka-teki ini mirip dengan Sudoku. Persamaannya, tujuan dari teka-teki ini adalah mengisi setiap sel (*cell*) dalam (*grid*) dengan angka 1 sampai n tanpa pengulangan angka dalam setiap kolomnya dan barisnya untuk *grid* berukuran $n \times n$, dengan n adalah ukuran *grid*. Tidak ada angka yang boleh muncul lebih dari sekali dalam setiap baris atau kolom dalam *grid*. Perbedaannya, jika pada Sudoku *grid* berukuran $n \times n$ dibagi menjadi n (*cage*) dengan setiap *cage* terdiri atas n sel, pada Calcudoku *grid* dibagi menjadi sejumlah *cage* yang jumlah selnya bervariasi. Setiap *cage* dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel. Angka-angka dalam satu *cage* yang sama harus menghasilkan angka tujuan yang telah ditentukan jika dihitung menggunakan operasi matematika yang telah ditentukan (penjumlahan, pengurangan, perkalian, atau pembagian). Angka-angka dalam satu *cage* juga boleh berulang, selama pengulangan tidak terjadi dalam satu kolom atau baris yang sama. Jika *cage* hanya berisi satu sel, maka satu-satunya kemungkinan jawaban untuk sel tersebut adalah angka tujuan dari *cage* tersebut. Angka tujuan dan operasi matematika dituliskan di sudut kiri atas *cage*. Pada awalnya, setiap sel dalam setiap *cage* dalam teka-teki ini kosong, belum terisi oleh angka-angka. *Border line* adalah garis pembatas terluar, *row line* adalah garis pembatas antar baris, dan *column line* adalah garis pembatas antar kolom. Gambar 1.1 menggambarkan contoh sebuah permainan teka-teki Calcudoku. [1] citejohanna:12:hybrid

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 di-



Gambar 1.1: Contoh permainan teka-teki Calcudoku dengan penjelasan elemen-elemen dari teka-teki ini [2]

gunakan 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 meng-*encode* 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 3×3 sampai dengan 9×9 .

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 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 membahas tentang 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 membahas tentang kesimpulan dari pembuatan perangkat lunak Calcudoku dan hasil pengujiannya, 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 Ivanky 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.

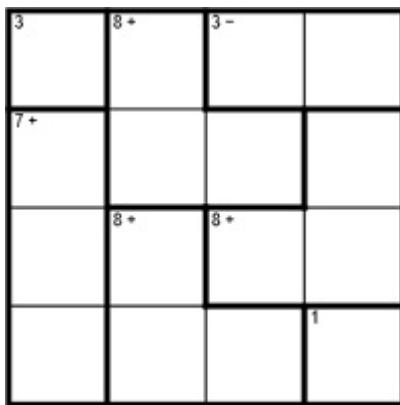
Seperti dalam Sudoku, dalam teka-teki ini, pemain diberikan sebuah *grid* dengan ukuran $n \times n$, dengan n biasanya $3 \leq n \leq 9$. *Grid* ini harus diisi dengan angka 1 sampai dengan n sehingga dalam setiap baris setiap angka hanya muncul sekali, dalam setiap kolom setiap angka hanya muncul sekali. 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 harus pertama-tama 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]

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 $O_i \in \{+, -, \times, \div, =\}$,

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

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

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. Menurut Johanna, Lukas, dan Saputra, 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*.

Menurut Johanna, Lukas, dan Saputra, 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.

Menurut Fahda, implementasi algoritma *backtracking* memiliki beberapa 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:

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

n adalah jumlah sel dalam satu baris atau kolom. X adalah sebuah *tuple* yang berukuran n^2 , 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 di sel pada sudut kanan bawah. S_i adalah sebuah himpunan yang berisi angka-angka dari 1 sampai 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 (*generating function*)

Fungsi pembatas dinyatakan sebagai:

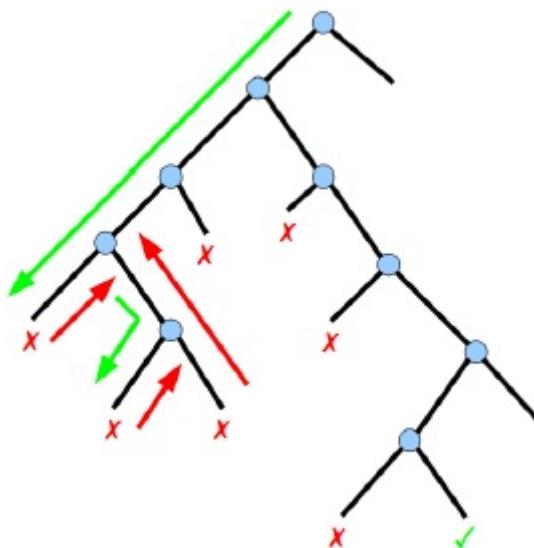
$$B(x_1, x_2, \dots, x_k)$$

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*).



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

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

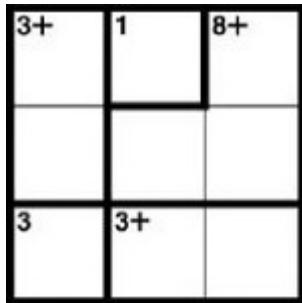
Setiap simpul di dalam *state space tree* terkait dengan panggilan rekursif. Jika jumlah simpul di dalam pohon $2n$ atau $n!$, maka pada kasus terburuk untuk algoritma *backtracking* ini memiliki kompleksitas waktu $O(p(n)2n)$ atau $O(q(n)n!)$, dengan $p(n)$ dan $q(n)$ sebagai polinomial dengan n -derajat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

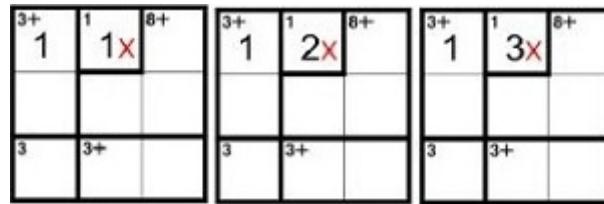


Gambar 2.4: Contoh permainan teka-teki Calcudoku dengan ukuran $grid$ 3×3 [1]

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 sel yang lainnya dikurangi dengan x_k menghasilkan nilai 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 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 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.

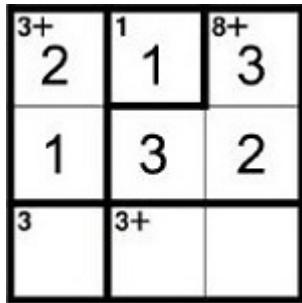
State space tree bersifat dinamis, berkembang secara terus-menerus sampai solusi ditemukan. Untuk mengilustrasikan berkembangnya *state space tree*, teka-teki Calcudoku yang digambarkan pada Gambar 2.4 akan digunakan. Berikut ini adalah tahap-tahap berkembangnya *state space tree* untuk teka-teki tersebut.

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 diisi pada sel pertama yang kosong, yaitu sel yang terletak di sudut kiri atas $grid$, atau sel pada kolom ke-1 dan baris ke-1 (*state 2*). Fungsi pembatas akan memeriksa jika langkah ini adalah langkah yang berlaku, dan ternyata langkah ini berlaku.
3. Untuk sel yang kosong berikutnya, yaitu x_2 , atau sel pada kolom ke-2 dan baris ke-1, fungsi pembangkit akan membangkitkan angka 1 (*state 3*), tetapi langkah ini gagal dalam pemeriksaan baris dalam fungsi pembatas karena angka 1 sudah pernah digunakan pada baris tersebut, ini membentuk sebuah simpul mati.



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

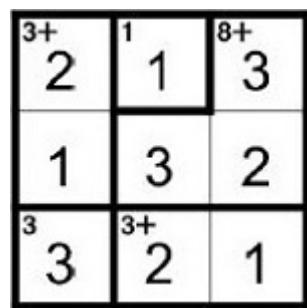
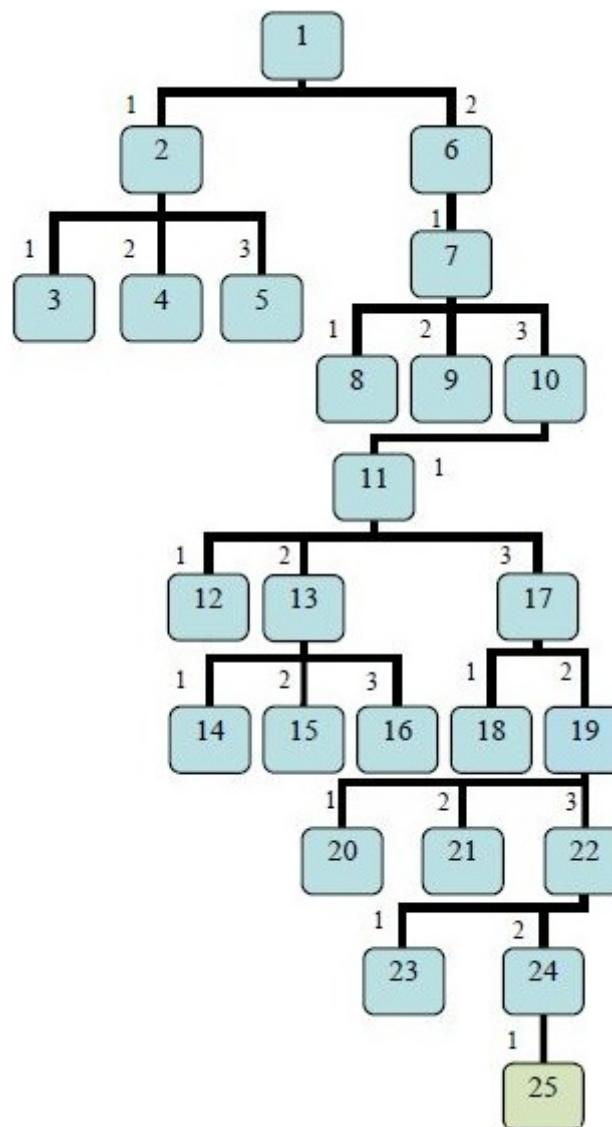
4. Fungsi pembangkit akan mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 4), tetapi langkah ini gagal dalam pemeriksaan *grid* dalam fungsi pembatas karena angka 2 tidak sama dengan angka tujuan, yaitu angka 1.
5. Fungsi pembangkit akan mencoba kemungkinan angka berikutnya, yaitu angka 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.
6. Karena tidak ada solusi yang mungkin, maka algoritma *backtracking* akan 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 algoritma 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 algoritma 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, maka algoritma telah selesai mengisi baris ke-1, dan akan mulai mengisi baris ke-2.
11. Algoritma lalu 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 algoritma 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 algoritma bisa maju ke sel berikutnya, yaitu x_6 , atau sel pada kolom ke-3 dan baris ke-2.
14. Algoritma mengisikan x_6 dengan angka 1 (*state* 14), tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan pada baris tersebut.
15. Algoritma lalu 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. Algoritma lalu 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.



