Perbandingan Algoritma *Backtracking* dan Algoritma *Hybrid Genetic* untuk Menyelesaikan Permainan Calcudoku

Who? Michael Adrian
2013730039
michaeladrian39@gmail.com

From? Program Studi Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan

When? 20 Desember 2017

Perbandingan Algoritma *Backtracking* dan Algoritma *Hybrid Genetic* untuk Menyelesaikan Permainan Calcudoku

Who? Michael Adrian
2013730039
michaeladrian39@gmail.com

From? Program Studi Teknik Informatika
Fakultas Teknologi Informasi dan Sains
Universitas Katolik Parahyangan

When? 20 Desember 2017

Calcudoku

- Salah satu jenis permainan teka-teki aritmatika dan logika
- Dikenal juga sebagai KenKen, KenDoku, atau Mathdoku

Calcudoku
- Salah satu jenis permainan teka-teki aritmatika dan logika
- Dikenal juga sebagai KenKen, KenDoku, atau Mathdoku

Sebagai salah satu jenis permainan teka-teki aritmatika dan *grid*, Calcudoku, atau dikenal juga sebagai KenKen, KenDoku, atau Mathdoku.

Aturan Permainan

- Pemain diberikan sebuah *grid* dengan ukuran $n \times n$
- n biasanya antara 3 sampai dengan 9
- *Grid* ini harus diisi dengan angka 1 sampai dengan n
- Dalam setiap baris setiap angka hanya muncul sekali
- Dalam setiap kolom setiap angka hanya muncul sekali
- *Grid* dibagi ke dalam *cage*
- *Cage* adalah sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan
- 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*

Aturan Permainan
- Pemain diberikan sebuah *grid* dengan ukuran $n \times n$
- n biasanya antara 3 sampai dengan 9
- *Grid* ini harus diisi dengan angka 1 sampai dengan n
- Dalam setiap baris setiap angka hanya muncul sekali
- Dalam setiap kolom setiap angka hanya muncul sekali
- *Grid* dibagi ke dalam *cage*
- *Cage* adalah sekelompok sel yang dibatasi oleh garis yang lebih tebal daripada garis pembatas antar sel dengan angka tujuan dan operator yang telah ditentukan
- 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*

Seperti dalam Sudoku, dalam teka-teki ini, pemain diberikan sebuah *grid* dengan ukuran $n \times n$, dengan n biasanya antara 3 sampai dengan 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 dengan 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*.

Operator-Operator Matematika

- Ada 5 kemungkinan operator:
  - $+$ (penjumlahan)
  - - (pengurangan)
  - $\times$ (perkalian)
  - $\div$ (pembagian)
  - $=$ (sama dengan)
- Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan *cage* harus berukuran dua sel

Operator-Operator Matematika
- Ada 5 kemungkinan operator:
  - + (penjumlahan)
  - - (pengurangan)
  - × (perkalian)
  - ÷ (pembagian)
  - = (sama dengan)
- Jika operasi matematika yang ditentukan adalah pengurangan atau pembagian, maka ukuruan cage harus berukuran dua sel

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 ukuruan *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-tekinya.

Contoh Permainan

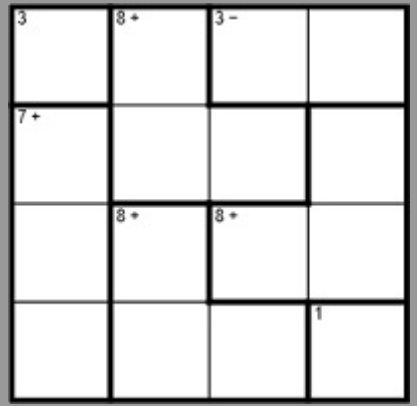


Figure 1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan.

Contoh Permainan

Figure 1: Contoh permainan teka-teki dengan ukuran *grid* 4 x 4 yang belum diselesaikan.

Contoh Solusi

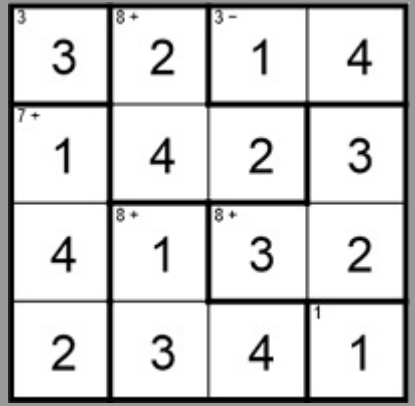


Figure 2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 1.

Contoh Solusi
3 2 1 4
1 4 2 3
4 1 3 2
2 3 4 1
Figure 2: Solusi untuk permainan teka-teki Calcudoku yang diberikan pada Gambar 1.

Algoritma *Backtracking*

- 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*
- 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* 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*).

Cara Kerja Algoritma *Backtracking* Secara Singkat

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

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 untul sel tersebut
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan

Cara Kerja Algoritma *Backtracking* Secara Singkat
Singkatnya, langkah-langkah dasar dari implementasi algoritma *backtracking* dapat dijelaskan sebagai berikut:
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 untul sel tersebut
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan

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 untul sel tersebut.
5. Jika tidak ada lagi sel yang kosong, solusi sudah ditemukan.

Algoritma *Hybrid Genetic*

- 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

Algoritma *Hybrid Genetic*

- 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

Dalam kasus ini, algoritma ini 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.

Algoritma *Rule Based*

- Sebuah algoritma berbasis aturan logika untuk menyelesaikan teka-teki Sudoku dan variasinya, termasuk Calcudoku
- Beberapa aturan logika yang digunakan dalam algoritma ini adalah:
  - *Single square rule*
  - *Naked subset rule*
  - *Hidden single rule*
  - *Killer combination*

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].

Algoritma Genetik

- Salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam
- Perpaduan dari bidang biologi dan ilmu komputer.
- Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom.

Algoritma Genetik

- Salah satu teknik heuristik *Generate and Test* yang terinspirasi oleh sistem seleksi alam
- Perpaduan dari bidang biologi dan ilmu komputer.
- Algoritma ini memanipulasi informasi, biasanya disebut sebagai kromosom.

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.

Cara Kerja Algoritma Genetik

- Cara kerja algoritma genetik adalah sebagai berikut:
  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 algoritm.

Cara Kerja Algoritma Genetik
- Cara kerja algoritma genetik adalah sebagai berikut:
  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 algoritm.

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.

