BAB 3
ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

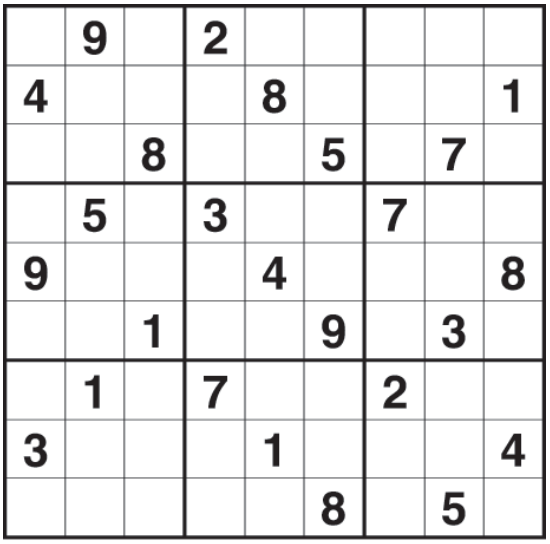
Analisis sistem merupakan penguraian dari suatu sistem informasi yang utuh ke dalam bagian-bagian komponennya yang dimaksudkan untuk mengidentifikasi dan mengevaluasi segala permasalahan dan hambatan yang terjadi serta kebutuhan yang diharapkan dapat menjadi acuan untuk diusulkannya perbaikan-perbaikan.

Untuk dapat melakukan analisis sistem maka dilakukan beberapa langkah yaitu sebagai berikut :

1. Analisis Masalah
2. Analisis Algoritma
3. Analisis Kebutuhan Non-Fungsional
4. Analisis Kebutuhan Fungsional

3.1.1 Analisis Masalah

Sudoku merupakan permainan teka-teki silang angka, dimana setiap baris dan kolom harus diisi dengan angka-angka dengan aturan tertentu. Sudoku adalah teka-teki penempatan angka didasarkan pada logika matematika. Sudoku terdiri dari kotak-kotak kecil yang membentuk baris dan kolom seperti halnya pada teka-teki silang (TTS). Pada TTS baris dan kolom diisi dengan huruf yang membentuk kata terbaca secara horizontal dan vertikal, sedangkan Sudoku baris dan kolom diisi dengan angka unik (tidak boleh ada angka yang sama dalam satu baris, atau dalam satu kolom, atau dalam satu region).



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

Gamber 3.1 Sudoku

Sudoku terdiri dari kisi persegi yang disebut grid berukuran 9x9 (9 baris x 9 kolom) yang dibagi menjadi sembilan bagian yang disebut subgrid berukuran 3x3. Teka-teki Sudoku menyediakan beberapa angka pada posisi acak sebagai patokan untuk menyelesaikan keseluruhan teka-teki tersebut. Kotak yang kosong harus diisi dengan angka unik dari 1 sampai 9 yang masing-masing hanya muncul sekali dalam setiap baris, kolom, dan persegi 3x3.

Permainan Sudoku ini jika diselesaikan secara manual akan memakan waktu yang cukup lama, maka untuk menyelesaikan permainan Sudoku dapat menggunakan algoritma *ant colony system* dan algoritma genetika. Pada bab 3 ini akan dilakukan analisis algoritma *ant colony system* dan algoritma genetika untuk mendapatkan solusi dalam penyelesaian masalah pada permainan Sudoku serta untuk mengetahui performasi kedua algoritma tersebut dalam menyelesaikan permainan Sudoku.

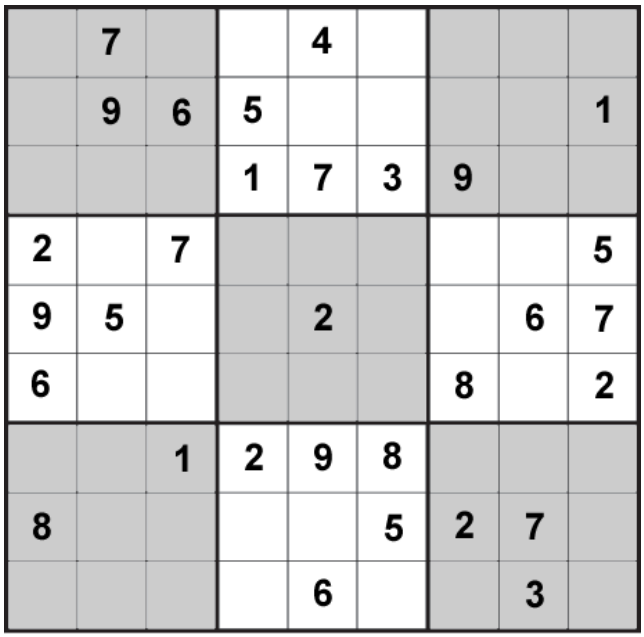
3.1.2 Analisis Algoritma

3.1.2.1 Analisis Algoritma Genetika

1. **Pembentukan Kromosom**

Kromosom merupakan representasi suatu solusi *puzzle* yang berisi angka yang diisikan pada kotak kosong. Panjang kromosom sama dengan jumlah kotak

kosong pada *puzzle*. Pada penelitian ini dibatasi dengan 50 kotak kosong pada *puzzle* maka panjang kromosomnya adalah 50. Representasi kromosom yang digunakan adalah deretan bilangan bulat. Nilai untuk masing-masing gen pada kromosom diambil secara acak dari angka 1 sampai 9.



Gambar 3.2 Sudoku dengan 50 kotak kosong

2. Inisialisasi

Proses inisialisasi dilakukan dengan cara memberikan nilai gen-gen awal dengan nilai acak dari angka 1 sampai 9. Setelah diketahui panjang kromosomnya adalah 50 dan ditentukan jumlah populasinya adalah 10.

Tabel 3.1 Inisialisasi

Kromosom	Gen
1	1,4,1,2,4,5,6,3,3,4,8,9,7,3,8,1,6,7,7,9,1,4,1,2,4,5,6,3,3,4,8,9,7,3,8,1,6,7,7,9,8,9,7,3,8,1,4,1,2,4
2	1,3,5,7,4,3,7,6,1,4,3,1,2,9,4,2,2,2,2,5,9,9,1,7,5,9,7,6,1,1,5,9,3,6,9,4,7,6,5,2,3,1,2,9,4,2,2,2,2,5
3	6,7,6,7,9,5,7,1,9,8,6,7,4,9,6,3,9,5,1,7,8,9,1,7,5,9,7,6,7,6,1,1,3,6,9,4,7,2,5,2,5,9,7,6,7,6,1,1,3,6
4	1,1,6,7,9,5,7,1,9,9,2,7,4,9,6,3,1,1,3,6,4,3,5,1,2,2,5,4,8,9,6,5,1,1,5,3,7,9,2,1,7,1,9,8,6,7,4,9,6,3

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

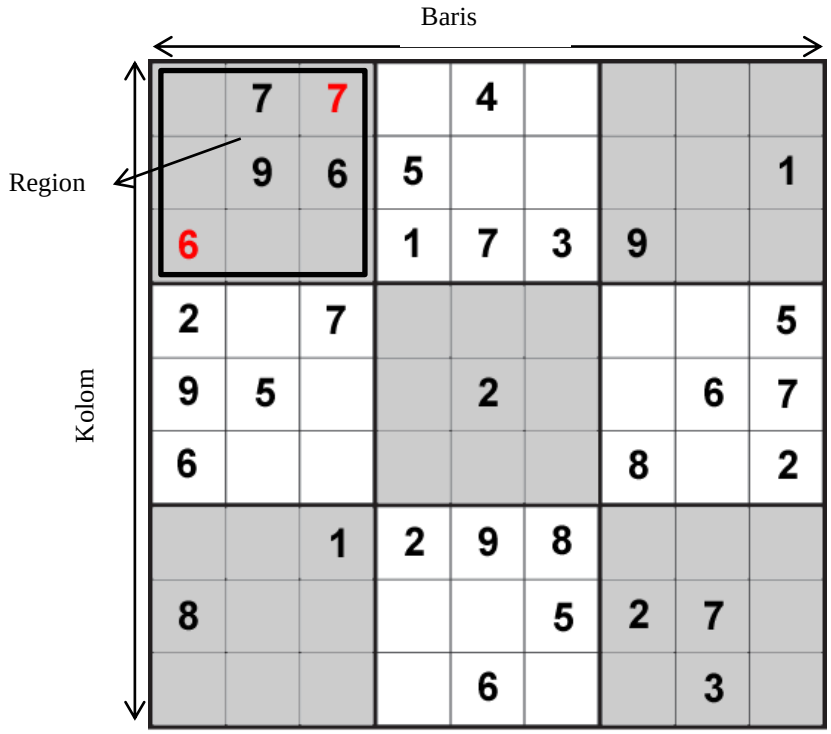
3. Nilai *Fitness*

Setelah kromosom tebentuk, langkah selanjutnya adalah mencari nilai *fitness*. Langkah pertama yaitu masukan gen-gen yang telah terbentuk kedalam kromosom, mulai dari kromosom 1 sampai kromosom 10.

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

Gambar 3.3 Sudoku dengan masukan gen-gen kromosom 1

Angka yang berwarna merah merupakan gen-gen yang *error*. Gen dapat dikatakan *error* apabila terdapat angka yang sama pada kolom yang sama, baris yang sama, dan dalam 1 region.



Gambar 3.4 Contoh gen dikatakan *error*

Pada Gambar 3.4 bisa dilihat angka 7 yang berwarna merah merupakan gen *error* karena pada baris yang sama terdapat angka yang sama dan dalam region juga terdapat angka yang sama. Angka 6 yang berwarna merah merupakan gen *error* karena pada kolom yang sama terdapat angka yang sama dan dalam region juga terdapat angka yang sama.

Pada Gambar 3.3, setelah dimasukan gen-gen dari kromosom 1 didapatkan 47 angka yang berwarna merah yaitu merupakan gen-gen *error*. Nilai *fitness* dapat dicari dengan panjang kromosom dikurangi dengan jumlah gen *error* kemudian dibagi dengan panjang kromosom.

Fitness = panjang kromosom – jumlah gen *error*

Kromosom[1] →*Fitness* = 50 – 47 = 3

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

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

Gambar 3.5 Sudoku dengan masukan gen-gen kromosom 2 dan 3

Kromosom[2] → *Fitness* = 50 – 46 = 4

Kromosom[3] → *Fitness* = 50 – 47 = 3

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

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

Gambar 3.6 Sudoku dengan masukan gen-gen kromosom 4 dan 5

Kromosom[4] → *Fitness* = 50 – 42 = 8

Kromosom[5] → *Fitness* = 50 – 48 = 2

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

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

Gambar 3.7 Sudoku dengan masukan gen-gen kromosom 6 dan 7

Kromosom[6] → *Fitness* = 50 – 45 = 5

Kromosom[7] → *Fitness* = 50 – 3 = 47

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

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

Gambar 3.8 Sudoku dengan masukan gen-gen kromosom 8 dan 9

Kromosom[8] → *Fitness* = 50 – 45 = 5

Kromosom[9] → *Fitness* = 50 – 46 = 4

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

Gambar 3.9 Sudoku dengan masukan gen-gen kromosom 10

Kromosom[10] → *Fitness* = 50 – 47 = 3

Solusi dengan nilai *fitness* sempurna (optimal) yaitu nilai *fitness* sama dengan panjang kromosom, dalam artian tidak memiliki gen *error*.

