BAB 4
IMPLEMENTASI DAN PENGUJIAN SIMULASI

4.1. Implementasi

Pada tahapan implementasi ini, algoritma *firefly* akan diimplementasikan ke dalam sistem menggunakan bahasa pemrograman C# sesuai dengan perancangan yang telah dilakukan. Implementasi ini merupakan tahap penerjemah kebutuhan pembangunan *simulator* ke dalam representasi perangkat lunak sesuai dengan hasil analisis yang telah dilakukan. Setelah implementasi maka dilakukan pengujuan sistem yang baru dimana akan dilihat kekurangan-kekurangan pada *simulator* yang baru untuk selanjutnya diadakan pengembangan sistem.

Tujuan implementasi sistem adalah untuk menjelaskan tentang manual modul kepada penguji sistem yang akan menggunakan sistem. Sehingga penguji sistem tersebut dapat merespon apa yang ditampilkan di sistem dan memberikan masukan kepada pembuat sistem untuk dilakukan perbaikan agar sistem lebih baik lagi.

4.1.1. Perangkat Lunak Pembangun

Perangkat lunak yang digunakan dalam implementasi algoritma *firefly* untuk memperkirakan jumlah dan biaya produksi roti adalah sebagai berikut:

1. Sistem operasi *Windows* 7
2. *Microsoft Visual* C# 2010 Express, pemrograman dengan Bahasa C#

4.1.2. Perangkat Keras Pembangun

Perangkat keras yang digunakan dalam implementasi algoritma *firefly* untuk memperkirakan jumlah dan biaya produksi roti adalah sebagai berikut:

1. *Processor* dengan kecepatan minimum 2.0 GHz.
2. *Memory* / RAM : minimum 2 GB
3. *Harddisk* minimum berkapasitas 120 GB
4. VGA minimum 1 GB

4.1.3. Implementasi Class

Implementasi class pada Implementasi Algoritma *Firefly* untuk memperkirakan jumlah dan biaya produksi roti di CV. Desi Bakery dapat dilihat pada Tabel 4.1.

Tabel 4.1 Implementasi Class

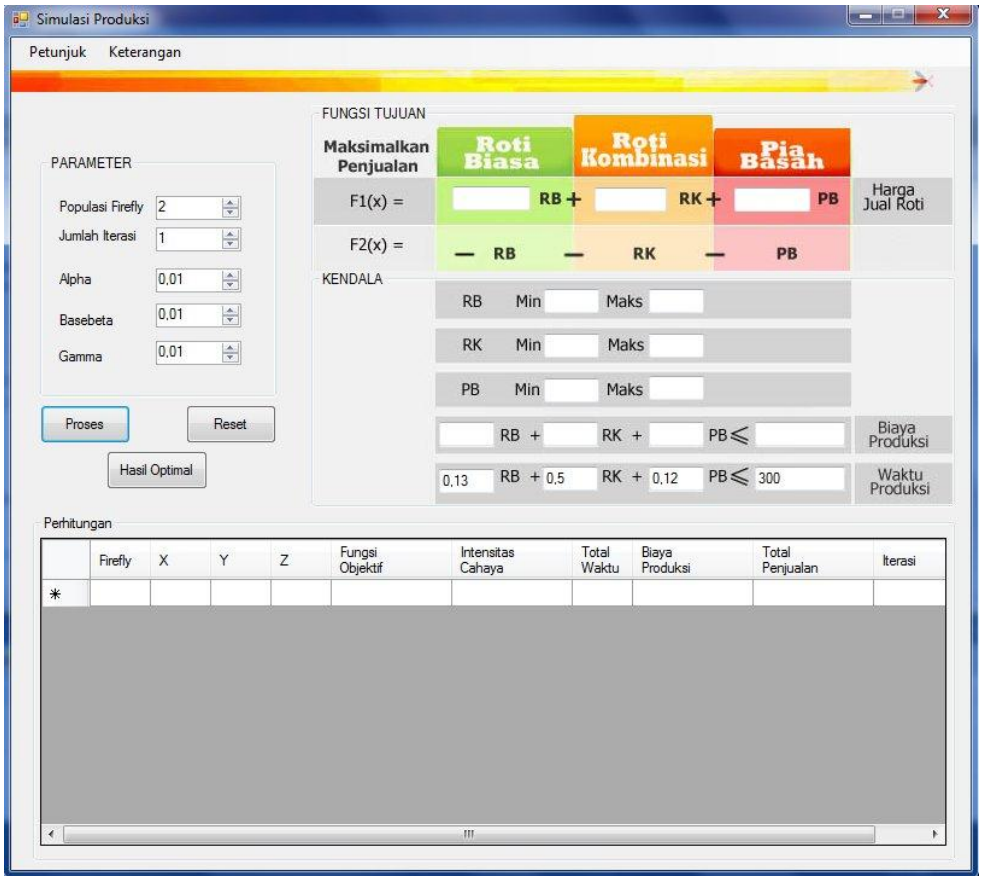
No	Nama Kelas	Nama File
1	MenuUtama	MenuUtama.cs
2	FireFly	FireFly.cs
3	HasilOptimasi	HasilOptimasi.cs
4	Petunjuk	Petunjuk.cs
5	Keterangan	Keterangan.cs

4.1.4. Implementasi Antarmuka

Implementasi antarmuka merupakan tampilan keseluruhan sistem yaitu menu utama, petunjuk, keterangan dan hasil optimasi sebagai berikut:

