Tesis – TE - 142599

PERGERAKAN SERANGAN KELOMPOK NPC BERBASIS AGEN OTONOM MENGGUNAKAN ALGORITMA ARTIFICIAL BEE COLONY

Wilujeng Jatiningsih

2213205702

DOSEN PEMBIMBING
DR. Eko Mulyanto Yuniarno, ST, MT
Mochamad Hariadi, ST, M. Sc, Ph. D

PROGRAM MAGISTER

BIDANG KEAHLIAN JARINGAN CERDAS MULTIMEDIA

JURUSAN TEKNIK ELEKTRO

FAKULTAS TEKNOLOGI INDUSTRI

INSITITUT TEKNOLOGI SEPULUH NOPEMBER

SURABAYA

2015

Tesis – TE - 142599

AUTONOMOUS AGENT BASED NPC SWARM ATTACK BEHAVIOUR USING BEE COLONY ALGORITHMN

Wilujeng Jatiningsih

2213205702

SUPERVISOR
DR. Eko Mulyanto Yuniarno, ST, MT
Mochamad Hariadi, ST, M. Sc, Ph. D

MAGISTER PROGRAM

INTELLIGENT NETWORK EXPERTISE MULTIMEDIA

DEPARTMENT OF ELECTRICAL ENGINEERING

FACULTY OF INDUSTRIAL TECHNOLOGY

SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY

SURABAYA

2015

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Teknik (M.T)
di
Institut Teknologi Sepuluh Nopember

Oleh:
Wilujeng Jatiningsih
NRP. 2213205702

Tanggal Ujian : 8 Januari 2015
Periode Wisuda : Maret 2015

Disetujui oleh :

1. DR. Eko Mulyanto Yuniarno, ST, MT (Pembimbing 1)
   NIP : 196806011995121009

2. Mochamad Hariadi, ST, M. Sc, Ph. D (Pembimbing II)
   NIP. 196912091997021002

3. Dr. Supeno Mardi Susiki N, S.T, M.T (Penguji)
   NIP. 19700313199512100l

4. Dr. I Ketut Eddy Purnama, S.T, M.T (Penguji)
   NIP. 196907301995121001

Direktorat Program Pasca Sarjana

Prof. Dr. Ir. Adi Soeprijanto, M.T
NIP. 196404051990021001

PERGERAKAN SERANGAN KELOMPOK NPC BERBASIS AGEN OTONOM MENGGUNAKAN ALGORITMA ARTIFICIAL BEE COLONY

Student Name	:	Wilujeng Jatiningsih
NRP	:	2213205702
Supervisor	:	Dr. Eko Mulyanto Yuniarno, ST, MT
Co-Supervisor	:	Mochamad Hariadi, ST., M.Sc., Ph

ABSTRAK

Game RTS merupakan game yang paling diminati untuk dimainkan. Penelitian ini merupakan penelitian untuk membuat simulasi pergerakan kelompok NPC yang mampu mendeteksi kedatangan musuh kemudian menyerangnya secara berkelompok secara acak dari arah dari arah yang berbeda. Pergerakan serangan berkelompok itu berjalan secara otonom tanpa campur tangan *player*. Pada saat bergerak mendekati lawan, NPC dirancang dapat memilih posisi baru yang membuatnya semakin dekat dengan lawan namun tetap memperhitungkan posisi NPC lain agar tidak terjadi tabrakan dan harus mampu menghindari halangan. Pergerakan serangan berkelompok dilakukan secara acak dan serempak meniru pergerakan lebah saat mencari sumber makanan. Algoritma ABC dipilih dalam penelitian ini karena mempunyai self-organization yang bagus dan memiliki pembagian tugas yang jelas.

Percobaan yang dilakukan telah berhasil membuktikan bahwa algoritma ABC mampu membuat simulasi pergerakan sekelompok agen otonom yang dapat mengetahui posisi lawan kemudian bergerak mendatangi lawan dengan rute atau formasi acak dengan tujuan konvergen atau berkerumun di sekitar posisi target/lawan. Agen cenderung telah konvergen di sekitar iterasi ke-30.

Kata Kunci: agen otonom, NPC, *self-organization, artificial bee colony algorithm.*

AUTONOMOUS AGEN BASED NPC SWARM ATTACK BEHAVIOUR USING BEE COLONY ALGORITHM

Student Name	: Wilujeng Jatiningsih
NRP	: 2213205702
Supervisor	: Dr. Eko Mulyanto Yuniarno, ST, MT
Co-Supervisor	: Mochamad Hariadi, ST., M.Sc., Ph.D

ABSTRACT

RTS games are the most attractive game to be played. This research is to create a simulation of the movement of groups of NPCs that can detect the arrival of the enemy then attack together in groups at random from different directions. The group attack movement runs autonomously without the intervention of the player. While moving closer to the opponent, the NPC is designed to be able to choose a new position which makes it closer to the opponent but still observing the position of the other NPCs in order to avoid collision and also can avoid obstacles. The movement of groups' attacks carried out simultaneously and randomly mimics the movement of bees when looking for food sources. ABC algorithm is chosen in this study because it has a good self-organization and has a clear division of tasks.

This research has proved that the ABC algorithm is able to simulate the movement of the group of autonomous agents that can determine the position of the opponent and then move towards the opponent through random routes and random formations with the goal of converging or clustered around the position of the target/opponent. Agents tend to convergent around the 30th iteration.

Key words: *autonomous agen*, NPC, *self-organization*, *artificial bee colony algorithm.*

KATA PENGANTAR

Alhamdulillah, segala puji bagi Allah Tuhan semesta alam. Berkat rahmat Allah SWT, penulis dapat menyelesaikan tesis ini dengan baik. Tesis dengan judul "PERGERAKAN SERANGAN KELOMPOK NPC BERBASIS AGEN OTONOM MENGGUNAKAN ALGORITMA ARTIFICIAL BEE COLONY" diselesaikan penulis dalam satu semester, yakni pada semester 3 program Pasca Sarjana ini. Tesis ini disusun guna memenuhi persyaratan untuk memperoleh gelar Magister Teknik pada bidang konsentrasi Teknologi Permainan, bidang studi Jaringan Cerdas Multimedia, jurusan Teknik Elektro, Institut Teknologi Sepuluh Nopember Surabaya.

Keterbatasan kemampuan penulis dalam mengerjakan Thesis ini tidak terlalu menghambat penyelesaian penelitian karena begitu banyak perhatian dan bantuan dari rekan-rekan, para dosen, dan kerabat yang dengan iklas meluangkan waktu dan pikirannya untuk membantu penulis. Beberapa pihak yang penulis sebutkan berperan besar dalam penyusunan Thesis ini. Terima kasih penulis ucapkan terutama untuk:

1. Toniku, sebagai suami dan sahabat terbaik, yang selalu memberikan motivasi dan bantuan berupa apapun
2. Radit (sang reviewer CoC), Salma, dan Allysha yang dengan sabar merelakan ibunya menempuh studi ini. Maaf untuk hari-hari tanpa Ibu.
3. Almarhum Bapak Soeroso dan almarhumah Ibu Titiek Sujati untuk setiap amal baiknya untuk orang lain dengan harapan kelak orang lain akan memudahkan urusan anak-anaknya. Dan dalam penelitian ini, harapan Bapak Ibu terpenuhi. Karena penulis sangat dimudahkan urusannya oleh dosen, pembimbim dan teman-teman penulis.
4. SEAMOLEC yang telah memberikan beasiswa untuk menempuh jenjang studi ini
5. Dr. Eko Mulyanto Yuniarno, S.T, MT alias Pak Akok atas bimbingan, fasilitas laboratorium dan motivasi ampuhnya "Gampang kabeh iku"
6. Pak Hariadi, Pak Surya, Pak Uki, Pak Ketut, Prof. Hery yang telah memberikan wawasan serta ilmu baru untuk meningkatkan pengetahuan
7. Teman-teman GamteTech terutama angkatan 2013, 2012 dan 2014 yang telah meluangkan waktu bersama untuk saling berdiskusi, untuk setiap kegilaan yang kita lakukan, untuk setiap kopi yang kita teguk, untuk setiap panik yang kita

nikmati. Karena banyaknya kontribusi dari teman-teman, penulis tidak mampu menguraikan kebaikan mereka di lembar Kata Pengantar ini.

8. Teman-teman STIKOM angkatan 1994 yang telah menyisihkan waktu untuk mengajak makan malam sebagai suatu usaha perbaikan gizi penulis.
9. Teman-teman di Grup WA SMA 1 Tulungagung lulusan 1994 untuk setiap percakapan penuh tawa dan undangan makan juga demi perbaikan gizi penulis.
10. Bupati Tulungagung, Kepala Sekolah SMPN 1 Sumbergempol yang telah memberikan dan mengijinkan tugas belajar
11. Teman-teman di SMP 1 Sumbergempol untuk setiap dukungan dan bantuannya.
12. Mas Gong, Mbak Lis, Mbak Yan, Mbak Nin, Mbak Ti, Mas Dadam, Budhe In, Pakde Karno, Buk Ning, Pak Hari, Buk Na dan Pak Eko untuk doa dan bantuan morilnya.
13. Mak Yun yang dengan sabar mengurus anak-anak penulis selama ini
14. Pak Man sebagai informan status keberadaan para dosen

Penulis sepenuhnya menyadari bahwa hasil karya ini sangatlah jauh dari sempurna. Walaupun penulis menganggapnya sebagai pencapaian yang luar biasa tapi tentulah masih banyak kekurangan yang dapat dikoreksi oleh pihak lain. Kritik, saran, maupun studi lebih lanjut dari topik yang penulis sajikan sangat membuat penulis bahagia.

Surabaya, Januari 2015
Penulis.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BIOGRAFI

Wilujeng Jatiningsih atau biasa dipanggil dengan Lulut lahir di bulan Mei 1976. Istri dari Toni Widianto, ibu dari 3 anak hebat yaitu Raditya Muhammad Salman, Audrey Salsabila Salma dan Allysha Tabina Aafia. Pendidikan TK sampai SMA ditempuh di Tulungagung. Kemudian meneruskan pendidikan S1 dan S2 di Surabaya. Saat ini menikmati profesi sebagai pendidik di SMP Negeri 1 Sumbergempol, Tulungagung. Menyukai coklat, brownies, dan pizza. Diwujudkannya kesukaannya itu dalam bentuk usaha kuliner. Tetapi yang tetap aktif adalah usaha pizzanya. Binatang peliharaan yang disukai adalah kucing. Sejak kecil sudah memelihara kucing hingga sekarang. Pergaulannya dengan kucing juga menepis kekuatiran banyak orang bahwa kucing adalah pembawa toksoplasma yang berbahaya bagi kehamilan

lulut_smart@yahoo.com

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Real Time Strategy (RTS) merupakan game perang ala militer. Kelompok-kelompok agen otonom atau biasa disebut juga *Non-Playable Character* (NPC) yang ada dalam game ini membangun kekuatan, mengatur pasukan tempur, mengendalikan unit untuk menyerang musuh dan mengumpulkan sumber daya untuk memperkaya kelompoknya. Serangan NPC pada game RTS dapat dilakukan secara individu maupun berkelompok.

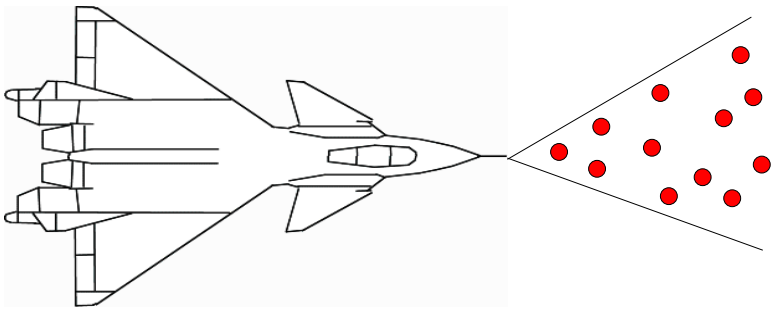
Salah satu contoh game RTS adalah *Clash of Clans* (CoC) yang dibuat dan disebarkan oleh Supercell. Contoh NPC yang melakukan serangan individu adalah *Archer* dan *Wizard*. *Archer* menyerang target dengan menggunakan panah. *Wizard* menembak target dengan bola api/cahaya. Sedangkan contoh NPC yang melakukan serangan berkelompok adalah *Barbarian, Giant, Ballon, Dragon*. Gambar 1.1 adalah salah satu contoh serangan sekelompok Dragon pada game CoC.

Gambar 1.1 Serangan Sekelompok Dragon pada CoC

Pergerakan kelompok atau *swarming* pada game sebenarnya meniru pergerakan dari kelompok hewan seperti kawanan ikan (*school of fish*), kawanan

burung (*flock of birds*), kawanan domba (*herd of sheeps*) dan serangga sosial seperti semut (*ant colony*) dan lebah (*bee colony*). Para agen dalam kelompok saling bertukar informasi dan tugas agar tujuan kelompok tercapai, hal ini menunjukkan bahwa kelompok itu mempunyai *self oganization*. *Swarm intelligence* adalah sekelompok agen sederhana yang memiliki *self organization* [1].

Dalam sebuah *game*, *swarm intelligence* digunakan untuk mengatur pergerakan sekelompok agen. Beberapa penelitian tentang penerapan *swarm intelligence* dalam *game* telah dilakukan. Ryan E. Leigh dan Tony Morelli meneliti penggunaan GA untuk menentukan strategi menyerang predator secara berkelompok [2]. Tahun 2013 Ruby L V Moritz dan Martin Middendorf meneliti pembagian tugas dalam kelompok untuk mencapai *goal* sesuai keahlian agen dengan menggunakan *Coalition Formation* [3]. Jiann-Hong Lin dan Li-Ren Huang pada tahun 2009 meneliti penggunaan Chaotic Bee Sarm Oprimization untuk menentukan jarak terpendek ke target [4]. Widi Sarinastiti pada tahun 2014 meneliti penggunaan algoritma Boid untuk mengatur jarak antar agen dan arah gerakan agen saat mengikuti gerakan *leader* [5] dan Alun Sujada pada tahun 2011 juga menggunakan algoritma Boid untuk menentukan formasi perang, penyerang dan bertahan [6]. Preetha Bhattacharje dan timnya meneliti pergerakan sekelompok robot mencari jarak terpendek menuju target dengan menggunakan algoritma ABC [8].



Gambar 1.2 Formasi Serangan *Convergent Approach*

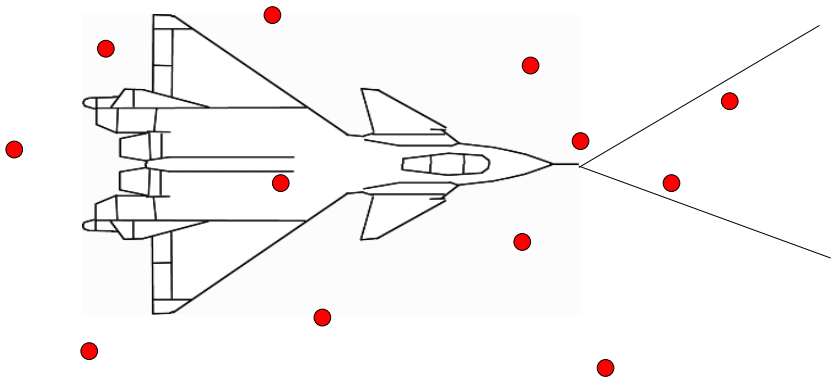
Penerapan *swarm intelligence* pada *game* biasanya digunakan untuk pergerakan sekelompok agen yang pergerakannya dapat ditebak, cenderung bergerombol menyerang musuh dari arah yang sama (*covergent approach*). Hal ini membuat pasukan agen itu mudah dilumpuhkan atau ditembak oleh lawan sehingga membuat *game* menjadi membosankan dan kurang menarik.

Gambar 1.2 adalah formasi menyerang *Convergent approach.* Dapat dilihat bahwa ketika agen menyerang musuh dari arah yang sama, maka kemungkinan agen

masuk zona pandang musuh lebih besar, sehingga lebih mudah ditembak [9]. Gambar 1.3 adalah salah satu contoh serangan Balons pada game CoC yang juga dilakukan mengelompok dari arah yang sama dan musuh dapat dengan mudah menembak Balons karena masuk dalam zona pandang musuh.

Gambar 1.3 Serangan *Convergent* Ballon pada CoC

Agar serangan kelompok agen tidak mudah ditembak oleh musuh, agen harus menyerang dari segala penjuru (*divergent approach*) dengan gerakan acak yang sulit untuk diprediksi musuh. Gambar 1.4 adalah salah satu contoh formasi menyerang musuh secara berkelompok tetapi dari arah yang berbeda-beda.



Gambar 1.4 Formasi Serangan *Divergent Approach*

Selain menyerang secara berkelompok dari arah yang sama, pada umumnya untuk menyerang musuh agen-agen itu harus di-*deploy* secara manual ke lokasi yang ditentukan oleh *player*. Sehingga jika ada serangan dari lawan dan *player* tidak men-

deploy NPC-nya maka lawan akan dapan dengan mudah memenangkan permainan. Hal ini juga membuat permainan kurang menarik.

Ryan E. Leigh dan Tony Morelli berhasil membuat agen yang mampu menyerang secara berkelompok tetapi dia mengakui bahwa agennya belum layak untuk disebut cerdas karena belum memiliki *self-organization* [2]. Penelitian pembagian tugas (*self organization*) dalam kelompok oleh Ruby L V Moritz dan Martin masih bergerak di koordinat x dan y [3]. Karena Jiann-Hong Lin dan Li-Ren Huang memfokuskan penggunakan Chaotic Bee Swarm Oprimization untuk menentukan rute terpendek, maka meskipun pada dunia nyata gerakan lebah termasuk bersifat *non-linear* penelitian ini belum digunakan untuk menciptakan simulasi gerakan acak sekelompok agen [4]. Sedangkan penelitian Widi Sarinastiti dan Alun Sujada menghasilkan simulasi pergerakan agen yang cenderung menggerombol pada satu titik [5, 6], hal ini perlu dihindari agar agen tidak menjadi sasaran empuk lawan. Penelitian Preetha Bhattacharje menciptakan pergerakan agen yang tidak acak meskipun menggunakan algoritma ABC dan agen dalam penelitian ini bergerak di bidang 2 dimensi [8].

Untuk menciptakan gerakan agen yang acak dapat mengadaptasi perilaku hewan sosial. Salah satu koloni hewan sosial yang diadaptasi ke dalam *swarm intelligence* adalah lebah, semut dan rayap. Dari ketiga hewan sosial itu, semut dan rayap bergerak mengikuti pola yang telah dibentuk oleh jejak feromon yang ditinggalkan semut/rayap sebelumnya. Pergerakan acak hanya dilakukan ketika mencari sumber makanan. Setelah sumber makan ditemukan, maka semut/rayap akan meninggalkan jejak feromon agar diikuti oleh semut/rayap lain. Sehingga, pergerakan semut dan rayap tidak murni *non linear*. Sedangkan lebah, mulai dari awal pencarian makanan sampai proses eksploitasi makanan tetap melakukan gerakan acak.

Penelitian ini menggunakan algoritma Artificial Bee Colony yang mengadaptasi perilaku lebah saat mencari makanan. Algoritma ABC terbukti memiliki *self organization* karena memenuhi 4 karakteristik yang ditentukan oleh Bonabeau [1] yaitu : *positive feedback* adalah saat *onlooker* memilih *foodsource*, *negative feedback* adalah saat *employed* mengesploitasi *foodsource* terpilih, *fluctuation* adalah saat *scout* melakukan pencarian *foodsource* baru secara acak dan *multiple interactioan* adalah saat *employed* selalu membagi informasi tentang *foodsource* di area dansa (*dance area*).

1.2. Rumusan Masalah

Gerakan sekelompok NPC yang menyerang suatu lawan dari arah yang sama membuat NPC lebih mudah dikalahkan/ditembak oleh lawan tersebut, karena mereka masuk ke dalam zona pandang lawan.

Selain itu, pada umumnya untuk melakukan serangan pada lawan NPC harus dikirim (deploy) secara manual oleh player, sehingga jika player tidak mengirimkan NPC untuk melakukan perlawanan maka lawan akan dengan mudah menang.

Dalam sebuah *game*, keberadaan NPC yang mudah dikalahkan membuat *game* tersebut menjadi kurang menarik.

1.3. Batasan Masalah

1. Algoritma ABC digunakan untuk menentukan rute sekelompok NPC bergerak bersama dari posisi awal menuju target secara acak untuk mengerumuni target dari arah yang berbeda.
2. Kelompok NPC bergerak dalam bidang 3 dimensi
3. Target statis
4. Obstacle statis

1.4. Tujuan Penelitian

Diperoleh sekelompok NPC cerdas yang mampu mengetahui posisi lawan, kemudian secara otonom bergerak menuju target dengan formasi acak dari arah yang berbeda-beda tanpa menabrak agen lain dalam kelompoknya dan mampu menghindari obstacle statis dengan menggunakan algoritma ABC.