4. Seleksi

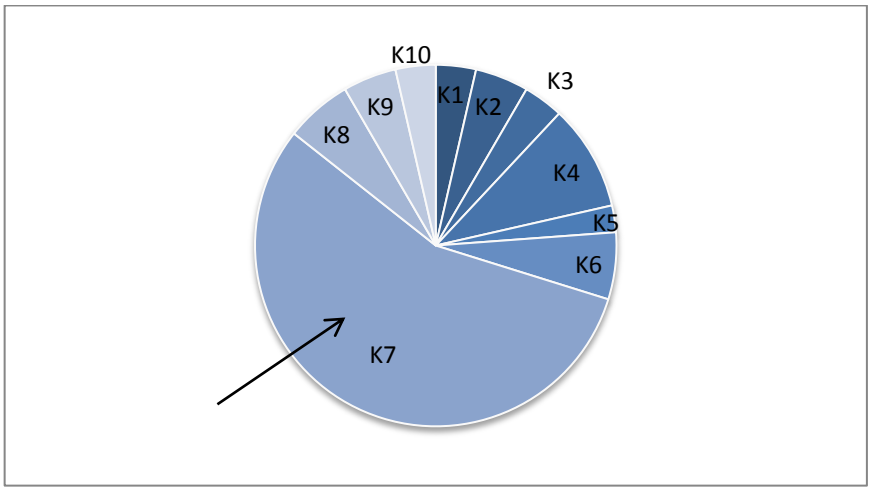
Metode seleksi yang digunakan adalah metode *Roulette-Wheel* yang membuat kromosom-kromosom dengan nilai *fitness* tinggi memiliki kemungkinan terpilih yang tinggi pula.

Probabilitas keterpilihan setiap kromosom ditentukan dengan cara membagi nilai *fitness* kromosom tersebut dengan total nilai *fitness*. Untuk itu, yang terlebih dahulu harus dihitung adalah total nilai *fitness*. Kemudian dilakukan pembagian masing-masing nilai *fitness* dengan total *fitness* tersebut. Nilai tersebut adalah probabilitas setiap kromosom yang berkisar antara nol dan satu.

Kemudian pada proses pemutaran *roulette-wheel*, teknisnya juga sederhana. Suatu bilangan acak dibangkitkan, dengan jangkauan nilai antara nol dan satu. Nilai acak tersebut kemudian dibandingkan dengan nilai probabilitas masing-masing kromosom. Pembandingan dilakukan dari kromosom yang terkecil. Kromosom yang memiliki nilai probabilitas yang lebih besar atau sama dengan nilai acak akan menjadi individu yang terpilih.

Tabel 3.2 Probabilitas Seleksi Untuk Setiap Kromosom

Kromosom	*Fitness*	Probabilitas	Nilai Probabilitas
K1	3	3/84	0.036
K2	4	4/84	0.048
K3	3	3/84	0.036
K4	8	8/84	0.095
K5	2	2/84	0.024
K6	5	5/84	0.060
K7	47	47/84	0.560
K8	5	5/84	0.060
K9	4	4/84	0.048
K10	3	3/84	0.036
Total	84	1	1



Gambar 3.10 Ilustrasi metode seleksi *roulette-wheel*

Misalkan bilangan acak yang dibangkitkan dengan jangkauan nilai dari 0 sampai 1 adalah 0,5. Kemudian nilai acak tersebut dibandingkan dengan nilai probabilitas masing-masing kromosom. Kromosom yang memiliki nilai probabilitas lebih besar atau sama dengan nilai acak yang dibangkitkan yaitu 0,5

akan terpilih. Pada Tabel 3.2 nilai probabilitas yang lebih besar dari 0.5 adalah kromosom 7 yang bernilai 0,56.

5. *Crossover*

Setelah melakukan proses seleksi, maka hasil tersebut akan digunakan dalam proses *crossover*. Metode *crossover* atau pindah silang yang digunakan adalah pindah silang dengan satu titik. Posisi *crossover* dipilih bilangan *random* dengan batasan dari 1 sampai N (N = panjang kromosom). Pertukaran gen kemudian dilakukan pada kedua kromosom mulai titik gen tertentu sampai akhir kromosom. Misalkan bilangan random yang dilipih adalah 32.

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

⇩

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

6. Mutasi

Mutasi dilakukan satu titik, yaitu dengan mengganti nilai gen suatu kromosom dengan suatu nilai yang dipilih secara acak dari titik-titik yang ada.

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

⇩

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

Langkah-langkah tersebut dilakukan terus hingga mendapatkan nilai *fitness* sempurna yaitu nilai *fitness* sama dengan panjang kromosom. Hasil akhri dari proses tersebut adalah sebagai berikut:

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

Gambar 3.11 Hasil akhir

Pseudocode algoritma genetika adalah sebagai berikut :

Tabel 3.3 *Pseudocode* Algoritma Genetika

```
Algoritma_Genetika
{ I.S. : Kondisi awal puzzle Sudoku (puzzle belum selesai) }
{ F.S. : Puzzle Sudoku selesai (hasil akhir didapatkan) }

Kamus:
    i,j,m,n,p,gen_error,iterasi,total_fitness,fitness,bilCross,bilCross2,bilMut : integer
    bilSel,tmpSel : real
    array_K = array [1..10,1..50] of integer
    K : array_K   //kromosom
    array_S = array [1..9,1..9] of integer
    S : array_S   //kotak pada sudoku 9x9
    array_tmpS = array [1..9,1..9] of integer
    tmpS : array_tmpS   //penyimpanan sementara
    array_fitnessK = array [1..10] of integer
    fitnessK : array_fitnessS   //nilai fitness
    array_tmpSeleksi = array [1..9,1..9] of integer
    tmpSeleksi : array_tmpSeleksi   //nilai fitness
    array_tmpK = array [1..9,1..9] of integer
    tmpK : array_tmpK   //penyimpanan sementara crossover
Algoritma:
    for i ← 1 to 10 do
        for j ← 1 to 50 do
1           K[i,j] ← random(1..9)   //bilangan random dari 1 hingga 9
        endfor
```

```
        endfor
        for m ← 1 to 9 do
            for n ← 1 to 9 do
2               read(S[m,n])
3               tmpS[m,n] ← S[m,n]
            endfor
        endfor
        tmpSel ← 0
        fitness ← 0
        iterasi ← 0
        repeat
            total_fitness ← 0
            for p ← 1 to 10 do
                i ← 1
                j ← 1
                gen_error ← 0
                for m ← 1 to 9 do
                    for n ← 1 to 9 do
4                       if tmpS[m,n] = 0 then
5                           tmpS[m,n] ← K[i,j]
                            j ← j + 1
                            //membandingkan dengan baris, kolom, dan region
6                           if m = 1 then
                                if n = 1 then
                                    if tmpS[m,n] = tmpS[1,2] or
                                    tmpS[m,n] = tmpS[1,3] or
                                    tmpS[m,n] = tmpS[1,4] or
                                    tmpS[m,n] = tmpS[1,5] or
                                    tmpS[m,n] = tmpS[1,6] or
                                    tmpS[m,n] = tmpS[1,7] or
                                    tmpS[m,n] = tmpS[1,8] or
                                    tmpS[m,n] = tmpS[1,9] or
                                    tmpS[m,n] = tmpS[2,1] or
                                    tmpS[m,n] = tmpS[3,1] or
                                    tmpS[m,n] = tmpS[4,1] or
                                    tmpS[m,n] = tmpS[5,1] or
                                    tmpS[m,n] = tmpS[6,1] or
                                    tmpS[m,n] = tmpS[7,1] or
                                    tmpS[m,n] = tmpS[8,1] or
                                    tmpS[m,n] = tmpS[9,1] or
                                    tmpS[m,n] = tmpS[2,2] or
                                    tmpS[m,n] = tmpS[2,3] or
                                    tmpS[m,n] = tmpS[3,2] or
                                    tmpS[m,n] = tmpS[3,3] then
7                                       gen_error ← gen_error + 1
                                    endif
                                endif
                                .
                                .
                                //bandingkan dari n=2 hingga n=9
                                .
                                .
```

```
                                    endif
                                    .
                                    .
                                    //bandingkan dari m=2 hingga m=9
                                    .
                                    .
                               endif
                          endfor
                     endfor
8                    fitnessK[p] ← 50 – gen_error
                     if fitnessK[p] = 50 then
                          fitness ← 50
                     endif
9                    total_fitness ← total_fitness + fitnessK[p]
                     i ← i + 1
               endfor
10             if fitness != 50 then
                     //seleksi
11                   bilSel ← random(0..1) //nilai yang akan dibandingkan dengan nilai probabilitas
                     for p ← 1 to 10 do
12                        if (fitnessK[p]/total_fitness) >= bilSel then
                               if (fitnessK[p]/total_fitness) > tmpSel then
                                    for i ← 1 to 9 do
                                         for j ← 1 to 9 do
13                                            tmpSeleksi[i,j] ← tmpS[i,j]
                                         endfor
                                    endfor
                               endif
                          endif
                     endfor
                     //crossover
14                   bilCross ← random(1..50) //posisi awal crossover
                     for i ← 1 to 10 do
15                        bilCross2 ← billCross
                          for j ← to 50 do
16                             tmpK[i,j] ← K[i,j]
                               if j = bilCross2 then
                                    tmpK[i,j] ← K[i+1,j]
                                    if i = 10 then
17                                       tmpK[i,j] ← K[1,j]
                                    endif
                               endif
                          endfor
                     endfor
                     //mutasi
18                   bilMut ← random(1..50) //posisi mutasi
                     for i ← 1 to 10 do
                          for j ← to 50 do
                               if j = bilMut then
19                                  tmpK[i,j] ← random(1..9)
                               endif
```

```
                                K[i,j] ← tmpK[i,j]
                            endfor
                        endfor
                    endif
                    iterasi = iterasi + 1
                until (fitness = 50 or iterasi = 1000)
                //hasil akhir
                for m ← 1 to 9 do
                    for n ← 1 to 9 do
                        if fitness = 50 then
                            S[m,n] ← tmpS[m,n]
                        else
                            S[m,n] ← tmpSeleksi[m,n]
                        endif
                    endfor
                endfor
```

3.1.2.2 Analisis Algoritma *Ant Colony Optimization (ACO) System*

Mengingat keberhasilan *ACO system* saat memecahkan *Travelling Salesman Problem* (TSP), maka masuk akal jika *ACO system* digunakan untuk memecahkan *puzzle* Sudoku. Pertama merepresentasikan setiap kotak Sudoku dalam 9x9 grid, dapat menggunakan koordinat (x,y) seperti pada kota-kota di TSP dan kemudian memperluas grid dengan menambahkan sumbu z sehingga menjadi (x,y,z). Sumbu z dapat direpresentasikan sebagai value atau nilai (z,y,value). Dengan demikian akan menghasilkan 729 node "*piece set*". Menggunakan permutasi dengan panjang 81 dari *piece set* yang dapat direpresentasikan menjari *puzzle* Sudoku lengkap. Kemudian dapat diterapkan aturan sederhana Sudoku untuk menentukan bahwa hanya solusi valid yang dipilih.

