BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Implementasi merupakan tahap penciptaan perangkat lunak dan juga tahap kelanjutan dari perancangan aplikasi. Selanjutnya dilakukan uji kelayakan terhadap aplikasi yang telah dibangun sehingga dapat memberikan masukan demi berkembangnya aplikasi yang telah dibangun. Tahap ini merupakan tahap dimana aplikasi siap untuk dioperasikan, tahap ini terdiri dari penjelasan mengenai lingkungan implementasi, baik itu lingkungan perangkat keras, lingkungan perangkat lunak, serta implementasi aplikasi.

4.1.1 Implementasi Perangkat Keras

Perangkat keras yang digunakan untuk menjalankan permainan menjaring ikan dapat dilihat pada tabel 4.1.

Tabel 4.1 Tabel Spefisikasi Perangkat Keras

No	Perangkat Keras	Keterangan
1	*Processor*	Kecepatan @ 1.83GHz (2 CPUs)
2	*Memory* (RAM)	1536MB
3	*Display Memory*	256 MB
4	*Display Mode*	1280 x 800 (32 bit) (60Hz)
5	Keyboard	Standard
6	Mouse	Standard

4.1.2 Implementasi Perangkat Lunak

Adapun spesifikasi perangkat lunak yang digunakan untuk menjalankan permainan menjaring ikan dapat dilihat pada tabel 4.2.

Tabel 4.2 Tabel Spefisikasi Perangkat Lunak

No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	Windows 7
2	*Tools Compiler*	Adobe Flash CS4

4.1.3 Implementasi Aplikasi

Aplikasi permainan menjaring ikan merupakan sebuah simulasi dari perilaku ikan berkelompok yang merupakan implementasi dari algoritma *boids* dengan tiga aturan sederhananya yaitu *alignment*, *separation* dan *cohesion*, selain itu diterapkan juga perilaku *obstacle avoidance* untuk ikan agar dapat bergerak menghindari jaring. Permainan menjaring ikan ini dapat dijalankan pada sistem operasi windows.

4.1.4 Implementasi Antarmuka

Antarmuka merupakan sarana interakasi antara pengguna dengan sistem operasi. Berikut ini merupakan antarmuka yang ada pada permainan menjaring ikan:

1. Antarmuka Menu Utama

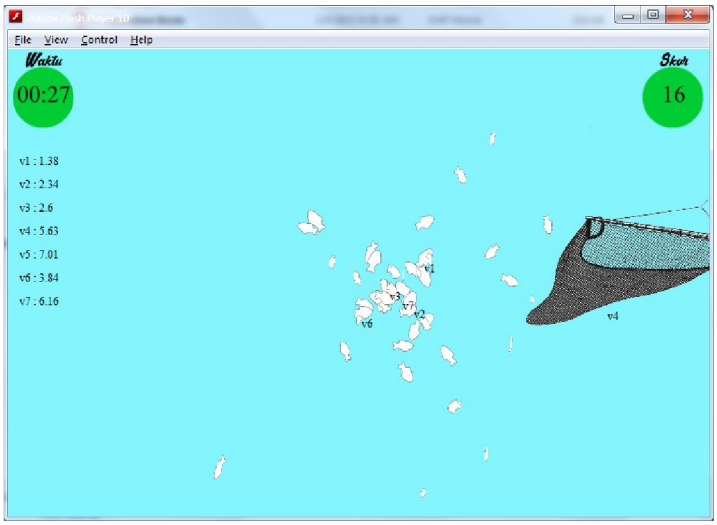
Antarmuka menu utama merupakan tampilan awal ketika aplikasi dijalankan. Tampilan antarmuka menu utama dapat dilihat pada gambar 4.1.