1. Implementasi Antarmuka Menu Utama

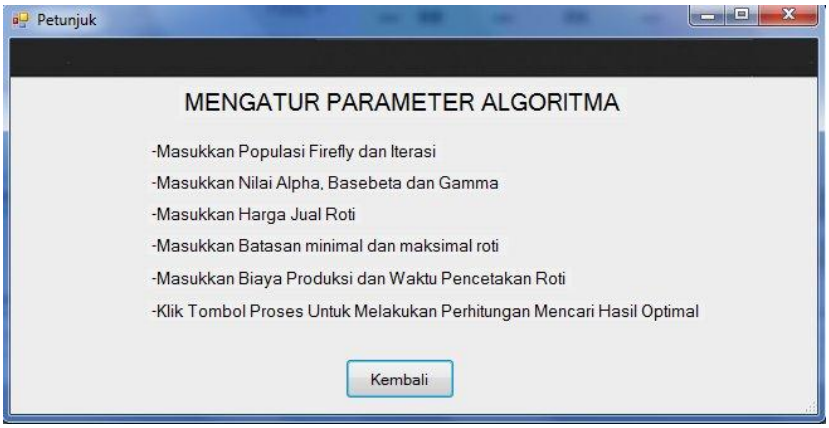
   Implementasi antarmuka menu utama merupakan tampilan awal *simulator* dari implementasi algoritma *firefly* untuk memperkirakan jumlah dan biaya produksi roti pada simulasi produksi. Isian parameter populasi *firefly*, jumlah iterasi, *alpha, basebeta, gamma* untuk memasukan parameter-parameter algoritma *firefly*, isian harga jual untuk menentukan harga jual setiap jenis roti, isian RB, RK, PB min/maks untuk menentukan jumlah acak setiap roti sesuai dengan batasan-batasannya, isian biaya produksi dan waktu pencetakan untuk memasukan kendala biaya dan waktu pencetakannya, tombol proses untuk menjalankan algoritma pencarian kemudian menampilkannya di *gridview*, tombol reset untuk mengembalikan data masukkan ke keadaan awal dan tombol hasil optimal untuk menampilkan hasil optimal dari hasil pencarian kombinasi jumlah solusi yang optimal. Implementasi antarmuka menu utama dapat dilihat pada Gambar 4.1.



Gambar 4.1 Implementasi Antarmuka Menu Utama

2. Implementasi Antarmuka Petunjuk

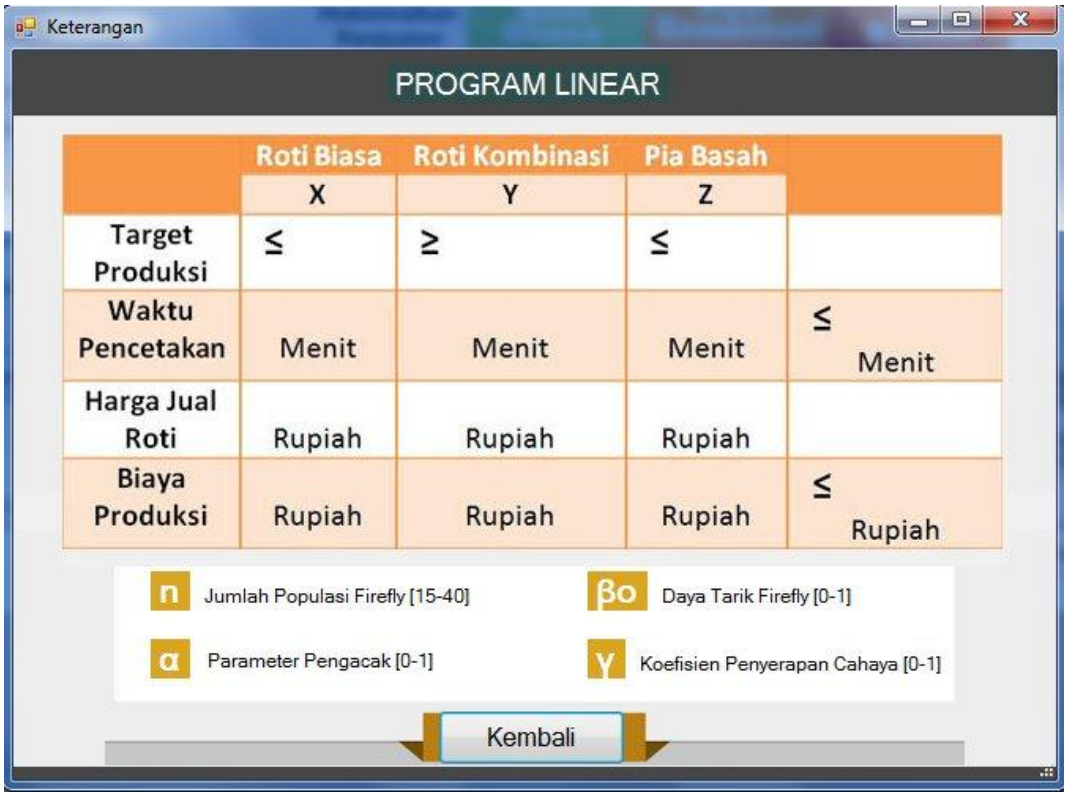
   Implementasi antarmuka petunjuk merupakan tampilan untuk menampilkan petunjuk penggunaan *simulator*, tombol kembali untuk kembali ke menu utama *simulator*. Implementasi antarmuka petunjuk dapat dilihat pada Gambar 4.2



