The group movement of Aptenodytes forsteri implemented as an 'emergent' phenomenon

Thesis M.Sc. graduate research project

Albert van der Heide
Artificial Intelligence
University of Groningen

April 17, 2003

Advisors:

dr. Martijn Schut, Vrije Universiteit Amsterdam[1]
dr. Rineke Verbrugge, University of Groningen[2]

[1] Department of Artificial Intelligence, Faculty of Sciences, Vrije Universiteit,
De Boelelaan 1081, 1081 HV, Amsterdam,
The Netherlands.
[2] Artificial Intelligence, University of Groningen,
Grote Kruisstr. 2/1, 9712 TS, Groningen,
The Netherlands.

Foreword

During my study it became clear that I have more interest in courses which one 'makes' things, rather than 'prove' them. The so-called practical courses are, to me, more enjoyable, for example programming and robotics courses. This has been the reason that, when the time came to search for a research project, I was looking for a project with the same qualities.
In the last years of study of AI I have had a number of courses concerning 'multi-agent' systems. These courses, given by Dr. R. Verbrugge, are mainly 'logical theorem' styled courses. During the lectures some interesting examples were named, such as multi agent systems used for the removal of mines. One simple robot can easily be replaced if it runs on a mine, as opposed to the classical single agent. The solo agent is highly capable, but also very expensive. The simple agents are disposable.
I have always been very much interested in collectives, such as ants in nature and the Borg in the science-fiction series StarTrek. How is it possible for all those individuals to work together as one? We humans cling to the idea of a self, and this is probably why I'm so fascinated by the idea of 'self sacrifice' for the good of the whole, shown by individuals in collectives.
So, now I found myself in R. Verbrugge's office, asking her if she knew places where work is done on simulation concerning multi-agent systems or preferably multi-agent robot systems, the real thing. She made it clear that there are two hot spots in Europe. The first location is in Brussels at the VUB and the second location in Amsterdam at the VU. After e-mailing both universities and getting replies from both, I decided to take a look in Amsterdam. There I met with Dr. M. Schut and PhD. researcher F. Wan. They had a project available waiting for someone to grab and use: The macroscopic behavioral pattern of Emperor Penguins. F. Wan showed me the simulation world that could be used. Wan had made simulations, and showed them to me. I was in heaven. This is exactly what I wanted to do. Making multi-agent, artificial life styled simulations.
The research offered falls within the category of 'self organising' systems: to create a whole with certain characteristics which the individuals do not have. This field of research is rather new, and the topic of 'emerging' behaviors is vague. The unexplored nature of the research excited me. An essential part of this research is brainstorming on what the causes can be of the overall pattern, how does it all work together to create the desired behavior. They offered, I gladly took it.

Before pressing onward I want to say thanks to a number of persons. First and foremost R. Verbrugge. She has been my guide during this endeavor. She counseled me, structured my project, gave inspiration, gave mental and emotional support, and she did not abandon me when I behaved chaotically or did not make deadlines. (She is also responsible for the grade I get.) No seriously, thank you.
During my time in Amsterdam at the VU, M. Schut has been a great supporter and help. He provided me with literature, read and corrected my work, gave

mental support, and much more. Yes, thank you too.
F. Wan is an expert on the used simulation world and possible ways of implementing agents in that world. He has helped me to get started. Wan provided me with inspiring articles on self organisation, very useful. Wan looked at my first simulations and gave inspiring recommendation for further work.

There have been an additional number of person whom I must, and want to thank also. M. Sc. G. Kootstra has been an inspirator for possible implementations to realize the desired penguin behavior. This must not be underestimated. During the implementation of the 'particle' model I have had contact with two experts in specific fields. I needed their expertise and they offered it quickly and were very friendly and nice to me. I needed information for modeling of wind, and R. Verbrugge got me into contact with Dr. L. Weber[1]. Unfortunately she could not help me directly, however, she has made it clear where I had to look and could look for the information I required. This is how I got into contact with Dr. G. Bussel[2]. We have had a number of e-mail discussions concerning the possible implementations of models for usage in my simulation world. At a certain point in time we even discussed the option to construct real-life-scaled penguins, for testing in the wind tunnel at their lab. In the end I had to abandon the 'particle' approach and consequently did not require the offered expertise any more. However, the friendly nature and the offered expertise of these two people I have appreciated very much.

Additionally there are a number of person who are not directly related to my research, but have proven invaluable also. Again I must name G. Kootstra, inspirator and friend. Concerning our used operating system on which Swarm is run, I give thanks to R. Zwaagstra at the RuG and G. Huisman at the VU. And of course G. Kloosterman. He is a UNIX wizard.
In Amsterdam I owe thanks to Karin Rijnders. She has been very nice to me in a, to me, unfamiliar and at times unpleasant surrounding. In Groningen I owe thanks to many, but especially Peter 'Back smash' Duifhuis and Arjan Stuiver, both for creating a familiar and pleasant working environment.
And of course, there are numerous others whom I could have named here. But then again, let's get on with the research.

[1] Dr. S. L. Weber,
Royal Netherlands Meteorological Institute (KNMI),
P.O. Box 201, 3730 AE De Bilt,
The Netherlands.

[2] Dr. Gerard J.W. van Bussel,
Section Wind Energy,
Faculty CiTG, TU Delft,
Stevinweg 1, 2628CN Delft,
The Netherlands.

Contents

Abstract

This thesis describes the modeling, implementation and simulation of the global rotational pattern of groups of Aptenodytes forsteri, commonly known as Emperor penguins. The global group pattern is modeled and implemented as being an 'emergent' phenomenon, resulting from simplistic penguin agents interacting in an environment. For the implementation of the agents a behavior based approach is used. The implementation and simulation of the models is done with the 'Swarm' software package, developed at the Santa Fe institute. The Swarm simulation system is a toolkit for building multi-agent simulations. The simulations are constructed in the Java programming language.

1 Introduction

The section now following will give an introduction into, and background knowledge to the field of Artificial Intelligence. First a short description of the *field of AI* will be given. The next subject is the appearance of *swarm intelligence* in the field. And finally, a more detailed discussion of the topic of *emergence* will follow.

1.1 Conventional AI

The field of artificial intelligence (AI) is relatively new. This is not surprising, since the electronic computer has only been around since the early 1940's. To give an indication of major paradigms in the field, a brief, not complete overview is given.
AI attempts to create intelligence. A system which is known to be capable of intelligent behavior, is the human. Then, to capture human like behavior, is to capture intelligence. This has been the general early view.
Early AI was mainly very successful in the field of games. Smart evaluation algorithms were created, such as the famous 'mini-max' algorithm. This algorithm is being used for evaluation of possible actions in a restricted domain, for example chess moves. And over the years, the successes of this approach increased. But the increase in success was not due to more human-like behavior. Instead, the brute computing force of computers had increased, so the algorithm could calculate more moves in advance.
Researchers realized that the apparent success of AI was actually not due to making human-like intelligent behavior. Master chess players see patterns on boards, the relative positions of chess pieces. The chess masters determine intelligent moves on a totally different basis than a mini-max algorithm. The limits of this approach are obvious, when there is no good evaluation function, like in the game "Go". As a consequence, no good AI "intelligence" has been designed yet.
Another mayor influence in the history of AI was due to McCarthy, who invented the influential 'Lisp' programming language [18]. The new grand idea of AI became *'good representation is the key'*. A huge success of this new approach was demonstrated by Shakey [20], a mobile robot. Shakey had a complete model of its world programmed into it. Shakey was able to navigate around obstacles, and to move through rooms.
The main view in AI became more and more the view of a central symbolic information processor. But AI realizes it is off track. There is a fundamental flaw, as Brooks states clearly:

> It relies on the assumption that a complete world model could be built internally and then manipulated. ... all relied on very simple worlds, and controlled situations. (From [6].)

Another big chunk of classical AI has been the parallel approach. Computers are fast serial computation machines. In nature, massive parallel slow computations are used. In AI neural networks tried to incorporate this. The neural network approach flourished in the late fifties and sixties. Later, when back propagation was invented by Rummelhart and McClelland [25], it flourished once more. But artificial networks learn slowly, and the learning rate is tuned by hand, unlike its natural counterparts.
Classical AI has had its share of successes. The success has often been *not* because of natural intelligence implementation, but due to fast and clever implementations.
Researchers have asked then, what is intelligence? The best know definition is the one from Alan Turing [29]. Roughly this definition states that, when an observer cannot make a distinction between the behavior of a computer and the behavior of for example a human or other animal, then one must conclude that that computer has the same level of intelligence as the animal in question.
A new widely advocated approach in the field of AI is the *behavior based* approach [7, 5]. A famous and successful example of this approach is the 'stabilizer disturber' architecture of Luc Steels [28]. Earlier AI tried to construct human-like intelligent behavior. More recently researchers are inspired by the behavioral based successes, and the intelligence these structures show. A new definition of intelligence is surfacing.

- "Intelligence is determined by the dynamics of interaction with the world." Under different circumstances different behaviors would be considered intelligent.
- "Intelligence is in the eye of the observer." When observing a system behaving in a certain way, we, the observers, determine if it is intelligent. The chess master thought 'deep blue', the computer chess player, behaved very clever and really thought things through. Deep Blue is a very fast computer, using an optimized mini-max algorithm.

For a more complete discussion the reader is referred to Brooks [6].

1.2 Self organisation and AI

There are many phenomena in the real world which have a structure. Examples of these phenomena are the whirlpools seen in the atmosphere, the star-like shapes of snowflakes under a microscope, the appearance of black and white stripes on zebras. At first sight this structure is not apparent in the parts which make up the structure. These characteristic structures appear when large numbers of molecules interact with each other: Self Organisation. Characteristic for self organisation is the appearance of structure, without a central controller. The structure is seen when the individual parts are put together. When looking at weather photos there are obvious high pressure and low pressure areas, structure. These structures appear when numerous molecules interact with each other. Galaxies always have a macroscopic disk-like structure, constructed

through the interactions of matter.
Self organisation has been extensively studied, for example simple cellular automata [11] or more appealing army ants [2], and is so interesting to AI because it has some desirable characteristics.
The Self Organising System (SOS) shows *adaptation*. In the example of ants this becomes apparent when a route is blocked. The ants quickly find the new shortest route [17]. A related issue is *robustness*. Reasonable numbers of new agents can be added or agents can be removed without compromising the total system. A related characteristic is due to its distributed nature. The system is more *reliable* than conventional complex agents. Single parts may break down without impairing the overall system.
These complex systems are desirable because of the *simplicity* of their individual parts. The desired characteristics (such as intelligent behavior) emerge from the interaction of the parts, without explicit supervision, or a central control system. Knowledge is distributed and becomes apparent in the interaction between agents and the environment [3].
All these traits are desired in AI, but conventional agents have trouble implementing these properties, because of the individualistic character of design, because one agent performs all tasks. Small conventional multi-agent systems have had more success implementing the desired traits [14].
With the study of groups of simple agents researchers hope to increase our knowledge of how desired emergent phenomena can arise, so that "the whole is more than the sum of the parts"[16].
Various researchers have studied and modeled SOS's. Models and simulations have been created describing traffic flow, humans in panic situations [12], schools of fish [30], flocking of birds [24], ant colonies, predators versus prey [26], and more. Through the modeling and simulation of specific topics our understanding of that specific phenomenon is increased. But more importantly it increases our understanding of the complex dynamics of simple parts which produce collective behaviors or properties.

1.2.1 Swarm intelligence

In the field of Artificial Intelligence (AI) there is growing attention for so called Self Organising Systems. Swarm intelligence is a specialization in the field of SOS. These are systems of numerous 'dumb' individual agents (heterogeneous or homogeneous groups [1]) that, by some kind of interaction, show emerging properties. The inspiration for design comes from natural swarms, such as bees, termites and ants.
Take for example a colony of ants (more examples in [4]). These individual agents (ants) are all the same, and simplistic in design. A solo ant can not survive. It will run around until it is exhausted and dies. However, as a colony they show intelligent properties. A colony of ants can find food sources, short routes, attack and defend: *Swarm intelligence*.
The ants are collectively capable of finding the shortest route to a food source, and cooperate in returning food to a central location [4]. An individual ant does

not know what the shortest route is. It also cannot return big prey on its own. But the colony as a whole does have these properties, which *emerge* from the interaction between the simple agents and the environment. These emergent properties are interesting because they are not explicitly programmed into the individual parts. These higher-order properties become apparent through the interaction of the agents with each other and the environment.
The term emergence requires additional explanation, and in the context of this particular research project the emergent behavior is the macroscopic behavior. So, the collective possesses a property that no individual part of the collective possesses. Still the collective's emergent behavior can be *understood* from the nature and behavior of its parts *plus* the knowledge of how these parts interact with each other and the world [10].