Gambar 4.1 Antarmuka Menu Utama

2. Antarmuka Mulai Simulasi

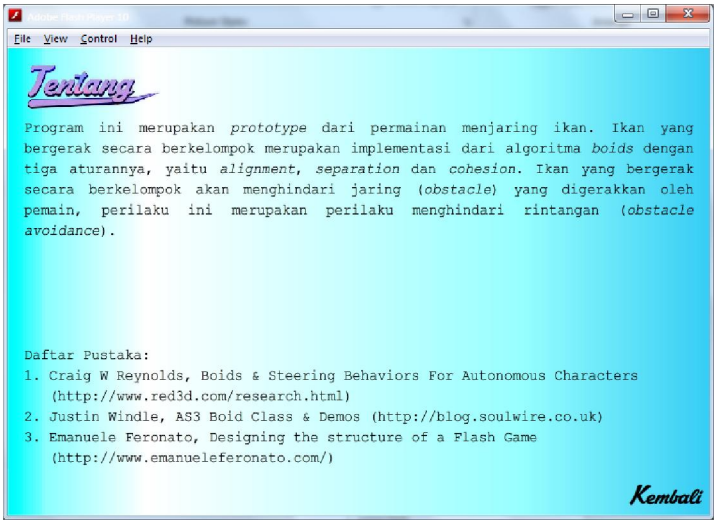
Antarmuka mulai simulasi merupakan antarmuka untuk menampilkan simulasi dari algoritma *boids*. Tampilan antarmuka mulai permainan dapat dilihat pada gambar 4.2.



Gambar 4.2 Antarmuka Mulai Simulasi

3. Antarmuka Tentang

Antarmuka Tentang merupakan tampilan tentang pembuat program. Tampilan antarmuka tentang dapat dilihat pada gambar 4.3.



Gambar 4.3 Antarmuka Tentang

4.2 Pengujian

Pengujian bertujuan untuk menemukan kekurangan atau kesalahan pada perangkat lunak, serta untuk mengetahui perangkat lunak yang dibuat sudah memenuhi kriteria yang sesuai dengan perancangan perangkat lunak tersebut. Adapun pengujian yang akan dilakukan adalah pengujian alpha.

4.2.1 Pengujian Alpha

Pengujian alpha merupakan pengujian program yang dilakukan oleh pembuat game ataupun orang - orang yang terlibat didalamnya. pengujian alfa hanya untuk sirkulasi internal dan masalah *(error)* atau ketidak lengkapan yang terdapat dalam game. Proses pengujian terhadap aplikasi akan menggunakan pengujian *black box* dan pengujian *white box*.

4.2.1.1 Skenario Pengujian

Skenario pengujian menjelaskan pengujian terhadap sistem yang ada pada permainan menjaring ikan. Adapun skenario pengujian yang ada pada permainan menjaring ikan dapat dilihat pada tabel 4.3.

Tabel 4.3 Tabel Skenario Pengujian

No	Komponen yang Diuji	Skenario Pengujian	Jenis Pengujian
1	Menu	Tampilkan Menu Mulai Pemainan	*Black Box*
		Tampilkan Menu Tentang	*Black Box*
2	Menu Mulai Permainan	Tampilkan Ikan	*Black Box*
		Tampilkan Jaring	*Black Box*
		Tampilkan Skor	*Black Box*
		Tampilkan Waktu	*Black Box*
3	Menu Tentang	Tampilkan Informasi Aplikasi	*Black Box*
4	*Obstacle Avoidance*	Ikan bergerak menghindari jaring	*Black Box*
5	Algoritma *Boids*	Algoritma *boids* pada permainan menjaring ikan	*White Box*

4.2.1.2 Pengujian *Black Box*

Berikut adalah hasil pengujian perangkat lunak menggunakan metode *black box* sesuai dengan skenario pengujian yang telah dirancang sebelumnya dapat dilihat pada tabel 4.4.

Tabel 4.4 Tabel Pengujian *Black Box*

No	Komponen yang Diuji	Skenario Pengujian	Hasil yang Diharapkan	Hasil Pengujian
1	Menu	Tampilkan Menu Mulai Pemainan	Menampilkan Menu Mulai Permainan	[√] Berhasil [ ] Tidak Berhasil
		Tampilkan Menu Tentang	Menampilkan Menu Tentang	[√] Berhasil [ ] Tidak Berhasil
2	Mulai Permainan	Tampilkan Ikan	Menampilkan Ikan	[√] Berhasil [ ] Tidak Berhasil
		Tampilkan Jaring	Menampilkan Jaring	[√] Berhasil [ ] Tidak Berhasil
		Tampilkan Skor	Menampilkan Skor	[√] Berhasil [ ] Tidak Berhasil
		Tampilkan Waktu	Menampilkan Waktu	[√] Berhasil [ ] Tidak Berhasil
3	Menu Tentang	Tampilkan Informasi Aplikasi	Menampilkan Informasi Aplikasi	[√] Berhasil [ ] Tidak Berhasil
4	*Obstacle Avoidance*	Ikan bergerak menghindari jaring	Ikan bergerak menghindari jaring	[√] Berhasil [ ] Tidak Berhasil