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

17. Karena semua kemungkinan angka gagal dalam pemeriksaan baris dan kolom, maka algoritma akan mundur ke *state* 11 dan mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 17), dan ternyata angka 3 berlaku sebagai x_5 , sehingga algoritma bisa maju ke sel berikutnya, yaitu x_6 .
18. Algoritma lalu mencoba angka 1 (*state* 18) sebagai x_6 , tetapi gagal dalam pemeriksaan baris karena angka 1 sudah pernah digunakan dalam baris tersebut.
19. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 19), dan ternyata angka 2 berlaku. Algoritma telah selesai mengisi baris ke-2. Gambar 2.6 menggambarkan *state* 19 dalam penyelesaian teka-teki Calcudoku ini.
20. Algoritma mulai mengisikan sel-sel yang terletak pada baris ke-3. Algoritma mengisi dari kolom yang paling kiri ke kolom yang paling kanan. Algoritma 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. Algoritma lalu 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. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state* 22), dan ternyata berhasil, sehingga algoritma bisa maju ke sel berikutnya, yaitu x_8 , atau sel pada kolom ke-2 dan baris ke-3.
23. Algoritma lalu mencoba mengisikan angka 1 pada x_8 (*state* 23), tetapi gagal dalam pemeriksaan kolom, karena angka 1 sudah pernah digunakan dalam kolom tersebut.
24. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state* 24), dan ternyata berhasil, sehingga algoritma 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*. Algoritma lalu mencoba mengisikan x_9 dengan angka 1 (*state* 25), dan ternyata berhasil. Algoritma telah selesai mengisikan seluruh sel dalam *grid* dengan benar. Gambar 2.7 menggambarkan *state* 25 dalam penyelesaian teka-teki Calcudoku ini. Algoritma 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 $n \times n$ seharusnya 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 2.7: *State 25*, simpul tujuan, sebagai hasil yang dicapai [1]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. Menurut Johanna, Lukas, dan Saputra, beberapa aturan logika yang digunakan dalam algoritma ini adalah *single square rule*, *naked subset rule*, *hidden single rule*, *evil twin rule*, *killer combination*, dan *X-wing* [2].

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

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

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

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

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

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

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

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

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 baikkah 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.

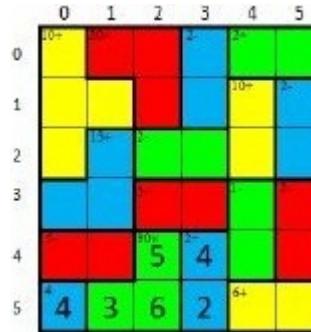
Cara kerja algoritma genetik adalah sebagai berikut [2]:

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

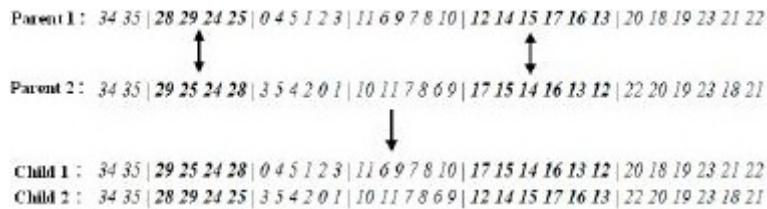
Gambar 2.14: Contoh permainan teka-teki Calcudoku dengan ukuran grid 6×6 [2]

$1, 2, \dots, n$. Setelah nilai dari satu sel sudah ditentukan, kemungkinan nilai untuk beberapa sel tertentu diperbaharui. Misalnya, penggunaan aturan *naked single* yang dinyatakan dalam persamaan 1 di bawah ini, akan mengakibatkan semua kemungkinan nilai untuk semua sel lain dalam baris yang 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]

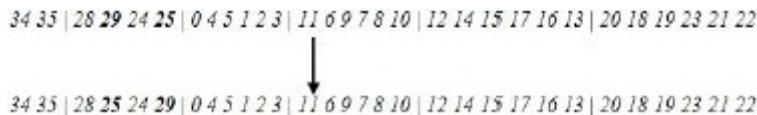
1. $|P(C_{b,k})| = 1 \wedge x \in P(C_{b,k}) \rightarrow V(C_{b,k}) = x$, artinya jika sebuah *cage* berukuran 1 sel, dan x adalah nilai tujuan dari *cage* tersebut, maka nilai dari sel tersebut adalah x .
2. $(V(C_{b,k}) = x) \wedge (\forall a \in \{1, 2, \dots, n\}) \rightarrow P(C_{a,k}) = P(C_{a,k}) - \{x\}$, 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.

Menurut Johanna, Lukas, dan Saputra, fungsi objektif, yang direpresentasikan dengan x_j , akan dihitung setelah pembangkitan nilai dari gen pada kromosom sudah dilakukan. Nilai untuk gen 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 ,



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



Gambar 2.16: Contoh proses mutasi [2]

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

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

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

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

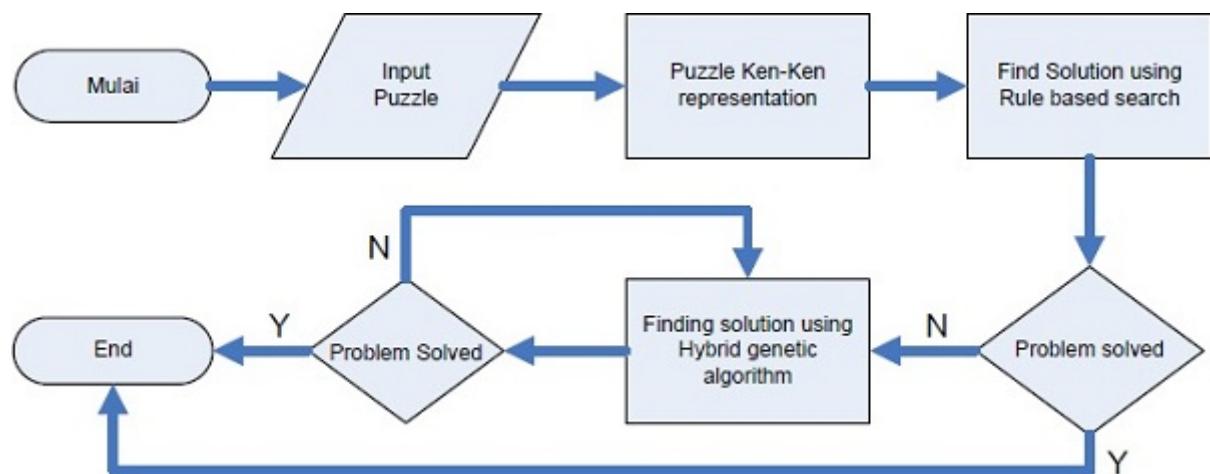
Dalam proses reproduksi kawin silang, dua kromosom, yaitu kromosom orang tua, disilangkan untuk membuat dua kromosom yang baru, yaitu kromosom anak, dengan metodologi kawin silang *N-segments*. Gambar 2.15 menggambarkan contoh proses kawin silang antara dua kromosom.

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* menurut Johanna, Lukas, dan Saputra adalah sebagai berikut [2]:

1. Masukkan teka-teki yang akan diselesaikan sebagai input. Teka-teki Calcudoku diinputkan oleh pemain dalam bentuk file.
2. Program akan merepresentasikan input yang dimasukkan dalam format teka-teki. File teka-teki Calcudoku yang telah diinputkan oleh pemain ditampilkan ke layar sebagai teka-teki Calcudoku.
3. Program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
4. Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based*, maka algoritma selesai.
5. Jika program gagal dengan menggunakan algoritma *rule based*, maka program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
6. Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik, maka algoritma selesai.
7. Jika program gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, artinya algoritma 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 3.1 sebagai contoh.

1. Algoritma *backtracking* dimulai dengan teka-teki yang belum diselesaikan, seperti yang digambarkan pada Gambar 3.1 (*state 1*).
2. Algoritma mengisikan sel pada baris ke-1 dan kolom ke-1 dengan angka 1 (*state 2*), tetapi angka 1 tidak sesuai dengan angka tujuan dari *cage* tersebut.
3. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 3*), tetapi angka 2 juga tidak sesuai dengan angka tujuan dari *cage* tersebut.
4. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 4*), seperti dapat dilihat pada Gambar 3.2, dan ternyata angka 3 sesuai dengan angka tujuan dari *cage* tersebut, sehingga algoritma dapat maju ke sel berikutnya.
5. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-2 dengan angka 1 (*state 5*). Algoritma lalu maju ke sel berikutnya.
6. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state 6*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
7. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 7*). Algoritma lalu maju ke sel berikutnya.

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

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

³ 3	⁸⁺	³⁻	
⁷⁺			
	⁸⁺	⁸⁺	
			1

Gambar 3.2: *State 4*

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

³ 3	⁸⁺ 1	³⁻ 3	
⁷⁺			
	⁸⁺	⁸⁺	
			1

Gambar 3.4: *State 12*

³ 3	⁸⁺ 1	³⁻ 4	4
7+			
	⁸⁺	⁸⁺	
			1

Gambar 3.5: *State 17*

³ 3	⁸⁺ 2	³⁻	
7+			
	⁸⁺	⁸⁺	
			1

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 3.5, tetapi angka 4 sudah pernah digunakan dalam baris tersebut.
18. Karena semua kemungkinan angka untuk baris ke-1 dan kolom ke-3 dan ke-4 telah dicoba dan gagal, maka algoritma harus mundur kembali ke (*state 5*). Algoritma mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 18*), seperti dapat dilihat pada Gambar 3.6. Algoritma lalu maju ke sel berikutnya.
19. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-3 dengan angka 1 (*state 19*), seperti dapat dilihat pada Gambar 3.7. Algoritma lalu maju ke sel berikutnya.
20. Algoritma lalu mengisikan sel pada baris ke-1 dan kolom ke-4 dengan angka 1 (*state 20*), tetapi angka 1 sudah pernah digunakan dalam baris tersebut.
21. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 21*), tetapi angka 2 sudah pernah digunakan dalam baris tersebut.