Gambar 4.2 Implementasi Antarmuka Petunjuk

3. Implementasi Antarmuka Keterangan

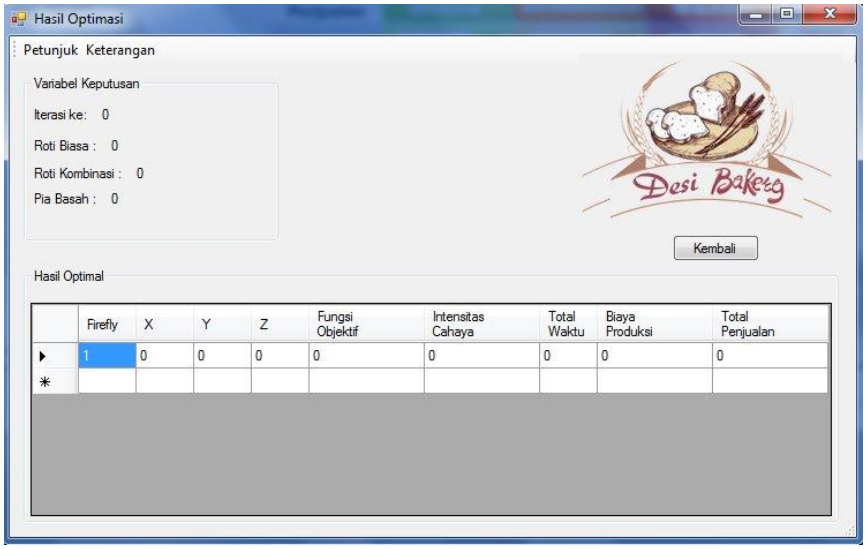
Implementasi antarmuka keterangan merupakan tampilan untuk menampilkan keterangan tentang keterbatasan atau kendala dalam masalah produksi, tombol kembali untuk kembali ke menu utama *simulator*. Implementasi antarmuka keterangan dapat dilihat pada Gambar 4.3Gambar 4.4



Gambar 4.3 Implementasi Antarmuka Keterangan

4. Implementasi Antarmuka Hasil Optimasi

Implementasi antarmuka hasil optimasi merupakan tampilan untuk menampilkan hasil optimal kombinasi jumlah roti dengan waktu dan biaya produksi yang sudah dioptimasi., tombol kembali untuk kembali ke menu utama. Implementasi antarmuka hasil optimasi dapat dilihat pada Gambar 4.5



Gambar 4.5 Implementasi Antarmuka Hasil Optimasi

4.2. Pengujian Sistem

Pengujian perangkat lunak perlu dilakukan untuk mengevaluasi baik secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum pada sistem yang telah dibuat, dan untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya. Pengujian dari penelitian tugas akhir ini adalah dengan menggunakan metode pengujian *black box* dan *white box* yang berfokus pada persyaratan fungsional dari sistem atau perangkat lunak yang dibangun.

4.2.1. Pengujian *Black Box*

Black-Box Testing merupakan pengujian yang berfokus pada spesifikasi fungsional dari perangkat lunak, *tester* dapat mendefinisikan kumpulan kondisi *input* dan melakukan pengetesan pada spesifikasi *fungsional* program. Pengujian *alpha* dilakukan menggunakan metode *black box*. Untuk menentukan pengujian *alpha* ini hanya dilakukan atau diuji pada simulasi yang digunakan oleh penguji simulasi. Tabel rencana pengujian simulasi produksi dari implementasi algoritma *firefly*, yaitu pada Tabel 4.2.

Tabel 4.2 Rencana Pengujian Simulasi Produksi

Kelas Uji	Pengujian	Jenis Pengujian
MenuUtama dan FireFly	Menampilkan Menu Utama dan perhitungan algoritma firefly	*Blackbox*
Petunjuk	Menampilkan Petunjuk Penggunaan simulasi	*Blackbox*
Keterangan	Menampilkan Keterangan Variabel Analisis	*Blackbox*
HasilOptimasi	Menampilkan Hasil Optimal	*Blackbox*

4.2.2. Kasus dan Hasil Pengujian Black Box

Kasus dan hasil pengujian berisi pemaparan dari rencana pengujian yang telah disusun pada skenario pengujian. pengujian *black box* ini menggunakan teknik pengujian *Equivalence Partitioning* adalah metode pengujian *black box* yg memecah atau membagi domain input dari program ke dalam kelas-kelas data sehingga *test case* dapat diperoleh, *input* data dan *output* hasil terdapat di kelas yang berbeda yang sesuai dengan kelas inputnya. Perancangan *test case equivalence partitioning* berdasarkan masing-masing kelas equivalensi partition diproses dimana program akan memproses anggota kelas-kelas tersebut secara *equivalen* dan *test cases* dipilih dari masing-masing partisi.

1. Pengujian Kelas MenuUtama dan FireFly

   Pengujian ini dilakukan untuk menguji persyaratan fungsional menu utama yang terdiri dari pengaturan parameter, menjalankan algoritma pencarian dan menampilkan solusi produksi. Kasus dan hasil uji dengan data normal dapat dilihat pada Tabel 4.3

Tabel 4.3 Kasus dan Hasil Uji (Data Benar)

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	**Yang diharapkan**	**Pengamatan**	**Kesimpulan**
Tampil Menu Utama	Menampilkan menu pilihan	Menampilkan menu yang dipilih	[√] Diterima [ ] Ditolak
Memasukkan parameter: populasi firefly, jumlah iterasi, alpha, basebeta, gamma.	Dapat terisi dengan masukkan data populasi firefly, jumlah iterasi, alpha, basebeta dan gamma.	Dapat mengisi populasi firefly, jumlah iterasi, alpha, basebeta dan gamma yang diharapkan	[√] Diterima [ ] Ditolak
Memasukkan fungsi tujuan: harga jual roti	Dapat terisi dengan masukkan data harga jual roti	Dapat mengisi harga jual roti yang diharapkan	[√] Diterima [ ] Ditolak
Memasukkan kendala: RB, RK, PB min/maks	Dapat terisi dengan masukkan data jumlah kombinasi roti acak sesuai dengan batasan yang ditentukan	Dapat mengisi jumlah kombinasi roti RB, RK dan PB yang diharapkan	[√] Diterima [ ] Ditolak
Memasukkan kendala: biaya produksi	Dapat terisi dengan masukkan data biaya produksi setiap roti	Dapat mengisi biaya produksi setiap jenis roti yang diharapkan	[√] Diterima [ ] Ditolak
Memasukkan kendala: waktu pencetakan	Dapat terisi dengan masukkan data waktu pencetakan setiap roti	Dapat mengisi waktu pencetakan setiap jenis roti yang diharapkan	[√] Diterima [ ] Ditolak
Menekan tombol proses	Menjalankan algoritma untuk mencari solusi dan menampilkan progressnya ke gridview	Tombol dapat berfungsi sesuai yang diharapkan	[√] Diterima [ ] Ditolak
Menekan tombol reset	Mengembalikan konfigurasi simulator	Tombol dapat berfungsi sesuai yang	[√] Diterima [ ] Ditolak

	ke kondisi terselesaikan	diharapkan	
Menekan tombol hasil optimal	Menampilkan form hasil optimasi	Tombol dapat berfungsi sesuai yang diharapkan	[√] Diterima [ ] Ditolak