4.2.1.3 Pengujian *White Box*

Pengujian *white box* merupakan metode perancangan *test case* yang menggunakan struktur kontrol dari perancangan metode dalam mendapatkan *test case*. Adapun metode yang digunakan dalam pengujian *white box* ini adalah metode *basis path*. Metode *basis path* mengijinkan pendesain kasus uji untuk mendapatkan perkiraan lojik yang kompleks dari desain metode dan menggunakan perkiraan ini untuk mendefinisikan aliran eksekusi.

1. Pengujian Aturan *Alignment*

Berikut adalah pengujian proses aturan *alignment* menggunakan pengujian *white box*.

a. *Pseudocode* Aturan *Alignment*

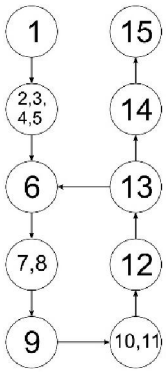
Berikut adalah *pseudocode* dari aturan *alignment* :

```
    public function align( boids : Vector.<Boid> ) : void
1   { _steeringForce = getAlignment(boids, neighborDistance); }
    private function getAlignment( boids : Vector.<Boid> ) : Vector3D
2           { var force : Vector3D = new Vector3D();
3             var distance : Number;
4             var count : int = 0;
5             var boid : Boid;
6           for (var i : int = 0;i < boids.length; i++){
```

```
7               boid = boids[i];
8               distance        =        Vector3D.distance(_position,
    boid.position);
9              if ( distance > 0 && distance < neighborDistance ){
10              force.incrementBy(boid.velocity);
11              count++;
12             }
13            }
14            return force;
15            }
```

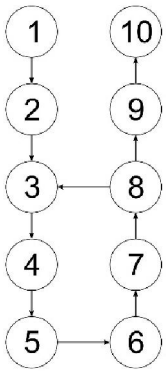
b. *Flow Graph*

Berdasarkan *pseudocode* diatas maka *flow graph* dari aturan *alignment* dapat dilihat pada gambar 4.4.



Gambar 4.4 *Flow Graph* Aturan *Alignment*

atau disederhanakan berdasarkan simpul dapat dilihat pada gambar 4.5.



Gambar 4.5 Penyederhanaan *Flow Graph* Aturan *Alignment*

Keterangan :

= *Nodes* (N)

= *Edges* (E)

Berdasarkan gambar 4.5 maka dapat dihitung *cyclomatic complexity* sebagai berikut :

E = 10, N = 10

V(G) = (E – N) + 2

V(G) = (10 – 10) + 2

V(G) = 2

Maka *cyclomatic complexity* yang didapatkan adalah 2. Berdasarkan *cyclomatic complexity* tersebut, maka terdapat 2 *path* yang terdiri dari :

Path 1 : 1-2-3-4-5-6-7-8-9-10

Path 2 : 1-2-3-4-5-6-7-8-3-4-5-6-7-8-9-10

c. *Graph Matrix*

Berikut adalah *graph matrix* yang menyatakan hubungan antara *nodes* dan *edges* pada gambar 4.5. Kolom *Sum*-1 yang merupakan nilai dihitung berdasarkan jumlah node yang ada dikurang 1. Jika node dari busur selanjutnya tidak ada, maka dinyatakan sebagai elemen kosong dan bernilai nol.

Tabel 4.5 Tabel *Graph Matrix Alignment Rule*

N	1	2	3	4	5	6	7	8	9	10	*Sum*-1
1		1									0
2			1								0
3				1							0
4					1						0
5						1					0
6							1				0
7								1			0
8			1						1		1
9										1	0
10											0
Jumlah (*Sum*-1) + 1											2

Kesimpulan :

Dari hasil pengujian *white box* dapat dilihat bahwa jumlah *edges* (E) pada *graph matrix* sama dengan jumlah *cyclomatic complexity* yaitu 2, maka dapat disimpulkan bahwa proses aturan *alignment* berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

2. Pengujian Aturan *Separation*

Berikut adalah pengujian proses aturan *separation* menggunakan pengujian *white box*.