³ 3	⁸⁺ 2	³⁻ 1	
7+			
	⁸⁺	⁸⁺	
			1

Gambar 3.7: *State 19*

³ 3	⁸⁺ 2	³⁻ 1	4
7+			
	⁸⁺	⁸⁺	
			1

Gambar 3.8: *State 23*

³ 3	⁸⁺ 2	³⁻ 1	4
7+ 1			
	⁸⁺	⁸⁺	
			1

Gambar 3.9: *State 24*

22. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 22*), tetapi angka 3 sudah pernah digunakan dalam baris tersebut.
23. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 23*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.8. Algoritma telah selesai 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 3.9. 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 3.10, tetapi hasilnya tidak sesuai dengan angka tujuan dari *cage* tersebut.

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

Gambar 3.10: *State 31*

³ 3	⁸⁺ 2	³⁻ 1	4
⁷⁺ 1	4		
	8+	8+	
			1

Gambar 3.11: *State 32*

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 3.11. 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.
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 3.12. 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 3.13. Algoritma telah selesai mengisikan baris ke-2, sehingga bisa maju ke baris berikutnya.

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

Gambar 3.12: *State 34*

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

Gambar 3.13: *State 37*

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

Gambar 3.14: *State 47*

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.
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 3.14, 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 3.15. Algoritma lalu maju ke sel berikutnya.

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

Gambar 3.15: *State 48*

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

Gambar 3.16: *State 52*

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.
52. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 52*), seperti dapat dilihat pada Gambar 3.16, 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 3.17, 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.

3	2	1	4
⁷⁺ 1	4	2	3
2	⁸⁺ 2	⁸⁺	
			1

Gambar 3.17: *State 53*

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

Gambar 3.18: *State 68*

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.
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 3.18. 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 3.19, tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.

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

Gambar 3.19: *State 69*

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

Gambar 3.20: *State 71*

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 3.20, 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 3.21, tetapi angka 2 sudah pernah digunakan dalam baris tersebut.
73. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 73*), tetapi angka 3 sudah pernah digunakan dalam baris dan kolom tersebut.
74. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 4 (*state 74*), seperti dapat dilihat di Gambar 3.22, 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 3.23, tetapi angka 4 sudah pernah digunakan dalam kolom tersebut.

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

Gambar 3.21: *State 72*

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

Gambar 3.22: *State 74*

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

Gambar 3.23: *State 75*

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 3.24, 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 3.25. 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 3.26. Algoritma lalu maju ke sel berikutnya.
79. Algoritma lalu mengisikan sel pada baris ke-3 dan kolom ke-3 dengan angka 1 (*state 79*), tetapi angka 1 sudah pernah digunakan dalam baris dan kolom tersebut.
80. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 80*), tetapi angka 2 sudah pernah digunakan dalam kolom tersebut.
81. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 81*), seperti dapat dilihat pada Gambar 3.27. 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.

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

Gambar 3.24: *State 76*

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

Gambar 3.25: *State 77*

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

Gambar 3.26: *State 78*

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

Gambar 3.27: *State 81*

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

Gambar 3.28: *State 83*

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

Gambar 3.29: *State 85*

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 3.28. 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 3.29. Algoritma lalu maju ke sel berikutnya.
86. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-2 dengan angka 1 (*state 86*), tetapi angka 1 sudah pernah digunakan dalam kolom tersebut.
87. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 2 (*state 87*), tetapi angka 2 sudah pernah digunakan dalam baris dan kolom tersebut.
88. Algoritma lalu mencoba kemungkinan angka berikutnya, yaitu angka 3 (*state 88*), seperti dapat dilihat pada Gambar 3.30. 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.

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

Gambar 3.30: *State 88*

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

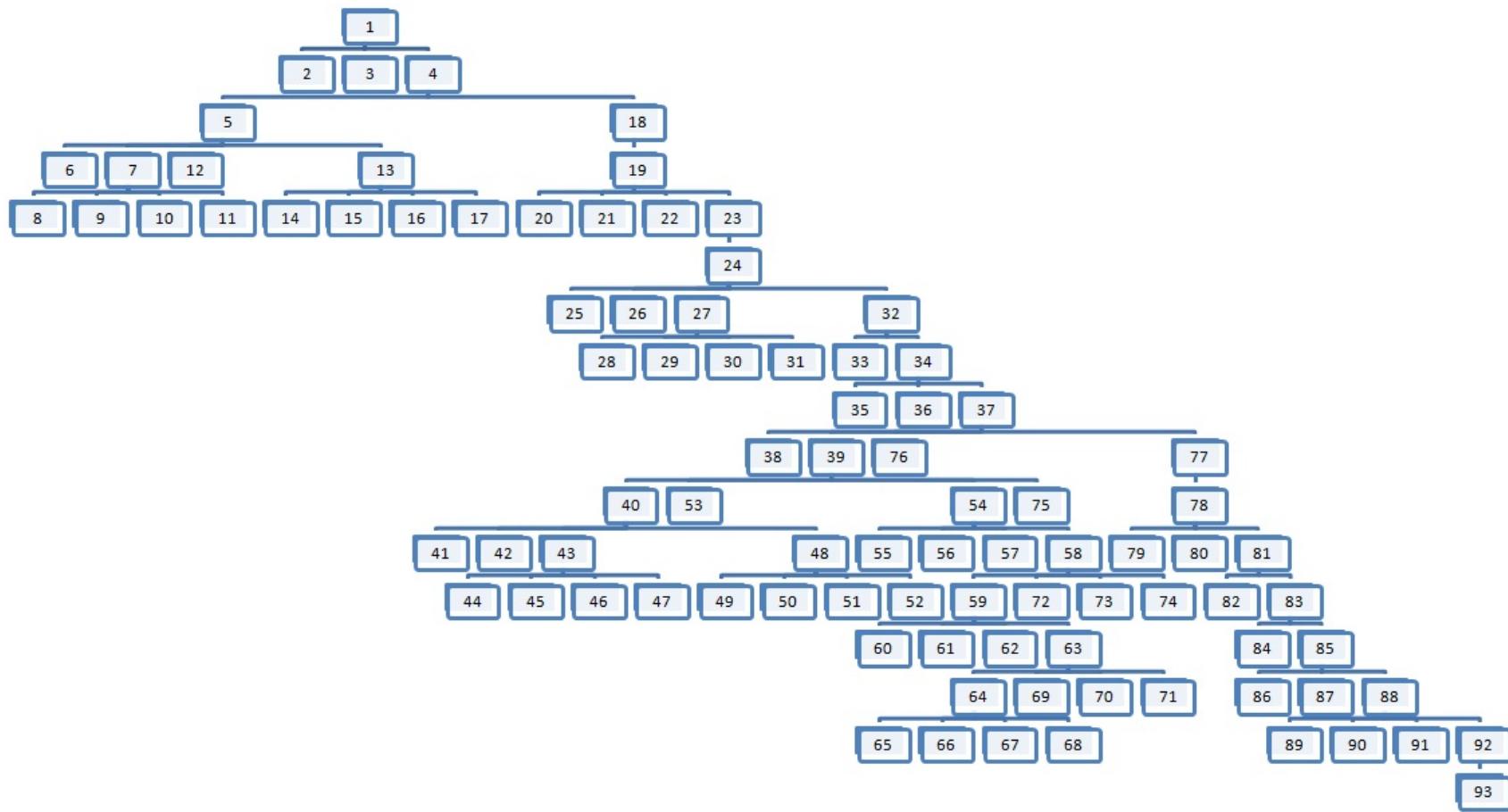
Gambar 3.31: *State 92*

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

Gambar 3.32: *State 93*

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 3.31. Algoritma lalu maju ke sel berikutnya.
93. Algoritma lalu mengisikan sel pada baris ke-4 dan kolom ke-4 dengan angka 1 (*state 93*), dan ternyata hasilnya sesuai dengan angka tujuan dari *cage* tersebut, seperti dapat dilihat pada Gambar 3.32. Algoritma *backtracking* telah selesai mengisi semua sel dalam permainan teka-teki Calcudoku ini dengan benar.

Algoritma ini mencapai solusinya pada state 93, seperti pada *state space tree* yang digambarkan dalam Gambar 3.33. *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.



Gambar 3.33: *State space tree* yang dikembangkan dalam proses menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.1

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

Gambar 3.34: Contoh permainan teka-teki Calcudoku dengan ukuran grid 6×6 yang belum diselesaikan, seperti yang digambarkan pada Gambar 1.1. [2]

3.2 Analisis Algoritma *Hybrid Genetic*

Untuk mengilustrasikan cara kerja algoritma *hybrid genetic*, akan digunakan permainan teka-teki Calcudoku yang digambarkan pada Gambar 3.34 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 3.35.

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

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

Tabel 3.1: Tabel parameter untuk algoritma genetik yang akan digunakan untuk menyelesaikan teka-teki Calcudoku yang digambarkan pada Gambar 3.35

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 3.36 menggambarkan Kromosom 1, gambar 3.37 menggambarkan Kromosom 2, gambar 3.38 menggambarkan Kromosom 3, gambar 3.39 menggambarkan Kromosom 4, gambar 3.40 menggambarkan Kromosom 5, gambar 3.41 menggambarkan Kromosom 6, gambar 3.42 menggambarkan Kromosom 7, gambar 3.43 menggambarkan Kromosom 8, gambar 3.44 menggambarkan Kromosom 9, gambar 3.45 menggambarkan Kromosom 10, gambar 3.46 menggambarkan Kromosom 11, dan gambar 3.47 menggambarkan Kromosom 12.