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

Gambar 3.12 Sudoku dengan titik (x,y)

Pada Gambar 3.16 merupakan titik-titik (x,y) mulai dari (1,1) hingga (9,9) yaitu memiliki 81 titik. Pada setiap titik dapat dimasukan nilai dengan memiliki 9 kemungkinan yaitu angka 1 sampa 9. Sudoku memiliki 81 titik dan setiap titik memiliki 9 kemungkinan, maka terdapat 729 kemungkinan.

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

Gambar 3.13 Sudoku

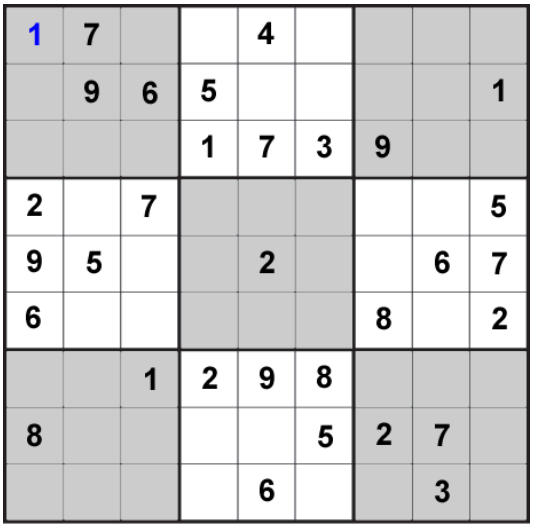
Setiap individu semut mengoperasikan *tabu search*, dapat menempuh banyak *node* tanpa melanggar aturan permainan. *Tabu search* sendiri merupakan sebuah metode optimasi yang berbasis pada *local search*. Setelah *tabu list* berisi semua *node*, semut akan menghitung jumlah feromon yang dibuang sepanjang jalan dan mendistribusikannya. Jumlah feromon disimpan pada setiap sisi jalur untuk memberikan petunjuk. Gen-gen pada setiap node akan ditemukan dengan membandingkan angka yang ada pada kolom, baris, dan region yang sama.

Tabel 3.4 Gen-gen pada setiap node

Node (x,y)	Gen (z atau value)	Node (x,y)	Gen (z atau value)
1,1	1	5,6	1,4
1,3	2,3,5,8	5,7	1,3,4
1,4	6,8,9	6,2	1,3,4
1,6	2,6,9	6,3	3,4
1,7	3,5,6	6,4	3,4,7,9
1,8	2,5,8	6,5	5
1,9	3	6,6	1,4,7,9
2,1	3,4	6,8	1,4,9
2,5	8	7,1	7
2,6	2	7,2	3,4,6
2,7	7	7,7	4,5,6
2,8	2,4,8	7,8	4,5
3,1	4,5	7,9	4,6
3,2	2,4,8	8,2	3,4,6
3,3	2,4,5,8	8,3	3,4,9
3,8	2,4,5,8	8,4	3,4
3,9	6	8,5	1
4,2	1,3,4,8	8,9	4,6,9

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

Node yang memiliki gen dengan jumlah 1 akan dipilih, sehingga hasilnya adalah sebagai berikut:



Gambar 3.14 Sudoku dengan *node* yang terpilih (1,1)

Setelah dipilih 1 *node* maka gen-gen pada setiap *node* akan berubah dan berikut merupakan perubahan gen-gen pada setiap *node*:

Tabel 3.5 Gen-gen pada setiap *node* setelah dipilih 1 *node*

Node (x,y)	Gen (z atau value)	Node (x,y)	Gen (z atau value)
		5,6	1,4
1,3	2,3,5,8	5,7	1,3,4
1,4	6,8,9	6,2	1,3,4

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

Seteleh itu, dilakukan lagi proses yang sama seperti sebelumnya, sehingga hasilnya adalah sebagai berikut :

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

Gambar 3.15 Sudoku dengan *node* yang terpilih (1,9)

Tabel 3.6 Gen-gen pada setiap *node* setelah dipilih 1 *node*

Node (x,y)	Gen (z atau value)	Node (x,y)	Gen (z atau value)
		5,6	1,4
1,3	2,5,8	5,7	1,3,4
1,4	6,8,9	6,2	1,3,4
1,6	2,6,9	6,3	3,4
1,7	5,6	6,4	3,4,7,9
1,8	2,5,8	6,5	5
		6,6	1,4,7,9
2,1	3	6,8	1,4,9
2,5	8	7,1	7
2,6	2	7,2	3,4,6
2,7	7	7,7	4,5,6
2,8	2,4,8	7,8	4,5
3,1	4,5	7,9	4,6
3,2	2,4,8	8,2	3,4,6
3,3	2,4,5,8	8,3	3,4,9

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

Proses tersebut akan dilakukan terus menerus hingga puzzle terpenuhi. Setelah melalui tahap-tahap tersebut, maka hasilnya adalah sebagai berikut :

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

Gambar 3.16 Hasil akhir

Pseudocode algoritma *ant colony optimization system* adalah sebagai berikut:

Tabel 3.7 *Pseudocode ACO System*

```
Algoritma_ACO_System
{ I.S. : Kondisi awal puzzle Sudoku (puzzle belum selesai) }
{ F.S. : Puzzle Sudoku selesai (hasil akhir didapatkan) }

Kamus:
    m,n,x,y,z : integer
    array_B = array [1..9,1..9,1..9] of integer
    B : array_B   //bilangan untuk inisialisasi
    array_S = array [1..9,1..9] of integer
    S : array_S   //kotak pada sudoku 9x9
    array_tmpS = array [1..9,1..9] of integer
    tmpS : array_tmpS   //penyimpanan sementara




Algoritma:
    for x ← 1 to 9 do
        for y ← 1 to 9 do
1           read(S[x,y])
2           tmpS[x,y] ← S[x,y]
            for z ←  1 to 9 do
3               B[x,y,z] ← z
            endfor
        endfor
    endfor
4   m ← 0
5   while (m<50) do
        for x ← 1 to 9 do
            for y ← 1 to 9 do
6               if tmpS[x,y] = 0 then
                    //membandingkan dengan baris, kolom, dan region
                    if x = 1 then
                        if y = 1 then
                            for z ← 1 to 9 do
                            if tmpS[1,2] = z or tmpS[1,3] = z or tmpS[1,4] = z or
                            tmpS[1,5] = z or tmpS[1,6] = z or tmpS[1,7] = z or
                            tmpS[1,8] = z or tmpS[1,9] = z or tmpS[2,1] = z or
                            tmpS[3,1] = z or tmpS[4,1] = z or tmpS[5,1] = z or
                            tmpS[6,1] = z or tmpS[7,1] = z or tmpS[8,1] = z or
                            tmpS[9,1] = z or tmpS[2,2] = z or tmpS[2,3] = z or
                            tmpS[3,2] = z or tmpS[3,3] = z then
7                               B[x,y,z] ← 0
                            endif
                            endfor
                        endif
                        .
```

```
                                .
                                //bandingkan dari y=2 hingga y=9
                                .
                                .
                            endif
                            .
                            .
                            //bandingkan dari x=2 hingga x=9
                            .
                            .
                        endif
                    endfor
                endfor
8               n ← 0
                for x ← 1 to 9 do
                    for y ← 1 to 9 do
                        for z ← 1 to 9 do
9                           if B[x,y,z] = 0 then
10                              n ← n + 1
                            endif
                        endfor

11                      if n.count = 8 then
                            //nilai 0 ada delapan, nilai selain 0 ada 1. (0,0,0,0,0,4,0,0,0)
                            for z ← 1 to 9 do
12                              if B[x,y,z] != 0 then
13                                  tmpS[x,y] ← z  //simpan nilai sementara
14                                  m ← m + 1
                                endif
                            endfor
                        endif
                    endfor
                endfor
15          endwhile
            //hasil akhir
            for x ← 1 to 9 do
                for y ← 1 to 9 do
16                  S[x,y] ← tmpS[x,y]
                endfor
            endfor
```

3.1.3 Analisis Kebutuhan Non-Fungsional

Analisis non fungsional merupakan analisis yang dibutuhkan untuk menentukan spesifikasi kebutuhan sistem. Spesifikasi ini juga meliputi elemen atau komponen-komponen apa saja yang akan dibutuhkan sistem yang akan dibangun, sampai sistem tersebut diimplementasikan. Adapun kebutuhan non

fungsionalnya yaitu meliputi analisis kebutuhan perangkat keras, analisis kebutuhan perangkat lunak, dan analisis pengguna.

3.1.3.1 Analisis Kebutuhan Perangkat Keras

Perangkat keras diperlukan untuk menjalankan suatu aplikasi guna mendukung proses kerja dari suatu sistem. Analisis perangkat keras digunakan untuk mengetahui mengenai spesifikasi perangkat keras dalam membangun dan menerapkan aplikasi, terdapat dua spesifikasi yang akan dijelaskan, yaitu spesifikasi perangkat keras pengembang dan spesifikasi perangkat keras pengguna.

Berikut ini merupakan spesifikasi perangkat keras yang digunakan dalam membangun aplikasi ini, yaitu:

1. *Processor* 2.1GHz
2. VGA 1GB
3. RAM 4GB
4. *Harddisk* 320GB
5. Resolusi Layar 1366 x 768
6. *Keyboard* dan *Mouse* standar

Berikut ini merupakan spesifikasi perangkat keras pengguna atau spesifikasi minimum untuk menjalankan aplikasi ini, yaitu:

1. *Processor* 1GHz
2. VGA 256MB
3. RAM 512GB
4. *Harddisk* 100MB

3.1.3.2 Analisis Kebutuhan Perangkat Lunak

Analisis perangkat lunak digunakan untuk mengetahui mengenai spesifikasi perangkat lunak untuk membangun dan menerapkan aplikasi, terdapat dua spesifikasi yang akan dijelaskan, yaitu spesifikasi perangkat lunak pengembang dan spesifikasi perangkat lunak pengguna.