Kasus dan hasil uji data salah dapat dilihat pada Tabel 4.4

Tabel 4.4 Kasus dan Hasil Uji (Data Salah)

Kasus dan Hasil Uji (Data Salah)			
Data Masukkan	**Yang diharapkan**	**Pengamatan**	**Kesimpulan**
Tampil Menu Utama	Menampilkan menu pilihan	Tidak dapat memilih menu	[ ] Diterima [√] Ditolak
Memasukkan fungsi tujuan: harga jual roti	Dapat terisi dengan masukkan data harga jual roti	Tidak mengisi atau tidak memasukkan data berupa angka	[ ] Diterima [√] Ditolak
Memasukkan kendala: RB, RK, PB min/maks	Dapat terisi dengan masukkan data jumlah kombinasi roti acak sesuai dengan batasan yang ditentukan	Tidak mengisi atau tidak memasukkan data berupa angka	[ ] Diterima [√] Ditolak
Menekan tombol proses	Menjalankan algoritma untuk mencari solusi & menampilkan progress nya ke gridview	Tidak mengisi atau mengisinya dengan data bukan angka pada parameter: populasi firefly, jumlah iterasi, alpha, basebeta, gamma, fungsi tujuan: harga jual, kendala: RB, RK, PB min/maks, biaya produksi dan waktu pencetakan.	[ ] Diterima [√] Ditolak

Menekan tombol hasil optimal	Menampilkan form hasil optimasi	Tidak mengisi atau mengisinya dengan data bukan angka pada parameter: populasi firefly, jumlah iterasi, alpha, basebeta, gamma, fungsi tujuan: harga jual, kendala: RB, RK, PB min/maks, biaya produksi dan waktu pencetakan.	[ ] Diterima [√] Ditolak

2. Pengujian Kelas Petunjuk

   Pengujian ini dilakukan untuk menguji persyaratan fungsional pemanggilan jendela petunjuk. Kasus dan hasil ujinya dapat dilihat pada Tabel 4.5

Tabel 4.5 Kasus dan Hasil Pengujian Petunjuk

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	**Yang diharapkan**	**Pengamatan**	**Kesimpulan**
Memilih Petunjuk dari menu bantuan	Menampilkan jendela petunjuk penggunaan simulator	Menampilkan informasi berupa teks di layar	[√] Diterima [ ] Ditolak
Menekan tombol kembali	Menampilkan jendela menu utama simulator	Menampilkan informasi berupa teks di layar	[√] Diterima [ ] Ditolak

3. Pengujian Kelas Keterangan

   Pengujian ini dilakukan untuk menguji persyaratan fungsional pemanggilan jendela keterangan. Kasus dan hasil ujinya dapat dilihat pada Tabel 4.6

Tabel 4.6 Kasus dan Hasil Pengujian Keterangan

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	**Yang diharapkan**	**Pengamatan**	**Kesimpulan**
Memilih Keterangan dari menu bantuan	Menampilkan jendela keterangan variabel analisis	Menampilkan informasi berupa teks dilayar	[√] Diterima [ ] Ditolak
Menekan tombol kembali	Menampilkan jendela menu utama simulator	Menampilkan informasi berupa teks di layar	[√] Diterima [ ] Ditolak

4. Pengujian Kelas HasilOptimasi

   Pengujian ini dilakukan untuk menguji persyaratan fungsional pemanggilan jendela hasil optimasi. Kasus dan hasil ujinya dapat dilihat pada Tabel 4.7

Tabel 4.7 Kasus dan Hasil Pengujian Hasil Optimasi

Kasus dan Hasil Uji (Data Benar)			
Data Masukkan	**Yang diharapkan**	**Pengamatan**	**Kesimpulan**
Menekan tombol hasil optimal	Menampilkan form hasil optimasi	Tombol dapat berfungsi sesuai yang diharapkan	[√] Diterima [ ] Ditolak
Menekan tombol kembali	Menampilkan jendela menu utama simulator	Menampilkan informasi berupa teks di layar	[√] Diterima [ ] Ditolak

4.2.3. Pengujian *White Box*

Pada dasarnya *white box testing* sistem di perlukan dalam membangun ataupun menganasila sebuah sistem apakah sudah berjalan sebagai mana mestinya atau tidak. Hal ini perlu dilakukan mengingat *white box testing* merupakan pengecekan sistem secara lebih detail dan komplek ketimbang system *black box testing*, di *white box testing* tidak hanya di cek dari fungsional luarnya saja melainkan juga dari berbagai aspek, termasuk diagram alur system dan berbagai komponent lainnya yang ada dalam sistem.