1.5. Manfaat Penelitian

Peneliti berharap penelitian ini memberikan manfaat sebagai berikut :

1. Terbentuknya alternatif serangan kelompok NPC pada game RTS yang mampu menyerang secara berkelompok dari arah yang berbeda.
2. Dapat memberikan kontribusi pada perkembangan teknologi permainan terutama pada *Swarm Intelligence* NPC

BAB 2
DASAR TEORI

Sebuah penelitian selalu membutuhkan dasar teori yang digunakan sebagai landasan penelitiannya. Sesuai dengan judul penelitian yaitu "Pergerakan Serangan Berkelompok NPC Berbasis Agen Otonom Menggunakan Algoritma Artificial Bee Colony", maka pada Bab 2 ini akan dibagi menjadi 4 sub bab. Sub bab 1 akan membahas serangan berkelompok pada game, sub bab 2 akan membahas tentang NPC berbasis agen otonom, sub bab 3 akan membahas algoritma Artificial Bee Colony dan sub bab 4 akan mereview game CoC yang digunakan sebagai acuan penelitian.

2.1. Serangan Berkelompok pada Game

"Menyerang adalah bentuk pertahanan terbaik" adalah jargon yang sering digunakan untuk menentukan strategi dalam bermain sepak bola. Hal ini juga berlaku dalam *game*. Ketika musuh memasuki wilayah kita, cara bertahan terbaik adalah menyerang musuh itu sampai musuh mati atau keluar dari wilayah kekuasaan kita.

Menyerang atau *attack* dalam bahasa Inggris, menurut en.wiktionary.org adalah sebuah usaha untuk menciptakan kerusakan atau cedera secara fisik, atau sebuah usaha yang dilakukan untuk mengurangi kredibilitas seseorang, posisi, ide, obyek, atau hal, dengan fisik, verbal, emosional, atau cara lainnya. Sedangkan dalam *game*, menyerang dapat diartikan sebagai usaha untuk menciptakan kerusakan pada pihak lawan dengan tujuan mendapatkan point.

Ditinjau dari jumlah agen saat melakukan serangan, serangan dalam game dibedakan menjadi 2, yaitu serangan tunggal dan serangan dalam kelompok. Yang dimaksud dengan serangan tunggal adalah agen mendatangi dan menyerang lawan sendiri. Contoh serangan tunggal dilakukan The Prince dalam game Prince of Persia saat melakukan *melee attack* atau serangan jarak dekat dengan menggunakan senjata pedang pada lawannya. Gambar 2.1 adalah contoh serangan tunggal yang dilakukan oleh The Prince dalam *game* Prince of Caribian.

Sedangkan serangan berkelompok adalah serangan yang dilakukan oleh sekelompok agen saat melawan musuh. Serangan berkelompok ini dapat dilakukan di darat, laut maupun udara. Pada CoC *troop* yang dapat melakukan serangan udara yang

dilakukan secara berkelompok antara lain adalah Dragon, Healer dan Ballon. Gambar 1.1 pada Bab 1 adalah salah satu contoh serangan sekelompok Dragon pada game CoC.

Gambar 2.1 Serangan Tunggal oleh The Prince dalam Game Prince of Caribean (http://www.myplaystationwallpapers.net/)

Diperlukan kecerdasan buatan yang ditanamkan pada agen agar terbentuk sekelompok agen yang dapat menyerang bersama-sama. Berkelompok atau *Swarming* pada dasarnya meniru perilaku hewan sosial, seperti kawanan ikan, burung dan serangga misalnya semut, rayap dan lebah. Setiap anggota kelompok berperilaku tanpa supervisi dan memiliki perilaku stokastik karena persepsi dirinya terhadap lingkungan sekitarnya. *Swarm intelligence* (SI) adalah perilaku menyebar secara kolektif dan memiliki aturan pengorganisasian sendiri (*self-organized*)[1].

Swarm Intelligence (SI) merupakan cabang dari *Artificial Intelligence* (SI) yang digunakan untuk memodelkan perilaku kolektif dari hewan sosial di alam seperti koloni semut, lebah atau burung. Meskipun setiap agen dalam *swarm* cenderung memiliki kemampuan yang sederhana tetapi mereka saling berkomunikasi dan melakukan pembagian tugas yang jelas untuk kelangsungan hidup mereka. Interaksi sosial antar anggota *swarm* dapat dilakukan secara langsung maupun tidak langsung. Interakasi langsung dilakukan dengan melakukan kontak visual atau suara, seperti pada saat lebah pekerja (*employed bee*) mempresentasikan informasi tentang jumlah *nectar* pada *food source* yang dikunjunginya di lantai dansa (*dance area*) dengan melakukan tarian (*wagle dance*) yang ditujukan pada lebah pengamat (*onlooker bee*). Sedangkan interaksi tidak

langsung terjadi pada saat ada anggota kelompok yang melakukan perubahan pada lingkungannya dan anggota yang lain berperilaku sesuai perubahan lingkungan itu. Contoh interaksi tidak langsung adalah ketika semut berkomunikasi dengan temannya dengan meninggalkan jejak feromon pada jalur yang dia lewati saat menuju sumber makanan, kemudian semut lain akan bergerak mengikuti jejak feromon itu [7].

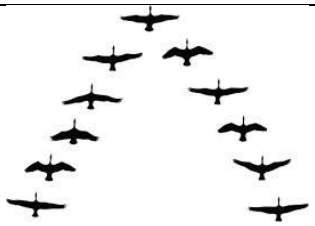
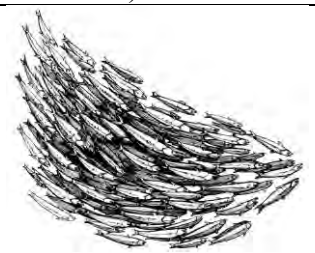
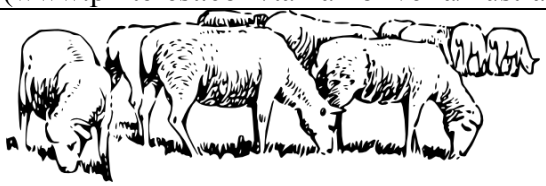
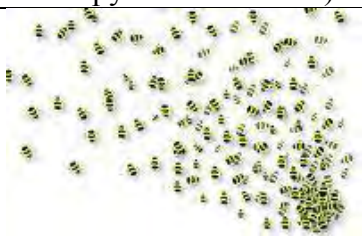
Tidak semua kelompok agen dapat dikatakan memiliki kecerdasan (*intelligent*). Kelompok agen dapat dikatakan cerdas jika memiliki *self-organization* dan pembagian tugas yang jelas. *Self-organization* adalah ciri utama dari sistem *swarm* yang menghasilkan perilaku kolektif yang berasal dari interaksi lokal antar agen (Bonabeau et al. 1999). 4 karakteristik *self-organization* menurut Bonabeau adalah :

1. *Positive feedback* : tiap anggota kelompok merasa nyaman berada di dalam kelompoknya. Sebagai contoh, proses rekruitmen dan penguatan formasi koloni semut saat mengikuti jejak yang telah dibentuk oleh anggota di depannya.
2. *Negative feedback* : sebagai penyeimbang *positive feedback* dan membantu menstabilkan pola formasi koloni. *Negative feedback* diperlukan untuk menghindari kejenuhan yang mungkin terjadi pada saat proses pencarian makanan.
3. *Fluctuations* : berjalan secara acak, melakukan kesalahan, saling bertukar tugas secara acak sesama anggota koloni merupakan kreatifitas yang penting. *Ramdomness* (mengacak) kadang kala diperlukan untuk menemukan solusi baru.
4. *Multiple interactions* : anggota koloni menggunakan informasi yang datang dari anggota koloni lain sehingga informasi menyebar ke seluruh jaringan.

Untuk menyempurnakan 4 karakteristik *self-organization* di atas, Bonabeau menambahkan bahwa pembagian tugas untuk menyelesaikan pekerjaan secara bersama-sama juga merupakan ciri penting sebuah koloni dapat disebut memiliki kecerdasan [1].

Table 2.1 merupakan contoh pergerakan *swarming* burung, ikan, domba, semut dan lebah. Setiap formasi perilaku hewan social di alam seperti pada table 2.1 dapat diadaptasi ke dalam *game* menjadi formasi menyerang secara berkelompok.

Tabel 2.1. Pergerakan Swarming Burung, Ikan, Domba, Semut dan Lebah

No.	Nama	*Swarming*
1	Burung	 (http://www.istockphoto.com/illustrations/birds+flying+in+v-formation)
2	Ikan	 (www.pinterest.com/taniamoliveira/ilustrations/)
3	Domba	 (http://www.freepik.com/free-vector/grazing-sheep-clip-art_381081.htm)
4	Semut	 (http://blog.timesunion.com/marymartin/fresh-new-typing-of-creepy-crawlies/1992/)
5	Lebah	 (http://www.shutterstock.com/s/swarming/search.html)

2.2. Perilaku NPC Berbasis Agen Otonom

NPC atau *Non Playable Character* adalah karakter dalam game yang perilakunya tidak dikontrol oleh manusia/*player.* Perilaku NPC dibagi menjadi 3, yaitu strategis (*strategic*), taktik (*tactical*) dan reaktif (*reactive*). Perilaku stategi digunakan untuk

mencapai tujuan jangka panjang, misalnya mengamankan wilayahnya. Setiap NPC selalu memiliki tujuan jangka panjang dan jangka pendek. Perilaku taktis digunakan untuk mencapai tujuan jangka pendek yang lebih spesifik lagi. Sedangkan perilaku reaktif adalah reaksi sederhana sesuai dengan persepsi audio visualnya pada saat itu, seperti melompat, berjalan, membidik atau menembak [12].

Dalam game strategi (RTS), player harus mengontrol pasukan untuk berperang melawan musuh. Emas, elixir atau tropi dikumpulkan player untuk menguatkan unit, membangun *hall*/barak dan merakit pesawat atau aset lain agar menang saat perang melawan musuh. Pengumpulan emas, *elixir* atau tropi dapat dilakukan dengan menyerang unit lawan atau menambang di tambang emas/*elixir*. Jika pengumpulan emas/*elixir* diperoleh dari hasil mengalahkan lawan, maka *player* harus mengkoordinir pasukan saat melakukan serangan pada lawan.

Ada 2 jenis game strategi, yaitu *turn based* dan *real time*. Pada *turn-base strategy games* player dan lawan secara bergantian mengeluarkan perintah untuk unit mereka, seperti bermain catur. Baik *player* maupun lawan bergantian menjalankan pasukannya. Sedangkan *real time strategy game player* dan komputer mengkoordinir pasukannya secara bersamaan (*real-time*).

Ada 2 jenis NPC dalam game strategy, yaitu *strategic* NPC dan *unit* NPC. Strategic NPC digunakan untuk mengendalikan tentara lawan. Mereka harus bisa mengatur strategi sama seperti yang *player* lakukan. Mereka juga mengumpulkan sumber daya yang ada dalam environment selama game dijalankan. *Stategic* NPC harus dapat melakukan *melee attack* sebagus *player*, tetapi di akhir *game* mereka harus membiarkan *player* yang menang.

Sedangkan *Unit* NPC adalah tentara atau karakter tunggal yang menjadi anggota pasukan *player* maupun *Strategic* NPC. Artinya, unit NPC ini pasukan yang digerakkan oleh *player* maupun *Strategic* NPC. *Unit* NPC harus memiliki kecerdasan agar dapat melaksanakan perintah *player* maupun perintah *Strategic* NPC. Perintah itu dapat berupa perintah menyerang, mengumpulkan sumber daya atau membangun gedung. *Unit* NPC juga harus mampu merencanakan rute jalan dan mengikuti rute itu untuk mencapai target mereka dan untuk melaksanakan tugas mereka secara efektif sementara pada saat yang bersamaa mereka juga harus bereaksi terhadap perubahan lingkungan [12].

Pada umumnya, untuk mengarahkan unit NPC *player* harus men-*deploy* mereka ke lokasi yang ditentukan oleh *player*. Sebagai contoh kasus, pada game CoC untuk mengirimkan pasukan menyerbu musuh yang memasuki wilayah, *player* harus

men*deploy* pasukan dulu ke lokasi musuh. Sehingga, jika *player* tidak men*deploy* unit NPC untuk menyerang musuh yang masuk ke wilayahnya maka dengan mudah unit NPC musuh dapat menghancurkan wilayah itu. Gambar 2.2 adalah gerakan tangan player menyentuh tab saat dia mengirim pasukannya secara manual ke lokasi yang dia inginkan.

Gambar 2.2 Gerakan Jari pada Tab Saat Deploy Pasukan
(http://www.i.ytimg.com)

Untuk menghindari hal tersebut, dibutuhkan *Unit* NPC berbasis agen otonom yang dapat bereaksi secara otonom terhadap setiap serangan yang masuk kewilayah *player*. Artinya, dengan adanya NPC berbasis agen otonom *player* tidak perlu secara manual men*deploy* pasukan ke lokasi musuh, karena *Unit* NPC *player* akan menyerang secara otomatis setiap NPC lawan yang masuk ke wilayahnya.

Agen otomon adalah sebuah sistem komputer yang hidup di dalam lingkungan buatan. Dia beraksi sesuai dengan agenda yang ditanamkan padanya sehingga dia bisa tahu harus melakukan apa jika menerima suatu rangsangan. (Franklin dan Graesser 1997). [10]

Agen otonom harus mampu beraksi sesuai dengan sensor yang dia terima. Artinya agen tidak hanya berada dan menjadi bagian dari sebuah lingkungan buatan, tetapi menjadi pasangan dari lingkungan buatan itu. Arsitektur dan mekanisme agen harus terhubung dengan lingkungannya sehingga dapat merasakan setiap tujuan yang dirancang untuknya dan beraksi untuk memenuhi tujuan itu. (Maturana 1975, Maturana dan

Varela 1980, Varela 1991). [10]

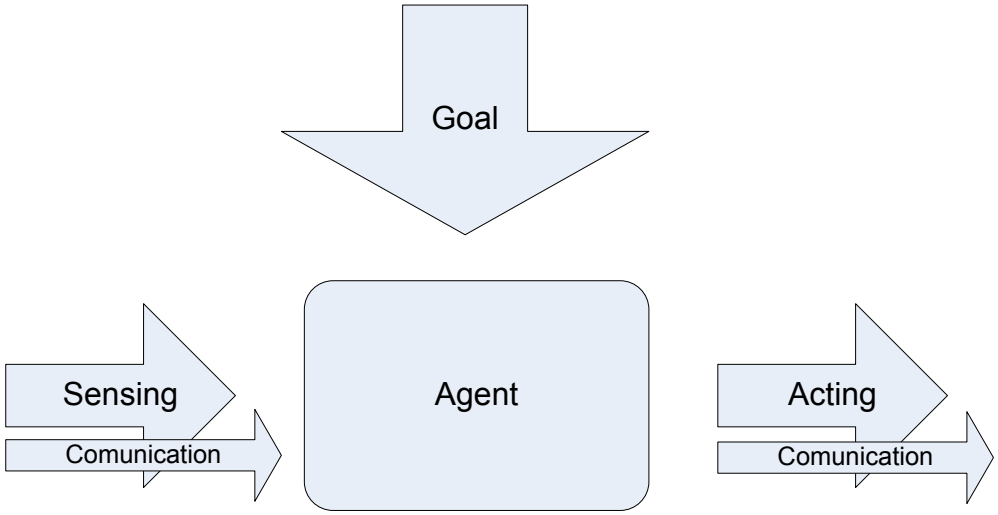
Agen otonom sesuai dengan namanya harus bersifat otonom, mandiri, reaktif, proaktif. Biasanya agen otonom terhubung dengan suatu *server*, sehingga mereka dapat berkomunikasi dengan agen lain dengan bahasa yang telah disepakati antar agen itu (ACL = *Agent Communication Language)* . Maksud dari sifat otonom adalah agen dapat bertindak tanpa adanya control dari manusia atau intervensi lain sesuai dengan kecerdasan buatan yang ditanamkan padanya. Agen juga harus bersifat reaktif, artinya agen mampu secara terus-menerus berinteraksi dengan lingkungannya dan mampu memberikan respon yang tepat terhadap setiap perubahan yang terjadi di lingkungannya. Selain bersifat reaktif agen juga harus pro-aktif, artinya agen harus mampu mengambil inisiatif sendiri, tidak menunggu sesuatu terjadi sesuatu terlebih dahulu baru bertindak. Sebagai contoh sifat pro aktif agen adalah jika ada musuh memasuki wilayahnya, agen tidak perlu menunggu musuh melepaskan tembakan dulu baru menyerang, tetapi agen akan proaktif menyerang setiap musuh yang memasuki wilayahnya [11].

Selanjutnya pada penelitian ini akan digunakan istilah agen untuk menyebut NPC berbasis agen otonom.

Setiap agen dalam *game* selalu memiliki tujuan (*goal*), seperti : menyerang musuh, tetap hidup, menangkap *avatar*. Agen atau *aotonomous* NPC juga mampu merasakan (*sensing*) perubahan pada lingkungannya, seperti kemampuan melihat halangan, mendengarkan suara. Dan agen juga dapat bertindak (*acting*) sesuai perubahan lingkungan yang ditemuinya, seperti : melompat untuk menghindari halangan, makan. agen diberi semacam indera untuk dapat mengindera lingkungannya dan untuk berkomunikasi dengan agen lain. Gambar 2.3 adalah ilustrasi interaksi agen/NPC dengan lingkungannya.

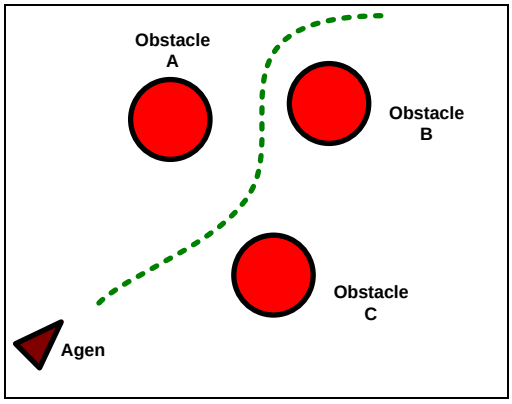
Saat bergerak di dalam lingkungannya, sekelompok agen harus mengatur pergerakaannya agar tidak berbenturan dengan agen lain. *Steering behavior* bertujuan untuk membantu agen bergerak secara realistik di lingkungan buatan di mana agen hidup. Pergerakan ini diciptakan mengacu pada informasi lokal tentang posisi agen lain yang berdekatan dengannya.

Penelitian ini lebih difokuskan pada bagian *steering* yang didalamnya berisi perilaku agen menentukan rencana-rencana untuk menyerang predator secara berkelompok.



Gambar 2.3 Ilustrasi Interaksi Agen Terhadap Lingkungannya

Craig W Reynold membagi steering behavior menjadi beberapa perilaku. Antara lain *seek, flee, pursuit, evasion, offset pursuit, arrival, obstacle avoidance, wander, path following, unligned collision avoidance, separation, cohesion* dan *alignment*. Tetapi pada penelitian ini hanya akan digunakan *obstacle avoidance*, dan *separation* saja.

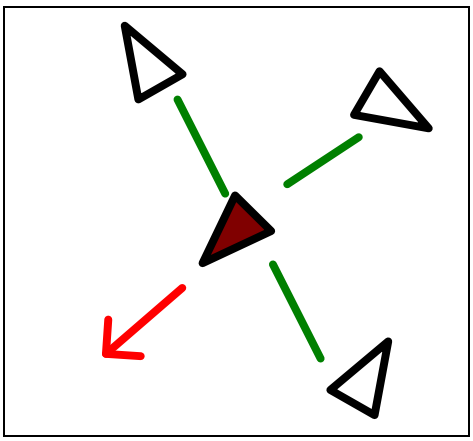


Gambar 2.4 Ilustrasi *Obstacle Avoidance*

Obstacle avoidance adalah perilaku agen melakukan manuver untuk mengindari benturan dengan hambatan (*obstacle*). Ada perbedaan mendasar antara *obstacle avoidance* dengan *flee*. *Flee* akan selalu bergerak menjauh dari target sedangkan *obstacle avoidance* hanya bergerak menghindari rintangan jika menemukan rintangan di depannya.

Gambar 2.4 adalah ilustrasi gerakan agen saat menghindari halangan (*obstacle avoidance*).

Separation adalah perilaku agen untuk menjaga jarak dengan agen lain saat dalam kerumunan/kelompok. Hal ini untuk menghindari agar agen-agen tidak menumpuk di satu tempat. Gambar 2.5 adalah ilustrasi agen saat menjaga jarak dengan agen lain agar tidak saling bertrabakan (*separation*).



Gambar 2.5 Ilustrasi Separation

2.3. Artificial Bee Colony Algorithm

Algoritma Artificial Bee Colony (ABC) pada dasarnya meniru cara kerja lebah madu dalam mencari makanan. Ada tiga komponen penting dalam pencarian makanan ini, yaitu : sumber makanan (*food source*), lebah pekerja (*employed foragers*) dan non lebah pekerja (*unemployed foragers*). Dan dua perilaku utama lebah madu digunakan dalam ABC adalah mencari sumber makanan yang mengandung banyak nektar dan mengabaikan sumber makanan yang mengandung sedikit nektar. Dalam ABC posisi dari sumber makanan mewakili solusi yang mungkin bisa menyelesaikan masalah dan jumlah nektar yang terkandung dalam sumber makanan mewakili kualitas (*fitness*) dari solusi-solusi tersebut [1].