Berikut ini merupakan spesifikasi perangkat lunak yang digunakan dalam membangun aplikasi ini, yaitu:

1. Sistem Operasi Microsoft Window 7
2. Microsoft Visual Studio

Berikut ini merupakan spesifikasi perangkat lunak yang digunakan dalam menjalankan aplikasi ini, yaitu:

1. Sistem Operasi Microsoft Window XP

3.1.3.3 Analisis Kebutuhan Pengguna

Selain dibutuhkannya perangkat lunak dan perangkat keras, *user* atau penguna adalah bagian yang sangat penting untuk dapat menerapkan aplikasi ini. Analisis pengguna diperlukan untuk dapat mengetahui sasaran calon pengguna yang dapat menggunakan aplikasi. Pada penelitian ini pembangunan aplikasi ditujukan untuk pengujian implementasi algoritma sehingga pengguna disini disarankan memiliki karakteristik sebagai berikut:

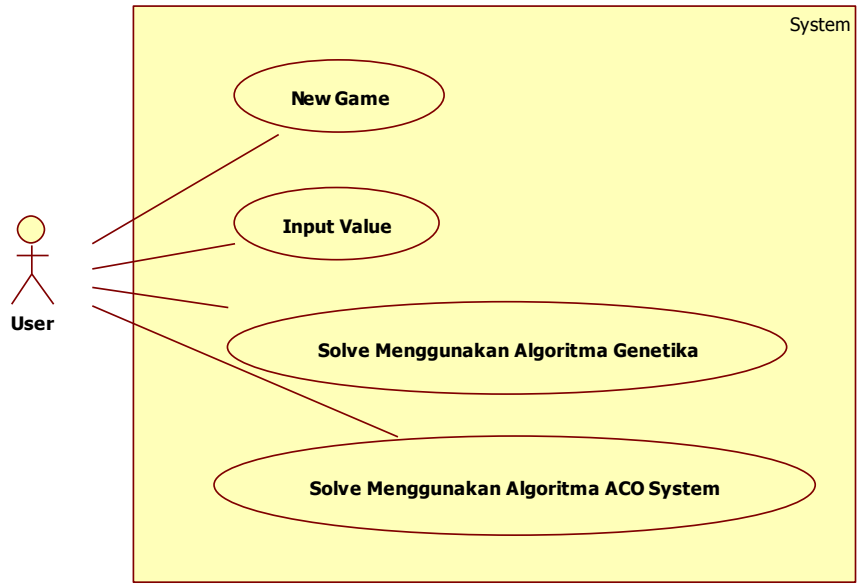
1. Bisa mengoperasikan komputer dengan sistem operasi *windows.*
2. Mengerti mengenai konsep algoritma Ant Colony Optimization (ACO) System dan algoritma genetika.

3.1.4 Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional menyangkut analisis sistem yang dilakukan yaitu dengan menggunakan UML. Adapun tahapan analisis sistem menggunakan UML adalah melalui *use case diagram, activity diagram, sequence diagram,* dan *class diagram.*

3.1.4.1 *Use Case Diagram*

Use case diagram merupakan konstruksi untuk mendeskripsikan hubungan yang terjadi antara aktor dengan aktivitas yang terdapat pada sistem. Berikut ini merupakan *use case diagram* pada permainan Sudoku yang akan diterapkan algoritma *ant colony optimization system* dan algoritma genetika.



Gambar 3.17 *Use Case Diagram*

3.1.4.1.1 Definisi Aktor

Definisi aktor berfungsi untuk menjelaskan aktor yang terdapat pada *use case* diagram. Definisi aktor dapat dilihat pada tabel berikut:

Tabel 3.8 Definisi Aktor

Aktor	Deskripsi
User	Pengguna sistem

3.1.4.1.2 Definisi *Use Case*

Definisi use case menjelaskan tentang fungsi *use case* yang terdapat pada *use case diagram*. Definisi *use case* dapa dilihat pada tabel berikut:

Tabel 3.9 Definisi *Use Case*

No	*Use Case*	Deskripsi
1	*New Game*	Fungsional untuk memulai permainan
2	*Input Value*	Fungsional untuk melakukan *input* nilai pada *cell* atau kotak kosong dalam permainan Sudoku
3	Algoritma Genetika	Fungsional untuk pencarian solusi atau untuk menyelesaikan permainan Sudoku menggunakan

		algoritma genetika
4	Algoritma *Ant Colony Optimization System*	Fungsional untuk pencarian solusi atau untuk menyelesaikan permainan Sudoku menggunakan algoritma *ant colony optimization system*

3.1.4.1.3 Skenario *Use Case*

Skenario pada diagram *use case* berisi tentang alur kejadian (*flow of event*) *use case* utama dengan menjelaskan urutan interaksi antara aktor dan *use case* tersebut dari awal sampai akhir.

1. **Skenario *Use Case New Game***

   Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu *user* dengan *use case New Game* yang dijelaskan pada tabel berikut:

Tabel 3.10 Skenario *Use Case New Game*

Identifikasi		
Nomor	1	
Nama	*New Game*	
Tujuan	Memulai permainan	
Deskripsi	Fungsional untuk memulai permainan	
Aktor	*User*	
Skenario Utama		
Kondisi Awal	*User* berada di tampilan awal permainan Sudoku	
Aksi Aktor		**Reaksi Sistem**
1. Memilih menu *New Game*		
		2. Melakukan proses pembangkitan puzzle 3. Menghasilkan *puzzle* baru 4. Menampilkan kondisi awal permainan
Kondisi Akhir	Aplikasi menampilkan kondisi awal permainan	

2. Skenario *Use Case Input Value*

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu *user* dengan *use case Input Value* yang dijelaskan pada tabel berikut:

Tabel 3.11 Skenario *Use Case Input Value*

Identifikasi		
Nomor	2	
Nama	*Input Value*	
Tujuan	Mengisi nilai pada kotak kosong	
Deskripsi	Fungsional untuk melakukan *input* nilai pada kotak kosong dalam permainan Sudoku	
Aktor	*User*	
Skenario Utama		
Kondisi Awal	*User* berada di tampilan awal permainan Sudoku	
Aksi Aktor		**Reaksi Sistem**
1. Melakukan input nilai pada kotak kosong		
		2. Menampilkan nilai yang telah dimasukan oleh *user*
Kondisi Akhir	Aplikasi menampilkan nilai yang telah dimasukan oleh *user*	

3. Skenario *Use Case* Algoritma Genetika

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu *user* dengan *use case* Algoritma Genetika yang dijelaskan pada tabel berikut:

Tabel 3.12 Skenario *Use Case* Algoritma Genetika

Identifikasi	
Nomor	3
Nama	Algoritma Genetika
Tujuan	Mencari solusi menggunakan algoritma genetika
Deskripsi	Fungsional untuk mencari solusi permainan Sudoku dengan menggunakan algoritma genetika

Aktor	*User*
Skenario Utama	
Kondisi Awal	*User* berada di tampilan awal permainan Sudoku
Aksi Aktor	**Reaksi Sistem**
1. Memilih tombol Genetika	
	2. Input bilangan 50 bilangan acak dari 1 hingga 9 3. Inisisalisasi kromosom 4. Evaluasi nilai fitness 5. Melakukan seleksi (*roulette wheel*) 6. Melakukan *crossover* 7. Melakukan mutasi 8. Menampilkan hasil
Kondisi Akhir	Aplikasi menampilkan hasil (*puzzle* Sudoku diselesaikan)

4. Skenario *Use Case* Algoritma *Ant Colony Optimization System*

Skenario *use case* ini menjelaskan interaksi antara aktor, yaitu *user* dengan *use case* Algoritma *Ant Colony Optimization System* yang dijelaskan pada tabel berikut:

Tabel 3.13 Skenario *Use Case* Algoritma *Ant Colony Optimization System*

Identifikasi	
Nomor	4
Nama	Algoritma *Ant Colony Optimization System*
Tujuan	Mencari solusi menggunakan algoritma *ant colony optimization system*
Deskripsi	Fungsional untuk mencari solusi permainan Sudoku dengan menggunakan algoritma *ant colony optimization system*
Aktor	*User*
Skenario Utama	
Kondisi Awal	*User* berada di tampilan awal permainan Sudoku
Aksi Aktor	**Reaksi Sistem**
1. Memilih tombol ACO *System*	

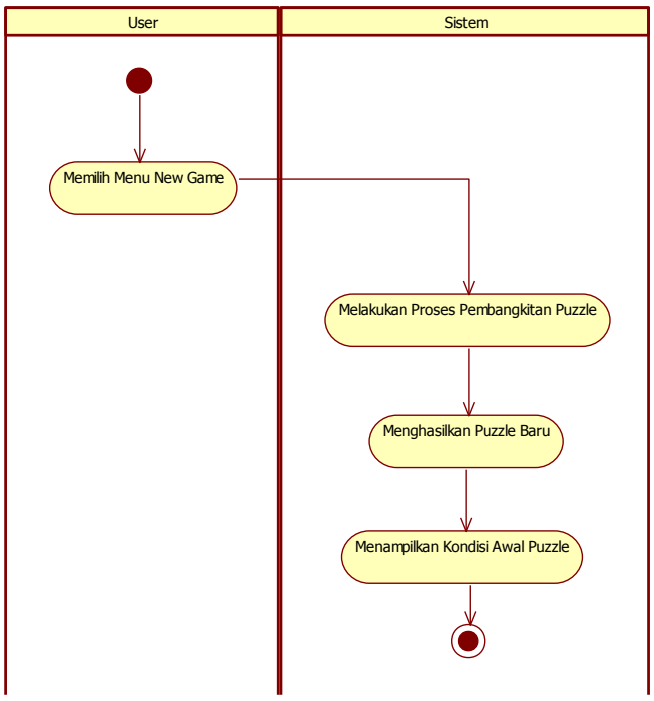
	2. Input bilangan dari 1 hingga 9 3. Menentukan posisi dan nilai awal setiap *node* 4. Membandingkan nilai dari *node* dengan baris, kolom, dan region 5. Simpan nilai 6. Menampilkan hasil
Kondisi Akhir	Aplikasi menampilkan hasil (*puzzle* Sudoku diselesaikan)

3.1.4.2 *Activity Diagram*

Activity diagram berisi penjelasan lebih lengkap mengenai urutan suatu proses yang terjadi antara pengguna dan sistem. Aktivitas pertama selalu dimulai dari pengguna, kemudian akan mendapat respon dari sistem yang nantinya akan menampilkan akhir dari proses tersebut.