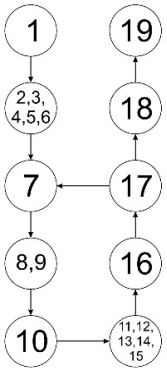
a. *Pseudocode* Aturan *Separation*

Berikut adalah *pseudocode* dari aturan *separation* :

```
    public function separate( boids : Vector.<Boid> ) : void
1   { _steeringForce = getSeparation(boids, separationDistance); }
    private function getSeparation( boids : Vector.<Boid> ) : Vector3D
2           { var force : Vector3D = new Vector3D();
3             var difference : Vector3D;
4             var distance : Number;
5             var count : int = 0;
6             var boid : Boid;
7           for (var i : int = 0;i < boids.length; i++){
8             boid = boids[i];
9             distance        =        Vector3D.distance(_position,
    boid.position);
10           if ( distance > 0 && distance < neighborDistance ){
11            difference = _position.subtract(boid.position);
12            difference.normalize();
13            difference.scaleBy(1 / distance);
14            force.incrementBy(boid.velocity);
15            count++;
16           }
17          }
18          return force;
19          }
```

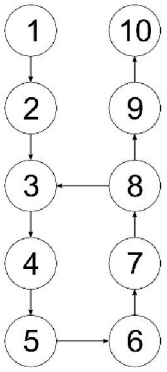
b. *Flow Graph*

Berdasarkan *pseudocode* diatas maka *flow graph* dari aturan *separation* dapat dilihat pada gambar 4.6.



Gambar 4.6 *Flow Graph* Aturan *Separation*

atau disederhanakan berdasarkan simpul dapat dilihat pada gambar 4.7.



Gambar 4.7 Penyederhanaan *Flow Graph* Aturan *Separation*

Keterangan :

= *Nodes* (N)

= *Edges* (E)

Berdasarkan gambar 4.7 maka dapat dihitung *cyclomatic complexity* sebagai berikut :

E = 10, N = 10

V(G) = (E – N) + 2

V(G) = (10 – 10) + 2

V(G) = 2

Maka *cyclomatic complexity* yang didapatkan adalah 2. Berdasarkan *cyclomatic complexity* tersebut, maka terdapat 2 *path* yang terdiri dari :

Path 1 : 1-2-3-4-5-6-7-8-9-10

Path 2 : 1-2-3-4-5-6-7-8-3-4-5-6-7-8-9-10

c. *Graph Matrix*

Berikut adalah *graph matrix* yang menyatakan hubungan antara *nodes* dan *edges* pada gambar 4.8. Kolom *Sum*-1 yang merupakan nilai dihitung berdasarkan jumlah node yang ada dikurang 1. Jika node dari busur selanjutnya tidak ada, maka dinyatakan sebagai elemen kosong dan bernilai nol.

Tabel 4.6 Tabel *Graph Matrix Separation Rule*

N	1	2	3	4	5	6	7	8	9	10	*Sum*-1
1		1									0
2			1								0
3				1							0
4					1						0
5						1					0
6							1				0
7								1			0
8			1						1		1
9										1	0
10											0
Jumlah (*Sum*-1) + 1											2

Kesimpulan :

Dari hasil pengujian *white box* dapat dilihat bahwa jumlah *edges* (E) pada *graph matrix* sama dengan jumlah *cyclomatic complexity* yaitu 2, maka dapat disimpulkan bahwa proses aturan *separation* berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

3. Pengujian Aturan *Cohesion*

Berikut adalah pengujian proses aturan *cohesion* menggunakan pengujian *white box*.