Koloni lebah madu dibagi menjadi 3 jenis, yaitu : *employed bee, onlooker bee* dan *scouts bee. Employed bee* bertugas mencari sumber makanan dan menghitung jumlah nektar. *Onlooker bee* mengamati informasi kualitas sumber makanan dari *employed bee* kemudian memutuskan sumber makanan mana yang akan dituju. Kemudian *employed bee* akan mengeksploitasi sumber makanan sampai sumber makanan itu habis diambil. Jika sumber makanan yang dieksploitasi tadi sudah habis, maka *employed bee* yang tadi

mengeksploitasinya akan berubah menjadi *scout bees* yang bertugas untuk mencari sumber makanan baru disekitar sumber makanan sebelumnya (*neighborhood*) secara acak [1].

Bonabeau berbendapat sebuah *artificial colony* dapat dikatakan cerdas jika memiliki *self-organization* yang bagus. David Karaboga telah membuktikan bahwa algoritma ABC memenuhi standar untuk dikatakan cerdas karena memenuhi 4 karakteriskit *self-organization* yang didifinisikan oleh Bonabeau. Berikui ini 4 karakteristik *self-organization* yang didefinisikan oleh Bonabeau jika diterapkan pada kasus perilaku lebah madu mencari makan :

1. *Positive feedback* : jika jumlah nektar dari sumber makanan meningkat, maka jumlah *onlooker* yang menuju ke sumber makanan juga ikut bertambah.
2. *Negative feedback* : proses eksploitasi akan dihentikan jika sumber makanan sudah habis.
3. *Fluctuation* : *scout* melakukan pencarian sumber makanan baru secara acak
4. *Multiple interaction* : *employed* berbagi informasi tentang sumber makanan dengan *onlooker* yang menunggu di *dance floor*.

Pada algoritma ABC, jumlah *employed bee* sama dengan jumlah *foodsource* (solusi) selama setiap *employed bee* hanya dipasangkan dengan satu *foodsource*. Gambar 2.6 adalah langkah-langkah dasar dari algoritma ABC.

Dalam algoritma ABC, setiap iterasi terdiri dari 3 langkah, yaitu :

1. Mengirim *employed* ke lokasi *foodsource* kemudian *employed* akan menghitung jumlah nektar yang terkandung dalam *foodsource* itu
2. Onlooker menyeleksi *foodsource* berdasarkan informasi yang diperoleh dari *employed* dan memastikan *foodsource* mana yang akan dipilih
3. Memunculkan *scout* kemudian mengirimnya ke kandidat *foodsource* baru yang dipilih secara acak.

Pada tahap inisialisasi, *employed bee* memilih secara acak posisi *foodosurce* dan menentukan jumlah nektarnya. Kemudian pada tahap pertama, *employed bee* menuju ke sarang dan berbagi informasi kandungan nektar yang ada pada *foodosurce* dengan *onlooker* yang menunggu di dalam *dance floor*. Setelah berbagi informasi, *employed bee* akan pergi menuju ke *foodosurce* siklus sebelumnya kemudian memilh *foodosurce* baru berdasarkan informasi yang diterima. Pada tahap ketiga, *onlooker* memilih *foodosurce* berdasarkan informasi jumlah nectar yang dipresentasikan di *dance floor*.

Jika jumlah nektar pada *foodosurce* meningkat, maka kemungkinan *onlooker* untuk memilih *foodosurce* itu juga meningkat. Setelah tiba di *foodosurce* yang dipilih, *onlooker* memilih *foodosurce* baru yang berdekatan dengan sumber makanan saat itu. Informasi visual didasarkan pada pembandingan posisi sumber makanan. Ketika *foodsource* diabaikan oleh lebah, *foodsource* baru dipilih secara acak oleh *scout.*

Inisialisasi (posisi foodsource, jumlah lebah, limit, vektor dimensi)
REPEAT
Menempatkan employe pada sumber makanan
Menempatkan onlooker pada sumber makanan
Mengirim scout untuk mencari sumber makanan baru
UNTIL (kriteria terpenuhi)

Gambar 2.6 Dasar Algoritma ABC

Dalam memodelkan perilaku lebah saat mencari sumber makanan menjadi algoritma ABC, posisi *foodsource* dianggap sebagai solusi yang mungkin diambil (*possible solution*) untuk suatu masalah optimasi dan jumlah nectar pada setiap *foodsource* menunjukkan kualitas (*fitness*) dari solusi itu. Jumlah employed atau *onlooker* sama dengan jumlah solusi dalam populasi.

Pada tahap awal, ABC akan me-*generate* (membuat) inisialisasi populasi awal P(G=0) yang didistribusikan secara acak untuk SN solusi (posisi *foodsource*), dimana SN mewakili jumlah dari populasi. Setiap posisi *foodsource* x_i (i = 1, 2, …, SN) berada pada D vektor dimensi. Dimana D adalah parameter optimisasi. Setelah inisialisasi, setiap tahap dalam algoritma ABC diiterasi sebanyak C kali (C = 1,2, …, C_{max}).

Artificial employed dan *onlooker* secara probabilistik membuat modifikasi posisi foodsource untuk menentukan *foodsource* baru. Di alam, lebah asli akan mengumpulkan kemudian membandingkan informasi visual disekitarnya sebelum menentukan *foodsource* baru yang akan dituju. Pada algoritma ABC penentuan posisi *foodsource* baru juga ditentukan berdasarkan pembandingan informasi yang dikumpulkan oleh lebah *artificial*. Hanya saja, di dalam pemodelan algoritma ABC, lebah *artificial* tidak

membandingkan informasi visual saat menentukan *foodsource* baru. Lebah *artificial* secara acak memilih posisi *foodsource* baru dan membuat posisi *foodsource* baru dengan menggunakan persamaan (2.2). Jika *fitness value* pada *foodsource* baru lebih tinggi dari *foodsource* sebelumnya, maka lebah akan menyimpan posisi *foodsource* baru dalam memorinya dan membuang posisi *foodsource* lama. Sebaliknya, lebah akan menyimpan posisi *foodsource* lama jika ternyata *fitness foodsource* baru lebih rendah dari *foodsource* lama.

Setelah semua proses pencarian *foodsource* baru selesai dilakukan oleh semua *employed*, mereka akan memberikan informasi pada *onlooker* tentang *fitness foodsource*-nya di dance area. *Onlooker* akan mengevaluasi informasi nektar yang disampaikan oleh *employed*, kemudian memilih *foodsource* sesuai nilai probabilitas jumlah nektar yang dibawa oleh setiap *employed* dengan menggunakan persamaan pada *equation* (2.1). Seperti halnya *employed*, *onlooker* akan membuat modifikasi posisi *foodsource* baru dan menghitung jumlah nektar *foodsource* baru itu. Jika jumlah nektar lebih tinggi dari nektar *foodsource* sebelumnya, maka dalam memori *onlooker* posisi *foodsource* baru menggantikan posisi *foodsource* lama.

Onlooker akan memilih *foodsource* berdasarkan pada nilai probabilitas *foodsource*-nya (p_i) yang dihitung menggunakan persamaan berikut ini :

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \tag{2.1}$$

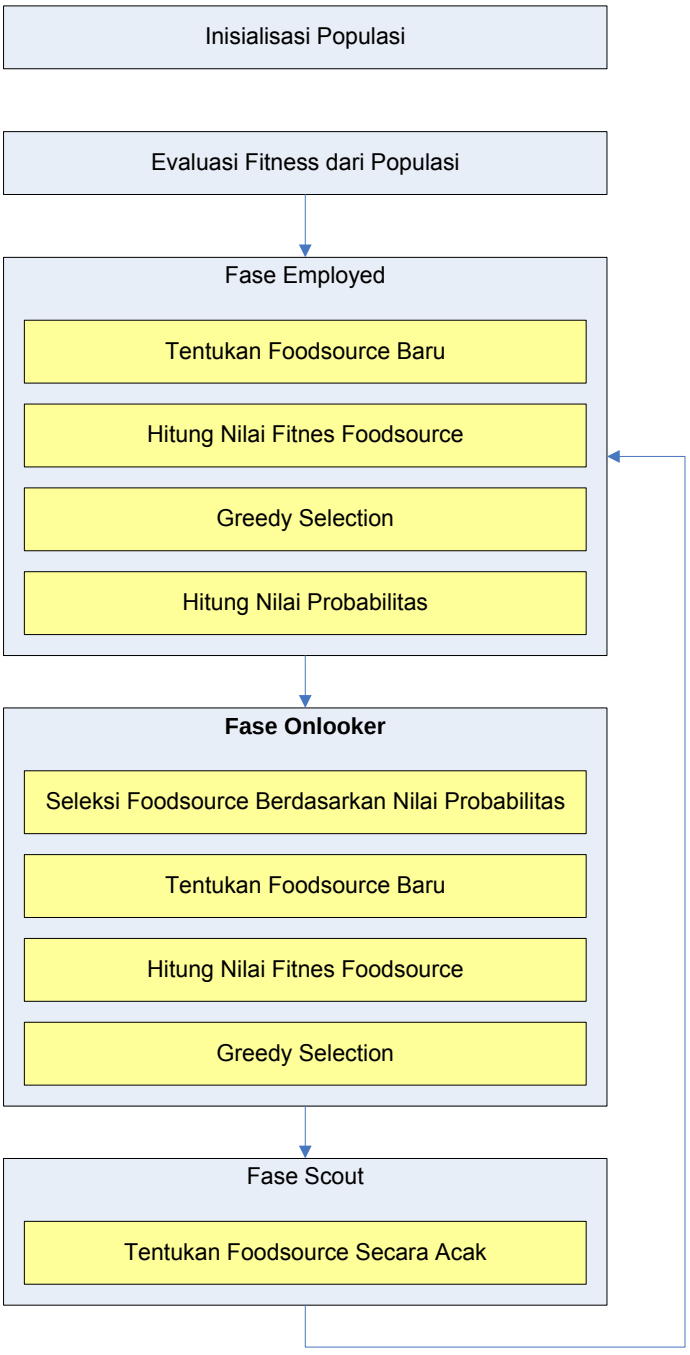
dimana fit_i adalah fitness dari *foodsource i* dan SN adalah jumlah *foodsource* (*solution*)

Sedangkan untuk menentukan kandidat posisi *foodsource* baru, algoritma ABC menggunakan persamaan berikut ini :

$$v_{ij} = x_{ij} + \emptyset_{ij}(x_{ij} - x_{kj}) \tag{2.2}$$

dimana $k \in \{1, 2, \ldots, BN\}$ yang dipilih secara acak dan $j \in \{1, 2, \ldots, D\}$. Meskipun k dipilih secara acak, tetapi nilai k tidak boleh sama dengan i. Φ_{ij} adalah nilai *random* antara [-1,1]. Persamaan ini mengontrol munculnya *foodsource* baru di sekitar posisi *foodsource* x_{ij} dan posisi *foodsource* yang baru akan dievaluasi lagi, apakah layak dianggap sebagai *foodsource* baru. Dari persamaan di atas dapat terlihat bahwa jika perbedaan antara posisi x_{ij} dan x_{kj} kecil, maka jarak antara posisi *foodsource* lama dengan *foodsource* baru pun tidak jauh.

Fitness setiap posisi *foodsource* baru akan dihitung oleh lebah, dan *foodsource* baru akan diabaikan jika nilai *fitness*nya lebih rendah dari *foodsource* lama. Lebah akan mencari kemungkinan posisi *foodsource* baru untuk menggantikan *foodsource* yang diabaikan tadi. Jika selama L (*limit*) kali pencarian posisi *foodsource* baru tidak ditemukan, maka *scout* akan mencarikan *foodsource* baru untuk lebah *i*.



Gambar 2.7 Flowchart Algoritma ABC

Algoritma ABC melakukan 4 proses seleksi yang berbeda:

1. Proses *global selection* digunakan oleh onlooker untuk menemukan *foodsource* baru dengan menggunakan persamaan (2.1)
2. Pemilihan acak *foodsource* baru dilakukan oleh *employed* dan *onlooker* dengan menggunakan persamaan (2.2)
3. Proses *greedy selection* dilakukan oleh semua lebah. Jika jumlah nektar *foodsource* baru lebih baik dari jumlah nektar *foodsource* lama, maka posisi *foodsource* baru menggantikan posisi *foodsource* lama dalam memori lebah. Sebaliknya, jika *foodsouce* baru jumlah nectarnya lebih sedikit dari jumlah nektar posisi lama, maka *foodsource* baru diabaikan dan lebah hanya mengingat posisi *foodsource* lama.
4. Pemilihan *foodsource* baru secara acak oleh *scout*.

Dari penjelasan di atas, dapat diketahui bahwa ada 3 parameter kontrol yang digunakan dalam algoritma ABC, yaitu : jumlah *foodsource* sama dengan jumlah lebah (SN), nilai Limit dan jumlah siklus maksimum. Gambar 2.7 adalah *flowchart* dari algortima Artificial Bee Colony

2.4. Clash of Clans

Penelitian ini difokuskan pada pencarian rute pergerakan sekelompok agen saat menuju target tertentu dan Clash of Clans (CoC) menjadi salah satu rujukan penelitian ini. Pasukan dalam CoC dikenal dengan istilah *troops*. Pasukan atau *troops* digunakan untuk mempertahankan wilayahnya (desa) atau menghancurkan *base* lawan, mengumpulkan *Tropi, Gold* dan *Elixir*. Setiap pasukan yang ada pada CoC memiliki kemampuan dan ciri khas yang berbeda-beda. Pasukan dalam CoC dikelompokkan dalam beberapa kelompok yaitu : Tier 1, Tier 2, Tier 3, Dark Elixir dan Heroes

Troops pada Tier 1 mudah dilatih dan harganya murah. Mereka memiliki *hitpoint* yang rendah dan tidak mampu membuat kerusakan yang fatal jika bekerja secara individu. Mereka lebih mampu membuat kerusakan besar jika bekerja dalam kelompok saat menyerang pertahanan tunggal. *Troop* pada Tier 1 mudah dikalahkan oleh Wizards Tower dan Mortars hanya dengan sekali tembak. Barbarian, Archer dan Goblin adalah *troop* yang ada di Tier 1.

Troop yang ada di Tier 2 lebih kuat daripada *troop* di Tier 1. Sama pada *troop* di Tier 1, *troops* pada Tier 2 juga memiliki keahlian khusus. Seperti Wall Breaker yang hanya bertugas menghancurkan dinding, Wizards meskipun tidak terlalu kuat namun

mampu menembakkan api pada musuh, Giant dan Ballons dirancang untuk mempertahankan wilayah. Ballons merupakan unit terbang pertama yang tersedia di CoC.

Troops Tier 3 merupakan *troop non hero* yang paling kuat dan paling bertenaga dalam CoC. Mereka mampu menghancurkan semua desa hanya dengan sedikit pasukan. Namun, *troops* di Tier 3 membutuhkan biaya yang mahal dan waktu yang lama untuk melatihnya. Pada umumnya, kekuatan 6 Barbarian atau Archer di *level* 6 setara dengan kemampuan *troop* di Tier 3 ini. Keunggulan lain *troop* di Tier 3 adalah mereka tidak mudah dikalahkan oleh Mortars dan Wizard Tower.

Kelompok *troop* Dark Elixir adalah *troops* yang memiliki fungsi khusus yang tidak dimiliki oleh *troop* biasa, seperti *troop* terbang, perusak dinding, *troop* yang bertugas membuat kerusakan darat dengan menggunakan senjata yang dilempar dan kekuatan besar. Troop yang menjadi anggota kelompok Dark Elixir adalah Minions, Hog Riders, Valkyries, Golems, Witches dan Lava Hounds.

Berikutnya adalah kelompok *troop* Heroes yang terdiri dari Barbarian King dan Archer Queen. Mereka adalah *troop* yang kekal atau *immortal*. Mereka memiliki kekuatan terbesar namun setiap *player* hanya memiliki 1 *troop* Heroes saja. Meskipun Heroes tidak dapat mati, tapi mereka tetap dapat terluka dan membutuhkan waktu istirahat untuk menyembuhkan luka mereka. Saat melakukan serangan dan untuk menjaga agar Heroes tidak terluka, Heroes perlu di*back up* oleh *troop* lain. *Troop* yang biasa digunakan untuk membackup serangan Heroes adalah Barbarian dan Archer. Setiap Barbarian King dan Archer Queen dapat mengeluarkan Barbarian dan Archer untuk membantu mereka menyerang lawan.

Berdasar medan geraknya, ada 2 jenis pasukan dalam CoC, yaitu unit darat (*ground unit*) dan unit udara (*flying unit*). Unit darat bergerak di darat, sedangkan unit udara adalah unit terbang bergerak di udara. Semua unit udara dalam CoC tidak mempunyai kemampuan untuk menghindari serangan yang diberikan padanya. Mereka akan bergerak langsung dari posisi awal mereka di*deploy* menuju posisi target tanpa memperhatikan arah serangan lawan. Hal ini membuat *troop* unit terbang mudah ditembak. Sedangkan unit darat bergerak sesuai dengan tugas yang diberikan padanya.

Ada 2 unit darat berdasarkan targetnya, yaitu unit darat yang memiliki target khusus dan unit darat yang tidak memiliki target khusus. Unit darat yang memiliki tujuan khusus akan melewati semua bangunan yang bukan targetnya. Jika saat bergerak menuju target unit darat terbentur dengan dinding, maka mereka akan menghancurkan dinding tersebut sebelum menuju bangunan targetnya. Unit darat yang tidak memiliki tujuan

khusus akan menghancurkan setiap bangunan lawan yang ditemuinya.

Penelitian ini akan meneliti pergerakan kelompok troop yang mempunyai tujuan khusus menghancurkan pertahanan lawan dan sumber daya lawan. Hog Rider, Giant adalah 2 jenis troop yang mempunyai tujuan menghancurkan pertahanan lawan. sedangkan Goblin mempunyai misi menghacurkan banguan penyimpan sumber daya lawan.

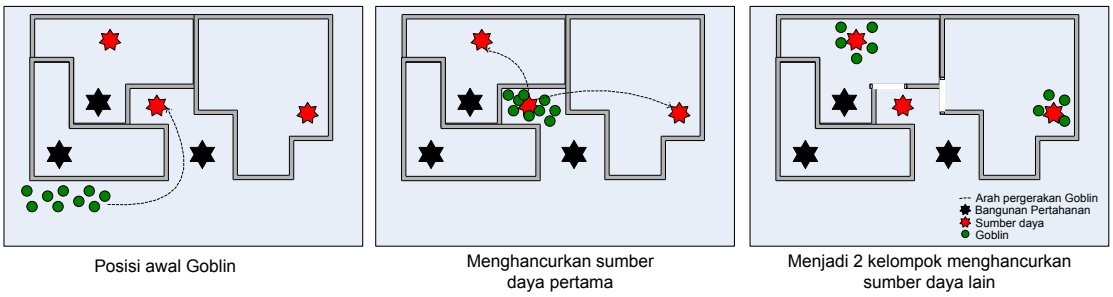
Karakter Goblin digambarkan sebagai mahluk hijau kecil dengan telinga runcing lebar, mata merah dengan pupil berwarna hijau dan hidung merah. Goblin memakai celana dan sepatu coklat serta membawa karung yang berisi sumber daya yang dicuri dari lawan. Goblin merupakan karakter tercepat dalam CoC. Bangunan sumber daya adalah target utama Goblin, seperti Gold, Elixir dan penyimpanan. Karena Goblin mengutamakan menyerang bangunan sumber daya, maka mereka akan mengabaikan semua pertahanan lawan, rentan terhadap serangan dan lemah jika digunakan tanpa *back up* dari *troop* lain. Gambar 2.8 adalah perubahan karakter Goblin di beberapa level game CoC.

Gambar 2.8 Karakter Goblin
(http://clashofclans.wikia.com/)

Saat mencuri sumber daya lawan Goblin membutuhkan bantuan Wall Breaker untuk menghancurkan dinding lawan. Namun jika terpaksa bergerak tanpa *back up* Goblin juga mampu merusak dinding. Barbarian, Giant atau troop dengan nyawa yang lebih tinggi dapat digunakan untuk membantu Goblin menghancurkan Mortar dan Wizard Tower yang melindungi sumber daya lawan. Karena Goblin mudah ditembah Mortar dan Wizard pada umumnya Goblin dikirim dalam jumlah yang besar agar kemungkinan jumlah Goblin mencuri sumber daya juga semakin besar. Ketika semua penyimpanan sumber daya telah dihancurkan, GOblin akan menghancurkan apapun yang bisa mereka serang.

Saat dikirim secara masal, dari posisi awal mereka dikirim Goblin akan langsung mencari posisi bangunan sumber daya terdekat. Mereka akan menuju bangunan sumber

daya yang sama dan mengabaikan bangunan lain. Jika bangun sumber daya yang diserbu pertama kali sudah mereka hancurkan, maka mereka akan pecah menjadi beberapa kelompok. Masing-masing kelompok akan mencari sumber daya lain yang bisa ditemukan. Jika sumber daya ada di balik dinding, maka Goblin harus menghancurkan dinding itu. Gambar 2.9 adalah ilustrasi pergerakan Goblin dari posisi awal dikirim ke posisi sumber daya yang diincar.