1. ***Activity Diagram New Game***

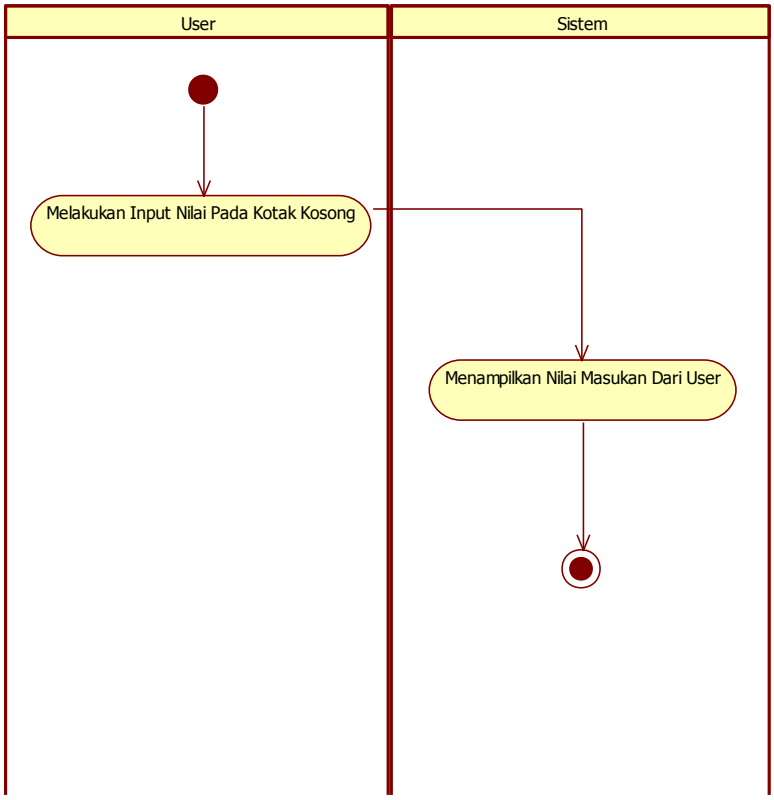
   Berikut ini merupakan diagram yang menunjukan alur aksi pada aktivitas *New Game* yang dapat dilihat pada gambar di bawah ini.



Gambar 3.18 *Activity Diagram New Game*

2. Activity Diagram Input Value

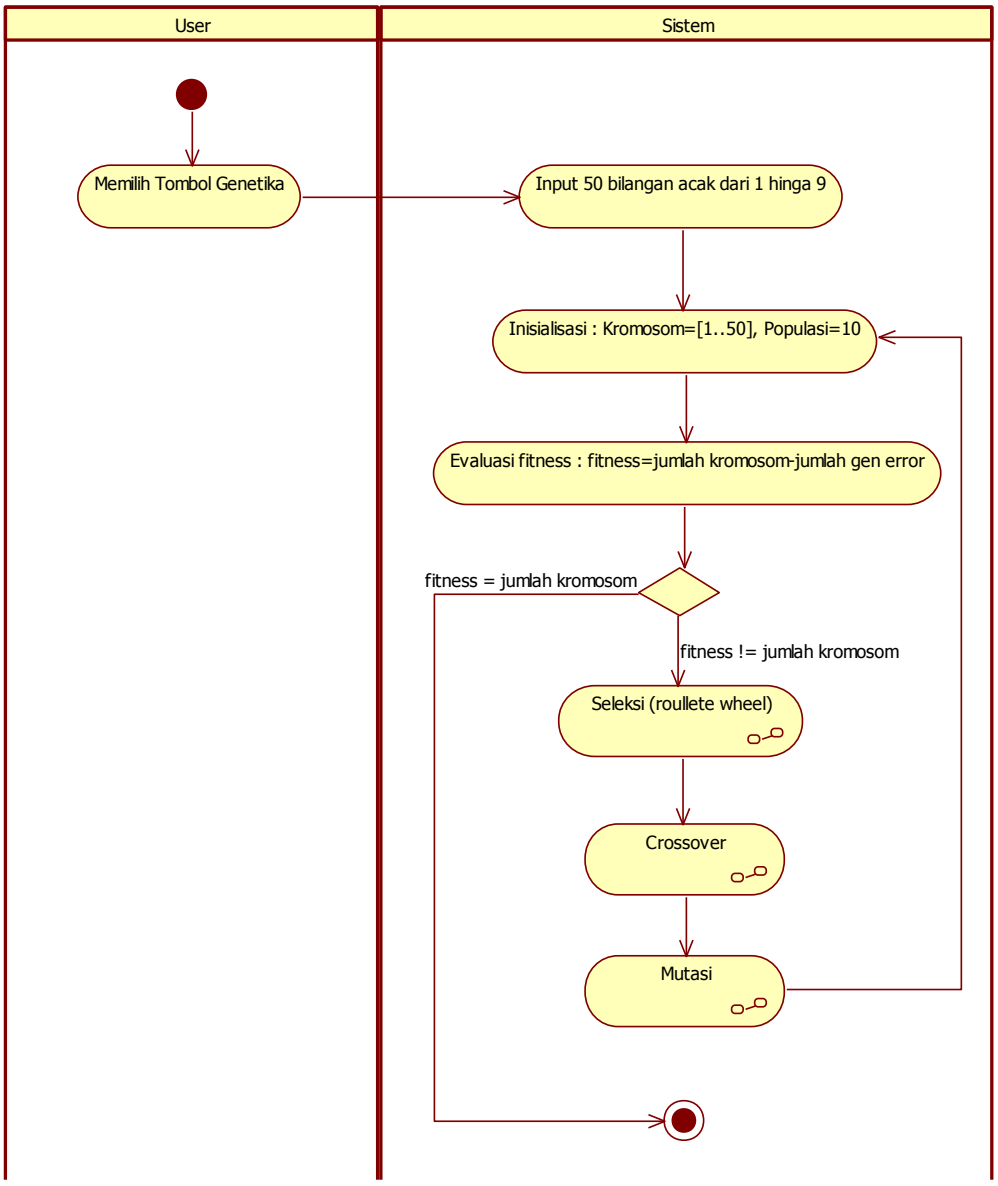
Berikut ini merupakan diagram yang menunjukan alur aksi pada aktivitas *Input Value* yang dapat dilihat pada gambar di bawah ini.



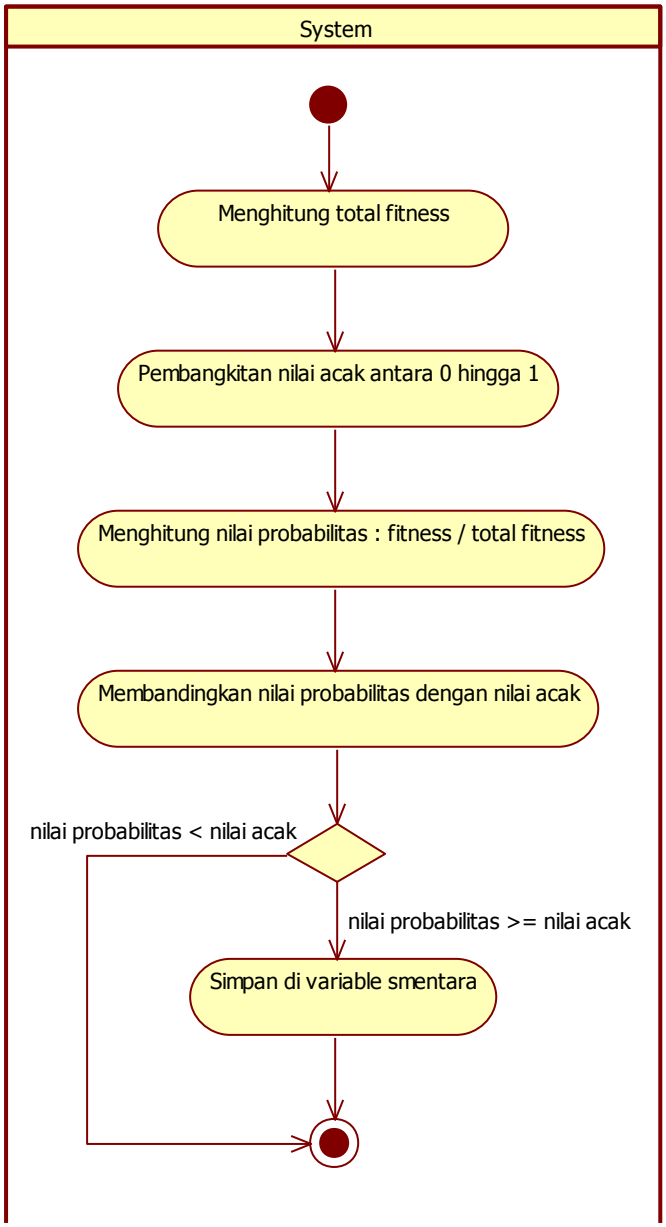
Gambar 3.19 *Activity Diagram Input Value*

3. *Activity Diagram* Algoritma Genetika

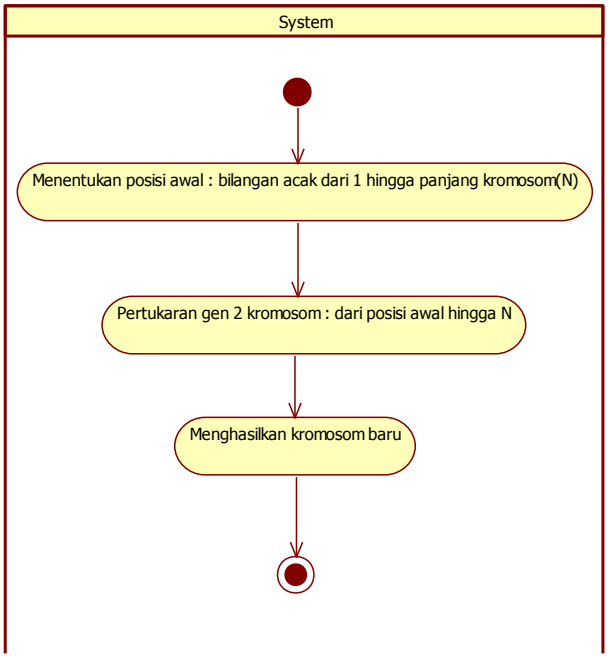
Berikut ini merupakan diagram yang menunjukan alur aksi pada aktivitas Algoritma Genetika yang dapat dilihat pada gambar di bawah ini.