Cara Kerja Algoritma *Hybrid Genetic*

- Cara kerja algoritma *hybrid genetic* adalah sebagai berikut:
  - Masukkan teka-teki yang akan diselesaikan sebagai input.
  - Program akan merepresentasikan input yang dimasukkan dalam format teka-teki.
  - Program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based*, maka algoritma selesai.
  - Jika program gagal dengan menggunakan algoritma *rule based*, maka program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik, maka algoritma selesai.
  - Jika program gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, artinya algoritma gagal dalam menyelesaikan teka-teki terseebut.

Cara Kerja Algoritma *Hybrid Genetic*
- Cara kerja algoritma *hybrid genetic* adalah sebagai berikut:
  - Masukkan teka-teki yang akan diselesaikan sebagai input.
  - Program akan merepresentasikan input yang dimasukkan dalam format teka-teki.
  - Program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based*, maka algoritma selesai.
  - Jika program gagal dengan menggunakan algoritma *rule based*, maka program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
  - Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik, maka algoritma selesai.
  - Jika program gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, artinya algoritma gagal dalam menyelesaikan teka-teki terseebut.

Cara kerja algoritma *hybrid genetic* menurut Johanna, Lukas, dan Saputra adalah sebagai berikut [2]:

- Masukkan teka-teki yang akan diselesaikan sebagai input.
- Program akan merepresentasikan input yang dimasukkan dalam format teka-teki.
- Program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based* terlebih dahulu.
- Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma *rule based*, maka algoritma selesai.
- Jika program gagal dengan menggunakan algoritma *rule based*, maka program akan mencoba menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik.
- Jika program berhasil menyelesaikan teka-teki tersebut dengan menggunakan algoritma genetik, maka algoritma selesai.
- Jika program gagal dalam menyelesaikan teka-teki tersebut setelah menggunakan algoritma genetik, artinya algoritma gagal dalam menyelesaikan teka-teki terseebut.

Alur Cara Kerja Algoritma *Hybrid Genetic*

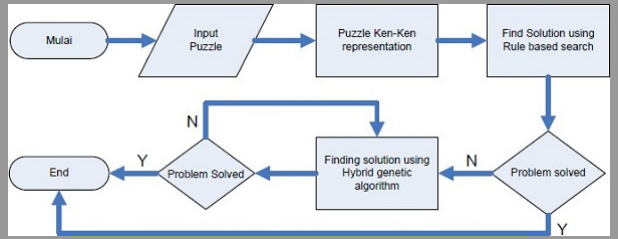


Figure 3: Alur penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic*

- Alur (*flow chart*) penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* dapat dilihat di Gambar 3.

Alur Cara Kerja Algoritma *Hybrid Genetic*
Alur (*flow chart*) penyelesaian permainan teka-teki Calcudoku dengan menggunakan algoritma *hybrid genetic* dapat dilihat di Gambar 3.

Analisis Perangkat Lunak

- Perangkat lunak ini akan menerima masukan dalam bentuk *file* yang berisi:
  1. Ukuran *grid*
  2. Jumlah *cage*
  3. Matriks *cage assignment*
  4. Matriks *cage objectives*
- Perangkat lunak ini akan menghasilkan keluaran berupa GUI permainan teka-teki Calcudoku berdasarkan isi *file* yang di-*load* oleh pengguna.
- Permainan ini dapat diselesaikan oleh pengguna, atau menggunakan salah satu dari dua *solver* yang disediakan. Kedua *solver* tersebut yaitu:
  1. Algoritma *backtracking*, dan
  2. Algoritma *hybrid genetic*.

Analisis Perangkat Lunak

- Perangkat lunak ini akan menerima masukan dalam bentuk *file* yang berisi:
  1. Ukuran *grid*
  2. Jumlah *cage*
  3. Matriks *cage assignment*
  4. Matriks *cage objectives*
- Perangkat lunak ini akan menghasilkan keluaran berupa GUI permainan teka-teki Calcudoku berdasarkan isi *file* yang di-*load* oleh pengguna.
- Permainan ini dapat diselesaikan oleh pengguna, atau menggunakan salah satu dari dua *solver* yang disediakan. Kedua *solver* tersebut yaitu:
  1. Algoritma *backtracking*, dan
  2. Algoritma *hybrid genetic*.

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*.

Diagram *Use Case*

- Diagram *use case* adalah diagram yang menggambarkan interaksi antara sistem (perangkat lunak) dengan pengguna.
- Skenario-skenario yang dapat dilakukan oleh pengguna adalah:
  1. Membuka file masukan.
  2. Memilih salah satu dari dua *solver* yang disediakan.
  3. Me-*reset* permainan.
  4. Memeriksa permainan.
  5. Menutup *file* masukan.
  6. Menyelesaikan permainan dengan usahanya sendiri.
  7. Mengatur nilai dari parameter-parameter untuk algoritma genetik.
- Diagram *use case* berdasarkan skenario-skenario tersebut dapat dilihat pada Gambar 4.

Diagram *Use Case*

- Diagram *use case* adalah diagram yang menggambarkan interaksi antara sistem (perangkat lunak) dengan pengguna.
- Skenario-skenario yang dapat dilakukan oleh pengguna adalah:
  1. Membuka file masukan.
  2. Memilih salah satu dari dua *solver* yang disediakan.
  3. Me-*reset* permainan.
  4. Memeriksa permainan.
  5. Menutup *file* masukan.
  6. Menyelesaikan permainan dengan usahanya sendiri.
  7. Mengatur nilai dari parameter-parameter untuk algoritma genetik.
- Diagram *use case* berdasarkan skenario-skenario tersebut dapat dilihat pada Gambar 4.

Diagram *use case* adalah diagram yang menggambarkan interaksi antara sistem (perangkat lunak) dengan pengguna.
Skenario-skenario yang dapat dilakukan oleh pengguna adalah:

1. Membuka file masukan untuk memulai permainan.
2. Memilih salah satu dari dua *solver* yang disediakan untuk menyelesaikan permainan berdasarkan *file* yang sudah di-*load*.
3. Me-*reset* permainan untuk mengulang permainan berdasarkan *file* masukan yang sudah di-*load* dari awal.
4. Meminta perangkat lunak untuk memeriksa permainan jika ada masukan yang salah di dalam *grid*.
5. Menutup *file* masukan untuk mengakhiri permainan.
6. Menyelesaikan permainan dengan usahanya sendiri.
7. Mengatur nilai dari parameter-parameter untuk algoritma genetik.

Diagram *use case* berdasarkan skenario-skenario tersebut dapat dilihat pada Gambar 4.