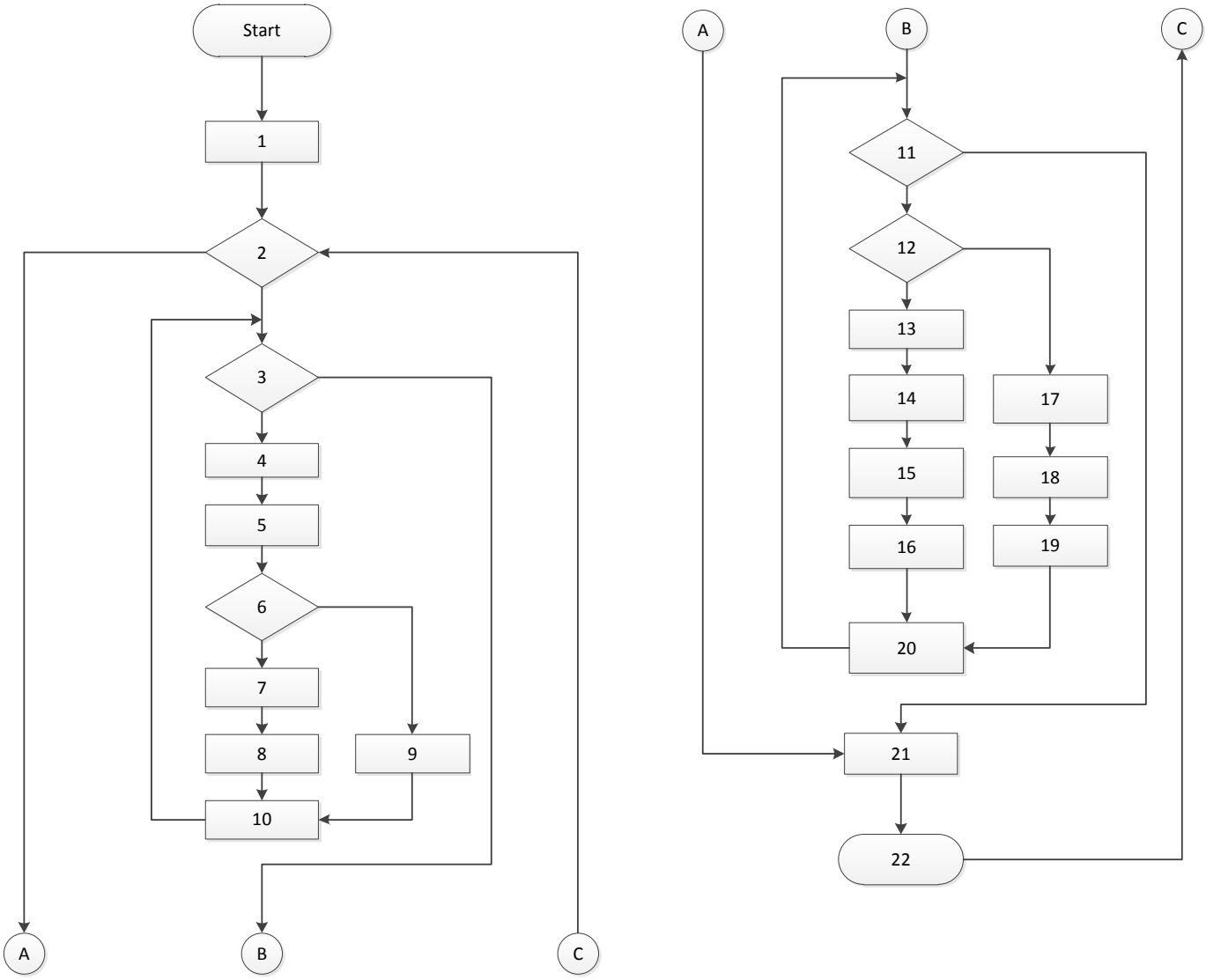
White Box Testing adalah salah satu cara untuk menguji suatu aplikasi atau *software* dengan cara melihat modul untuk dapat meneliti dan menganalisa kode dari program yang di buat ada yang salah atau tidak. Jika modul yang telah dan sudah di hasilkan berupa output yang tidak sesuai dengan yang di harapkan maka akan di compile ulang dan di cek kembali kode-kode tersebut hingga mencapai sesuai denganyang di harapkan. Metode *white box* mengijinkan pendesain kasus uji untuk mendapatkan perkiraan lojik yang kompleks dari desain metode dan menggunakan perkiraan ini untuk mendefinisikan aliran eksekusi. Tahapan *white box* meliputi :

1. Mengubah *source code* dari algoritma *firefly* menjadi *flowchart*. Perubahan *source code* tersebut dapat dilihat pada Gambar 4.6

```
1   public AlgoFirefly(){
    InitializeComponent();
    gBestPosition = 0;
    Random rand = new Random();
    fireFlyResult = new FireFly();
    this.randPergerakan = rand.NextDouble();
    int maxIterasi = 0;
2         while (maxIterasi < nudIterasi.Value) {
3            for (int i = 0; i < fireFly.Length; i++) {
4            fireFly[i].evaluasiFireFly());
5            fireFly[i].hitungIntensitas();
6               if (fireFly[i].nilaiIntensitasCahaya > this.gBest)  {
7               this.gBest = fireFly[i].nilaiIntensitasCahaya;
                this.gBestPosition = i;
8               }
9                   fireFly[i].dimensiRandomNumber();
10           }
11           for (int i = 0; i < fireFly.Length; i++) {
12              if (i != this.gBestPosition) {
13  fireFly[i].hitungJarak(fireFly[gBestPosition].posisiRB,
    fireFly[gBestPosition].posisiRK, fireFly[gBestPosition].posisiPB);
14  fireFly[i].hitungKeaktraktifan((d(double)nudBeta.Value);
15  fireFly[i].hitungPergerakan((double)nudAlpha.Value,
    this.randPergerakan,fireFly[gBestPosition].posisiRB,
    fireFly[gBestPosition].posisiRK, fireFly[gBestPosition].posisiPB);
16              }
17               fireFly[i].hitungTotalWaktu();
18               fireFly[i].hitungBiayaProduksi();
19               fireFly[i].hitungTotalPenjualan();
20           }
21               MaxIterasi++;
22  }
```