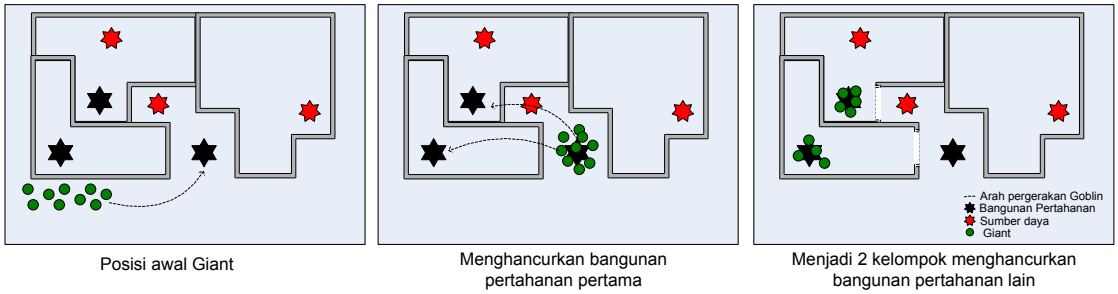
Gambar 2.9 Ilustrasi Pergerakan Goblin Menuju Target

Giant adalah *troop* berbadan besar yang mampu memporakporandakan lawan. Troop ini mengincar pertahanan musush seperti Canon, Wizard Tower dan Archer Tower. Dalam jumlah yang besar mereka dapat menghancurkan sebuah desa. Karena *hitpoint* Giant sangat besar, pada umumnya *player* menempatkan mereka terlebih dahulu untuk melindungi pasukan lain yang lebih lemah. Gambar 2.10 adalah perbedaan perubahan Giant di setiap level.

Gambar 2.10 Karakter Giant
(http://clashofclans.wikia.com/)

Hampir sama dengan Goblin, Giant juga dapat menyerang dalam kelompok. Mereka cenderung akan menyerang bangunan yang sama terlebih dahulu. Jika bangunan yang pertama dituju sudah berhasil dihancurkan, maka mereka juga akan dibagi menjadi beberapa kelompok untuk mencari target bangun pertahan lain untuk dihancurkan. Saat bergerak menuju target, Giant tidak diberi kecerdasan untuk menghindari serangan lawan. Selain itu Giant juga tidak diberi kecerdasan untuk menjaga jarak antar agen, sehingga

saat menggerombol di tempat yang sama mereka akan terlihat saling tumpang tindih. Gambar 2.11 adalah ilustrasi pergerakan Giant dari posisi awal mereka di*deploy* sampai menuju target.



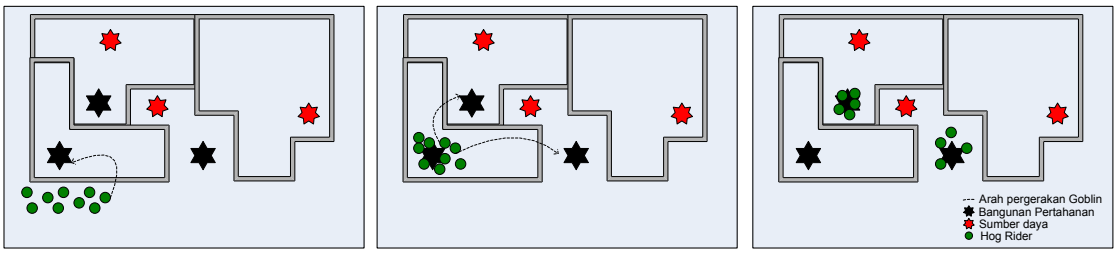
Gambar 2.11 Ilustrasi Pergerakan Giant Menuju Target

Hog Riders digambarkan sebagai pria berkulit gelap, kasar, menaiki babi besar, bertelanjang dada, mengenakan celana pendek kulit berwarna coklat dengan sabuk merah, memakai sandal kulit, memakai dua gelang dan anting-anting emas, bersenjatakan Warhammer besar. Bersama babinya, troop ini mampu melompati dinding lawan dan bergerak menghancurkan *base* lawan dengan cepat. Gambar 2.12 adalah perubahan penampilan Hog Rider di beberapa level.

Gambar 2.12 Karaakter Hog Rider
(http://clashofclans.wikia.com/)

Hog Riger akan menyerang pertahanan lawan yang terdekat darinya. Hog Rider dapat digunakan sebagai pengganti Giant karena keduanya sama-sama memiliki tubuh yang kuat. Babi tunggangan Hog Riders mampu melompati semua *level* Wall sehingga Hog Riders dapat menghancurkan semua targetnya meskipun target itu ada di dalam dinding yang tinggi. Namun meskipun Hog Riders dapat melompati semua dinding, tetapi jika Hog Riders membutuhkan *back up* dari *ground troop* lain, maka *player* harus mengirimkan Wall Breaker sebagai pembuka jalan bagi *troop* lain yang akan membantu Hog Riders. Hog Rider yang menyerang secara berkelompok akan sangat berbahaya bagi

lawan, karena *troop* ini sangat kuat dan memiliki *hit point* yang tinggi. Musuh utama Hog Riders adalah Giant Bomb, oleh karena itu *player* disarankan menjauhkan diri dari Giant Bomb atau mengorbankan Barbarian atau satu Hog Rider untuk menghancurkan Giant Bomb sebelum mengirimkan Hog Rider.Gambar 2.13 adalah ilustrasi pergerakan Hog Rider saat bergerak ke posisi target. Terlihat pada Gambar 2.13 Hog Rider tidak menghancurkan dinding jika ingin menuju target yang berada di dalam dinding.



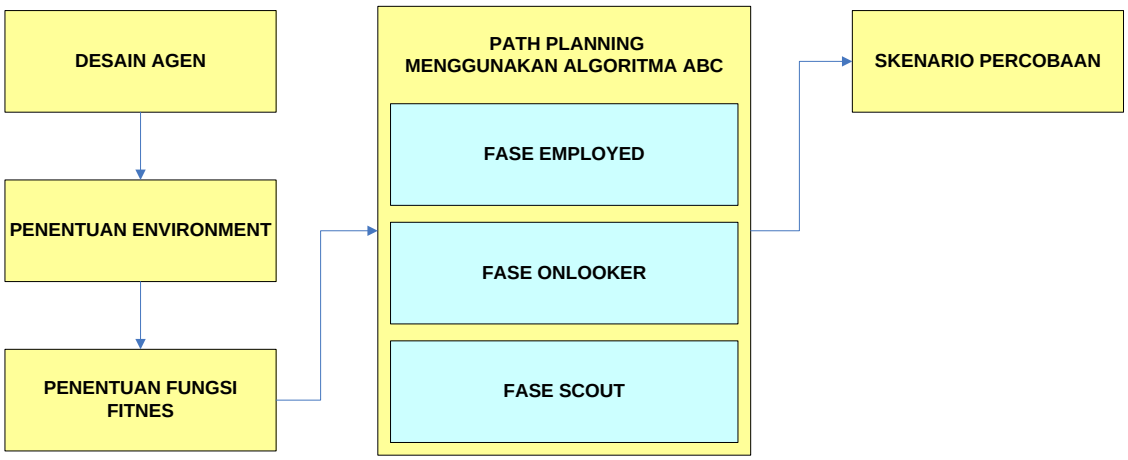
Gambar 2.13 Ilustrasi Pergerakan Hog Rider Menuju Target

BAB 3
METODOLOGI PENELITIAN

Hog Rider, Goblin dan Giant adalah 3 contoh unit darat yang bergerak berdasarkan tujuan khusus. Target utama Hog Rider adalah pertahanan lawan yang terdekat darinya. Target Goblin adalah sumber daya dan target Giant adalah pertahanan lawan seperti Canon, Wizard Tower dan Archer Tower. Saat bergerak bersama-sama menuju target *troops* ini cenderung konvergen, hal ini membuat mereka mudah dihancurkan oleh pertahanan lawan.

Agen dalam game membutuhkan *path planning* untuk menentukan jalur menuju posisi berikutnya atau menuju targetnya. Penelitian ini dilakukan untuk membuat *path planning* sekelompok NPC otonom yang bergerak bersama menuju target dari arah yang berbeda-beda dengan menggunakan algoritma Artificial Bee Colony di ruang 3 dimensi. Hasil dari penelitian ini dapat diterapkan untuk menentukan pergerakan Hog Rider, Goblin maupun Giant. Jika ketiga *troop* itu dapat bergerak bersama dari arah yang berbeda maka kemungkinan mereka menjadi target pertahanan lawan akan semakin kecil sehingga kemungkinan mereka untuk mencapai tujuannya semakin besar.

Gambar 3.1 adalah diagram alur langkah-langkah pelaksanaan penelitian ini. Diawali dengan penentuan desain perilaku agen sampai dengan visualisasi pergerakan agen di dalam ruang 3 dimensi.



Gambar 3.1 Skema Metodologi Penelitian

3.1. Desain Agen

Hasil dari penelitian ini dapat diterapkan untuk pergerakan Giant, Hog Rider, Goblin atau NPC lain yang membutuhkan pergerakan sekelompok agen saat menyerang secara berkelompok untuk melumpuhkan target yang memiliki kekuatan lebih besar. Dibutuhkan kerja sama antar agen untuk mengalahkan lawan, seperti pada serangan berkelompok lebah madu ketika bekerja sama mengalahkan lawan yang kekuatannya lebih besar. Mereka mengerumuni lawan, memposisikan diri menyelimuti lawan mengakibatkan suhu disekitar lawan meningkat melebihi suhu maksimum yang dapat diterimanya, sehingga lawan mati karena tidak tahan terhadap suhu yang tinggi.

Pada algoritma ABC, simulasi dilakukan pada agen lebah yang mencari sumber makanan dan mengeksploitasinya. Posisi *food source* atau sumber makanan ditentukan secara acak. *Foodsource* dianggap sebagai posisi baru agen ketika bergerak mendekati target/lawan. Modifikasi algoritma ABC ditujukan untuk melakukan simulasi pergerakan serangan terhadap lawan.

Setiap agen memiliki property *speed, power, sensor* dan efektor. Tabel 3.1 adalah parameter serangan yang dimiliki oleh agen dan target.

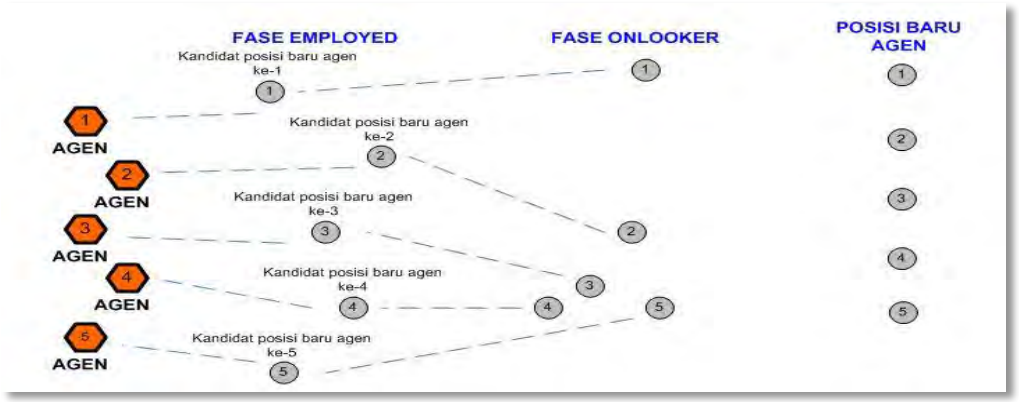
Setiap agen mempunyai kecepatan yang tinggi saat bergerak tetapi kekuatannya rendah. Agen hanya dapat menyerang sekali kemudian mati. Sedangkan lawan dianggap sebagai target diam yang memiliki kekuatan 10 kali lebih kuat dari agen. Sehingga butuh 10 kali serangan agen untuk menghancurkan.

Tabel. 3.1 Parameter Serangan

Parameter	**Agen**	**Lawan**	**Keterangan**
Speed	Tinggi	0	
Power	1	10	
Sensor	Barrier		Untuk mendeteksi keberadaan dinding/obstacle
	Beacon		Untuk mendeteksi keberadaan musuh
Effector	Direction		Menentukan arah
	Speed		Menentukan kecepatan
Behaviour	Wander		Menyusuri environment
	Track target		Mencari musuh
	Avoid barrier		Menghindari obstacle
	Avoid beacon		Menghindari agen lain

Setiap agen memiliki sensor *barrier* untuk mendeteksi keberadaan *obstacle* dan sensor *beacon* untuk mendeteksi keberadaan target didekatnya. Setiap agen juga

memiliki 4 perilaku, yaitu *wander* (perilaku berkelana menyusuri lingkungan), *track target* (mencari musuh), *avoid barrier* (mengindari dinding) dan *avoid beacon* (menghindari lawan/agen lain).



Gambar 3.2 Ilustrasi Penentuan *Path Planning*

Pada penelitian ini, semua agen dianggap sebagai populasi algoritma ABC. Artinya, setiap agen tidak diberi algoritma ABC penuh melainkan menjadi bagian dari algoritma ABC itu sendiri. Dengan harapan ketika divisualisasikan gerakan agen akan lebih acak. Selain itu, jika agen diposisikan sebagai parameter dari algoritma ABC maka proses komputasi akan lebih ringan. Gambar 3.2 adalah ilustrasi pencarian posisi baru menggunakan algoritma ABC pada penelitian ini.

Penelitian yang dilakukan oleh Preetha B [8] pada tahun 2011 membuktikan bahwa setiap agen yang diberi kecerdasan ABC penuh bergerak lurus menuju target. Hal ini akan membuat lawan dapat membaca gerakan agen sehingga lawan dapat dengan mudah menembak agen.

Pada tahap Inisialisasi, posisi agen akan ditentukan secara acak. Kemudian pada fase *Employed* posisi baru agen akan ditentukan secara acak pula. Jika posisi baru itu membuat agen lebih dekat dengan target maka posisi itu akan ditempati. Dari fase *Employed* agen akan masuk fase *Onlooker*. Pada fase *Onlooker* posisi baru agen akan ditentukan perbadasarkan posisi agen yang memiliki nilai probabilitas terbaik.

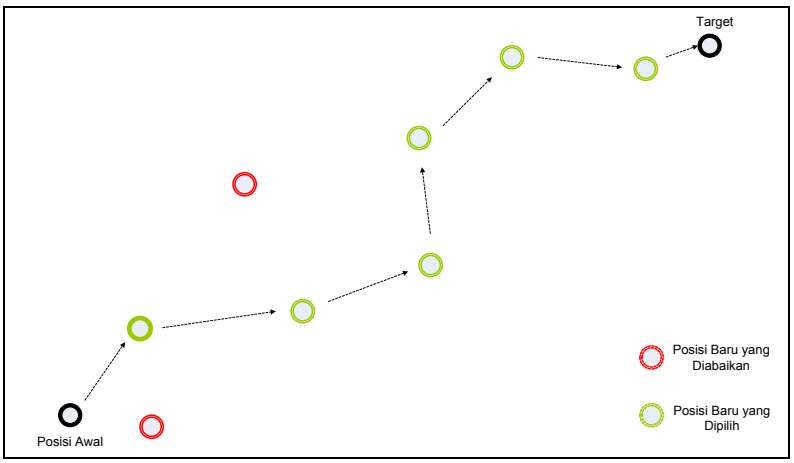
Jika setelah dievaluasi posisi baru pada fase *Onlooker* ini lebih baik dari posisi agen sebelumnya, maka agen akan menempati posisi baru. Sedangkan jika posisi baru baik di fase *Employed* maupun *Onlooker* tidak lebih baik dari posisi saat ini, maka posisi baru itu akan diabaikan oleh agen. Agen akan diam ditempat sampai posisi baru ditemukan. Jika setelah sekian kali pencarian posisi baru bagi agen tidak membuat agen pindah tempat, maka pada fase *Scout* akan dicarikan posisi baru secara acak untuk agen

tersebut.

Agen dalam penelitian ini bergerak dalam ruang 3 dimensi dengan perilaku sebagai berikut :

1. Agen akan memilih jalur yang terpendek menuju targetnya secara acak. Setiap akan menentukan posisi baru, agen akan menghitung apakah posisi barunya akan membuatnya lebih dekat dengan targetnya atau tidak. Jika posisi baru akan membuatnya lebih dekat dengan target, maka posisi itu akan ditempati. Sebaliknya, jika posisi baru membuat agen lebih jauh dari targetnya, maka untuk sementara agen akan diam ditempat dan mencari posisi baru pada iterasi berikutnya sampai ditemukan posisi yang membuatnya lebih dekat dengan target.

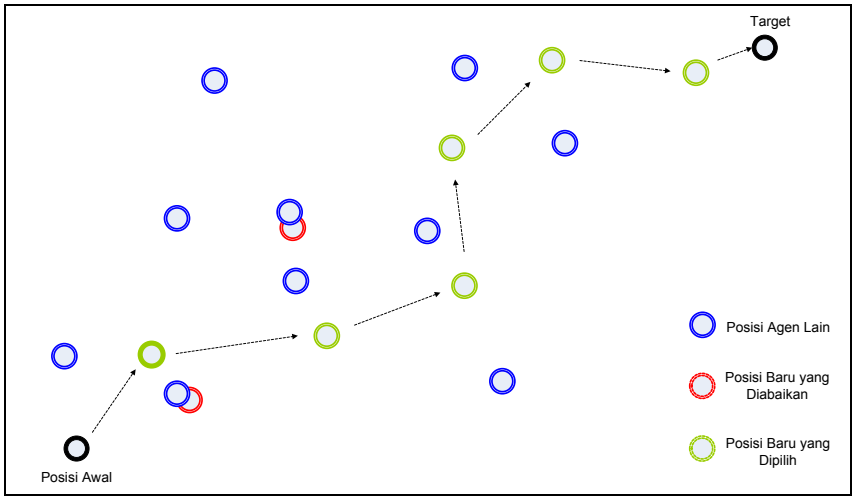
   Gambar 3.3 adalah simulasi pergerakan agen saat mencari posisi baru untuk mendekati posisi target.



Gambar 3.3 Ilustrasi Pergerakan Agen Menuju Target

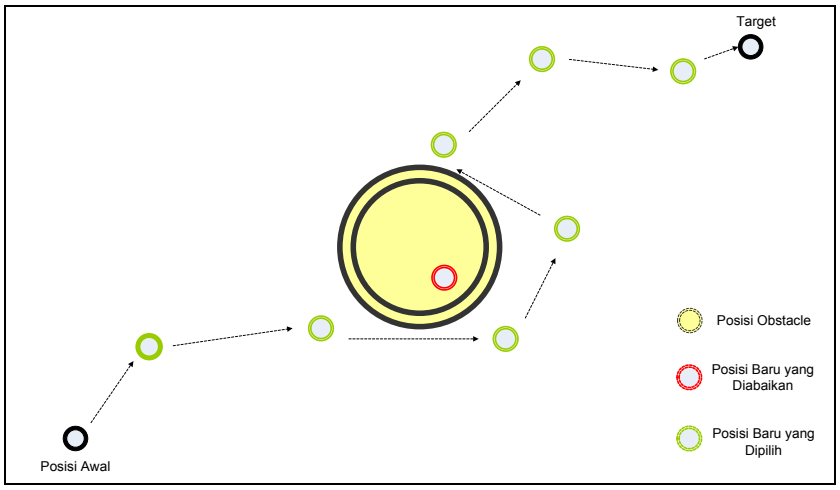
2. Agen akan menghindari benturan dengan kawan. Jika posisi baru agen terdapat agen lain atau terlalu dekat dengan agen lain, maka agen tetap di tempat agar tidak terjadi benturan dengan agen lain tersebut. Agen akan menentukan kandidat posisi baru pada iterasi berikutnya.

   Gambar 3.4 adalah simulasi pergerakan agen saat bergerak mencari posisi baru dengan memperhatikan posisi agen lain untuk menghindari tabrakan dengan mereka.



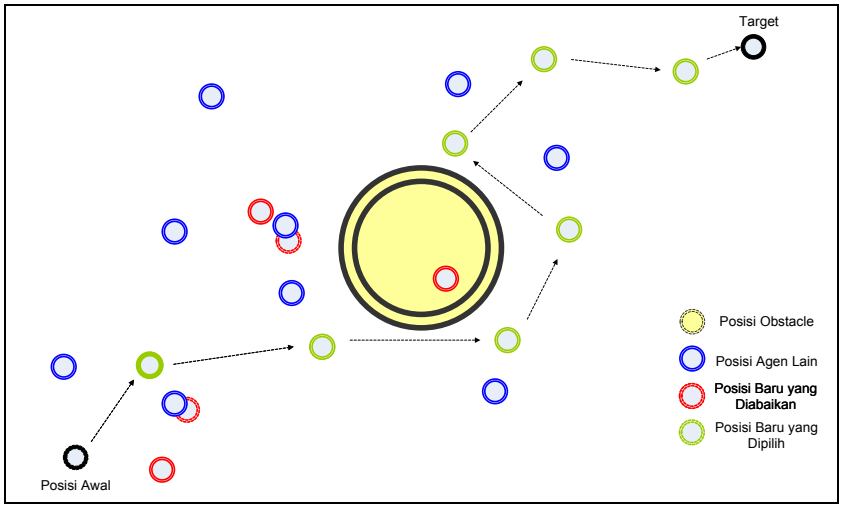
Gambar 3.4 Ilustrasi Pergerakan Agen Menghindari Tabrakan dengan Agen Lain

3. Agen akan menghindari benturan dengan *obstacle* statis. Jika posisi baru merupakan posisi *obstacle* maka agen harus mencari posisi lain dan menambahkan nilai 1 pada status *trial*-nya. Hal ini menandakan bahwa agen sudah pernah mencoba melakukan penentuan posisi baru tetapi posisi baru kurang menguntungkan bagi agen. Agen tetap diam di tempat menunggu iterasi berikutnya untuk penentuan posisi baru selanjutnya.



