Cooperative Problem Solving With a Distributed Agent System - Swarm Intelligence

Diplomarbeit / Thesis

Fachbereich Angewandte Informatik
Fachhochschule Ravensburg-Weingarten
Doggenriedstraße, 88250 Weingarten

und / and

Center for Self-Organizing and Intelligent Systems (CSOIS)
Dept. of Electrical and Computer Engineering
Utah State University, Logan, Utah 84322-4160, USA

vorgelegt von / presented by: Fernando Buck
Zweierweg 5
88250 Weingarten

Matrikel-Nr / matriculation number: 12817

Erstbetreuer / first supervisor: Prof. Dr. rer.nat. Wolfgang Ertel
Zweitbetreuer / second supervisor: YangQuan Chen, PhD

Abgabetermin / closing date: 30.11.2005 / 11-30-2005

Abstract

In this thesis a swarm intelligent approach for controlling and coordinating a multi-agent system is studied. Considering an example where bodyguard agents have to protect one or more VIP agents, it is shown that the swarm intelligent approach can be a flexible and robust way to control a multi-agent system.

Contents

List of Figures

List of Code Examples

1 Introduction

This thesis discusses the topic of cooperative problem solving with a distributed agent system by using a rather new approach called swarm intelligence. Swarm intelligence is an interdisciplinary research topic that is highly inspired by observations of the behavior of social insect colonies and by other observations from biology. It is a bottom-up-approach: simple actions, which are carried out by a typically huge amount of agents, emerge into a complex group behavior. A general introduction is given in 2, starting with observations from nature. By observing social insect colonies (2.1), several questions arise: What is what governs there, is there anybody in charge? How does cooperation on the group level arise from interactions of the individuals? We will see that self-organization and stigmergy are the basic ingredients that let individual behaviors emerge into a collective group behavior (2.2, 2.3). It is shown that by analyzing and modeling the behavior of social insect colonies, knowledge can be extracted that can be used to solve distributed problems. As an example the foraging behavior of ants is studied (2.4.1), and with the principles that can be learned from this example the Traveling Salesman Problem is solved (2.4.2). It is only naturally that an efficient way to solve the Traveling Salesman Problem could also be a suitable approach to solve telecommunication application problems (2.5). In 2.6 a short summary of current and future swarm-based applications is given. In 2.7 the swarm intelligent approach is summarized by pointing out the pros and cons of this approach.

The swarm intelligent approach is then applied to a scenario where bodyguard agents protect VIP agents. It is shown that the bodyguard agents can achieve that by using only local information and without communicating with each other. To verify the proposed behavior rules, NetLogo simulations are done and the results of these simulations are presented. In 3.1 the details of the envisaged scenario are given. In 3.2 a simulation environment called NetLogo is presented, in which all the simulations were done.

In 3.3 the agent behavior rules that lead to the desired group behavior are presented and verified in simulations.

Finally, in 4, the advantages and disadvantages of the proposed approach are investigated. It is shown that the swarm intelligent approach can deliver a robust and flexible multiple agent system, but that it is very hard to “program” and to design.

2 Swarm Intelligence

Many distributed problems have been solved by self-organizing autonomous agent systems. In such systems agents with a local view of their environment take actions that advance global system objectives. This approach is called swarm intelligence. Swarm intelligence is highly inspired by observations from nature, especially from observations of social insects, like e.g. ants. The number of successful Swarm intelligence applications is growing fast, especially in combinatorial optimization, communicational networks, and robotics. A good starting point to learn about swarm intelligence is [13] (especially the first chapter). The rest of this chapter is based on this work; some pages are directly copied out of it.

2.1 Social Insects

Social insects are fascinating beings. Every single insect in a social insect colony seems to have its own agenda, and yet an insect colony looks so organized. The integration of all individual behaviors does not seem to require any supervisor.

For example, Leafcutter ants cut leaves from plants and trees to grow fungi (Fig. 2-1). Workers forage for leaves hundreds of meters away from the nest, literally organizing highways to and from their foraging sites [14].

Fig. 2-1 Leafcutter Ants Bringing Back Cut Leaves to the Nest

Weaver ant workers form chains of their own bodies, allowing them to cross wide gaps and pull stiff leaf edges together to form a nest. Several chains can join to form a bigger one over which workers run back and forth. Such chains create enough force to pull leaf edges together. When the leaves are in place, the ants connect both edges with a continuous thread of silk emitted by a mature larva held by a worker [14].

Army ants organize impressive hunting raids, involving up to 200,000 workers, during which they collect thousands of prey.

Honey bees build series of parallel combs by forming chains that induce a local increase in temperature. The wax combs can be more easily shaped thanks to this temperature increase. With the combined forces of individuals in the chains, wax combs can be untwisted and be made parallel to one another. Each comb is organized in concentric rings of brood, pollen, and honey. Food sources are exploited according to their quality and distance from the hive. At certain times, a honey bee colony divides: the queen and approximately half of the workers leave the hive in a swarm and first form a cluster on the branch of a nearby tree. Potential nesting sites are carefully explored by scouts. The selection of the nesting site can take up to several days, during which the swarm precisely regulates its temperature [15].

Nest construction in the wasp Polybia occidentalis involves three groups of workers, pulp foragers, water foragers, and builders. The size of each group is regulated according to colony needs through some flow of information among them (Fig. 2-2).

Fig. 2-2 Nest Building in the Neotropical Wasp Polybia Occidentalis

Tropical wasps build complex nests, comprised of a series of horizontal combs, protected by an external envelope and connected to each other by a peripheral or central entrance hole (Fig. 2-3).

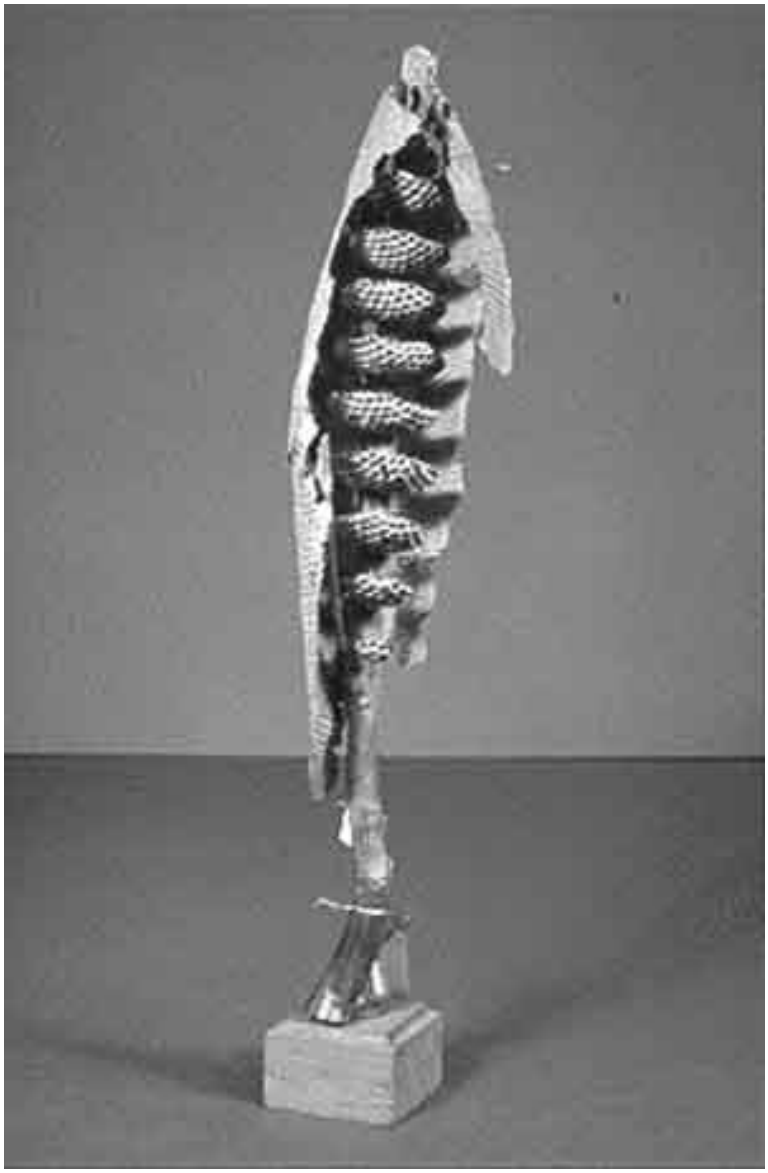

Fig. 2-3 Nest of a Parachartergus Wasp Species

Some species of termites build even more complex nests (Fig. 2-4), comprised of roughly cone-shaped outer walls that often have conspicuous ribs containing ventilation ducts which run from the base of the mound toward its summit, brood chambers within the central “hive” area, which consists of thin horizontal lamellae supported by pillars, a base plate with spiral cooling vents, a royal chamber, which is a thick-walled protective bunker with a few minute holes in its walls through which workers can pass, fungus gardens, draped around the hive and consisting of special galleries of combs that lie between the inner hive and the outer walls, and, finally, peripheral galleries constructed both above and below ground, which connect the mound to its foraging sites.

Fig. 2-4 Macrotermes Mound

And there are many more examples of the impressive capabilities of social insects. If no one is in charge, how can one explain the complexity and sophistication of their collective productions? An insect is not a simple creature, but the complexity of one single insect is not sufficient to explain the complexity of the behavior of a social insect colony. Perhaps the most difficult question is how to connect individual behavior with collective performance or in other words, how does cooperation arise?

2.2 Self-Organization

The key point to this question is self-organization (SO): The complex collective behavior may emerge from actions among individuals that act rather simple. The discovery of SO provides us with a powerful tool to transfer knowledge about social insects to the field of intelligent system design. One of the most important features of social insects is that they can solve distributed problems in a very flexible and robust way, and because social insects act rather simple, it allows us to design agents that mimic their behavior at some level of description.

One problem is that swarm intelligent systems are hard to design, because the paths to problem solving are not predefined but emergent in these systems. One possible solution for that is to study how social insects collectively perform some specific task, model their behavior, and use the model as a basis upon which artificial variations can be developed, either by tuning the model parameters beyond the biological relevant range or by adding non-biological features to the model. Of course an engineer is not constrained by what is observable in nature. An engineer can resort to whatever available techniques that are appropriate to solve a problem. But often natural selection may have picked those biological organizations that are most efficient, flexible and robust. Therefore looking how nature solved a problem can be a good starting point to design a swarm intelligent system.

SO is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components. The rules specifying the interactions among the system's constituent units are executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system by an external ordering influence. Self-organization relies on four basic ingredients:

1) Positive feedback consists of a set of simple behavioral rules that promote the creation of structures. An example of positive feedback is recruitment and reinforcement. For instance, recruitment to a food source is a positive feedback that relies on trail laying and trail following in some ant species, or dances in bees.

2) Negative feedback is the counterpart to positive feedback and helps to stabilize the collective pattern: it may take the form of saturation, exhaustion or competition. In the example of foraging, negative feedback stems from the limited number of available foragers, satiation, food source exhaustion, crowding at the food source, or competition between food sources.

3) SO relies on the amplification of fluctuations (random walks, errors, random task-switching, and so on). Not only does collaborative behavior emerge despite randomness, but randomness is often crucial, since it enables the discovery of new solutions, and fluctuations can act as seeds from which structures nucleate and grow. For example foragers in ant colonies might get lost; even though it may seem inefficient, lost foragers can find new, unexploited food sources, and recruit nest mates to these food sources.

4) All cases of SO rely on multiple interactions. A single individual can generate a self-organized structure such as a stable trail provided pheromonal lifetime is sufficient, because trail-following events can then interact with trail-laying actions. However, SO generally requires a minimal density of mutually tolerant individuals. Moreover, individuals should be able to make use of the results of their own activities as well as of others' activities (although they may perceive the difference): for instance, trail networks can self-organize and be used collectively if individuals use others' pheromone. This does not exclude the existence of individual chemical signatures or individual memory which can efficiently complement or sometimes replace response to collective marks.

A self-organized phenomenon can usually be characterized by a few key properties:

1) The creation of structures in an initially homogeneous medium. Such structures include nest architectures, foraging trails or social organization.

2) The possible coexistence of several stable states (multi-stability). Because structures emerge by amplification of random deviations, any such deviation can be amplified, and the system converges to one among several possible stable states, depending on initial conditions.

3) The existence of bifurcations when some parameters are varied. The behavior of a self-organized system can change dramatically as a result of small changes in the parameters.

2.3 Stigmergy

In combination with self-organization, stigmergy is the other most important concept of swarm intelligence. The term was introduced by French biologist Pierre-Paul Grassé in 1995 to refer to termite behavior. He defined it as: “Stimulation of workers by the performance they have achieved.” In a stigmergic system communication doesn’t take place in a direct but in an indirect way. Individual parts of the system modify their local environment, and other parts, or even the same individuals, react to these modifications of the environment. For example task coordination and regulation in the context of nest reconstruction in some termites species does not depend on the workers themselves but are mainly achieved by the nest structure: a stimulating configuration triggers the response of the termite worker, transforming the configuration into another configuration that may trigger in turn another action performed by the same termite or any other worker in the colony. Another example of how stigmergy and self-organization can be combined is recruitment in ants: self-organized trail laying by individual ants is a way of modifying the environment to communicate with nest mates that follow such trails.

Replacing direct communication by indirect communication not only allows agents to be simple, but is also often associated with flexibility: If the environment changes because of external perturbations, the agents respond appropriately to that perturbation, as if it were a modification of the environment caused by the colony’s activities.

2.4 An Example from Nature and its Application

2.4.1 Ant Foraging

A very interesting study of swarm intelligence investigated the foraging behavior of ants. Everybody has seen the so called “ant highways”, connecting ant nests to food sources, often in the seemingly shortest way. How do these rather simple creatures solve that problem? In experiments with the Argentine ant Linepithema humile, Deneubourg et al showed that path selection to a food source is based on self-organization. In one experiment Deneubourg presented a food source to an ant colony, separated by a bridge with two equally long branches (Fig. 2-5). The observation was the following: The first few minutes oscillations appeared, which means that the one or the other branch was used more often, caused by random fluctuations. But after a while the ants tended to use all the same branch.