1.3 Emergence

In the previous section we have discussed 'swarm intelligence', and how properties emerge from simplistic parts interacting in a certain situation. But what do we mean when we say some property is emergent? For example in the described ant colony example, an individual ant cannot do much. When in isolation, it wanders around until it is exhausted and dies. However, the colony as a whole can organize a nest, attack and defend. Another example is that some researchers believe that human behavior, the feeling of identity or conscience is an emergent property of our neurons interaction. It somehow comes forth out of the interactions between the parts. The ultimate goal of AI is to create an artificial living being. With this emergent view of properties we consider key to us humans, this is theoretically possible. In a sense what is meant with emergent is that a property is systemic. No single part possesses a certain property, but the system as a whole does possess it. What these systems have in commen is their non-linearity. The functionallity of the constituant parts is not directly related to the functionallity of the whole. It *is* the non-linearity of these systems that decrees that the whole may exceed the sum of the parts.
But why is it necessary to describe properties as emergent? This is opposed to the traditional reductionist view of making the parts smaller and smaller until all is known. In [10] Damper gives an, as he said, 'arguable' example of an emergent property. When considering locomotion in animals one cannot say that this is a property of individual neurons, or muscles or bones. However, locomotion can be understood by the way that these separate parts work together. In other words, a satisfying explanation of walking relies on getting the level of abstraction right, and our surprise over the system's behavior evaporates.
As Steels [27] pointed out 'Emergent functionality means that a function is not achieved directly by a component or a hierarchical system of components, but indirectly by the interaction of more primitive components among themselves *and with the world.*' And it is exactly this difference, the interaction with the world, that is the distinction between chemistry and physics, and biology and chemistry, according to Damper [10].

Brooks [8] asked himself what the key feature to life might be. What is the key feature? One can imagine that it is just another phenomenon waiting for a correct discovery. A century ago there were causal relations to be seen, but it could not be explained. Then x-rays were discovered. A discovery of this kind might occur with respect to 'the stuff of life'. But of course, this is all highly speculative.
Before closing this section I would like to quote Brooks. He makes a nice statement about 'thinking in living systems' and our ability to reconstruct them artificially.

> My feeling is that thought and consciousness are epiphenomena of the process of being in the world. As the complexity of the world increases, and the complexity of processing to deal with that world rises, we will see the same evidence of thought and consciousness in our systems as we see in people other than ourselves now. Thought and consciousness will not need to be programmed in. They will emerge. (From [6])

2 Goal definition

The topic of this research project is the modeling and the implementation of swarm behavior. A group of simple agents interact with an environment, and show macroscopic behavior, which the individual parts do not possess. The main goal of this study is to gain insight into 'emergence' of such macroscopic behaviors. Thus, it will become clear how simplistic parts make a whole with desired complex characteristics.
To accomplish this goal, a particular swarm behavior will be studied, modeled and implemented, i.e, the macroscopic huddling behavior of the emperor penguins. By studying the natural penguins collective behavior and individual behavior we will gain 'inspiration' for our implementation of the simulated penguin agents. We will try to reproduce the natural collective behavior in our simulation.

2.1 Emergence and penguins

The emperor penguins live in the arctic, where extreme cold conditions can occur, in particular during storms. The penguins have adapted to their harsh environment by huddling together. But not only do the penguins huddle together, they also rotate the positions of penguins, so every penguin will stand for some time at the cold rim of the group. Apparently, every penguin behaves according to its own egoistic motives to minimize its own exposure to the cold. It is unlikely that an individual penguin has explicit knowledge of a center or a rim of the colony, but still, every penguin moves to and from it.
Our hypothesis is that the collective dynamics of such a group of penguins, apparent in a typical collective movement pattern, is emergent. See figure 1 on page 14 for the collective movement pattern. It is unlikely that penguins are always aware of their position relative to the group as a whole, especially during blizzards when vision is minimal. Therefore we hypothesize that this macroscopic behavior is emergent. We will attempt to validate this hypothesis by implementing and simulating artificial agents exhibiting the same macroscopic behavior, based on simple rules without knowledge of the collectives movement pattern. The pattern will emerge through the interaction of penguins with each other and penguins with the environment.
The study, modeling and implementation of the dynamic macroscopic pattern can be considered as a case study to accomplish a higher goal, namely, gaining knowledge into emergence of macroscopic patterns.

2.2 Model goals

In the developmental process of the project, models are constructed. Concerning these models it must be stressed that the goal of this research project is not to implement realistic emperor penguin behavior or environment. The goal is to implement the macroscopic group movement behavior of these penguins. This does not mean that this project will not utilize data about penguin behavior,

environment and wind flow modeling. In fact, this data is useful in determining what the influential parameters for modeling will be.

2.2.1 Models

For the simulation of the macroscopic behavior, a minimal amount of models is needed. First, a model of individual penguin agent behavior needs to be described. Second, a model of the harsh arctic winds is described, which is an important part of the penguin environment. The extreme cold winds appear to be the initiating force for the emergence of the huddling behavior of emperor penguins. Third, a model will be necessary describing how warmth is distributed over the environment. This can be done explicitly, by creating a warmth distribution, or implicitly, by creating behaviors which are only active in certain situations.

2.2.2 Method of design

The penguin model and the wind model are made as autonomous as possible. From these two models and a current world situation a certain "warmth" distribution can be derived, implicitly or explicitly. The distribution of warmth is assumed to be the main motivation for the emergence of the desired pattern.
The design process is iterative, mainly because of the nature of emergent phenomena. By definition, no clear predefined path of successful modeling can be applied.
First, a simple implementation of all models will be made. As a result of this simulation, new models are formulated from the insights gained. These new models constitute the basis for the final implementation. 11 5

2.3 Implementation

The final implementation would be considered a success, if it exhibits the macroscopic behavioral pattern exhibited by the emperor penguins *without* being explicitly programmed to exhibit this behavior. Furthermore, the implementation will aim for flexibility and adaptability by modular design. The division in modules is a direct reflection of the previously constructed models.

2.3.1 Software SWARM

For the implementation of the models an already existing software package is used. This software is called 'Swarm'[3]. The user can create swarms using the

[3]Swarm is a multi-agent software platform for the simulation of complex adaptive systems. In the Swarm system the basic unit of simulation is the swarm, a collection of agents executing a schedule of actions. Swarm supports hierarchical modeling approaches whereby agents can be composed of swarms of other agents in nested structures. Swarm provides object oriented libraries of reusable components for building models and analyzing, displaying, and controlling experiments on those models. Swarm is currently available as a beta version in full, free source code form. It requires the GNU C Compiler, Unix, and X Windows. More information about Swarm can be obtained from the web pages:

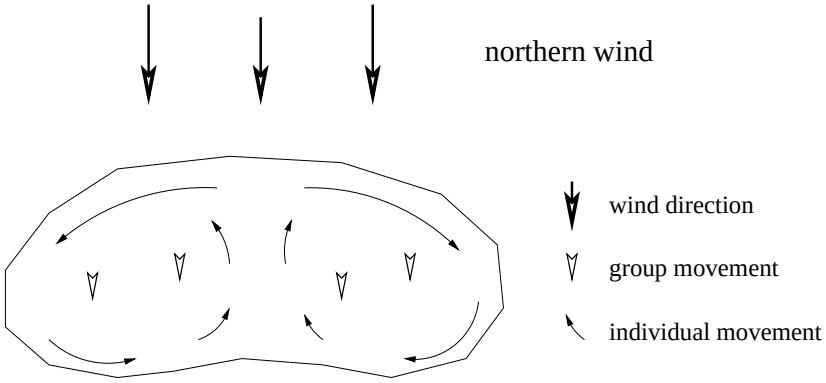

Figure 1: A simplified picture of the macroscopic movement pattern of a colony of emperor penguins. The top bold arrows show the wind direction.
The general shape of a colony in extreme cold conditions is as depicted. The arrows in the colony of penguins indicate movement of the group as a whole and movement for individual penguins. There is explicit individual penguin movement at the top of the colony, where the penguins stand in full wind and start to peel off to the sides, trying to reach the lee side of the cluster. The individual movement at the center of the colony is relative to the group, as the group as a whole slowly moves down wind.
It is not clear from the data whether the penguins in the center of the cluster actually move toward the wind direction, or whether these penguins only move relative to the others. It is likely that the agents at the core are standing still, although there is no explicit confirmation of this from the data.

programming language Java. The programmers of Swarm give the following description:

> Swarm is a software package for multi-agent simulation of complex systems, originally developed at the Santa Fe Institute. Swarm is intended to be a useful tool for researchers in a variety of disciplines. The basic architecture of Swarm is the simulation of collections of concurrently interacting agents: with this architecture, we can implement a large variety of agent based models [19].

Thus, Swarm is an already constructed useful visualization tool. The availability of this software should reduce development and implementation time.

---

http://www.santafe.edu/projects/swarm/swarmdoc/swarmdoc.html
http://www.swarm.org/

3 The data

To be able to implement natural macroscopic behavior, in this case, of the emperor penguins (Aptenodytes forsteri) huddle, one has to understand how the macroscopic behavior comes about in nature. This section will describe what is known about the penguin behavior and the penguin environment. Information is taken mainly from a field study report undertaken by Roger Kirkwood [15].

3.1 The Emperor Penguin

Emperor penguins live in the arctic all year round. These birds breed in colonies which are mainly located on the antarctic fast ice were the ice stays stable from winter to early summer. Thirty colony breeding sites are known, and the estimated total population is 200,000 breeding pairs.
Emperor penguins are the largest sea birds, standing 115 centimeters tall and weighing up to 40 kilograms. Remarkable about the penguins is their breeding behavior. The female lays an egg, and passes it on to the male. The male balances the egg on its feet, pressed against a warm blood patch and protected from the cold by a skin fold. The egg will stay there for the total incubation time. The females leave to forage, and the males stay. For a total time of four, up to five, months the males do not eat.
The emperor penguins are well adapted to the cold environment. Their dense plumage provides good insulation, better than any other penguin species. But in the harsh arctic conditions, which this species has made its habitat, that will not suffice. Emperor penguins show adaptation by exhibiting unique huddling behavior. The penguins huddle together to share body warmth and minimize energy expenditure. Through the huddling behavior the penguins reduce their energy loss to approximately half the energy loss of penguins standing in isolation [22, 21].

3.1.1 Arctic environment

The arctic environment is extreme. The obvious cause is the position on the earth. This results in low light intensity levels, which results in a cold climate. The mean wind speed in the arctic region in which the penguins live is variable, because the regions of the colonies are variable. Generally speaking, winds come from the south. Data collected at Mawson Station during 1993 (by Bureau of Meteorology) indicate a mean speed of 36 (km/h) and a maximum daily mean of 125 (km/h). At the Auster penguin colony, the maximum mean daily speed was 108 (km/h) during the same year. (See also figure 2 and figure 3.)
The temperatures in the arctic are cold. At Mawson Station the data for 1993 indicate an average temperature of 3.9°C in January to -22.1°C in July and August. Sometimes temperatures reach well below -40°C.
These numbers can be taken as indications of how harsh the conditions in the arctic can be.

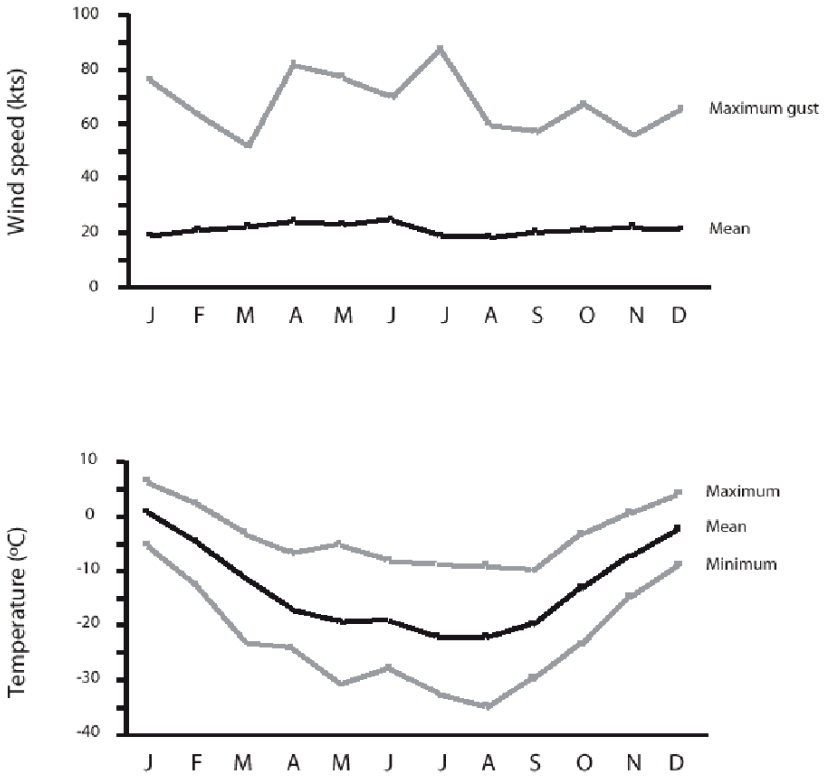

Figure 2: Wind speed and ambient temperature at Mawson Station during 1993 (Data collected by Bureau of Meteorology).
The wind speeds are shown in meters per second. The upper line shows the maximum gust reported, which could reach a mean daily speed of $108(km/h)$ $(30(m/s))$. The lower line shows the mean monthly wind speed.
The upper temperature line depicts the maximum mean daily temperature. The middle line depicts the mean monthly temperature. The lower line depicts the minimum mean daily temperature. All data are as presented in [15]

.