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

Gambar 3.36: Kromosom 1 dalam Generasi ke-1

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

Gambar 3.37: Kromosom 2 dalam Generasi ke-1

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

Gambar 3.38: Kromosom 3 dalam Generasi ke-1

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

Gambar 3.39: Kromosom 4 dalam Generasi ke-1

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

Gambar 3.40: Kromosom 5 dalam Generasi ke-1

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

Gambar 3.41: Kromosom 6 dalam Generasi ke-1

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

Gambar 3.42: Kromosom 7 dalam Generasi ke-1

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

Gambar 3.43: Kromosom 8 dalam Generasi ke-1

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

Gambar 3.44: Kromosom 9 dalam Generasi ke-1

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

Gambar 3.45: Kromosom 10 dalam Generasi ke-1

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

Gambar 3.46: Kromosom 11 dalam Generasi ke-1

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

Gambar 3.47: Kromosom 12 dalam 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

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

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 3.48 menggambarkan Kromosom 1, yaitu Kromosom 12 dari Generasi ke-1, gambar 3.49 menggambarkan Kromosom 2, yaitu Kromosom 5 dari Generasi ke-1, gambar 3.50 menggambarkan Kromosom 3, yaitu Kromosom 7 dari Generasi ke-1, gambar 3.51 menggambarkan Kromosom 4, yaitu Kromosom 11 dari Generasi ke-1, gambar 3.52 menggambarkan Kromosom 5, yaitu Kromosom 1 dari Generasi ke-1, gambar 3.53 menggambarkan Kromosom 6, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 12 dari Generasi ke-1, gambar 3.54 menggambarkan Kromosom 7, yaitu hasil kawin silang dari Kromosom 3 dan Kromosom 8 dari Generasi ke-1, gambar 3.55 menggambarkan Kromosom 8, yaitu hasil kawin silang dari Kromosom 7 dan Kromosom 10 dari Generasi ke-1, gambar 3.56 menggambarkan Kromosom 9, yaitu hasil kawin silang dari Kromosom 7 dan Kromosom 10 dari Generasi ke-1, gambar 3.57 menggambarkan Kromosom 10, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 5 dari Generasi ke-1, gambar 3.58 menggambarkan Kromosom 11, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 5 dari Generasi ke-1, dan gambar 3.59 menggambarkan Kromosom 12, yaitu hasil mutasi dari Kromosom 12 dari Generasi ke-1.

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

Gambar 3.48: Kromosom 1 dalam Generasi ke-2

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

Gambar 3.49: Kromosom 2 dalam Generasi ke-2

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

Gambar 3.50: Kromosom 3 dalam Generasi ke-2

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

Gambar 3.51: Kromosom 4 dalam Generasi ke-2

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

Gambar 3.52: Kromosom 5 dalam Generasi ke-2

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

Gambar 3.53: Kromosom 6 dalam Generasi ke-2

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

Gambar 3.54: Kromosom 7 dalam Generasi ke-2

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

Gambar 3.55: Kromosom 8 dalam Generasi ke-2

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

Gambar 3.56: Kromosom 9 dalam Generasi ke-2

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

Gambar 3.57: Kromosom 10 dalam Generasi ke-2

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

Gambar 3.58: Kromosom 11 dalam Generasi ke-2

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

Gambar 3.59: Kromosom 12 dalam 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

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

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 3.60 menggambarkan Kromosom 1, yaitu Kromosom 1 dari Generasi ke-2, Gambar 3.61 menggambarkan Kromosom 2, yaitu Kromosom 12 dari Generasi ke-2, Gambar 3.62 menggambarkan Kromosom 3, yaitu Kromosom 2 dari Generasi ke-2, Gambar 3.63 menggambarkan Kromosom 4, yaitu Kromosom 3 dari Generasi ke-2, Gambar 3.64 menggambarkan Kromosom 5, yaitu Kromosom 4 dari Generasi ke-2, Gambar 3.65 menggambarkan Kromosom 6, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 12 dari Generasi ke-2, Gambar 3.66 menggambarkan Kromosom 7, yaitu hasil kawin silang dari Kromosom 2 dan Kromosom 12 dari Generasi ke-2, Gambar 3.67 menggambarkan Kromosom 8, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 9 dari Generasi ke-2, Gambar 3.68 menggambarkan Kromosom 9, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 9 dari Generasi ke-2, Gambar 3.69 menggambarkan Kromosom 10, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 6 dari Generasi ke-2, Gambar 3.70 menggambarkan Kromosom 11, yaitu hasil kawin silang dari Kromosom 5 dan Kromosom 6 dari Generasi ke-2, dan Gambar 3.71 menggambarkan Kromosom 12, yaitu hasil mutasi dari Kromosom 2 dari Generasi ke-2.

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

Gambar 3.60: Kromosom 1 dalam Generasi ke-3

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

Gambar 3.61: Kromosom 2 dalam Generasi ke-3

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

Gambar 3.62: Kromosom 3 dalam Generasi ke-3

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

Gambar 3.63: Kromosom 4 dalam Generasi ke-3

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

Gambar 3.64: Kromosom 5 dalam Generasi ke-3

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

Gambar 3.65: Kromosom 6 dalam Generasi ke-3

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

Gambar 3.66: Kromosom 7 dalam Generasi ke-3

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

Gambar 3.67: Kromosom 8 dalam Generasi ke-3

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

Gambar 3.68: Kromosom 9 dalam Generasi ke-3

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

Gambar 3.69: Kromosom 10 dalam Generasi ke-3

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

Gambar 3.70: Kromosom 11 dalam Generasi ke-3

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

Gambar 3.71: Kromosom 12 dalam 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

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

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

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

Tabel 3.5: Skenario me-load file

dalam sebuah *cage* tidak mencapai angka tujuan yang ditentukan setelah dihitung dengan operasi matematika yang ditentukan.

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

3.3.1 Diagram *Use Case* dan Skenario

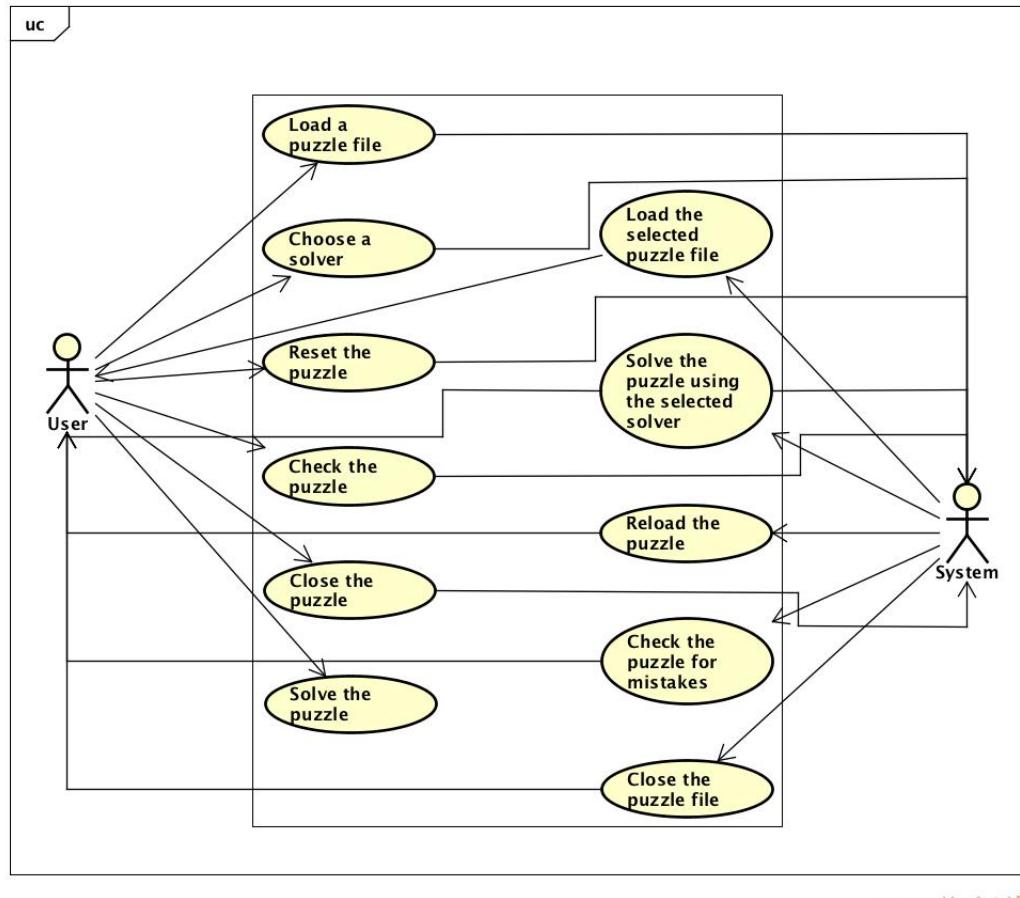
Diagram *use case* adalah diagram yang menggambarkan interaksi antara sistem (perangkat lunak) dengan pengguna. Berdasarkan analisis perangkat lunak yang telah dilakukan, maka pengguna dapat:

1. Me-load 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.

Diagram *use case* untuk perangkat lunak permainan teka-teki Calcudoku dapat dilihat pada Gambar 3.72.

Berdasarkan diagram *use case* yang dapat dilihat pada Gambar 3.72, skenario-skenario yang dapat dilakukan oleh pengguna adalah:

1. Me-load 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.
4. Meminta perangkat lunak untuk memeriksa permainan. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.8.



powered by Astah

Gambar 3.72: Diagram *use case* untuk perangkat lunak permainan teka-teki Calcudoku

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

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 error "Puzzle file not loaded".

Tabel 3.7: Skenario me-reset 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 error "Puzzle file not loaded".

Tabel 3.8: Skenario meminta perangkat lunak untuk memeriksa permainan

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

Tabel 3.9: Skenario menutup *file* masukan

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 n , dengan n 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 n , dengan n merupakan ukuran dari <i>grid</i> .