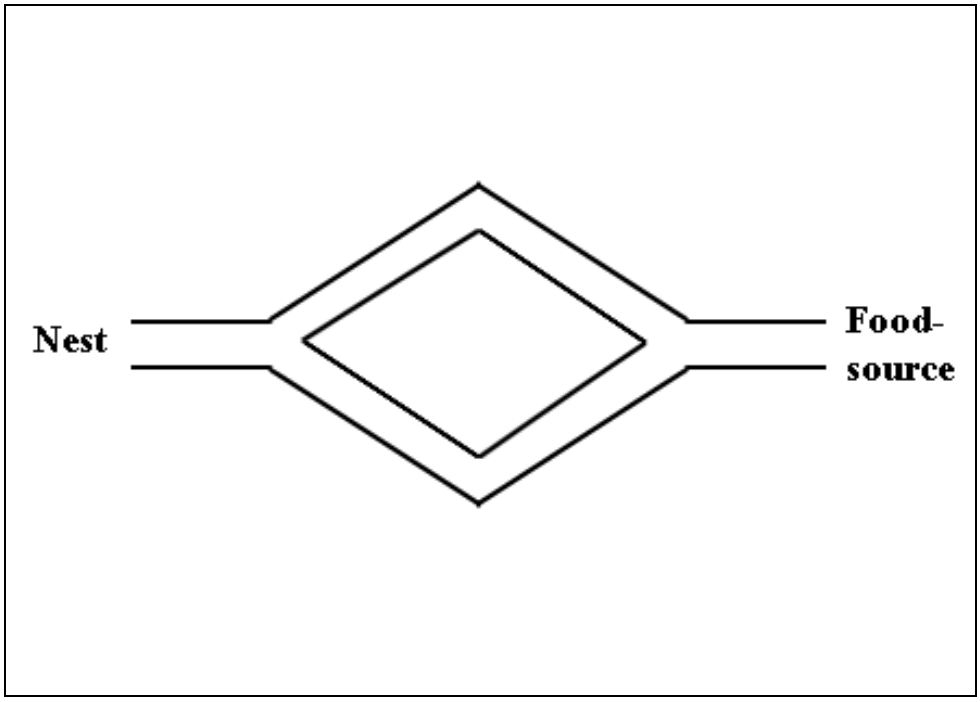

Fig. 2-5 Binary Bridge Experiment

In Fig. 2-6 the usage of the two branches in percent is plotted against the time. Obviously the initial fluctuation in this experiment was not strong enough to prevent exploitation of the other branch.

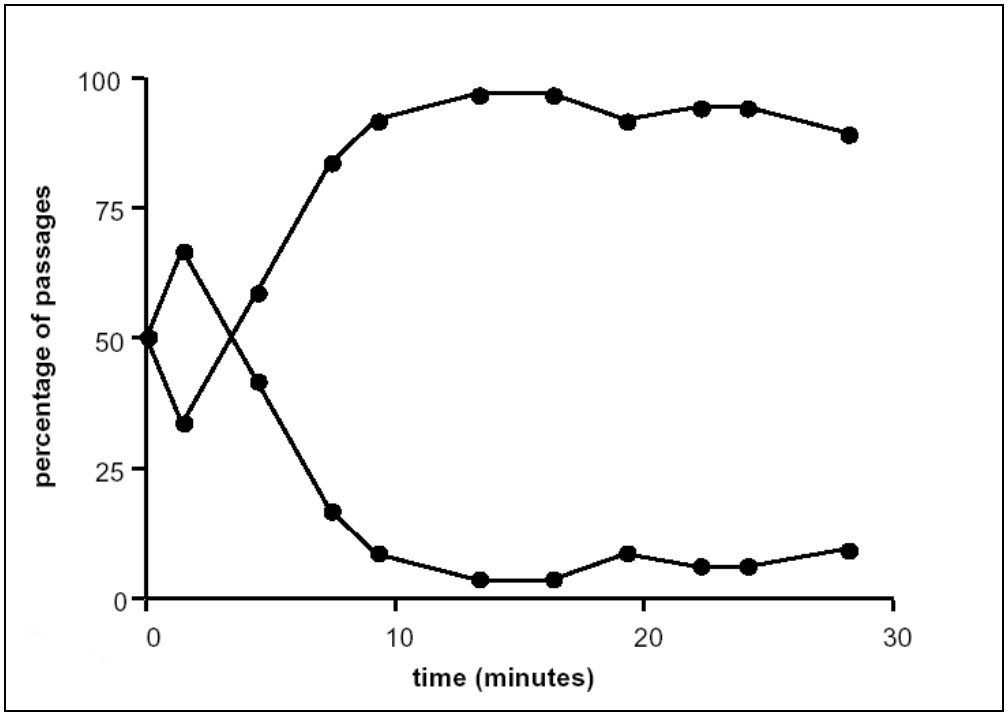

Fig. 2-6 Usage of the Two Branches in Percent

This behavior of the ant colony can be explained quite simple. What happens is that each ant lays a pheromone trail on the way to the food source as well as on the way back to the nest. The amount of pheromone on one branch is proportional to the number of ants which have been using the branch in the past. The probability of choosing a branch at a certain time depends on the amount of pheromone on the branch. So after random fluctuations the system eventually goes into a steady state, because one branch gets chosen more often by chance, which leads to a higher pheromone concentration on that branch. The higher pheromone concentration on one of the branches leads to a higher usage of that branch by the ants. This in turn increases the concentration of pheromone and so on. This indirect communication between the ants is an ideal example for stigmergy. One ant changes the environment by laying a pheromone trail and the other ants react on that changed environment by following that trail.

In a second experiment Deneubourg constructed a bridge with two branches, one twice as long as the other, that separated a nest from a food source. Within just a few minutes the colony usually selected the shorter branch. Again this behavior can be explained by laying and following pheromone trails. The first ants returning to the nest from the food source are those that have taken the shorter path in both directions. Because this route is the first to be marked with pheromone, nest mates are attracted to it. This is shown schematically in Fig. 2-7.

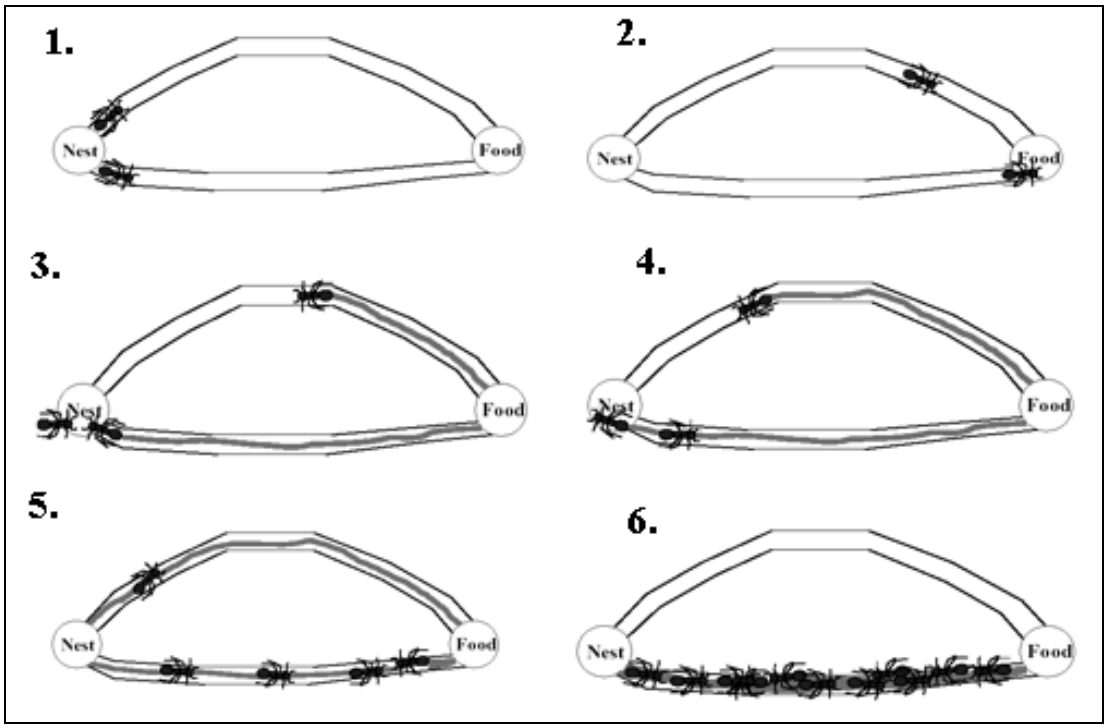

Fig. 2-7 Bridge Experiment with Unequally Large Branches

Simulations of this behavior were successfully done. In Fig. 2-8 a simulation is shown where 3 food sources of identical quality are presented to a colony of artificial ants (a). Pheromone decay is assumed to occur over short time scales. First the ants explore their environment randomly (b). After a while the closest food sources are connected to the nest by trails (c). After the closest food sources are exploited the exploitation of farther food sources begins (d).

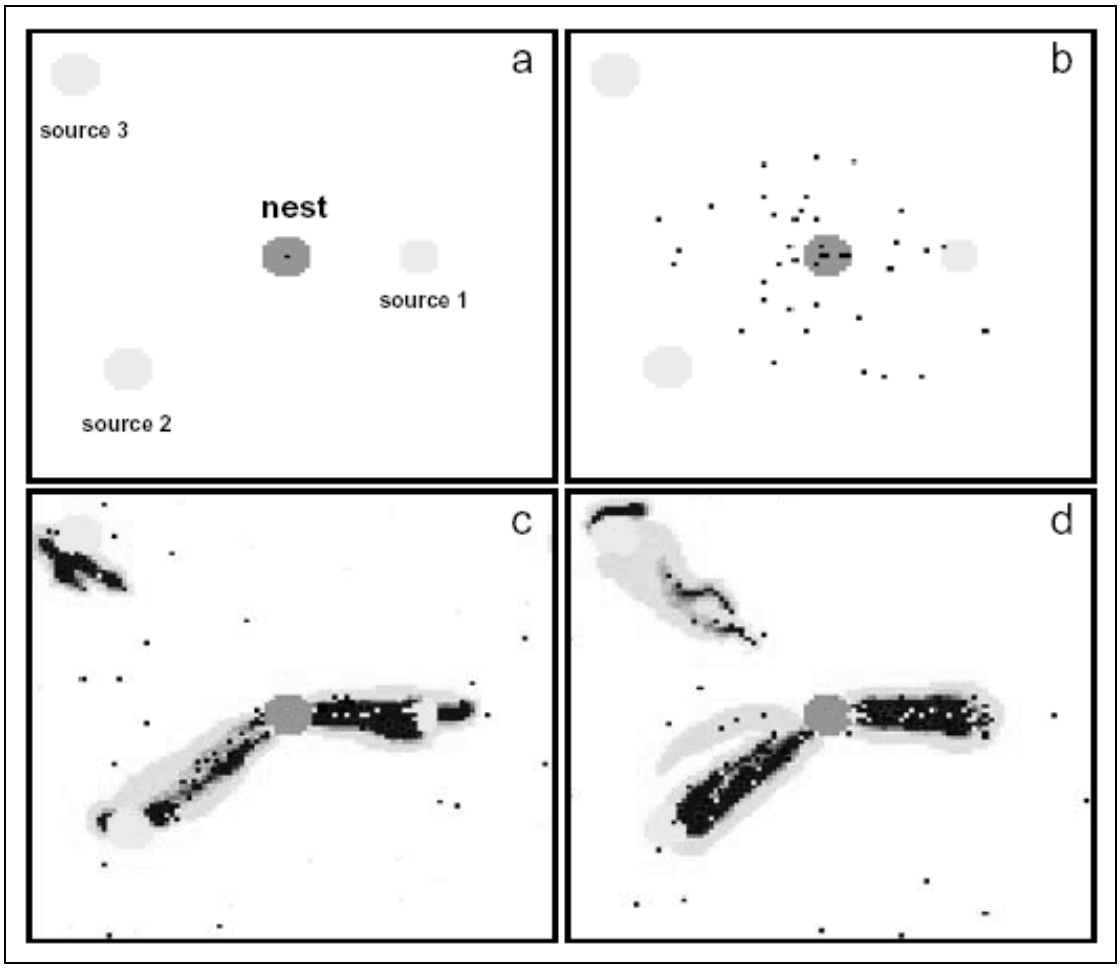

Fig. 2-8 Simulation of Ant Foraging

2.4.2 Solving the Traveling Salesman Problem with Artificial Ants

If ants find the shortest path between two or more locations, it is only natural that one of the first applications of swarm intelligence was a path optimization problem: the traveling salesman problem (TSP).

The basic idea behind solving TSP with ant-based algorithms is using a positive feedback algorithm, based on an analogy with the trail-laying and trail-following behavior of some ants, to reinforce those portions of good solutions that contribute to the quality of these solutions, or to directly reinforce good solutions. To avoid that good - but not very good - solutions become reinforced to an extent where they constrain the search of new solutions too much, a negative feedback is introduced. This negative feedback consists of pheromone evaporation. One first approach to solve the TSP with an ant-based algorithm was called Ant System (AS).

In AS artificial ants are deployed randomly on the nodes of a TSP graph. Then they move around from one node to another until they complete a tour. A tour is completed if every node is visited once. After each completed tour each ant deposits a certain amount of virtual pheromone on each edge of the problem graph, depending on how far the ant traveled during its tour- shorter tours are rewarded with more pheromone. A certain amount of virtual pheromone will also decay, causing older solutions to fade away.

The decision of each ant in each time step to which node to go next is done with the following rules:

- An ant avoids visiting a node that has been already visited by maintaining a tabu list.
- The distance to the next nodes. This information is based on strictly local information and represents the heuristic desirability of choosing nodes j when in city i. Nodes that are closer are selected with higher probability.
- The amount of virtual pheromone on the edges that connect node j with node i.

Fig. 2-9 shows a possible solution of a given TSP problem, found by artificial ants. The remarkable thing to notice is that the algorithm finds multiple solutions, which makes this approach highly flexible.