Diagram *Use Case*

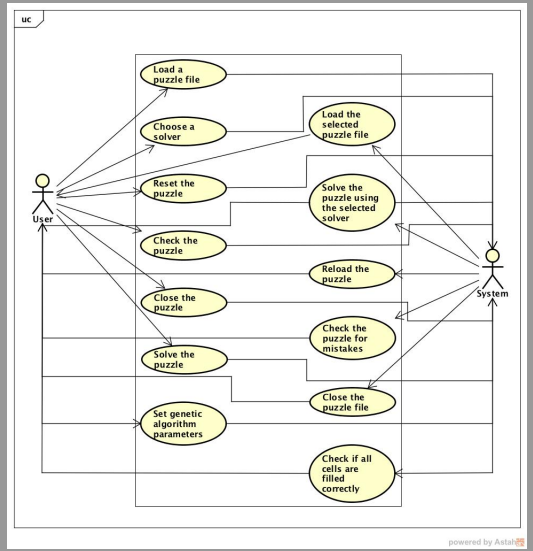


Figure 4: Diagram *use case* untuk perangkat lunak permainan teka-teki Calcudoku

Diagram *Use Case*

Figure 4: Diagram use case untuk perangkat lunak permainan teka-teki Calcudoku

Perancangan Masukan

- Masukan untuk perangkat lunak Calcudoku ini berupa sebuah *file* teks, seperti yang ditunjukkan pada Gambar 5.

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

Figure 5: Contoh *file* masukan.

Perancangan Masukan
Masukan untuk perangkat lunak Calcudoku ini berupa sebuah *file* teks, seperti yang ditunjukkan pada Gambar 5.
Figure 5: Contoh file masukan.

Masukan untuk perangkat lunak permainan teka-teki Calcudoku ini berupa sebuah *file* teks, seperti yang ditunjukkan pada Gambar 5.

Perancangan Masukan

- Adapun rincian dari *file* teks masukan tersebut adalah sebagai berikut:
  1. Baris pertama berisi ukuran *grid* dan banyaknya *cage* dari teka-teki Calcudoku tersebut.
  2. Baris kedua sampai ke baris ke-$2 + (n - 1)$, dengan n adalah ukuran *grid*, berisi matriks *cage assignment*.
  3. Baris ke-$2 + n$ dan seterusnya berisi *cage objectives* untuk setiap *cage*.

Perancangan Masukan

- Adapun rincian dari *file* teks masukan tersebut adalah sebagai berikut:
  1. Baris pertama berisi ukuran *grid* dan banyaknya *cage* dari teka-teki Calcudoku tersebut.
  2. Baris kedua sampai ke baris ke-$2 + (n - 1)$, dengan n adalah ukuran *grid*, berisi matriks *cage assignment*.
  3. Baris ke-$2 + n$ dan seterusnya berisi *cage objectives* untuk setiap *cage*.

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

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

Perancangan Keluaran

- Keluaran untuk perangkat lunak Calcudoku ini berupa sebuah matriks yang berisi solusi dari teka-teki Calcudoku yang sudah diselesaikan oleh program, seperti dapat dilihat pada Gambar 6.

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

Figure 6: Contoh keluaran.

Perancangan Keluaran

- Keluaran untuk perangkat lunak Calcudoku ini berupa sebuah matriks yang berisi solusi dari teka-teki Calcudoku yang sudah diselesaikan oleh program, seperti dapat dilihat pada Gambar 6.

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

Figure 6: Contoh 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 6.

Perancangan Antarmuka

- Antarmuka ini terdiri dari sebuah *frame* yang berisi sebuah menu *bar* dan GUI dari Calcudoku.
- GUI hanya akan ditampilkan jika sudah membuka *file* permainan. Jika *file* permainan ditutup, maka GUI juga akan ditutup.
- Menu *bar* terdiri dari dua menu, yaitu:
  1. *File*
  2. *Solve*

Perancangan Antarmuka

- Antarmuka ini terdiri dari sebuah *frame* yang berisi sebuah menu *bar* dan GUI dari Calcudoku.
- GUI hanya akan ditampilkan jika sudah membuka *file* permainan. Jika *file* permainan ditutup, maka GUI juga akan ditutup.
- Menu *bar* terdiri dari dua menu, yaitu:
  1. *File*
  2. *Solve*

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

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

Perancangan Antarmuka

- Gambar 7 menunjukkan perancangan GUI sebelum *file* permainan dibuka.

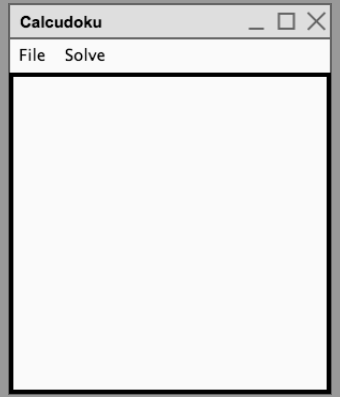


Figure 7: Perancangan GUI sebelum membuka *file* permainan

Gambar 7 menunjukkan perancangan GUI sebelum *file* permainan dibuka.

Perancangan Antarmuka

- Gambar 8 menunjukkan perancangan GUI sesudah *file* permainan dibuka.

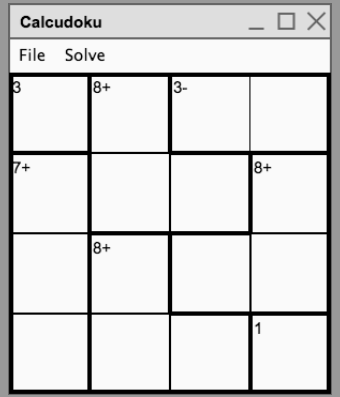


Figure 8: Perancangan GUI sesudah membuka *file* permainan

Gambar 8 menunjukkan perancangan GUI sesudah *file* permainan dibuka.

Perancangan Antarmuka

- Gambar 9 menunjukkan perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

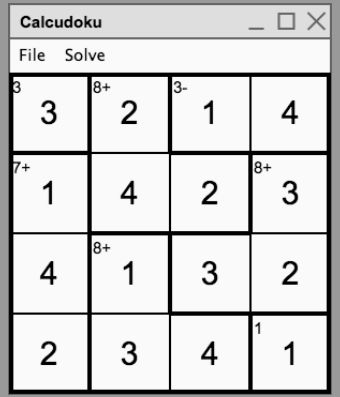


Figure 9: Perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

Perancangan Antarmuka
- Gambar 9 menunjukkan perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

Figure 9: Perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

Gambar 9 menunjukkan perancangan GUI sesudah permainan berdasarkan *file* permainan yang dibuka diselesaikan.

