

Implementasi Algoritma Genetika pada Permainan *Mosaic*

Pengantar Sistem Cerdas



Dibuat oleh:

Wilson 6182001039

Alvin Mishael Halim 6182001047

Muhammad Rafif Pratama 6182001061

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

2022

DAFTAR ISI

BAB 1	
Pendahuluan	4
1.1 Latar Belakang	4
1.2 Rumusan Masalah	6
1.3 Tujuan	6
1.4 Batasan Masalah	6
1.5 Pembagian Pekerjaan	7
BAB 2	
Landasan Teori	8
2.1 Permainan Mosaic	8
2.2 Algoritma Genetika	9
BAB 3	
Analisis Masalah	14
BAB 4	
Implementasi Perangkat Lunak	16
4.1 Diagram Kelas	16
4.1.1 Kelas MainTubesPSC	16
4.1.2 Kelas Populasi	17
4.1.3 Kelas Generasi	20
4.1.4 Kelas Individual	22
4.2 Rancangan Operator genetika	23
4.2.1 Seleksi	23
4.2.2 Crossover	25
4.2.3 Elitism	26
4.2.4 Mutasi	26
4.3 Spesifikasi input dan output	27
4.3.1 Input	27
4.3.1 Output	28
4.4 Penambahan Fitur untuk Mencatat Waktu Eksekusi	29
BAB 5	
Eksperimen	30
5.1 Lingkungan Eksperimen	30
5.1.1 Model Papan	30
5.1.2 Parameter Tetap	32
5.2 Uji Parameter	32

5.2.1 Uji Parameter Pertama	33
5.2.2 Uji Parameter Kedua	34
5.2.3 Uji Parameter Ketiga	35
5.2.4 Uji Parameter Keempat	36
5.2.5 Uji Parameter Kelima	37
5.2.6 Uji Parameter Keenam	38
5.2.7 Uji Parameter Ketujuh	39
5.2.8 Uji Parameter Kedelapan	40
5.2.9 Uji Parameter Kesembilan	41
5.2.10 Uji Parameter Kesepuluh	42
5.2.11 Uji Parameter Kesebelas	43
5.2.12 Uji Parameter Keduabelas	44
5.3 Perbandingan Eksekusi Algoritma Genetika	45
5.4 Eksperimen Menggunakan Seratus Buah Papan 5x5	48
BAB 6	
Kesimpulan	50
Daftar Pustaka	52

BAB 1

Pendahuluan

1.1 Latar Belakang

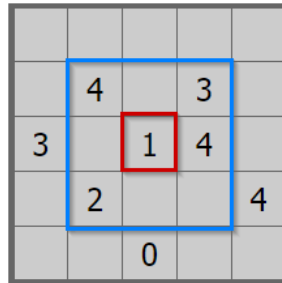
Minesweeper merupakan salah satu permainan komputer yang cukup dikenal luas oleh masyarakat Indonesia dan dunia. Pada awalnya, permainan *minesweeper* terinspirasi dari gim yang bernama *Mined-Out* yang diciptakan oleh Ian Andrew menggunakan bahasa BASIC pada tahun 1983. Kemudian pada tahun 1990, Robert Donner dan Curt Johnson meminjam dan menyempurnakan gim tersebut hingga menjadi permainan *minesweeper* yang dikenal hingga saat ini. *Minesweeper* tersebut diperkenalkan pada acara di Windows Entertainment Park saat peluncuran Windows 3.11. Hadirnya gim tersebut direspon baik oleh masyarakat sehingga *minesweeper* menjadi salah satu game yang selalu disertakan dalam sistem operasi dan GUI, seperti *Minesweeper* for IBM's OS/2, KDE, GNOME dan Palm OS. Microsoft juga secara otomatis selalu menyertakan gim tersebut dalam sistem operasinya hingga Windows 8. Kepopuleran sistem operasi windows juga mempengaruhi meningkatnya jumlah masyarakat yang mengetahui gim tersebut dan membuat *minesweeper* menjadi salah satu game terlaris sepanjang masa. Sekilas permainan *minesweeper* terlihat cukup sederhana untuk dimainkan, hanya terdiri dari sebuah papan dengan kotak-kotak berisi angka, dan ranjau. Namun, pada kenyataannya masih banyak orang yang belum mengetahui cara memainkan gim tersebut. Kepopuleran *minesweeper* membuat beberapa orang memodifikasi *minesweeper* agar menjadi lebih menantang. Salah satu hasil modifikasi dari permainan *minesweeper* adalah *mosaic*.

Mosaic adalah variasi permainan dari *minesweeper* yang terdapat pada laman [Mosaic - online puzzle game \(puzzle-minesweeper.com\)](http://puzzle-minesweeper.com) atau laman lainnya. Mirip seperti *minesweeper*, gim *mosaic* terdiri dari sebuah kotak-kotak berisi angka, tetapi tidak terdapat ranjau, melainkan harus meletakkan kotak-kotak berwarna hitam. Angka yang tertera menandakan jumlah kotak hitam yang bertetangga pada angka tersebut secara vertikal, horizontal, diagonal dan pada kotak angka itu sendiri. Antarmuka dari permainan ini dapat dilihat pada Gambar 1.1

	4		3	
3		1	4	
	2			4
		0		

Gambar 1.1: Tampilan dari permainan *Mosaic*

Tujuan dari permainan *minesweeper* adalah membuka setiap kotak pada sel hingga semua kotak terbuka seluruhnya tanpa menekan ranjau yang ada, sedangkan tujuan dari permainan *mosaic* adalah meletakkan semua kotak hitam pada seluruh sel-sel yang memungkinkan dengan memperhatikan angka-angka pada sel tertentu sehingga seluruh kotak hitam sesuai dengan angka tersebut.



Gambar 1.2: Contoh lokasi keberadaan kotak hitam.

Pada contoh permainan di Gambar 1.2, angka 1 (kotak merah) yang terletak di tengah papan menandakan bahwa ada sejumlah satu kotak hitam yang dapat diletakkan di sekitar sel angka tersebut termasuk angka itu sendiri (dapat diletakkan di dalam kotak biru termasuk kotak merah). Peletakan kotak hitam perlu diisi sedemikian rupa dengan memperhatikan angka-angka di sel lain. Permainan *mosaic* dirancang dengan menggunakan logika untuk menyelesaikan permainan tersebut, sehingga menjadi menarik dan menantang karena perlu berpikir secara teliti. Gambar 1.3 merupakan contoh permainan yang telah diselesaikan karena seluruh kotak hitam sudah diletakkan dengan tepat.

Banyak orang yang berlomba-lomba untuk mencari cara tercepat agar dapat menyelesaikan permainan tersebut, sehingga beberapa orang mencoba menyelesaikan permasalahan tersebut menggunakan komputer. Akan tetapi, berdasarkan penelitian yang ditulis oleh Richard Kaye dalam Jurnal *Mathematical Intelligencer*, permainan *minesweeper* memiliki kompleksitas waktu sebesar n -exponential time (n^n) dan termasuk dalam kategori permainan *non-deterministic*, sehingga termasuk ke dalam masalah NP-Complete. Dalam masalah NP-Complete, untuk mendapatkan jawaban yang benar dengan akurasi 100%, perlu dilakukan penelusuran ke seluruh state space yang ada, sedangkan waktu yang dibutuhkan untuk menelusuri seluruh state space membutuhkan waktu yang sangat lama. Masalah tersebut menyebabkan hingga saat ini belum ada algoritma yang dapat memberikan jawaban benar dalam waktu yang cukup singkat.

Oleh karena itu, untuk menyelesaikan kasus di atas, algoritma local search dapat menjadi salah satu solusi. Algoritma local search tidak bekerja untuk mencari jawaban optimal atau benar secara keseluruhan, melainkan mencari jawaban hampir benar atau sub-optimal. Mengingat butuh waktu yang sangat lama untuk dapat menyelesaikan permainan *mosaic*, algoritma local search dapat menjadi salah satu pilihan terbaik untuk memecahkan permasalahan tersebut.

1.2 Rumusan Masalah

Berikut rumusan masalah yang dibahas berdasarkan pemaparan latar belakang di atas:

1. Bagaimana cara menyelesaikan permainan *mosaic* agar dapat memodelkan fungsi fitness terbaik?
2. Bagaimana cara mengimplementasikan algoritma genetika agar dapat menyelesaikan permainan *mosaic* dengan menggunakan bahasa Java?
3. Bagaimana performa algoritma genetika dengan menggunakan berbagai parameter tertentu dalam memainkan permainan *mosaic*?

1.3 Tujuan

Berikut tujuan yang ingin dicapai berdasarkan masalah yang telah dirumuskan sebelumnya:

1. Mempelajari cara dan pola permainan *mosaic* agar dapat menyelesaikan permainan tersebut.
2. Menganalisis penyelesaian permainan *mosaic* yang telah dipelajari sebelumnya untuk mendapatkan nilai-nilai heuristik sehingga dapat diimplementasikan ke dalam algoritma genetika.
3. Mencatat waktu eksekusi dan nilai *fitness function* dari masing-masing parameter yang digunakan.

1.4 Batasan Masalah

Berikut batasan-batasan masalah yang diberikan:

1. Versi permainan *mosaic* yang digunakan adalah versi yang dibuat oleh *qqwref* yang terdapat pada tautan [Mosaic - online puzzle game \(puzzle-minesweeper.com\)](http://puzzle-minesweeper.com).
2. Jumlah generasi pada pengujian performa agen permainan *mosaic* dibatasi sehingga satu permainan tidak menghabiskan generasi lebih dari 1500.
3. Ukuran papan yang diuji adalah sebesar 5x5, 7x7, 10x10 dengan *level easy* dan *hard*, dan papan sebesar 15x15 dan 20x20 dengan *level easy*.

1.5 Pembagian Pekerjaan

Mahasiswa	Pekerjaan
Alvin Mishael Halim	<ul style="list-style-type: none">○ Memahami tugas 1 dan 2○ Mempelajari permainan <i>mosaic</i>○ Membuat dokumentasi○ Membuat rancangan implementasi algoritma genetika○ Menjadi project manager dan membuat repository di github○ Melakukan implementasi nilai fitness function○ Membuat dan mengimplementasi kelas MainTubesPSC○ Membuat dan mengimplementasi kelas Individual○ Melakukan eksperimen
Muhammad Rafif Pratama	<ul style="list-style-type: none">○ Memahami tugas 1 dan 2○ Mempelajari permainan <i>mosaic</i>○ Membuat dokumentasi○ Membuat rancangan fitness function berserta nilai-nilai heuristik○ Membuat dan mengimplementasi kelas Generasi○ Melakukan eksperimen
Wilson	<ul style="list-style-type: none">○ Memahami tugas 1 dan 2○ Mempelajari permainan <i>mosaic</i>○ Membuat dokumentasi○ Membuat rancangan fitness function berserta nilai-nilai heuristik○ Membuat dan mengimplementasi kelas Populasi○ Mengubah format input dan output pada kelas main○ Melakukan eksperimen

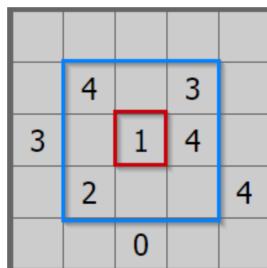
BAB 2

Landasan Teori

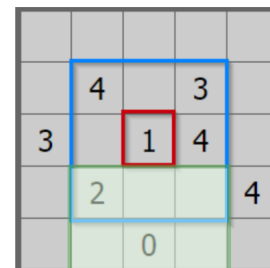
2.1 Permainan *Mosaic*

Minesweeper adalah permainan komputer untuk satu orang pemain yang bertujuan untuk membuka kotak-kotak yang ada tanpa mengenai ranjau. Pemain akan membuka kotak-kotak yang ada pada papan permainan. Jika kotak yang terbuka berisi ranjau, maka pemain kalah. Sementara jika kotak yang terbuka berisi angka, maka angka tersebut menandakan jumlah ranjau yang ada di sekelilingnya.

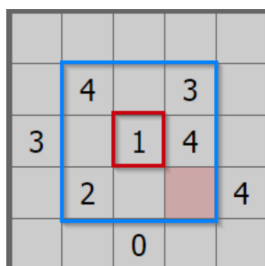
Mosaic merupakan salah satu macam permainan *minesweeper* yang memiliki cara bermain yang sedikit berbeda. Seperti permainan *minesweeper*, *mosaic* dimainkan oleh satu pemain dengan memberikan papan permainan yang terdiri dari kotak kosong atau pun kotak yang berisi angka. Pada permainan *mosaic* tidak terdapat ranjau seperti pada permainan *minesweeper*. Angka yang terdapat pada menunjukkan batas maksimum kotak hitam yang dapat terbentuk di sekitar angka tersebut. Pemain dapat mengklik kotak yang ada dan kotak akan berubah warna menjadi hitam atau putih. Jika terdapat kotak hitam yang lebih di sekitar angka, maka kotak yang berisi angka tersebut akan berubah warna menjadi merah untuk memberikan peringatan kepada pemain. Permainan *mosaic* akan berakhir ketika semua kotak pada papan sudah berubah warna menjadi hitam atau putih dan tidak ada kotak yang berisi angka dalam keadaan berwarna merah.



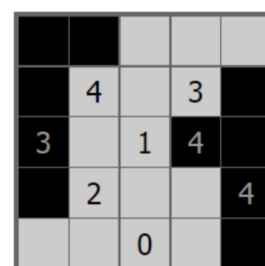
(a) Area berisi kotak hitam



(b) Probabilitas kotak hitam setelah melihat sel 0



(c) Kondisi saat terjadi kesalahan



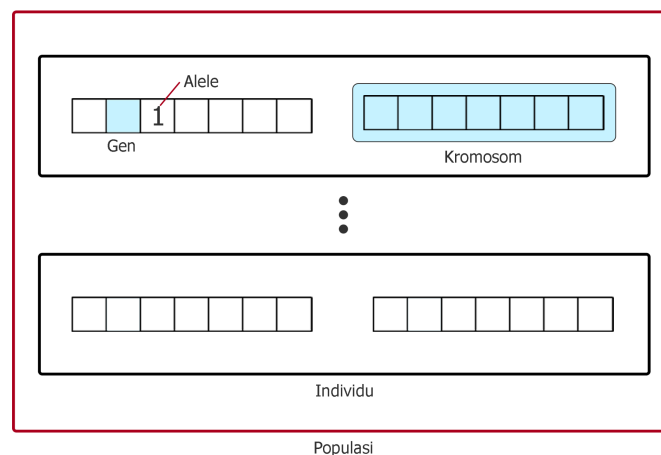
(d) Kondisi semua sel sudah terisi dengan benar

Pada contoh permainan di Gambar A, angka 1 (kotak merah) yang terletak di tengah papan menandakan bahwa ada sejumlah maksimal satu kotak hitam yang dapat diletakkan di sekitar sel angka tersebut termasuk angka itu sendiri (dapat diletakkan di dalam kotak biru). Namun, peletakan kotak hitam juga perlu memperhatikan angka di sel-sel yang berdekatan seperti pada Gambar B. Daerah yang diarsir berwarna hijau pada Gambar B menandakan bahwa tidak terdapat kotak hitam karena pusat sel bernilai 0, sehingga peletakan kotak hitam tidak dapat diletakkan pada daerah yang diarsir berwarna hijau. Gambar C menunjukkan contoh error yang terjadi ketika terdapat kotak hitam yang di tempat tidak seharusnya berada. Seluruh letak kotak hitam yang benar dapat dilihat pada Gambar D.