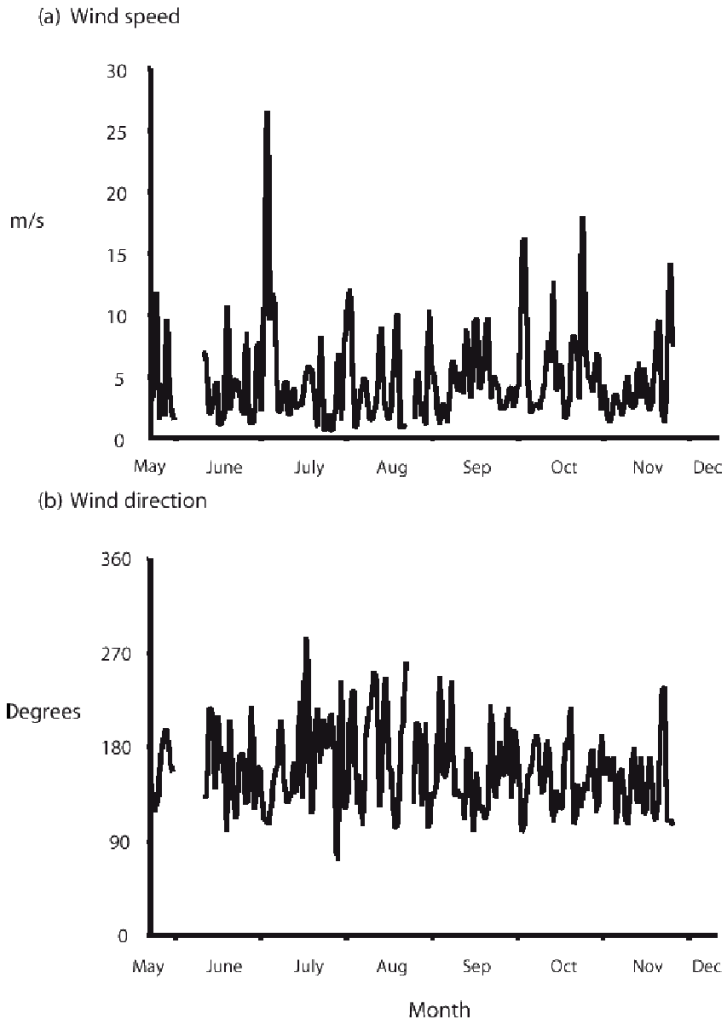

Figure 3: Mean daily wind speed (a) and direction (b) experienced by emperor penguins at Auster Colony during 1993. Data collected by Bureau of Meteorology, as presented in [15]

3.1.2 Penguin adaptation, huddling

The emperor penguins live in extreme conditions. The penguins increase their chances of survival by unique behavioral adaptation. This adaptation is huddling, the penguins group together. The huddling behavior is especially useful during storms. In these conditions temperatures can drop to -40℃, with wind speeds of 200 (km/h).
When the penguins huddle the density of penguins increases. Measurements indicate a density of up to 10 $birds/m^2$. Temperatures in these groups can reach 23℃[4].
Through the huddling behavior, the penguins reduce their energy loss to approximately half the energy loss of penguins standing in isolation. This is very useful, since at the breeding colonies, these male penguins do not feed for about four months. There is also evidence of huddling behavior of female penguins when they are off to forage.
There has to be a minimum of penguins for possibility of the huddle to emerge. At colony breeding sites, these numbers range from 2.700 to 14.500 penguins. But there is also evidence of penguins huddling when out to forage, in which case the numbers are considerably less. About the off colony huddles during foraging, no clear data is available about the precise numbers of penguins involved in huddles. The minimal number of penguins needed to let the huddle emerge is not known. But it does seem profitable for only two individual penguins to huddle together, even if it would only provide minimal energy savings.
When penguins huddle, researchers have reported remarkable collective behavior. As the penguins huddle together, the penguins standing at the windward side get cold. These penguins peel off to the sides of the colony, seeking the lee side of the group. Now other penguins stand in full wind, and they too start to peel off the sides of the colony. The result is a continuous shift of penguins, every penguin taking its turn standing at the cold rim. Collectively, the group is slowly moving downwind (See figure 1 on page 14.)

3.2 Emergence and emperor penguins

Individual penguins group together to save energy and minimize exposure to the cold environment. When an individual penguin stands at the cold windward side of the colony, it loses warmth rapidly. The individual penguin will try to move around the group, peel off, to get to the lee side of the group. The penguin has less exposure to the wind there, and consequently loses less body heat.
The sum of all penguins behaving individually and on selfish motives is an *emergent* macroscopic behavior. The collective moves in two circles. The parts, meaning the penguins, of the collective move from the center of the group to the windward rim. Next, the parts move around the group, to the lee side. Finally,

[4]The temperature of 23 ℃, reported by Kirkwood in [15], was the upper limit of the sensors being used.

the parts move to the center again, completing the cycle. The behavior can be seen with a group of bicyclers in full wind. Every bicycler takes head position and then falls off to the sides. A difference is that the bicyclers moves up wind, and the penguin group moves down wind.
The described collective behavior is emergent, because no individual penguin has knowledge of a center of the group. It is likely that agents can not even see the center of the group, due to other penguins obstructing their view. In storms this is even more evident. No individual penguin has knowledge of distributing cold exposure over all penguins. This comes about through interaction of many penguins with each other, and of interaction between penguins and the environment.
The described macroscopic behavior is extracted from reports, provided by biologists. The biologists studying Emperor Penguins do not look explicitly for a macroscopic pattern in the penguin behavior. Due to this fact, the certainty with which the general behavioral pattern can be stated, is limited. But still, to our best current knowledge, the macroscopic pattern is as described above.

4 Modeling

It is attempted to construct a model which has, when implemented, the same *collective* or group behavior as the Emperor Penguins have (see figure 1). In such a case, it is a good idea to look closely at what clues nature can provide. A model is by definition an abstraction and simplification of the real world equivalent. We describe what in our view are the minimal requirements for a model in this particular case.
The behavior to be modeled is the macroscopic behavioral pattern exhibited by the Emperor Penguins. The literature provides clues to what the core influences could be. A characteristic behavior of the penguins is *clustering*. When the temperature in the environment drops, penguins group together. In these clusters the penguins stand closely together and share body warmth. A model will have to incorporate this clustering behavior.
A clustered circle of penguins sharing body warmth is not enough. The biologists observing the penguins describe:

> During periods of strong wind ... the colony ... progressed downwind, as birds at the windward edge of the huddle felt the cold and shuffled around the huddle's flanks to re-join the group at the leeward edge. (From [15].)

It seems reasonable to hypothesize that the influence of the wind is initiating individual movement. Penguins standing in full wind, will be motivated to move around the group, to the lee area of the penguin group. So, the second part of a model will have to incorporate how wind flow influences the penguin environment.
To summarize, the combination of clustering of agents, and the influence of wind on the agents, could be a sufficient basis for the collective rotational pattern to emerge.
The two stated microscopic considerations, clustering and wind influence, do not translate directly to the macroscopic pattern. The assumption is that the small scale individual behavior will be enough to make the large scale pattern emerge. Macro behavior will emerge through the interaction of wind with penguins, and the stimergetic[5] interactions between penguins.
In the process of modeling a first model was constructed. Even though it has not been the basis of the final implementation, the results of this model have had significant impact on our later models. This is why a short summary is given of this model in section 4.2. But first we will give an explanation of our

[5]Stimergetic communication is indirect communication through environment.
The famous example of stigmergetic communication is the pheromone traces of ants. Ants deposit pheromone on their paths, a sort of path marking. Ants choose their way based on the extent of pheromone on a particular trail. It has been shown that the pheromone deposition strategy can solve classic mathematical problems like for example the 'traveling salesman' problem [17].
In the our model this is 'heat distribution'. In a first model, discussed briefly in section 4.2, this was modeled explicitly. In the current model this is implicit, expressed through the behavioral activation functions.

tools used, and some practical considerations. Section 4.3 will state the latest model, the basis for our final implementation.

4.1 Tools and practical considerations

Creation of models has to be based on facts or assumptions. The problem with the very nature of the subject, *emergence* of a macroscopic pattern, is that it is vague and obscure. The tools that are used to design and implement the goal, are all applied to the microscopic or individual agent level.
For the modeling two main tools are used. The first one is the available literature on the emperor penguin. Literature is useful for specific parameter settings, and more importantly, gaining insight into the specifics of the phenomenon to be modeled.
The second tool is the study of already existing swarm implementations. These implementations show how the dynamics of simple parts can provide the emergent behavior in simulations or applications.
Additionally, during the modeling phase we have to keep in mind the implementational world. The implementational environment is already set to a large degree. This practical realization leads to extra constraints on possibilities for modeling. For a more complete discussion of these considerations, the reader is referred to section 5.

4.1.1 Literature and data

From the available literature one can acquire usable data. It must be stressed though, that the amount of usable literature is small. For every part to be modeled, literature is valuable as a basis on which to build. Certain parts of the project have the problem of *unavailability* of literature[6].
Other parts of the project do have literature available. Unfortunately, much of this literature is often too *complex* for direct usage. The goal is to implement a working simulation with many agents. The number of computations per element to be implemented in the simulation, needs to be kept to a minimum. Also, it is a fundamental hypothesis that complex behavior can arise from simple elements and environment.
In our search for usable and relevant literature for modeling, we encountered various disciplines of science. These scientific fields are not directly related to AI, such as biology, meteorology and physics. In these cases, complete usage of the literature is not possible, because of lack of expertise in those specific fields[7]. Still, these articles have enhanced insight into critical subtopics. Due to

[6]The literature concerning the Emperor Penguins is small. This is mainly due to the inhospitability of their natural habitat. As a consequence there have been few researchers studying Emperor Penguins.

[7]The search for relevant literature has led to exploration of scientific fields, unknown to the typical researcher in the field of AI. Example of some of the new topics are: wind flow dynamics and obstacles, models for description of flow of masses of particles, natural structuring in fluids, specific biology. The articles concerning these topics have been very useful in construction of models, though not directly. Often these articles have helped to, if you will, create "feeling"

this literature a better foundation is available on which decisions can be made. For the *modeling of huddling* behavior of penguins, information from field studies has been used, mainly from Kirkwood [15]. This study contains a wealth of data concerning environment temperature, temperature inside huddles, wind speed recordings and many more. This data can be used for general parameter settings.
Good *modeling of wind flow* is essential. But from analysis of articles, it became clear that exact precise modeling was impossible. The available literature is tremendous, and models are available. But these models are too complex, or specific to a specialized subfield.
The overall *macroscopic pattern* will not be implemented through direct modeling, but through modeling *microscopic characteristics* of agents and environment. For the creative process of the development of models, some additional papers have been studied. These papers give inspiration for possible underlying individual behavior, from which the macroscopic pattern emerges, for example, a paper on the topic of particle swarm [9], which describes the flow of particles through a space. Also experiments with self organisation in fluids have helped to shape the view of the overall pattern[8].

4.1.2 Swarm implementations

Another useful tool in the process of construction of models is the study of working swarm implementations. These implementations show how complex dynamics can emerge out of the simple parts and environment.
An example of such an implementation is the BOIDS [24] implementation. This well known example shows how flocking behavior of birds can be simulated on a computer, by the usage of three simple rules:

- separation, steer to avoid crowding local flock mates,
- alignment, steer towards the average heading of local flock mates,
- cohesion, steer to move toward the average position of local flock mates.

Remarkably, these three rules applied to agents, show the same flocking behavior as their natural counterparts.
Another example is the appliance of ant colony behavior to the "traveling salesman" problem[9]. Ants are able to exploit multiple food sources efficiently, through placement of pheromone which evaporates over time. When an ant reaches a food source, it returns to the colony nest. The shorter the route taken, the sooner the ant returns. On a short route more ants pass in the same amount of time to place pheromone and as a consequence more pheromone is deposited on that particular trail. This implicit knowledge is used by ants; the ants have a tendency to choose the trails with high pheromone. Simulated ants

for the specific subtopic.

[8] http://www.fluid.tue.nl/WDY/vort/ntvn/zelforg.html

[9] The traveling salesman has a number of cities to visit. The salesman tries to visit every city only once, and tries to make his route of travel as short as possible.

have been shown to effectively use this trail marking technique to solve the traveling salesman problem [17].
It was known beforehand that the model would be implemented with the 'Swarm' [19] programming tool. To get an idea of what is possible to implement, how certain behaviors can be implemented, existing implementations have been studied. These implementations were made available through the swarm web site. The Swarm software gives great freedom for implementation of models, but also constraints. It would be wise to take these constraints into account when modeling. These constraints are discussed more elaborately in section 5.

4.2 First shot : heat particles

Emergent phenomena are by definition not defined into the individual parts. This realization has led to a pragmatic approach to the modeling problem. The construction of the first simple model has one main goal. This goal is to get insight into the dynamics of penguins, penguins huddling and wind. These components combined will constitute the overall macroscopic behavior.
This pragmatic approach is justifiable on the grounds that there is no exact knowledge of what specific individual parts will realize the macroscopic pattern. This first model envisions the agents as heat particles. These particles flow towards the higher temperature grid compartments[10] in their world. The particles will try to climb to higher temperatures. The model is inspired by particle swarm and explicit 'heat distribution'. The article by Clerc and Kennedy [9], which describes the flow of particles through a space, has been useful for gaining insight into the movement of penguin particles moving though a temperature space.
As in the ants and their pheromone, it is attempted to create stigmergetic communication through 'heat distribution' or heat placement in the world. For general parameter setting, additional data can be used. Data from the weather station "Mawson Station"[11] is very useful for general environmental temperature settings. Mawson station is located near an emperor penguin colony. For individual agent parameter settings concerning the temperature, the field study from Kirkwood [15] is used.
This first model is constructed with the knowledge that it *is* a first simple model. The goal is to create greater understanding into what the possible causes are for the macroscopic pattern, and secondly, to see if this first 'heat distribution' model is a sufficient analogy.

4.2.1 Core characteristics

This model models agents as being heat seekers, in other words, actively searching neighboring compartments for a higher temperature. The temperature of the compartments is influenced by body heat of agents and wind.