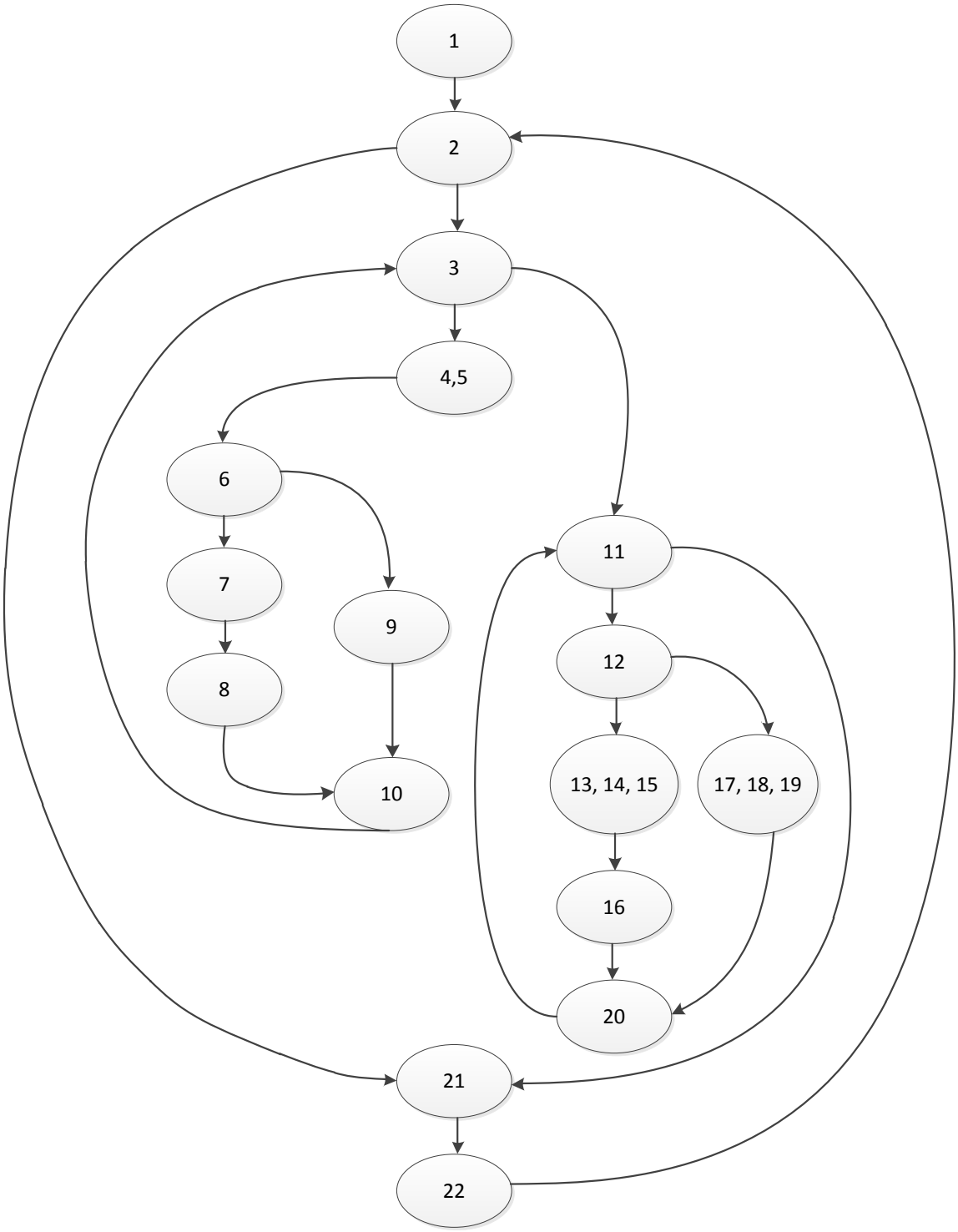
Gambar 4.6 Soure Code Methode Algoritma Firefly

Hasil pengubahan dan penomoran dari *source code* menjadi *flowchart* dapat dilihat pada Gambar 4.7.



Gambar 4.7 Penomoran Flowchart Algoritma Firefly untuk Simulasi Produksi

2. Mengubah *flowchart* menjadi *flowgraph*. Hasil dari pengubahan *flowchart* menjadi *flowgraph* dapat dilihat pada Gambar 4.8.



Gambar 4.8 Flowgraph Algoritma Firefly untuk Simulasi Produksi

3. *Cyclomatic complexity*

   Tahapan selanjutnya yaitu menghitung *cyclomatic complexity*. Berikut ini adalah hasil perhitungan *cyclomatic complexity*:

   V(G) = E – N + 2

   V(G) = 22 Busur - 17 Simpul + 2

   V(G) = 7

4. *Predicate node*

   Predicate node adalah node yang mengandung kondisi

   V(G) = P + 1

   V(G) = 6 *Predicate* + 1

   V(G) = 7

5. *Independent Path*

   Dari hasil perhitungan *cyclomatic complexity* terdapat 7 *independent path* yaitu :