Perancangan Antarmuka - Menu *File*

- Menu *File* mempunyai beberapa menu *item*, yaitu:
  1. *Load Puzzle File*
  2. *Reset Puzzle*
  3. *Close Puzzle File*
  4. *Check Puzzle*
  5. *Exit*
- Gambar 10 menunjukkan isi dari menu *File*.

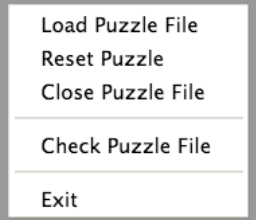


Figure 10: Menu *File*

Perancangan Antarmuka - Menu *File*
Menu *File* mempunyai beberapa menu *item*, yaitu:
1. *Load Puzzle File*
2. *Reset Puzzle*
3. *Close Puzzle File*
4. *Check Puzzle*
5. *Exit*
Gambar 10 menunjukkan isi dari menu *File*.
Load Puzzle File
Reset Puzzle
Close Puzzle File
Check Puzzle File
Exit
Figure 10: Menu *File*

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

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

Gambar 10 menunjukkan isi dari menu *File*.

Perancangan Antarmuka - Menu *Solve*

- Menu *Solve* mempunyai beberapa menu *item*, yaitu:
  1. *Backtracking*
  2. *Hybrid Genetic*
  3. *Set Genetic Algorithm Parameters*
- Gambar 11 menunjukkan isi dari menu *Solve*.

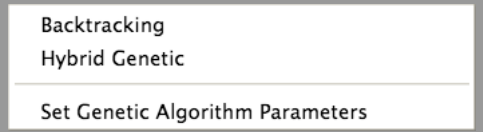


Figure 11: Menu *Solve*

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

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

Gambar 11 menunjukkan isi dari menu *Solve*.

Diagram Kelas

- Perangkat lunak Calcudoku ini terdiri dari beberapa kelas, yang dikelompokkan dalam tiga package, yaitu:
  1. Model
  2. View
  3. Controller
- Diagram kelas untuk perangkat lunak ini dapat dilihat pada Gambar 12.

Diagram Kelas

- Perangkat lunak Calcudoku ini terdiri dari beberapa kelas, yang dikelompokkan dalam tiga package, yaitu:
  1. Model
  2. View
  3. Controller
- Diagram kelas untuk perangkat lunak ini dapat dilihat pada Gambar 12.

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

1. Model, yaitu *engine* dari perangkat lunak ini.
2. View, yaitu GUI dari perangkat lunak ini.
3. Controller, yaitu penghubung antara *package* model dan *package* view

Diagram kelas untuk perangkat lunak ini dapat dilihat pada Gambar 12.

Diagram Kelas

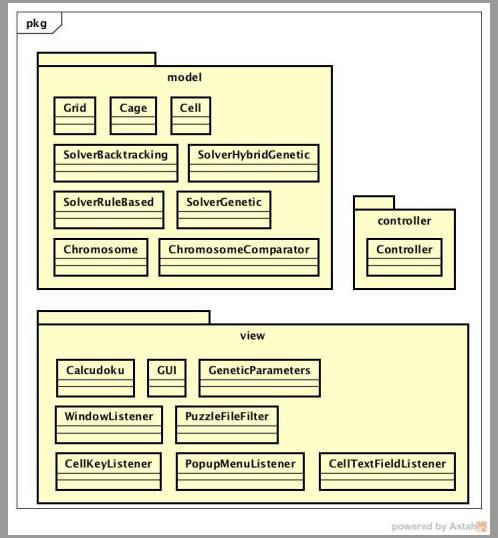


Figure 12: Diagram kelas untuk perangkat lunak Calcudoku.

Diagram Kelas

Figure 12: Diagram kelas untuk perangkat lunak Calcudoku.

Diagram Kelas - *Package* Model

- *Package* model memiliki beberapa kelas, yaitu:
  1. Grid
  2. Cell
  3. Cage
  4. SolverBacktracking
  5. SolverHybridGenetic
  6. SolverRuleBased

Diagram Kelas - *Package* Model
- *Package* model memiliki beberapa kelas, yaitu:
  - Grid
  - Cell
  - Cage
  - SolverBacktracking
  - SolverHybridGenetic
  - SolverRuleBased

Package model memiliki beberapa kelas, yaitu:

1. Grid, yaitu kelas yang merepresentasikan *grid* dalam teka-teki Calcudoku.
2. Cell, yaitu kelas yang merepresentasikan sel dalam teka-teki Calcudoku.
3. Cage, yaitu kelas yang merepresentasikan *cage* dalam teka-teki Calcudoku.
4. SolverBacktracking, yaitu kelas *solver* untuk teka-teki Calcudoku menggunakan algoritma backtracking.
5. 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.
6. 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.

Diagram Kelas - *Package* Model dan Controller

- *Package* model memiliki beberapa kelas, yaitu:
  7. SolverGenetic
  8. Chromosome
  9. ChromosomeComparator
- *Package* controller hanya memiliki satu kelas, yaitu kelas Controller.

Package model memiliki beberapa kelas, yaitu:

7. 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.
8. Chromosome, yaitu kelas yang merepresentasikan sebuah kromosom untuk algoritma genetik dalam solver *hybrid genetic*.
9. ChromosomeComparator, yaitu kelas pembanding *custom* (*custom comparator*) yang berfungsi untuk mengurutkan kromosom berdasarkan nilai kelayakkannya (*fitness value*).

Package model memiliki satu kelas, yaitu kelas Controller. Kelas ini berfungsi untuk menghubungkan kelas-kelas dalam *package* model dengan kelas-kelas dalam *package* view.