[10] The world in which the agents move is a two dimensional grid, like a chess board. A world compartment can contain an agent or can be empty.
[11] www.antdiv.gov.au/stations/mawson/

The temperature influence of agents and wind can be seen as deposits of "temperature pheromone" trails. Based on these temperature trails, agents decide where to move to. The analogy fails with respect to pheromone evaporation. If an agent moves, the agent's temperature influence moves with it. No dissipating temperature is left behind. The resultant temperature landscape is in turn the basis on which the individual agents determine their behavior.
The first model models *huddling* by implementing individual penguin agents as heaters. Whenever there is a penguin at a certain point on the grid, that compartment and every one of its eight surrounding compartments, will be given a $+k$℃ modification. The compartment influenced by body heat can be defined in several ways. Figure 5 shows two *huddle matrices*. The combination of the modeling of an agent as actively searching neighboring compartments for a higher temperature and modeling of agents as being 'heaters', could suffice for huddling to emerge.
In the penguin environment there are very few obstacles influencing wind flow. The main obstacles behind which the penguins find shelter from wind are other penguins, which is the second modeling consideration. This consideration has led to the definition of a *wind lee matrix*, a lee area modeled as a 'tail' behind each penguin. Examples of lee areas are given in figure 6.
From the observational studies, one can conclude that penguins begin to move when sufficiently cold. In the observations, this is referred to as 'peeling off' of penguins from the wind side of the group. The model uses the following heuristic: when an agent is cold, it is highly motivated to move. When an agent is warm and comfortable, surrounded by other body heat sharing agents, its motivation to move is low.
The decision where an agent moves to is based on the temperatures of the neighboring compartments. The warmer a compartment, the more desirable that compartment is.
In short the goal of the definition of our heat producing matrices is to create a heat landscape in the agents' world. This 'heatscape' has its higher temperature average top slightly displaced downwind in comparison with the agents positions. It is expected that the penguin model is the minimum required for the emergence of clustering of penguin agents, with a stable non moving center, and a turbulent outer rim. Together with the wind influence, displacing the heatscape downwind, the agents will peel off.
Due to this displaced heatscape, the simulation will not reach a steady state. The penguins at the rim of the cluster, it is predicted, will fall down the sides to the lee aria.

4.2.2 Emerging results

During implementation and simulation or the first model, results were obtained.

- Huddling. The agents cluster together to form large groups. Agents at the center of a cluster stand still more often than the agents at the rim. The percentage of agents forming a stable core, the number of non-moving

individuals in a cluster, can be increased by use of the motivational function.

- Group movement downwind. The group moves collectively in a direction directly related to the definition of the wind matrix.
- Peeling off behavior. Individual agents sporadically show peeling off behavior. This is shown as individuals 'falling' down the sides of the cluster, to the lee area.
- No macroscopic circling movement. The individual agents do not move as depicted by the arrows in figure 1. The simulation becomes trapped in a steady state.

These results are remarkable for such a simple model. There were however many reoccurring flaws. The most remarkable is the minimal cohesion of the clusters. Too quickly and easily individual agents break free from the groups.
Another flaw is a direct result of the definition of our matrices. This makes the compartments on the lee side of an agent a higher temperature, to result in a group walking down wind. However, isolated individuals would also walk solely down wind, towards their own 'lee side'. The sum of the agents own wind matrix and huddle matrix would always make the compartment down wind of the agents current position preferable in temperature. The solo agents walking towards their 'own' lee side is quite unrealistic[12].
A rather large flaw is the observation that individual agents move chaotically. At one time step an agent moves south, while at the next it can move north again. This individual behavior is unrealistic and there seems no solution to this problem in the current approach. This can be remedied, but this goes against the used analogy of an agent as a 'heat seeking particle'.
The macroscopic pattern we are looking for does show up sporadically, shown by individuals falling down the sides. But often the simulation will become trapped in a steady state of a non moving group.

An agent tries to move to high temperature regions. The agent probes all eight compartments it can move to for that compartment's temperature. Through that temperature it can determine, implicitly, whether other agents are near. This means that the size of e.g. the body-heat matrix corresponds directly to the view an agent has. To remedy the low cohesion of clusters one can enlarge the body heat and wind matrices. This method *greatly* increases cohesion. However it is unlikely that natural penguins feel each others' body heat at a couple of compartments, in other words, body distances away. The same comments hold for wind. A bigger wind matrix increases results. And again the same criticism holds because it is implausible that lee effects are noticeable more than two penguin body distances away.

[12] This behavior can be seen in cartoons, when a donkey constantly walks towards a carrot hanging in front of it. The donkey never gets the carrot.

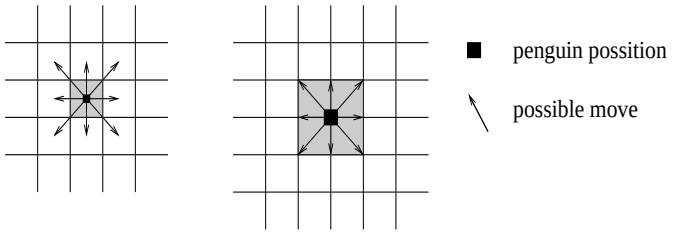

Figure 4: An agent can move on the world grid to every neighboring compartment. The number of possible moves is eight in both depicted cases. The left part of the figure shows the possible moves for an agent modeled as occupying one 'world-grid' compartment. The right part of the figure shows the possible moves for an agent modeled as occupying four grid compartment. In this second case the agent is possitioned at the center of its four 'body parts'.
Every simulation time step an agent determines its new position. An agent walking from the bottom left corner to the top right corner, or, an agent walking from the left side to right side, would make no difference in simulation time. However, in reality the diagonal distance is much longer. This gives a distortion of the world. A more realistic depiction of the situation would be pull the corners of the grid towards the center, making the grid into a circle, and making distance equally related to simulation time in all possible directions. (Artificial life models frequently use four possible moves: north, south, east and west.)

In summary, the most striking observation during simulation is that the results can be increased significantly by enlarging relevant matrices, implicitly increasing view of agents. It also has been made clear that there is no real world justification for these enlarged matrices in our current model. A second observation is the chaotic unrealistic behavior of individual agents. These observations were the main reason for a new model and a new approach.

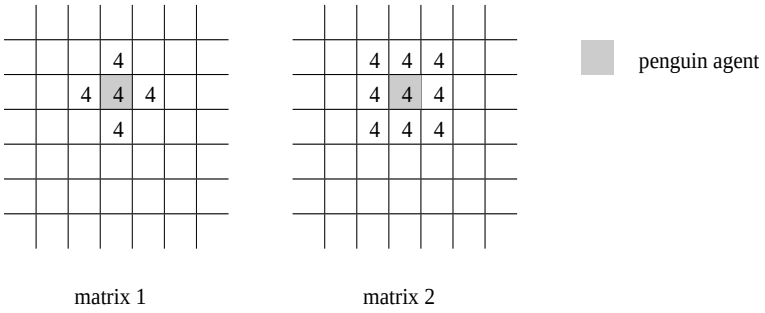

Figure 5: Two different models of huddle heat. The penguin is situated at the center compartment, and radiates a temperature modifier +k to its neighboring compartments. Matrix 1 radiates only to its horizontal and vertical neighbors. In matrix 2 a penguin gives body heat to all neighboring compartments. Note that the penguin also generates heat in its own compartment. If this was not the case, a single penguin standing alone, would likely be motivated to walk towards a neighboring compartment, following its own body heat.

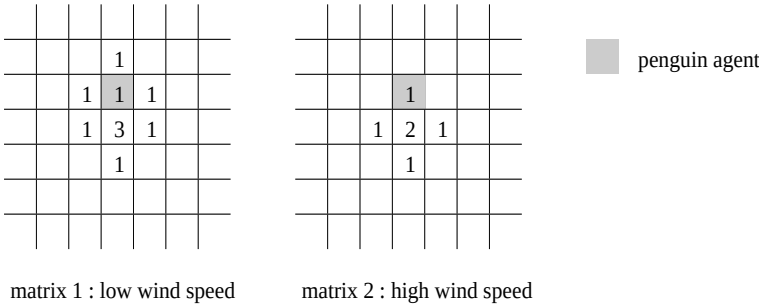

Figure 6: Two different models for wind flow around an object (penguin). The object is situated in the center compartment, wind is coming from the north. Wind influence is set to an absolute value (e.g, -30℃). The maximum lee is the maximum temperature cancelable (e.g, 25℃). If an agent is enclosed by other agents it receives maximum shelter (resulting in $-30 + 25$ = -5℃ wind temperature influence).
The percentage lee in a particular compartment is the value in that compartment divided by the sum of the lee factors in the matrix (model 1: $\sum(x, y) = 10$; model 2: $\sum(x, y) = 6$).

4.3 Model 2: behavior based

The results of the first implementation and the analysis of its results (see section 6.1), have led to the conclusion that a different approach is necessary. The new approach to tackle the problem is a more behavior based. This means that explicit behavioral agent characteristics will be set up, to generate the emergent phenomena.
In the 'heatscape' approach agents could see each other implicitly through the deposits of heat in the world. It was observed that increase in 'vision' by enlargement of the heat matrices, increased cohesion of clusters significantly. There was no natural justification for this matrices enlargement in the then used analogy, for example by enlarging 'body heat' influence. However, vision range does seems directly related to possibility of emergence of a cohesive cluster. Now agents have an explicit vision range and consequently, increased cohesion of groups is more likely to be possible to create. More important is that the relatively large vision range, as compared to the implicit short vision range in the first model, is now not unrealistic and possible.
By using a behavior based approach we attempt to remove chaotic individual behavior. Erratic movement is removed by introducing a heading for agents. This makes it possible to state where an agent can move to, relative to its current heading.
In hindsight the particle approach of the first model departed from the implementational successes already present in the swarm intelligence literature. When considering, for example, the BOIDS [24] implementation, the emphasis has been on the behaviors of the agents, and not so much on modeling of environment as has been done in our first approach. The new model will attempt to correct this discrepancy.
Emphasis in the new model will be on the penguin agent. First we will discuss the penguin agent and its behaviors, and secondly we will describe the wind model.

4.3.1 An agent

The model for the penguin envisions agents as having a set number of core behaviors. These behaviors are grouping, repulsion, alignment and peeling off. But before we will go into the why and how of these behaviors we have to discuss what an agent actually is.
The previous model showed chaotic behavior, and we will try to remedy this by introducing an agent *heading*. Basically an agent now has an orientation into a certain direction, which makes it possible to state restrictions on possible moves. Now a penguin will not move backwards, although from the report of Kirkwood [15] this can not be concluded. The exact parameters concerning this can only be determined experimentally. On common-sensical ground it is decided that the area possible to move to is approximately 90 degrees. In other words, previously an agent could walk to all eight neighboring compartments, now it can move to only two (relative to its heading). An agent can rotate

a maximum every simulation time step. We have set this to 45 degrees tops. Again, this is on common-sensical ground because of lack of specific data.
A second new characteristic of an agent is its bigger *size*. When agents occupy more space, more often they will obstruct each other's movement. We are interested into the influence of this on collective behavior. The second reason for creation of larger agents is the introduction of a 'repulsive' behavior in the agent's behavioral repertoire. The reason for introduction of the 'repulsive' behavior is discussed in section 6.1.2. Introduction of a repulsive area requires the definition of range in which an agent is repulsed. The model world is already set. This is a two dimensional grid of compartments. The minimal size of a repulsive area around an agent is one compartment wide. When an agent is the size of one compartment, this results in unrealistically large repulsive areas. Minimally size is one body length. The only possible option to make the repulsive area smaller is to make the agents larger, and therefore making repulsive area relatively smaller. An agent is now four compartments large.
Agents have a collection of behaviors. These behaviors are based on the local situation. What the local situation is, is defined through a *vision range*. This range states how far, or how many world compartments the agent can look around. An agent has 360 degrees view, which is normal for birds. The distance an agent can look away is kept minimal, not only for computational reasons. The other reason is that local influences are the core of existing successful swarm implementations, and is fundamental to self organisation processes. We model the vision range to be about $3\frac{1}{2}$ simulated-agent body lengths away. This is approximately a range of a meter.

4.3.2 Agent behaviors

The penguin agents all possess a set number of behaviors. The first three are analogous to the rules used in the BOIDS implementation. To this a forth behavior is added, namely, peeling off.

- Grouping:
  Move towards the high agent density area. Determine where the other agents are located, locally, and attempt to move towards these agents.
- Repulsion:
  Keep a minimal distance from other agents.
- Alignment:
  Look around, and turn your head in the same direction.
- Peel off:
  When there is no shelter from the wind, move (towards shelter).

The stated behaviors all give a resultant vector. The length of a vector reflects the importance of the specific behavior in the current agent's situation. These resultant vectors are added to a resultant vector. The resultant vector is the bases for adjustment to the agents current speed and heading. A schematic

depiction is given in figure 7.
The choice for creation of this behavioral inter influential approach is motivated by observations in a preliminary behavior based implementation. Only one specific behavior would be active at a certain simulation time step for a particular agent. This proved ineffective at certain point, for example, agents clustered together in groups, as desired, due to the sole activation of grouping at appropriate times. However, holes would emerge in these large groups of agents because in that situation no grouping, but alignment is the sole active behavior. The only way to remedy this surprising effect is to allow grouping to be active to some degree also. For a more complete discussion the reader is referred to section 6.1.2.