Gambar 3.5 Ilustrasi Pergerakan Agen Menghindari *Obstacle*

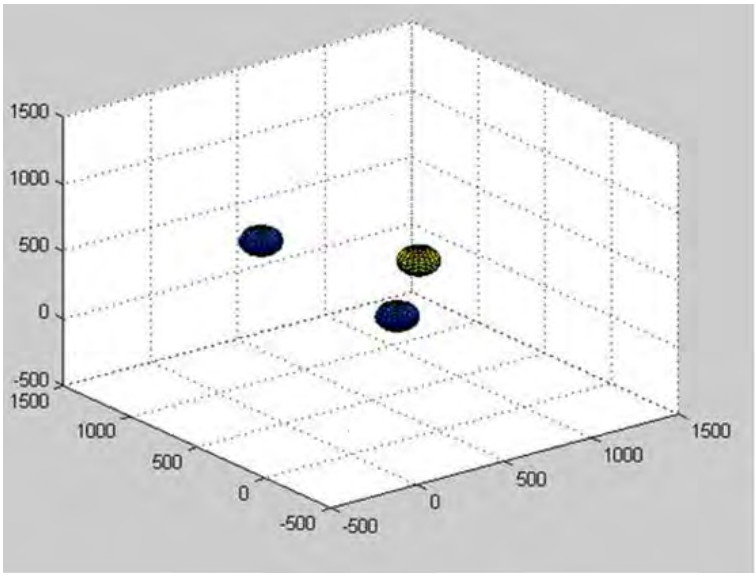
Gambar 3.5 adalah simulasi pergerakan agen saat bergerak mencari posisi baru menuju target dengan mengevaluasi posisi agen lain dan posisi *obstacle* agar tidak menabrak agen lain dan *obstacle* statis.



Gambar 3.6 Ilustrasi Pergerakan Agen

Gambar 3.6 adalah gambar simulasi pergerakan agen dari posisi awal menuju target dengan memilih posisi baru secara acak kemudian mengevaluasi posisi baru tersebut agar posisi baru membuat agen lebih dekat dengan target. Dan saat bergerak menuju target, agen tidak boleh bertabrakan dengan agen lain atau dengan obstacle statis.

3.2. Environment



Gambar 3.7 Ruang Gerak Agen

Agen akan bergerak di ruang 3 dimensi di mana didalamnya terdapat obstacle diam yang harus dihindari oleh agen saat bergerak menuju target. Gerakan menghindar dilakukan dengan mengitari obstacle melewati sebelah kiri, sebelah kanan, sebelah atas, atau sebelah bawahnya. Gambar 3.7 adalah *environment* agen bergerak.

3.3. Penentuan Fitness Function

Di setiap fase *Employed* dan fase *Onlooker* agen selalu melakukan proses *greedy selection* untuk menentukan *local optimal* dengan harapan agen dapat segera menemukan *global optimal*nya yaitu berada sedekat mungkin dengan posisi target. Proses *greedy selection* dilakukan dengan cara membandingkan fitness posisi baru dengan posisi lama. Hanya posisi yang memiliki *fitness* lebih bagus saja yang akan dipilih oleh agen. Setiap posisi dengan fitness rendah akan diabaikan oleh agen dan agen akan diam ditempat hingga N kali. Jika setelah N kali *greedy selection* lebah tidak bergerak, maka pada fase *Scout* akan ditentukan posisi baru agen secara acak.

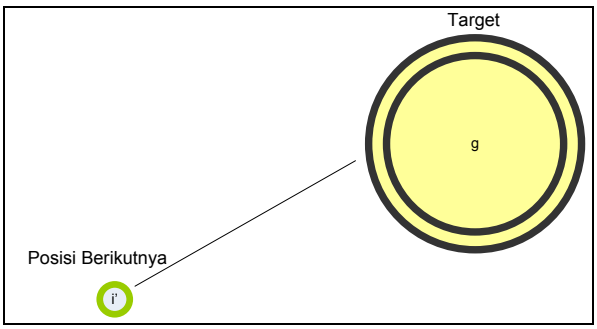
Untuk mengukur sebagus apa setiap posisi baru yang diciptakan, maka *fitness function* pada penelitian ini mempunyai 2 komponen, yaitu :

1. Fungsi objektif untuk menentukan jarak terdekat menuju target
2. *Constrain*, untuk menghindari tabrakan dengan agen lain dan obstacle statis

Untuk memenuhi fungsi objektif, maka agen harus menghitung jarak posisi baru (x'_i, y'_i, z'_i) ke posisi target (x_{ig}, y_{ig}, z_{ig}).

$D_{i\prime-g}$ adalah jarak antara posisi baru dengan posisi target, maka untuk mencari besarnya jarak antara posisi baru dengan posisi target adalah

$$D_{i\prime-g} = \sqrt{(x'_i - x_g)^2 + (y'_i - y_g)^2 + (z'_i - z_g)^2} \qquad (3.1)$$



Gambar 3.8 Jarak (D) Posisi Baru (i') dengan posisi target (g)

Jarak inilah yang akan menjadi fitness dari posisi agen, di mana semakin kecil jarak antara agen dengan target maka fitness akan dianggap semakin bagus. Jadi untuk bergerak ke arah target maka dilakukan minimasi nilai fitness ini.

Sedangkan untuk menghindari tabrakan dengan agen lain, dievaluasi posisi tiap agen lain apakah dalam jarak yang cukup jauh atau terlalu dekat. Dua agen dikatakan

bertabrakan jika jarak keduanya lebih kecil dari jumlah radius (R) ukuran 2 lebah tersebut.

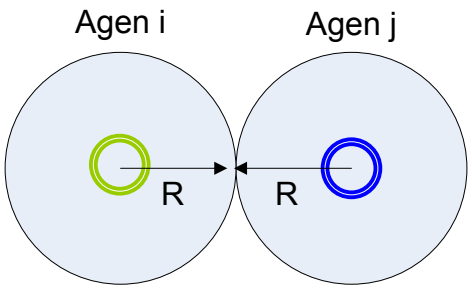
D_{i-j} adalah jarak antara lebah ke-*i* dan lebah ke-*j*. Untuk menghitung jarak antara lebah ke-*i* dengan lebah ke-*j* adalah sebagai berikkut

$$D_{i-j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad (3.2)$$

Gambar 3.9 Jarak Agen ke-i dengan Agen ke-j

Agar dua agen tidak bertabrakan, maka jarak kedua agen itu jika dikurangkan dengan radius atau ukuran keduannya, nilainya harus lebih besar dari 0. Sehingga, jika R adalah radius atau ukuran dari lebah, maka

$D_{i-j} - 2R > 0$



Gambar 3.10 Dua Agen Berhimpitan

Untuk menghindari tabrakan dengan semua agen lain, lebah ke-*i* harus mengevaluasi jaraknya dengan semua agen. Fungsi untuk menghindari agen lain (f_{tm-i}) adalah sebagai berikut

$$f_{tm-i} = \sum_{j=1}^{n} (\min(0, (D_{i-j} - 2R)))^2 \quad (3.3)$$

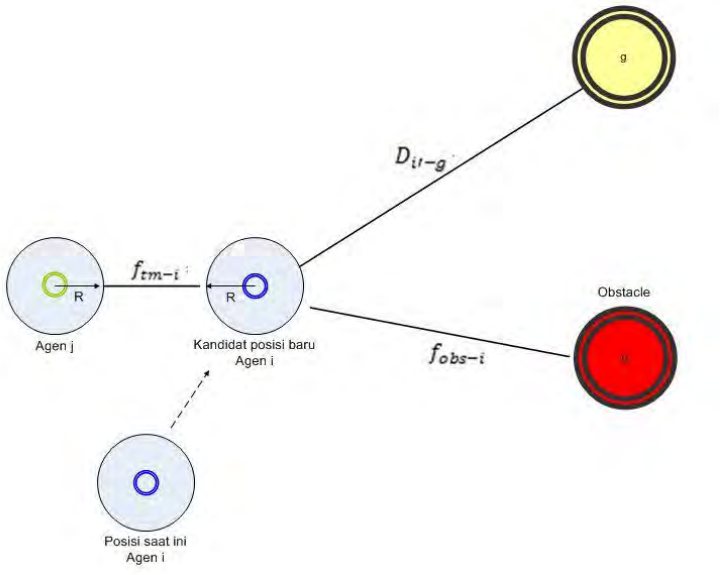
Jika $(\min(0, (D_{i-j} - 2R)))$ bernilai negative itu berarti kedua agen bertabrakan. Sehingga ketika nilai itu dikuadratkan, akan menghasilkan nilai positif. Ketika nilai ini dijumlahkan dengan fungsi obyektif akan menyebabkan nilai *fitness function* menjadi besar. Karena nilai *fitness function* lebih besar dari nilai sebelumnya, maka agen tidak

akan memilih posisi yang memiliki nilai fitness yang besar ini.

f_{tm-i} bernilai 0 jika semua lebah berada pada jarak yang cukup berjauhan, sebaliknya bernilai positif jika ada agen yang bertabrakan. Pada penelitian ini, setiap radius (R) setiap agen diberi nilai 1.

Untuk menghindari *obstacle*, hal yang sama akan diberlakukan sama seperti saat menghindari agen lain (team mates). Agar simulasi berjalan baik, diberikan bobot Δ = 5000.

$$f_{obs-i} = \sum_{j=1}^{n\_obs} \frac{\Delta}{(D_{i-obs})} \tag{3.4}$$



Gambar 3.11 Ilustrasi Perhitungan Fitnes Function

Setelah fungsi objektif menghitung jarak terdekat diketahui dan fungsi untuk memenuhi 2 constrain menghidari team mates dan obstacle telah ditentukan, maka untuk mencari nilai fitness dari posisi baru adalah sebagai berikut :

$$fit(p_i') = D_{i\prime-g} + f_{tm-i} + f_{obs-i} \tag{3.5}$$

dimana, p_i' adalah posisi baru agen ke-i, $D_{i\prime-g}$ menghitung jarak posisi baru agen ke target, f_{tm-i} untuk mengevaluasi apakah agen bertabrakan dengan agen lain dan f_{obs-i} untuk mengevaluasi apakah posisi baru agen adalah posisi *obstacle*. Semakin rendah nilai *fitness* menandakan bahwa semakin dekat posisi baru dengan target sekaligus menandakan posisi baru bukanlah posisi agen lain atau posisi *obsctacle*. Sehingga, agen akan selalu memilih posisi baru yang memiliki *fitness* yang lebih rendah dari *fitness* posisi yang saat ini ditempatinya. Artinya, fitness dianggap bagus jika nilainya rendah.

3.4. Path Planning

Simulasi pergerakan agen lebah dengan menggunakan algoritma ABC ini menampilkan animasi pergerakan agen lebah dari posisi asal menuju ke posisi target dengan arah dan posisi yang acak, tetapi tetap cenderung mengarah kepada target. Selama dalam perjalanannya menuju target, setiap agen memperhatikan posisi agen lain agar tidak terjadi tabrakan. Setiap agen juga memperhatikan posisi *obstacle* statis yang ada dan berusaha menghindarinya.

Path Planning adalah perencanaan rute perjalanan lebah dari posisi asal ke posisi target tanpa bertabrakan dengan sesama lebah dan tidak menabrak obstacle juga. *Path* atau rute yang direncanakan diharapkan dapat meniru pergerakan sekerumunan lebah yang berangkat bersama-sama dari suatu posisi awal, bergerak bersama-sama dan menyebar ke berbagai arah tetapi kemudian berkerumun di sekitar posisi target.

Setiap iterasi dalam ABC terdiri dari 3 langkah, yaitu : mengirim *employed* ke lokasi *foodsource* kemudian *employed* akan menghitung jumlah nectar yang terkandung dalam *foodsource* itu; *onlooker* menyeleksi *foodsource* berdasarkan informasi yang diperoleh dari *employed* dan memastikan *foodsource* mana yang akan dipilih; memunculkan *scout* kemudian mengirimnya ke kandidat *foodsource* baru. Iterasi akan dilakukan sampai dengan batasan yang ditentukan tercapai.

3.4.1. Inisialisasi

Pada tahap Inisiasi akan ditentukan sejumlah populasi lebah (BN), posisi awal lebah, posisi akhir lebah (target), vector dimensi (D) dan limit posisi baru yang akan diabaikan (L). Pada penelitian ini populasi lebah akan dianggap sebagai agen.

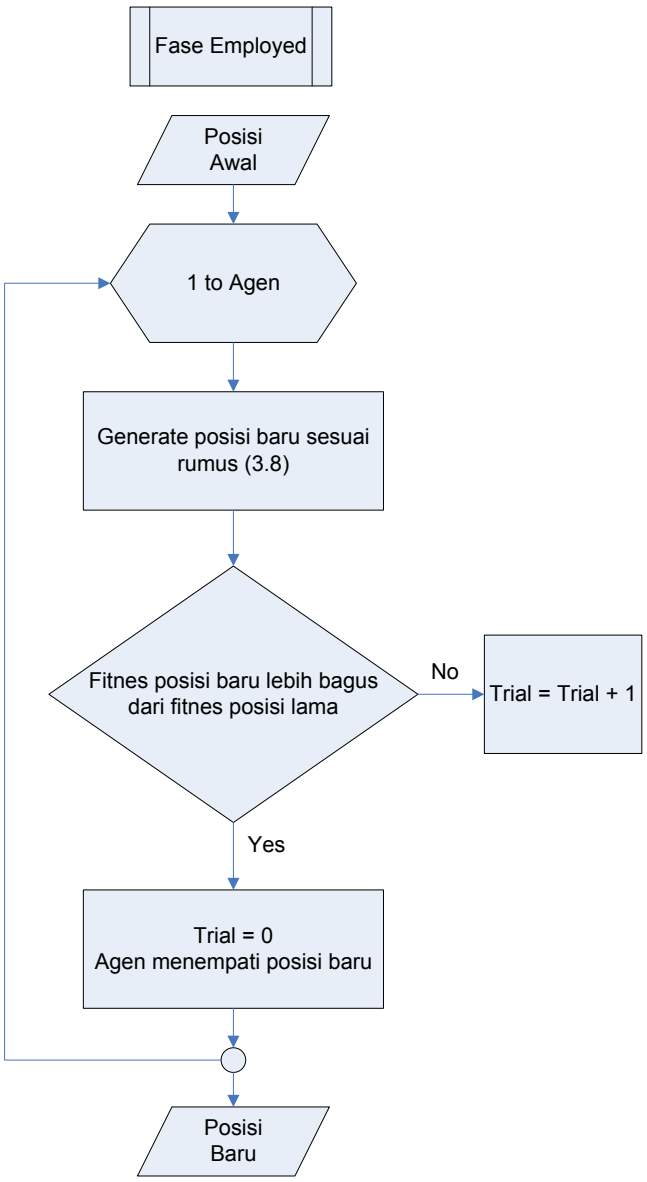
Input penelitian ini adalah posisi agen yang ditentukan secara acak di fase inisialisasi dan outpunya adalah rute saat bergerak dari posisi awal agen menuju posisi target.

3.4.2. Fase *Employed*

Pada fase *Employed*, setiap agen akan ditentukan posisi barunya secara acak dengan menggunakan rumus berikut ini,

$$p'_{ij} = p_{ij} + \emptyset(p_{ij} - p_{kj}) \quad (3.6)$$

dimana p_{ij} adalah poisisi lebah ke-*i* saat ini, p_{ij}' merupakan posisi baru lebah ke-*i* dan p_{kj} adalah posisi acak yang dipilih untuk mengurangi posisi lebah ke-*i*. Selain menggunakan p_{kj} yang dipilih secara acak untuk menentukan posisi baru, Φ juga berpengaruh membuat posisi baru yang acak. Φ merupakan angka acak antara -1 dan 1. *i* adalah jumlah posisi lebah, sedangkan $j \in D$ dan $k \in BN$, dipilih secara acak. Meskipun dipilih secara acak tetapi *k* tidak boleh sama dengan *i*.



Gambar 3.12 *Flowchart* Fase *Employed*

Setelah posisi baru untuk agen ditentukan, tahap berikutnya adalah mengevaluasi *fitness* dari posisi baru itu dengan menggunakan fungsi fitness (3.7). Jika nilai $fit(p'_{ij}) < fit(p_{ij})$ maka agen akan menempati posisi baru dan L_i. Sedangkan jika nilai $fit(p'_{ij}) > fit(p_{ij})$ maka posisi baru tidak akan ditempati dan $L_i = L_i + 1$.

Gambar 3.12 adalah *flowchart* pada fase *Employed*. Input dari fase *Employed* adalah posisi awal agen yang ditentukan acak di tahap Inisisaliasi. Dan outpunya adalah posisi baru dari agen. Posisi baru agen yang keluar dari fase *Employed* merupakan input di fase *Onlooker*.

3.4.3. Fase *Onlooker*

Output dari fase *Employed*, yaitu posisi baru agen, adalah input untuk fase *Onlooker*. Pada tahap onlooker ini, akan dihitung probabilitas *fitness* semua posisi agen. Nilai probabilitas *fitness* agen dihitung dengan menggunakan rumus :

$$prob\_value_i = \frac{fit(p_i\ )}{\sum_1^n fit(p_i\ )} \tag{3.7}$$

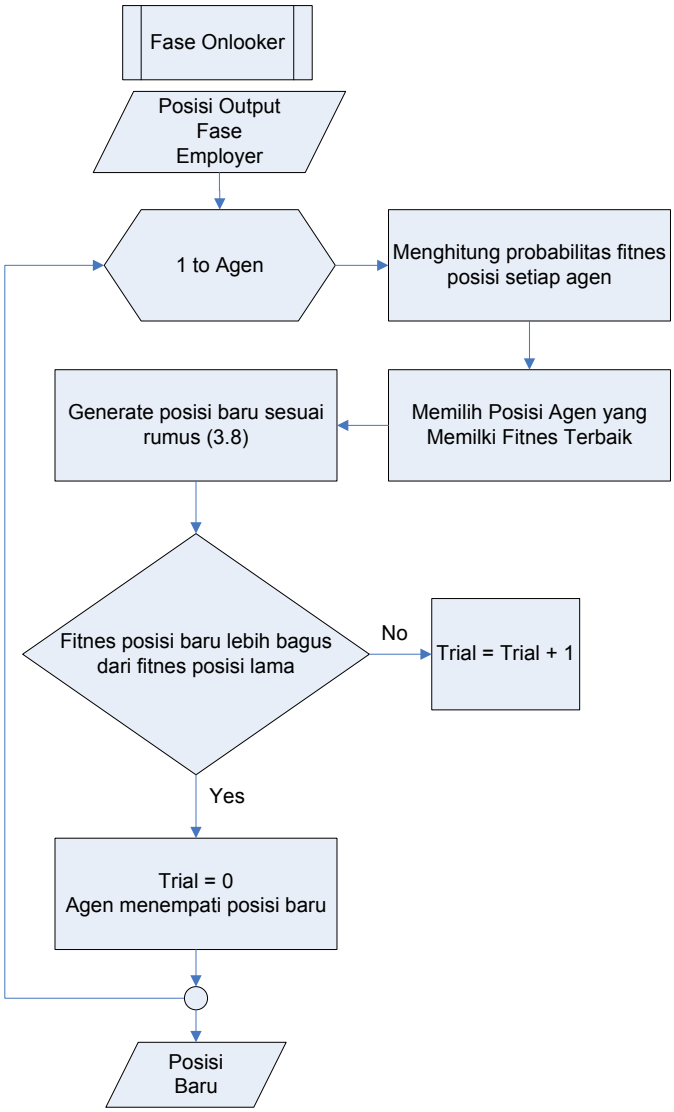
$fit(p_i\ )$ adalah nilai fitness dari agen ke-i. Untuk menentukan nilai probabilitas *fitness* maka *fitness* agen akan dibagi dengan total *fitness* semua agen ($\sum_1^n fit(p_i\ )$). Setelah semua nilai probabiltas *fitness* diketahui, akan dipilih agen yang memiliki nilai probabiltas terbaik. Dalam penelitian ini, probabilitas terbaik adalah probabilitas yang memiliki nilai terkecil. Posisi agen yang memiliki nilai probabilitas terbaik akan mempengaruhi posisi baru agen ke-*i*.

Pada fase *Employed* penentuan posisi baru agen ke-*i* ditentukan dengan mengurangkan posisi abaru agen ke-*i* dengan posisi agen ke-*k* yang dipilih secara acak. Pada fase *Onlooker*, penentuan posisi baru untuk agen ke-*i* ditentukan berdasarkan posisi agen yang memiliki nilai probabilitas tertinggi. Di bawah ini adalah persamaan yang digunakan untuk menentukan posisi baru untuk agen ke-*i* :

$$p_i' = p_{prob\_tertinggi} + \emptyset(p_{prob\_tertinggi} - p_k) \tag{3.8}$$

Setelah posisi baru agen ke-*i* ditentukan, sama seperti pada fase *Employed*, posisi baru agen ke-*i* itu akan dihitung fitnesnya dengan menggunakan rumus (3.5), rumus yang sama yang digunakan untuk menghitung fitness posisi agen pada fase Employed. Dan jika *fitness* posisi baru lebih baik dari posisi sekarang, maka agen ke-*i* akan menempati posisi barunya dan nilai Trial diset 0. Sebaliknya, jika nilai *fitness* posisi baru agen ke-*i* lebih jelek dari posisi agen ke-*i* saat ini, maka nilai Trial akan ditambah 1 dan posisi agen ke-*i* tidak berubah.