2.2 Algoritma Genetika

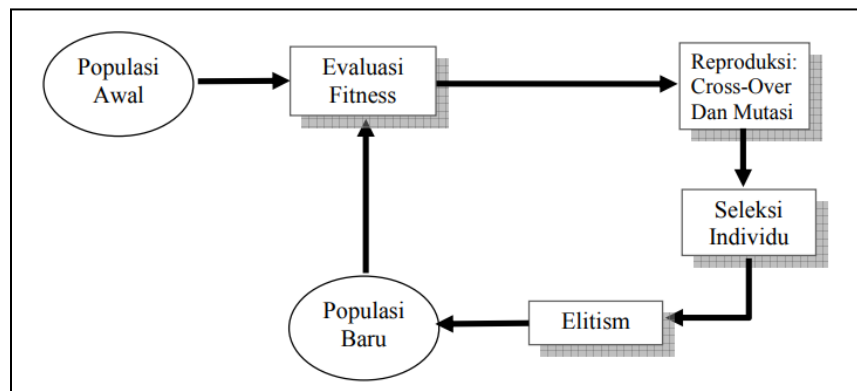
Algoritma genetika pertama kali diperkenalkan oleh John Holland bersama para rekan dan mahasiswanya di University of Michigan pada tahun 1962. Algoritma genetika adalah algoritma pencarian yang merupakan variasi dari stochastic beam search dimana *child* dihasilkan melalui proses penggabungan dua *parent* dibandingkan hanya dengan memodifikasi satu *single state*. Kemudian pada tahun 1975, John Holland bersama para rekannya menerbitkan buku yang berjudul *Adaptation in Natural and Artificial Systems* untuk membuktikan algoritma genetika dapat digunakan dalam menyediakan algoritma pencarian yang kompleks.

Algoritma genetika mengadaptasi teori evolusi dalam dunia biologi seperti *inheritance*, *mutation*, *selection*, and *crossover*. Algoritma ini menggunakan populasi yang terdiri dari individu-individu. Tiap individu merupakan representasi dari sebuah solusi untuk masalah yang diberikan. Masing-masing individu juga memiliki nilai *fitness* yang digunakan untuk mencari solusi. Secara sederhana, algoritma genetika bekerja dengan cara mengawinkan tiap individu dan menghasilkan keturunan yang membawa sifat dari *parent*nya. Karena menggunakan teori seleksi alam, maka individu yang memiliki nilai *fitness* yang buruk akan punah atau tidak terpakai. Melalui sistem tersebut, individu-individu baru dengan nilai *fitness* yang baik akan bermunculan, sehingga akan menghasilkan kemungkinan solusi terbaik.



Berikut beberapa definisi penting dalam algoritma genetika (E. Satriyanto, 2009):

1. Gen
Bagian terkecil dalam algoritma genetika yang dapat berupa nilai integer, float, biner, dan sebagainya.
2. Kromosom
Kumpulan atau gabungan dari gen-gen yang membentuk nilai tertentu.
3. Individu
Merupakan suatu nilai yang menyatakan salah satu solusi untuk memecahkan permasalahan yang ada.
4. Populasi
Merupakan kumpulan dari individu yang akan diproses dalam satu siklus proses evolusi.
5. Generasi
Menyatakan satu iterasi dalam algoritma genetika atau satu siklus proses evolusi.



Siklus Algoritma genetika yang diperbaiki oleh Zbigniew Michalewicz

2.2.1 Pembentukan Populasi Awal

Langkah pertama dalam algoritma genetika adalah menentukan *encoding* sebagai representasi dari gen dan kromosom. Berikut beberapa teknik *encoding* yang umum dipakai:

- *Bit-string* : 10001 01000 00110 dst.
- Array bilangan riil : 30.56, 12.80, -12.91, dst.
- Elemen permutasi : E2, E5, E10 dst.
- Value encoding : 1.234, ABCDF. (left), dst.
- Struktur lainnya

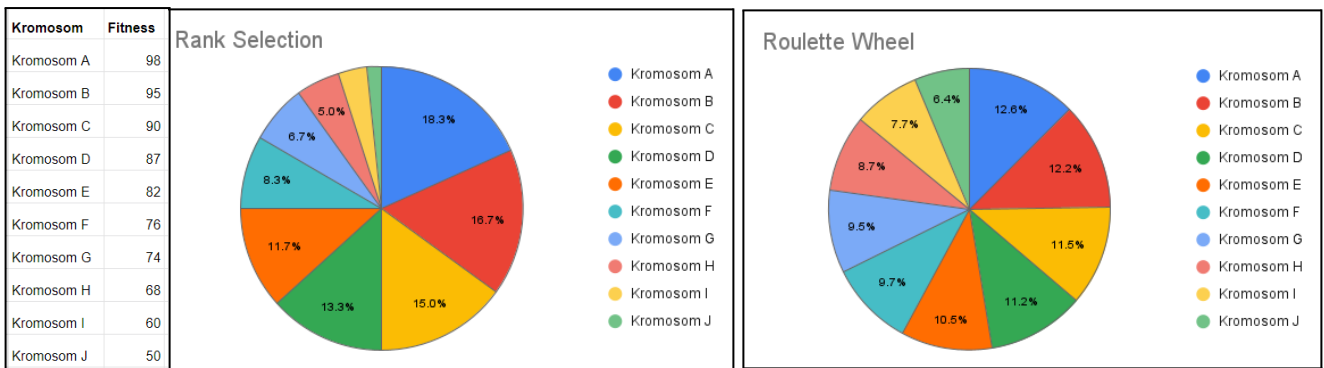
2.2.2 Fitness Function

Langkah selanjutnya setelah menentukan populasi awal adalah dengan mengevaluasi populasi dengan nilai *fitness function*. Nilai fitness digunakan untuk menentukan seberapa baik nilai dari suatu kromosom. Nilai fitness dijadikan acuan dalam mencari solusi paling optimal yang bisa didapatkan.

2.2.3 Seleksi

Seleksi merupakan langkah untuk mendapatkan calon *parent* terbaik saat akan melakukan regenerasi. Proses pemilihan seleksi dilakukan dengan cara melihat nilai fitness tiap individu. Semakin tinggi nilai fitness maka akan semakin besar kemungkinan individu tersebut terpilih. Terdapat beberapa metode seleksi dalam algoritma genetika, diantaranya adalah:

- *Roulette Wheel*
- *Rank Selection*
- *Tournament Selection*
- dan sebagainya

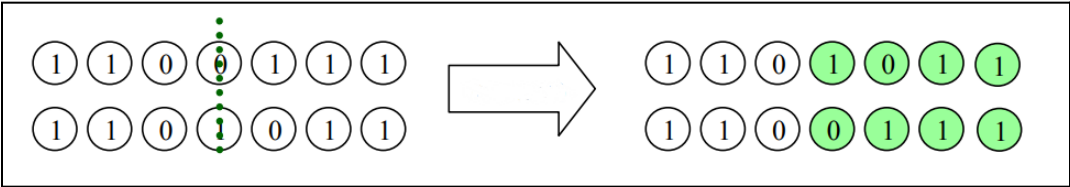


Perbandingan seleksi menggunakan Roulette Wheel dan Rank Selection

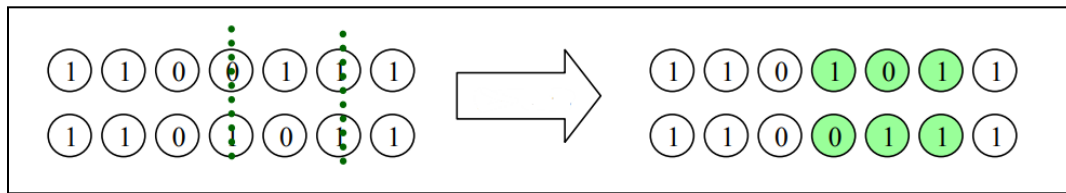
2.2.4 Crossover

Kawin silang (crossover) adalah operasi dari algoritma genetika yang melibatkan dua parent untuk membentuk kromosom anak (offspring). Terdapat beberapa metode crossover algoritma genetika, diantaranya adalah:

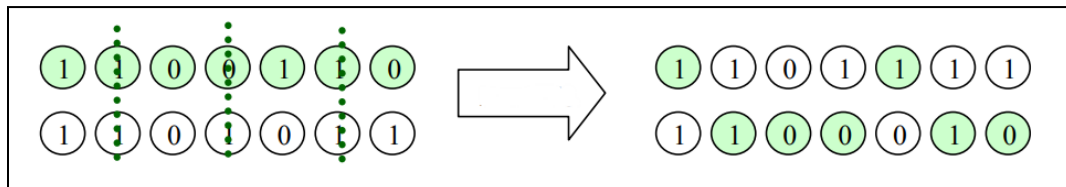
- *Single point crossover*



- *Double point crossover*

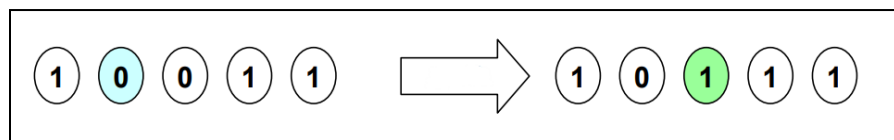


- *Multiple point crossover*

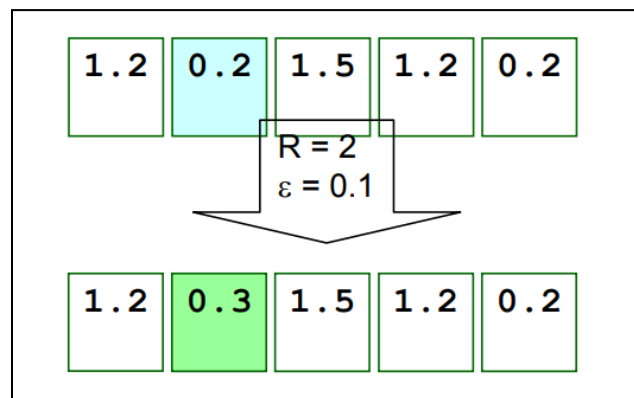


2.2.4 Mutasi

Mutasi berperan untuk menggantikan gen yang hilang dari populasi akibat proses seleksi sehingga dapat mengembalikan solusi optimal. Operator ini juga dapat berperan untuk mencegah terjadinya *local maximum* (N. Muniarti, 2009). Kromosom offspring dimutasi dengan menambahkan nilai random yang sangat kecil dan probabilitas yang rendah.

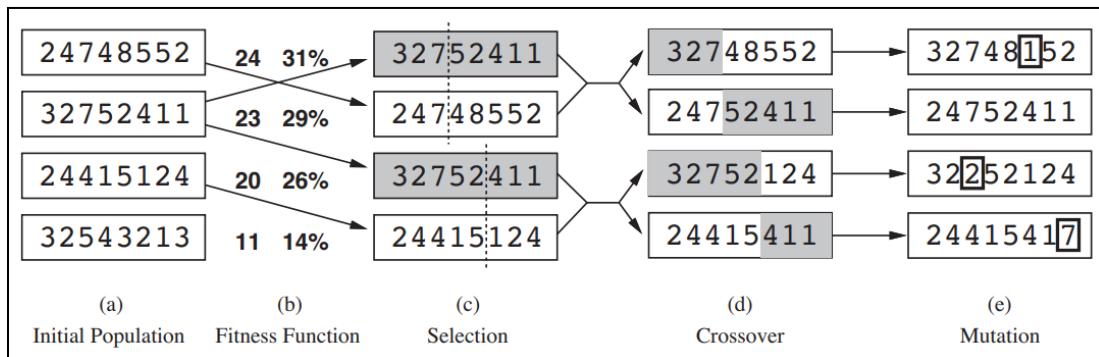


(a) Contoh mutasi pada kromosom bertipe bilangan biner



(b) Contoh mutasi pada kromosom bertipe bilangan riil

Hasil implementasi menggunakan algoritma genetika dapat dilihat pada Gambar 2.3 untuk kasus permasalahan 8-Queen.



Gambar 2.3: Implementasi algoritma genetika pada 8 Queens Problem

Tiap kromosom terdiri dari 8 bilangan yang merupakan representasi letak dari tiap Queen, kemudian diurutkan berdasarkan fitness function. Masing-masing kromosom melakukan kawin silang dengan operator seleksi 1 titik dan menghasilkan keturunan baru. Terakhir, terdapat mutasi pada gen-gen tertentu.

Berikut merupakan salah satu implementasi algoritma genetika ke dalam pseudocode pada Gambar 2.4 yang terdapat pada buku *Artificial Intelligence: a modern approach*.

```

function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
           FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population  $\leftarrow$  empty set
    for  $i = 1$  to SIZE(population) do
       $x \leftarrow$  RANDOM-SELECTION(population, FITNESS-FN)
       $y \leftarrow$  RANDOM-SELECTION(population, FITNESS-FN)
      child  $\leftarrow$  REPRODUCE( $x, y$ )
      if (small random probability) then child  $\leftarrow$  MUTATE(child)
      add child to new_population
    population  $\leftarrow$  new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN



---


function REPRODUCE( $x, y$ ) returns an individual
  inputs:  $x, y$ , parent individuals

   $n \leftarrow$  LENGTH( $x$ );  $c \leftarrow$  random number from 1 to  $n$ 
  return APPEND(SUBSTRING( $x, 1, c$ ), SUBSTRING( $y, c + 1, n$ ))

```

Gambar 2.4: Pseudocode algoritma genetika

BAB 3

Analisis Masalah

Dalam permainan *mosaic*, terdapat level *easy* dan *hard* yang disediakan. Level *hard* pada permainan ini cenderung lebih sering menampilkan angka-angka yang relatif lebih besar(>5). Selama bermain, papan dengan angka-angka yang lebih besar cenderung lebih berdampak pada tingkat kesusahan untuk mencari solusi. Semakin besar angka yang ada, semakin banyak kotak hitam yang harus ada di dalam papan. Hal ini mengakibatkan kemungkinan terjadinya kesalahan dengan kotak lain di sekitarnya jika kotak-kotak di sekitar angka tersebut berada di posisi yang salah.