The first behavior an agent has is grouping. The average position of other agents (now referred to as 'other') in the vision range is calculated, and translated directly to a resultant vector. When $(\overline{x}, \overline{y})$ is far from the agent's position, one can conclude that there are few agents near. When $(\overline{x}, \overline{y})$ is close the agent is surrounded by others. When an agent is surrounded grouping is not that important any more. This is translated to the resulting grouping vector by an additional function.

The second behavior is repulsion. When agents move too close to each other they will be repulsed by them. It is intended to cancel out grouping influences and so prevent groups to become trapped in a steady state, giving agents room to move. An agent occupies four grid compartments, an agent is size four. The repulsive area has, due practical reasons, e.a, the simulation world is already set, a minimal size of one compartment. Relative to the current size of an agent of four compartments, this is large compared to the natural penguins. However, this is a trade-off, because considerably more computations are needed when agents increase in size. Body size and vision range are linked to each other. Vision range translates directly to the area an individual agent needs to check, which in turn translates to simulation and computation time.
The repulsive force an agent is experiencing is related to the closeness of others. The agent has to react to the closest other. This is why the repulsive vector is modeled as 'quadratically related to closeness'.

Alignment is the third behavior. A flaw of the first model was chaotic individual behavior of agents, not shown by the natural penguins. For masses of agents to be able to move, some cooperation is needed. To incorporate more coordinated movement, alignment is introduced. To be able to align with other agents, an agents needs to have an orientation, or heading. In our model it now has a heading.
An agent determines its heading based on the heading of others in its vision range. It is optional to relate the importance of an others heading to the distance it has from the agent.

Still this model can not give rise to the macroscopic behavior of natural pen-

guins. Therefore another behavior is explicitly introduced. When in full wind, when there is no shelter from other agents, try to move around the group [15]. A group needs to be present for this to be successful.
We attempt to model this by checking $(\overline{x}, \overline{y})$, already calculated by the grouping behavior. If there are many others present and $(\overline{x}, \overline{y})$ is relatively positioned down wind, then an agent can conclude it is standing at the top of a cluster in full wind. A second possibility is to check if there is an other standing in front of the agent, blocking the wind.
When an agent has determined it needs to peel off there are two possibilities. The first one is to make the agent turn away from the wind. This requires an agent to actually have its heading toward the wind. A second option is to make the agent 'go wander', in other words, let it move or rotate randomly. Hopefully it will peel off in combination with the other behaviors.

The individual behavioral vectors are added up to create a resultant vector which makes the agent move and rotate. It is expected that this will require a lot of fine-tuning time. A change in one behavior will change the balance with other behaviors, which have to be tuned again as well.
During implementation and simulation it might become necessary to incorporate speed into our model. We attempt to create a stable non moving core, like the natural penguin groups have. However, we hope the obstruction of each other will suffice to create a non moving core, without explicit velocity influence.
In the previously stated BOIDS implementation behaviors are prioritized. In that implementation a set amount of activation is used, from which the individual behaviors eat away activation until it is consumed. An example of the usefulness of activation was shown during obstacle avoidance. When a simulated bird was at risk of colliding with an obstacle it needed to react to that immediately. In such a situation alignment and grouping are not that important. We will use prioritized activation if this proves useful.

4.3.3 Wind model

The implementation of wind or wind flow in our model is a direction from which the wind blows. The behavior of natural penguins describes them as peeling off from the windward side. A direction from which the wind blows, with local situational awareness of an individual penguin, could be enough. In our current view this is all that is needed.

4.3.4 Expected macroscopic behavior

The expected emergent macroscopic behavior will be first a clustering of agents with their orientation towards the center of the group. It is expected that repulsion and grouping will cancel each other out. Due to this canceling out of each other alignment will get the upper hand. This results in a stable non moving cluster with locally aligned agents.
Peeling off behavior can create instability at the side in wind by 'randomizing'

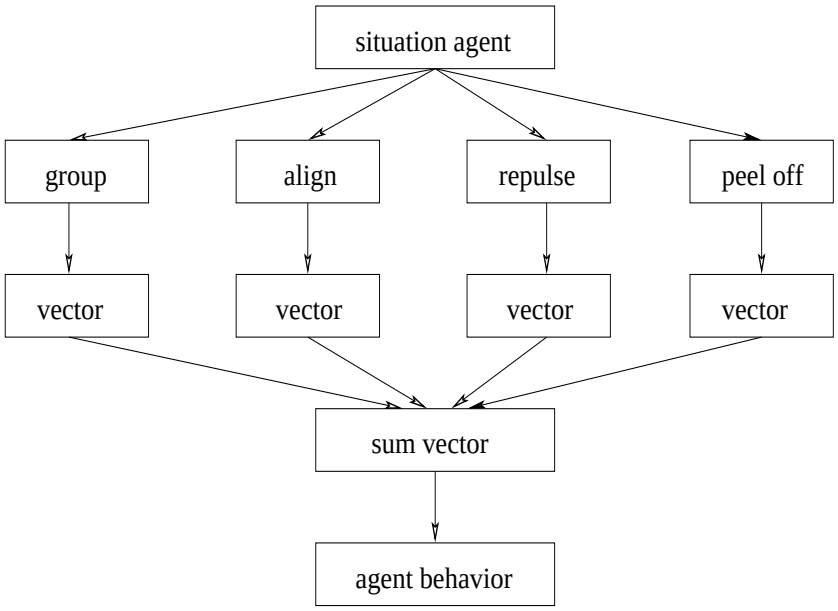

Figure 7: A schematic depiction of the behavior based model. The current local situation of an agent is the input for the individual behavioral 'modules'. Every behavioral module calculates a behavioral vector, representing the importance of that behavior in the agent's current situation. These four behavioral vectors are added to a resultant behavioral vector. The resultant vector influences the agent's exhibited behavior.

movement. The second option is to create explicitly movement perpendicular to the wind.
The second option of peeling off might require some additional requirements. It is likely that all agents will turn their heads collectively downwind. An option to remedy this is to make peeling off inactive for a set time in which a stable aligned group can emerge. When the stable group is there, initiate the wind. The average heading of the clustered agents can be calculated, and the wind will blow head on. Then the macroscopic pattern could emerge.
A possible undesired effect is a recursive 'push forward'. Because agents have an aligned heading and are repulsed by each other, this could lead to a constant movement forward, because movement forward is the only option. If and when this occurs, repulsive force has to be adjusted.
The predicted macroscopic behavior will incorporate a splicing of agents at the center of the collective which is in full wind. The orientation of agents at the center of this side will be chaotic, and more coherent to the sides.

5 Implementation

When attempting to construct a program it is wise to first look around. There are tools already available and this can reduce development time. This project makes use of such a tool, namely, 'Swarm' [19] software for multi agent simulation. In section 5.1 we discuss the general structure and usage of the Swarm simulation tool. Section 5.2 discusses the general model structure implementation, and in section 5.3 the specific agent behaviors are layed out as well as the interaction among them.
Sometimes we use programming code in our discussion of the implementation. This is done to clarify the structure and working of our implementation. However, the code used is *not* actual used code. It has been edited to its basic structural form to make it more readable, in other words, pseudo code.

5.1 'Swarm' simulation tool

> The basic architecture of Swarm is the simulation of collections of concurrently interacting agents: with this architecture, we can implement a large variety of agent based models [19].

We use Java Swarm, as opposed to the more often used Object C based implementations. Java works with classes, and consequently, Swarm has a hierarchical class structure. Classes can have subclasses, which in turn can have subclasses. The Swarm software has a largely pre defined basic structure. This structure consists of a Model Swarm, where our model is implemented. On top of this Model Swarm is an Observer Swarm. The Observer Swarm is a large toolkit for displaying actions and making the simulation visible. We discuss the *general structure* briefly. Also we will give a short description of how the simulation tool defines the order in the simulation of the simulation actions.

5.1.1 The observer and the model

Observer Swarm is the top class in our swarm implementation, a top skin. With this class it is possible to display *swarm activity*. With swarm activity we mean the actions of all agents in the current simulation.. This class makes it possible to visualize our simulation world and our agents. Roughly observer-swarm first builds its sub class, the model. After this it builds a schedule for displaying. A colormap is created and colors are defined.
Observer-swarm creates a window on screen, a 'raster', the size of our simulation world. The objects, the penguin agents, of our simulation all have a position on this world grid. Observer swarm offers tools to display these objects on screen. Deeper in the class structure objects can call a displaying function to 'draw themselves' on the display raster.

```
public Object drawSelfOn (Raster raster) {
  raster.drawPointXYColor (penguin.xPos, agent.yPos, color);
  return this;
```

```
}
```

The core of the actual model within swarm is named appropriately 'model swarm'. Model swarm is implemented as a sub class of 'observer swarm' in which it is displayed. Model swarm is a collection of all parts of our model.
The fist part is a model world. The world model is defined as a two-dimensional grid of size x by y. The world is the environment in which the agents are positioned and can move. The programmer can define the rules for the world, for example, whether an agent can walk over the edges of the world or not. The second part of the model is a collection of agents, our swarm. The agents are typically defined as a list of any number of identical agents. The programmer has to define a minimum of one type of agent, but any number is possible. We only need one type of agent.
'Swarm'-software offers useful predefined functionality. For example, in our model we have a class 'Penguin', and within this class the main function is 'walk'. We want our model to execute a 'walk' for every agents. In our model we have created a list of penguin agents; 'penguinList'. The solution is made quite simple. First we have to create a *selector* to access the appropriate function within the 'Penguin' class. Then we can use our list of agents, and create that action for each agent in our list.

```
selector = SwarmUtils.getSelector("Penguin", "walk");
modelActions.createActionForEachMessage(penguinList, selector);
```

5.1.2 Schedule

What we want to create is a running simulation of walking penguin agents. Every simulation time step we want to see the results of our agents 'walk' on screen. Swarm uses *schedules* for this. Schedules are basically a list of action in order of execution. 'Swarm' itself executes these schedules when compiled and run. Observer swarm has a schedule giving the order of updating the screen display. Model swarm has its own sub-schedule within the observer, giving the order of our model updating or modeling steps.
In the previous section we gave an example of the 'selector' tool. This tool is used again for schedule creation. A schedule is a collection of actions to execute every simulation time step. The only thing we have to do is insert the desired action in the model action collection at the correct place. This action collection is inserted in the simulation schedule. 'simulationZone' is the overall simulation zone, in which our schedule is inserted.

```
modelActions = new ActionGroupImpl( simulationZone );

selector = SwarmUtils.getSelector("Penguin", "walk");
modelActions.createActionForEachMessage(penguinList,
                                        selector);

// Now create the schedule and set the repeat interval
```

```
// to unity.
modelSchedule = new ScheduleImpl( simulationZone , 1);

// Finally, insert the action list into the schedule
// at simulation time-step zero
modelSchedule.atCreateAction(0, modelActions);
```

As stated before, ObserverSwarm displays the simulation on screen. As a consequence one can find the 'drawSelfOn' selector in the observer schedule, as well as the penguins 'setColor' action, which is accessed more directly during initialization.

```
ListImpl penguinList = modelSwarm.getPList();
for (int i = 0; i < penguinList.getCount(); i++) {
  Penguin penguin = penguinList.atOffset(i);
  penguin.setPColor((byte) 1);
}

selector = SwarmUtils.getSelector("Penguin", "drawSelfOn");
penguinDisplay = new Object2dDisplayImpl(simulationZone(),
                                         worldRaster,
                                         modelSwarm.getWorld(),
                                         selector);
```

The schedules in ObserverSwarm are somewhat less insightful, but we will not go further in explaining it. For more details the reader is referred to the 'Swarm tutorial' to be found on the swarm-website[13].

5.2 Model structure

Before we go into the specific behavioral repertoire and structure of an individual penguin agent in section 5.3, we will discuss the more general structure of our model. The relevant parts of the model for our implementation are the 'model world', the 'model wind' and the general layout of the 'penguin agent'.

5.2.1 Agent world

The world used in our model consists of a two dimensional grid. Every grid compartment is a possible location. At this location an environmental influence can be located, an agent, or just nothing.
In the model world it is possible to travel from the top of the world to the bottom and vice versa. The same applies to left to right movement. Such a world is known as a *torus world*, and is chosen because of the desired movement pattern of penguin agents. The desired macroscopic pattern is a group of agents moving, as a whole, downwind. As a consequence the simulated penguins require a lot of space to move in. A torus world creates an infinitely large space, without having to implement an infinitely large world.

[13] website: www.swarm.org

5.2.2 Wind

In the environment of a penguin a huge force is wind. We believe we need wind, as initiator of the macroscopic behavioral pattern. However, the minimal implementation of wind will suffice.
Currently wind is implemented as a directional vector. This vector has an orientation modeling the direction of wind flow. We make no use of wind speed what so ever. There is also no direct temperature influence of wind on environment or penguin agents.