Gambar 3.13 adalah *flowchart* pada fase *Onlooker*. Input dari fase *Onlooker* berasal dari output pada fase *Employed*. Sama seperti pada fase *Employed*, output dari fase *Onlooker* ini adalah posisi baru dari agen. Untuk setiap agen yang tidak berhasil menemukan posisi baru hingga L kali, maka pada fase Scout, akan ditentukan posisi baru secara acak untuk agen tersebut.



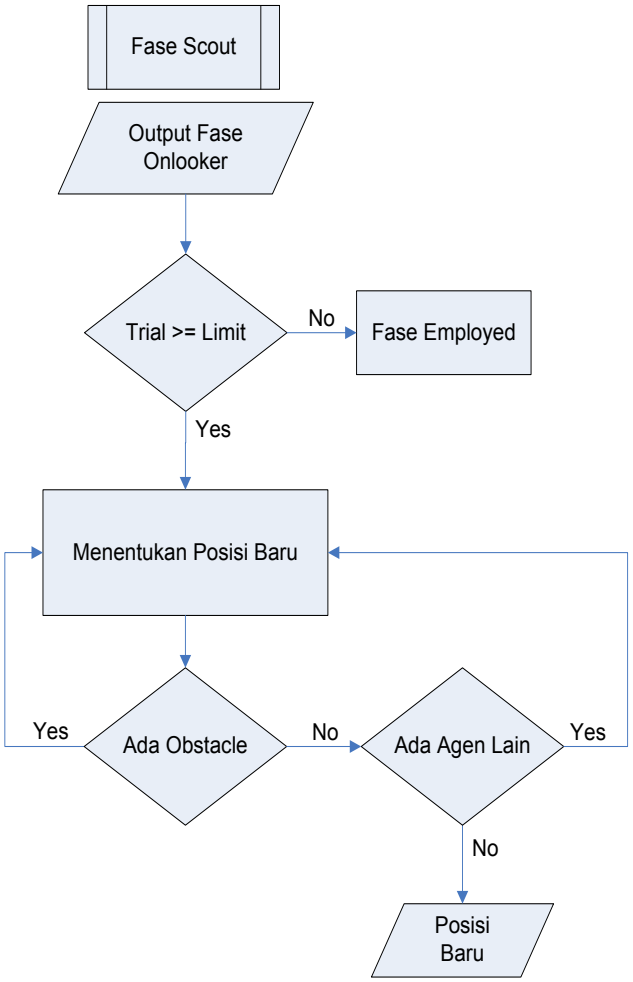
Gambar 3.13 *Flowchart* Fase Onlooker

3.4.4. Fase Scout

Jika setelah sekian kali pencarian posisi baru tidak ditemukan posisi baru yang lebih baik bagi agen, maka lebah memasuki fase *scout*. Batasan jumlah iterasi maksimal yang dilakukan, ditentukan nilainya pada *variable limit*. Jadi jika trial >= limit maka lebah memasuki fase *scout*.

Pada fase ini posisi baru lebah ditentukan secara acak tanpa memperhatikan *fitness function*-nya, namun tetap memperhitungkan *constraint* yang ada yakni tidak boleh menabrak *obstacle* atau sesama lebah. Posisi baru agen yang memiliki nilai trial lebih besar atau sama dengan nilai limit ditentukan dengan persamaan berikut ini :

$$p_i' = p_{min} + rand(0,1) * (p_{max} - p_{min}) \quad (3.9)$$



Gambar 3.14 *Flowchart* Fase Scout

Posisi agen ke-*i* akan ditentukan berdasarkan posisi agen yang memiliki fitness paling tinggi dan agen yang memiliki fitness yang paling rendah. Agen yang memiliki nilai fitness tinggi menandakan bahwa posisinya paling dekat dengan target (p_{min}). Dan agen yang memiliki nilai fitness paling rendah menandakan bahwa posisinya paling jauh dengan target (p_{max}).

Lebah tidak boleh terjebak/terhenti pada posisi/kondisi tertentu. Lebah dipaksa untuk keluar dari kondisi terjebaknya dengan membuat kandidat solusi (posisi) baru secara acak. Jika posisi baru tidak bisa diambil karena menabrak obstacle atau sesama lebah, maka diulang lagi sampai ditemukan posisi baru yang bebas dari constraint. Gambar 3.14 adalah *flowchart* fase *scout* dalam menentukan posisi acak agen yang memiliki trial lebih besar atau sama dengan limit.

3.5. Skenario Percobaan

Tujuan dari penelitian ini adalah memperoleh sekelompok NPC cerdas yang mampu bergerak menuju target dengan formasi acak dari arah yang berbeda-beda tanpa menabrak agen lain dan kelompoknya mampu menghindari obstacle statis. Untuk memenuhi tujuan tersebut, penelitian ini melakukan percobaan dengan 5 skenario yaitu, menciptakan pergerakan agen dari posisi asal menuju ke posisi target, penambahan perhitungkan posisi agen yang lain untuk menghindari tabrakan, penambahan perhitungkan menghindari 1 obstacle statis agar terhindar dari tabrakan, menghindari 3 obstacle statis, dan perhitungan waktu ketika menghindari 5 *obstacle* statis dengan variasi jumlah lebah 10 – 100.

BAB 4
PERCOBAAN DAN SIMULASI KOMPUTER

Pergerakan sekelompok agen yang telah diuraikan pada Bab 3 diimplementasikan dalam simulasi komputer dengan menggunakan Matlab R2013. Simulasi ini melibatkan 10 agen (SN), satu target diam di koordinat (500, 1000, 1000) , dan beberapa *obstacle* statis. Limit (L) pencarian posisi baru untuk agen adalah 5, artinya jika lebih dari 5 kali percobaan pencarian posisi baru tidak ditemukan kandidat posisi baru untuk agen, maka *scout* akan mencarikan posisi baru secara acak. Radius agen (R) = 5. Agen akan mencari posisi baru sebanyak 100 iterasi.

Input untuk setiap percobaan pada skenario 1-4 berupa posisi awal 10 agen yang ditentukan secara *random* di ruang 3 dimensi. Sedangkan outputnya rute pergerakan agen dari posisi awal ke posisi akhir mendekati target sebanyak 100 iterasi. Input untuk percobaan pada skenario 5 berupa posisi awal 10-100 agen dan dilakukan dalam 50 iterasi.

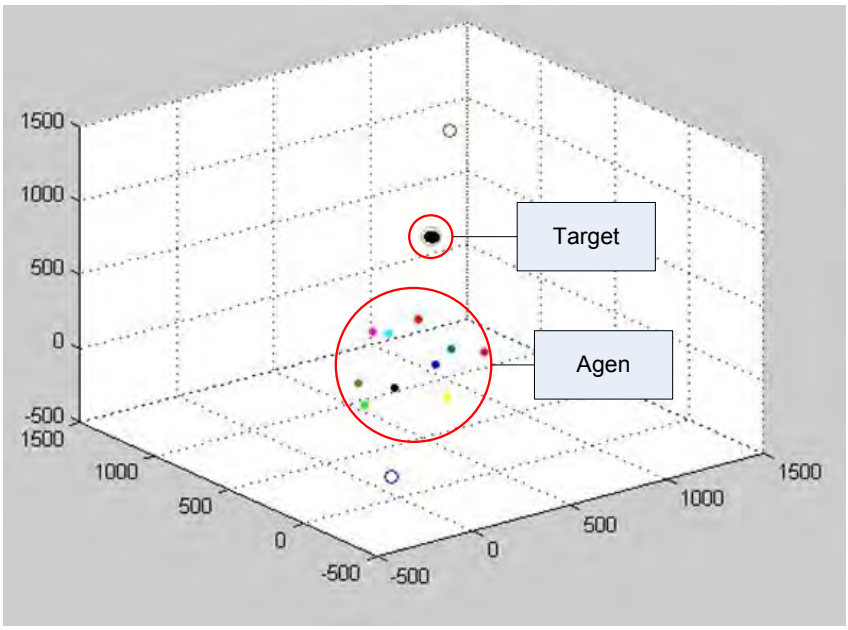
Untuk lebih memudahkan analisa, percobaan dan simulasi ini dibagi dalam 4 bagian yakni : pergerakan agen mencari jalur terpendek menuju target dengan menggunakan algoritma ABC, pergerakan agen dengan menghindari tabrakan dengan sesama agen, penambahan satu *obstacle* statis pada lingkungan agen dan penambahan 3 *obstacle* statis pada lingkungan agen.

4.1. Pergerakan Agen Mencari Jarak Terdekat

Percobaan pertama dilakukan untuk menguji kemampuan agen dalam mencari jarak terpendek menuju target dengan formasi acak menggunakan algoritma ABC. Dalam simulasi ini, lingkungan yang digunakan adalah lingkungan 3 Dimensi berupa sebuah kubus yang didalamnya terdapat 10 agen, satu target. Tujuan utama dari simulasi ini adalah menggerakkan sekelompok agen dari posisi awal ke posisi target dengan lintasan yang acak dan tidak dapat diprediksi arahnya.

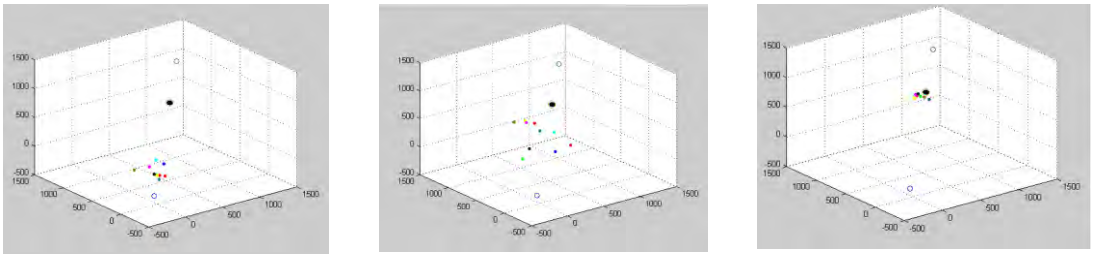
Titik awal atau posisi awal lebah adalah di sekitar lingkaran berwarna biru yang berada pada koordinat di sekitar titik (0,0). Target yang harus dicapai adalah bulatan

berwarna hitam dengan garis tepi berwarna kuning yang berada di koordinat (500, 1000, 1000). Gambar 4.1 menunjukkan bulatan warna-warni yang melambangkan agen dan 1 bulatan hitam yang lebih besar menggambarkan target.



Gambar 4.1 Lingkungan 3D Simulasi Agen

Agen yang dilibatkan dalam simulasi ini ada 10 buah, yang dilambangkan dengan bulatan kecil warna-warni. Pembedaan warna dimaksudkan untuk mempermudah pengamatan pergerakan tiap agen lebah selama simulasi berlangsung.



Posisi awal agen

Bergerak bersama menuju target

Memposisikan diri mengerumuni target

Gambar 4.2 Simulasi Pergerakan Lebah

Pergerakan agen dapat dikategorikan dalam 3 bagian yakni keberangkatan, perjalanan dan kedatangan. Pada bagian pertama agen mengumpul pada lokasi keberangkatan. Selama perjalanan menuju target, masing-masing agen berpencar menempati posisi masing-masing sesuai kaidah pada algoritma ABC. Pada bagian

terakhir, agen mengerumuni target sehingga posisinya kembali berkumpul pada titik tujuan. Gambar 4.2 adalah ilustrasi dari pergerakan agen dari titik keberangkatan sampai dengan mendatangi target.

Fungsi tujuan atau *fitness function* untuk pergerakan agen adalah menempatkan agen pada posisi sedekat mungkin dengan posisi target. Dengan kata lain fungsi tujuan yang hendak dicapai adalah meminimalkan jarak antara tiap agen dengan target. Jadi setiap kali iterasi diharapkan posisi agen semakin mendekati target atau jarak agen ke target semakin kecil. Tabel 4.1 adalah posisi awal agen di percobaan pertama.

Tabel 4.1 Posisi awal agen di percobaan ke-1

Agen	posX	posY	posZ
1	47.2086	-51.358	23.3611
2	60.8688	70.5889	-69.6432
3	-55.952	68.575	52.3694
4	62.0064	-2.1937	65.099
5	19.8539	45.0421	26.8103
6	-60.3689	-53.717	38.661
7	-33.2253	-11.7358	36.4699
8	7.0322	62.3603	-16.1659
9	68.626	43.8311	23.3217
10	69.7333	68.9239	-49.322

Percobaan pada scenario ini menunjukkan bahwa tidak semua agen menemukan nilai *fitness* terbaiknya pada iterasi yang sama. Tabel 4.2 adalah data di iterasi ke berapa suatu agen mencapai nilai terbaiknya.

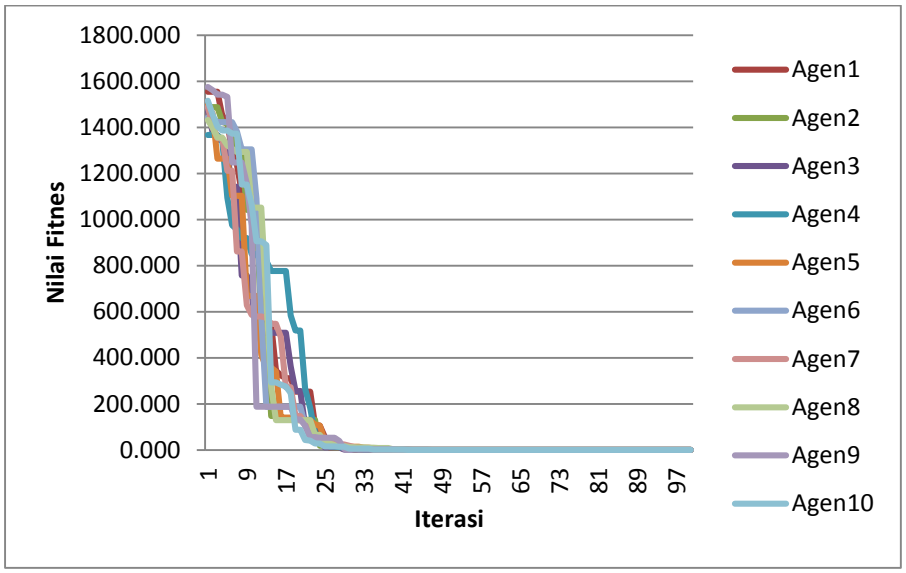
Tabel 4.2 *Fitness* terbaik setiap agen pada percobaan petama

Agen	Nilai Fitnes Terbaik	Di Iterasi ke-
1	0,000065	87
2	0,000083	89
3	0,000089	81
4	0,000034	84
5	0,000034	95
6	0,000034	90
7	0,000117	96
8	0,000032	82
9	0,000046	81
10	0,000032	84

Dari data percobaan pertama diperoleh grafik untuk mengetahui visualisai

pengerucutan fitness function dari semua agen. Gambar 4.3 adalah grafik fitness function semua agen di percobaan pertama ini.

Gambar 4.3 menunjukkan bahwa agen mulai konvergen di iterasi 30-an. Namun agen sudah mencapai fitness terbaik di iterasi ke-47.



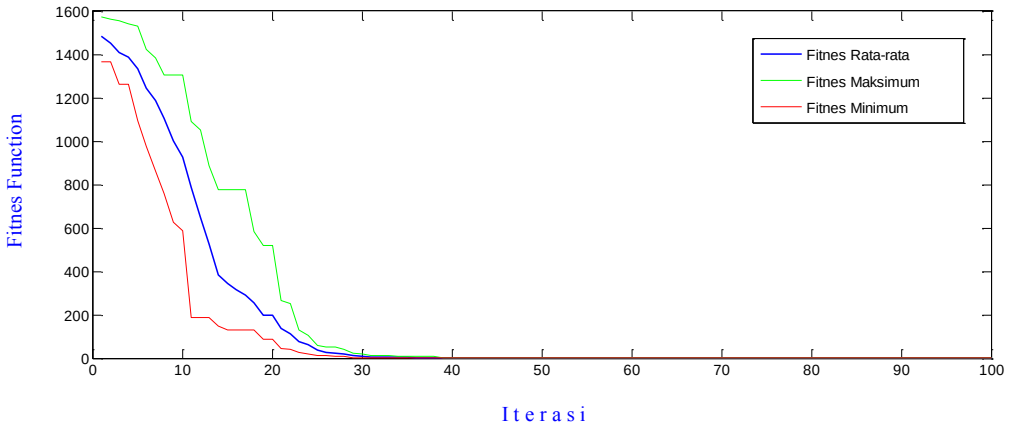
Gambar 4.3 Grafik Fitness Function 10 Agen di Percobaan ke-1

Tabel 4.3 adalah tabel posisi agen pada iterasi ke-47. Koordinat (x, y, z) setiap agen menunjukkan bahwa posisi agen sudah hampir satu posisi dengan posisi target yaitu di titik (500, 1000, 1000).

Tabel 4.3 Posisi agen di iterasi ke-47

Agen	posX	posY	posZ
1	499.7193	999.3256	999.926
2	500.0701	999.905	999.8643
3	499.9537	1000.011	999.8729
4	500.1648	999.9884	999.6915
5	500.0094	999.8229	999.8755
6	499.7366	999.9228	1000.112
7	500.3891	999.6759	999.9234
8	500.0763	1000.185	999.3313
9	499.6633	999.6186	999.9881
10	499.7262	1000.007	1000.182

Rata-rata nilai *fitness function* mencapai *fitness* terbaik di iterasi ke 95 dengan nilai fitness 0,000057.

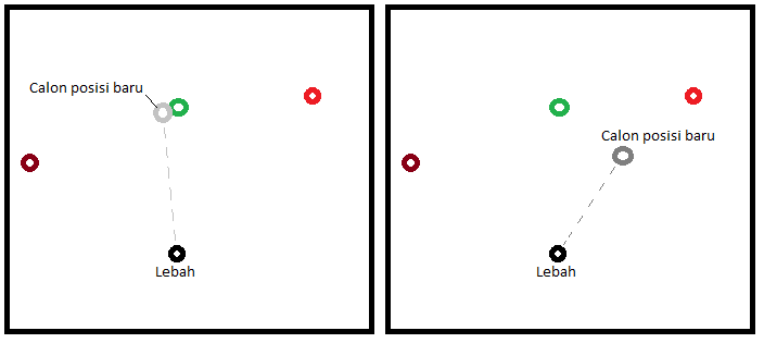


Gambar 4.4 Grafik Fitness Rata-rata Percobaan ke-1

Gambar 4.4 adalah grafik nilai maksimum, nilai minimum dan nilai rata-rata *fitness function* percobaan skenario pertama. Pada Gambar 4.4 terlihat terdapat tiga garis berwarna hijau, merah dan biru. Garis biru adalah jarak rata-rata setiap agen ke target. Dari grafik tersebut terlihat jelas grafiknya konvergen posisi agen di sekitar target pada kisaran iterasi ke-30. Garis hijau menunjukan jarak terjauh antara agen dan target. Sedangkan garis berwarna merah adalah jarak terdekat antara agen dengan target. Artinya ketika iterasi dilakukan sebanyak 30 kali posisi setiap agen sudah sangat dekat berada di sekitar target.

4.2. Pergerakan lebah dengan menghindari tabrakan dengan sesama lebah

Mengikuti algoritma ABC, pergerakan agen tidak membentuk garis lurus atau formasi lain yang bisa dibaca. Pada setiap iterasi, arah dan kecepatan agen dapat berganti sewaktu-waktu. Jadi untuk menghindari tabrakan dengan sesama agen, juga tidak dapat diramalkan sebelum eksekusi pergerakan agen terjadi. Yang dilakukan untuk menghindari benturan adalah dengan mengecek posisi baru suatu agen beririsan dengan posisi agen lain atau tidak. Jika posisinya beririsan maka agen tersebut tertahan untuk mengupdate posisinya dan menunggu iterasi berikutnya untuk menentukan calon posisi baru yang tentunya juga harus dicek apakah menabrak agen lain atau tidak. Jika calon posisi baru benar-benar kosong maka agen tersebut dapat mengupdate posisinya yang baru.



Gambar 4.5 Pencarian Posisi Baru Menghindari Posisi Agen Lain

Gambar 4.5 adalah simulasi kondisi agen pada suatu tahap iterasi. Gambar sebelah kiri adalah ketika calon posisi baru ternyata beririsan dengan posisi agen lain, maka agen harus membatalkan rencana pergerakannya. Pada gambar sebelah kanan terlihat bahwa calon posisi baru tidak beririsan dengan posisi agen lain, maka agen dapat mengupdate posisinya dan berpindah ke lokasi baru tersebut.

Tabel 4.4 Posisi Awal Agen Percobaan ke-2

Agen	posX	posY	posZ
1	-68.0109	1.248361	-36.2839
2	128.9281	65.37248	73.15502
3	7.400476	60.41138	-53.3141
4	-36.5581	70.86714	94.21494
5	9.490685	78.57055	47.13939
6	50.83025	74.16214	-52.3457
7	24.45367	-39.3251	82.97112
8	-90.8309	-4.25929	-38.7002
9	-20.1754	1.865708	188.9179
10	56.40319	34.71699	39.13568

Sama dengan percobaan pertama, posisi awal semua agen pada percobaan ini juga ditentukan secara *random*. Tabel 4.4 adalah data posisi awal semua agen untuk percobaan ke-2 dengan tujuan mencari rute terdekat namun tetap menjaga jarak antar agan agar tidak bertabrakan.