   Path 1 : 1-2-21-22

   Path 2 : 1-2-3-11-21-22

   Path 3 : 1-2-3-11-12-17-18-19-21-22

   Path 4 : 1-2-3-11-12-13-14-15-16-20-21-22

   Path 5 : 1-2-3-4-5-6-9-10-3-11-12-17-18-19-21-22

   Path 6 : 1-2-3-4-5-6-7-8-10-3-11-12-17-18-19-20-11-21-22

   Path 7 : 1-2-3-4-5-6-7-8-10-3-11-12-13-14-15-16-20-11-21-22

6. Hitung Jumlah *Region*

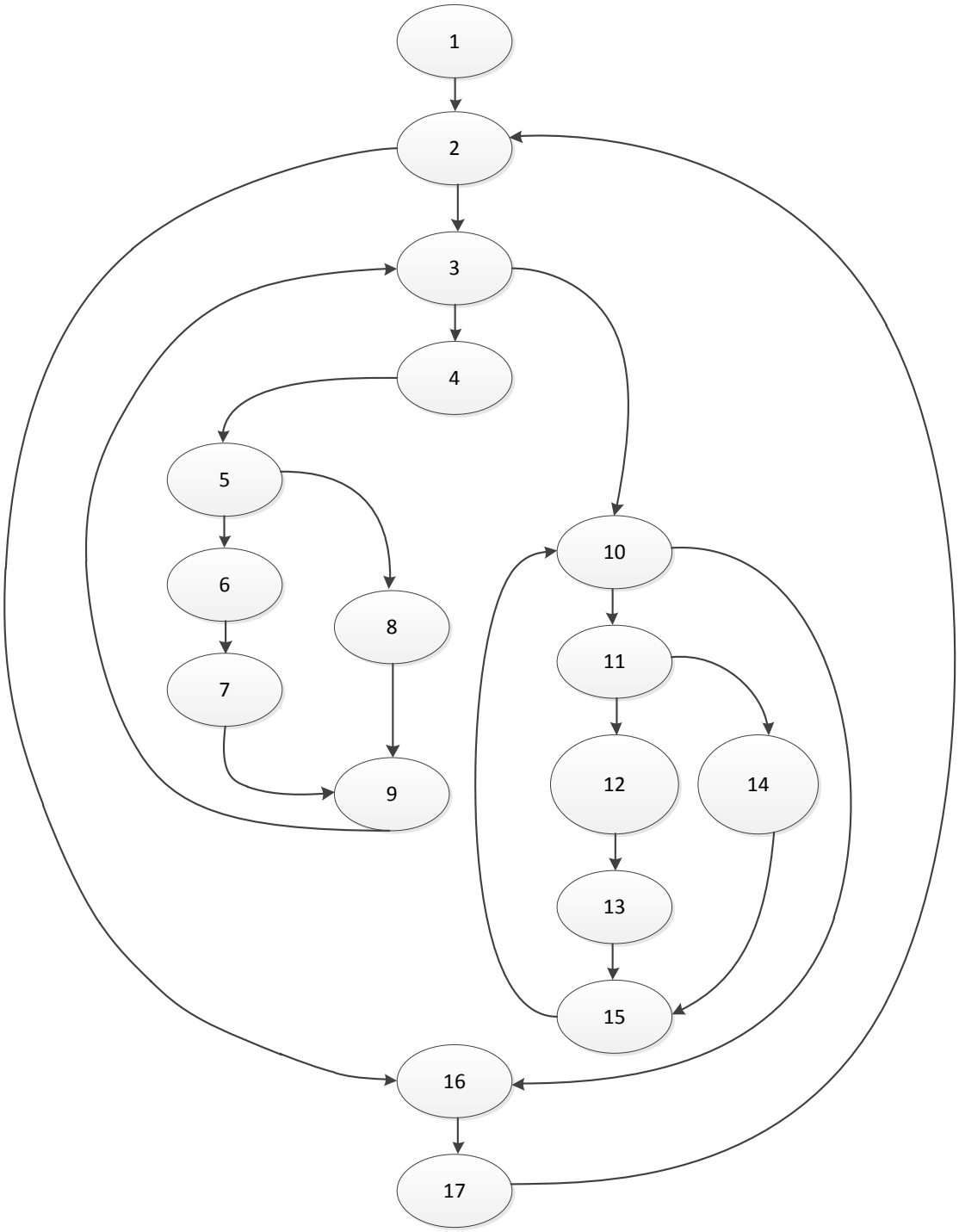
   Jumlah *region* = jumlah *kurva* tertutup + 1 (*region* terluar)

   Jumlah *region* = 6 + 1

   Jumlah *region* = 7

7. Penomoran ulang *flowgraph*

Penomoran ulang *flowgraph* di perlihatkan pada Gambar 4.9.



Gambar 4.9 Penomoran Ulang Flowgraph

8. *Graph Matrix*

Pembentukan *Matrix* dan perhitungan V(G) diperlihatkan pada Tabel 4.8.

Tabel 4.8 Graph Matrix

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Perhitungan
1		1																0
2			1													1		1
3				1						1								1
4					1													0
5						1		1										1
6							1											0
7									1									0
8									1									0
9			1									1						1
10											1					1		1
11												1		1				1
12													1					0
13															1			0
14															1			0
15										1								0
16																	1	0
17		1																0
Jumlah (X)																		6

V(G) = X + 1

V(G) = 6 + 1

V(G) = 7

Berdasarkan pengujian pada setiap metode, dihasilkan nilai *Cyclomatic Complexity* yang sama yaitu 7. Maka dapat disimpulkan bahwa pengujian *white box* pada algoritma *firely* berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

4.2.4. Skenario Pengujian

Pengujian simulasi produksi yang telah diimplementasi algoritma *firefly* untuk menentukan solusi optimal dalam menentukan kombinasi jumlah produksi roti dengan parameter yang akan diuji dan perbandingan kombinasi acak dengan solusi yang dihasilkan. Pengujian untuk parameter dilakukan dengan jumlah populasi *firefly* 15, iterasi 100 dan pengujian dengan mengubah nilai dari parameter algoritma firefly yaitu *alpha*, *basebeta* dan *gamma*. Adapun Fungsi tujuan dan kendala yang digunakan adalah variabel analisis yang diperoleh dari studi kasus di CV. Desi Bakery yang ditunjukan pada Tabel 4.9 dan Tabel 4.10

Tabel 4.9 Pengujian Fungsi Tujuan

Fungsi Tujuan			
	Roti Biasa	Roti Kombinasi	Pia Basah
Harga Jual Roti	Rp 1600	Rp 4000	Rp 800

Tabel 4.10 Pengujian Kendala

Kendala				
	Roti biasa	Roti Kombinasi	Pia Basah	
min/maks	0 - 600	120 - 667	0 - 700	
Biaya produksi	Rp 300	Rp 600	Rp 200	≤ Rp 400.000
Waktu Pencetakan	0,13 menit	0,5 menit	0,12 menit	≤ 300 menit