5.2.3 Penguin agent

The third key part of the model implementation is the 'penguin agent'. The main routine of a penguin agent is the 'walk' function. This function gives the skeleton of the underlying subfunctions. The 'walk' function is called every simulation time-step for every agent because was inserted into the model simulation schedule. (As stated in section 5.1.2.)
The general structure of the 'walk' function is to, firstly reset all variables used. The second thing an agent needs to do is 'look' around for agents in our vision range. When an 'other' agent is seen, 'add' the influence of that seen 'other' penguin to the individual behavior vectors. When all surrounding grid compartments have been checked for 'others', 'calc' the resulting vectors of the individual behaviors. The specifics of the individual behaviors and the calculation of the vectors is discussed in section 5.3. These resultant vectors are added to create a 'resultHeading' and 'resultVelocity', which in turn are used in determining the 'newHeading' and the 'newVelocity'. Then finally an agent can 'move' 'leftOrRight'.

```
public void walk() {
  Penguin other;
  resetVar();
  while((other = look.next(this)) != null) {
    grouping.   add(deltax, deltay);
    alignment.  add(deltax, deltay,
           other.heading-x, other.heading-y,
           other.velocity);
    repulsive.  add(deltax, deltay);
    peeling-off.add(deltax, deltay);
  }

  grouping.   calculate();
  alignment.  calculate();
  repulsive.  calculate();
  peeling-off.calculate();

  // create the resultant vector. (only two behaviors are given.)
  resultHeading (grouping.getX(), grouping.getY());
```

```
  resultVelocity(grouping.getX(), grouping.getY(),
                 grouping.getVelocity);
  resultHeading (aligment.getX(), alignment.getY());
  resultVelocity(aligment.getX(), alignment.getY(),
                 alignment.getVelocity());

  // adjust velocity and heading according to the influence
  // of the resultant vectors.
  newVelocity();       // determine new speed.
  newHeading();        // turn head.

  if(move()) { leftOrRight(); };
}
```

In the situation that the agent does not see any 'others', the agent gets a random result vector to base it's behavior on, resulting in a 'random walk' or 'wandering behavior'.

5.3 Penguin structure

Our simulated penguins all consist of four body parts. These four body parts are positioned in a square on the world grid. The agent considers itself as if it is at the center of these four compartments. Meaning that in reality the agent does *not* have a position on the world grid. The reason for this artificially created position is that agents need to determine if an 'other' agent is in vision range, and therefore an agent needs a position to be able to determine the distance between this agent and an 'other' agent.
The 'look' class will not be explained in our discussion of our penguin agent. It will suffice to point it out to the reader that this class is a tool for looking up the next agent in vision range. This visible agent is the basis for behavioral influences on the current agent. The penguins *vision range* is set to 7 grid compartments, which is $3\frac{1}{2}$ body lengths far.
The basic penguin class structure and the used subclasses are given below.

```
public class Penguin {
  // the class for looking in the world.
  private Look look(visionRange);

  // Penguin body in the (simulation) world.
  public BodyPart BP1;
  public BodyPart BP2;
  public BodyPart BP3;
  public BodyPart BP4;

  // Behavioral sub classes.
  public Group   grouping;
  public Repulse repulse;
```

```
  public Align   align;
  public Peeloff peeloff;
}
```

5.3.1 The penguin agent

A penguin has an internal state. This state consists first of all of its position. The agent is positioned at the center of its own body-parts. Secondly an agent has an orientation and velocity. The heading is relevant when an agent wants to move, because this is only done in the direction of its heading. Heading has been given 'double' precision. In a previous implementation heading was an 'int', directly related to the eight possible compartments an agent can move to. This proved too discrete, and resulted in more deterministic agent behavior. (See section 6.1.2 for the complete discussion.) Now an agent has more *memory* in its heading, due to the continuous character of the heading vector.
The last internal state variable is velocity. Velocity is a value between one and zero, and states the odds for an agent to try to move forward. Velocity is a variable which was not used in our previous implementations. (For a complete discussion of the reasons for introduction of this variable, the reader is referred to section 6.1.)
To summarize, the internal state variables are:

```
  // My position in the world.
  public double position_x;
  public double position_y;

  // My orientation or heading.
  public double heading_x;
  public double heading_y;

  // My current speed.
  public double velocity;
```

In figure 8 a depiction of a penguin agent and its internal variables is given.

5.3.2 Behaviors to vectors

The behaviors of penguins are influenced by the agents surrounding an agent. The range of influence is set by the agents vision range. The agent checks goes through all compartments in a *square* around it. The size of this square is $2 * visionradius$ by $2 * visionradius$. This entails that some compartments out of *'real'* vision range are checked, namely, the compartments in the corners of the vision square. The distance a 'seen object' is away at such a position, is more than the vision radius. As a consequence there is some useless search.
This extra useless search of compartments can be avoided by stating a vision matrix of to-be-searched compartments, as is used in the previous implementations. However in the four compartment version of an agent this leads to

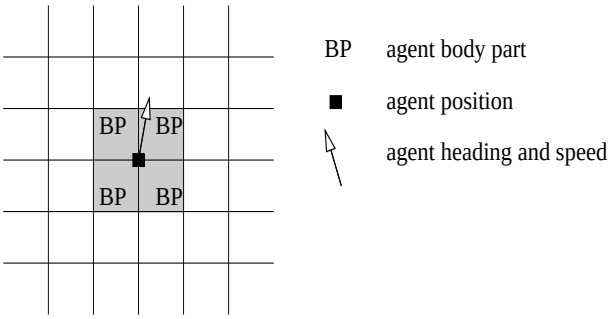

Figure 8: The penguin agent consists of four body parts. The body parts have a position on the simulation world grid. At the center of these four body parts the agent is located. The position of the agent is virtual, meaning, the agent is not really located on the world grid. However, the agent synchronizes its position with its body parts.
The agent has a heading into a certain direction, representing the orientation an agent has. The velocity of an agent also has a direction, and this direction is equal to the heading of the agent.

complications, as opposed to the earlier one compartment sized agents. This is due to the fact that now agents are not positioned in the world, but the body parts of agents. In other words, there is no possibility to directly access a seen agent, because only 'body-parts' are grounded in the world. A vision matrix, as used before, proved complicate to implement effectively due to a lot of additional communication between classes and conditional checks. Also, every agents needs to possess this vision matrix individually. In the end we chose for the simple and relatively fast implementation for the price of some useless search.
The function 'Look.next()' is used for locating the next agent in vision range on the world grid. This 'seenPenguin' is returned. The following 'Look.next()' call will give the next penguin in vision range until all agents in vision range have been located.
The 'seenPenguins' are the local influences on an agents behaviors. The influence of these agents is expressed through their influence on the individual behavioral vectors. These behavioral vectors are in their turn the basis for the resultant or summed vector, which influences the agents orientation and velocity.
Every individual behavior, e.a, grouping, repulsion, alignment and peeling off, is implemented as a separate class. This is not strictly necessary, but it does give a much more insightful implemenational structure. Each class has two main functions. The first function is 'add', and is used by the penguin agent to add the seen agent's effect on the particular behavior. The second function is 'calc' which calculates the result.

The resultant behavioral vectors are designed to have a length of maximally 1 and minimally zero. The standardization of vector lengths makes it easier to adjust specific behavioral influences relative to other behaviors.
How situations exactly translate or result in specific individual behaviors is made clear in the next paragraphs.

Grouping: An agent is implemented as drawn towards the average position of its surrounding seen agents. The function 'add' is called with δx and δy. $(\delta x, \delta y)$ represents the distance the seen penguin agent is away from this penguin agents current position. The function 'add' adds these differences in position and registers how many other agents it has seen until now in 'numSeen'.

```
// function 'add' in class group
public void add(double deltax, double deltay) {
   numSeen++;
   x += deltax;
   y += deltay;
}
```

The 'calc' function translates the vector stored in (x, y) to $(\overline{x}, \overline{y})$ by dividing (x, y) through the number of agents seen. The length vector $(\overline{x}, \overline{y})$ can be seen as indicative for the importance of grouping. If this vector is small, one can deduce that the agent is surrounded by others because the average position of seen agents in this situation is close to zero. If this vector is large one can conclude that the agent only sees some agents in a particular direction, or that this agent is standing at the edge of a group of agents.
The length of $(\overline{x}, \overline{y})$ is directly translated to importance of the grouping behavior. If $(\overline{x}, \overline{y})$ is large, grouping behavior is important. if $(\overline{x}, \overline{y})$ is small, grouping behavior is less important. We have implemented this relation by use of a linear function, defined by the variables $f1$ and $f2$. This relation is as depicted in figure 10. The program code for the 'calc' function is:

```
// function 'calc' in class group
public void calc() {
   x = x / numSeen;
   y = y / numSeen;

   importance = distance(x,y);
   double a = 1 / (f_2 - f_1);
   double b = (-a * f_1);
   velocity = max(0, min(1, ((importance * a) + b)));
   x = velocity * x;
   y = velocity * y;
}
```

Repulsion: An agent is repulsed by other agents, if the particular seen agent is too close. We do not want repulsive force to be a dominant force. A large

repulsive force results in low penguin agent densities on the world grid, because agents will have larger distances in between them. This would be undesirable and unrealistic compared to the natural penguins. Our model states that repulsive force is designed to counter-act grouping, and in doing so, avoiding the emergence of clusters stuck in a steady state due to a dominant grouping force. An agent needs to respond to a particular individual repulsing it. For this reason the repulsive force is directly calculated for each seen individual.
The force experienced is determined by a quadratic function defined by through $f1$, which is set to $4\frac{1}{2}$. It must be pointed out that the input distance $(\delta x, \delta y)$ has a minimal distance of 2 because the centers of the four-compartment-sized agents is used. Effectively this entails that agents experience a repulsive force, of importance, only when 'skin to skin' with one-another. The 'add' function will not be called when the vector $(\delta x, \delta y)$ is larger then $4\frac{1}{2}$.

```
// function 'add' in class repulse
public void add(double deltax, double deltay) {
   double importance = distance(deltax,deltay);
   importance = ( 2 * (importance - f_1)) / f_1;
   importance = importance * importance;

   double angle = vectorAngle(deltax, deltay);
   sumImportance += importance;

   // resize the vector to 'importance'-size, and add it
   // (inversely) to the resultant repulsive vector.
   x -= tool.x_vectToSize(angle,importance);
   y -= tool.y_vectToSize(angle,importance);
}
```

As can be seen in the above stated pseudo code, a variable 'sumImportance' is used. 'sumImportance' always has a positive value and is a summation of all importances or repulsive vector lengths. In function 'calc' the repulsive-force vector (x, y) is divided by the sum of importance vectors, to make the resultant repulsive force vector a standardized value between zero, no repulsion, and one, maximum repulsion. However, when only *one* small repulsive force has been experienced, this would result in a repulsive behavioral vector with a vector length of one, in other words, maximum repulsion. Therefore 'sumImportance' is corrected by setting its value to unity when it has a value below one.

```
// function 'calc' in class repulse
public void calc() {
   sumImportance = max(1, sumImportance);
   x = (x / sumImportance);
   y = (y / sumImportance);
}
```

In repulsion we do not use a extra variable for the storage of a velocity vector. Vector (x, y) is translated directly to velocity. The length of this vector is

indicative for determining the repulsive force expressed in velocity, and therefore used directly.
A depiction of the relation of distance to an object and the influence on the repulsive behavioral vector is given in figure 11.

Alignment: An agent has observed an agent in vision range. It will attempt to move its own heading into the direction of the *average* heading of the surrounding agents. As a consequence the function 'add' of in the class alignment is called with `head_x` and `head_y`, the heading of the 'seenAgent'.
Additionally 'add' requires the relative position of the 'seenAgent'. Alignment uses the heuristic that nearby agents influence alignment more than far away agent in vision range. Importance is determined according to a linear function, as depicted in figure 10. The importance can be parameterized by setting the $f1$ variable and is limited to unity importance. $f1$ is currently set to the value of 3, in other words, 'seenAgents' only influence alignment maximally when close by.
We have also experimented with velocity alignment. The idea is the same as with heading alignment, only now the agent will try to match its own speed to that of surrounding agents. (Velocity alignment is however still in its infancy in our current implementation.)

```
// function 'add' in class align
public void add(double deltax, double deltay,
                double head_x, double head_y,
                double velocity) {
   numSeen++;
   double a = 1 / (f_1 - vision);
   double b = -a * vision;
   double importance = min(1, (distance(deltax, deltay)
                               * a + b));
   double angle = vectorAngle(head_x, head_y);
   x += x_vectToSize(angle, importance);
   y += y_vectToSize(angle, importance);
   velocity_x += x_vectToSize(angle, velocity);
   velocity_y += y_vectToSize(angle, velocity);
}
```

To align towards the average heading, we need to divide though the number of 'seenAgents', given by 'numSeen'. The function 'calc' does exactly this. The function 'calc' also determines the average velocity in the same manner as with the average heading.

```
// function 'calc' in class align
public void calc() {
   x = (x / numSeen);
   y = (y / numSeen);
```

```
   velocity_x = velocity_x / numSeen;
   velocity_y = velocity_y / numSeen;
}
```

It must be stated that velocity alignment is *not* heading independent in our implementation. In the 'add' function the velocity is translated to an x and y component related to the 'seenAgent's heading, which is added to the result velocity. One can imagine a group of agents all oriented in different directions. In such a situation the resultant heading for alignment is near zero. However, if we used velocity adjustment independent of heading, agents would get a high velocity due to alignment. We chose not to do this, but instead making velocity alignment heading dependent.

Peeling off: Peeling off is designed to be active when two conditions hold:

- The agent is located at the rim of a group.
- The agent has no shelter from the wind.