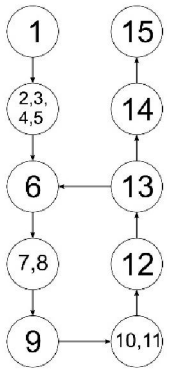
a. *Pseudocode* Aturan *Cohesion*

Berikut adalah *pseudocode* dari aturan *cohesion* :

```
    public function cohesion( boids : Vector.<Boid> ) : void
1   { _steeringForce = getCohesion(boids, neighborDistance); }
    private function getCohesion( boids : Vector.<Boid> ) : Vector3D
2           { var force : Vector3D = new Vector3D();
3             var distance : Number;
4             var count : int = 0;
5             var boid : Boid;
6           for (var i : int = 0;i < boids.length; i++){
7             boid = boids[i];
8             distance        =        Vector3D.distance(_position,
    boid.position);
9            if ( distance > 0 && distance < neighborDistance ){
10            force.incrementBy(boid.velocity);
11            count++;
12           }
13          }
14          return force;
15          }
```

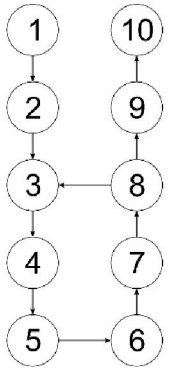
b. *Flow Graph*

Berdasarkan *pseudocode* diatas maka *flow graph* dari aturan *cohesion* dapat dilihat pada gambar 4.8.



Gambar 4.8 *Flow Graph* Aturan *Cohesion*

atau disederhanakan berdasarkan simpul dapat dilihat pada gambar 4.9.



Gambar 4.9 Penyederhanaan *Flow Graph* Aturan *Cohesion*

Keterangan :

= *Nodes* (N)

= *Edges* (E)

Berdasarkan gambar 4.9 maka dapat dihitung *cyclomatic complexity* sebagai berikut :

E = 10, N = 10

V(G) = (E – N) + 2

V(G) = (10 – 10) + 2

V(G) = 2

Maka *cyclomatic complexity* yang didapatkan adalah 2. Berdasarkan *cyclomatic complexity* tersebut, maka terdapat 2 *path* yang terdiri dari :

Path 1 : 1-2-3-4-5-6-7-8-9-10

Path 2 : 1-2-3-4-5-6-7-8-3-4-5-6-7-8-9-10

c. *Graph Matrix*

Berikut adalah *graph matrix* yang menyatakan hubungan antara *nodes* dan *edges* pada gambar 4.9. Kolom *Sum*-1 yang merupakan nilai dihitung berdasarkan jumlah node yang ada dikurang 1. Jika node dari busur selanjutnya tidak ada, maka dinyatakan sebagai elemen kosong dan bernilai nol.

Tabel 4.7 Tabel *Graph Matrix Cohesion Rule*

N	1	2	3	4	5	6	7	8	9	10	*Sum*-1
1		1									0
2			1								0
3				1							0
4					1						0
5						1					0
6							1				0
7								1			0
8			1						1		1
9										1	0
10											0
Jumlah (*Sum*-1) + 1											2

Kesimpulan :

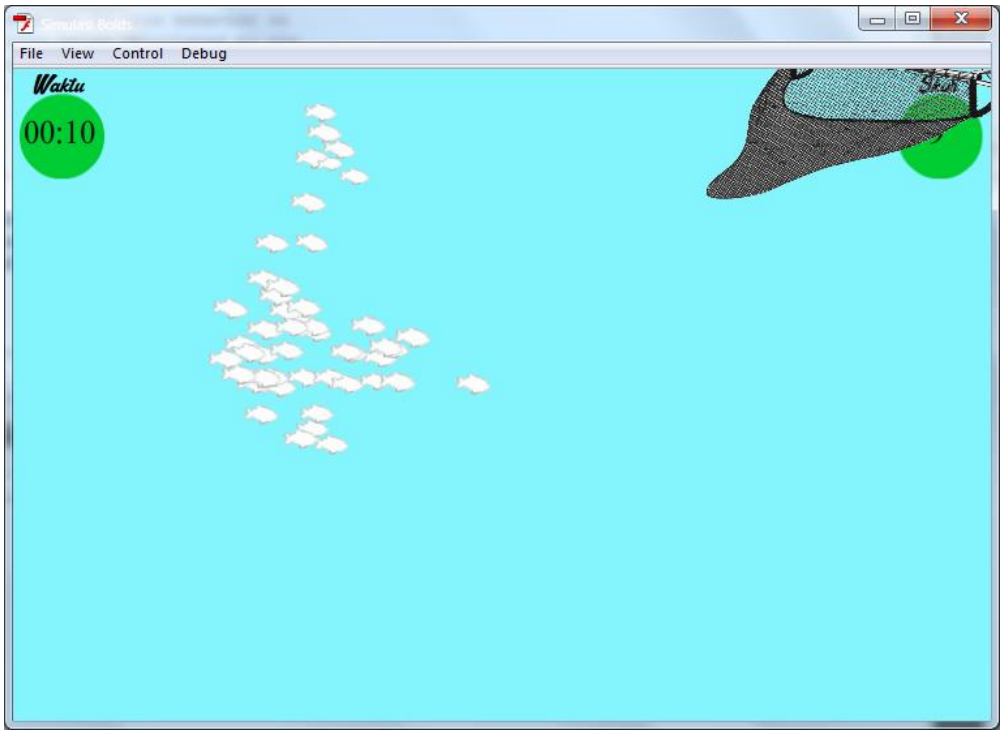
Dari hasil pengujian *white box* dapat dilihat bahwa jumlah *edges* (E) pada *graph matrix* sama dengan jumlah *cyclomatic complexity* yaitu 2, maka dapat disimpulkan bahwa proses aturan *cohesion* berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