Gambar 3.20 *Activity Diagram* Algoritma Genetika



Gambar 3.21 *Subactivity* Seleksi



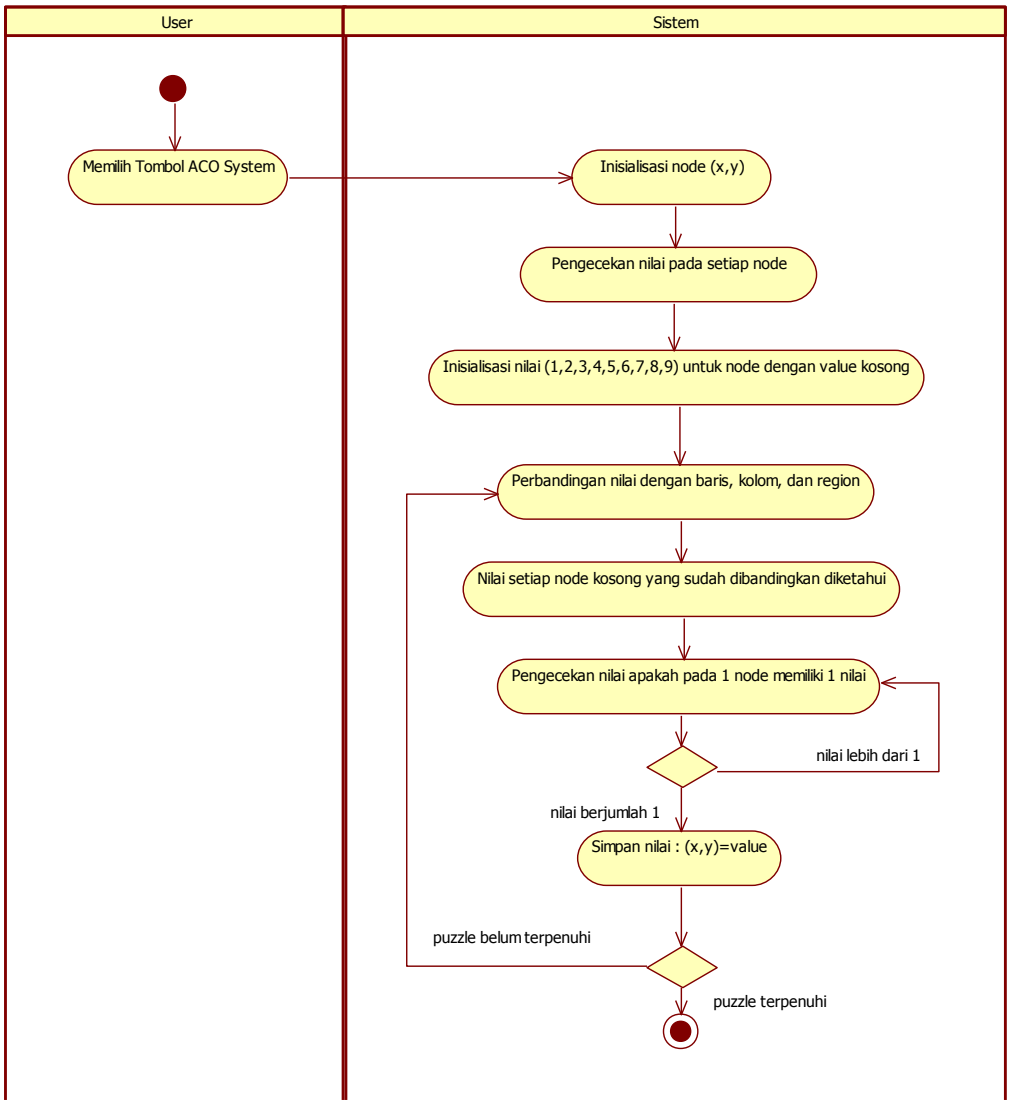
Gambar 3.22 *Subactivity Crossover*



Gambar 3.23 *Subactivity* Mutasi

4. *Activity Diagram* Algoritma *Ant Colony Optimization System*

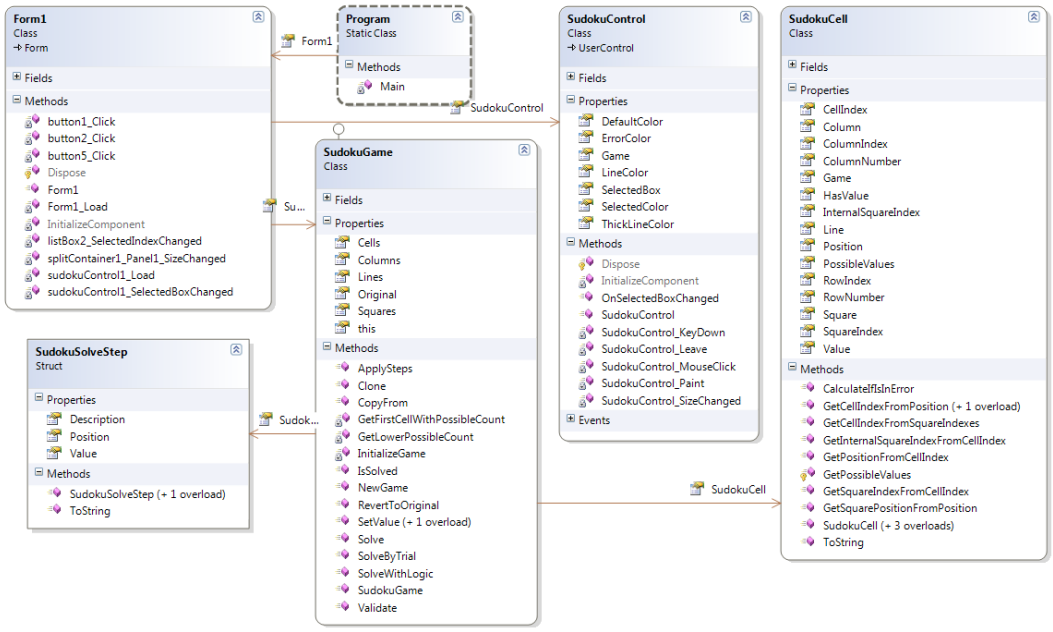
Berikut ini merupakan diagram yang menunjukan alur aksi pada aktivitas Algoritma *Ant Colony Optimization System* yang dapat dilihat pada gambar di bawah ini.



Gambar 3.24 *Activity Diagram* Algoritma *Ant Colony Optimization System*

3.1.4.3 *Class Diagram*

Class diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem.

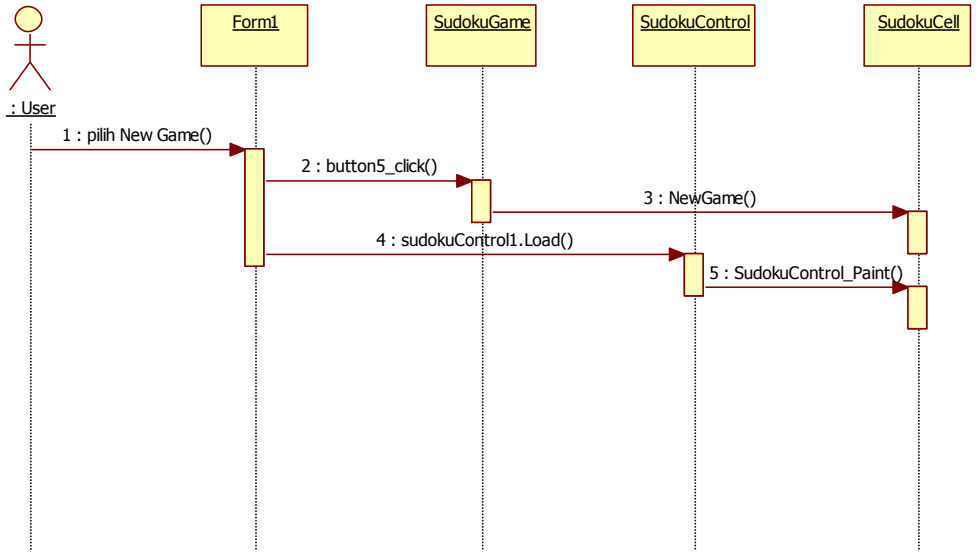


Gambar 3.25 *Class Diagram*

3.1.4.4 *Sequence Diagram*

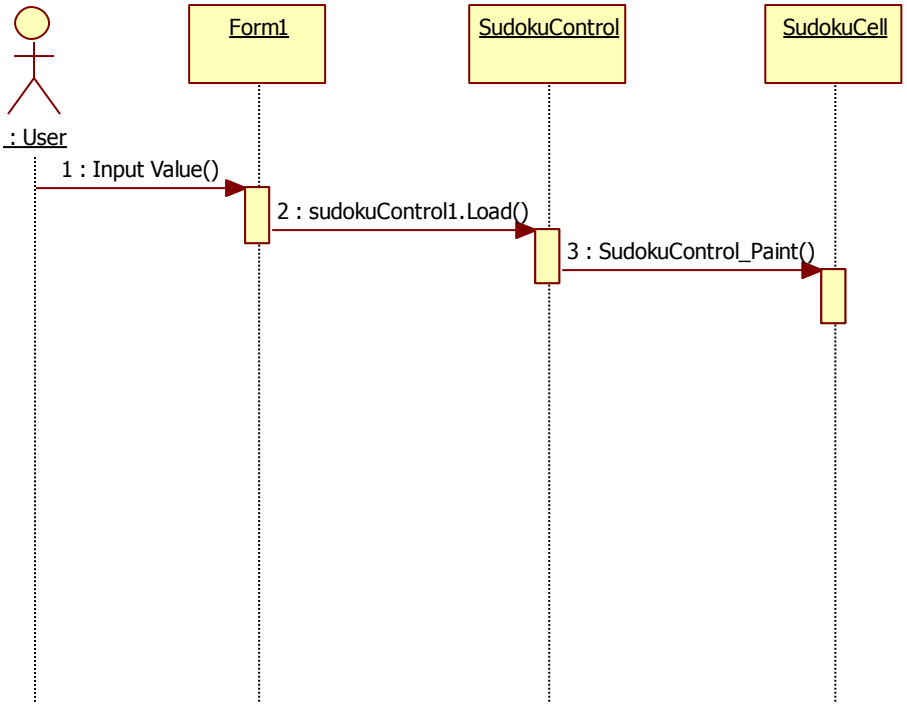
Sequence diagram menggambarkan apa yang dilakukan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan message yang dikirmkan dan diterima objek.