Sekanrio Pengujian dilakukan yaitu melalui 4 tahap pengujian, pertama pengujian terhadap nilai alpha, kedua pengujian nilai basebata, ketiga pengujian nilai gamma dan terakhir pengujian menggunakan iterasi yang berbeda-beda. Adapun populasi kunang-kunang yang digunakan dalam setiap pengujian menggunakan 15 kunang-kunang. Hasil pengujian ditunjukan pada Tabel 4.11 dan Tabel 4.12

Tabel 4.11 Pengujian dengan mengubah nilai alpha

Jumlah Firefly	Kandidat Solusi	Jumlah Iterasi	Parameter			Intensitas Cahaya
			alpha	basebeta	gamma	
15	278, 373, 325	100	0,001	1	10	1021112
15	292, 386, 340	100	0,01	1	10	1067491
15	273, 369, 319	100	0,1	1	10	1041919,5
15	323, 372, 276	100	0,5	1	10	1015114,5
15	303, 353, 396	100	0,8	1	10	1195074
15	254, 352, 296	100	1	1	10	974749

Dari pengujian parameter alpha diatas didapatkan nilai parameter alpha yang terbaik yaitu 0,8 dengan menghasilkan nilai intensitas yang paling tinggi yaitu 1195074. Maka nilai alpha ini akan digunakan untuk pengujian selanjutnya.

Tabel 4.12 Pengujian dengan mengubah nilai basebeta

Jumlah Firefly	Kandidat Solusi	Jumlah Iterasi	Parameter			Intensitas Cahaya
			alpha	basebeta	gamma	
15	265, 362, 309	100	0,8	0,1	10	1059132
15	313, 365, 269	100	0,8	0,25	10	1087526,5
15	387, 332, 422	100	0,8	0,5	10	1155829,5
15	272, 367, 317	100	0,8	0,75	10	1077922
15	396, 353, 303	100	0,8	0,9	10	1195074
15	335, 287, 382	100	0,8	1	10	1127098

Dari pengujian parameter basebeta diatas didapatkan nilai parameter basebeta yang terbaik yaitu 0,9 dengan menghasilkan nilai intensitas yang paling tinggi yaitu 1195074. Maka nilai basebeta ini akan digunakan untuk pengujian selanjutnya.

Tabel 4.13 Pengujian dengan mengubah nilai gamma

Jumlah Firefly	Kandidat Solusi	Jumlah Iterasi	Parameter			Intensitas Cahaya
			alpha	basebeta	gamma	
15	399, 357, 306	100	0,8	0,9	0,01	1185069
15	328, 281, 376	100	0,8	0,9	0,5	1126307,5
15	304, 357, 260	100	0,8	0,9	3	905539,5
15	302, 356, 259	100	0,8	0,9	6	1039541,5
15	399, 306, 357	100	0,8	0,9	8,5	1056669
15	391, 298, 347	100	0,8	0,9	10	1158682

Dari pengujian parameter gamma diatas didapatkan nilai parameter gamma yang terbaik yaitu 0,01 dengan menghasilkan nilai intensitas yang paling tinggi yaitu 1185069. Maka nilai gamma ini akan digunakan untuk pengujian tahap terakhir. Setelah didapatkan parameter-parameter terbaik dari pengujian sebelumnya maka nilai alpha, basebeta dan gamma yang didapatkan akan digunakan untuk pengujian iterasi. Hasil pengujian ditunjukan pada Tabel 4.14.

Tabel 4.14 Pengujian dengan mengubah iterasi

Jumlah Firefly	Kandidat Solusi	Jumlah Iterasi	Parameter			Intensitas Cahaya
			alpha	basebeta	gamma	
15	359, 262, 306	10	0,8	0,9	0,01	911936,5
15	388, 343, 294	50	0,8	0,9	0,01	1017087,5
15	405, 313, 365	100	0,8	0,9	0,01	1205858,5
15	375, 322, 413	300	0,8	0,9	0,01	1123845
15	371, 275, 321	500	0,8	0,9	0,01	1048316,5
15	380, 326, 417	1000	0,8	0,9	0,01	1187038,5

Dari hasil pengujian di atas memperlihatkan nilai kandidat solusi dan intensitas cahaya yang berbeda-beda setiap kali algoritma dijalankan. Permasalahan ini disebabkan hampir seluruh proses dalam algoritma *firefly* mengandalkan teknik acak. Adapun Hasil pengujian mengubah nilai parameter

alpha, basebeta dan gamma menunjukan solusi dengan dihasilkan keluaran yang berbeda-beda atau tidak menemukan solusi unik namun intensitas cahaya yang terbaik ditunjukan oleh nilai intensitas cahaya paling tinggi.

4.3. Kesimpulan Pengujian

Berdasarkan hasil pengujian yang telah dilakukan dapat ditarik kesimpulan bahwa sistem bebas dari kesalahan sintaks dan secara fungsional mengeluarkan hasil yang sesuai dengan yang diharapkan.

Jumlah kandidat solusi acak yang membangkitkan kombinasi jumlah roti sangat berpengaruh dalam menghasilkan solusi optimal. Semakin banyak jumlah kandidat solusi maka semakin besar kemungkinan mendapatkan hasil optimal dan semakin banyak iterasi yang dilakukan maka waktu pencarian akan semakin lama, serta solusi kombinasi jumlah roti cenderung berbeda-beda setiap kali algoritma dijalankan yang disebabkan karena hampir seluruh proses dalam algoritma *firefly* mengandalkan teknik acak. Sehingga pada kasus terburuk, performasi algoritma firefly lebih lama dibandingkan dengan algoritma optimasi lain, karena terdapat kemungkinan terjebak dalam *local optimum*, yaitu solusi-solusi parsial yang dihasilkan dalam setiap iterasi algoritma firefly.