Diagram Kelas - *Package* View

- *Package* view memiliki beberapa kelas, yaitu:

1. Calcudoku
2. PuzzleFileFilter
3. GUI
4. CellKeyListener
5. PopupMenuListener
6. CellTextFieldListener
7. GeneticParameters

Diagram Kelas - *Package* View
- *Package* view memiliki beberapa kelas, yaitu:
1. Calcudoku
2. PuzzleFileFilter
3. GUI
4. CellKeyListener
5. PopupMenuListener
6. CellTextFieldListener
7. GeneticParameters

Package view memiliki beberapa kelas, yaitu:

1. Calcudoku, yaitu kelas *frame* yang berisi menu *bar* dan instansiasi kelas panel GUI.
2. WindowListener, yaitu kelas *listener* untuk kelas Calcudoku. *Listener* ini berfungsi untuk menambahkan pesan peringatan saat akan menutup perangkat lunak.
3. PuzzleFileFilter, yaitu kelas filter untuk *file chooser*. Filter ini membatasi agar *file chooser* hanya bisa membuka *file* teks.
4. GUI, yaitu kelas panel yang merepresentasikan GUI dari permainan teka-teki Calcudoku.
5. CellKeyListener, yaitu kelas *listener* untuk kelas GUI. *Listener* ini berfungsi untuk menggerakkan kursor dari sebuah sel ke sel di sebelahnya menggunakan tombol-tombol panah ke kiri, ke atas, ke bawah, dan ke kanan. Listener ini juga berfungsi untuk membatasi agar sel hanya bisa diisi oleh satu angka.
6. PopupMenuListener, yaitu kelas *listener* untuk kelas GUI. *Listener* ini berfungsi untuk mengisi sel dengan angka menggunakan menu *pop up*, sel akan diisi dengan angka yang dipilih.
7. CellTextFieldListener, yaitu kelas *listener* untuk kelas GUI. *Listener* ini berfungsi untuk mengisikan sel dalam kelas Grid dengan angka yang diisikan ke dalam sel dalam GUI.

Kode Program - Algoritma *Backtracking*

- Isilah mulai dari sel pada sudut kiri atas.
- Algoritma *backtracking* selesai jika semua sel sudah terisi dengan benar.

```
public boolean solve()
{
    if (solve(0, 0) == true)
    {
        this.solution = grid;
        ...
        return true;
    }
    else
    {
        return false;
    }
}
```


Kode Program - Algoritma *Backtracking*
- Isilah mulai dari sel pada sudut kiri atas.
- Algoritma *backtracking* selesai jika semua sel sudah terisi dengan benar.

Isilah mulai dari sel pada sudut kiri atas. Algoritma selesai jika semua sel sudah terisi dengan benar.

Kode Program - Algoritma *Backtracking*

- Jika sudah mengisi sel yang paling kanan, isilah sel yang paling kiri pada baris berikutnya.

```
private boolean solve(int row, int column)
{
    if (column >= size)
    {
        column = 0;
        row++;
        if (row >= size)
        {
            ...
            return true;
        }
    }
    ...
```

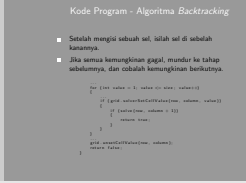

Kode Program - Algoritma *Backtracking*
Jika sudah mengisi sel yang paling kanan, isilah sel yang paling kiri pada baris berikutnya.

Jika sudah mengisi sel yang paling kanan, isilah sel paling kiri pada baris berikutnya.

Kode Program - Algoritma *Backtracking*

- Setelah mengisi sebuah sel, isilah sel di sebelah kanannya.
- Jika semua kemungkinan gagal, mundur ke tahap sebelumnya, dan cobalah kemungkinan berikutnya.

```
        ...
        for (int value = 1; value <= size; value++)
        {
            ...
            if (grid.solverSetCellValue(row, column, value))
            {
                if (solve(row, column + 1))
                {
                    return true;
                }
            }
        }
        ...
        grid.unsetCellValue(row, column);
        return false;
    }
```

Setelah mengisi sebuah sel, isilah sel di sebelah kanannya. Jika semua kemungkinan gagal, mundur ke tahap sebelumnya, dan cobalah kemungkinan berikutnya

Kode Program - Algoritma *Hybrid Genetic*

- *Solver* akan mencoba menyelesaikan permainan dengan algoritma *rule based*.

```
public boolean solve()
{
    SolverRuleBased srb = new SolverRuleBased(grid);
    boolean isFilled = srb.solve();
    this.gridRuleBased = srb.getGrid();
    if (isFilled)
    {
        this.solution = srb.getSolution();
        ...
        return true;
    }
    ...
```


Kode Program - Algoritma *Hybrid Genetic*
- *Solver* akan mencoba menyelesaikan permainan dengan algoritma *rule based*.

Solver akan mencoba menyelesaikan permainan dengan algoritma *rule based*.

Kode Program - Algoritma *Hybrid Genetic*

- Jika algoritma *rule based* tidak berhasil mengisi semua sel dalam *grid* dengan benar, maka *solver* akan mencoba menyelesaikan permainan dengan algoritma genetik.

```
        ...
        else
        {
            SolverGenetic sg = new SolverGenetic(gridRuleBased,
                    generations, populationSize, elitismRate,
                    crossoverRate, mutationRate);
            if (sg.solve() == true)
            {
                this.solution = sg.getSolution();
                ...
                return true;
            }
        }
        return false;
    }
```


Kode Program - Algoritma *Hybrid Genetic*
Jika algoritma *rule based* tidak berhasil mengisi semua sel dalam *grid* dengan benar, maka *solver* akan mencoba menyelesaikan permainan dengan algoritma genetik.

Jika algoritma *rule based* tidak berhasil mengisi semua sel dalam *grid* dengan benar, maka *solver* akan mencoba menyelesaikan permainan dengan algoritma genetik.

Kode Program - Algoritma *Rule Based*

- Algoritma *rule based* dimulai dengan mengaplikasikan aturan logika *single square* dan aturan logika *killer combination*.
- Kedua aturan logika ini hanya diaplikasikan sekali, yaitu di awal algoritma.

```
public boolean solve()
{
    singleSquare();
    killerCombination();;
    ...
```

Algoritma *rule based* dimulai dengan mengaplikasikan aturan logika *single square* dan aturan logika *killer combination*. Kedua aturan logika ini hanya diaplikasikan sekali, yaitu di awal algoritma.

Kode Program - Algoritma *Rule Based*

- Algoritma lalu mengaplikasikan aturan logika *naked subset* dan aturan logika *hidden single*.
- Kedua aturan logika ini diulang sampai algoritma tidak bisa lagi mengisi sel-sel dalam *grid* atau sampai semua sel dalam *grid* sudah terisi dengan benar.

```
...
ArrayList<ArrayList<Integer>> currentGridArrayList
        = getGridArrayList();
ArrayList<ArrayList<Integer>> newGridArrayList
        = solveLoop();
while(!currentGridArrayList.equals(newGridArrayList))
{
    ...
    currentGridArrayList = newGridArrayList;
    newGridArrayList = solveLoop();
}
...
```


Kode Program - Algoritma *Rule Based*
- Algoritma lalu mengaplikasikan aturan logika *naked subset* dan aturan logika *hidden single*.
- Kedua aturan logika ini diulang sampai algoritma tidak bisa lagi mengisi sel-sel dalam *grid* atau sampai semua sel dalam *grid* sudah terisi dengan benar.