4.2.2 Pengujian Algoritma *Boids* pada Perilaku Ikan

Pengujian dilakukan untuk membuktikan bahwa algoritma *boids* telah bekerja sesuai rancangan. Pengujian ini dilakukan dengan menguji tiga aturan dari algoritma *boids* yaitu aturan *alignment*, aturan *separation* dan aturan *cohesion*.

1. Pengujian Aturan *Alignment*

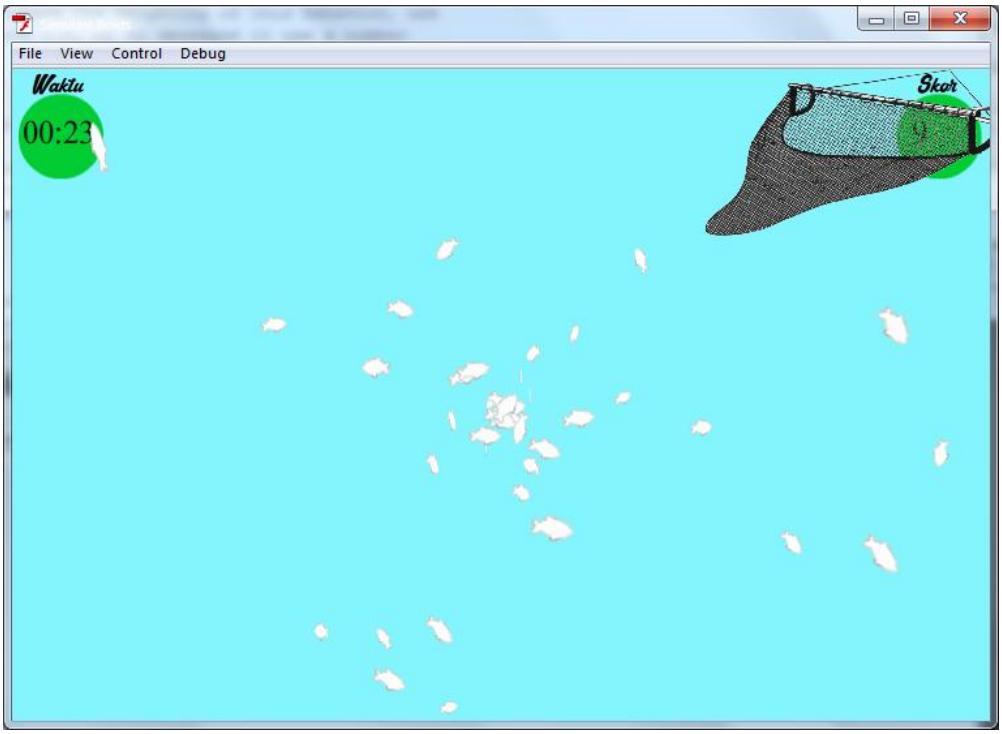
Pengujian ini akan membuktikan bahwa aturan *alignment* dari algoritma *boids* yang merupakan kemampuan untuk mensejajarkan diri dan menuju ke arah yang sama bekerja sesuai rancangan. Jika aturan *alignment* mempunyai nilai yang lebih besar dari aturan *separation* dan *cohesion*, maka kemampuan ikan untuk mensejajarkan diri akan lebih besar. Berikut pengujian dari aturan *alignment* dapat dilihat pada gambar 4.10.