The first condition is determined by use of the average group position of agents in vision range, which is calculate in the same manner as for 'grouping'-behavior. How exactly this is used will be made clear when discussing the 'calc' function. The 'add' function code is the same as for grouping.

```
// function 'add' in class peeloff
public void add(double deltax, double deltay) {
   numSeen++;
   x += deltax;
   y += deltay;
}
```

The function 'calc' gives an output vector representing the importance of the peeling off behavior in the current situation. First the function determines $(\overline{x}, \overline{y})$ or the average group position. The following program code determines the angle between the direction from which the wind blows and the group position. An angle of zero would mean that the agent is positioned at the in-wind side of a group, as can be seen in figure 9. This angle is translated to the first importance factor. When the angle is zero, the importance is unity. When the angle is 120 degrees or more, the importance is zero. See also figure 11.
However, the angle between wind direction and $(\overline{x}, \overline{y})$ can be at zero also in situations in which the agent is *not* positioned at the wind-side rim of a group, for example, when the agent is inside a group of agents and the average group position is coincidentally a little bit positioned towards the wind-lee side. In such a case peeling off is not an appropriate behavior. We define 'at the rim' of a group by usage of the average group position. The vision range of an agent is $3\frac{1}{2}$ body lengths or 7 compartments far. If $(\overline{x}, \overline{y})$ is small, then the agent is sufficiently centered at a group. If $(\overline{x}, \overline{y})$ is large, only a small number of agents are seen and the agent needs to 'group' instead of 'peel off'. To accommodate

this observation we have introduced a second importance function directly related to the length of vector $(\overline{x}, \overline{y})$, depicted in figure 11. The importance is at its peak when $(\overline{x}, \overline{y})$ has a value between 2 and 3.
The combination of these two importance vectors is indicative of the current relative position of the agent, and are multiplied. This resultant importance is translated directly to the resultant peeling off output vector.
Currently the agent does not peel off into a specified direction. The agent gets a peeling off vector into a random direction. The idea is to make the agents turbulent and highly active at the wind-side of a group and in combination with the other behaviors the agents will eventually become at ease again when sufficiently sheltered. This will occur when the agents have moved along the sides of the group, which is our desired peeling off behavior.

```
// function 'calc' in class peeloff
public void calc() {
   x = x / numSeen;
   y = y / numSeen;

   double angle = min(120, abs(vectorAngle(x,y,
                                wind_x, wind_y)));
   importance1 = abs(angle - 120) / 120;

   double length = distance(x,y);
   if     (length <= 2) { importance2 = length / 2; }
   else if(length <= 3) { importance2 = 1; }
   else { importance2 = max(0, 2.5 - (length / 2)); }

   importance = importance1 * importance2;

   angle = randomDbl(-180,180);
   x = x_vectToSize(angle, importance);
   y = y_vectToSize(angle, importance);
}
```

Currently the peeling off behavior is also active when, for example, only one agent is seen which is positioned downwind and has a δx and δy within range of the second importance function. An additional function checking the number of agents seen should also have been implemented. However we have not done this.

5.3.3 Activity and result vectors

All behaviors, grouping, alignment, repulsion and peeling off, give resultant vectors. The lengths of these vectors gives the importance of that specific behavior. The behaviors all return standardized vector sizes of lengths between zero and unity. We have implemented the option to specify the importance of every behavior individually. The implementer can specify a factor with which the length

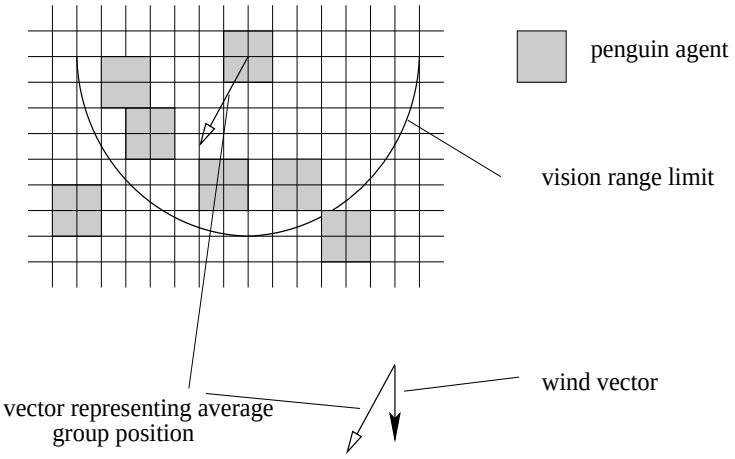

Figure 9: Natural 'peeling off' behavior is active in the situation that the agent is standing in the group at the cold wind-side. In the implementation the wind is modeled as coming from the north and represented as a vector. When an agent is standing at the wind-side, the direction of the vector $(\overline{x}, \overline{y})$ is similar to the direction of the wind vector. The angle between these vectors can be directly translated to importance of peeling off behavior in a particular situation.

of vectors are multiplied, and in doing so, the relative importance of a specific behavior. Currently these vectors are all set to unity.

```
public double max_activity_peeloff   = 1;
public double max_activity_grouping  = 1;
public double max_activity_aligning  = 1;
public double max_activity_repulsive = 1;
```

All behaviors, grouping, alignment, repulsion and peeling off, give resultant vectors, now scaled between *zero* and $unity * maxActivityBehavior$. The length of these vectors represents the importance. However, an agent only has a set amount of activity which it can use. As a result an agent may have a specific behavioral vector which it can not attend to.
A preset variable is the maximum activity an agent can use up. Currently this is set to the value of 3. At the beginning of the agent's 'walk', the variable 'availableActivity' is set to this value. Before a specific resultant behavioral vector is added to the end-resultant vector, which influences actual behavior, there is a check forced to see if there is any activity to use left. The function requires the specific behavioral vector when called and returns the activity it can use.

```
private double getActivity(double x, double y) {
   double length = 0;
```

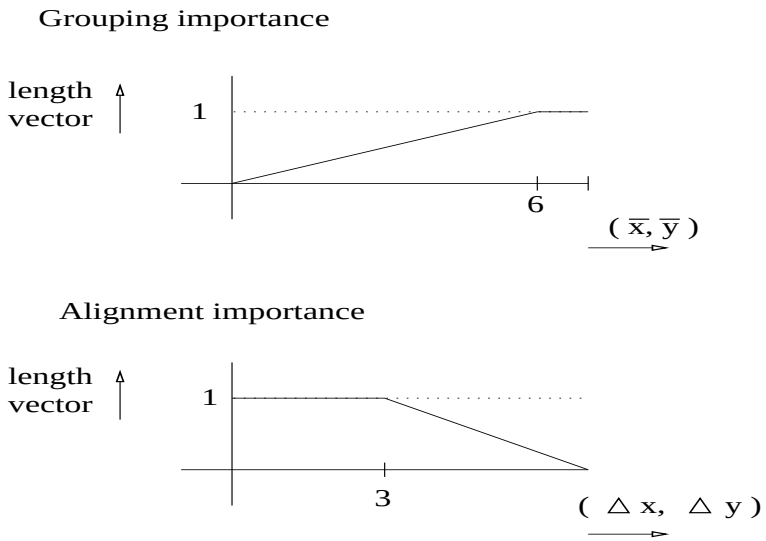

Figure 10: Grouping behavior is based on $(\overline{x}, \overline{y})$; the average position of the visible agents. When $(\overline{x}, \overline{y})$ is small, grouping behavior is less important, since the group average position is near and the agent is located at the group's center. When $(\overline{x}, \overline{y})$ is large and near the vision boundary of 7, the group is positioned far away and grouping behavior is important.
Alignment influence of a seen agent is based on the distance $(\delta x, \delta y)$ the seen agent is away. The resultant alignment vector is the average of all these alignment importance vectors.

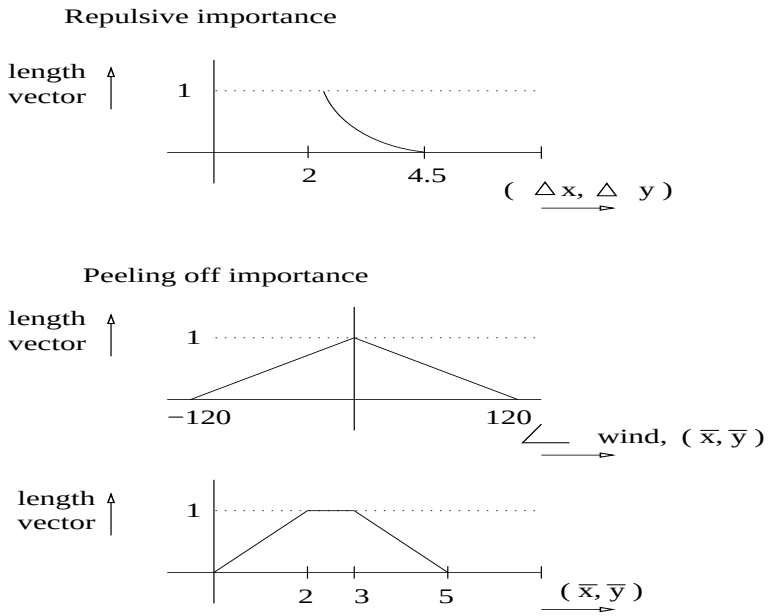

Figure 11: The repulsive force produced by a seen agent is based on $(\delta x, \delta y)$, the distance the agent is away. The agent has a size of 4 world compartments and considers the center of it's four body parts as it's actual position. Thus, the minimal distance $(\delta x, \delta y)$ can have is 2, when two agents are standing next to each other.
The importance of peeling off behavior in an agent's situation is based on two things. The first importance factor is the angle between the wind vector and the group vector $(\overline{x}, \overline{y})$. The second importance factor is the distance the group average position is away from the agent.

```
   if(available_activity > 0) {
      length = vectToAmpl(x,y);
      if(available_activity < length) {
         length = available_activity;
      }
      available_activity -= length;
   }
   return length;
}
```

When there is enough activity left the 'getActivity' function returns the input vector length. When there is only a small part of the activity available, it returns that activity. The behavioral vector in question is consequently resized to the activity it had available for it.
The sum of the behavioral vectors, and consequently the activity, can exceed the available activity. The agent has a maximum to be used activity of 3. When all behaviors are maximally active the sum of their lengths would be: $\sum maxActivityBehaviors = 4$. This excludes the last behavior from having an influence on actual penguin agent behavior. The order in which the behaviors are called is, from first to last, grouping, alignment, repulsion and peeling off. In effect peeling off runs the risk of being excluded from having influence.

Now all resultant behavioral vectors have been resized to the scale determined by the available activity. The four behavioral vectors are added to create the 'new heading (x, y)' and the 'new velocity (x, y)'. These vectors are the basis for change in the internal state of an agent. What the effects are is discussed in the following section.

5.3.4 Movement and rotation

The results of behavioral influences are summed up by the vectors `new_heading(x,y)` and `new_velocity(x,y)`. These two vectors are the basis for adjustments to the agents current speed and heading.
The result vector `new_heading(x,y)` gives behavioral motivation to turn or rotate in a direction. We do not want agents to be able to make a u-turn based on this vector. We have implemented a maximum influence of 45 degrees to the left or right. In effect the angle between the result vector and the agent heading vector is trimmed to $+/-45$ degrees. The new heading vector of the agent is set to the angle of it's current heading plus the trimmed result vector.

```
private void newHeading() {
   if(vectToAmpl(new_heading_x, new_heading_y) > 0) {
      angle = vectorAngle(headx, heady,
                          new_heading_x, new_heading_y);
      if(angle > 45 ) { angle =  45; }
      if(angle < -45) { angle = -45; }
```

```
      angle = angle + vectorAngle(headx, heady);

      headx = x_vectToSize(angle,1);
      heady = y_vectToSize(angle,1);
   }
}
```

The new velocity of the agent is the average of it's current velocity and the velocity determined by the result vector.

```
private void newVelocity() {
   angle = vectorAngle(headx, heady);
   sumx = (x_vectToSize(angle, velocity) + new_velocity_x)/2;
   sumy = (y_vectToSize(angle, velocity) + new_velocity_y)/2;
   velocity = vectToAmpl(sumx, sumy);
}
```

All behaviors have had their influence on the agent, resulting in a new heading and a new velocity. The last step in an agent's 'walk' is to actually make a step in the simulation world. First an agent decides if forward is done. The function 'move' makes a decision based on the value of velocity. A random value is calculated between zero and one, and is compared with the current speed. When velocity has a higher or equal value compared to the random value the function returns 'true'. All other situation result in a 'false' and in such a case the agent will attempt to move.

```
private boolean move() {
   if(random(0,1) < velocity) { return true; }
   else { return false; }
}
```

When an agent has decided to move, the following action done is to decide where to move to. The heading of an agent is a real value, and so not directly related to the eight possible direction an agent can possibly move to.
We have restricted the possibilities of movement to two possibilities. The two options are the two closest obvious choices *related to heading*, as depicted in figure 12. First we calculate the amount of degrees the 'left move' actually is and secondly the 'right move'.
Finally we can make the choice between a left or a right move. We have incorporated the element of chance into the decision making process. When the penguin agent's heading in degrees is closer to 'left', the left move is preferred and a right move is less likely.

```
private void leftOrRight() {
   int left = 0;   int right;
   double angle = vectorAngle(headx,heady);
   for(int i = -180; i < angle; i += 45) { left = i; }
```

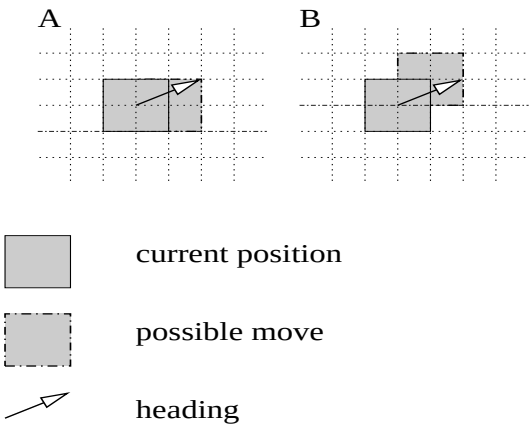

Figure 12: The agent is restricted in possible locations to move to, relative to its heading. The heading an agent has makes two moves possible. The angle between the heading vector and the vector representing the move is in situation A: $26.6-0 = 26.6$ degrees and in situation B: $45-26.6 = 18.4$ degrees. The agent has a chance of $1 - \frac{26.6}{45} = 0.41$ to make move A and a chance of $1 - \frac{18.4}{45} = 0.59$ to make move B.

```
   right = left + 45;

   if(random(0,1) > ((right - angle)/45)) {
      go(right);
   }
   else {
      go(left);
   }
}
```

The agent has decided to move and has decided where to move to. It might however be the case that the compartment(s) an agent wants to move to are already occupied by another agent. In such a case the movement will fail.