Algoritma lalu mengaplikasikan aturan logika *naked subset* dan aturan logika *hidden single*. Kedua aturan logika ini diulang sampai algoritma tidak bisa lagi mengisi sel-sel dalam *grid* atau sampai semua sel dalam *grid* sudah terisi dengan benar.

Kode Program - Algoritma *Rule Based*

- Algoritma *hybrid genetic* selesai jika semua sel dalam *grid* sudah terisi dengan benar.
- Algoritma genetik akan dimulai jika ada sel-sel di dalam *grid* yang masih kosong.

```
        ...
        if (grid.isFilled())
        {
            this.solution = grid;
            return true;
        }
        else
        {
            return false;
        }
    }
```

Algoritma *hybrid genetic* selesai jika semua sel dalam *grid* sudah terisi dengan benar.
Algoritma genetik akan dimulai jika ada sel-sel di dalam *grid* yang masih kosong.

Kode Program - Algoritma Genetik

- Algoritma genetik dimulai dengan membangkitkan generasi awal secara acak.

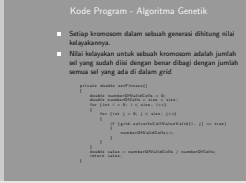
```
public SolverGenetic(...)
{
    ...
    generatePopulation();
    ...
}
```

Algoritma genetik dimulai dengan membangkitkan generasi awal secara acak.

Kode Program - Algoritma Genetik

- Setiap kromosom dalam sebuah generasi dihitung nilai kelayakannya.
- Nilai kelayakan untuk sebuah kromosom adalah jumlah sel yang sudah diisi dengan benar dibagi dengan jumlah semua sel yang ada di dalam *grid*.

```
private double setFitness()
{
    double numberOfValidCells = 0;
    double numberOfCells = size * size;
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (grid.solverIsCellValueValid(i, j) == true)
            {
                numberOfValidCells++;
            }
        }
    }
    double value = numberOfValidCells / numberOfCells;
    return value;
}
```

Setiap kromosom dalam sebuah generasi dihitung nilai kelayakannya. Nilai kelayakan untuk sebuah kromosom adalah jumlah sel yang sudah diisi dengan benar dibagi dengan jumlah semua sel yang ada di dalam *grid*.

Kode Program - Algoritma Genetik

- Algoritma selesai jika solusi ditemukan.
- Solusi adalah kromosom dengan nilai kelayakan 1.

```
public boolean solve()
{
    for (int i = 0; i < generations; i++)
    {
        solveLoop();
        sortChromosomes();
        for (int j = 0; j < populationSize; j++)
        {
            ...
            if (currentGeneration.get(j).getFitness() == 1.0)
            {
                this.solution
                        = currentGeneration.get(j).getGrid();
                return true;
            }
        }
    }
    return false;
}
```

Kode Program - Algoritma Genetik

- Algoritma selesai jika solusi ditemukan.
- Solusi adalah kromosom dengan nilai kelayakan 1.

Algoritma selesai jika solusi ditemukan. Solusi adalah kromosom dengan nilai kelayakan 1.

Kode Program - Algoritma Genetik

- Jika solusi tidak ditemukan, maka algoritma genetik akan membangkitkan generasi berikutnya, sampai solusi ditemukan.

```
private void solveLoop()
{
    int elitismNumber
            = (int) Math.round(populationSize * elitismRate);
    int mutationNumber
            = (int) Math.round(populationSize * mutationRate);
    int crossoverNumber
            =  (int) Math.round((populationSize * crossoverRate)
                    / 2);
    sortChromosomes();
    ...
```


Kode Program - Algoritma Genetik

Jika solusi tidak ditemukan, maka algoritma genetik akan membangkitkan generasi berikutnya, sampai solusi ditemukan.

Jika solusi tidak ditemukan, maka algoritma genetik akan membangkitkan generasi berikutnya, sampai solusi ditemukan.

Kode Program - Algoritma Genetik

- Generasi berikutnya dibangkitkan dari generasi sebelumnya menggunakan operator-operator algoritma genetik, seperti *elitism*, kawin silang, dan mutasi.

```
    ...
    for (int i = 0; i < elitismNumber; i++)
    {
        if (!nextGeneration.contains(currentGeneration.get(i)))
        {
            nextGeneration.add(cloneChromosome(
                    currentGeneration.get(i)));
        }
    }
    for (int i = 0; i < mutationNumber; i++)
    {
        Chromosome parent = randomSelection(currentGeneration);
        nextGeneration.add(mutation(parent));
    }
    for (int i = 0; i < crossoverNumber; i++)
    {
        nextGeneration.addAll(crossover(
                randomSelection(currentGeneration),
                randomSelection(currentGeneration)));
    }
    ...
    currentGeneration = nextGeneration;
    nextGeneration = new ArrayList<Chromosome>();
}
```


Kode Program - Algoritma Genetik
- Generasi berikutnya dibangkitkan dari generasi sebelumnya menggunakan operator-operator algoritma genetik, seperti *elitism*, kawin silang, dan mutasi.

Generasi berikutnya dibangkitkan dari generasi sebelumnya menggunakan operator-operator algoritma genetik, seperti *elitism*, kawin silang, dan mutasi.

Hasil Pengujian - Algoritma *Backtracking*

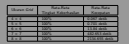
- Pengujian algoritma *backtracking* dilakukan pada permainan dengan *grid* yang berukuran 4×4 sampai dengan 8×8.
- Hasil pengujian algoritma *backtracking* dapat dilihat pada Tabel 1.

Table 1: Hasil pengujian algoritma *backtracking* untuk Calcudoku

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

Hasil Pengujian - Algoritma *Backtracking*
- Pengujian algoritma *backtracking* dilakukan pada permainan dengan *grid* yang berukuran 4×4 sampai dengan 8×8.
- Hasil pengujian algoritma *backtracking* dapat dilihat pada Tabel 1.