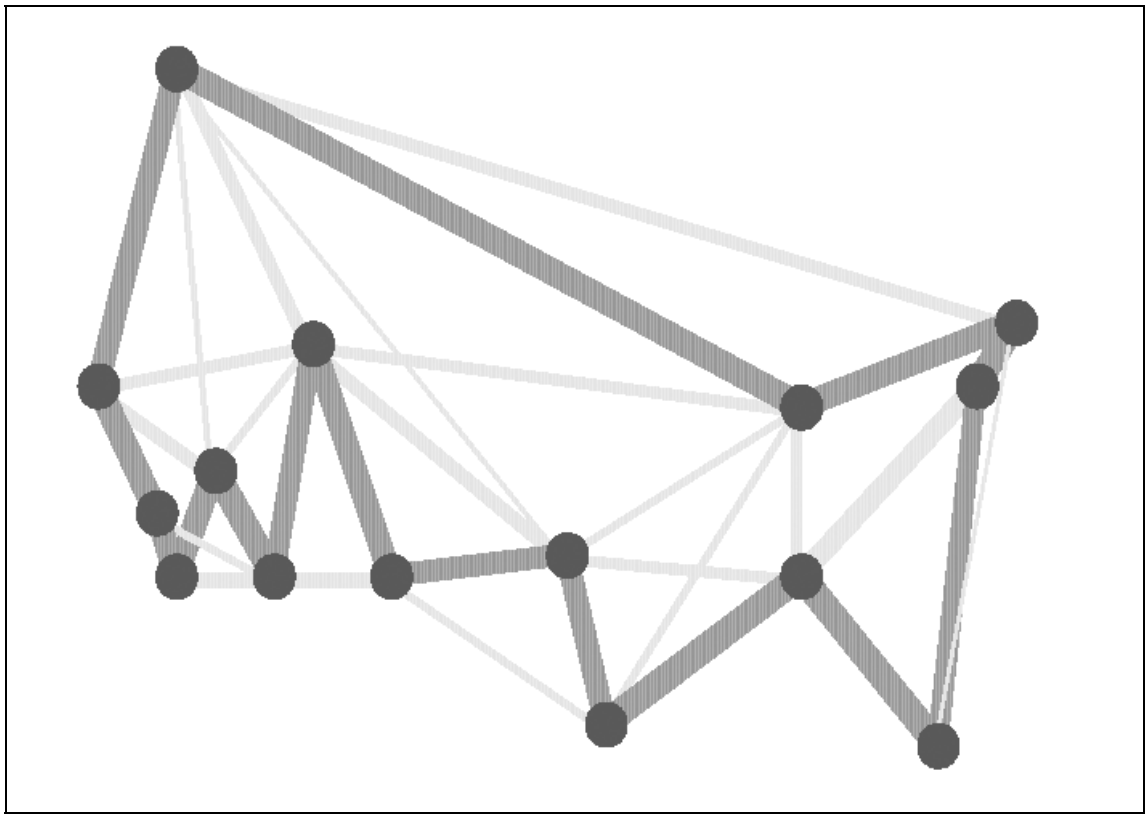

Fig. 2-9 Solution of a TSP Problem Found by AS

Ant system has been compared to other general purpose heuristics on relatively small TSP (e.g. 30 cities) and performed pretty well. Unfortunately, for problems of growing dimensions (e.g. 75 cities), AS never reached the best known solutions after 3000 iterations. But an improved version of AS equipped with a simple local search reaches outstanding performance. That algorithm (Ant Colony System) will not be discussed in detail here, for details see [16][17]. The Ant Colony System algorithm belongs to the best TSP solving algorithms.

2.5 Telecommunication Applications

If swarm intelligent approaches deliver good algorithms for solving TSP, it is only natural to consider these approaches for telecommunication applications. Similar to TSP, routing is a process where information is transmitted from a source to a destination through a sequence on switching/buffering stations or nodes. The objective of any routing algorithm is to direct traffic from sources to destinations maximizing network performance while minimizing costs (like rate of call rejection, throughput, packet delay...). What makes this problem especially interesting is that network conditions constantly change. Routers may fail, new routers may be added, local congestions may arise and so on. Very interesting and successful swarm-based telecommunication applications have been developed. One is called ant-based control (ABC), introduced by Schoonderwoerd et al [18]. This algorithm was designed with a telephone network application in mind. Another algorithm, called AntNet was proposed by Di Caro and Dorigo [19]. AntNet was primarily designed for packet-switching and connectionless networks like the Internet. Several companies are exploring these approaches for handling their traffic on networks, e.g. the France Telecom, British Telecommunications, and the American MCI. These algorithms will not be presented here.

2.6 Other Swarm-Based Applications

Probably the most successful swarm-based applications have been developed in communication and optimization problems. But also the controlling of a group of robots by adopting the behavior of insect swarms is increasingly being investigated. In another project, a model that was initially introduced to explain how ants cluster their dead and sort their larvae has become the basis of a new approach for analyzing financial data. This approach is a new and promising method to solve sorting problems. The scheduling of jobs, e.g. in a factory, is another promising topic that is under investigation, inspired by the way honeybees perform task allocation. There are numerous other examples: investigation in controlling unmanned vehicles is done, the NASA is investigating the use of swarm technology for planetary mapping. Even the usage of "nanobots" within the human body for the purpose of killing cancer tumors is discussed as a future application.

2.7 Summary

Swarm intelligence is a growing research field for solving distributed problems. Observations from nature showed that problem solving by social insects is done in a flexible, robust, decentralized and self-organized way. Swarm intelligent systems, if well designed, can deliver similar results. Furthermore some tasks may be too complex for a single agent to perform. Increased speed can result from using several agents. Designing and building several simple agents can be cheaper than designing a few complex agents. They can be more flexible, reliable and fault-tolerant, because one or several agents may fail without affecting task completion. Randomness or fluctuations may enhance the system's ability to explore new behaviors and find new solutions. The use of stigmergy can greatly reduce the need of communication between agents. Furthermore the fact that no central controlling unit is in charge can be a big advantage: In a centralized system the failure of the central control normally means the failure of the whole system; while the failure of an individual in a swarm intelligent systems normally has only little impact on the performance of the whole system.

Indeed, the potential of swarm intelligence is enormous. It offers an alternative way of designing systems that have traditionally required centralized control and extensive preprogramming. It instead boasts autonomy and self-sufficiency, relying on direct or indirect interactions among simple individual agents. As mentioned, one big advantage is that this can lead to systems that can adapt quickly to rapidly fluctuating conditions.

Of course swarm intelligence has not only advantages. For example stagnation can occur: Because of the lack of global knowledge, a group of agents may find itself in a deadlock, where it cannot make any progress. But the biggest problem is to determine how the agents of a swarm-based solution should be programmed. Because the pathways to solutions are not predefined but emergent, there is no general guideline how to design a swarm intelligent system. A good example where both of the mentioned problems appear is AS (see 2.4.2): As mentioned one parameter that needs to be tuned is the evaporation rate of the deposited pheromone. This parameter has no biological relevance, because in reality the pheromone evaporates very slowly. With such an evaporation rate, the systems would stagnate very fast, because initial random fluctuations would be amplified. In other words all the ants would end up doing the same tour. The solutions found by such a system would very likely not be optimal. On the other hand, if the pheromone decay is tuned too fast, the routes are not stabilized fast enough, so that again no optimal solutions are found. Another parameter that has to be tuned is the total number of ants: Too many ants would quickly reinforce suboptimal trails and lead to early convergence to bad solutions, whereas too few ants would not produce the expected synergistic effects of cooperation, because of the process of pheromone decay. So here we have an example of two parameters that need to be tuned to have a good working system. But how to tune these parameters? This is exactly one of the big questions, and probably the biggest drawback of swarm intelligence. Nobody knows the answer to that question; it is a process of trial and error. There is a similarity here with the problem of designing neuronal networks: How can one tune the weights of the connections so that the network performs a given task?

One approach to overcome this problem is to tune the parameters according to the values that can be observed in nature. But often the processes found in nature are not well understood, and even if such parameter can be measured in nature, it is not guaranteed that these values are optimal. And in most cases we can't even find a direct analogy in nature for a given distributed problem.

To tune a system with two or three parameters can be a tiring job. But what are two parameters? In a real world application we typically speak of more than 10 or maybe even a hundreds of parameters. How to tune efficiently the parameters of such a system is a current research topic. One promising solution might be to let these parameters being tuned by genetic algorithms.

But not only "programming" a swarm intelligent application is a problem. The bigger problem might be to define it. How complex should individual agents be? Should they be identical? Should they have the ability to learn? Should they have a memory? How local should their knowledge of the world be? And so on. These questions are problem specific, and unfortunately there is no general guideline at all. One approach could be to start with the simplest possible agents and then progressively to increase the complexity until the given problem is solved satisfactorily. Genetic algorithms might solve the problem of tuning parameters, but finding the relevant agent rules and the relevant parameters that might be considered for a given problem is a very different beast. In other words, designing a swarm intelligent system seems to be impossible to be automated, at least in the near future.

Last there is another major problem with swarm intelligent applications: How do we measure the performance of such an application? Swarm intelligent approaches typically work as poor to average in static problems, but fully express their capabilities only in the context of dynamically changing problems. But designing benchmarks that take into account dynamic changes is difficult.

In view of all these issues, I think it is fair to say that swarm intelligence is on the right track, but a lot more work needs to be done.

3 Bodyguard Scenario

3.1 Scenario Details

In this thesis a bodyguard scenario is envisaged. A swarm intelligent system is developed, in which multiple autonomously acting bodyguard agents shall protect one or more VIP agents. To protect a VIP in this context means:

- to find the VIP
- to encircle the VIP
- to follow the VIP in a given distance

The VIPs are agents that move around randomly. All agents shall be distributed randomly at startup. The amount of agents shall be flexible, both the amount of VIPs and the amount of bodyguards. No agent shall use any global information; all agents shall use exclusively local information. That also includes the amount of VIPs and the amount of bodyguards that are deployed. All decisions shall be done individually by each agent, and no direct communication between the agents is desired to keep the agents as simple as possible.

The scenario takes place in a closed world. To make it as simple as possible a quadratic area with no obstacles is used. For further simplification it is assumed that the VIPs always move with a constant speed. The bodyguards can move with a non-constant speed if desired.

The main attention is turned on the behavior of the bodyguards. The VIPs are just very simple agents that move around randomly, without any intelligence at all. Furthermore the problem of avoiding collisions is not envisaged in this work.

First a solution for only one VIP is developed; later this solution will be extended to take two or even more bodyguards into account.

3.2 Simulation Environment

NetLogo [20] is used to simulate the VIP scenario. NetLogo is a programmable modeling environment to simulate complex multi-agent systems that develop over time. Instructions to hundreds or even thousands of independent agents can be given, all operating concurrently. This makes it a perfect tool to simulate swarm intelligent systems, because the connections between the micro-level behavior of individual agents and the macro-level patterns that emerge from the interactions of the individuals can be explored. It is quite simple to create new models in NetLogo, yet it is a powerful tool for studying multi agent systems. Last but not least it is free of charge. See Fig. 3-1 for a screenshot of NetLogo. The screenshot shows a run of an ant foraging model, which is already included in NetLogo.

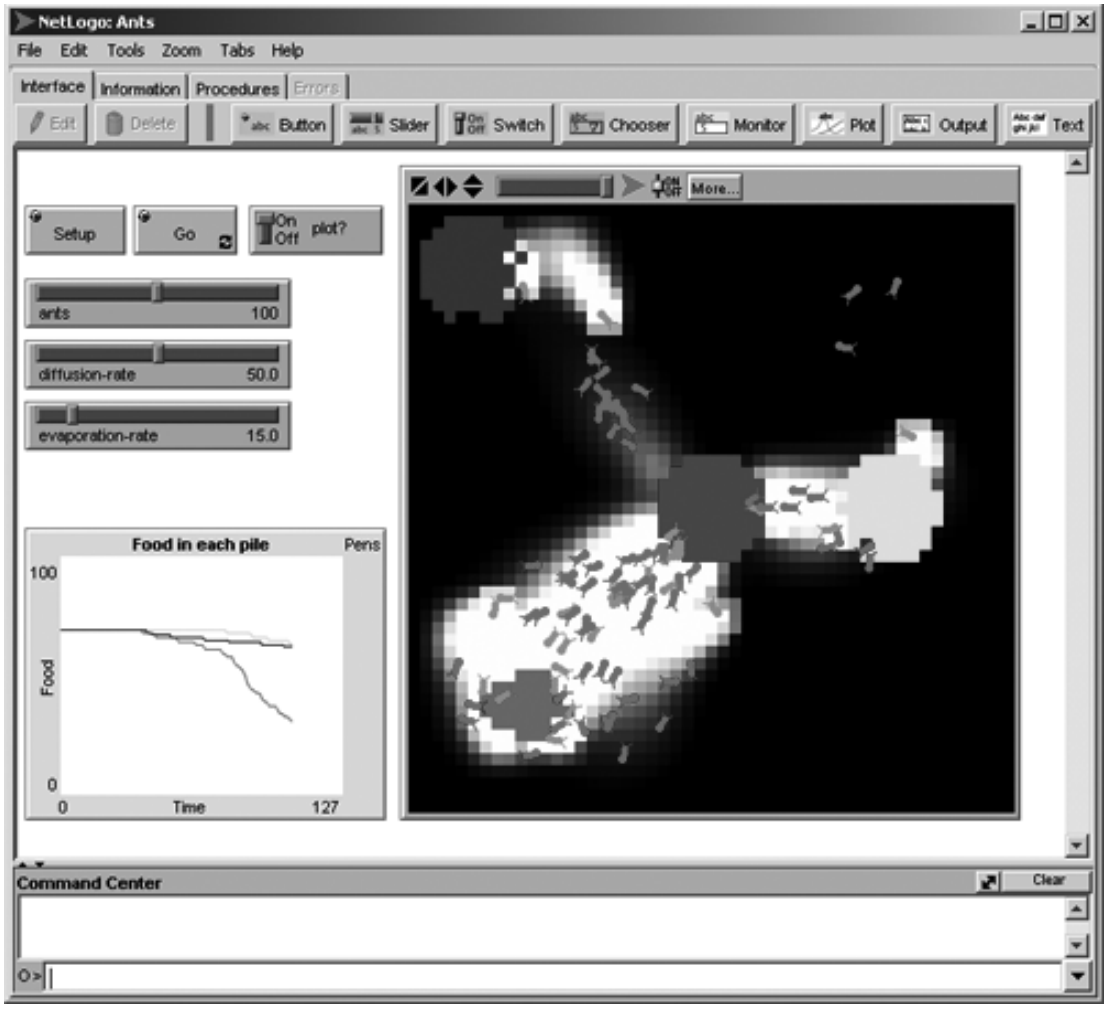

Fig. 3-1 Screenshot of a NetLogo Simulation

3.3 Agent Behavior Rules

Like mentioned in the introduction there is no general guideline for how to design the behavior rules of the individual agents of a multi agent system, so that a collective pattern arises. To learn the behavior rules of the agents by copying them from nature also failed for this application, because no suitable analogy in nature was found. Of course there are similar looking behaviors in nature, for example a pride of wolfs may follow an alpha wolf or something like that, or we might even look how human bodyguards do their job. But these analogies are not suitable. They are not suitable because in those examples we have highly complex and intelligent individuals. The intelligent group behavior is not the emergent result of simple individual behavior, but rather a result of a complex individual behavior. And in social insects there is usually no leader-follower principle, which could be compared to a VIP-bodyguard scenario.