Gambar 4.10 Pengujian Aturan *Alignment*

2. Pengujian Aturan *Separation*

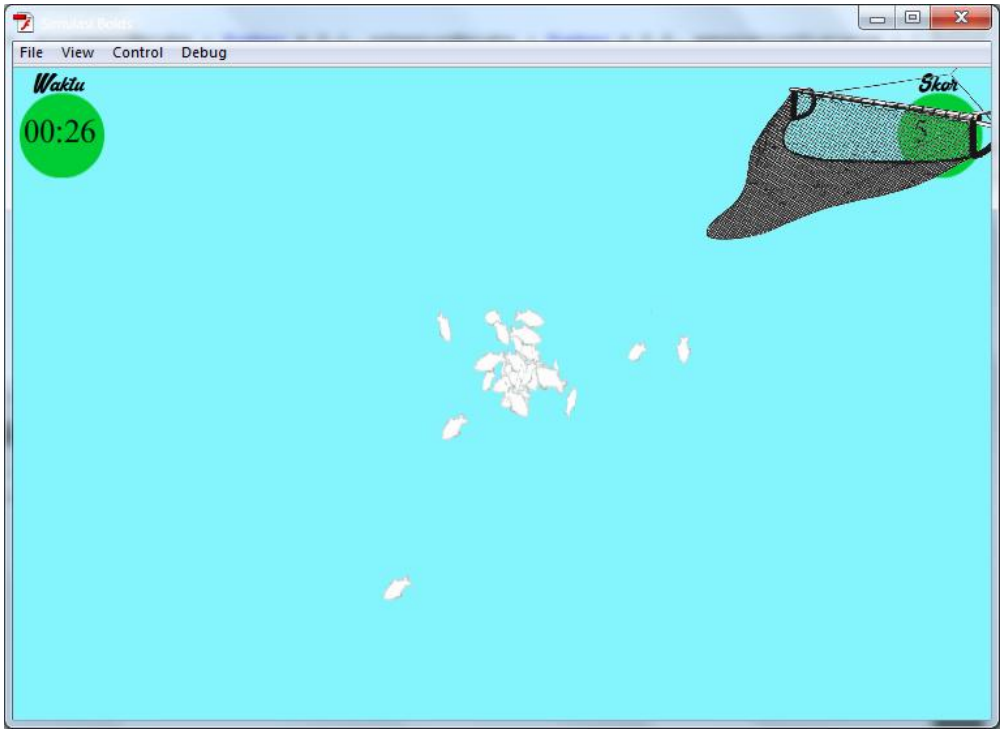
Pengujian ini akan membuktikan bahwa aturan *separation* dari algoritma *boids* yang merupakan kemampuan untuk mempertahankan atau memelihara jarak bekerja sesuai rancangan. Jika aturan *separation* mempunyai nilai yang lebih besar dari aturan *alignment* dan *cohesion*, maka kemampuan ikan untuk mempertahankan atau memelihara jarak akan lebih besar. Berikut pengujian dari aturan *separation* dapat dilihat pada gambar 4.11.



Gambar 4.11 Pengujian Aturan *Separation*

3. Pengujian Aturan *Cohesion*

Pengujian ini akan membuktikan bahwa aturan *cohesion* dari algoritma *boids* yang merupakan kemampuan untuk bergerak menuju posisi rata-rata kawanan terdekatnya bekerja sesuai rancangan. Jika aturan *cohesion* mempunyai nilai yang lebih besar dari aturan *alignment* dan *separation*, maka kemampuan ikan untuk bergerak menuju posisi rata-rata kawanannya akan lebih besar. Berikut pengujian dari aturan *cohesion* dapat dilihat pada gambar 4.12.



Gambar 4.12 Pengujian Aturan *Cohesion*

Kesimpulan :

Dari hasil pengujian algoritma *boids* pada perilaku ikan, dapat dilihat bahwa tiga aturan dari algoritma *boids* yaitu aturan *alignment*, aturan *separation* dan aturan *cohesion* dapat bekerja sesuai dengan rancangan.