Tabel 3.10: Skenario Menyelesaikan permainan dengan usahanya sendiri

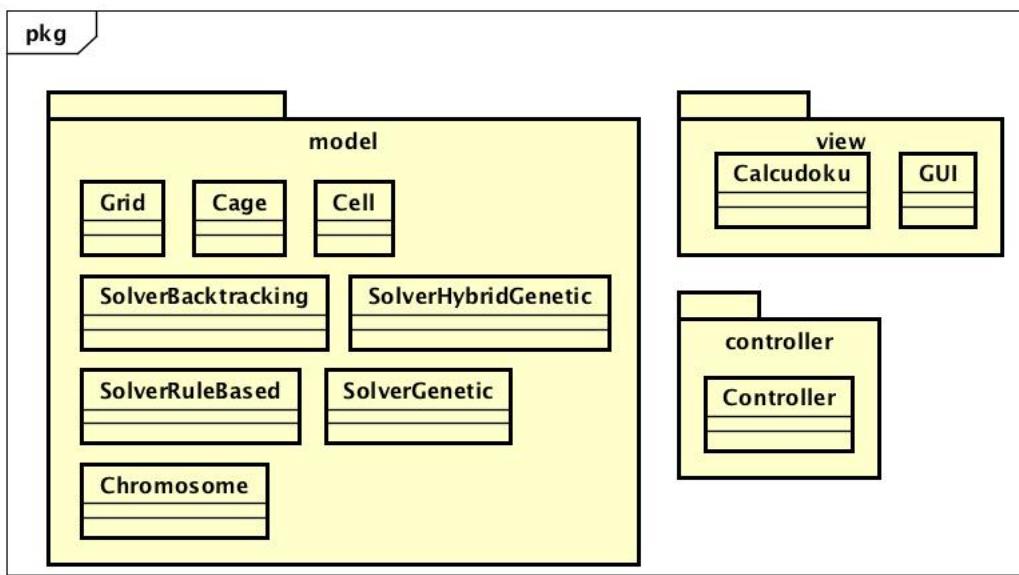
5. Menutup *file* masukan. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.9.
6. Menyelesaikan permainan dengan usahanya sendiri. Penjelasan untuk skenario ini dapat dilihat pada Tabel 3.10.

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 3.73.

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.
 - (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:



powered by Astah

Gambar 3.73: Diagram kelas untuk perangkat lunak permainan teka-teki Calcudoku

- 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).
 - Kelas GUI, yaitu kelas yang merepresentasikan *grid* untuk permainan Calcudoku.
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 text*, seperti yang ditunjukkan pada Gambar 4.1.

Adapun rincian dari *file text* 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.

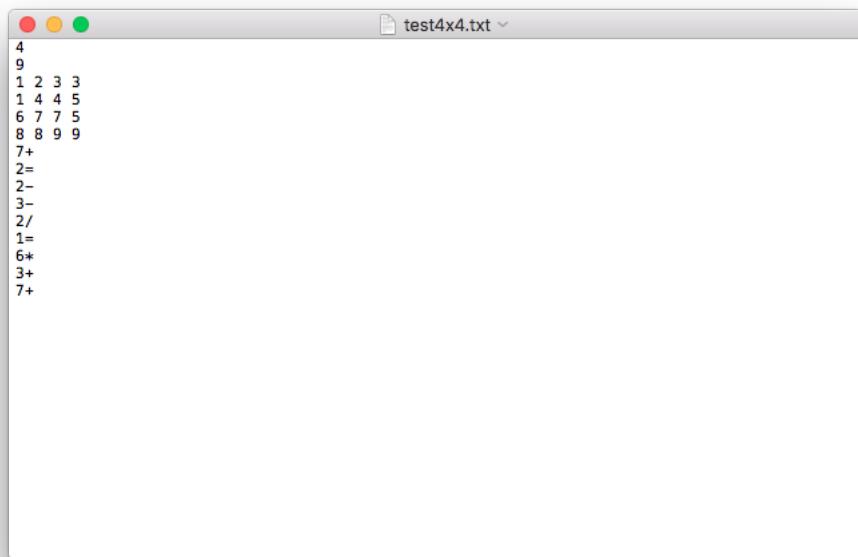
4.2 Perancangan Keluaran

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 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.

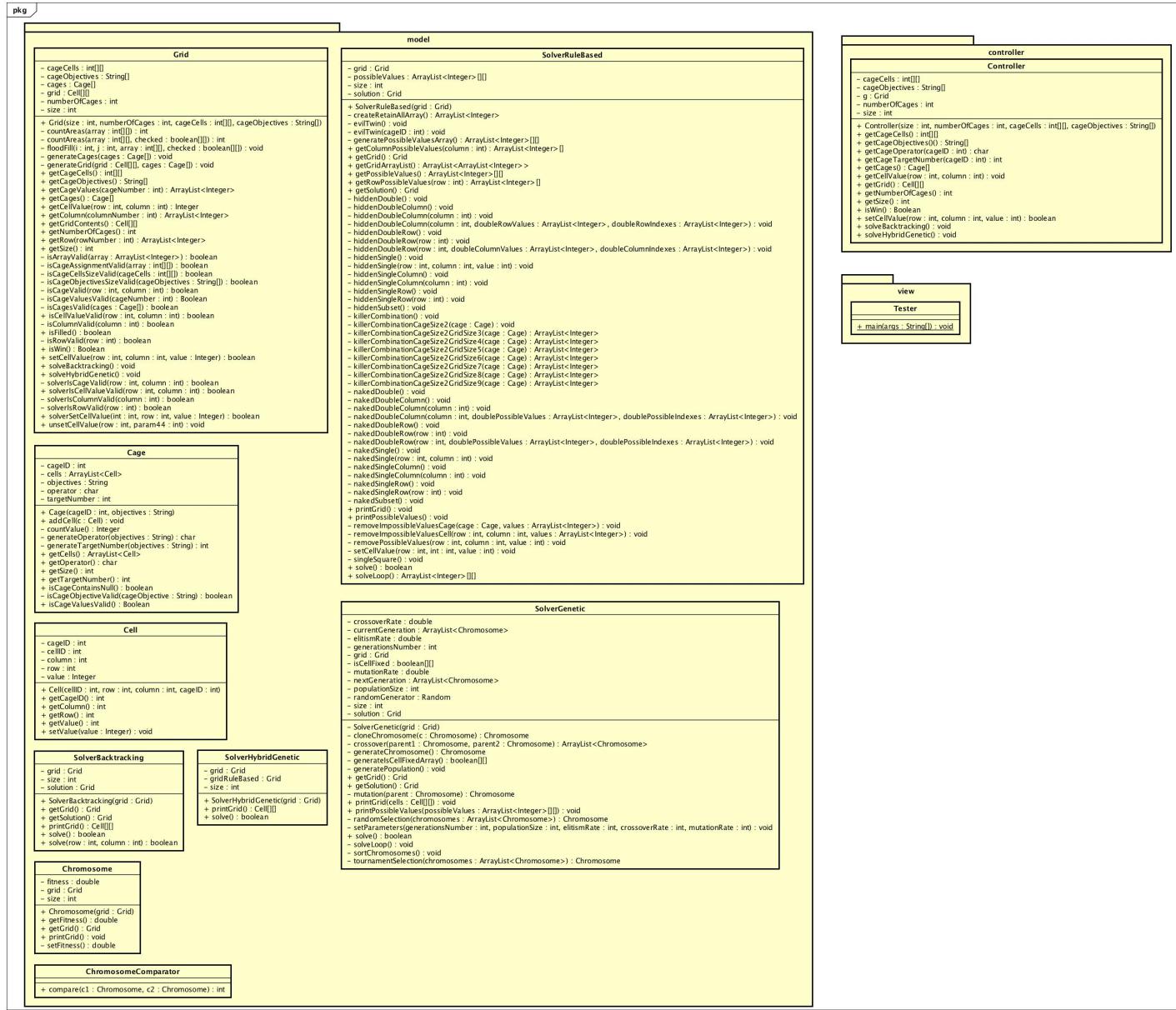


Gambar 4.1: Contoh *file* masukan.

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

Gambar 4.2: Contoh keluaran.

- (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 tampilan dari perangkat lunak ini. Package ini memiliki beberapa kelas, yaitu:
- (a) Tester, yaitu kelas untuk menguji program ini.
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 4.3.
- Berikut ini adalah rincian dari setiap kelas, dengan setiap atribut dan setiap *method* yang dimilikinya.
- #### 4.3.1 Kelas Grid
- Kelas Grid mempunyai beberapa atribut, yaitu:
1. size, yaitu ukuran dari matriks *grid*.
 2. numberOfCages, yaitu banyaknya *cage* yang terdapat dalam *grid*.
 3. cageCells, yaitu sebuah matriks *cage assignment*. Matriks ini merepresentasikan posisi dari setiap *cage* dalam *grid*.
 4. cageObjectives, yaitu sebuah *array* yang berisi *cage objectives* untuk setiap *cage*. *Cage objectives* berisikan angka tujuan dan operasi matematika yang telah ditentukan.
 5. grid, yaitu representasi dari *grid* dalam teka-teki Calcudoku. *grid* adalah sebuah matriks yang berisi sel-sel. Matriks ini berukuran $n \times n$.
 6. cages, yaitu representasi dari sebuah *cage* dalam sebuah *grid*.
- Kelas Grid mempunyai beberapa *method*, yaitu:



Gambar 4.3: Diagram kelas untuk perangkat lunak Calcudoku.

1. Grid(int size, int numberOfCages, int[][] 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(int[][] 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(int[][] 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, int[][] 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(int[][] 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(int[][] array), yaitu *method* yang memeriksa apakah *cage assignment* untuk sebuah *cage* *valid* atau tidak. *Method* ini menerima masukan berupa matriks *cage assignment* untuk sebuah *cage* dan menghasilkan keluaran apakah matriks tersebut *valid* atau tidak. Matriks tersebut *valid* jika jumlah area dari *cage* tersebut adalah satu.
8. isCageValid(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. `isCage(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.
20. `solverIsCageValid(int column)`, yaitu *method* yang sama dengan `isCageValid`, tetapi *method* ini hanya untuk dipanggil oleh solver.
21. `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.
22. `solverIsCellValueValid(int column)`, yaitu *method* yang sama dengan `isCellValueValid`, tetapi *method* ini hanya untuk dipanggil oleh solver.
23. `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.
24. `solverSetCellValue(int row, int column, Integer value)`, yaitu *method* yang sama dengan `setCellValue`, tetapi *method* ini hanya untuk dipanggil oleh solver.