So how can we define the agents? How to find the behavior rules that emerge in a collaborative group behavior? Like mentioned in 2.7, probably the only possibility to solve this problem is to start with a very simple agent, and progressively add more complexity. Therefore a very simple agent is developed that is able to find a VIP. Once this task is satisfactorily accomplished by this agent, further ability is added, like the ability to follow a VIP in a given distance and so forth. Step by step an agent is developed that will be (hopefully) capable of completing the whole task. After each step the agent is tested in simulations.

3.3.1 Finding the VIP

Like mentioned all agents shall be distributed randomly at startup. To find a VIP, the bodyguards must have the ability to sense a VIP somehow, provided that they are close enough. This can be achieved in several ways, e.g. through visual information. In this work it is assumed that a VIP “smells”, and that the bodyguards can climb up a smell gradient. This is done with the pheromone trail following capabilities of some ant species in mind. The smell of a VIP is assumed to diffuse evenly over the neighbor environment. It is assumed that the bodyguards only react to a smell intensity greater than a given threshold. If the smell intensity is below that threshold, the bodyguards can’t sense anything, and therefore they don’t know where a VIP could be. Hence they keep on searching. Searching means that they keep on moving randomly until they sense the smell of a VIP. If they sense the smell of a VIP, they climb up the smell gradient, which leads them directly to the VIP.

In a real robot application, this behavior could be achieved quite simple. Of course, nobody would use smell but rather visual information or maybe the strength of a radio signal. One example of a very simple real robot that would perform very similar to the above described behavior would be a Braitenberg vehicle, type2 (Fig. 3-2) [21].

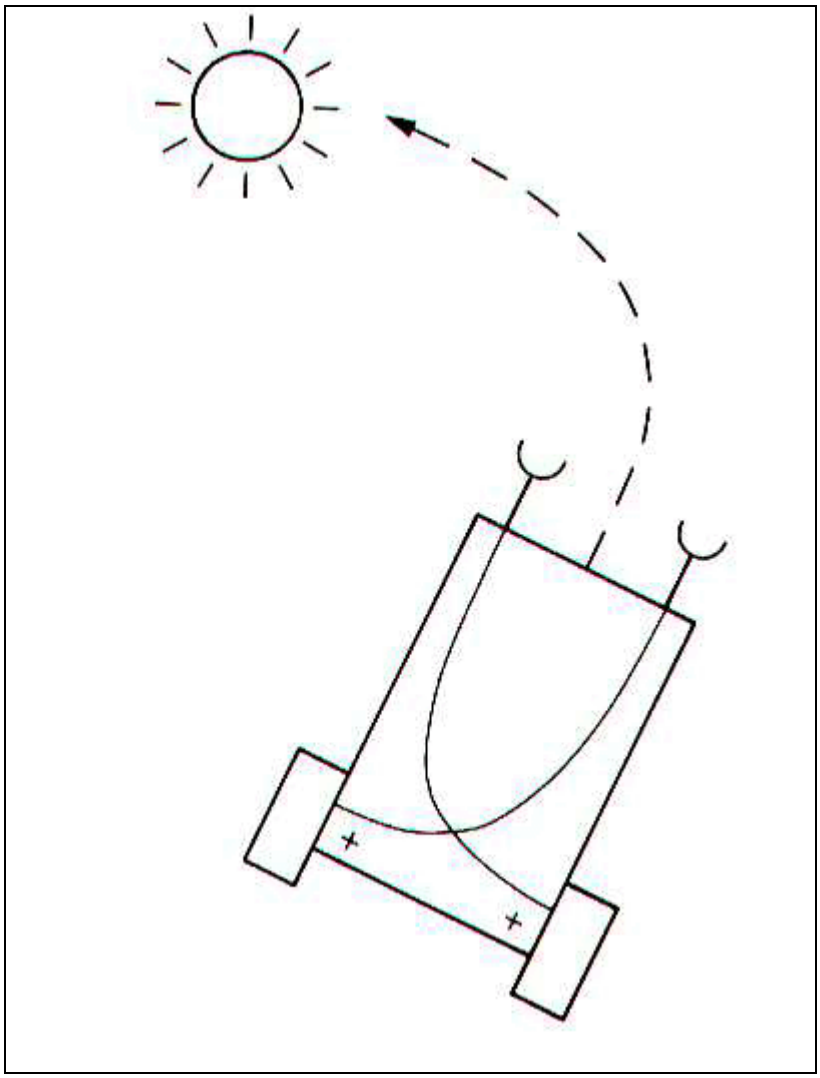

Fig. 3-2 Braitenberg Vehicle, Type 2

This type of vehicle has two sensors and two motors; the left sensor is connected to the right motor and vice versa. If the left sensor gets more input (e.g. light), the right motor turns faster, which lets the vehicle turn left. If both sensors have the same input, the vehicle will go straight. Therefore this vehicle turns towards a source and approaches the source until it hits the source. This behavior will be refined in the next section.

The pseudo code for such a bodyguard agent that approaches a smelling VIP is given in Code Example 1.

```
Function Search_VIP()
    loop do
        smell <- actual smell intensity
        direction <- direction where smell comes from
        newheading <- Setheading(smell,direction)
        set the new heading of agent to value of newheading
        move agent one step forward

Function Setheading(smell,direction) returns a new heading
    newheading <- a random value
    if smell > 5 then newheading <- direction
    return newheading
```

Code Example 1 Pseudo Code for Approaching a VIP

Note that “direction” and “newheading” are variables that represent headings in degrees. In the function Setheading() the value for the new heading is set to a random value first. If the intensity of smell is bigger than a threshold (5) the value for the new heading is overwritten with the value of the parameter direction. The lower threshold for the smell intensity is obtained experimentally and has no physical or biological relevance.

A NetLogo code, executed each time step for each bodyguard could look like this:

```
ifelse value-from (patch-here) [ smell ] >= 5 [ set heading uphill smell ]
[
    lt random 45
    rt random 45
]
fd 1
```

Code Example 2 NetLogo Code for Approaching a VIP

Note that the heading of each agent is set towards the smell source, only if the smell is stronger than 5. If the smell intensity is lower than that threshold, the heading is set randomly. Note furthermore that the code is not embedded in a loop. It is supposed that that piece of code is implemented within (or called from) a go() procedure, which is called automatically by the NetLogo system at each time step.

3.3.2 Following a VIP in a Given Distance

After being able to find a VIP the bodyguards should be able to follow it in a given distance. So first of all, the bodyguard should stop at a certain distance when approaching a VIP. Because the smell of a VIP diffuses evenly, the intensity of smell is proportional to the distance to a VIP. Therefore the easiest thing to do is to stop moving forward if the smell exceeds a second upper threshold. The next pseudo code is only slightly modified from the one in Code Example 1 and realizes exactly that behavior.

```
Function Search_VIP()
    loop do
        smell <- actual smell intensity
        direction <- direction from where smell comes from
        newheading <- Setheading(smell,direction)
        set the new heading of agent to value of newheading
        if smell < 550 then move agent one step forward
```

Code Example 3 Pseudo Code for Approaching a VIP and Stopping When Close Enough

Note that only the last line has changed compared to Code Example 1. The function Setheading() is the same like in Code Example 1 and is not repeated here.

Code Example 4 shows the corresponding NetLogo code.

```
if value-from (patch-here) [ smell ] >= 5 [ set heading uphill smell ]
if value-from (patch-here) [ smell ] <= 550 [ fd 1 ]
```

Code Example 4 NetLogo Code for Approaching a VIP and Stopping When Close Enough

Note that the only difference to Code Example 2 is that the agent moves only forward (fd 1) if the smell doesn't exceed a specific value, otherwise it stops moving.

Again there is a counterpart from the Braitenberg vehicles: Vehicle 3 (Fig. 3-3) [22]. This vehicle has also two sensors and two motors. But this time the sensors

are inhibitory, which means that the weaker the input the stronger the motor turns and vise versa. Furthermore the connections between sensors and motors are straight. This vehicle acts exactly as the agent described above: It approaches a source (light, smell or whatever) until the intensity of light, smell or whatever exceeds a certain limit and then stops.

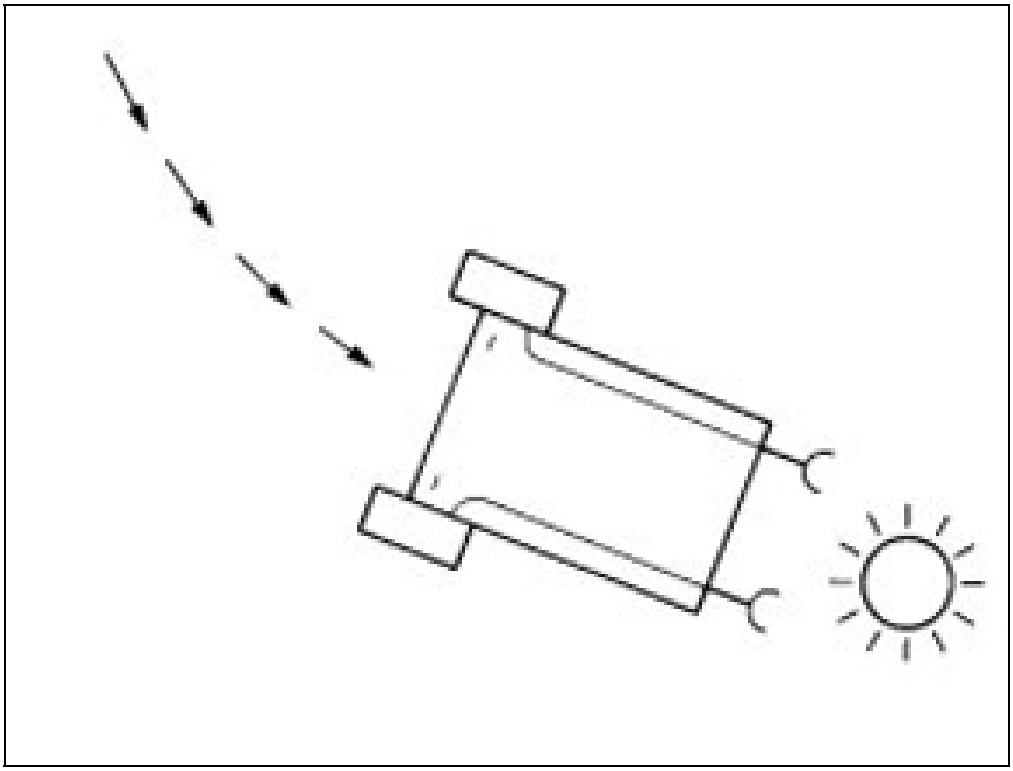

Fig. 3-3 Braitenberg Vehicle, Type 3

Of course this behavior is not satisfactory yet. What if the VIP is moving towards the bodyguard? In this case they might collide. So the next thing to do would be that the bodyguard moves away from the VIP if it comes too close. And one other modification greatly improves the behavior of the bodyguards: If the smell of a VIP lays in-between two upper thresholds, the bodyguard aligns its heading to the VIP. Hence the bodyguard will follow the VIP always in a tunable distance. Note that for this capability the vision of the bodyguards has to be increased. So far the only capability the bodyguards needed was to sense smell and to determine the direction where the smell comes from. This can be done by using purely local information. In NetLogo e.g. the bodyguards only have to watch the patch they are on and the direct neighbor patches to determine the smell intensity and the direction where the smell comes from. Now they also need to be able to determine the heading of the VIP they want to follow. But still we don't need to provide any global information to the bodyguards, because the heading of the VIP is only relevant when the bodyguards are very close to the VIP. But of course this ability increases the complexity of the bodyguard agents, especially if we think of a real robot implementation. It would increase the complexity because now the robots would not

only have to be able to determine distances to each other, but also to determine the heading of other robots. To determine the distance to another robot should be quite easy, e.g. with a laser sensor, but to determine the heading of other robots is probably a very different beast. But to discuss the problems that would arise when applying the proposed approaches to a real robot application goes beyond the scope of this thesis.

The bodyguard agents we have developed so fare consider three different thresholds (Fig. 3-4). In the vertical the threshold values are given (5, 300 and 550). Those values are obtained experimentally and have no biological or physical relevance. The area around a VIP is divided into four zones (1, 2, 3, 4, in the horizontal), bounded by the three thresholds.

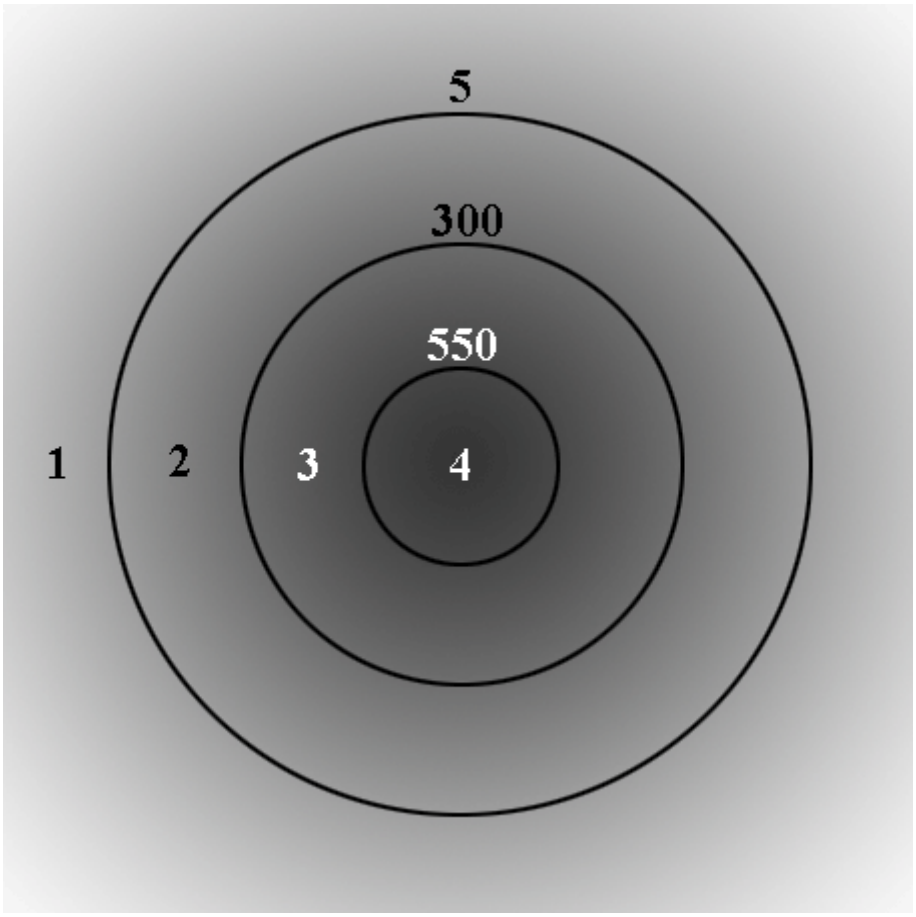

Fig. 3-4 The Different Thresholds and Zones of Smell Around a VIP

The bodyguard agents act on four different rules, based on the intensity of smell. Here are the rules:

- If the bodyguard is too far away from a VIP and can't smell it, it turns randomly (zone 1)
- If the bodyguard can smell a VIP but the smell doesn't exceed 300, it turns towards the VIP (zone 2)
- If the smell exceeds 300 but is smaller than 550, the bodyguard aligns its heading with the heading of the VIP (zone 3)
- If the smell exceeds 550, the bodyguard turns away from the VIP (zone 4)
- After the heading has been set by these rules, the bodyguard will move one patch forward

In Code Example 6 an agent is presented that implements the rules from above. Only the function that sets the new heading is modified from Code Example 1, the main function searchVIP() hasn't changed and is not repeated.

```
Function Setheading(smell,direction,headingVIP) returns a new heading
    newheading <- a random value
    if smell > 5 and smell <= 300 then newheading <- direction
    if smell > 300 and smell < 550 then newheading <- headingVIP
    if smell > 550 then newheading <- direction - 180
    return newheading
```

Code Example 5 Pseudo Code of Bodyguard First Attempt

Code Example 6 shows the corresponding NetLogo code.

```
 if value-from (patch-here) [ smell ] >= 5 and value-from (patch-here) [
smell ] <= 300
          [ set heading uphill smell ]

if value-from (patch-here) [ smell ] > 300 and value-from (patch-here) [
smell ] < 550
         [
      set heading value-from vip [ heading ]
         ]
if value-from (patch-here) [ smell ] >= 550
          [ set heading downhill smell ]
fd 1
```

Code Example 6 NetLogo Code of Bodyguard First Attempt

Note that each agent moves one step forward in each time step (fd 1). The rules above are implemented solely by setting the heading dependent on the intensity of smell. Fig. 3-5 shows a screenshot of a simulation with one VIP and 10 bodyguards from the first attempt. The VIP agent is the ant in the middle of the bright area. Only 8 bodyguards can be seen, 2 bodyguards are outside the visible range, because they haven't found the VIP yet. The bright area around the VIP shows the intensity of smell: the brighter the stronger the smell. The screenshot is taken approximately 30 seconds after starting the simulation.

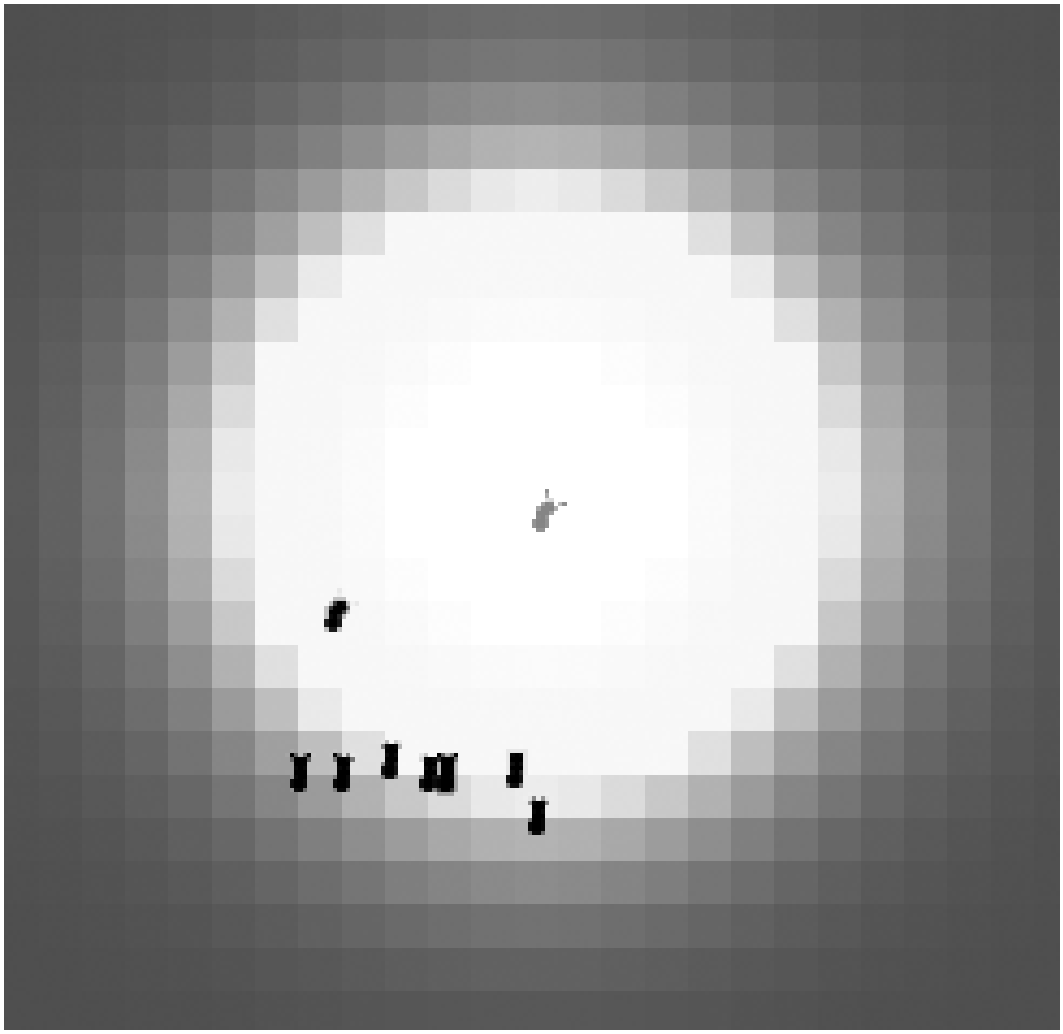

Fig. 3-5 Simulation Screenshot with Bodyguard First Attempt

As it can be seen in Fig. 3-5 the performance of the bodyguard agent first attempt is not satisfactory. Instead of encircling the VIP, the bodyguards end up behind the VIP, following it all very close to each other.

3.3.3 Distributing Evenly Around (Encircling) a VIP

By changing only one of the rules from above (3.3.2 below Fig. 3-4) the performance of the bodyguards can be increased greatly. The idea is that instead of aligning their heading with the heading of the VIP the bodyguards steer away from each other when they are in a desired distance to the VIP. The easiest way to do that is that each bodyguard steers away from its nearest neighbor. This will lead to an even distribution after a while. This behavior is inspired by a physical self-organizing system drafted in Fig. 3-6.

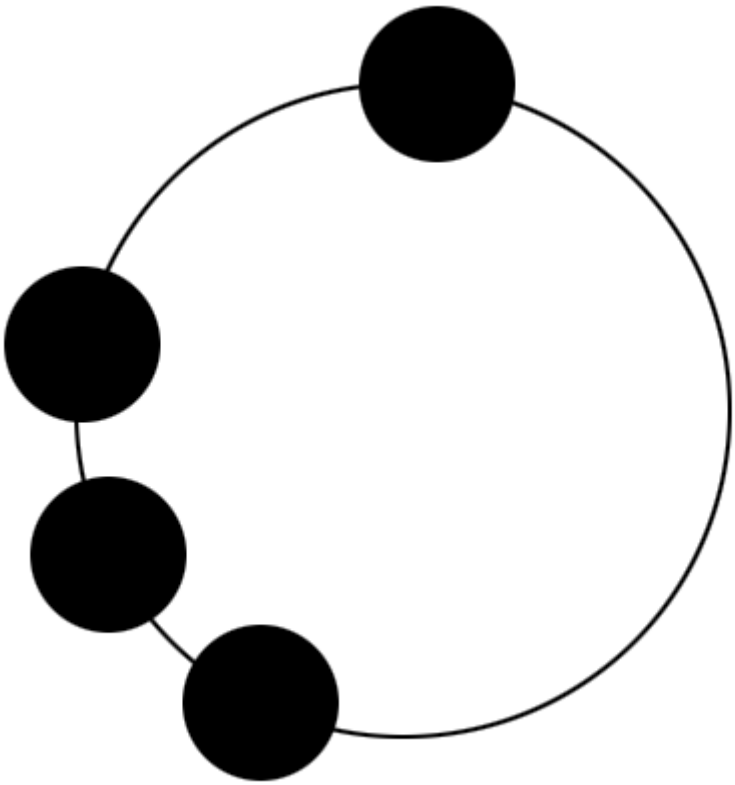

Fig. 3-6 Self-Organizing Physical System- Initial Situation

In this example some objects are supported on a ring so that they can move freely in the tangential direction. If these objects exert a repelling force on each other, the system will end up in a stable state where the distances between the objects will be equally. This is drafted in Fig. 3-7.

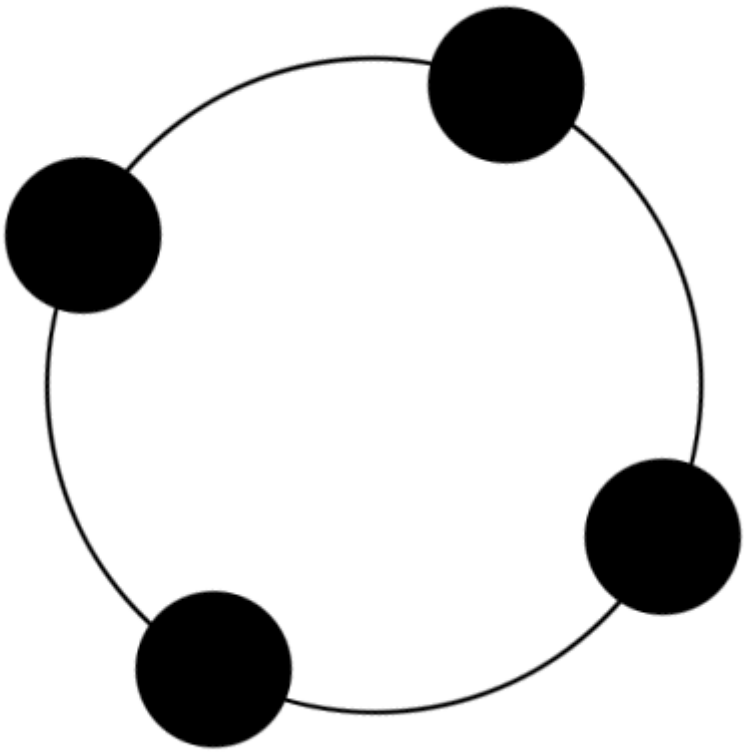

Fig. 3-7 Self-Organizing Physical System in Stable State

Note that this system works independently of the amount of objects that are supported on the ring. The same is true for the modeled behavior: The bodyguards of version 1 will always distribute quite evenly around a VIP.

So we need to change the rule that let the bodyguards align their headings with the VIP when they are in the desired distance to the VIP. Instead they have to steer away from their nearest neighbor. An efficient way to do that is to turn away in a 90 degrees angle relative to the smell gradient. Fig. 3-8 is similar to Fig. 3-4, but it shows how ants would align their headings according to the new rules. The three ants in zone 3 show in which angle they would steer away from each other.

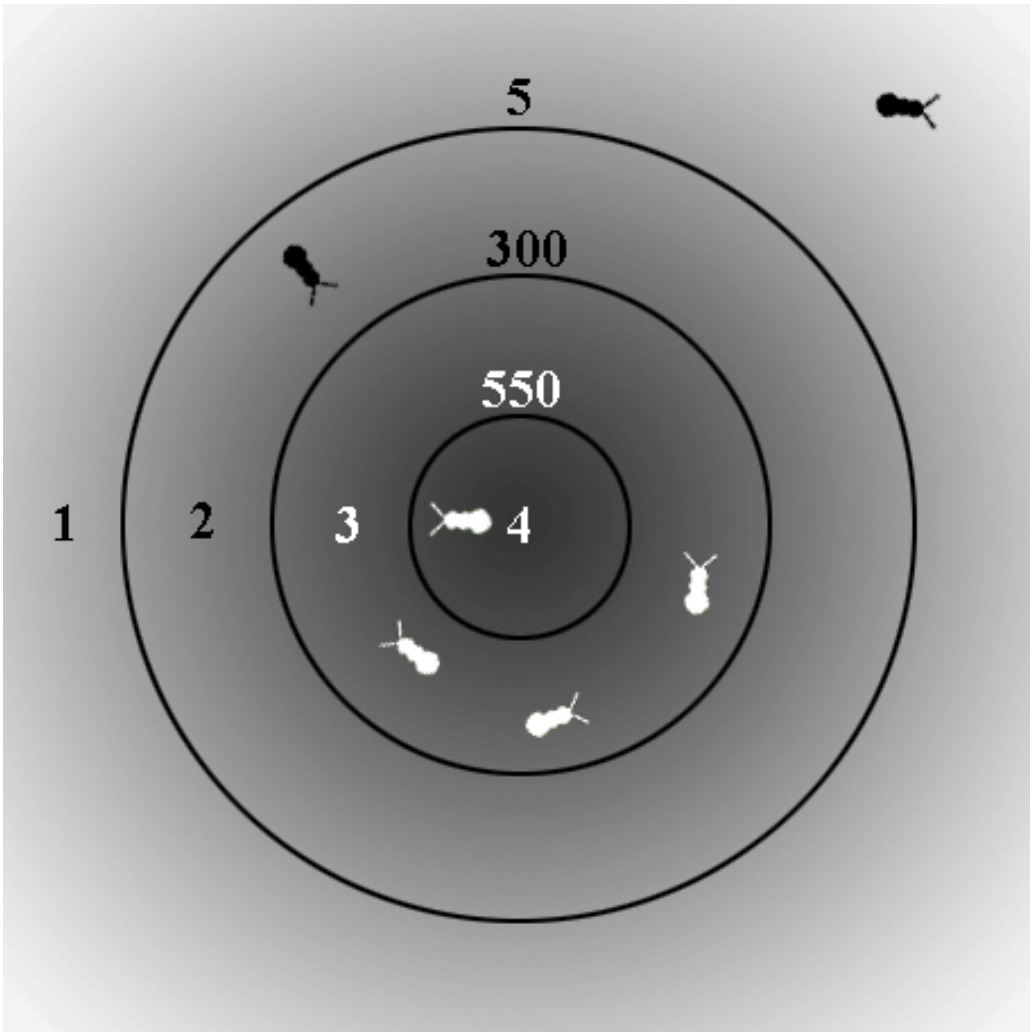

Fig. 3-8 The Different Thresholds and Zones for the Bodyguard Version 1

This is the complete set of rules that each bodyguard of the version 1 follows:

- If the bodyguard is too far away from a VIP and can't smell it, it turns randomly (zone 1)
- If the bodyguard can smell a VIP but the smell doesn't exceed 300, it turns towards the VIP (zone 2)
- If the smell exceeds 300 but is smaller than 550, the bodyguard turns away from its nearest neighbor (zone 3)
- If the smell exceeds 550, the bodyguard turns away from the VIP (zone 4)
- After the heading is set by these rules, the bodyguard moves two patches forward

Note that this time the bodyguards move 2 patches forward. They have to move faster to make up for the extra way they have to move to get away from their nearest neighbor.