Table 1: Hasil pengujian algoritma *backtracking* untuk Calcudoku



Pengujian algoritma *backtracking* dilakukan pada permainan degan *grid* berukuran 4×4 sampai dengan 8×8. Hasil pengujian algoritma *backtracking* dapat dilihat pada Tabel 1.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Pengujian algoritma *hybrid genetic* dilakukan pada *grid* dengan ukuran 4×4 sampai dengan 8×8.
- Dilakukan 16 skenario pengujian.
- Nilai dari parameter-parameter untuk algoritma genetik berbeda-beda untuk setiap skenario.
- Daftar nilai dari parameter-parameter untuk algoritma genetik untuk setiap skenario dapat dilihat pada Tabel 2.

Hasil Pengujian - Algoritma *Hybrid Genetic*
- Pengujian algoritma *hybrid genetic* dilakukan pada *grid* dengan ukuran 4×4 sampai dengan 8×8.
- Dilakukan 16 skenario pengujian.
- Nilai dari parameter-parameter untuk algoritma genetik berbeda-beda untuk setiap skenario.
- Daftar nilai dari parameter-parameter untuk algoritma genetik untuk setiap skenario dapat dilihat pada Tabel 2.

Pengujian algoritma *hybrid genetic* dilakukan pada *grid* dengan ukuran 4×4 sampai dengan 8×8. Dilakukan 16 skenario pengujian. Nilai dari parameter-parameter untuk algoritma genetik berbeda-beda untuk setiap skenario. Daftar nilai dari parameter-parameter untuk algoritma genetik untuk setiap skenario dapat dilihat pada Tabel 2.

Hasil Pengujian - Algoritma *Hybrid Genetic*

Table 2: Nilai untuk parameter-parameter algoritma genetik untuk setiap percobaan yang dilakukan

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

Hasil Pengujian - Algoritma *Hybrid Genetic*
Table 2: Nilai untuk parameter-parameter algoritma genetik untuk setiap percobaan yang dilakukan

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 1 dapat dilihat pada Tabel 3.

Table 3: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 1)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 1 dapat dilihat pada Tabel 3.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 2 dapat dilihat pada Tabel 4.

Table 4: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 2)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 2 dapat dilihat pada Tabel 4.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 3 dapat dilihat pada Tabel 5.

Table 5: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 3)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 3 dapat dilihat pada Tabel 5.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 4 dapat dilihat pada Tabel 6.

Table 6: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 4)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 4 dapat dilihat pada Tabel 6.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 5 dapat dilihat pada Tabel 7.

Table 7: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 5)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 5 dapat dilihat pada Tabel 7.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 6 dapat dilihat pada Tabel 8.

Table 8: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 6)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 6 dapat dilihat pada Tabel 8.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 7 dapat dilihat pada Tabel 9.

Table 9: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 7)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 7 dapat dilihat pada Tabel 9.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 8 dapat dilihat pada Tabel 10.

Table 10: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 8)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 8 dapat dilihat pada Tabel 10.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 9 dapat dilihat pada Tabel 11.

Table 11: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 9)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 9 dapat dilihat pada Tabel 11.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 10 dapat dilihat pada Tabel 12.

Table 12: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 10)

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

Hasil Pengujian - Algoritma *Hybrid Genetic*

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 10 dapat dilihat pada Tabel 12.

Table 12: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 10)

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 10 dapat dilihat pada Tabel 12.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 11 dapat dilihat pada Tabel 13.

Table 13: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 11)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 11 dapat dilihat pada Tabel 13.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 12 dapat dilihat pada Tabel 14.

Table 14: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 12)

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

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 12 dapat dilihat pada Tabel 14.

Table 14: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 12)

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 12 dapat dilihat pada Tabel 14.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 13 dapat dilihat pada Tabel 15.

Table 15: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 13)

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

Hasil Pengujian - Algoritma *Hybrid Genetic*
Hasil pengujian algoritma *hybrid genetic* untuk Skenario 13 dapat dilihat pada Tabel 15.

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 13 dapat dilihat pada Tabel 15.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 14 dapat dilihat pada Tabel 16.

Table 16: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 14)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 14 dapat dilihat pada Tabel 16.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 15 dapat dilihat pada Tabel 17.

Table 17: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 15)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 15 dapat dilihat pada Tabel 17.

Hasil Pengujian - Algoritma *Hybrid Genetic*

- Hasil pengujian algoritma *hybrid genetic* untuk Skenario 16 dapat dilihat pada Tabel 18.

Table 18: Hasil pengujian algoritma *hybrid genetic* untuk Calcudoku (Skenario 16)

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

Hasil pengujian algoritma *hybrid genetic* untuk Skenario 16 dapat dilihat pada Tabel 18.

Demo Program

Simpulan

- Perangkat lunak permainan teka-teki Calcudoku dengan dua *solver*, yaitu *solver* dengan algoritma *backtracking* dan *solver* dengan algoritma *hybrid genetic*, berhasil dibuat.
- Algoritma *backtracking* dapat menyelesaikan semua permainan yang diujikan, tetapi pada ukuran *grid* yang besar, algoritma *backtracking* sangat lambat dalam menyelesaikan permainan.
- Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan karena sifat acak dari algoritma *hybrid genetic* ini. Semakin besar ukuran *grid*, maka kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan semakin besar.

Simpulan

- Perangkat lunak permainan teka-teki Calcudoku dengan dua *solver*, yaitu *solver* dengan algoritma *backtracking* dan *solver* dengan algoritma *hybrid genetic*, berhasil dibuat.
- Algoritma *backtracking* dapat menyelesaikan semua permainan yang diujikan, tetapi pada ukuran *grid* yang besar, algoritma *backtracking* sangat lambat dalam menyelesaikan permainan.
- Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan karena sifat acak dari algoritma *hybrid genetic* ini. Semakin besar ukuran *grid*, maka kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan semakin besar.

Perangkat lunak permainan teka-teki Calcudoku dengan dua *solver*, yaitu *solver* dengan algoritma *backtracking* dan *solver* dengan algoritma *hybrid genetic*, berhasil dibuat. Perangkat lunak ini menerima input berupa soal teka-teki dan mampu menyelesaikan soal teka-teki tersebut menggunakan algoritma *backtracking* dan *hybrid genetic*.
Algoritma *backtracking* dapat menyelesaikan semua permainan yang diujikan, tetapi pada ukuran *grid* yang besar, algoritma *backtracking* sangat lambat dalam menyelesaikan permainan.
Ada kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan karena sifat acak dari algoritma *hybrid genetic* ini. Semakin besar ukuran *grid*, maka kemungkinan algoritma *hybrid genetic* gagal dalam menyelesaikan permainan semakin besar.