Ketika memulai permainan, diberikan papan dengan ukuran tertentu. Papan tersebut diimplementasikan sebagai kromosom yang menggunakan tipe data array(int[]). Kotak kosong pada papan dikodekan dengan nilai -1, sementara untuk kotak yang berisi angka dikodekan dengan nilai angka tersebut. Untuk solusi dari permainan tersebut diimplementasikan dengan array(int[]) dimana nilai 1 berarti kotak berwarna hitam dan nilai 0 berarti kotak berwarna putih.

		3		
1	3		3	
	2	2		0
		1		2
	1			

5x5 Easy Mosaic Puzzle ID: 11,065,139

Dari contoh kasus di atas, papan permainan di-*encoding* menjadi array 1 dimensi seperti pada tabel di bawah.

-1	-1	3	-1	-1	1	3	-1	3	-1	-1	2	2	-1	0	-1	-1	1	-1	2	-1	1	-1	-1	-1
----	----	---	----	----	---	---	----	---	----	----	---	---	----	---	----	----	---	----	---	----	---	----	----	----

Solusi untuk permasalahan permainan *mosaic* ini didapatkan dari pencarian generasi-generasi menggunakan algoritma genetika. Dalam hal ini, ada beberapa hal yang perlu diperhatikan seperti berikut.

1. Fitness function

Perhitungan nilai fitness dari sebuah individu ditentukan dengan yang pertama adalah menelusuri kromosom untuk mencari jumlah angka yang terdapat pada tabel. Selanjutnya, mengkalikan jumlah angka yang ada pada kromosom dengan 9. Hal ini karena satu angka pada tabel dapat mempengaruhi 9 kotak lain disekitarnya. Hasil perkalian tersebut kemudian dikalikan dengan 10 karena angka yang muncul berada pada rentang 0-9. Jika penelusuran kromosom menemukan angka -1 pada kotak(kosong), cari kota-kotak di sekelilingnya. Selanjutnya lakukan penelusuran untuk mencari kotak hitam yang ada dalam kromosom. Jika terdapat perbedaan jumlah kotak hitam disekeliling angka yang ada di tabel (misalnya angka yang muncul adalah 6 namun hanya ada 3 kotak hitam), maka nilai fitness sebelumnya dikurang sebanyak selisih angka dengan kotak dikali angka yang terdapat pada tabel+1 (untuk menghindari perkalian dengan 0). Hal ini dilakukan untuk membedakan jika angka pada tabel yang salah merupakan angka yang besar atau angka yang kecil. Dengan demikian, rumus untuk menghitung fitness sebuah individu adalah sebagai berikut.

2. Crossover

Crossover dilakukan untuk menghasilkan generasi dari parent pilihan. Dalam percobaan pencarian solusi permainan *mosaic* ini, dilakukan dua tipe cara crossover yaitu *one point crossover* dan *two point crossover*. Pada *one point crossover* dilakukan pemilihan 1 titik crossover pada kromosom secara acak. Kemudian lakukan persilangan antara kedua parent yang sudah terpilih. Pada *two point crossover* dilakukan pemilihan 2 titik crossover dari kromosom parent sehingga kromosom parent akan terbagi menjadi 3 bagian. Kemudian lakukan persilangan pada bagian tengah kedua kromosom parent. Dari kedua cara crossover ini maka akan didapatkan generasi baru.

3. Mutasi

Pada pembentukan generasi baru tidak menutup kemungkinan terjadinya mutasi. Mutasi ini dapat membentuk generasi yang berbeda dari generasi lainnya. Pada percobaan ini, kemungkinan terjadinya mutasi pada pembentukan generasi adalah

4. Elitism

Elitism digunakan untuk melakukan pengambilan sejumlah individu bagus dari sebuah populasi. Hal ini dilakukan untuk dapat menghasilkan generasi yang diharapkan memiliki fitness yang baik. Pada proses percobaan pencarian solusi permainan *mosaic*, dilakukan percobaan dengan menggunakan beberapa besaran elitism.

5. Parent Selection

a. Roulette Wheel

Pemilihan parent dari populasi menggunakan Roulette Wheel dilakukan dengan cara menghitung jumlah keseluruhan fitness yang ada lalu memilih sebuah bilangan

acak antara 1 sampai dengan hasil penjumlahan keseluruhan fitness yang ada. Selanjutnya lakukan penelusuran pada populasi yang ada sambil menjumlahkan fitness dari masing-masing individu. Jika penjumlahan ini sudah melebihi bilangan acak yang dipilih sebelumnya, maka parent yang dipilih dari populasi tersebut adalah individu yang fitnessnya terakhir ditambahkan.

b. Rank Selection

Pemilihan parent menggunakan Rank Selection diawali dengan melakukan pengurutan individu di dalam populasi berdasarkan nilai fitnessnya. Individu dengan nilai fitness paling kecil berada di index pertama. Selanjutnya, menentukan sebuah bilangan acak dari angka 1 sampai dengan $(ukuranPopulasi \times (ukuranPopulasi+1)/2)$. Pada array populasi yang sudah terurut berdasarkan *rank*, dilakukan penelusuran sambil menjumlahkan fitness dari masing-masing individu. Jika hasil penjumlahan sudah melebihi bilangan acak yang telah dipilih sebelumnya, maka individu yang dipilih sebagai parent adalah individu yang nilai fitnessnya terakhir ditambahkan.

Selanjutnya, pencarian solusi dapat dilakukan dengan algoritma genetik seperti contoh kasus sederhana seperti berikut.

1. Input

a. Papan *mosaic*

Diberikan papan *mosaic* seperti gambar di bawah.

		3		
1	3		3	
	2	2		0
		1		2
	1			

5x5 Easy Mosaic Puzzle ID: 11,065,139

Papan ini kemudian di-encoding menjadi array 1 dimensi yang menggambarkan kromosom dari permasalahan ini dan gene dari kromosom tersebut berisi angka-angka dari kotak-kotak pada papan *mosaic*.

-1	-1	3	-1	-1	1	3	-1	3	-1	-1	2	2	-1	0	-1	-1	1	-1	2	-1	1	-1	-1	-1
----	----	---	----	----	---	---	----	---	----	----	---	---	----	---	----	----	---	----	---	----	---	----	----	----

b. Jumlah Populasi

Menentukan jumlah populasi dari permasalahan ini yang kemudian akan membentuk populasi awal untuk mencari individu-individu parent. Untuk mencari solusi dari kasus ini, diberikan ukuran populasi misalnya sebesar 10.000.

2. Membentuk Populasi Awal

Setelah memiliki ukuran populasi, pertama lakukan inisiasi populasi awal yang memiliki individu acak sejumlah ukuran populasi. Populasi awal digunakan untuk menghasilkan generasi selanjutnya. Populasi awal juga menjadi generasi ke-1.

3. Menghitung Fitness

Setiap individu di dalam populasi dihitung fitnessnya menggunakan rumus yang telah dijelaskan sebelumnya. Setelah semua individu dihitung fitnessnya, telusuri kembali semua individu untuk mencari individu dengan fitness terbaik dan individu dengan fitness terburuk. Untuk contoh kasus ini, fitness solusi benar adalah sebesar 900.

Generasi ke-1 Fitness terbesar: 896 Fitness terkecil: 766 Fitness benar: 900
Generasi ke-2 Fitness terbesar: 896 Fitness terkecil: 769 Fitness benar: 900
Generasi ke-3 Fitness terbesar: 893 Fitness terkecil: 773 Fitness benar: 900
Generasi ke-4 Fitness terbesar: 897 Fitness terkecil: 772 Fitness benar: 900

4. Menentukan Parent Melalui Selection

Setelah memiliki populasi awal, dilakukan seleksi parent dari individu-individu yang ada pada populasi. Seleksi parent dilakukan secara berulang pada setiap pembentukan generasi sampai ditemukan generasi yang merupakan solusi optimal untuk contoh kasus ini. Seleksi parent dapat dilakukan dengan cara *Roulette Wheel* atau *Rank Selection*.

5. Melakukan Crossover

Parent yang telah dipilih melalui seleksi kemudian dilakukan crossover dengan menukar bagian kromosom pada parent yang sudah ditandai dengan *crossover point*.

6. Melakukan Pengecekan Kemungkinan Mutasi

Setelah menghasilkan generasi baru dari hasil crossover, dilakukan pengecekan kemungkinan mutasi pada individu tersebut. Kemungkinan mutasi dihitung secara acak dari nilai probabilitas mutasi. Jika didapatkan nilai <1 , maka individu tersebut akan mengalami mutasi pada salah satu gene di dalam kromosomnya. Pemilihan gene ini dilakukan secara acak. Misalnya jika pada gene terpilih berisikan angka 1, maka proses mutasi akan membuat gene tersebut berubah menjadi angka 0.

7. Melakukan Pengecekan Generasi

Setiap mendapatkan generasi baru, dilakukan pengecekan nilai fitness apakah sudah mencapai nilai 900 atau belum dan apakah generasi saat ini sudah mencapai batas generasi atau belum. Jika kedua hal ini sudah tercapai, maka akan didapatkan generasi yang merupakan solusi dari kasus ini. Jika tidak, maka akan terus dilakukan pencarian dengan mengulangi langkah-langkah pembentukan generasi. Gambar berikut menunjukkan hasil pencarian menggunakan algoritma genetik untuk contoh kasus ini.

```
Generasi solusi berada pada generasi populasi ke-82
Waktu eksekusi: 24.6947593 detik
Fitness akhir: 900/900
Genes terbaik:
0 0 1 0 1
0 1 1 0 0
0 0 0 0 0
0 0 0 0 0
1 0 0 1 1
```

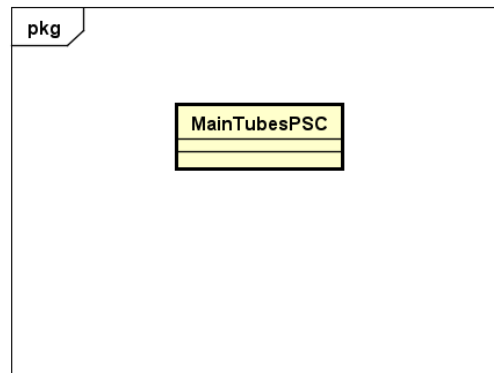
BAB 4

Implementasi Perangkat Lunak

4.1 Diagram Kelas

Pembuatan program dibuat dengan memakai pendekatan *Object Oriented Programming* (OOP). Diagram kelas digunakan untuk mendeskripsikan *method-method* pada suatu objek, beserta hubungan antar objek. Untuk membangun sebuah algoritma genetika diperlukan kelas-kelas yang dapat merepresentasikan generasi, individu, dan populasi. Oleh karena itu, pada eksperimen berikut dibuatlah kelas-kelas yang terdiri dari kelas Individu, kelas Generasi, dan kelas Populasi. Selain itu, untuk menggabungkan ketiga buah kelas tersebut, dibuatlah kelas utama agar dapat menghubungkan ketiga kelas tersebut sekaligus sebagai pintu masuk utama untuk input dan output.

4.1.1 Kelas MainTubesPSC



Gambar 4.1: Diagram Kelas MainTubesPSC.java

Kelas MainTubesPSC berfungsi sebagai kelas utama dan digunakan untuk melakukan input dan output dalam program. Kelas ini tidak memiliki atribut apapun. Sementara itu, kelas ini memiliki *method-method* sebagai berikut:

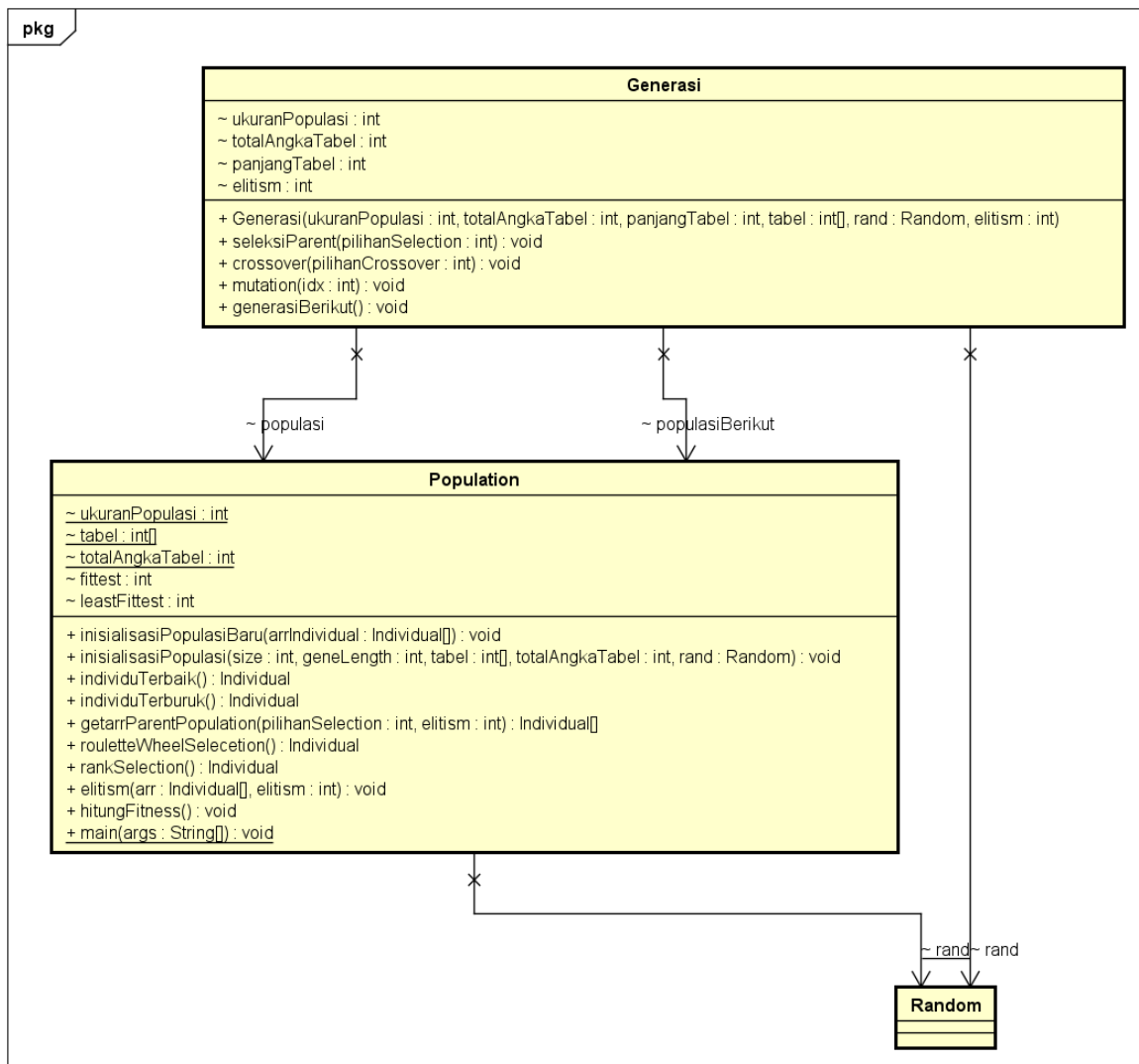
- `main(String[]): void`

Method ini bertugas melakukan pemanggilan ke kelas Generasi dan memberikan konfigurasi-konfigurasi yang diperlukan untuk memulai pengujian.