25. 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.
26. 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 yang *valid* atau tidak. *Method* ini menghasilkan *null* jika ada sel yang belum diisi.
27. isFilled(), yaitu *method* yang memeriksa apakah semua sel sudah diisi atau tidak. *Method* ini menghasilkan keluaran apakah semua sel sudah diisi atau tidak.
28. 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.
29. getSize(), yaitu *method* untuk mendapatkan ukuran dari *grid*. *Method* ini menghasilkan keluaran berupa ukuran dari *grid*.
30. getNumberOfCages(), yaitu *method* untuk mendapatkan jumlah *cage* yang ada di dalam *grid*. *Method* ini menghasilkan keluaran berupa jumlah *cage* yang ada di dalam *grid*.
31. getCageCells(), yaitu *method* untuk mendapatkan matriks *cage assignment* dari *grid*. *Method* ini menghasilkan keluaran berupa matriks *cage assignment* dari *grid*.
32. 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*.
33. getGridContents(), yaitu *method* untuk mendapatkan nilai dari setiap sel *grid*. *Method* ini menghasilkan keluaran berupa sebuah matriks yang berisi nilai dari setiap sel *grid*.
34. getCages(), yaitu *method* untuk mendapatkan semua *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa array yang berisi semua *cage* dalam *grid*.
35. solveBacktracking(), yaitu *method* untuk memanggil solver untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *backtracking*.
36. solveHybridGenetic(), yaitu *method* untuk memanggil solver untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *hybrid genetic*.

Diagram kelas Grid dapat dilihat pada Gambar 4.4.

4.3.2 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:

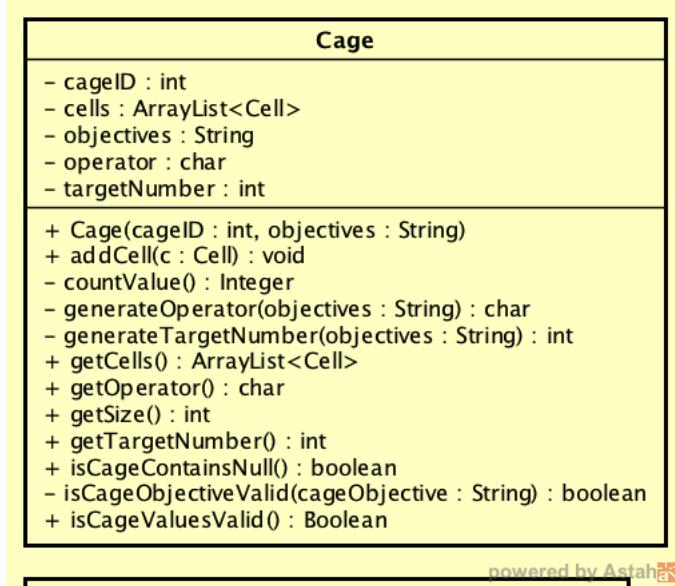


powered by Astah

Gambar 4.4: Diagram kelas Grid.

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*.
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. isCageValid(), 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. getSize(), yaitu *method* untuk mendapatkan jumlah dari sel-sel anggota sebuah *cage*. *Method* ini menghasilkan keluaran berupa 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 4.5.



Gambar 4.5: Diagram kelas Cage.

4.3.3 Kelas Cell

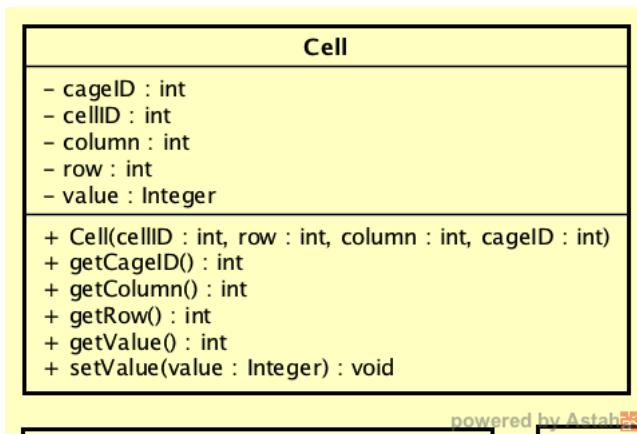
Kelas Cell mempunyai beberapa atribut, yaitu:

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. getRow(), yaitu *method* untuk mendapatkan nomor baris dari sebuah sel. *Method* ini menghasilkan keluaran berupa nomor baris dari sel tersebut.
4. getColumn(), yaitu *method* untuk mendapatkan nomor kolom dari sebuah sel. *Method* ini menghasilkan keluaran berupa nomor kolom dari sel tersebut.
5. getCageID(), yaitu *method* untuk mendapatkan nomor *cage* yang berisi sebuah sel. *Method* ini menghasilkan keluaran berupa nomor *cage* yang berisi sel tersebut.
6. getCellID(), yaitu *method* untuk mendapatkan nomor sel dari sebuah sel. *Method* ini menghasilkan keluaran berupa nomor sel dari sel tersebut.

Diagram kelas Cell dapat dilihat pada Gambar 4.6.



Gambar 4.6: Diagram kelas Cell.

4.3.4 Kelas SolverBacktracking

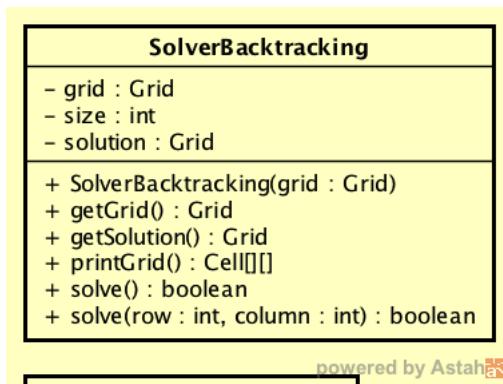
Kelas SolverBacktracking mempunyai beberapa atribut, yaitu:

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. *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.

Diagram kelas SolverBacktracking dapat dilihat pada Gambar 4.7.



Gambar 4.7: Diagram kelas SolverBacktracking.

4.3.5 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*.

Kelas SolverHybridGenetic mempunyai beberapa *method*, yaitu:

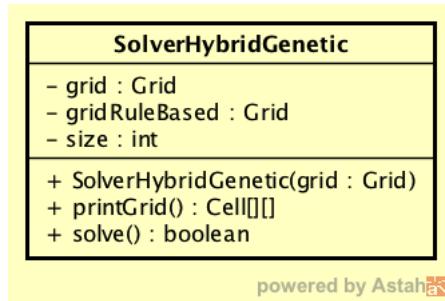
1. SolverHybridGenetic(Grid grid), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa *grid* yang akan diselesaikan oleh *solver* dengan algoritma *hybrid genetic*.
2. solve(), yaitu *method* pembungkus dari *method* solve(int row, int column). *Method* ini menghasilkan keluaran 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. *Method* ini akan memanggil solver dengan algoritma *rule based*. Jika algoritma *rule based* gagal dalam menyelesaikan teka-teki Calcudoku, maka *method* akan memanggil solver dengan algoritma genetik.
4. printGrid(), yaitu *method* untuk mencetak isi *grid* ke layar.

Diagram kelas SolverHybridGenetic dapat dilihat pada Gambar 4.8.

4.3.6 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 4.8: 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*.
2. generatePossibleValuesArray(), yaitu *method* yang membangkitkan kemungkinan angka-angka yang mungkin untuk setiap sel yang ada di dalam *grid*. Angka-angka yang mungkin adalah dari 1 sampai ke ukuran dari *grid*. *Method* ini menghasilkan keluaran berupa array yang menampung semua kemungkinan angka yang mungkin untuk setiap sel yang ada di dalam *grid*.
3. solve(), yaitu *method* yang mencoba untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *rule based*. *Method* ini menghasilkan keluaran apakah *solver* berhasil menyelesaikan teka-teki Calcudoku atau tidak. *Solver* akan mencoba menyelesaikan teka-teki Calcudoku menggunakan aturan-aturan logika, misalnya *single square rule*, *killer combination rule*, *naked subset rule*, *hidden subset rule*, dan *evil twin rule*. Aturan *single square* dan *killer combination* hanya dipakai sekali, dan dilakukan oleh *method* ini, sedangkan aturan *naked subset*, *hidden subset*, dan *evil twin* dapat dipakai berkali-kali, dan dilakukan oleh *method* solveLoop().
4. solveLoop(), yaitu *method* yang mengaplikasikan aturan *naked subset*, *hidden subset*, dan *evil twin* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa *array* kemungkinan angka yang mungkin untuk setiap sel yang ada di dalam *grid*. *Method* ini akan diulang sampai *method* ini tidak bisa mengisi sel-sel yang ada di dalam *grid*.
5. getRowPossibleValues(int rowNumber), yaitu *method* untuk mendapatkan kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam baris yang diminta. *Method* ini menerima masukan berupa nomor baris yang diminta dan menghasilkan keluaran berupa kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam baris yang diminta.
6. getColumnPossibleValues(int rowNumber), yaitu *method* untuk mendapatkan kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam kolom yang diminta. *Method* ini menerima masukan berupa nomor kolom yang diminta dan menghasilkan keluaran berupa kemungkinan angka-angka yang mungkin dari sel-sel yang ada di dalam kolom yang diminta.
7. singleSquare(), yaitu *method* yang mengaplikasikan aturan *single square* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*.
8. killerCombination(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Dalam perangkat lunak ini aturan ini dibatasi hanya untuk *cage* yang berukuran dua sel.