1. ***Sequence Diagram New Game***



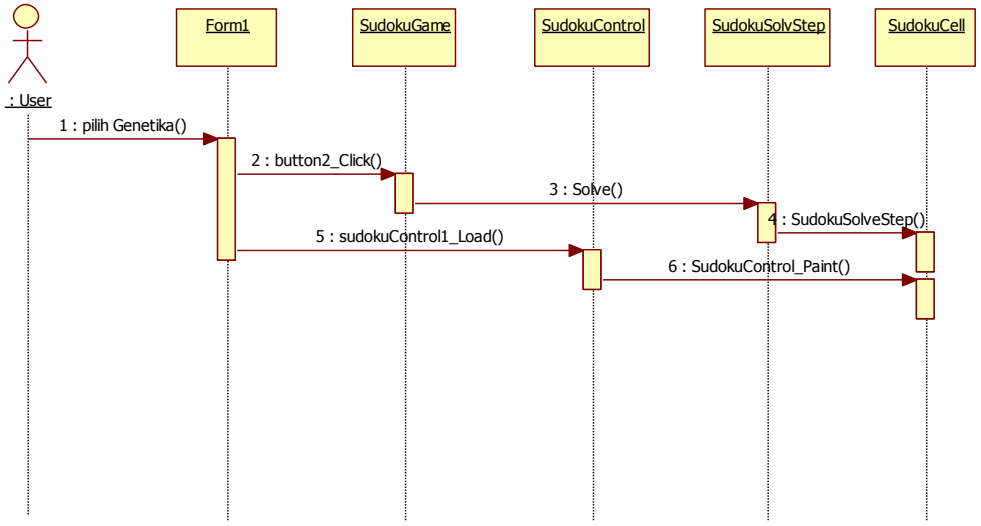
Gambar 3.26 *Sequence Diagram New Game*

2. ***Sequence Diagram Input Value***



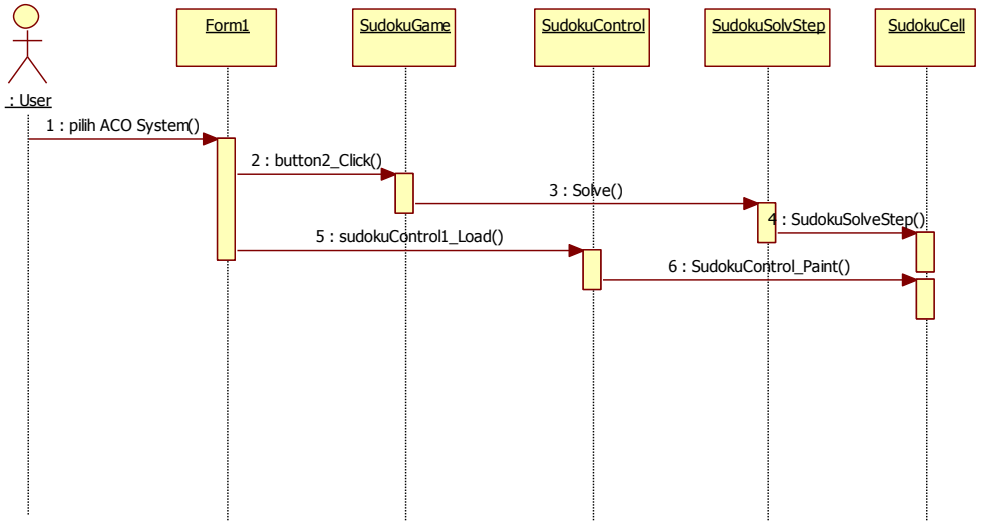
Gambar 3.27 *Sequence Diagram Input Value*

3. *Sequence Diagram* Algoritma Genetika



Gambar 3.28 *Sequence Diagram* Algoritma Genetika

4. *Sequence Diagram* Algoritma ACO *System*



Gambar 3.29 *Sequence Diagram* Algoritma ACO *System*

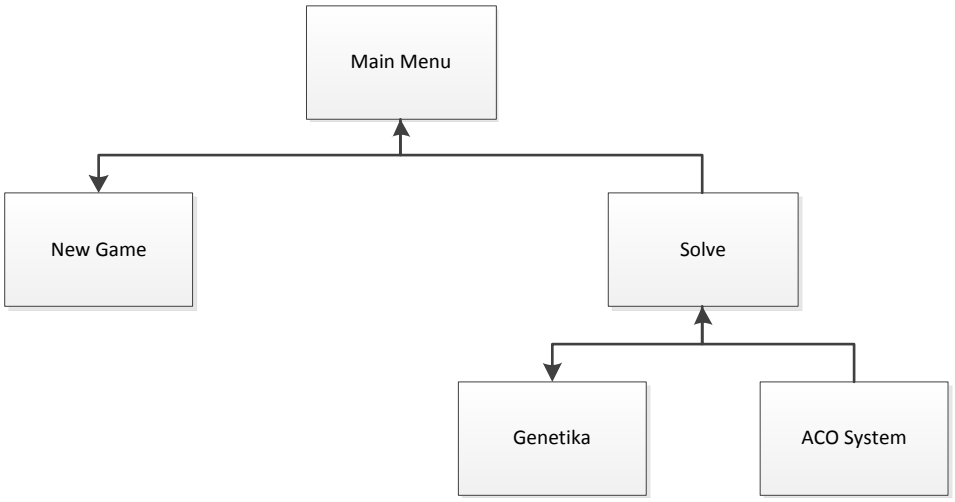
3.2 Perancangan Sistem

Perancangan sistem adalah suatu bagian dari metodologi pembangunan suatu perangkat lunak yang dilakukan setelah tahapan analisis sistem untuk memberikan gambaran secara terperinci sistem yang akan dibangun. Dalam perancangan sistem pada aplikasi ini, maka dilakukan beberapa langkah, yaitu:

1. Perancangan Struktur Menu
2. Perancangan Antarmuka
3. Jaringan Semantik
4. Perancangan *Method*

3.2.1 Perancangan Struktur Menu

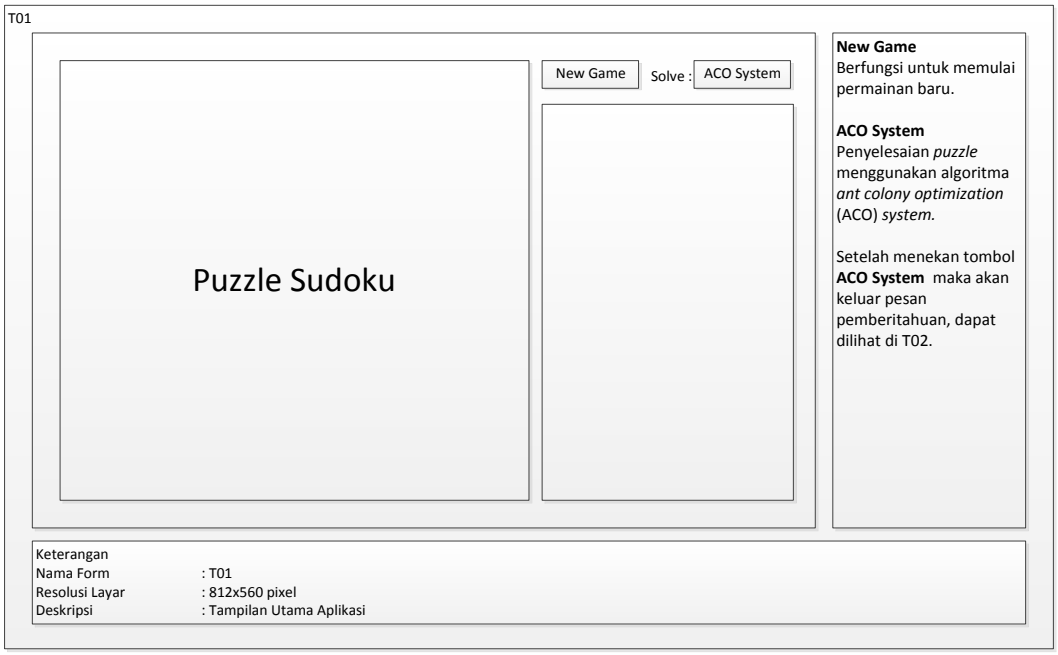
Perancangan Struktur menu adalah bagian dari perancangan sistem yang gambarannya disusun berstruktur, dapat berfungsi sebagai navigasi halaman-halaman yang ada dalam aplikasi yang dibangun. Perancangan struktur pada aplikasi ini adalah sebagai berikut:



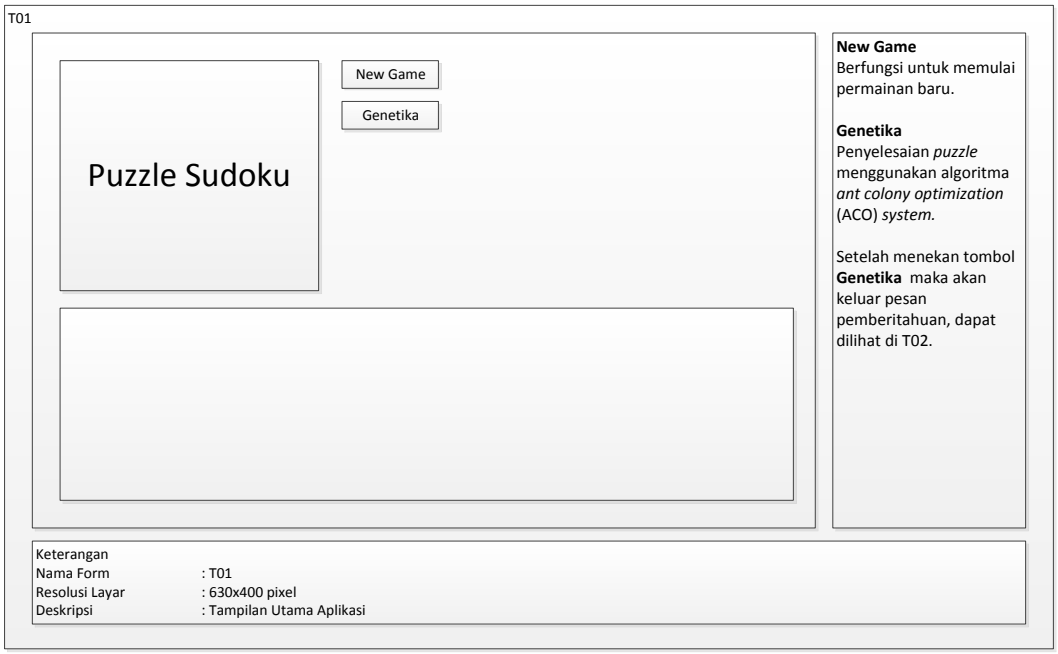
Gambar 3.30 Struktur Menu

3.2.2 Perancangan Antarmuka

Perancangan antarmuka dibutuhkan untuk mewakili keadaan sebenarnya dari aplikasi yang akan dibangun. Perancangan antarmuka pada aplikasi ini adalah sebagai berikut:



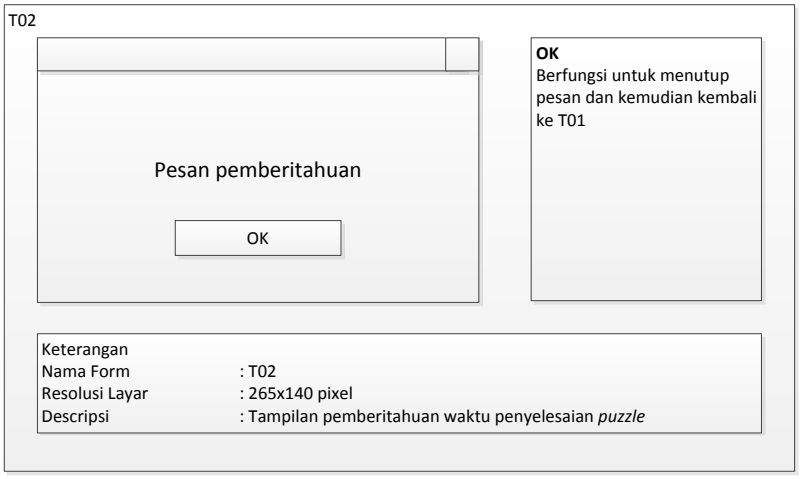
Gambar 3.31 Tampilan Utama Aplikasi (*ACO System*)