Input: file dengan format .txt

Output: file dengan format .txt

4.1.2 Kelas Populasi



Gambar 4.2: Diagram Kelas Populasi.java

Kelas Populasi digunakan untuk merepresentasikan sebuah populasi dalam algoritma genetika dan sekaligus menyimpan *method-method* untuk melakukan seleksi. Atribut-atribut dalam kelas ini adalah: Berikut *method-method* pada kelas populasi:

- `static int ukuranPopulasi`
Variabel untuk menyimpan banyaknya kromosom untuk setiap populasi.
- `static int[] tabel`
Variable untuk menyimpan tabel soal yang dimasukan.
- `static int totalAngkaTabel`
Variable untuk menghitung banyaknya angka yang berada pada tabel soal.

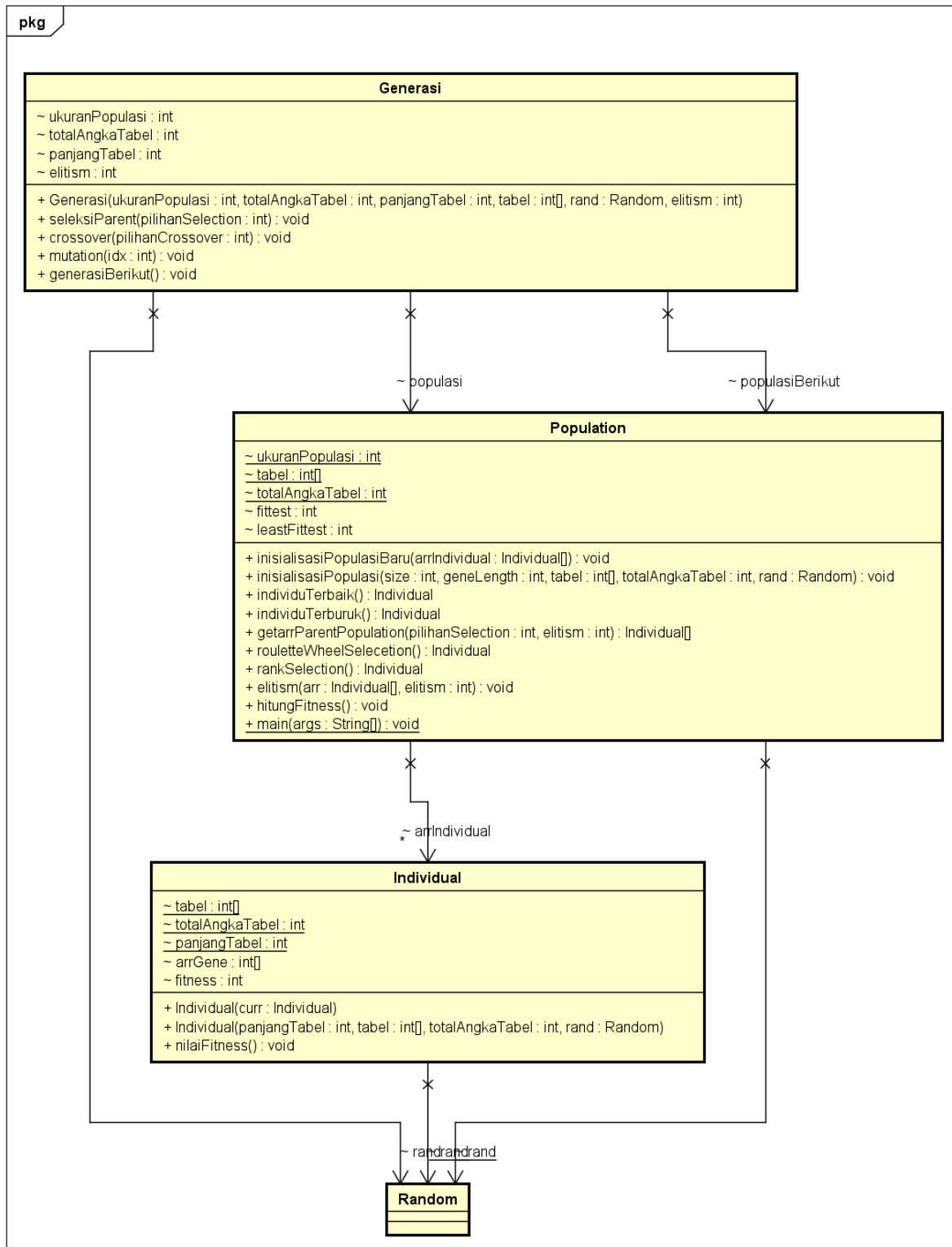
- `static Random rand;`
Sebuah pseudorandom number generator.
- `Individual[] arrIndividual`
Variabel untuk merepresentasikan populasi .
- `int fittest`
Variabel untuk menyimpan fitness terbaik dari sebuah populasi.
- `int leastFittest`
Variabel untuk menyimpan fitness terburuk dari sebuah populasi.

Selain itu, kelas ini juga memiliki *method-method* sebagai berikut:

- `inisialisasiPopulasiBaru(Individual[] arrIndividual) : void`
Method ini bertugas untuk menginisialisasi populasi baru menggunakan array of Individual.
Input: array individual
Output: -
- `inisialisasiPopulasi(int size, int geneLength, int[] tabel, int totalAngkaTabel, Random rand) : void`
Method ini bertugas untuk menginisialisasi populasi baru menggunakan variabel ukuran.
Input: bilangan bulat ukuran array individual
Output: -
- `individuTerbaik() : Individual`
Method untuk mencari dan mendapatkan individu dengan nilai fitness terbaik
Input: kromosom-kromosom individu
Output: -
- `individuTerburuk() : Individual`
Method untuk mencari dan mendapatkan individu dengan nilai fitness terburuk
Input: kromosom-kromosom individu
Output: -
- `getarrParentPopulation(int pilihanSelection, int elitism) : Individual[]`
Method untuk mendapatkan offspring baru dari tiap parent dengan melakukan elitism dan seleksi
Input: array individu
Output: array baru hasil seleksi
- `rouletteWheelSelecection() : Individual`
Method ini bertugas melakukan seleksi menggunakan metode roulette wheel
Input: file dengan format .txt
Output: array baru hasil seleksi
- `ranklSelecection() : Individual`
Method ini bertugas melakukan seleksi menggunakan metode rank selection
Input: array individu
Output: array baru hasil seleksi

- elitism(Individual arr[], int elitism) : void
Method ini bertugas melakukan elitism yaitu dengan mempertahankan individu-individu terbaik
Input: array individu
Output: -
- hitungFitness() : void
Method ini berfungsi untuk menghitung nilai fitness dari masing-masing individu
Input: array individu
Output: nilai fitness

4.1.3 Kelas Generasi



Gambar 4.3: Diagram Kelas Generasi.java

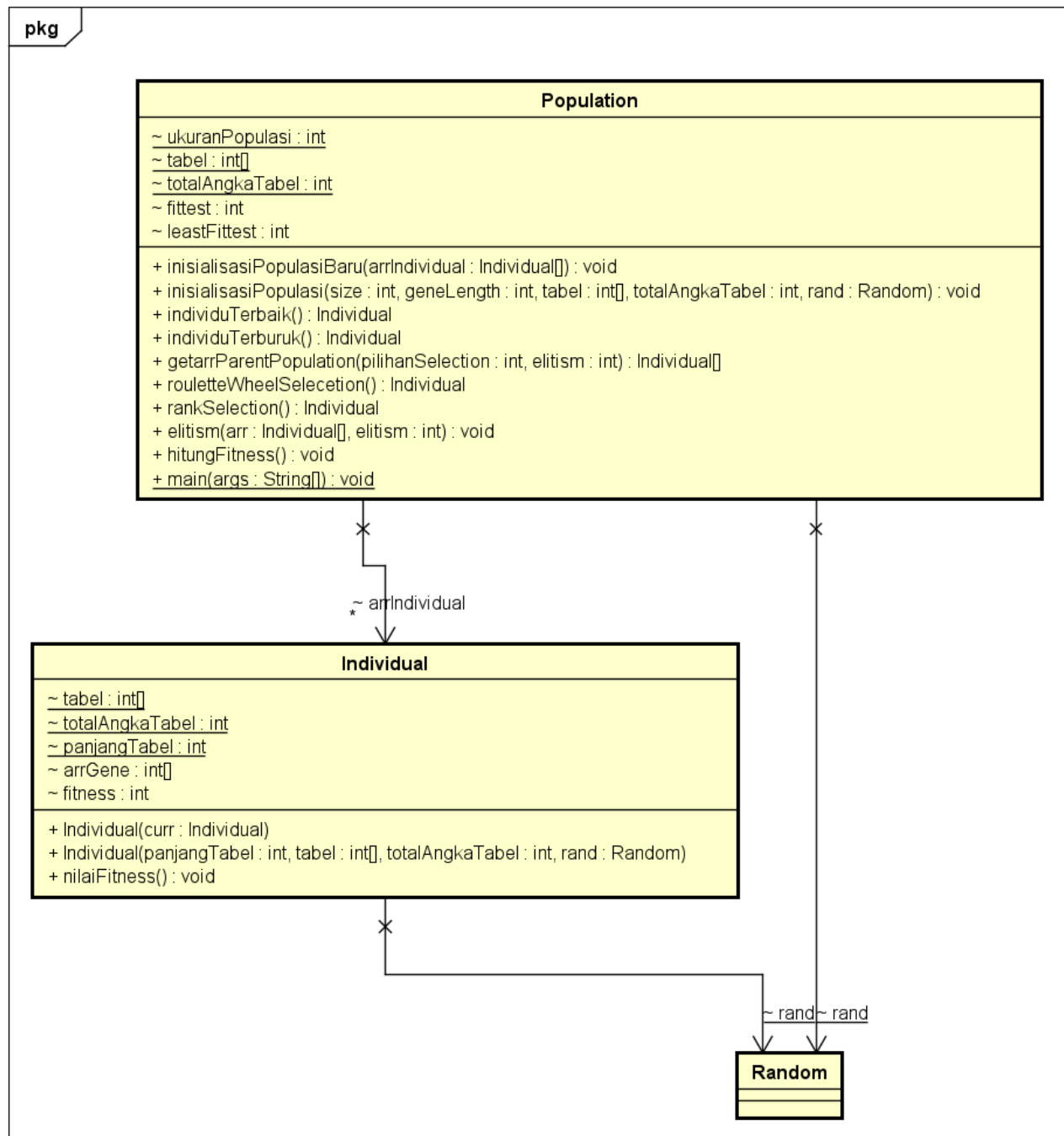
Kelas Generasi digunakan untuk merepresentasikan generasi dan menyimpan populasi-populasi dalam pelaksanaan algoritma genetika dan sekaligus menyimpan metode-metode crossover. Atribut-atribut dalam kelas ini adalah:

- `Population populasi = new Population()`
Membuat atribut populasi dan menginisialisasikan nya
- `Population populasiBerikut`
Membuat atribut untuk menyimpan populasi generasi berikutnya
- `int ukuranPopulasi;`
Membuat atribut untuk ukuran populasi
- `int totalAngkaTabel`
Membuat atribut untuk menghitung berapa banyak angka pada tabel awal
- `int panjangTabel`
Membuat atribut untuk menyimpan panjang tabel
- `int elitism`
Membuat atribut untuk menyimpan banyaknya individu yang mau dilakukan elitism
- `Random rand`
Membuat variabel untuk seedRandom

Selain itu, kelas ini juga memiliki *method-method* sebagai berikut:

- `seleksiParent(int pilihanSelection) : void`
Method untuk menseleksi parent dan disimpan dalam atribut `populasiBerikut`
Input: populasi dan pilihan seleksi
Output: -
- `crossover(int pilihanCrossover) : void`
Method ini bertugas melakukan crossover atau kawin silang berdasarkan operator algoritma genetika
Input: populasi dan pilihan crossover
Output: -
- `mutation(int idx) : void`
Method ini bertugas melakukan mutasi berdasarkan operator algoritma genetika
Input: populasi dan pilihan mutasi
Output: -
- `generasiBerikut() : void`
Method ini berfungsi mengubah populasi lama ke populasi yang baru setelah melewati operator algoritma genetika
Input: -
Output: -

4.1.4 Kelas Individual



Gambar 4.4: Diagram Kelas Individual.java

Kelas Individual digunakan untuk merepresentasikan sebuah kromosom dalam algoritma genetika.