9. killerCombinationCageSize2(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
10. killerCombinationCageSize2GridSize3(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 3×3 yang sedang diselesaikan oleh algoritma *rule based*.
11. killerCombinationCageSize2GridSize4(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 4×4 yang sedang diselesaikan oleh algoritma *rule based*.
12. killerCombinationCageSize2GridSize5(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 5×5 yang sedang diselesaikan oleh algoritma *rule based*.
13. killerCombinationCageSize2GridSize6(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 6×6 yang sedang diselesaikan oleh algoritma *rule based*.
14. killerCombinationCageSize2GridSize7(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 7×7 yang sedang diselesaikan oleh algoritma *rule based*.
15. killerCombinationCageSize2GridSize8(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 8×8 yang sedang diselesaikan oleh algoritma *rule based*.
16. killerCombinationCageSize2GridSize9(), yaitu *method* yang mengaplikasikan aturan *killer combination* kepada setiap *cage* yang berukuran 2 sel di dalam *grid* yang berukuran 9×9 yang sedang diselesaikan oleh algoritma *rule based*.
17. evilTwin(), yaitu *method* yang mengaplikasikan aturan *evil twin* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Dalam perangkat lunak ini aturan ini dibatasi hanya untuk *cage* dengan operator + dan ×.
18. evilTwin(int cageID), yaitu *method* yang mengaplikasikan aturan *evil twin* kepada sebuah *cage* di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*. *Method* ini menerima masukan berupa nomor *grid* yang akan diaplikasikan dengan aturan *evil twin*.
19. nakedSubset(), yaitu *method* yang mengaplikasikan aturan *naked subset* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Dalam perangkat lunak ini aturan ini dibatasi hanya untuk *cage* yang berukuran maksimum dua sel (*naked single* dan *naked double*).
20. 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()*).
21. nakedSingleRow(), yaitu *method* yang mengaplikasikan aturan *naked single* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
22. 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*.

23. `nakedSingleColumn()`, yaitu *method* yang mengaplikasikan aturan *naked single* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
24. `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*.
25. `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()`).
26. `nakedDoubleRow()`, yaitu *method* yang mengaplikasikan aturan *naked double* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
27. `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*.
28. `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.
29. `nakedDoubleColumn()`, yaitu *method* yang mengaplikasikan aturan *naked double* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
30. `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*.
31. `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.
32. `hiddenSubset()`, yaitu *method* yang mengaplikasikan aturan *hidden subset* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Dalam perangkat lunak ini aturan ini dibatasi hanya untuk *cage* yang berukuran maksimum dua sel (*hidden single* dan *hidden double*).
33. `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()`).
34. `hiddenSingleRow()`, yaitu *method* yang mengaplikasikan aturan *hidden single* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.

35. 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*.
36. hiddenSingleColumn(), yaitu *method* yang mengaplikasikan aturan *hidden single* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
37. 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*.
38. hiddenDouble(), yaitu *method* yang mengaplikasikan aturan *hidden double* kepada *grid* yang sedang diselesaikan oleh algoritma *rule based*. Aturan ini diaplikasikan untuk baris (hiddenSingleDouble()) dan untuk kolom (hiddenSingleDouble()).
39. hiddenDoubleRow(), yaitu *method* yang mengaplikasikan aturan *hidden double* kepada baris-baris yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
40. hiddenDoubleRow(int row), yaitu *method* yang mengaplikasikan aturan *hidden 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 *hidden double*.
41. hiddenDoubleRow(int row, ArrayList<Integer> doublePossibleValues, ArrayList<Integer> doublePossibleIndexes), yaitu *method* yang mengaplikasikan aturan *hidden 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 *hidden double*, sebuah *array* yang berisi nilai-nilai, dan sebuah *array* yang berisi nomor-nomor kolom.
42. hiddenDoubleColumn(), yaitu *method* yang mengaplikasikan aturan *hidden double* kepada kolom-kolom yang ada di dalam *grid* yang sedang diselesaikan oleh algoritma *rule based*.
43. hiddenDoubleColumn(int column), yaitu *method* yang mengaplikasikan aturan *hidden 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 *hidden double*.
44. hiddenDoubleColumn(int column, ArrayList<Integer> doublePossibleValues, ArrayList<Integer> doublePossibleIndexes), yaitu *method* yang mengaplikasikan aturan *hidden 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 *hidden double*, sebuah *array* yang berisi nilai-nilai, dan sebuah *array* yang berisi nomor-nomor baris.
45. 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.

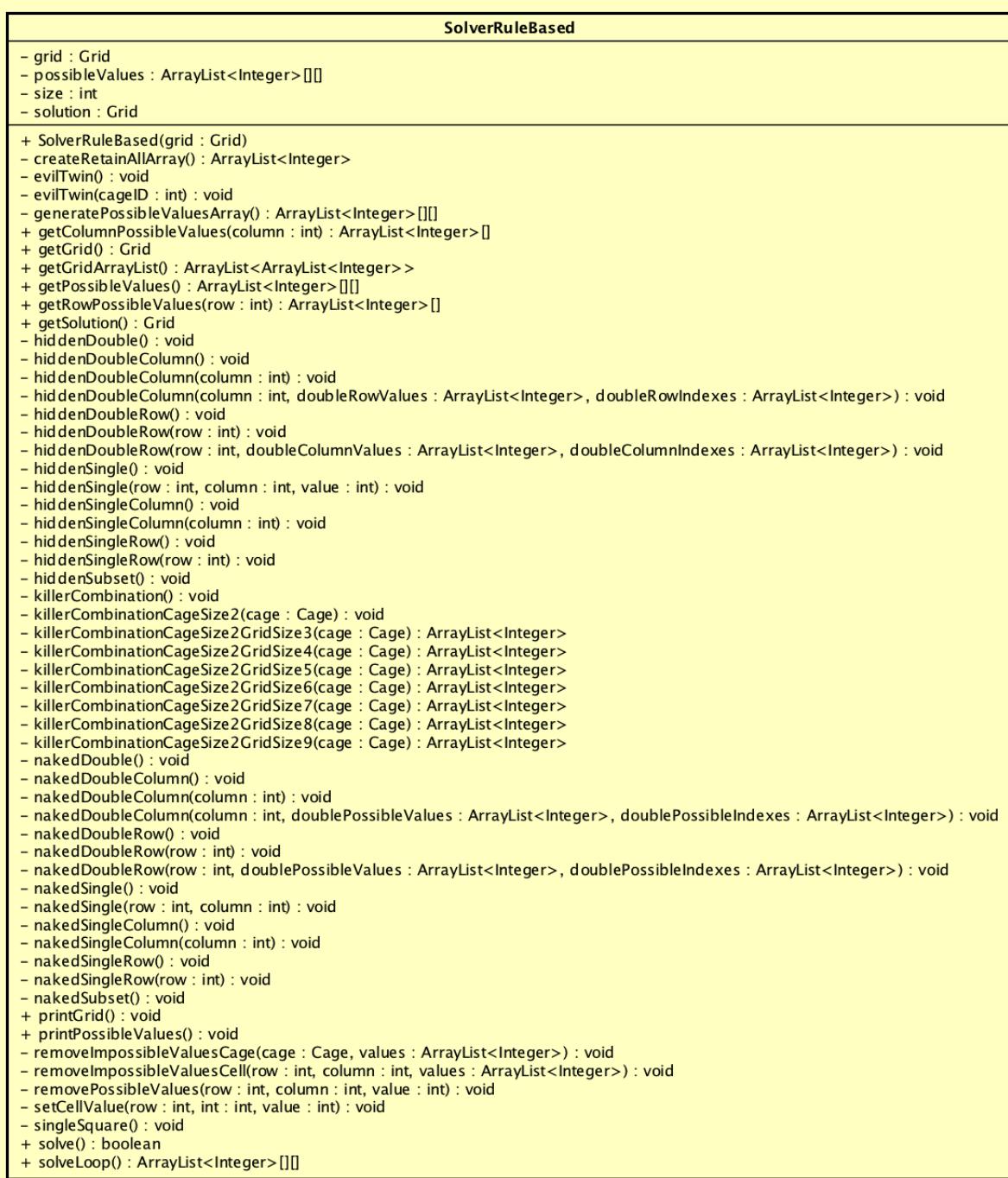
46. removePossibleValues(int row, int column, int value), yaitu *method* untuk menghapus kemungkinan nilai yang sudah digunakan dalam sebuah sel. *Method* ini menghapus nilai tersebut dari baris yang sama dan kolom yang sama. *Method* ini menerima masukan berupa nomor baris, nomor kolom, dan nilai yang akan dihapus dari sel-sel lain dalam baris dan kolom tempat sel tersebut berada.
47. removeImpossibleValuesCage(Cage cage, ArrayList<Integer> values), yaitu *method* untuk menghabus kemungkinan nilai yang tidak mungkin dari sel-sel di dalam sebuah *cage*. Method ini menerima masukan berupa sebuah *cage* dan sebuah *array* yang berisi nilai-nilai yang mungkin.
48. removeImpossibleValuesCell(int row, int column, ArrayList<Integer> values), yaitu *method* untuk menghabus kemungkinan nilai yang tidak mungkin dari sebuah sel yang diminta. Method ini menerima masukan berupa nomor baris dan nomor kolom dari sel yang diminta, dan sebuah *array* yang berisi nilai-nilai yang mungkin.
49. 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*.
50. getGridArrayList(), yaitu *method* untuk mendapatkan isi dari *grid* dalam bentuk *ArrayList*. Method ini menghasilkan keluaran berupa isi dari *grid* dalam bentuk *ArrayList*.
51. getGrid, yaitu *method* untuk mendapatkan *grid*. Method ini menghasilkan keluaran berupa *grid*.
52. 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*.
53. 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*.
54. printGrid(), yaitu *method* untuk mencetak isi *grid* ke layar.
55. 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 4.9.

4.3.7 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. generationsNumber, yaitu jumlah generasi maksimum yang akan dibangkitkan oleh algoritma genetik.