Gambar 3.32 Tampilan Utama Aplikasi (Genetika)

3.2.3 Perancangan Pesan

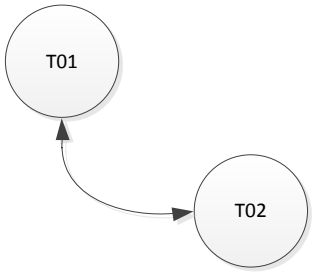
Perancangan pesan merupakan pesan yang muncul guna memberi peringatan atau pemberitahuan kepada *user*. Pada aplikasi ini perancangan pesan digunakan untuk memberi tahu *user* waktu penyelesaian *puzzle*. Perancangan pesan pada aplikasi ini adalah sebagai berikut:



Gambar 3.33 Pesan Pemberitahuan

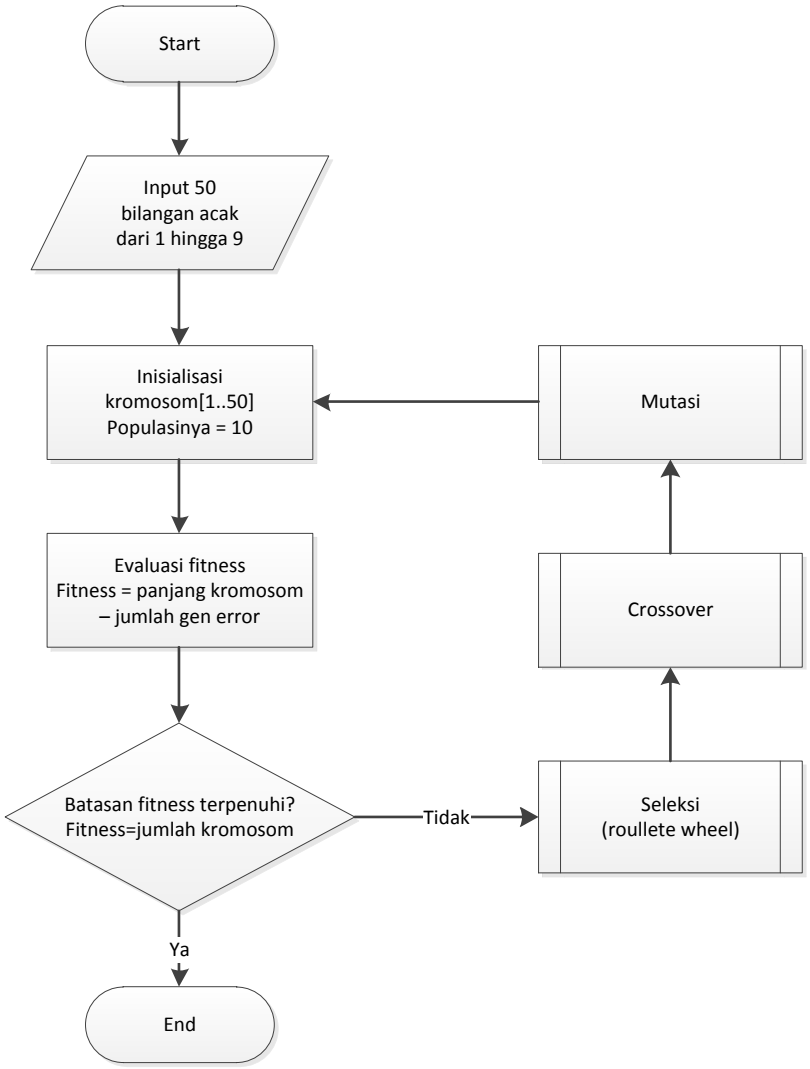
3.2.4 Jaringan Semantik

Jaringan semantik merupakan gambaran tentang relasi dari masing-masing halaman yang bisa diakses oleh pengguna. Jaringan semantik pada aplikasi ini adalah sebagai berikut:

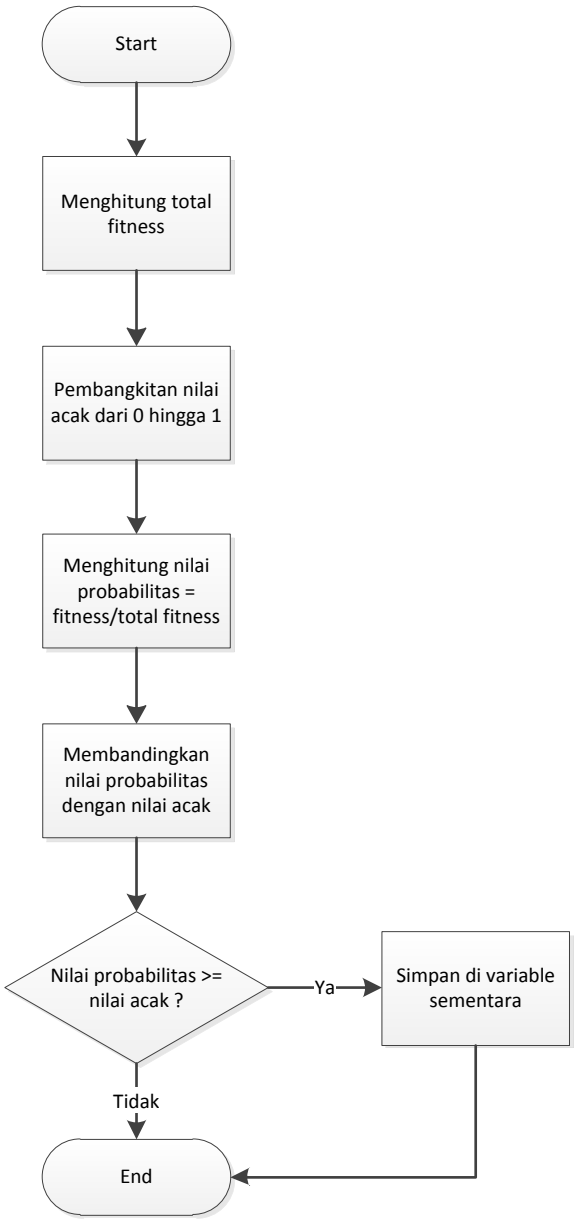


Gambar 3.34 Jaringan Semantik

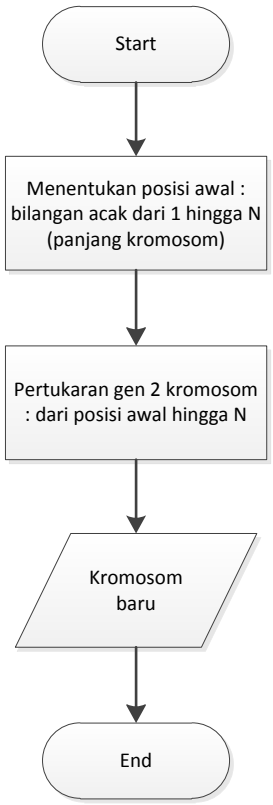
3.2.5 Perancangan *Method*



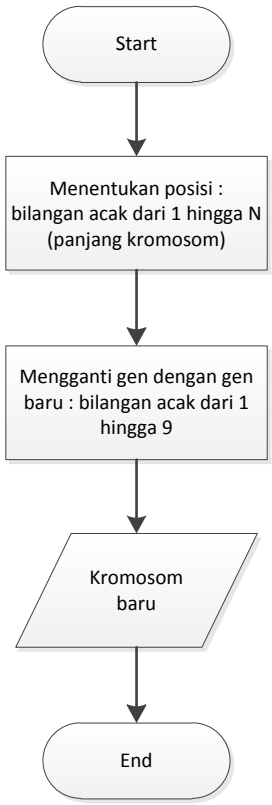
Gambar 3.35 *Method* Genetika



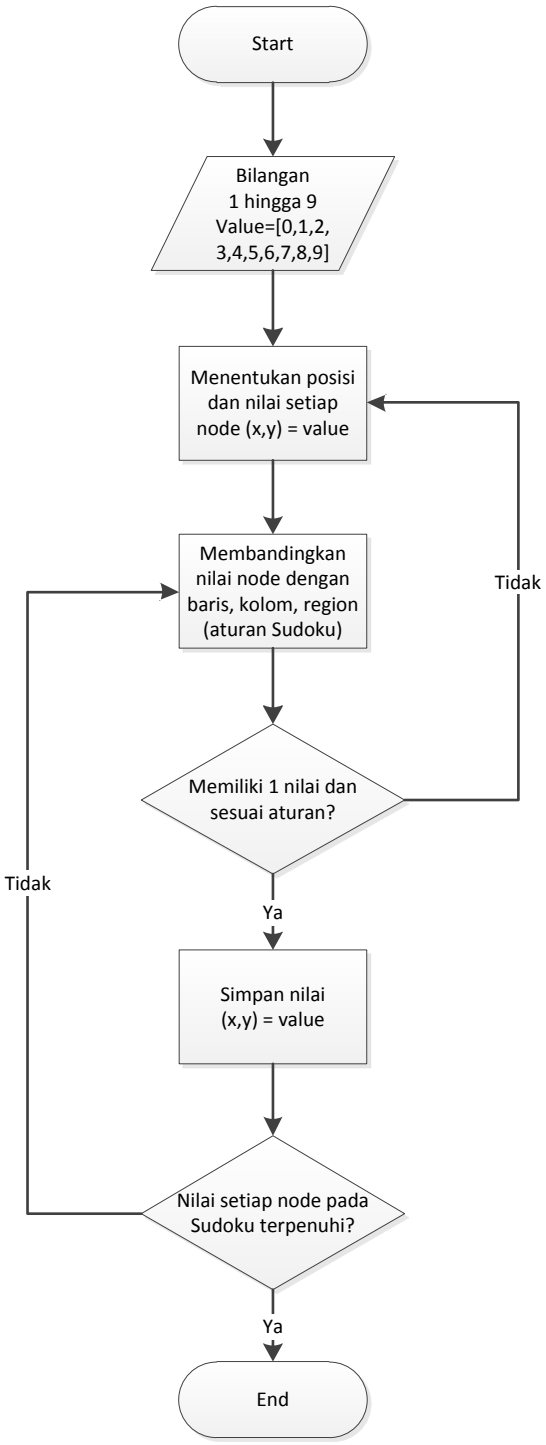
Gambar 3.36 *Submethod* Seleksi



Gambar 3.37 *Submethod Crossover*



Gambar 3.38 *Submethod* Mutasi



Gambar 3.39 *Method* ACO *System*