Atribut-atribut dalam kelas ini adalah:

- `static int[] tabel`
Variable untuk menyimpan tabel soal yang dimasukan
- `static int totalAngkaTabel`
Variable untuk menghitung banyaknya angka yang berada pada tabel soal
- `static int panjangTabel`
Variabel untuk menyimpan panjang tabel
- `static Random rand`
Variabel untuk menggunakan random yang sudah di inisialisasi di awal
- `int[] arrGene`
Variabel untuk merepresentasikan kromosom
- `int fitness`
Variabel untuk menyimpan fitness dari kromosom tersebut

Selain itu, kelas ini juga memiliki *method-method* sebagai berikut:

- `nilaiFitness() : void`
Method ini berfungsi untuk menghitung nilai fitness dari tiap generasi
Input: individu
Output: -

4.2 Rancangan Operator genetika

Implementasi dari rancangan operator algoritma genetika di bab sebelumnya ke dalam bahasa Java dapat dilihat pada subbab-subbab berikut:

4.2.1 Seleksi

Implementasi untuk operasi seleksi pada penelitian ini menggunakan metode seleksi roulette wheel dan rank selection. Berikut masing-masing implementasinya yang terdapat pada Kode 4.1 dan Kode 4,2

```

1  public Individual rouletteWheelSelecection(){
2      int totalSum = 0;
3      for(int i =0; i < this.ukuranPopulasi; i++){
4          totalSum += this.arrIndividual[i].fitness;
5      }
6      int random = rand.nextInt(totalSum-1);
7      random++;
8      int sum = 0;
9      for(int i =0; i < this.ukuranPopulasi; i++){
10         sum += this.arrIndividual[i].fitness;
11         if(sum ≥ random){
12             return this.arrIndividual[i];
13         }
14     }
15     return null;
16 }

```

Kode 4.1: Metode seleksi *Roulette Wheel*

```

1  public Individual rankSelection(){
2      int[][] arrTemp = new int[ukuranPopulasi][2];
3      for(int i =0; i < ukuranPopulasi; i++){
4          arrTemp[i][0] = this.arrIndividual[i].fitness;
5          arrTemp[i][1] = i;
6      }
7      Arrays.sort(arrTemp, (int[] pertama, int[] kedua) → {
8          if(pertama[0] < kedua[0]){
9              return -1;
10         }else if(pertama[0] > kedua[0]){
11             return 1;
12         }else{
13             return 0;
14         }
15     });
16     int randomPoint;
17     if(ukuranPopulasi%2 == 0){
18         randomPoint = rand.nextInt((ukuranPopulasi/2)*(ukuranPopulasi+1));
19     }else{
20         randomPoint = rand.nextInt((ukuranPopulasi)*((ukuranPopulasi+1)/2));
21     }
22     randomPoint++;
23     int sum = 0;
24     for(int i =1; i ≤ this.ukuranPopulasi; i++){
25         sum += i;
26         if(sum ≥ randomPoint){
27             return this.arrIndividual[arrTemp[i-1][1]];
28         }
29     }
30     return null;
31 }

```

Kode 4.2: Metode seleksi *Rank*

4.2.2 Crossover

Implementasi untuk operasi crossover pada penelitian ini menggunakan metode 1-point crossover dan 2-point crossover . Berikut masing-masing implementasinya yang terdapat pada Kode 4.3 dan Kode 4.4

```
1 public void crossover(int pilihanCrossover) {
2     if(pilihanCrossover == 1){
3         int titikCrossoverPertama = Math.abs(rand.nextInt(panjangTabel));
4         for (int j = 0; j < panjangTabel; j++) {
5             if(j ≤ titikCrossoverPertama ){
6                 int temp = populasiBerikut.arrIndividual[i].arrGene[j];
7                 populasiBerikut.arrIndividual[i].arrGene[j] = populasiBerikut.arrIndividual[i+1].arrGene[j];
8                 populasiBerikut.arrIndividual[i+1].arrGene[j]= temp;
9             }
10        }
11    }
12 }
```

Kode 4.3: Metode *1-point crossover*

```
1 else {
2     int titikCrossoverPertama = Math.abs(rand.nextInt(panjangTabel));
3     int titikCrossoverKedua = Math.abs(rand.nextInt(panjangTabel));
4     while(Math.abs(titikCrossoverPertama - titikCrossoverKedua) == 0
5         || Math.abs(titikCrossoverPertama - titikCrossoverKedua) == (panjangTabel-1)){
6         titikCrossoverPertama = Math.abs(rand.nextInt(panjangTabel));
7         titikCrossoverKedua = Math.abs(rand.nextInt(panjangTabel));
8     }
9     for (int j = 0; j < panjangTabel; j++) {
10        if((j ≤ titikCrossoverPertama && j ≥ titikCrossoverKedua)
11        || (j ≤ titikCrossoverKedua && j ≥ titikCrossoverPertama )){
12            int temp = populasiBerikut.arrIndividual[i].arrGene[j];
13            populasiBerikut.arrIndividual[i].arrGene[j] = populasiBerikut.arrIndividual[i+1].arrGene[j];
14            populasiBerikut.arrIndividual[i+1].arrGene[j]= temp;
15        }
16    }
17 }
```

Kode 4.4: Metode *2-point crossover*

4.2.3 Elitism

Berikut implementasi untuk operasi elitism pada penelitian ini yang terdapat pada Kode 4.5

```
1 public void elitism(Individual arr[], int elitism){
2     int[][] arrTemp = new int[ukuranPopulasi][2];
3     for(int i =0; i < ukuranPopulasi; i++){
4         arrTemp[i][0] = this.arrIndividual[i].fitness;
5         arrTemp[i][1] = i;
6     }
7     Arrays.sort(arrTemp, (int[] pertama, int[] kedua) → {
8         if(pertama[0] > kedua[0]){
9             return -1;
10        }else if(pertama[0] < kedua[0]){
11            return 1;
12        }else{
13            return 0;
14        }
15    });
16    for(int i =0; i < elitism; i++){
17        arr[i] = this.arrIndividual[arrTemp[i][1]];
18    }
19 }
```

Kode 4.5: Metode *elitism*

4.2.4 Mutasi

Berikut implementasi untuk operasi mutasi pada penelitian ini yang terdapat pada Kode 4.6

```
1 public void mutation(int idx) {
2     int mutationPoint = rand.nextInt(panjangTabel);
3     if(populasiBerikut.arrIndividual[idx].arrGene[mutationPoint] == 0){
4         populasiBerikut.arrIndividual[idx].arrGene[mutationPoint] = 1;
5     }else{
6         populasiBerikut.arrIndividual[idx].arrGene[mutationPoint] = 0;
7     }
8 }
```

Kode 4.6: Metode mutasi

4.3 Spesifikasi input dan output

Masukan dan keluaran pada program berikut menggunakan file dengan format txt. Ketentuan masukan program terdapat pada Subbagian 4.3.1 dan hasil keluaran terdapat pada Subbagian 4.3.2

4.3.1 Input

Berikut merupakan spesifikasi masukan untuk mengeksekusi program:

1. Seed random (long)
Membuat variabel untuk variabel random
2. Ukuran populasi (integer)
Menginisialisasi ukuran populasi
3. Lebar papan (integer)
Ukuran dari papan *mosaic*, namun cukup menuliskan panjang/lebar papan
4. Batas generasi (integer)
Jumlah batas generasi dalam algoritma genetika
5. Probabilitas mutasi (integer)
Nilai masukan akan dibagi 1 (menjadi 1/probabilitas mutasi)
6. Pilihan metode seleksi (integer)
Roulette wheel = 1, Rank Selection = 2
7. Pilihan jumlah titik crossover (integer)
Crossover 1 point = 1, Crossover 2 point = 2
8. Banyak elisitm (integer)
Jumlah individu yang akan dilakukan elisitm
9. Papan (array 2 dimensi)
Memasukkan angka-angka pada papan mosaic, untuk sel yang tidak terdapat angka ditulis -1

```
123456
10000
5
20000
10000
1
1
1000
-1 -1 -1 -1 -1
-1 4 -1 3 -1
3 -1 1 4 -1
-1 2 -1 -1 4
-1 -1 0 -1 -1
```

Gambar 4.5: Contoh masukan untuk program dalam bentuk format .txt

4.3.1 Output

Berikut merupakan spesifikasi keluaran dari eksekusi program:

1. Generasi Fitness
Hasil tiap generasi yang terbentuk (Generasi, Fitness terbesar, Fitness terkecil, dan Fitness benar)
2. Generasi Solusi
Keluaran jika sebuah solusi sudah ditemukan atau sampai batas akhir generasi
3. Waktu eksekusi
Waktu yang diperlukan untuk mencari sebuah solusi
4. Fitness akhir
Nilai fitness terbaik dari sebuah generasi
5. Tabel Solusi terbaik
Solusi kotak hitam dari sel-sel pada papan *mosaic*

```
Generasi ke-1 Fitness terbesar: 712 Fitness terkecil: 605 Fitness benar: 720
Generasi ke-2 Fitness terbesar: 715 Fitness terkecil: 609 Fitness benar: 720
Generasi ke-3 Fitness terbesar: 716 Fitness terkecil: 608 Fitness benar: 720
Generasi ke-4 Fitness terbesar: 717 Fitness terkecil: 603 Fitness benar: 720
Generasi ke-5 Fitness terbesar: 718 Fitness terkecil: 598 Fitness benar: 720
Generasi ke-6 Fitness terbesar: 717 Fitness terkecil: 610 Fitness benar: 720
Generasi ke-7 Fitness terbesar: 719 Fitness terkecil: 609 Fitness benar: 720
Generasi ke-8 Fitness terbesar: 717 Fitness terkecil: 622 Fitness benar: 720
Generasi ke-9 Fitness terbesar: 719 Fitness terkecil: 623 Fitness benar: 720
Generasi ke-10 Fitness terbesar: 719 Fitness terkecil: 625 Fitness benar: 720
Generasi ke-11 Fitness terbesar: 719 Fitness terkecil: 618 Fitness benar: 720
Generasi ke-12 Fitness terbesar: 720 Fitness terkecil: 623 Fitness benar: 720
Generasi solusi berada pada generasi populasi ke-13
Waktu eksekusi: 1.004087599 detik
Fitness akhir: 720/720
Genes terbaik:
1 1 0 0 0
1 0 0 0 1
1 0 0 1 1
1 0 0 0 1
0 0 0 0 1
```

Gambar 4.6: Contoh keluaran program dalam bentuk format .txt

4.4 Penambahan Fitur untuk Mencatat Waktu Eksekusi

Pada kelas main, terdapat penambahan kode pada baris 35, 62, 63, dan 65 yang berfungsi untuk mencatat lama waktu eksekusi program. Penambahan kode tersebut tidak mempengaruhi cara kerja program secara keseluruhan, namun sedikit mengubah format output karena menampilkan catatan lama waktu eksekusi. Berikut potongan kode program tersebut:

```
double Time_start = System.nanoTime();
double Time_finish = System.nanoTime();
double waktuTotal = (Time_finish - Time_start) / 1000000000;
fileWriter.write("Waktu eksekusi: " + waktuTotal + " detik");
```


BAB 5

Eksperimen

Eksperimen ini dilakukan dengan cara menguji papan *mosaic* berukuran 5x5, 7x7, 10x10 di level *easy* dan hard, dan papan *mosaic* berukuran 15x15 dan 20x20 di level *easy*. Pada bab ini, dibahas lingkungan dan parameter yang dipakai oleh agen dalam eksperimen.

5.1 Lingkungan Eksperimen

Komputer lokal digunakan untuk melakukan pengujian. Pengujian dilakukan di dalam IDE Visual Studio Code 2022.

Berikut detail dari sistem yang digunakan ketika melakukan pengujian di komputer lokal:

OS : Microsoft Windows 11 Home
CPU : Intel(R) Core(TM) I5-1135G7 CPU 8 cores 2.40GHz
RAM : 16GB
Java Version : JDK 19.0.1
IDE : Visual Studio Code 1.74.2

5.1.1 Model Papan

Berikut jenis model-model papan *mosaic* yang akan diuji:

		3		
1	3		3	
	2	2		0
		1		2
	1			

5x5 Easy Mosaic Puzzle ID: 11,065,139

	5		4	
3		8	7	
	5	7		5
1	4			4
		4		

5x5 Hard Mosaic Puzzle ID: 11,725,567

3				3	3	
		4				
6	7			0		2
		5		3		4
					7	
	6	7		5		
					3	

7x7 Easy Mosaic Puzzle ID: 9,428,188

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

7x7 Hard Mosaic Puzzle ID: 5,010,516

				4	6		4	
	6	6			7			
1				4	6	7		3
0	3		7		6	6		
	5		9		8	7		5
			8		5			4
	3		6	6			4	
2		2		6	6	5	4	
				6	6			2
3		3	4	5	4	3		1

10x10 Easy Mosaic Puzzle ID: 1,004,005

					2	4		3
3		4		5	5			
	4	3		7	6		6	
	4	2		3	5		5	4
2					4	4		2
	4	5	4	3			4	
2		3			3	4	4	3
3	5	4	3		2		5	5
				3		2	3	4
3	4				2	2		2

10x10 Hard Mosaic Puzzle ID: 700,595

2		2	3	2	2	0		2			2		
								7					1
	6	7	5					5	5	6	5	4	1
3			6		3	6		5	5			3	2
	3				2			5	6	4		3	3
		5		4		2		4		5	4		4
3			7			2	4	5		6		4	3
	5		7		6			4		4	2		4
3		4		5		4	3		4	5		5	4
		4	4		4		3		2	3		5	
		4	4	4			4	3	2			6	
			4			5		2	0	1		6	
4		5			4		4	3	2		4		2
		7	5	6				2					6
	4					4						4	

15x15 Easy Mosaic Puzzle ID: 2,607,878

3	3		1		3			2	2					3
		4		5	4			2	2		3	4		5
		5	6	5	4		2		3	3	2		5	
2		8	8	7	4	3						2	4	
3		9	8		4	3		4	3				5	
	5	7		6					3			7	6	
	5	7				4	3	3				5	5	
3		6	4	5	3	3	3	3	6			4	3	
		7	5			4		2	6	6		3		
			3				3		6	6			2	
	7			1	3		3	1		3	5			
4			3	1	3		2		2				5	
4				3	3	2	2	2		2	3	5		
		4	3		3	3		4				6		
			3		4	6			6	2		4		
					5	7	8	8	5		3	3	4	
	4	6	4	6		6		6			3			
		5			3	4	3	4		3	5	4		
			2	3	4	4		2		3	4		6	
4							2		0	0				
													3	
														1

20x20 Easy Mosaic Puzzle ID: 4,930,495

5.1.2 Parameter Tetap

Pengujian diatur menggunakan parameter-parameter tertentu agar agen dapat bermain secara general. Hal ini juga dilakukan dikarenakan keterbatasan resource dan waktu untuk melakukan eksperimen.