In Code Example 7 the pseudo code for this type of bodyguard is given, in Appendix A the complete NetLogo code is listed.

```
Function Protect_VIP()
    loop do
        smell <- actual smell intensity
        direction <- direction from where smell comes from
        newheading <- Setheading(smell,direction)
        set the new heading of agent to value of newheading
        move agent 2 patches forward
```

```
Function Setheading(smell,direction) returns a new heading
    newheading <- a random value
    if smell > 5 and smell < 300 then newheading <- direction
    if smell >= 300 and smell < 550 then newheading <- 90 degrees
                relative to smell gradient away from nearest neighbor
    if smell  >= 550 newheading <- direction - 180
    return newheading
```

Code Example 7 Pseudo Code of Bodyguard Version 1

One difference to the physical system described above is that the bodyguard agents will always move away from their nearest neighbors, even if the distances are already equally. Unlike the physical system, which “knows” when the distances between the objects are equally and goes into a stable state, the bodyguards don’t know when they are distributed evenly around the VIP. Therefore they will keep on moving away from their nearest neighbor all the time. This leads to a very anxious looking behavior, the bodyguards move forth and back all the time. Another disadvantage is that they have to move much faster than the VIPs to keep up with them, because of the extra way the move forth and back. Furthermore, thinking of a real robot implementation, this behavior would be even less efficient because each time the robots would have to turn around, they would loose additional time. Nevertheless the agents perform pretty well in the simulations. Fig. 3-9 shows screenshots of two simulation runs with the bodyguard version 1.

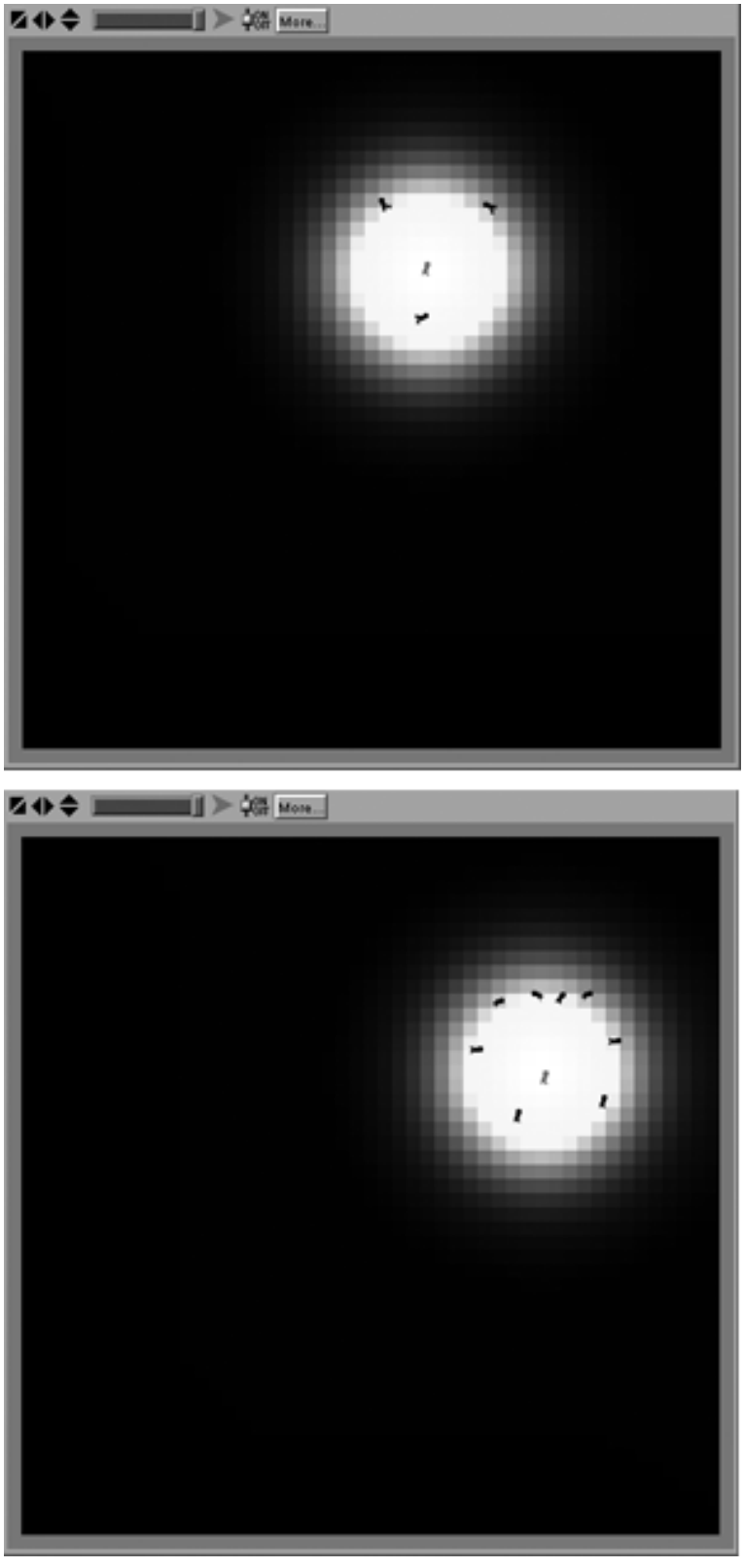

Fig. 3-9 Simulation Run with Bodyguard Version 1

In the upper screenshot one VIP and three bodyguards were deployed, in the lower screenshot one VIP and 8 bodyguards were deployed. Both screenshots were taken about 20 seconds after starting the simulation.

As it can be seen in the screenshots, another disadvantage of this type of bodyguard is that they tend to end up rather behind the VIP.

3.3.4 Further Improvement of the Bodyguard Behavior

The bodyguards presented in 3.3.3 perform satisfactorily, but they have a little drawback: they move very anxious forth and back, even if the bodyguards are distributed evenly around the VIP. It would be better if they would align their headings with the VIP and follow it, once they are distributed evenly around the VIP. The bodyguards presented in 3.3.2 align their headings with the heading of the VIP and follow it in the desired distance, but they don't distribute evenly around the VIP. A combination of both agent types might be a good solution. Indeed this can be achieved: By introducing a new smell zone and a new rule the bodyguards first distribute evenly around the VIP and then follow it in the desired distance without moving forth and back all the time as the bodyguard from 3.3.3 does. The thresholds and zones for this bodyguard agent are shown in Fig. 3-10.

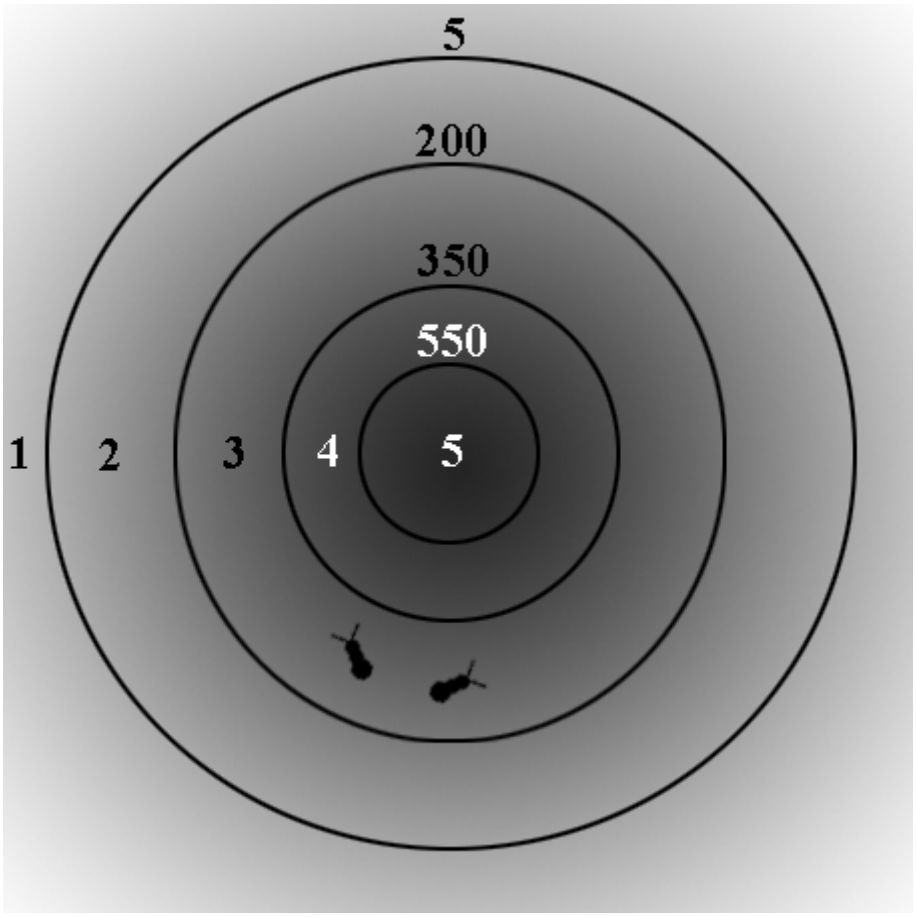

Fig. 3-10 Thresholds and Zones for Bodyguards Version 2

In zone 1 the bodyguards moves randomly, in zone 2 the bodyguards climb up the smell gradient (just like the bodyguard version 1 from 3.3.3). New is that in zone 3 the bodyguards steer away from each other, but not in a 90 degrees angle relative to the smell gradient (like the bodyguard version 1), but rather in a smaller angle, e.g. 80 degrees. By doing that the bodyguards distribute evenly around the VIP and they approach the VIP at the same time, until they get into zone 4. In zone 4 the bodyguards align their heading with the VIP and follow the VIP. In this zone they don't worry about the distance they have to each other, because it is assumed that they already distributed evenly around the VIP in zone 3. In zone 5 they just climb down the smell gradient, like the bodyguards have already done before.

This is the complete set of rules that each bodyguard of the version 2 follows:

- If the bodyguard is too far away from a VIP and can't smell it, it turns randomly (zone 1)
- If the bodyguard can smell a VIP but the smell doesn't exceed 200, the bodyguard turns towards the VIP (zone 2)
- If the smell exceeds 200 but is smaller than 350, the bodyguard turns away from its nearest neighbor (zone 3)
- If the smell exceeds 350 but is smaller than 550, the bodyguard aligns its heading with the heading of the VIP (zone 4)
- If the smell exceeds 550, the bodyguard turns away from the VIP (zone 5)
- After the heading is set by these rules, the bodyguard moves two patches forward. But if it is in zone 4 it moves with the same peed as the VIP.

The pseudo code for the bodyguard version 2 is given in Code Example 8. The complete NetLogo Code is given in Appendix B.

```
Function Protect_VIP()
    loop do
        smell <- actual smell intensity
        direction <- direction from where smell comes from
        headingVIP <- the actual heading of the VIP
        newheading <- Setheading(smell,direction,headingVIP)
        set the new heading of agent to value of newheading
        patchestomove <- Move(smell)
        move agent patchestomove patches forward
```

```
Function Setheading(smell,direction,headingVIP) returns a new heading
    newheading <- a random value
    if smell > 5 and smell < 200 then newheading <- direction
    if smell >= 200 and smell < 350 then newheading <- 80 degrees
                relative to smell gradient away from nearest neighbor
    if smell >= 350 and smell < 550 then newheading <- headingVIP
    if smell  >= 550 then newheading <- direction - 180
    return newheading
```

```
Function patchestomove(smell)
    if smell >= 350 and smell < 550 then return 1
    else return 2
```

Code Example 8 Pseudo Code of Bodyguard Version 2

The bodyguard version 2 performs similar as the bodyguard version 1 but has the advantage that the agents follow the VIP smoothly instead of moving anxiously forth and back all the time. But it has also a little drawback: If the VIP is already protected by a group of bodyguards and a new bodyguard joins, the others don't rearrange. They only rearrange if all bodyguards have to rearrange, e.g. after bouncing from the borders. This can be seen in Fig. 3-11. The screenshots show the following situations (from top left to bottom right):

First, shortly after starting the simulation two bodyguards found the VIP. They distributed around in the biggest distance to each other.

Shortly after that, the third bodyguard also found the VIP. The third bodyguard tries to keep a maximum distance to the two other bodyguards, but the two other bodyguards don't rearrange.

Then the VIP bounces from the border. Because of that perturbation the bodyguards have to rearrange.

After rearranging, the bodyguards are distributed quite evenly around the VIP.