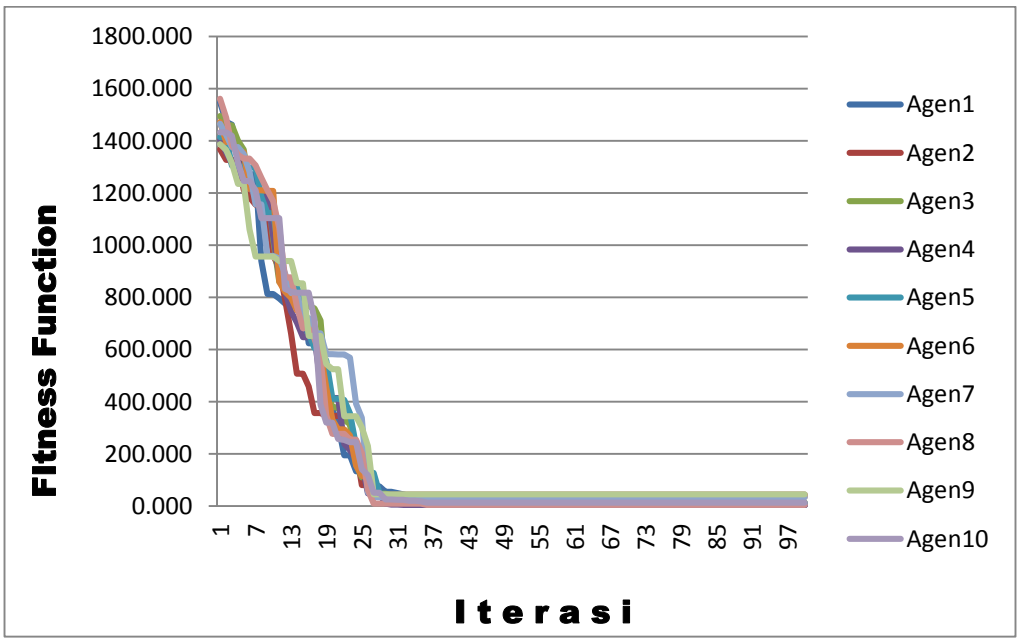
Setelah program simulasi pergerakan agen menuju posisi target dengan memenuhi *constrain* menghindari tabrakan dengan agen lain dalam kelompok dijalankan, maka diperoleh data *fitness function* setiap agen selama 100 iterasi. Di percobaan skenario kedua, agen lebih cepat menemukan *fitness* terbaiknya dibandingkan dengan percobaan untuk skenario pertama. Tetapi nilai fitness pada percobaan untuk

skenario kedua jauh lebih besar dari pada fitness agen pada percobaan untuk skenario pertama. Tabel 4.5 adalah data pada iterasi ke berapa agen menemukan *fitness* terbaiknya pada percobaan kedua.

Tabel 4.5 Fitnes terbaik setiap agen pada percobaan kedua

Agen	Nilai Fitnes Terbaik	Di Iterasi ke-
1	23,968790	36
2	44,272276	29
3	13,313603	35
4	2,520272	33
5	11,005869	35
6	35,821548	29
7	34,490186	27
8	5,338960	36
9	45,271449	27
10	12,865739	36

Gambar 4.6 menunjukkan bahwa semua agen mulai konvergen antara iterasi ke-25 dan iterasi ke-30. Percobaan pada skenario kedua menunjukkan bahwa pada iterasi ke-36 sampai dengan iterasi ke-100 nilai *fitness* setiap agen sudah tetap. Artinya agen mulai menemukan posisi paling optimal sedekat mungkin dengan posisi target pada iterasi ke 36.



Gambar 4.6 Grafik *Fitnes Function* Agen di Percobaan ke-2

Tabel 4.6 adalah data yang berisi posisi agen pada iterasi ke 36. Data ini

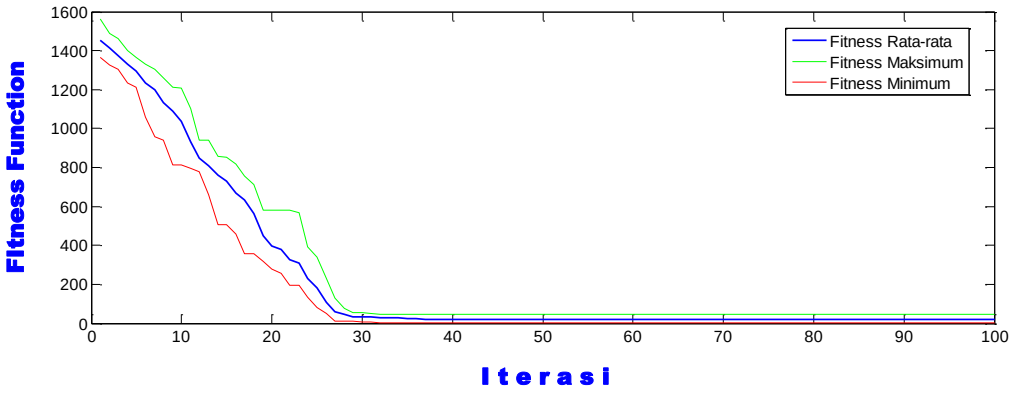
menunjukkan bahwa pada iterasi ke-38 kesepuluh agen sudah berada di sekitar target yang berada di koordinat (500, 1000, 1000).

Tabel 4.6 Posisi agen di iterasi ke-36

Agen	posX	posY	posZ
1	519.2344	987.9209	1007.657
2	500.3959	994.5268	1002.415
3	504.6637	988.3793	1004.524
4	501.1721	994.0435	1002.456
5	508.2049	993.0034	1002.204
6	500.4173	996.6464	1001.755
7	501.2663	993.3409	1002.845
8	501.8129	995.6713	1002.546
9	501.2075	992.1326	1002.803
10	505.3446	989.4062	1004.973

Sampai dengan akhir iterasi, tidak ada satupun *fitness* posisi agen yang mendekati nilai 0. Hal ini menunjukkan bahwa meskipun berusaha selalu mencari posisi sedekat mungkin dengan target, tetapi agen juga selalu mengevaluasi posisi dirinya dengan agen lain. Sehingga jika ada 2 atau lebih agen yang mempunyai nilai *fitness* 0, hal itu membuktikan bahwa semua posisi agen-agen itu saling berhimpitan, sehingga *constrain* untuk menghindari tabrakan dengan agen lain tidak terpenuhi.

Setelah mendapatkan data *fitness function* semua agen selama 100 iterasi, kemudian dicari *fitness* maksimum, *fitness* minimum dan *fitness* rata-rata dari kesepuluh agen tersebut. Gambar 4.7 merupakan grafik nilai rata-rata, maksimum dan minimum nilai fitness 10 agen.



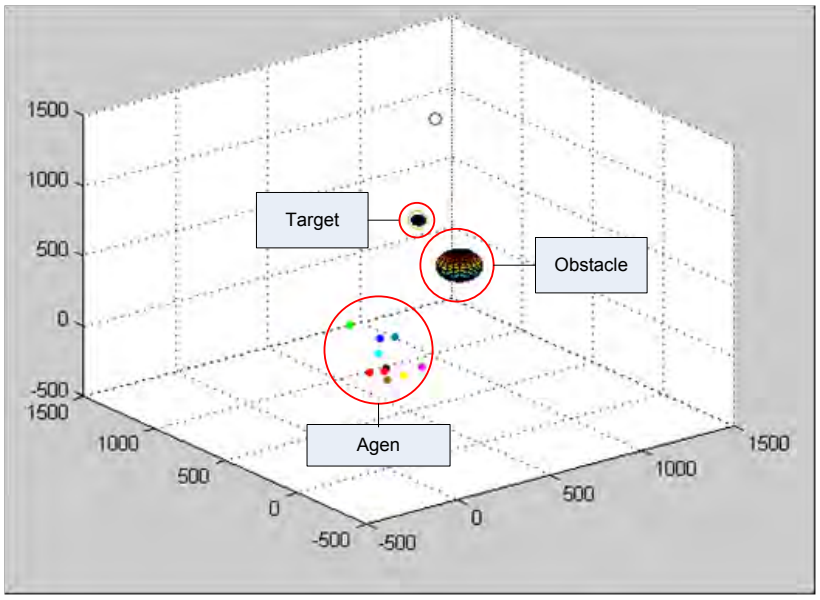
Gambar 4.7 Grafik Fitness Rata-rata, Minimum dan Maksimum

Dari gambar 4.7 dengan memperhatikan *fitness* posisi agen terjauh dan *fitness* posisi agen terdekat dengan target, terlihat bahwa kawanan agen tidak terlalu menyebar dan tetap konvergen pada iterasi yang tidak terlalu banyak, yaitu pada kisaran iterasi ke 30. Rata-rata nilai *fitness* terbaik dicapai di iterasi ke-37 dengan nilai 21,74828.

4.3. Penambahan *Obstacle* Statis

Penambahan *obstacle* akan membuat kompleksitas simulasi menjadi bertambah. Agen harus mendatangi target dengan memperhatikan adanya *obstacle* dan berusaha menghindarinya. Saat bergerak menghindari *obstacle*, agen juga harus tetap memperhitungkan posisi agen lain agar tidak terjadi benturan antar agen. seperti halnya koloni lebah, meskipun mereka bergerak secara acak namun mereka selalu menjaga jarak antar agen dan halangan yang ditemuinya.

Gambar 4.8 adalah gambar sekelompok agen dalam lingkungan dengan satu buah bola yang dianggap sebagai *obstacle* statis didalamnya.



Gambar 4.8 Agen dalam Lingkungan dengan 1 Obstacle

Prosedur pengecekan *obstacle* sama dengan pengecekan sesama teman. Jika calon posisi baru beririsan dengan posisi obstacle maka calon posisi baru tersebut tidak diambil. Agen menunggu iterasi berikutnya untuk kembali mencoba lagi meng*update* posisinya. Tabel 4.7 adalah data posisi awal untuk percobaan skenario 3.

Tabel 4.7 Posisi Awal Agen Percobaan ke-3

Agen	posX	posY	posZ
1	30.9069	-9.1883	-33.5962
2	-70.2251	-17.7662	26.9554
3	-33.4616	39.8275	23.2647
4	-68.0743	44.28	-50.6082
5	-60.4302	-46.9691	-57.1503
6	48.5187	-1.5353	-0.2454
7	29.2243	-8.1621	68.9616
8	-27.4351	21.947	-23.9421
9	67.5333	31.4047	12.7902
10	-69.8331	38.203	-41.4282

Dari hasil percobaan ternyata penambahan *obstacle* tidak banyak mempengaruhi performa karena memang hanya ada penambahan pengecekan satu objek lain serupa dengan pengecekan sesama teman.

Tabel 4.8 Fitnes terbaik setiap agen pada percobaan ketiga

Agen	Nilai Fitnes Terbaik	Iterasi ke-
1	171,116660	46
2	22,090933	31
3	17,240706	39
4	20,470370	31
5	48,357957	38
6	46,914931	37
7	9,826067	36
8	27,680014	24
9	25,490617	34
10	35,040434	39

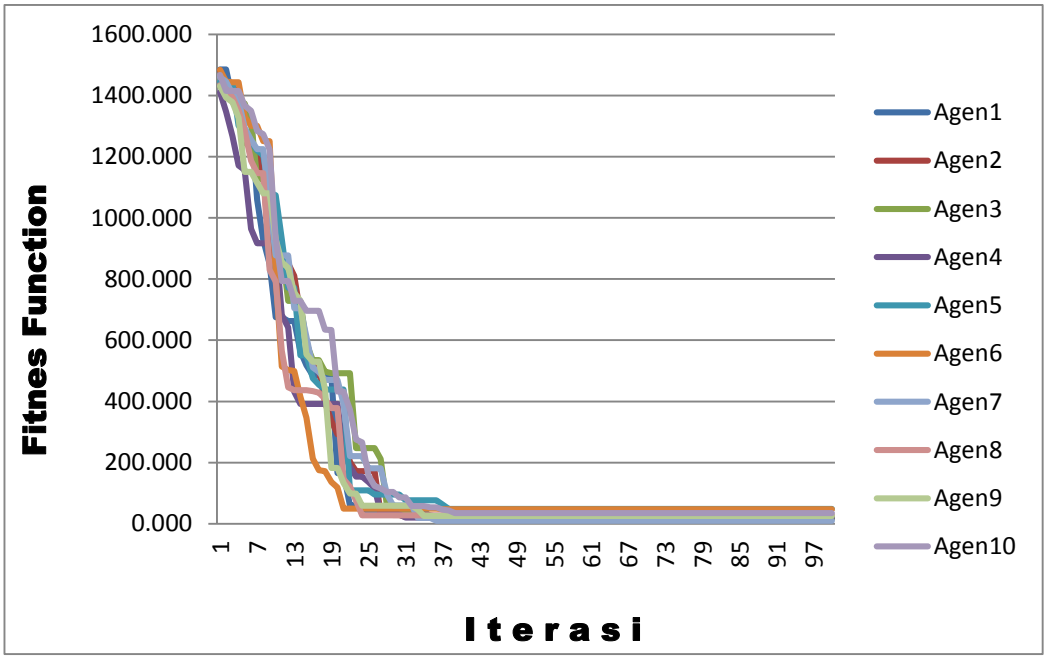
Pada percobaan ini fitness terbaik dimiliki oleh agen 7 pada iterasi ke-36 dengan nilai *fitness* 9,826067. Di bawah ini adalah data nilai *fitness* terbaik setiap agen. Tabel 4.8 adalah *fitness* terbaik setiap agen di percobaan ke-3 ini.

Percobaan pada penambahan obstatcle statis menunjukkan bahwa nilai *fitness function* tidak berubah mulai iterasi ke-47. Tabel 4.9 menunjukkan posisi agen pada iterasi ke-47.

Tabel 4.9 Posisi Agen di Iterasi ke-47

Agen	posX	posY	posZ
1	501.1352	996.231	989.1674
2	471.6185	996.9758	986.6745
3	471.7184	997.1236	991.7772
4	471.4717	996.7972	988.8634
5	471.4719	996.8287	987.1065
6	471.4611	996.7187	987.4774
7	471.636	997.0298	990.4351
8	471.5754	996.9062	986.762
9	471.4982	996.8824	987.9727
10	471.486	996.8983	987.083

Gambar 4.9 adalah grafik *fitness fanction* dari semua agen. Grafik terebut menunjukkan bahwa kawanan agen konvergen di iterasi kisaran 40, tidak banyak berbeda dengan yang tanpa *obstacle*.

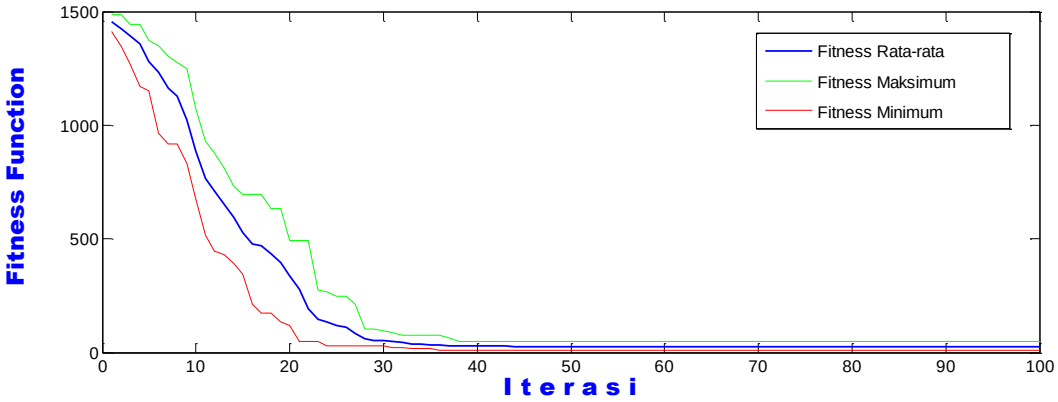


Gambar 4.9 Grafik Konvergensi Kawanan Agen dengan 1 Obstacle

Sama dengan kasus menghindari tabrakan dengan agen lain, tidak ada 2 agen atau lebih yang memiliki *fitness function* 0 karena jika mereka mempunyai *fitness* mendekati 0 maka posisi mereka berhimpitan. Dan jika itu terjadi maka kondisi itu tidak memenuhi *constrain* menghindari agen lain.

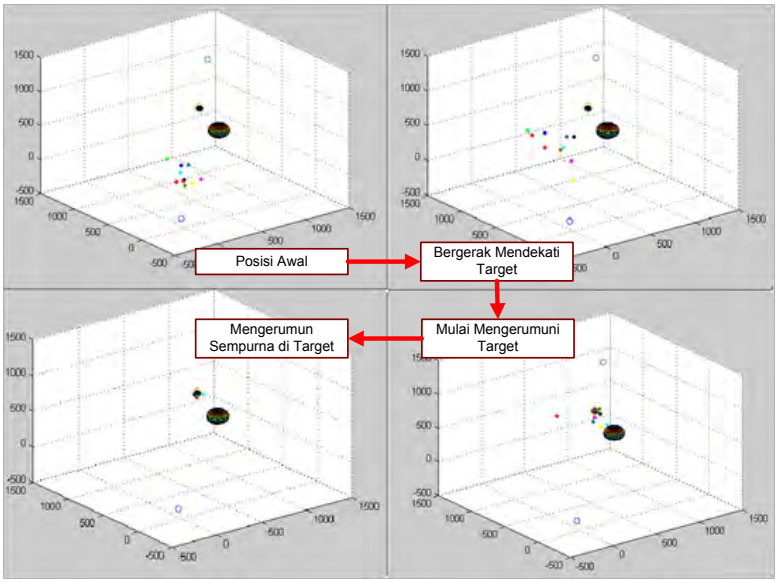
Gambar 4.10 adalah grafik nilai maksimum, nilai minimum dan nilai rata-rata *fitness function* agen. Nilai rata-rata *fitness* terbaik semua agen ada di iterasi ke-46

dengan nilai *fitness* 27,022869



Gambar 4.10 Fitnes Maksimum, Minimum dan Rata-rata 1 Obstacle

Gambar 4.11 adalah ilustrasi pergerakan agen dengan satu *obstacle*. Mulai dari kiri atas searah putaran jarum jam, urutan simulasinua adalah sebagai berikut : mulai dari posisi awal, bergerak bersama-sama, sampai dengan mengerumuni target.

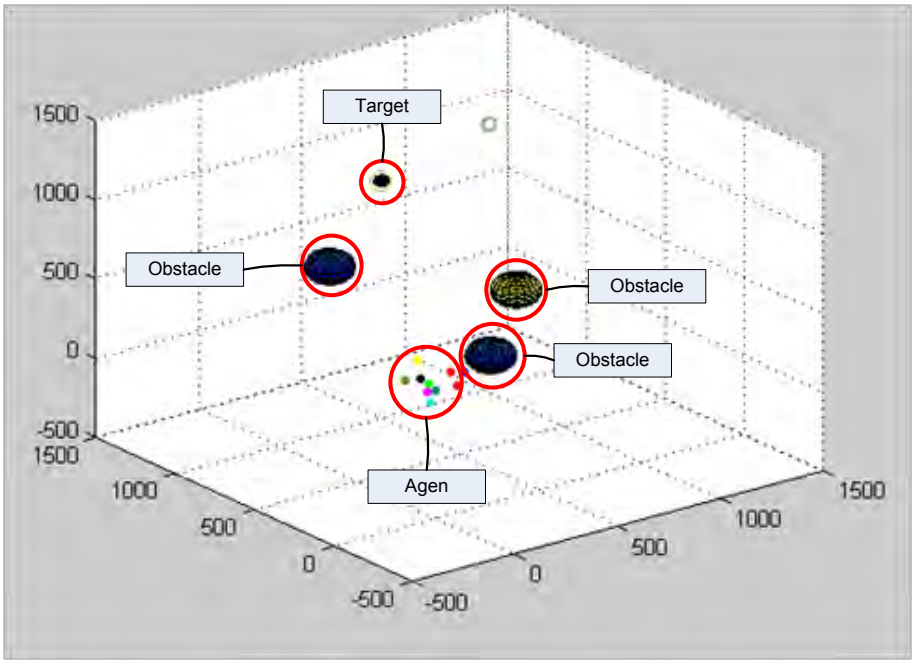


Gambar 4.11 Simulasi Pergerakan Agen dengan 1 Obstacle

4.4. Penambahan 3 obstacle statis

Penambahan beberapa *obstacle* statis sebenarnya tidak mempengaruhi kerumitan program karena hanya menambah perulangan aktifitas pengecekan tanpa penambahan prosedur baru. Pengecekan dilakukan terus menerus sampai lebah mencapai target yang diharapkan. Gambar 4.12 adalah ilustrasi lingkungan yang diberi beberapa *obstacle*

statis.