Berikut diberikan detail parameter akan yang digunakan untuk melakukan pelatihan pada agen untuk setiap gim

- Seed random : 123456
- Ukuran populasi : 10000
- Batas generasi : 1500
- Mutasi : 0,0001

5.2 Uji Parameter

Selain itu, terdapat juga parameter-parameter dinamis yang akan diujikan dalam melakukan eksperimen. Pengujian dilakukan dengan memperhatikan perubahan pada model seleksi, jumlah titik crossover, dan jumlah elitism yang digunakan.

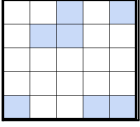
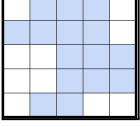
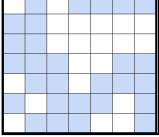
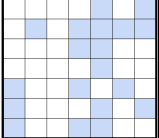
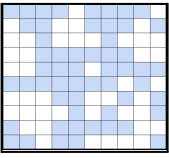
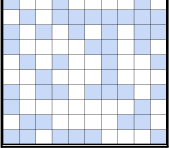
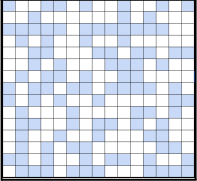
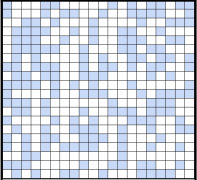
Berikut detail dari masing-masing parameter yang akan diuji:

Parameter 1	Parameter 2	Parameter 3	Parameter 4
roulette wheel	Rank selection	roulette wheel	Rank selection
crossover 1 point	crossover 1 point	crossover 2 point	crossover 2 point
elitism 10%	elitism 10%	elitism 10%	elitism 10%

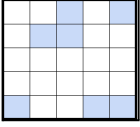
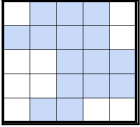
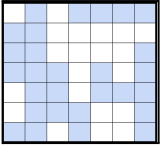
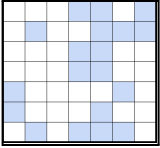
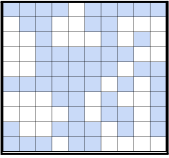
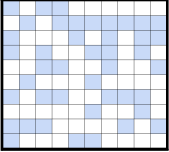
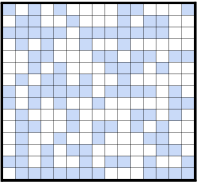
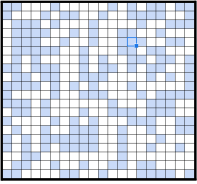
Parameter 5	Parameter 6	Parameter 7	Parameter 8
roulette wheel	Rank selection	roulette wheel	Rank selection
crossover 1 point	crossover 1 point	crossover 2 point	crossover 2 point
elitism 1%	elitism 1%	elitism 1%	elitism 1%

Parameter 9	Parameter 10	Parameter 11	Parameter 12
roulette wheel	Rank selection	roulette wheel	Rank selection
crossover 1 point	crossover 1 point	crossover 2 point	crossover 2 point
elitism 0.1%	elitism 0.1%	elitism 0.1%	elitism 0.1%

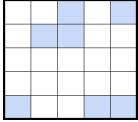
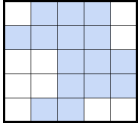
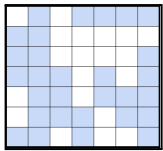
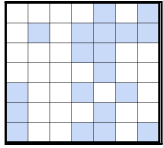
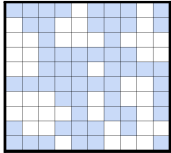
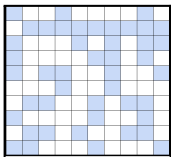
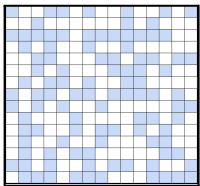
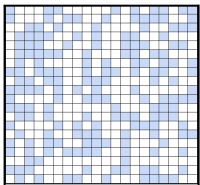
5.2.1 Uji Parameter Pertama

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	10	0.837 detik	900/900	Optimal	
5x5	<i>Hard</i>	10	0.879 detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	151.704 detik	1420/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	178.447 detik	2337/2340	Sub-Optimal	
10x10	<i>Easy</i>	1500	194.975 detik	3957/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	190.473 detik	4584/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	224.830 detik	9485/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	381.624 detik	17633/17820	Sub-Optimal	

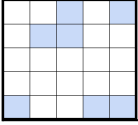
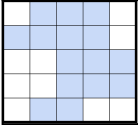
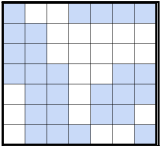
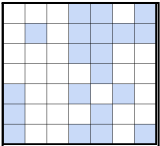
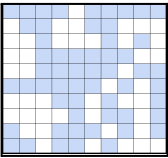
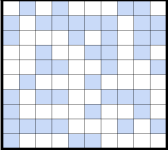
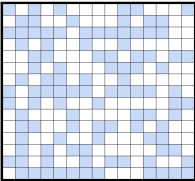
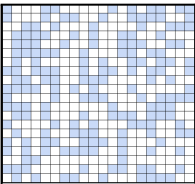
5.2.2 Uji Parameter Kedua

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	8	77.797 detik	900/900	Optimal	
5x5	<i>Hard</i>	6	59.846 detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	3674.824 detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	20	241.890 detik	2340/2340	Optimal	
10x10	<i>Easy</i>	1500	5718.726 detik	3957/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	5695.311 detik	4581/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	5055.677 detik	9481/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	5173.408 detik	17643/17820	Sub-Optimal	

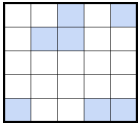
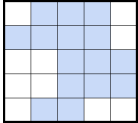
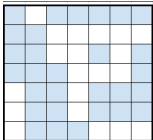
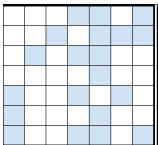
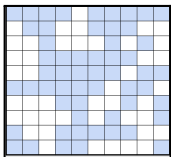
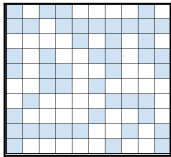
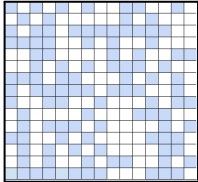
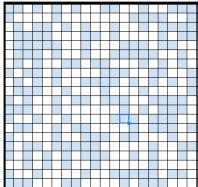
5.2.3 Uji Parameter Ketiga

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	21	1.934 detik	900/900	Optimal	
5x5	<i>Hard</i>	11	0.933 detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	153.309 detik	1417/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	182.172 detik	2337/2340	Sub-Optimal	
10x10	<i>Easy</i>	1500	172.096 detik	3957/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	186.461 detik	4581/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	254,770 detik	9508/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	384.189 detik	17710/17820	Sub-Optimal	

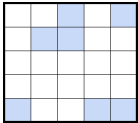
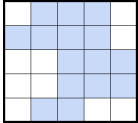
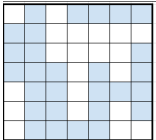
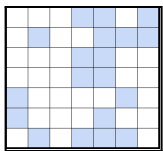
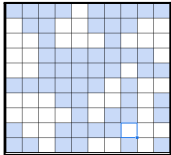
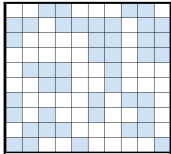
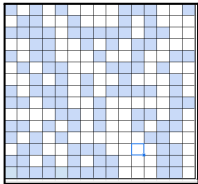
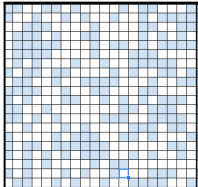
5.2.4 Uji Parameter Keempat

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	8	75.796 detik	900/900	Optimal	
5x5	<i>Hard</i>	8	86.592 detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	3824.544 detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	3561.481 detik	2337/2340	Sub-Optimal	
10x10	<i>Easy</i>	1500	4835.657 detik	3957/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	5055.516 detik	4581/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	10366.163 detik	9492/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	5388.69 detik	17715/17820	Sub-Optimal	

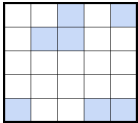
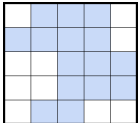
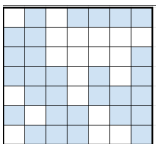
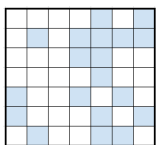
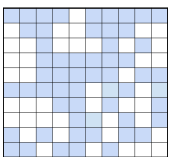
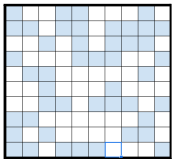
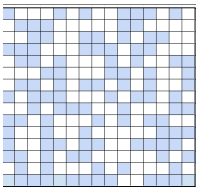
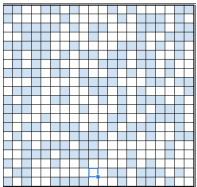
5.2.5 Uji Parameter Kelima

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	17	4.5395 Detik	900/900	Optimal	
5x5	<i>Hard</i>	36	8.7311 Detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	408.1482 Detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	515.2299 Detik	2335/2340	Sub-Optimal	
10x10	<i>Easy</i>	1500	1804.733 Detik	3957/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	1914.302 Detik	4571/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	1827.095 Detik	9465/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	1504.063	17575/17820	Sub-Optimal	

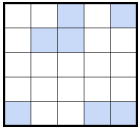
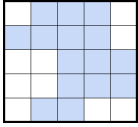
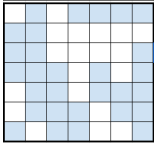
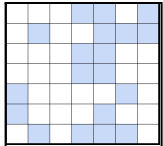
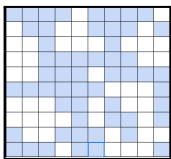
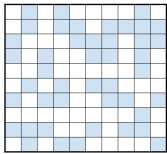
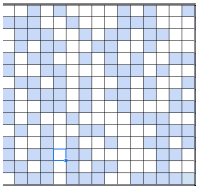
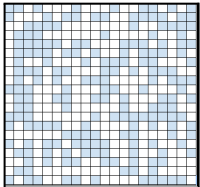
5.2.6 Uji Parameter Keenam

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	9	161.252 Detik	900/900	Optimal	
5x5	<i>Hard</i>	8	138.696 Detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	6459.258 Detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	20	1069.284 Detik	2340/2340	Optimal	
10x10	<i>Easy</i>	1500	13158.557 Detik	3952/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	9118.623 Detik	4574/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	10454.219 Detik	9469/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	12220.586 Detik	17622/17820	Sub-Optimal	

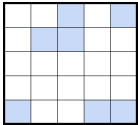
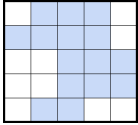
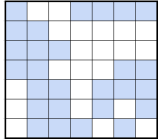
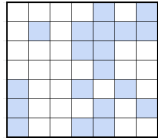
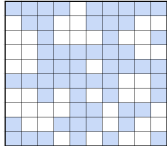
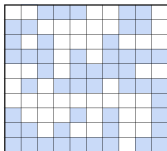
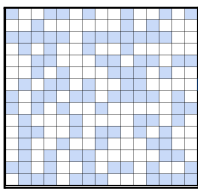
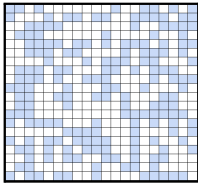
5.2.7 Uji Parameter Ketujuh

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	30	9.298 Detik	900/900	Optimal	
5x5	<i>Hard</i>	14	3.521 Detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	622.133 Detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	674.510 Detik	2337/2340	Sub-Optimal	
10x10	<i>Easy</i>	1500	1804.203 Detik	3951/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	1907.095 Detik	4569/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	1822.898 Detik	9454/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	1451.806 Detik	17626/17820	Sub-Optimal	

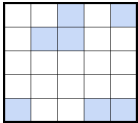
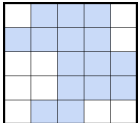
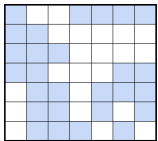
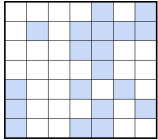
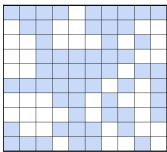
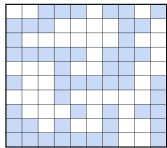
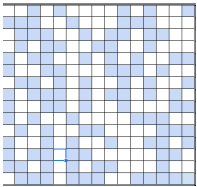
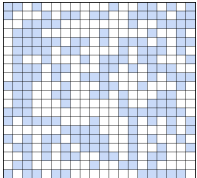
5.2.8 Uji Parameter Kedelapan

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	12	187.8816 Detik	900/900	Optimal	
5x5	<i>Hard</i>	11	190.229 Detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	6881.849 Detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	28	1257.142 Detik	2340/2340	Optimal	
10x10	<i>Easy</i>	1500	13452.114 Detik	3957/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	9120.924 Detik	4586/4590	Sub-Optimal	
15x15	<i>Easy</i>	1500	10527.475 Detik	9502/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	12321.385 Detik	17699/17820	Sub-Optimal	

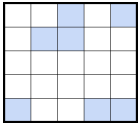
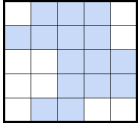
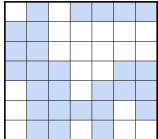
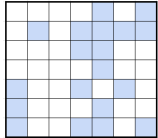
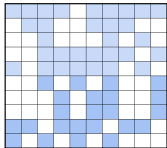
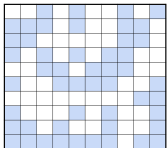
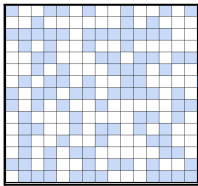
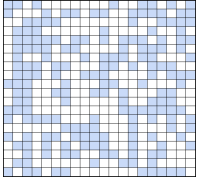
5.2.9 Uji Parameter Kesembilan

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	81	24.6947593 detik	900/900	Optimal	
5x5	<i>Hard</i>	116	32.3605065 detik	1080/1080	Optimal	
7x7	<i>Easy</i>	5000	435.469 detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	5000	408.518 detik	2424/2430	Sub-Optimal	
10x10	<i>Easy</i>	1500	444.632 detik	3947/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	431.216 detik	4289/4320	Sub-Optimal	
15x15	<i>Easy</i>	1500	945.374 detik	9390/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	662.194 detik	17371/17820	Sub-Optimal	