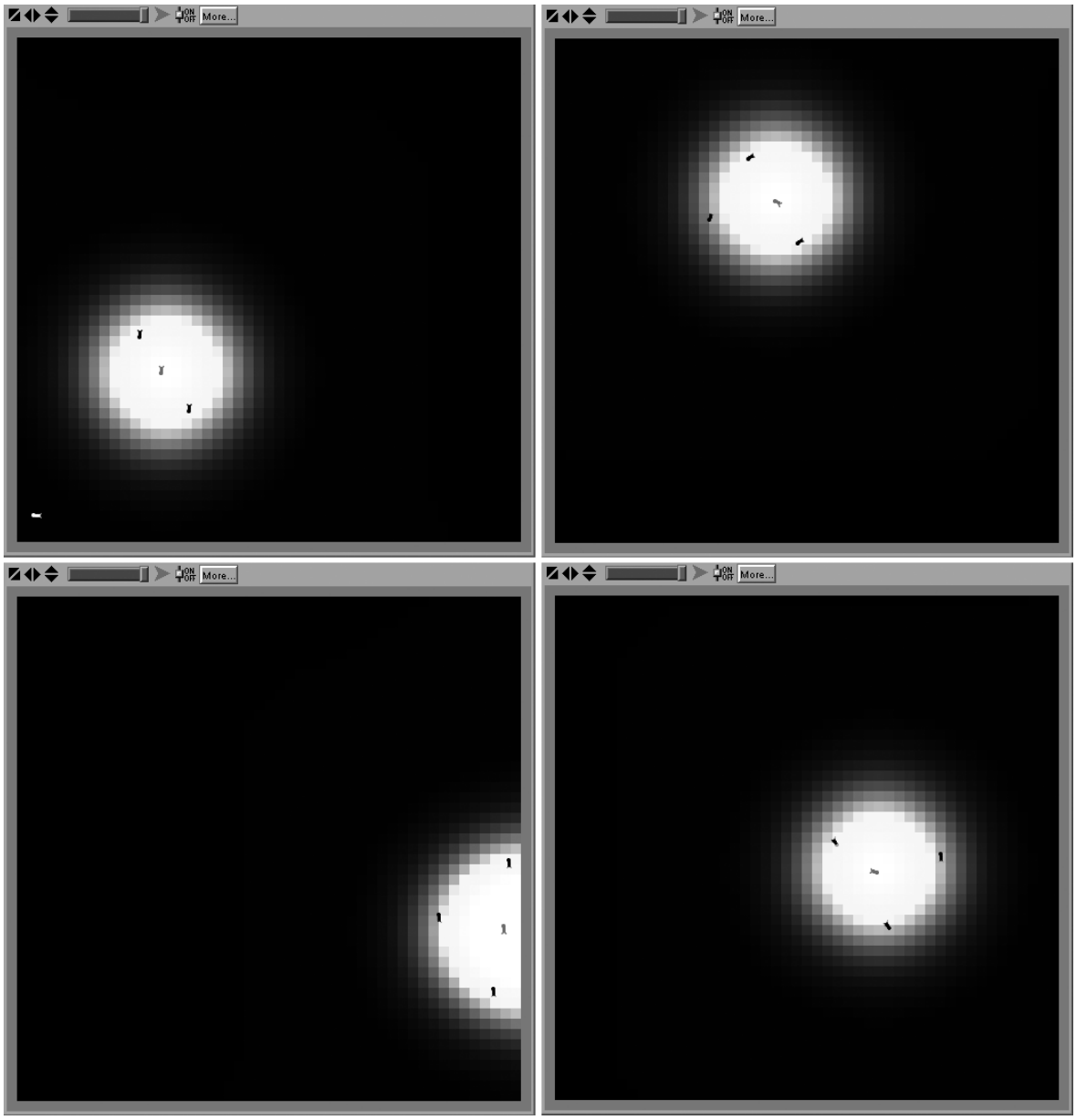

Fig. 3-11 NetLogo Simulation of Bodyguards Version 2

3.3.5 Protecting More Than One VIP

As we saw in the last chapter the bodyguards do quite a good job in protecting one VIP. But how do they perform with more than one VIP? How do we need to adjust the behavior of the bodyguards so that they can protect more than one VIP? The surprising answer is that we don't have to do anything. The bodyguards as described in the former chapter work pretty well even for more than one VIP. In Fig. 3-12 a simulation run is shown in which 15 bodyguards try to protect 3 VIPs. The

initial situation is shown in the top left, the last screenshot is the bottom right. From each screenshot to the next approximately 20 seconds passed.

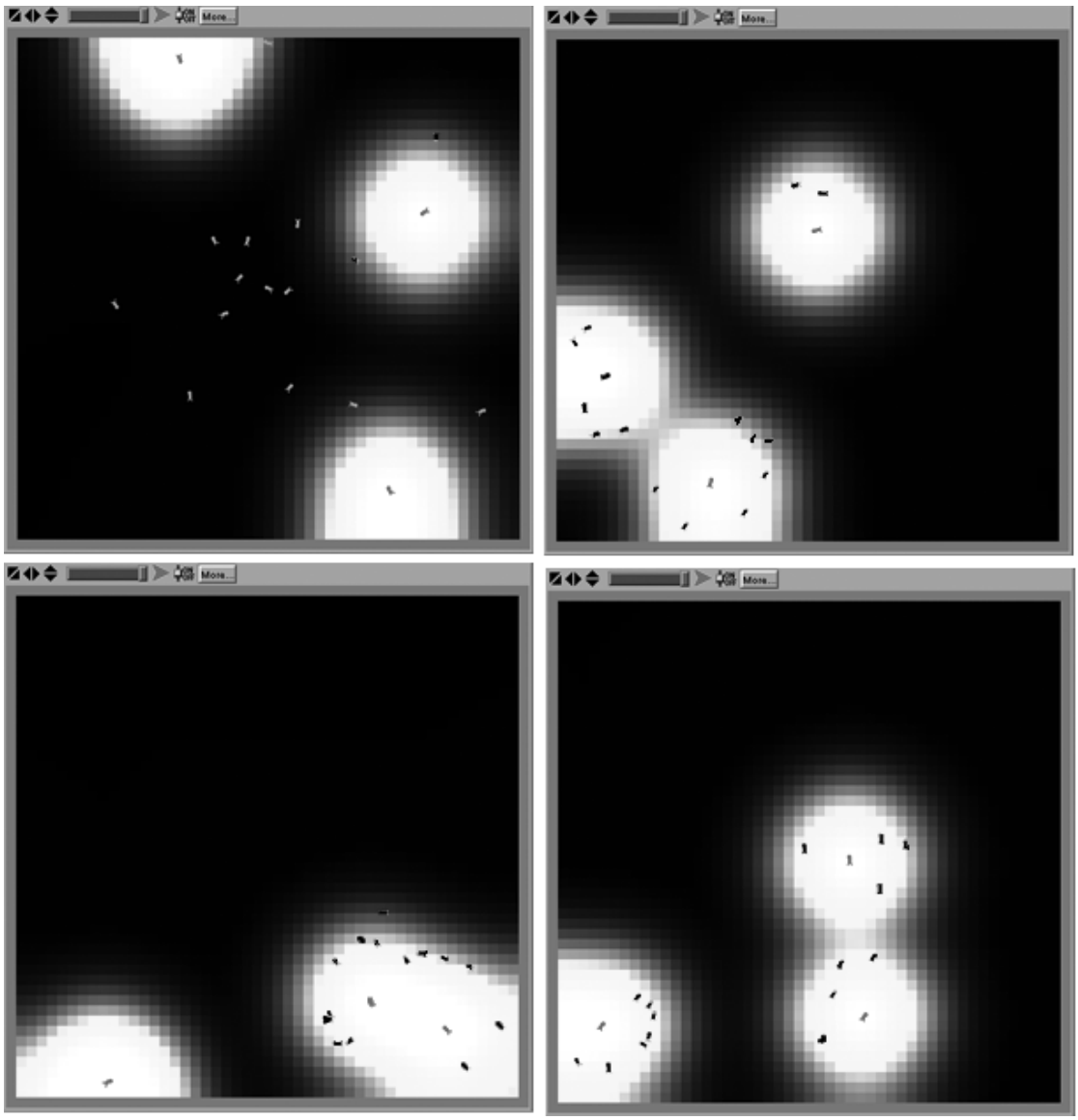

Fig. 3-12 NetLogo Simulation with 3 VIPs and 15 Bodyguards

Even though the bodyguard agents are doing a good job in most situations, sometimes a VIP is left alone, while others are protected by more bodyguards than necessary. This can be seen in the third screenshot. Several attempts to overcome this problem have been undertaken, unfortunately without any success: It seems to be impossible to solve this problem without providing global information, which is quite plausible. Dependent on the number of VIPs and the number of bodyguards, the bodyguards would have to act slightly different. But no agent knows how much VIPs are to be protected nor does any agent know how much bodyguards there are altogether. Of course it is possible to tune the parameters that it

works for a given situation, let's say 3 VIPs and 15 bodyguards. But obviously there is no generic solution to this problem.

Of course one very simple solution to this problem is to deploy a huge amount of bodyguards. By doing so it is very unlikely that one VIP will be left alone.

But even though that problem couldn't be solved, the bodyguards work satisfactorily most of the time. Without any communication or synchronization they tend to divide themselves quite even-numbered to the different VIPs by random fluctuation and self organization.

4 Results and Conclusions

In this thesis a multi agent system with a swarm intelligent approach is developed. An agent type is developed, that in a group is capable of protecting one or more VIPs by finding them, encircling them, and following them in a tunable distance. The group behavior of the bodyguard agents emerges from the rather simple behavior of individuals. No bodyguard is in charge; all bodyguards follow exactly the same set of simple rules based on local information.

It is shown that the swarm intelligent approach can deliver flexible and robust solutions. An example for the flexibility of the system is that the amount of bodyguards deployed can be chosen arbitrarily. The bodyguards perform similarly well for both cases, whether the amount of bodyguards is rather low or rather high. They even perform reasonably in most cases when more than one VIP needs to be protected, especially when many bodyguards are deployed. The interesting aspect is that they do that without having any information about the amount of bodyguards or the amount of VIPs. And they don't need to know it, because the rules work in a way that the knowledge of the total number of agents is not crucial.

The system is robust, because even if some bodyguard agents may fail, e.g. they might take quite long to find a VIP. Yet the bodyguards as a group perform well in most cases. Again, this is especially true for a high number of bodyguards being deployed.

Two different bodyguard types have been developed, both perform similar. Bodyguard version 1 is more reliable in distributing randomly around the VIP but has the disadvantage that it moves anxiously forth and back all the time. Bodyguard version 2 is not as reliable in distributing evenly around the VIP but follows the VIP smoothly without moving forth and back all the time. This makes that agent type more useful, especially in the case of a real robot application.

In simulations it has been shown that these kinds of agents perform satisfactorily in theory. Even though this work could be a starting point for somebody that wants to implement a bodyguard-VIP-scenario on a real robot platform, it is not within the scope of this thesis to discuss problems by doing so. Nor it is intended to show that this approach is better than any other approach. The aim is just to show that simple agents can perform a collaborative task by self-organization and emergence. A swarm-based approach might be outperformed by other approaches in terms of speed for example. But the strength of a swarm-based approach is the simplicity of agents and the flexibility and robustness of the system.

Of course there are not only benefits by using such an approach. In fact swarm intelligent solutions are very hard to design. It's difficult because it is a bottom-up-approach without a clear connection between the low-level interactions and the upper-level global outcome. Indeed it was a long and tiring process to find rules and parameters to make the system working reasonably. And still the parameters found are most likely not optimal. The problem of linking the lower-level agent interactions with the upper-level group behavior with clearly defined paths is a current research topic. A promising way around this problem is to let the parameters being tuned automatically, e.g. by genetic algorithms. But first this approach is beyond the scope of this thesis, and second it doesn't provide a solution on how to

design a swarm intelligent system, because before tuning parameters, rules have to be found that could work, and parameters have to be defined that seem to be relevant for a problem. And this seems to be impossible to be done automatically, at least in the near future.

It was challenging and very interesting to work on that topic. Not only because of the topic of swarm intelligence is rather new, but it is also interdisciplinary. It is very interesting to see how knowledge from different disciplines (like in this case biology) can be used in computer science.

References

[1] E. H. Callaway, Wireless Sensor Networks: Architectures and Protocols. Boca Raton, FL: CRC Press LLC, 2003.

[2] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, “Wireless Sensor Networks for Habitat Monitoring," ACM International Workshop on Wireless Sensor Networks and Applications (WSNA02), Sept. 2002.

[3] F. J. Cilluffo, S. L. Cardash, G. N. Lederman, and C. H. D. Project, Combating
Chemical, Biological, Radiological, and Nuclear Terrorism: A Comprehensive Strategy. Washington, DC: Center for Strategic and International Studies, 2001.

[4] Committee on Army Science and Technology for Homeland Defense, Science and technology for army homeland security. Report 1, ch. Executive Summary, pp. 13-21. Washington, DC: National Academies Press, 2003.

[5] Committee on Army Science and Technology for Homeland Defense, Science and technology for army homeland security. Report 1, ch. Executive Summary, pp. 4-13. Washington, DC: National Academies Press, 2003.

[6] Committee on Army Science and Technology for Homeland Defense, Science and technology for army homeland security. Report 1, ch. chaper 3, Denial and Survivability Technologies. Washington, DC: National Academies Press, 2003.

[7] K. L. Moore, Y. Chen, and S. Zeng, “Diffusion based path planning in mobile actuator-sensor networks (MAS-net) - some preliminary results," Proceedings of SPIE Conf. on Intelligent Computing: Theory and Applications II, part of SPIE's Defense and Security, 2004.

[8] Y. Chen, K. Moore, and Z. Song, „Diffusion boundary determination and zone control via mobile actuator-sensor networks (MAS-net): Challenges and opportunities," Intelligent Computing: Theory and Applications II, part of SPIE's Defense and Security, 2004.

[9] K. Moore and Y. Chen, “Model-based approach to characterization of diffusion processes via distributed control of actuated sensor networks," IFAC Symposium of Telematics Applications in Automation and Robotics, Helsinki University of Technology, 2004.