Gambar 4.12 Lingkungan Agen dengan 3 *Obstacle*

Tabel 4.10 Posisi Awal Agen di Percobaan ke-4

Agen	posX	posY	posZ
1	37.6901	51.1076	-22.2511
2	-36.7357	-36.8577	49.6243
3	0.8936	47.1427	12.7896
4	29.8615	-38.4713	7.4585
5	58.6355	64.3895	62.579
6	68.8937	-22.5024	-32.1241
7	7.0823	-45.5107	38.58
8	-54.2063	-37.3374	38.0594
9	-52.6059	17.4067	-17.9331
10	-36.3738	-4.0067	10.1732

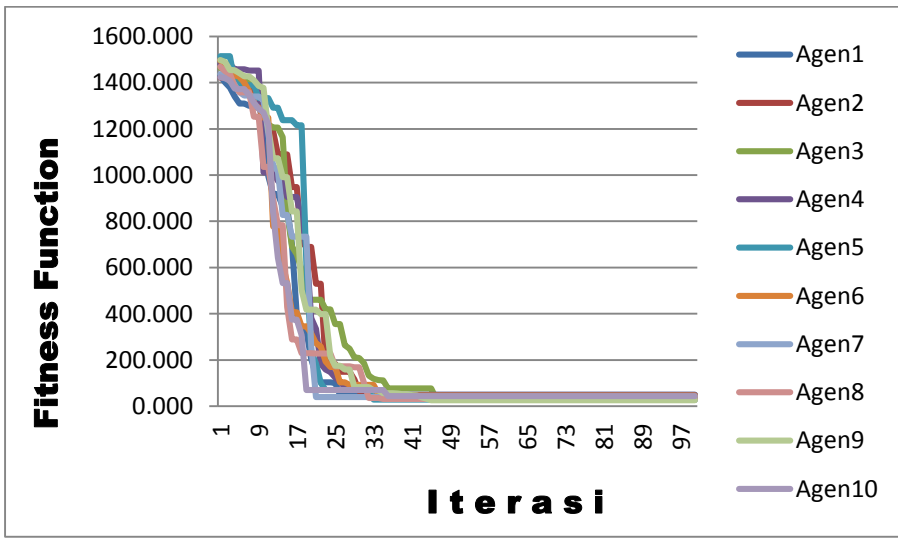
Penambahan beberapa *obstacle* cukup mempengaruhi performa agen dalam mencari jarak terdekat untuk menuju target. Kawanan agen konvergen pada iterasi ke 40, lebih banyak daripada percobaan-percobaan sebelumnya. Hal ini disebabkan oleh radius *obstacle* yang lebih besar daripada radius sesama agen. Ketika ada beberapa *obstacle* pada jarak yang cukup berdekatan, menyebabkan ruang yang dapat ditempati atau dilintasi lebah menjadi sempit sehingga harus lebih banyak mengulang iterasi untuk mendapatkan kandidat posisi baru yang dapat diambil. Setiap agen mencapai *fitness*

optimal pada iterasi yang berbeda. Tabel 4.11 adalah data nilai *fitness* terbaik setiap agen.

Tabel 4.11 Fitnes terbaik setiap agen pada percobaan ketiga

Agen	Nilai Fitness Terbaik	Iterasi ke-
1	49,27669	26
2	28,82417	48
3	34,67761	46
4	49,58943	36
5	29,56495	33
6	46,46492	35
7	26,83735	46
8	31,42283	35
9	25,70339	45
10	44,28971	36

Gambar 4.13 adalah grafik *fitness function* semua agen. Grafik itu menunjukkan adanya penundaan konvergensi kawanan lebah yang cukup signifikan. Ketika tidak ada *obstacle* atau ada satu obstacle saja, kawanan lebah konvergen pada iterasi ke 30 dan 35. Sementara ketika jumlah obstacle menjadi tiga, kawanan lebah konvergen pada iterasi ke 46. Tabel 4.12 adalah tabel posisi agen pada iterasi ke-46

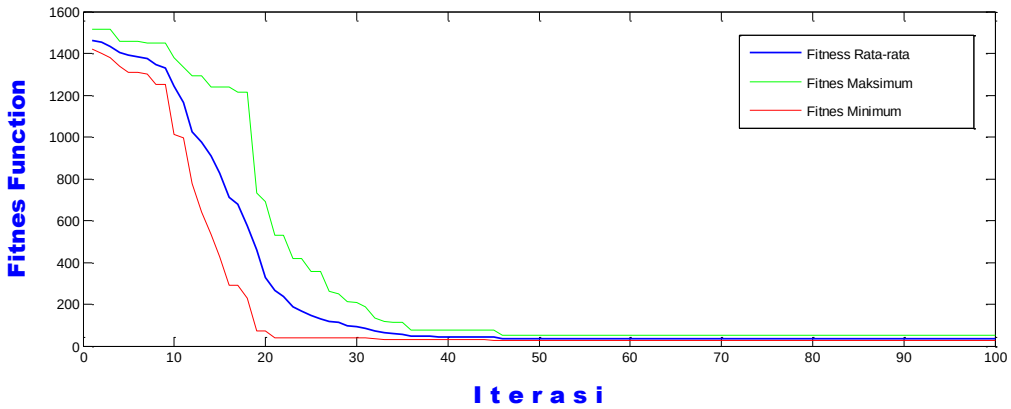


Gambar 4.13 Fitness Function Menghindari 3 Obstacle

Tabel 4.12 Posisi Agen di Iterasi ke-46

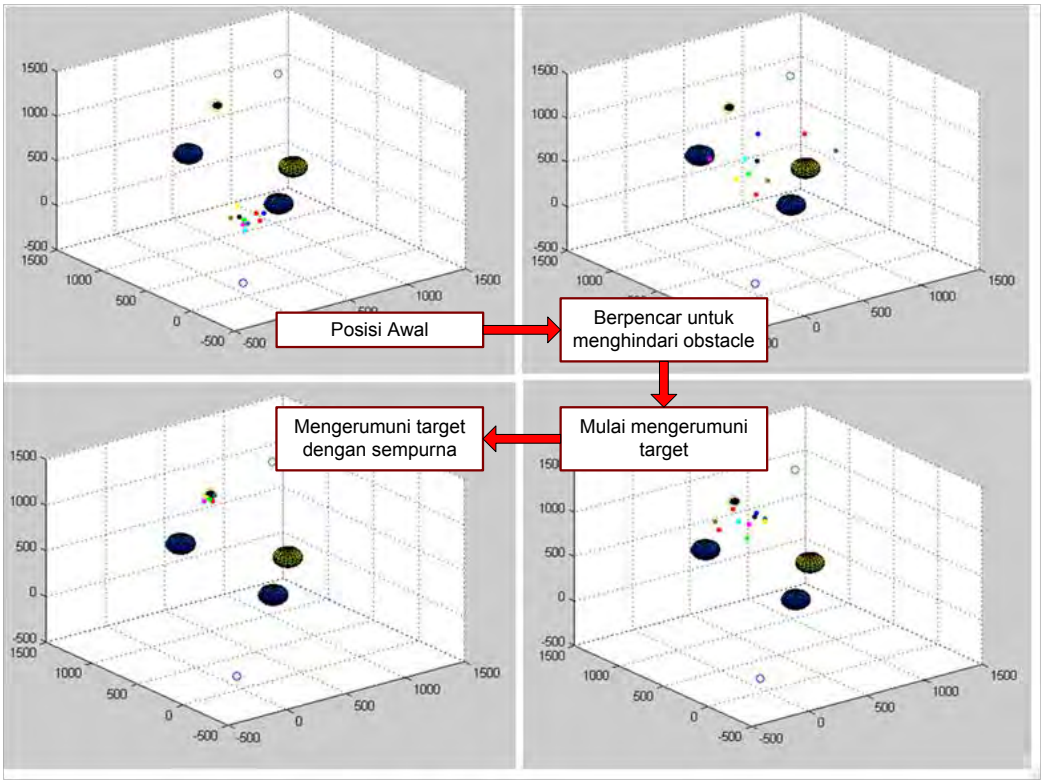
Agen	posX	Posy	posZ
1	493.0756	969.8478	963.2912
2	495.711	971.0686	990.1508
3	492.6065	989.9645	987.3191
4	496.5983	992.3401	991.652
5	494.2534	997.6287	992.8336
6	493.0139	970.0223	962.9138
7	497.4913	997.8565	990.4536
8	493.0934	970.0438	963.2907
9	506.0693	998.9765	993.4273
10	500.8859	998.3937	989.6621

Untuk mengetahui posisi agen terjauh dan terdekat dari target di setiap iterasinya, dapat diketahui dari Gambar 4.14. Gambar 4.13 adalah nilai maksimum, nilai minimum dan nilai rata-rata *fitness function* agen.



Gambar 4.14 *Fitness* Maksimum, Minimum dan Rata-rata dengan 3 *Obstacle*

Pada Gambar 4.15 searah jarum jam adalah ilustrasi pergerakan lebah dari mulai berangkat, mendatangi target dengan menghindari *obstacle*, sehingga berkerumun di sekitar target.



Gambar 4.15 Ilustrasi Pergerakan Agen Menghindari 3 *Obstacle*

4.5. Penambahan 5 obstacle statis dan penghitungan waktu

Percobaan pada skenario kelima ini ditujukan untuk mengetahui berapa waktu yang dibutuhkan agen untuk konvergen pada posisi target. Selain itu juga akan dihitung kompleksitas program percobaan ini dengan kata lain akan dicari rumus Big O dari program untuk penelitian ini. Sebuah algoritma perlu dihitung tingkat kompleksitasnya. Karena kompleksitas penerapan algoritma juga mempengaruhi pemakaian CPU dan *memory*. Semakin komplek suatu algoritma maka akan semakin besar kebutuhan pemakaian CPU dan *memory* serta ruang pada *disk*. Program untuk penelitian ini dijalankan pada netbook dengan CPU AMD Dual-Core C60/1.333GHz dan RAM 2 GB DDR3.

Percobaan ini dirancang untuk menggerakkan 10 agen, 20 agen, 30 agen, 40 agen, 50 agen, 60 agen, 70 agen, 80 agen, 90 agen dan 100 agen. Iterasi untuk percobaan skenario kelima adalah 50 dan 100. Sama seperti percobaan pada skenario sebelumnya, input percobaan skenario ke-5 ini adalah posisi random dari setiap agen. Pada percobaan ini *script* untuk menganimasikan pergerakan agen tidak akan digunakan agar diperoleh data waktu kompilasi yang akurat.

Tabel 4.13 adalah waktu yang dibutuhkan agen untuk konvergen selama 50 dan 100 iterasi mulai saat jumlah agen 10 sampai dengan ketika jumlah agen mencapai 100 agen. Data Tabel 4.13 menunjukkan bahwa setiap penambahan 10 agen, waktu yang dibutuhkan untuk menyelesaikan iterasi tidak meningkat sebanyak 10 kali, namun hanya meningkat kurang lebih 1,5 kali lebih banyak.

Tabel 4.13 Waktu eksekusi 50 dan 100 iterasi

No.	Jumlah Agen	Waktu (detik) 50 Iterasi	Waktu (detik) 100 Iterasi
1	10	0.0090407	0.016346
2	20	0.025381	0.042318
3	30	0.039063	0.080433
4	40	0.062814	0.13026
5	50	0.089649	0.19697
6	60	0.13627	0.26587
7	70	0.16638	0.3387
8	80	0.20176	0.43798
9	90	0.2504	0.53644
10	100	0.30282	0.74475

Untuk mengetahui kompleksitas dari program penelitian ini perlu ditentukan persamaan Big O program ini. Gambar 4.16 adalah gambar *nested loop* dari progam penelitian. Dimana N adalah iterasi yang dilakukan sebanyak 50 atau 100 kali, M adalah jumlah agen dan P adalah jumlah dimensi ruang 3 dimensi tempat agen bergerak.

```
for i=1:N
      %statemen
      for j=1:M
            %statemen
            for k=1:M
                  %statemen
                  for l=1:P
                        %statemen
                  end
             end
      end
end
```

Gambar 4.16 *Nested Loop* Program Penelitian

Gambar 4.16 menunjukkan terdapat 4 level loop dalam program. Berdasarkan teori Big O untuk menghitung kompleksitas diketahui bahwa kompleksitas dari program ini adalah $O(N*M^2*P)$. Tabel 4.14 merupakan perhitungan kompleksitas program jika

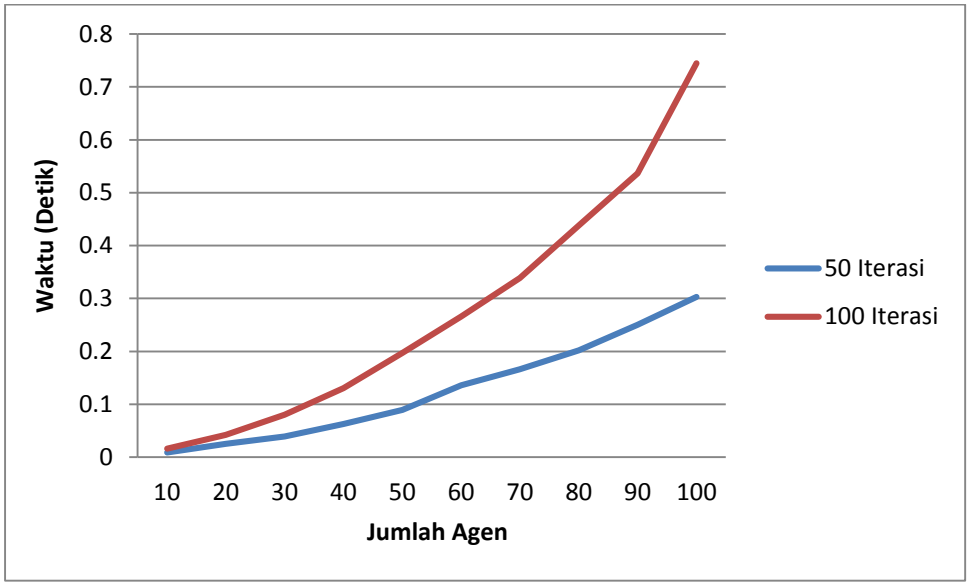
jumlah iterasi 50 dan 100, jumlah agen 10-100 dan bergerak pada ruang 3 dimensi.

Tabel 4.14 Kompleksitas program

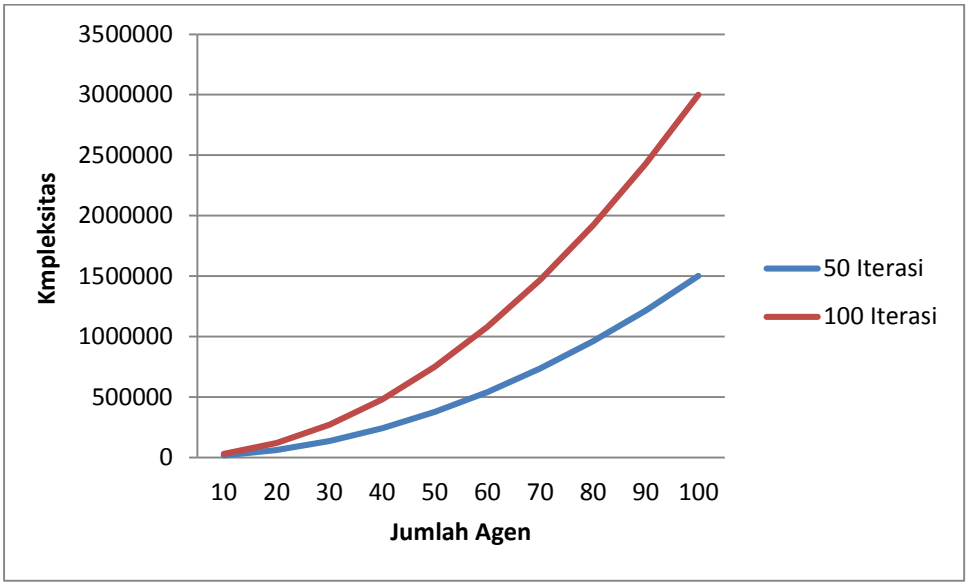
No	Juml. Agen	Kompleksitas $O(N*M^2*P)$	
		50 Iterasi	100 Iterasi
1	10	15000	30000
2	20	60000	120000
3	30	135000	270000
4	40	240000	480000
5	50	375000	750000
6	60	540000	1080000
7	70	735000	1470000
8	80	960000	1920000
9	90	1215000	2430000
10	100	1500000	3000000

Tabel 4.13 menunjukkan bahwa setiap 10 agen waktu yang dibutuhkan untuk menyelesaikan 50 atau 100 iterasi adalah 1,5 kali lebih banyak. Sedangkan pada tabel 4.14 setiap penambahan 10 agen akan menambah nilai kompleksitas program sebanyak 1,8 kali.

Gambar 4.17 adalah grafik perbandingan perubahan waktu yang dibutuhkan agen selama 50 dan 100 iterasi. Sedangkan gambar 4.18 adalah grafik perubahan nilai kompleksitas program saat program dijalankan sebanyak 50 dan 100 iterasi.



Gambar 4.17 Grafik Waktu Iterasi



Gambar 4.18 Grafik Kompleksitas Program

Gambar 4.17 dan Gambar 1.8 menunjukkan kesamaan pola. Gambar 4.17 terlihat tidak selandai Gambar 4.18 karena pada program terdapat penentuan skenario percobaan penelitian dan random pencarian posisi baru di semua fase. Sehingga ketika mencari skenario yang tepat untuk meneliti perilaku agen akan berpengaruh juga pada waktu yang dibutuhkan agen untuk menyelesaikan iterasi yang ditentukan.

BAB 5
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Percobaan yang dilakukan telah berhasil membuktikan bahwa algoritma ABC mampu membuat simulasi pergerakan sekelompok agen otonom yang dapat mengetahui posisi lawan kemudian bergerak mendatangi lawan dengan rute atau formasi acak dengan tujuan konvergen atau berkerumun di sekitar posisi target/lawan. Posisi agen selalu berubah di setiap iterasi dan agen hanya menempati posisi baru yang membuatnya semakin dekat dengan targetnya. Hal ini terjadi karena agen mengikuti prinsip pencarian posisi baru secara random pada fase Empoyed, fase Onlooker dan fase Scout.

Penambahan *constrain* untuk menghindari benturan dengan agen lain atau *obstacle* statis meningkatkan jumlah iterasi yang diperlukan agen untuk konvergen di posisi target. Sementara, jumlah agen yang terlibat dalam percobaan tidak terlalu berpengaruh pada jumlah iterasi yang diperlukan. Agen cenderung telah konvergen di sekitar iterasi ke-30. Peningkatan jumlah agen pada percobaan hanya berpengaruh pada waktu komputasi. Semakin banyak jumlah agen, semakin banyak waktu yang dibutuhkan untuk konvergen.

5.2. Saran

Kompleksitas pada penelitian ini dapat ditingkatkan dengan memberikan target yang bergerak dan obstacle yang bergerak pula. Karena pada umumnya, lawan pada game adalah target yang bergerak dan ketika bergerak menuju posisi lawan pun kadang kala agen harus bertemu dengan karakter-karakter lain yang harus dihindari. Misalnya, ada sekelompok agen lebah yang terbang mengejar manusia yang mengganggu sarangnya. Selama pengejaran tersebut koloni agen lebah harus terbang menghindari burung atau kupu-kupu yang juga terbang didepannya.

DAFTAR PUSTAKA

[1] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, Nurhan Karaboga, "A Comprehensive Survey : Artificial Bee Colony (ABC) Algotirhm and Applications", *Springer Science + Business Media B. V*, 11 March 2012

[2] Ryan E. Leigh, Tony Morelli, Sushil J. Louis, Monica Nicolescu, Chris Miles, "Finding Attack Strategis for Predator Swarm Using Genetic Algorithms", *Evolutionary Computation The 2005 IEEE Congres on Vol 3*, 2005, p. 2422-2428

[3] Ruby L V Moritz, Martin Middendorf, "Self-Organized Cooperation Between Agents that have to Solve Resource Collection Tasks", *2013 IEEE Symposium on Swarm Intelligence (SIS)*, 2013, p. 206-212

[4] Jiann-Hong Lin, Li-Ren Huang, "Chaotic Bee Swarm Optimization Algorithm for Path Planning of Mobile Robots", *Proceedings of The 10th WSEAS International Conference on Evolutionary Computing*, 2009, p. 84-89

[5] Widi Sarinastiti, "Animasi Perilaku Pasukan pada Game RTS Mengunakan Flocking Behaviour", *Teknik Elektro - Insitut Teknologi Sepuluh Nopember Surabaya*, 2014

[6] Alun Sujada, "Formasi Perang Menggunakan Algoritma Boid", *Teknik Elektro – Institut Teknologi Sepuluh Nopember Surabaya*, 2011

[7] Hazem Ahmed, Janice Glasgow, "Swarm Intelligence : Concepts, Models and Applications", *School of Computing Queen's University Kingston, Ontario, Canada K7L3N6*, 2012-585

[8] Preetha Bhattacharje, Pratyusha Pakshit, Indrani Goswami (Chakraborty), Amit Konar, Amit Konar, Atulya K. Nagar, "Multi-Robot Path Planning Using Artificial Bee Colony Optimization Algorithm", *2011 Third World Congress on Nature and Biologically Inspired Computing*, 2001, p. 219 - 224

[9] Russel Ahmed Apu, Marina L. Gavrilova, *Battle Swarm : An Evolutianary Approach to Complex Swarm Intelligence*, Department of Computer Science, University of Calgary.

[10] Stan Franklin, "Autonomous Agents as Embodied AI", *Cybernetics and System*, 28 : 6 (1997) 499-520

[11] James Ingham, "What is an Agent?", *Centre for Software Maintenance University of Durham*, 1997

[12] Darren Doherty, Colm O'Riordan, "The Design and Implementation of AI in Modern Computer Games", Department of Information Technology National University of Ireland Galway

[13] http://clashofclans.wikia.com/wiki