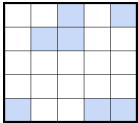
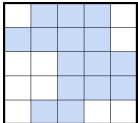
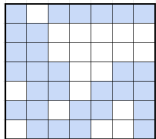
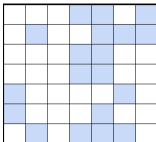
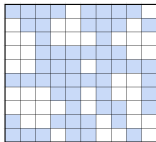
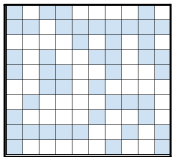
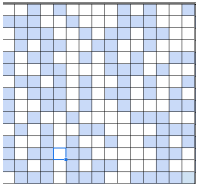
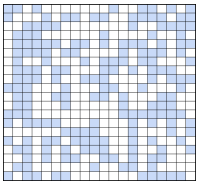
5.2.10 Uji Parameter Kesepuluh

Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	9	129.6573452 detik	900	Optimal	
5x5	<i>Hard</i>	9	156.3298811 detik	1080	Optimal	
7x7	<i>Easy</i>	1500	5704.681 detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	7079.987 detik	2424/2430	Sub-Optimal	
10x10	<i>Easy</i>	1500	7440.273 detik	3957/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	6827.755 detik	4293/4320	Sub-Optimal	
15x15	<i>Easy</i>	1500	7170.579 detik	9482/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	9170.812 detik	17588/17820	Sub-Optimal	

5.2.11 Uji Parameter Kesebelas

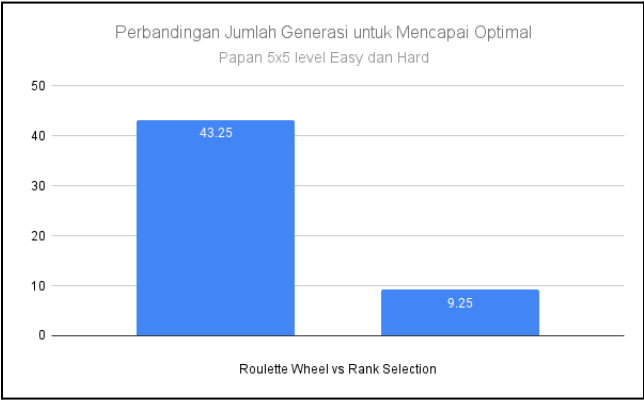
Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	114	28.043 detik	900/900	Optimal	
5x5	<i>Hard</i>	59	13.899 detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	406.778 detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	411.411 detik	2424/2430	Sub-Optimal	
10x10	<i>Easy</i>	1500	426.903 detik	3930/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	422.361 detik	4288/4320	Sub-Optimal	
15x15	<i>Easy</i>	1500	557.511 detik	9424/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	600.493 detik	17488/17820	Sub-Optimal	

5.2.12 Uji Parameter Keduabelas

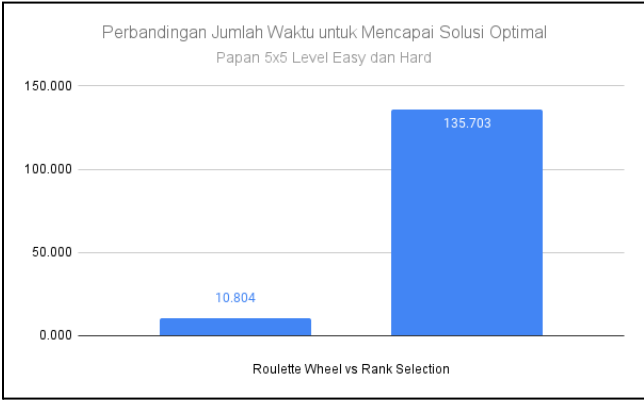
Ukuran Papan	Level	Generasi	Waktu eksekusi	Fitness Akhir	Optimal	Hasil Akhir
5x5	<i>Easy</i>	11	166.928 detik	900/900	Optimal	
5x5	<i>Hard</i>	12	197.491 detik	1080/1080	Optimal	
7x7	<i>Easy</i>	1500	6885.099 detik	1439/1440	Sub-Optimal	
7x7	<i>Hard</i>	1500	11340.227 detik	2427/2430	Sub-Optimal	
10x10	<i>Easy</i>	1500	10978.091 detik	3967/3960	Sub-Optimal	
10x10	<i>Hard</i>	1500	8712.919 detik	4299/4320	Sub-Optimal	
15x15	<i>Easy</i>	1500	10189.188 detik	9468/9540	Sub-Optimal	
20x20	<i>Easy</i>	1500	12109.909 detik	17611/17820	Sub-Optimal	

5.3 Perbandingan Eksekusi Algoritma Genetika

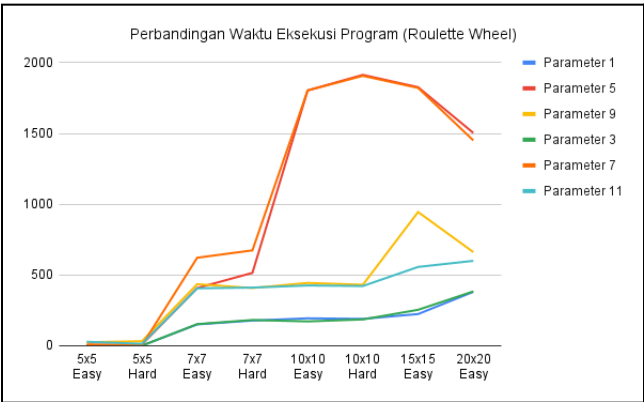
Berdasarkan data-data di atas, berikut beberapa ilustrasi perbandingan eksekusi algoritma genetika dalam berbagai parameter tertentu ke dalam sebuah grafik:



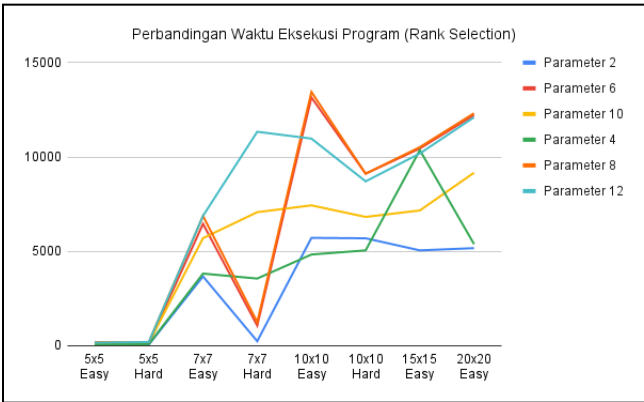
Gambar 5.3.1: Rata-rata generasi



Gambar 5.3.2: Rata-rata waktu eksekusi



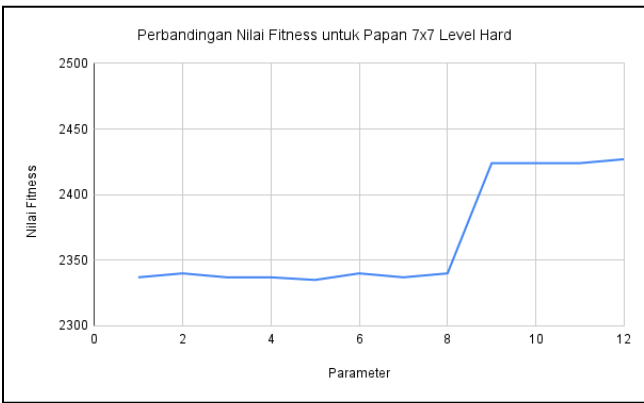
Gambar 5.3.3: Waktu eksekusi roulette wheel



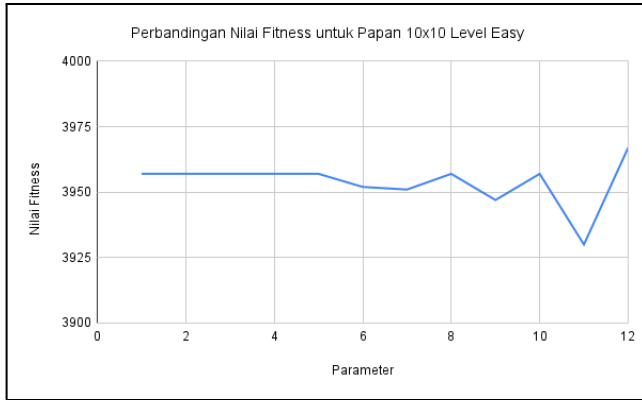
Gambar 5.3.4: Waktu eksekusi rank selection