[10] MAS-mote – A Mobility Node for MAS-net (Mobile Actuator Sensor Networks),
http://www.csois.usu.edu/people/yqchen/paper/04C14_robio_Zhen.pdf

[11] Crossbow, “MICA2”,
http://www.xbow.com/Products/productsdetails.aspx?sid=69

[12] “TinyOS”, http://webs.cs.berkeley.edu/tos/

[13] Eric Bonabeau, Marco Dorigo, Guy Theraulaz, “Swarm Intelligence-From Natural to Artificial Systems”, Oxford University Press, 1999, ISBN: 0-19-513159-2

[14] Hölldobler, B., and E. O. Wilson. The Ants. Cambridge, MA: Harward University Press, 1990

[15] Heinrich, B. “The Regulation of Temperature in the Honey Bee Swarm.” Sci. Am. 244 (1981): 146-160

[16] Dorigo, M., and L. M. Gambardella. “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.” IEEE Trans. Evol. Comp. 1 (1997): 53-66

[17] Dorigo, M., and L.M. Gambardella. “Ant Colonies for the Traveling Salesman Problem.” BioSystems 43 (1997): 73-81

[18] Schoonderwoerd, R., O. Holland, J. Bruten, and L. Rothkrantz. “Ant-Based Load Balancing in Telecommunications Networks.” Adapt. Behav. 5 (1996): 169-207

[19] Di Caro, G., and M. Dorigo. “AntNet: Distributed Stigmergetic Control for Communications Networks.” J. Art. Int. Res. 9 (1998): 317-365

[20] Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[21] Valentino Braitenberg. “Vehicles. Experiments in Synthetic Psychology.” MIT Press, (1984): 6-9

[22] Valentino Braitenberg. “Vehicles. Experiments in Synthetic Psychology.” MIT Press, (1984): 10-14

[23] Craig W. Reynolds. “Flocks, Herds, and Schools: A Distributed Behavioral Model.” http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/

Appendix

A Appendix 1: Complete NetLogo Code of Bodyguard Version 1

```
;Different agent types in this simulation
breeds [ bodyguards vips ] ; In this world we have vips and bodyguards. The vips
;move around randomly. The bodyguards shall protect the vips.

;global variables
globals
[ real-patches ;; patches in which to diffuse the to-diffuse variable
  border       ;; patches for whom the value of t-diffuse should remain 0
  vips-movements ;; counter for the vips movements, every 10th movement make
;a random turn
  mouse-clicked ;; keeps track of click-and-hold
  clicked-turtle ;; who was clicked on...
]

;patches variables
patches-own
[ smell ;; the smell of each patch. the patch where the vips are on, has the maxi-
;mum smell. this smell is diffused to the neighbor patches
  my-real-neighbors  ;; all "valid" neighbors of a patch- no wrapping is allowed
]

;turtles variables
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
]

;setting up the initial situation
to setup
  ca ;; clear the screen
  ;Setup vehicles and patches
  setup-bodyguards
  setup-vips
  setup-patches
  set mouse-clicked false
  set clicked-turtle nobody
end

;creating and ditributing the bodyguard agents
to setup-bodyguards
  create-bodyguards bodyguard-count ;; create the bodyguards
  ask turtles [
    set shape "bug"
    set size agents-size
    set color green
    ;;distribute the bodyguards randomly
```

```
    fd (random screen-edge-x)
    lt random 360
  ]
end

;creating and distributing the vip agents
to setup-vips
  create-vips vips-count
  ask vips [
    set shape "bug"
    set size agents-size
    set color red
    ;;distribute the vips randomly
    lt random 360
    fd (random screen-edge-x)
  ]
end

;setting up the simulation world
to setup-patches
  ;; set the border pachtes, where no diffusion takes place
  set border patches with [ abs pxcor = screen-edge-x
                          or abs pycor = screen-edge-y ]
  ;; set the rest of the patches, where diffusion of smell takes place
  set real-patches patches with [ abs pxcor < screen-edge-x
                                and abs pycor < screen-edge-y ]
  ask patches [
    set my-real-neighbors neighbors with
      [abs pxcor < screen-edge-x and abs pycor < screen-edge-y]
  ]
  ;Color the border patches as boundary
  ask border [
    set pcolor grey
  ]
  ;; Call the procedure which sets the smell on the patches where the vips are on
  set-smell-source
end

;setting the smell variable on all patches with vips and distributin over neighbor
;patches
to set-smell-source
  ;Reset former smell-sources
  ask real-patches [ set smell 0 ]
  ;Setup the patches where a vip is on as the smell sources and diffuse smell
  let value 0
  ask patches with [any? vips-here] [ set smell 90000]
  repeat 25 [
    diffuse smell 1.0
    without-interruption
    [ ask border [
```

```
        set value (smell / (count my-real-neighbors))
        ask my-real-neighbors [ set smell (smell + value) ]
        set smell 0
    ] ]
  ]
  ;; paint the smell if desired
  ifelse (show-smell-of-vips = true) [
    ;Color the patches according to their brightness
    ask real-patches [
      set pcolor scale-color yellow smell 0 730
    ]
  ]
  ;; if not set all real patches to black
  [ ask real-patches [ set pcolor black ] ]
end

;; run the simulation, called by NetLogo System automatically at each time step
to go
  ask turtles [
    set-heading
  ]
  ask turtles [
    bounce
  ]
  move-turtles
  set-smell-source
end

;moving the turtles forward
to move-turtles
  ask vips [
    fd vip-speed
  ]
  ask bodyguards [
    fd bodyguards-speed
  ]
  set vips-movements (vips-movements + 1)
end

;setting the heading of the turtles
to set-heading
  ;heading of bodyguards
  if breed = bodyguards [
    ;random heading
    if value-from (patch-here) [ smell ] < 5 [
      lt random 45
      rt random 45
    ]
    ;climb up smell gradient
    if value-from (patch-here) [ smell ] >= 5 and value-from (patch-here) [ smell ] <=
```

```
300 [ set heading uphill smell ]
   ;move away from nearest neighbor in a 90 degrees angle from direction to vip
   if value-from (patch-here) [ smell ] > 300 and value-from (patch-here) [ smell ] <
550 [
   find-flockmates
   ifelse any? flockmates
    [ find-nearest-neighbor
       ;only if turtles are not on the same patch- otherwise simulation runtime error
       ifelse patch-here != value-from (nearest-neighbor) [ patch-here ] [
          set flockmates vips
          let nearest-vip min-one-of flockmates [distance myself]
          let vipheadingright towards nearest-vip + 90
          let vipheadingleft towards nearest-vip – 90
          let rightpatch patch-at-heading-and-distance vipheadingright bodyguards-
speed
          let leftpatch patch-at-heading-and-distance vipheadingleft bodyguards-
speed
          let distanceleft value-from (nearest-neighbor) [ distance-nowrap leftpatch ]
          let distanceright value-from (nearest-neighbor) [ distance-nowrap right
patch ]
          ifelse distanceleft > distanceright [ set heading vipheadingleft ]
            [ set heading vipheadingright ]
       ]
       ;if turtles are on same patch turn randomly
       [
         lt random 90
         rt random 90
       ]
     ]
     [
         set flockmates vips
         let nearest-vip min-one-of flockmates [distance myself]
         set heading value-from nearest-vip [heading]
     ]
   ]
    ;climd down smell gradient
    if value-from (patch-here) [ smell ] >= 550 [ set heading downhill smell ]
  ]
  ;heading of vips
  if breed = vips [
    if remainder vips-movements 10 = 0 [
      lt random 90
      rt random 90
    ]
  ]
end

;find all bodyguards in radius of vision
to find-flockmates  ;; turtle procedure
  set flockmates (bodyguards in-radius vision) with [self != myself]
```

```
end

;find the nearest flockmate
to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end

;; this procedure checks the coordinates and makes the turtles
;; reflect according to the law that the angle of reflection is
;; equal to the angle of incidence
to bounce
  ; check: hitting left or right wall?
  if (abs pxcor-of patch-ahead 1 = screen-edge-x)
    ; if so, reflect heading around x axis
    [ set heading (- heading) ]
  ; check: hitting top or bottom wall?
  if (abs pycor-of patch-ahead 1 = screen-edge-y)
    ; if so, reflect heading around y axis
    [ set heading ( 180 - heading ) ]
end

;help procedure for drag-and-drop with mouse
to drag-drop
    ;; detects a single mouse click
    if not mouse-clicked and mouse-down?
    [
      set mouse-clicked true
      if any? turtles-at round mouse-xcor round mouse-ycor
      [  set clicked-turtle random-one-of turtles-at round mouse-xcor round mouse-
ycor
      ]
    ]
    ;; if a turtle is only clicked, then it moves to match the mouse
    if is-turtle? clicked-turtle
    [  ask clicked-turtle
       [ setxy mouse-xcor mouse-ycor ]
       if breed-of clicked-turtle = vips [ set-smell-source ]
    ]
    ;; detects raising the mouse button
    if mouse-clicked and not mouse-down?
    [  set mouse-clicked false
       if is-turtle? clicked-turtle
       [ set clicked-turtle nobody]
    ]
end
```

B Appendix 2: Complete NetLogo Code of Bodyguard Version 2

```
;Different agent types in this simulation
breeds [ bodyguards vips ] ; In this world we have vips and bodyguards. The vips
move around randomly. The bodyguards shall protect the vips.

;global variables
globals
[ real-patches ;; patches in which to diffuse the to-diffuse variable
  border       ;; patches for whom the value of t-diffuse should remain 0
  vips-movements ;; counter for the vips movements, every 10th movement make
a random turn
  mouse-clicked ;; keeps track of click-and-hold
  clicked-turtle ;; who was clicked on...
]

;patches variables
patches-own
[ smell ;; the smell of each patch. the patch where the vips are on, has the maxi-
mum smell. this smell is diffused to the neighbor patches
  my-real-neighbors  ;; all "valid" neighbors of a patch- no wrapping is allowed
]

;turtles variables
turtles-own [
  flockmates         ;; agentset of nearby turtles
  nearest-neighbor   ;; closest one of our flockmates
]

;setting up the initial situation
to setup
  ca ;; clear the screen
  ;Setup vehicles and patches
  setup-bodyguards
  setup-vips
  setup-patches

  set mouse-clicked false
  set clicked-turtle nobody
end

;creating and ditributing the bodyguard agents
to setup-bodyguards
  create-bodyguards bodyguard-count ;; create the bodyguards
  ask turtles [
    set shape "bug"
    set size agents-size
    set color green
    ;;distribute the bodyguards randomly
    fd (random screen-edge-x)
```

```
    lt random 360
  ]
end

;creating and distributing the vip agents
to setup-vips
  create-vips vips-count
  ask vips [
    set shape "bug"
    set size agents-size
    set color red
    ;;distribute the vips randomly
    lt random 360
    fd (random screen-edge-x)
  ]
end

;setting up the simulation world
to setup-patches
  ;; set the border pachtes, where no diffusion takes place
  set border patches with [ abs pxcor = screen-edge-x
                          or abs pycor = screen-edge-y ]
  ;; set the rest of the patches, where diffusion of smell takes place
  set real-patches patches with [ abs pxcor < screen-edge-x
                                and abs pycor < screen-edge-y ]
  ask patches [
    set my-real-neighbors neighbors with
      [abs pxcor < screen-edge-x and abs pycor < screen-edge-y]
  ]
  ;Color the border patches as boundary
  ask border [
    set pcolor grey
  ]
  ;; Call the procedure which sets the smell on the patches where the vips are on
  set-smell-source

end

;setting the smell variable on all patches with vips and distributin over neighbor
patches
to set-smell-source
  ;Reset former smell-sources
  ask real-patches [ set smell 0 ]
  ;Setup the patches where a vip is on as the smell sources and diffuse smell
  let value 0
  ask patches with [any? vips-here] [ set smell 90000]
  repeat 25 [
    diffuse smell 1.0
    without-interruption
    [ ask border [
```

```
        set value (smell / (count my-real-neighbors))
        ask my-real-neighbors [ set smell (smell + value) ]
        set smell 0
    ] ]
  ]
  ;; paint the smell if desired
  ifelse (show-smell-of-vips = true) [
    ;Color the patches according to their brightness
    ask real-patches [
      set pcolor scale-color yellow smell 0 730
    ]
  ]
  ;; if not set all real patches to black
  [ ask real-patches [ set pcolor black ] ]
end

;; run the simulation, called by NetLogo System automatically at each time step
to go
  ask turtles [
    set-heading
  ]
  ask turtles [
    bounce
  ]
  move-turtles
  set-smell-source
end

;moving the turtles forward
to move-turtles
  ask vips [
    fd vip-speed
  ]
  ask bodyguards [
      ifelse value-from (patch-here) [ smell ] > 350 and value-from (patch-here) [
smell ] < 550
        [ fd vip-speed ]
        [ fd bodyguards-speed ]
  ]
  set vips-movements (vips-movements + 1)
end

;setting the heading of the turtles
to set-heading
  ;heading of bodyguards
  if breed = bodyguards [
    ;random heading
    if value-from (patch-here) [ smell ] < 5 [
      lt random 45
      rt random 45
```

```
    ]
    ;climb up smell gradient
    if value-from (patch-here) [ smell ] >= 5 and value-from (patch-here) [ smell ] <=
200 [ set heading uphill smell ]
    ;move away from nearest neighbor in a 70 degrees angle from direction to vip
    if value-from (patch-here) [ smell ] > 200 and value-from (patch-here) [ smell ]
<= 350
    [
      ;;Separation
      find-flockmates
      ifelse any? flockmates
        [ find-nearest-neighbor
          ifelse patch-here != value-from (nearest-neighbor) [ patch-here ] [
            set flockmates vips
            let nearest-vip min-one-of flockmates [distance myself]
            let vipheadingright towards nearest-vip + 70
            let vipheadingleft towards nearest-vip - 70
            let rightpatch patch-at-heading-and-distance vipheadingright body-
guards-speed
            let leftpatch patch-at-heading-and-distance vipheadingleft bodyguards-
speed
            let distanceleft value-from (nearest-neighbor) [ distance-nowrap leftpatch
]
            let distanceright value-from (nearest-neighbor) [ distance-nowrap right-
patch ]
            ifelse distanceleft > distanceright [ set heading vipheadingleft ]
              [ set heading vipheadingright ]
          ]
          [
            lt random 90
            rt random 90
          ]
      ]
      [
        set heading uphill smell
      ]
    ]
    ;align heading with vip
    if value-from (patch-here) [ smell ] > 350 and value-from (patch-here) [ smell ] <
550 [
      set flockmates vips
      let nearest-vip min-one-of flockmates [distance myself]
      set heading value-from nearest-vip [ heading ]
    ]
    ;climb down smell gradient
    if value-from (patch-here) [ smell ] >= 550 [ set heading downhill smell ]
  ]
  ;heading of vips
  if breed = vips [
   if remainder vips-movements 10 = 0 [
```

```
      lt random 90
      rt random 90
    ]
   ]
end

;find all bodyguards in radius of vision
to find-flockmates  ;; turtle procedure
  set flockmates (bodyguards in-radius vision) with [self != myself]
end

;find the nearest flockmate
to find-nearest-neighbor ;; turtle procedure
  set nearest-neighbor min-one-of flockmates [distance myself]
end


;; this procedure checks the coordinates and makes the turtles
;; reflect according to the law that the angle of reflection is
;; equal to the angle of incidence
to bounce
  ; check: hitting left or right wall?
  if (abs pxcor-of patch-ahead 1 = screen-edge-x)
    ; if so, reflect heading around x axis
    [ set heading (- heading) ]
  ; check: hitting top or bottom wall?
  if (abs pycor-of patch-ahead 1 = screen-edge-y)
    ; if so, reflect heading around y axis
    [ set heading ( 180 - heading ) ]
end


;help procedure for drag-and-drop with mouse
to drag-drop
    ;; detects a single mouse click
    if not mouse-clicked and mouse-down?
    [
      set mouse-clicked true
      if any? turtles-at round mouse-xcor round mouse-ycor
      [  set clicked-turtle random-one-of turtles-at round mouse-xcor round mouse-
ycor
      ]
    ]
    ;; if a turtle is only clicked, then it moves to match the mouse
    if is-turtle? clicked-turtle
    [  ask clicked-turtle
       [ setxy mouse-xcor mouse-ycor ]
       if breed-of clicked-turtle = vips [ set-smell-source ]
    ]
    ;; detects raising the mouse button
```

```
  if mouse-clicked and not mouse-down?
  [  set mouse-clicked false
     if is-turtle? clicked-turtle
     [ set clicked-turtle nobody]
  ]
end
```