Simpulan

- Pada ukuran *grid* yang kecil, algoritma *hybrid genetic* cenderung menyelesaikan permainan lebih lambat daripada algoritma *backtracking*.
- Pada ukuran *grid* yang besar, algoritma *hybrid genetic* mungkin mampu menyelesaikan permainan lebih cepat daripada algoritma *backtracking*, tetapi hal ini tidak dapat dibuktikan karena algoritma *hybrid genetic* gagal dalam menyelesaikan permainan dengan ukuran *grid* yang besar.
- Banyaknya sel yang diisi dalam tahap algoritma *rule based* dan nilai dari parameter-parameter untuk algoritma genetik mempengaruhi kecepatan dan tingkat keberhasilan algoritma *hybrid genetic* dalam menyelesaikan permainan.

Simpulan
- Pada ukuran *grid* yang kecil, algoritma *hybrid genetic* cenderung menyelesaikan permainan lebih lambat daripada algoritma backtracking.
- Pada ukuran *grid* yang besar, algoritma *hybrid genetic* mungkin mampu menyelesaikan permainan lebih cepat daripada algoritma backtracking, tetapi hal ini tidak dapat dibuktikan karena algoritma *hybrid genetic* gagal dalam menyelesaikan permainan dengan ukuran *grid* yang besar.
- Banyaknya sel yang diisi dalam tahap algoritma *rule based* dan nilai dari parameter-parameter untuk algoritma genetik mempengaruhi kecepatan dan tingkat keberhasilan algoritma *hybrid genetic* dalam menyelesaikan permainan.

Pada ukuran *grid* yang kecil, algoritma *hybrid genetic* cenderung menyelesaikan permainan lebih lambat daripada algoritma *backtracking*. Tetapi, pada ukuran *grid* yang besar, algoritma *hybrid genetic* mungkin mampu menyelesaikan permainan lebih cepat daripada algoritma *backtracking*, tetapi hal ini tidak dapat dibuktikan karena algoritma *hybrid genetic* gagal dalam menyelesaikan permainan dengan ukuran *grid* yang besar.

Banyaknya sel yang diisi dalam tahap algoritma *rule based* dan nilai dari parameter-parameter untuk algoritma genetik mempengaruhi kecepatan dan keberhasilan algoritma *hybrid genetic* dalam menyelesaikan permainan.

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

Semakin besar populasi dalam sebuah generasi sampai ke titik tertentu, dan semakin banyak generasi sampai ke titik tertentu, maka semakin besar juga kemungkinan algoritma *hybrid genetic* berhasil dalam menyelesaikan permainan. Semakin besar tingkat *elitism* dan tingkat mutasi sampai ke titik tertentu, maka semakin cepat juga algoritma *hybrid genetic* dalam menyelesaikan permainan.

Saran

- Memperbaiki GUI dari perangkat lunak ini agar petunjuk dapat ditampilkan sebagaimana mestinya.
- Menambah aturan-aturan logika untuk algoritma *rule based*, misalnya aturan *naked subset* untuk *cage* yang berukuran lebih besar dari 3 sel, aturan *hidden subset* untuk *cage* yang berukuran lebih besar dari 2 sel, aturan *killer* combination untuk *cage* yang berukuran lebih besar dari 2 sel.
- Memperbaiki algoritma genetik, misalnya proses pemberian nilai kelayakan untuk kromosom, proses pemilihan kromosom untuk kawin silang dan mutasi, proses *elitism*, proses kawin silang, dan proses mutasi.

Saran
- Memperbaiki GUI dari perangkat lunak ini agar petunjuk dapat ditampilkan sebagaimana mestinya.
- Menambah aturan-aturan logika untuk algoritma *rule based*, misalnya aturan *naked subset* untuk *cage* yang berukuran lebih besar dari 3 sel, aturan *hidden subset* untuk *cage* yang berukuran lebih besar dari 2 sel, aturan *killer* combination untuk *cage* yang berukuran lebih besar dari 2 sel.
- Memperbaiki algoritma genetik, misalnya proses pemberian nilai kelayakan untuk kromosom, proses pemilihan kromosom untuk kawin silang dan mutasi, proses *elitism*, proses kawin silang, dan proses mutasi.

Memperbaiki GUI dari perangkat lunak ini agar petunjuk, yaitu angka tujuan dan operasi matematika yang ditentukan untuk sebuah *cage*, dapat ditampilkan sebagaimana mestinya, yaitu pada di sudut kiri atas sel yang paling atas dan yang paling kiri dalam *cage* tersebut.
Menambah aturan-aturan logika untuk algoritma *rule based*, misalnya aturan *naked subset* untuk *cage* yang berukuran lebih besar dari 3 sel, aturan *hidden subset* untuk *cage* yang berukuran lebih besar dari 2 sel, aturan *killer* combination untuk *cage* yang berukuran lebih besar dari 2 sel, dan aturan *evil twin* untuk *cage* yang berukuran minimal 2 sel. Dengan menambah aturan-aturan logika untuk algoritma *rule based*. Diharapkan, dengan menambah aturan-aturan logika untuk algoritma *rule based*, maka tingkat kesuksesan algoritma *hybrid genetic* dalam menyelesaikan permainan Calcudoku dapat meningkat.
Memperbaiki algoritma genetik, misalnya proses pemberian nilai kelayakan untuk kromosom, proses pemilihan kromosom untuk kawin silang dan mutasi, proses *elitism*, proses kawin silang, dan proses mutasi, sehingga tingkat kesuksesan algoritma *hybrid genetic* dalam menyelesaikan permainan Calcudoku dapat meningkat.

Daftar Pustaka

- Asanilta Fahda, *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 2015.
- Olivia Johanna, Samuel Lukas, Kie Van Ivanky Saputra, *Solving and Modeling Ken-ken Puzzle by Using Hybrid Genetics Algorithm*, 1st International Conference on Engineering and Technology Development (ICETD 2012), Faculty of Engineering and Faculty of Computer Science, Bandar Lampung University, 2012.

Daftar Pustaka

Asanilta Fahda, *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 2015.

Olivia Johanna, Samuel Lukas, Kie Van Ivanky Saputra, *Solving and Modeling Ken-ken Puzzle by Using Hybrid Genetics Algorithm*, 1st International Conference on Engineering and Technology Development (ICETD 2012), Faculty of Engineering and Faculty of Computer Science, Bandar Lampung University, 2012.

Terima Kasih

- Ada pertanyaan?

Terima Kasih
Ada pertanyaan?