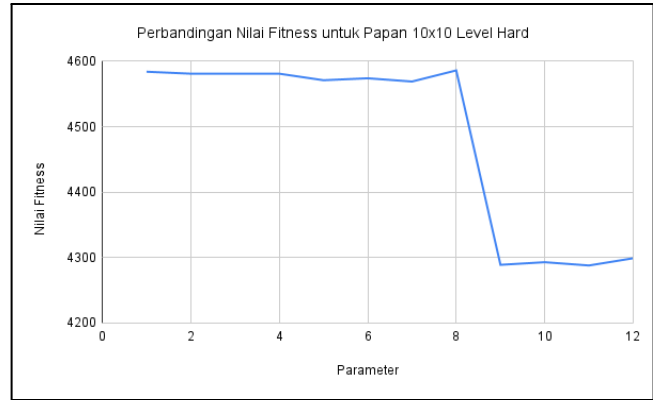
Gambar 5.3.5: Fitness yang didapatkan



Gambar 5.3.6: Fitness yang didapatkan



Gambar 5.3.7: Fitness yang didapatkan



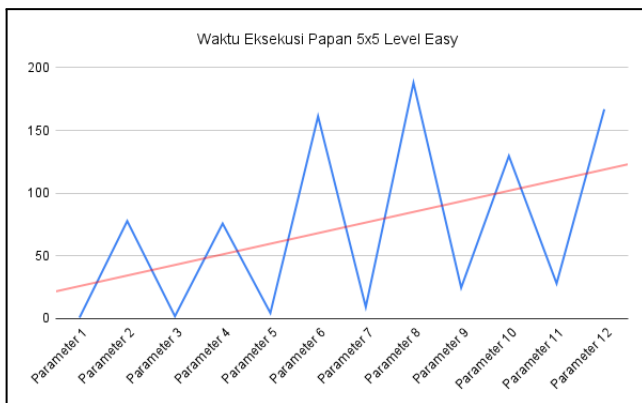
Gambar 5.3.8: Fitness yang didapatkan



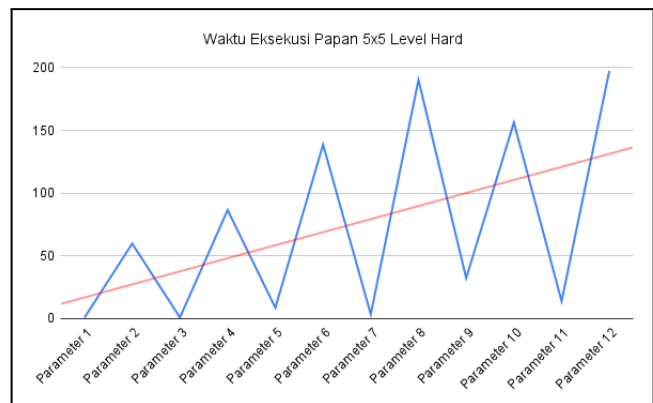
Gambar 5.3.9: Fitness yang didapatkan



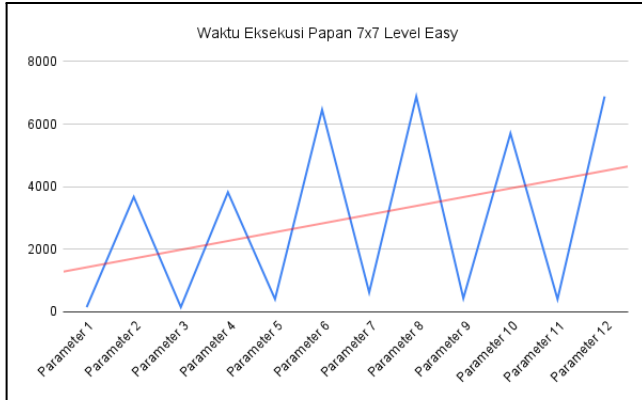
Gambar 5.3.10: Fitness yang didapatkan



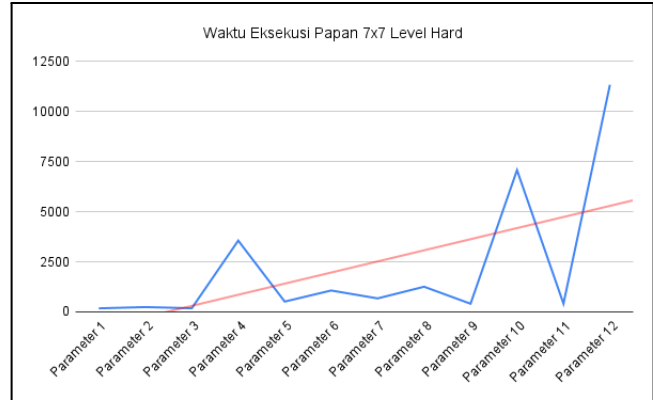
Gambar 5.3.11: Waktu Eksekusi



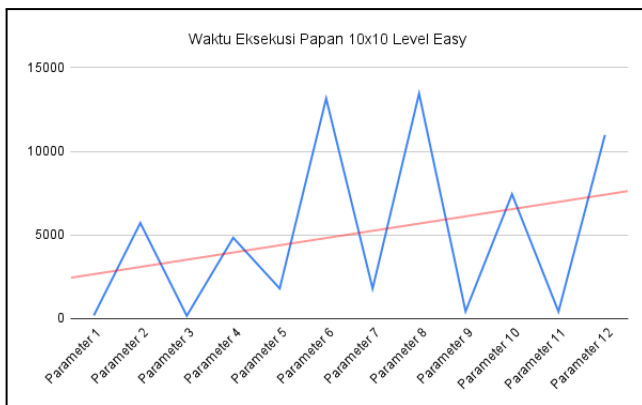
Gambar 5.3.12: Waktu Eksekusi



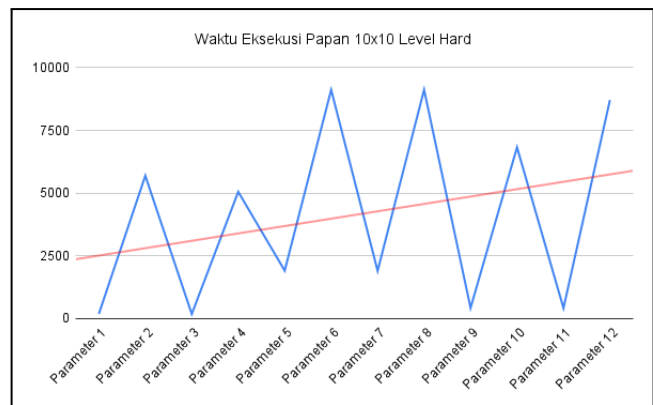
Gambar 5.3.13: Waktu Eksekusi



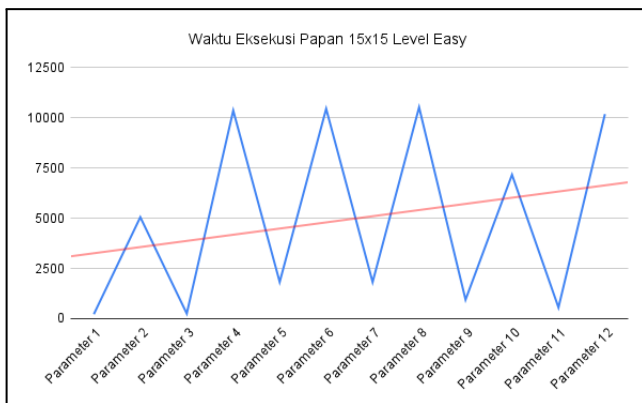
Gambar 5.3.14: Waktu Eksekusi



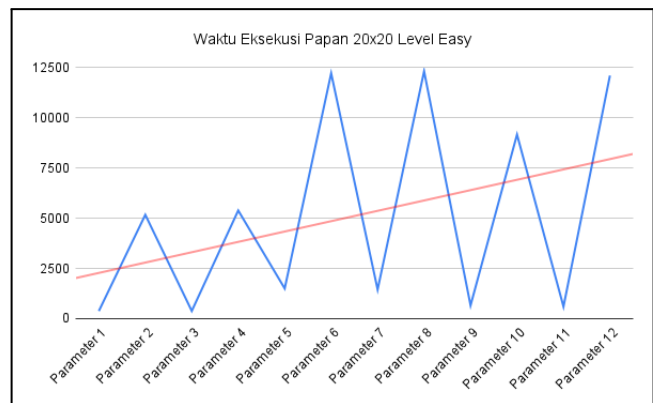
Gambar 5.3.15: Waktu Eksekusi



Gambar 5.3.16: Waktu Eksekusi



Gambar 5.3.17: Waktu Eksekusi



Gambar 5.3.18: Waktu Eksekusi

5.4 Eksperimen Menggunakan Seratus Buah Papan 5x5

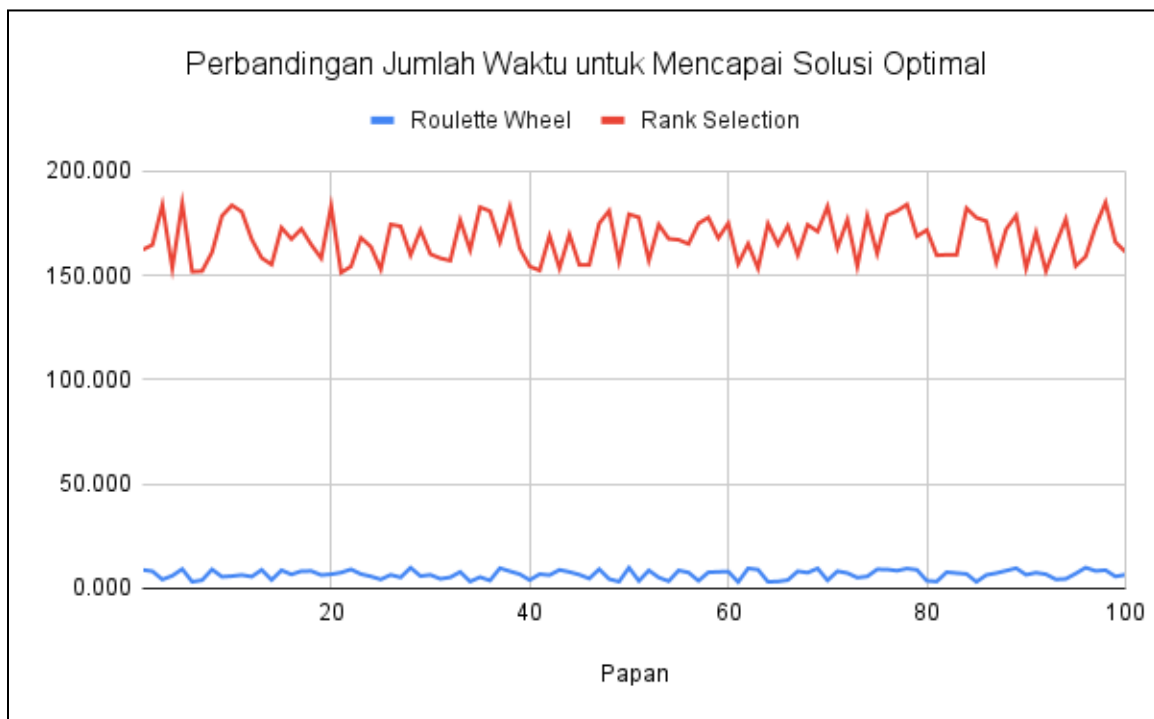
Eksperimen berikutnya adalah dengan mencoba melakukan pengujian pada 100 buah papan berukuran 5x5 pada level easy. Pengujian hanya dilakukan pada papan 5x5 level easy karena keterbatasan waktu dan sumber daya. Berikut parameter-parameter yang digunakan dalam melakukan pengujian

Parameter:

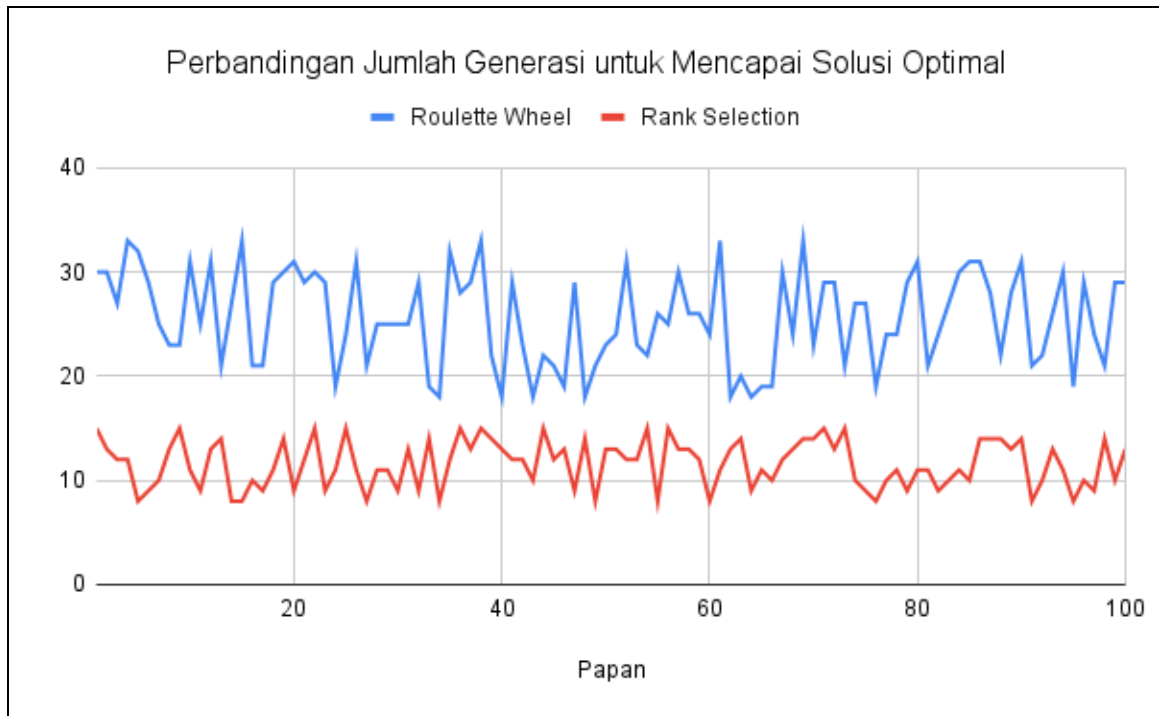
- Seed random : 123456
- Ukuran populasi : 10000
- Batas generasi : 1500
- Mutasi : 0,0001
- Elitism : 0.1%
- Crossover : 2 titik

Rata-rata	Roulette Wheel	Rank Selection
Waktu (detik)	6.667	167.518
Generasi	25.63	11.58

Tabel Perbandingan Metode Seleksi Roulette Wheel dan Rank Selection



Gambar 5.4.1 : Perbandingan Jumlah Waktu yang dibutuhkan untuk Mencapai Solusi



Gambar 5.4.2 : Perbandingan Jumlah Generasi yang dibutuhkan untuk Mencapai Solusi

Berdasarkan hasil eksperimen di atas, metode rank selection memiliki algoritma yang lebih efektif dalam segi generasi. Metode rank selection membutuhkan rata-rata sebesar 11.58 generasi, sedangkan metode roulette wheel membutuhkan rata-rata sebesar 25.63 generasi. Akan tetapi, metode roulette wheel memiliki algoritma yang lebih efektif dalam segi waktu. Metode rank selection membutuhkan rata-rata sebesar 167.518 detik, sedangkan metode roulette wheel hanya membutuhkan rata-rata sebesar 6.667 detik.

BAB 6

Kesimpulan

Berikut adalah hasil pencapaian tujuan dari penelitian ini:

1. Pembuatan fungsi *fitness* adalah pertama dengan menghitung banyaknya angka pada papan lalu dikali 9 (karena satu sel bisa mempengaruhi 9 kotak), lalu dikali 10 (karena satu sel bisa terdiri dari angka 0-9). Kemudian variable counter melakukan penelusuran pada papan dan jika sel pada papan bukan merupakan -1 (kosong), maka dilakukan pencarian terhadap sel sekelilingnya (karena sel tersebut berisi angka). Jika terdapat perbedaan dengan angka aslinya (misal angka pada sel adalah 6 namun hanya terdapat 3 kotak hitam), maka *fitness* yang benar tersebut dikurangi sebanyak selisih angka, kemudian dikali dengan angka yang terdapat pada tabel dan ditambah 1 (agar menghindari perkalian dengan 0). Hal ini dilakukan untuk bisa membedakan jika angka pada tabel yang salah adalah angka yang besar atau yang kecil.

2. Perancangan algoritma genetika menggunakan bahasa Java dibuat dengan cara memakai pendekatan *Object Oriented Programming* (OOP). Kemudian untuk mendukung nilai-nilai heuristik pada algoritma genetika, dibuatlah kelas-kelas yang terdiri dari kelas Individu, kelas Generasi, dan kelas Populasi. Selain itu, untuk menggabungkan ketiga buah kelas tersebut, dibuatlah kelas utama agar dapat menghubungkan ketiga kelas tersebut sekaligus sebagai pintu masuk utama untuk input dan output. Beberapa ide pembuatan *method-method* dan operator genetika diambil dari sumber yang berbeda, dan kemudian disesuaikan agar dapat menjalankan permainan *mosaic*. Agar dapat berjalan dengan baik, ide dari fungsi *fitness* sebelumnya juga diimplementasikan ke dalam bahasa Java, sehingga program dapat menyelesaikan permainan (mendekati optimal).

3.

- Seleksi

Berdasarkan hasil eksperimen pada Subbab 5.2, terdapat beberapa nilai menarik seperti waktu dan jumlah generasi yang dibutuhkan untuk mencapai optimal pada metode rank selection dan roulette wheel berbanding terbalik. Selain itu pada papan 7x7 level hard, *rank selection* berhasil mendapatkan solusi optimal sebanyak 3 buah, sedangkan pada *roulette wheel* tidak memiliki solusi optimal. Hal ini menandakan *rank selection* memiliki sistem yang lebih optimal dalam segi generasi, sedangkan *roulette wheel* efektif dalam segi waktu.

- Crossover

Berdasarkan hasil eksperimen pada Subbab 5.2 dan Subbab 5.3, metode *crossover* menggunakan antara 1 titik dan 2 titik tidak terlalu terdapat perbedaan yang signifikan. Pada beberapa kasus seperti Gambar 5.3.7, 5.3.8, dan Gambar 5.3.10 *crossover* dengan 2 titik memberikan hasil yang lebih baik dibandingkan crossover 1 titik.

- Elitism (waktu meningkat, eksploit / eksplor)

Berdasarkan grafik yang terdapat pada Gambar 5.3.11 hingga Gambar 5.3.18 terlihat semakin rendah elitism yang digunakan, maka waktu eksekusi akan menjadi semakin besar. Namun, jika nilai elitism semakin tinggi akan berpengaruh terhadap nilai fitness yang diberikan. Karena elitism mengambil individu-individu terbaik, maka semakin tinggi elitism akan memungkinkan terjadinya lokal maksimum akibat berfokus pada eksploitasi. Jika elitism semakin rendah maka algoritma genetika akan bekerja secara eksplorasi, sehingga tidak terjebak pada lokal maksimum, namun memerlukan waktu yang lebih lama.

Daftar Pustaka

- Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley.
- Russell, S., & Norvig, P. (2020). *Artificial intelligence: A modern approach* (4th ed.). Pearson Education.
- E Satriyanto (2009). *Algoritma genetika*. [Microsoft Word - Bab 7.doc \(pens.ac.id\)](#), Desember 2022.
- Sukaton, R. M. (2011). *Penggunaan Algoritma genetika dalam Masalah Jalur Terpendek pada Jaringan Data*. Skripsi FMIPA Universitas Indonesia: 1-67.
- Hermanto, B. (2022). *Implementasi Agen Self-Play di LUDII*. Skripsi FTIS Universitas Katolik Parahyangan: 1-207.
- Kaye, R. (2007). *Some Minesweeper Configurations*. School of Mathematics The University of Birmingham: 9-16 (22).
- A simple implementation of a genetic algorithm · GitHub. Diakses pada tanggal 5 Desember 2022, dari <https://gist.github.com/Vini2/bd22b36ddc69c5327097921f5118b709>
- Genetic Algorithms 15/30: Java Implementation of the Roulette Wheel Selection Method. Diakses pada tanggal 5 Desember 2022, dari [\(35\) Genetic Algorithms 15/30: Java Implementation of the Roulette Wheel Selection Method - YouTube](#)
- 2D Array in Java. Diakses pada tanggal 9 Desember 2022, dari <https://linuxhint.com/2d-array-java/>