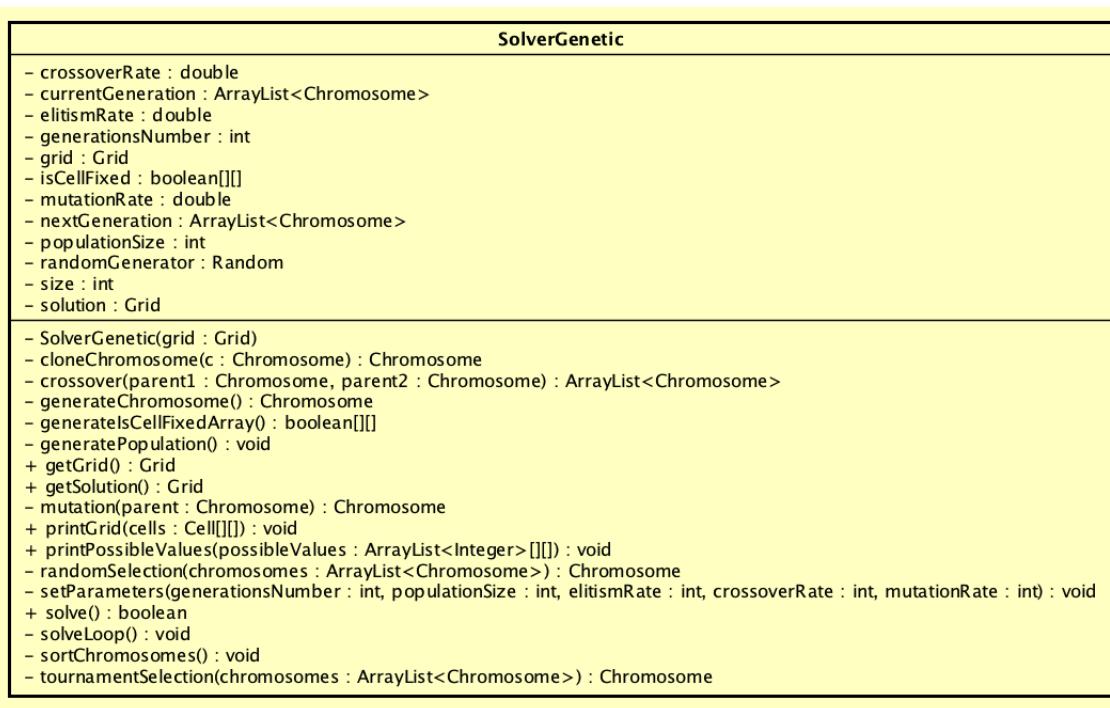
powered by Astah

Gambar 4.9: Diagram kelas SolverRuleBased.

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)`, 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 generationsNumber, 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 sebuah kromosom. *Method* ini menghasilkan keluaran berupa sebuah kromosom.
7. `sortChromosomes()`, yaitu *method* yang mengurutkan kromosom-kromosom dalam generasi saat ini berdasarkan nilai kelayakannya.
8. `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.
9. `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.



powered by Astah

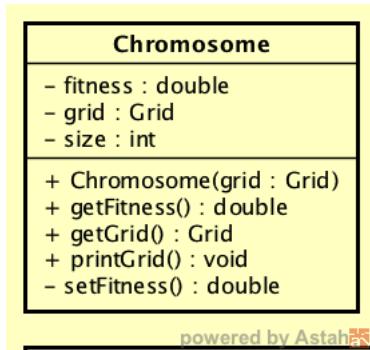
Gambar 4.10: Diagram kelas SolverGenetic.

10. `cloneChromosome(Chromosome c)`, yaitu *method* untuk mengkopi sebuah kromosom. *Method* ini menerima masukan berupa kromosom yang akan dikopi dan menghasilkan keluaran berupa kromosom baru hasil kopasi dari kromosom yang dikopi tersebut.
11. `crossover(Chromosome parent1, Chromosome parent2)`, yaitu *method* yang mengaplikasikan operator kawin silang kepada dua kromosom. *Method* ini menerima masukan berupa dua kromosom yang akan dikawinsilangkan dan menghasilkan keluaran berupa sebuah `ArrayList` yang berisi dua kromosom hasil kawin silang.
12. `mutation(Chromosome parent)`, yaitu *method* yang mengaplikasikan operator mutasi kepada sebuah kromosom. *Method* ini menerima masukan berupa kromosom yang akan dimutasi dan menghasilkan keluaran berupa kromosom hasil mutasi.
13. `getGrid`, yaitu *method* untuk mendapatkan *grid*. *Method* ini menghasilkan keluaran berupa *grid*.
14. `getSolution`, yaitu *method* untuk mendapatkan *grid* yang sudah diselesaikan oleh algoritma genetik. *Method* ini menghasilkan keluaran berupa *grid* yang sudah diselesaikan oleh algoritma genetik.
15. `printGrid()`, yaitu *method* untuk mencetak isi *grid* ke layar.
16. `printPossibleValues()`, yaitu *method* untuk mencetak kemungkinan angka-angka yang valid untuk setiap sel di dalam *grid* ke layar.

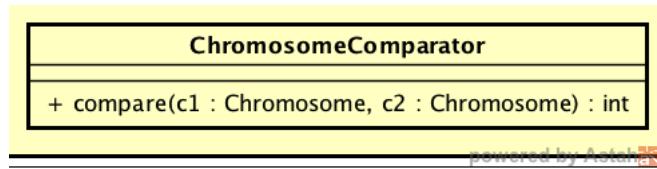
Diagram kelas SolverGenetic dapat dilihat pada Gambar 4.10.

4.3.8 Kelas Chromosome

Kelas Chromosome mempunyai beberapa atribut, yaitu:



Gambar 4.11: Diagram kelas Chromosome.



Gambar 4.12: Diagram kelas ChromosomeComparator.

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

Kelas Chromosome mempunyai beberapa *method*, yaitu:

1. Chromosome(Grid grid), yaitu konstruktor dari kelas ini. Konstruktor ini menerima masukan berupa sebuah *grid* yang sudah diisi dengan angka-angka secara acak.
2. setFitness(), yaitu *method* yang menghitung nilai kelayakan untuk sebuah *grid*. *Method* ini menghasilkan keluaran berupa nilai kelayakan untuk *grid* tersebut.
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 4.11.

4.3.9 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 4.12.

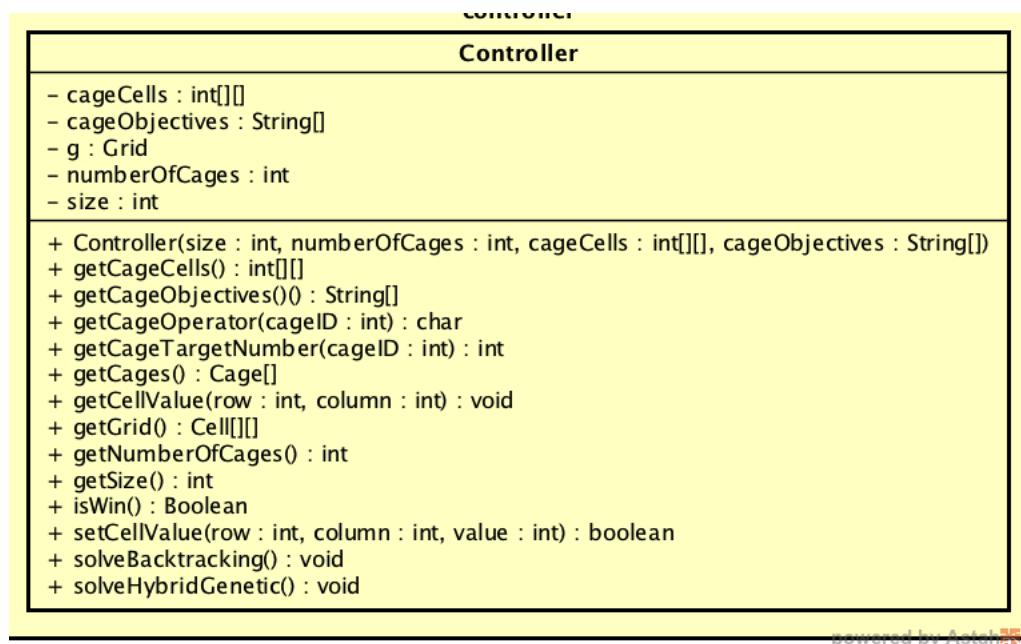
4.3.10 Kelas Controller

Kelas Controller mempunyai beberapa atribut, yaitu:

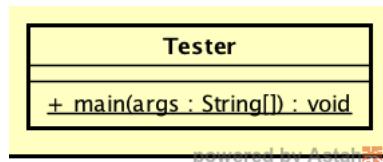
1. size, yaitu ukuran dari matriks *grid*.
2. numberCages, yaitu banyaknya *cage* yang terdapat dalam *grid*.
3. cageCells, yaitu sebuah matriks *cage assignment*. Matriks ini merepresentasikan posisi dari setiap *cage* dalam *grid*.
4. cageObjectives, yaitu sebuah *array* yang berisi *cage objectives* untuk setiap *cage*. *Cage objectives* berisikan angka tujuan dan operasi matematika yang telah ditentukan.
5. 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(int size, int numberCages, int[][] 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. 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 yang *valid* atau tidak. *Method* ini menghasilkan *null* jika ada sel yang belum diisi.
4. isFilled(), yaitu *method* yang memeriksa apakah semua sel sudah diisi atau tidak. *Method* ini menghasilkan keluaran apakah semua sel sudah diisi atau tidak.
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*.



Gambar 4.13: Diagram kelas Controller.



Gambar 4.14: Diagram kelas Tester.

11. `getCages()`, yaitu *method* untuk mendapatkan semua *cage* dalam *grid*. *Method* ini menghasilkan keluaran berupa array yang berisi semua *cage* dalam *grid*.
12. `solveBacktracking()`, yaitu *method* untuk memanggil solver untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *backtracking*.
13. `solveHybridGenetic()`, yaitu *method* untuk memanggil solver untuk menyelesaikan teka-teki Calcudoku menggunakan algoritma *hybrid genetic*.

Diagram kelas Controller dapat dilihat pada Gambar 4.13.

4.3.11 Kelas Tester

Kelas Tester tidak mempunyai variabel, tetapi kelas ini mempunyai satu *method*, yaitu `main(String[] args)`. Fungsi dari *method* ini adalah untuk menjalankan program ini. Diagram kelas Tester dapat dilihat pada Gambar 4.14.

DAFTAR REFERENSI

- [1] Fahda, A. (2015) Kenken puzzle solver using backtracking algorithm. Makalah IF2211 Strategi Algoritma - Semester II Tahun 2014/2015, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2014-2015/Makalah2015/Makalah_IF221_Strategi_Algoritma_2015_016.pdf.
- [2] 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.