6 Results

The implementation of the models has been an iterative[14] process. Piece by piece the implementation is extended to incorporate a new element of the model. The discussion of the result follows the same iterative process. However before we discuss the results of the current simulation in section 6.3, we describe the developmental process in section 6.1. During development and simulation interesting unexpected situations occurred, which gave rise to new models which finally led to the current model and current simulation. Therefore the developmental process could not be omitted in this chapter. In the last section, section 6.4, we reflect on our goal definition and our simulation. This section also describes some additional general results, such as, the influence of the number of agents on the resultant simulation, and more.

6.1 Developmental process

Before going into the specifics of the currently implemented model in section 6.3 we describe the evolution of this model. We elaborate on the evolution of our current implementation to give the reader a clear picture of what considerations and observations are key to the introduction of new concepts and/or approaches in our models. Because the process of development has been an iterative one, building upon the results of previous models, this has to be included in the results.
In the following sections we will recreate the iterative developmental process. The modeling process, which has experienced the same iterative process, has been discussed in sections 4.2 and section 4.3. The modeling and results are obviously linked to each other. Concerning the results, the developmental process has had some interesting characteristics and thus cannot be omitted in the results section. First we describe the 'particle model' and simulation, followed by the first 'behavior based' simulation and its successor, the second and last 'behavior based' model and the resultant implemented simulation. Concluding results based on this developmental process are given in section 6.1.4.

6.1.1 Particle simulation

The first resultant simulation was based on the conceptional view of an analogy between penguin behavior and particle flow. Individual penguin agents are viewed as simple particles which flow towards higher temperature regions on the world grid. A agent is the size of one grid compartment and can move to all of its eight neighboring compartments. The agent decides where to move based on the temperatures in the surrounding eight compartments.
Every agent deposits 'heat' upon the world. The agents produce *body heat* in

[14]Iterative development process: first a model element is implemented, tested and corrected. After completion of this element, a new element is added to the implementation. This element goes through the same process of implementation, testing and correction. This process continues till completion.

the compartments around them, defined by a heat matrix. (See figure 5.) The second heat producing factor is *wind*. Wind is implemented, not by lowering the temperature in compartments in full wind, but by raising the temperature in compartments which are in lee. Since penguin agents are the only modeled wind obstruction and thus lee creating influences in the world, lee is defined around every agent. How wind direction and lee influence the world grid and the deposits of heat is defined by the wind matrix. (See figure 6.)
The expectations of our first model are that a 'heatscape' is created over the agent world. Agents themselves essentially probe the surrounding eight compartments for the highest temperature and want to move there. One could envision the idea behind our 'heatscape' as a temperature mountain, slightly displaced downwind compared to the agents positions. Due to the body heat of agents, agents will cluster together, and due to the wind the heatscape-mountain is displaced downwind.
The resultant simulation displayed clustering behavior of agents into large groups. These groups were largely static or non moving. Occasionally an individual agent 'in wind', thus in the colder region on the grid world, would fall like a drop of water down the sides of the group to the lee area of the group. The macroscopic pattern we attempt to model is there in a basic form, however, the simulation is too *static*, in other words *not moving* and quickly stuck in a steady state. To remove the static character of the overall behavior the chance factor can be increased, in turn leading to very *low cohesion* of the groups. The low cohesion is due to the limited vision range of agents. Agents see each other *implicitly* through the deposited heat in a neighboring grid compartment. The size of the heat matrices determines implicitly the vision range of agents. Now an individual agent quite often wanders off far enough to make the group it belonged to impossible to see.
Agents in isolation produce their own little heat-mountain with a displaced peak toward their own lee side. The isolated agent walks downwind, towards the heat-peak created by itself. This is also quite unrealistic individual behavior. Even if one makes the heat producing matrices in such a way, that an agent in isolation has the peak of the local heat-landscape at it's current position, still the agent would have to be highly motivated to seek out a higher temperature world compartment for it needs to find a group of other agents. So this still would result in movement towards its own lee side, which in such a case, is the second best temperature compartment.
An interesting observation is that the enlargement of the heat matrices defining body-heat and wind-heat increase cohesion enormously. Enlargement entails influence in temperature in the world further away from the agent. The matrices define the vision range implicitly, and thus quite logically this is a determining factor for the cohesion of the groups. However the enlargement of the heat matrices, especially of body-heat, is not valid. Body-heat cannot be realistically felt more than one compartment or agent body size away.
From the irratic or chaotic individual movement resulting from always walking to the highest temperature neighboring grid compartment, we conclude that the direction an agent can move in must be restricted. This restriction of movement

can be based on a to be introduced concept of heading or orientation an agent has. Another method to combat the chaotic individual behavior is to make explicit behaviors active in certain situations. We also want to increase the cohesion of the groups. Cohesion can be increased by stating larger matrices in the particle simulation, thus implicitly enlarging the vision range. Somehow a larger vision must be created, some sort of vision range is needed.
These considerations have led to a new approach; a behavior based approach. Within this frame it is possible to make behaviors and introduce an increased vision range. The behaviors an agent exhibits can be based on the agents around it in a set vision range.

6.1.2 First behavior based simulation

We want to create logical individual behaviors for agents. To accomplish this we give each agent a set number of behaviors. In different local situations, different behaviors are active. To avoid tedious time consuming parameterizing we state that only one specific behavior can be active at a time. A depiction of the model is given in figure 13.
The agents now have an internal state. This internal state is an agents heading or orientation. This is a *characteristic vector*,

$$\{vector(x,y) \in \mathbf{R}\times\mathbf{R} \mid (1,1),(1,0),(1,-1),(0,1),(0,-1),(-1,1),(-1,0),(-1,-1)\}$$

representing one of the possible eight compartments an agent can walk to. Also an agent has a vision range determining how far an agent will look for other agents. The vision field consist of all grid compartments around the agent within the radius set by vision range. Other agents in vision range are the basis for determination what specific behavior needs to be performed at that particular time.
We had two distinct behaviors implemented, alignment and grouping, and were working on the third behavior, peeling off. The behaviors are implemented as follows:

- Grouping: Determine in what quadrant *relative to the heading of the agent*, north, south, east and west, the most agents are located, and try to move there.
- Alignment: Add all heading vectors of agents in vision range. The result is the orientation of the group.
- Peeling off: Determine if there are enough agents obstructing wind, in other words, if the agent is in lee or not.

The use of quadrants as the basis for decision making of what particular behavior needs to be active at the current time followed logically from our implementational design. Easily and rapidly the values needed can be calculated. We will not go into the specifics of the implementation.

The flaws in the resultant simulation, with only two behaviors active, are already very apparent. Agents group together into cohesive groups. When sufficient agents are present the grouping behavior is abandoned and alignment becomes solely active. The groups align into the same direction. However, because of the *sole activation* of alignment in the current situation wholes emerged within the clusters. The number of agents in vision range is too high for the grouping behavior to become active. When the threshold determining the shift between the grouping behavior and alignment behavior is set higher, the group will often become trapped in a steady state. Too many agents at the rim of the clusters are grouping and obstructing the aligning agents at the center. When the threshold determining the shift from grouping to alignment is set lower, the simulation shows clusters of agents with more and larger wholes within these clusters. Apparently there is still the need for a grouping force within the clusters. We conclude that every behavior needs to have influence at all time, only not to the same extent. The influence a behavior must have is based on that particular local situation.
A second observation is the highly deterministic behavior of individual agents. This is a result of the bases on which behaviors are chosen: characteristic quadrants. Also, the heading of an agent is always into one of the eight characteristic directions. The characteristic quadrants and heading directions work as *generalizers over situations*, resulting in often identical behaviors in different but similar situations. We conclude that we need a continuous internal representation of the current state, meaning heading. The continuous internal representation has more expressiveness for representing behavioral influences as opposed to the eight characteristic vectors. The same holds for the division of the 'vision field' in four characteristic quadrants, north, south, east and west, from which the four basic variables resulting in agent behavior are determined. Behaviors must not be based on characteristic quadrants, but a more continuous and therefore more expressive implementation is needed. Situations must be discriminatory for the agent to avoid the current deterministic behavior.

6.1.3 Second behavior based simulation

The last and final simulation created is based upon a second behavior based model. The basis of the model is analogous the BOIDS [24] implementation and consists of the same basic behaviors, namely, alignment, grouping (cohesion) and repulsion (separation). An advanced BOIDS[15] implementation which is capable of avoiding objects is used for the design of activation values for behaviors and ordering of the behaviors. The individual behaviors all give standardized vectors, which in turn result in an end-resultant vector to influence actual behavior of the agent. The end-resultant vector influences actual movement behavior of the agent.
The internal state of an agent is a continuous heading, no longer restricted to

[15]The advanced BOIDS implementation can be found at the website:
www.red3d.com/cwr/boids/index.html

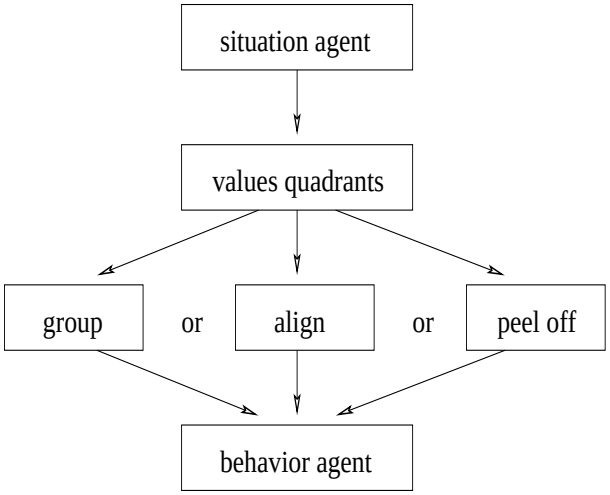

Figure 13: A schematic depiction of the first behavior based model. The current local situation of an agent determines which behavior is active. This particular behavior is sole 'influencer' on the resultant penguin agent behavior.

the possible locations an agent can move to. A complete discussion is given in section 6.3.

6.1.4 General developmental results

During the development, implementation and simulation of our concurrent models we have had a general shift towards *increasingly continuous* or *'elaborate' implementations*. The first 'particle' model is a simple, very localized implementation. In hindsight it is striking that this implementation already exhibited the basic 'peeling off' behavior, shown as individual agents falling along the sides of the group. Individually the agents are very unrealistic in this particle simulation.
The first behavior based model introduced a vision range, heading of agents and behaviors. Quite quickly it is realized during simulation that the heading restricted to the eight possible locations to move to was too restrictive. Additionally the way the grouping and alignment are implemented, using characteristic quadrants, proved to be a source for deterministic behavior as well. Finally the behaviors have been made more or less continuous, resulting in resultant vectors not restricted to the value of active or not, but also all values in between. Heading has gone through the same process. We elaborate on the importance of 'continuousness' in section 6.4.3.

6.2 General simulation results

Before going into the details of the implementation's results, a general implementational issue needs to be addressed concerning 'world updating'. This is done in section 6.2.1.

6.2.1 World updating

There are basically two different ways to update the world: but each with great implications for the result. The developer has to be conscious of these implications. A clarification is given, with an example. This example is taken from the 'particle'[16] implementation of the first model. Knowledge of the specifics of the particle model is not needed. We only wish to illustrate the general structure of world updating in a simulation.

- collective update: The changes in the environment are determined at the beginning of a cycle. In the model these changes are described in terms of temperature influence. When the world has been updated, all agents move. In pseudo code this looks like:

```
Init ( World )

Do ( HuddleHeat ( PenguinList ) )
// calculate huddling effect on world

Do ( WindInfluence ( PenguinList ) )
// calculate wind temperature effect on world

ForAll ( PenguinList ) Do ( Walk ( aPenguin ) )
// let the penguins walk in world
```

- individual update: The changes in the environment are determined individually. In the particle example the environmental influences are produced by the penguin agents themselves. This results in the following pseudo code:

```
Init ( World )

ForAll ( PenguinList ) Do {
  Do ( Walk ( aPenguin ) )

  Do ( HuddleHeat ( aPenuin ) )
```

[16]The environment of the penguin agents consists of a 'heatscape'. This is a landscape in which the agents move, with high peaks where temperature is high and lower peaks where temperatures are lower. The shape of the heatscape is determined by the position of individual agents, and the influence of the wind. The idea is that an individual agent will try to move to the highest peaks.

```
    Do ( WindInfluence ( aPenguin ) )
    // make changes in temperature in the world,
    // due to this particular agents movement.
    // changes are defined in body-heat and wind-lee.
  }
```

The above mentioned remarks may seem trivial or obvious. This is not so for the novice programmer of artificial life. The two different implementational versions have varying results. These differences can be quite significant. Ordinarily, these difference are unexpected.
When collective world updating is used in stead of individual world updating, the cohesion amongst the agents is greater. This is an unexpected emergent property. The reason for this seems to be that in the individual update approach different temperature worlds arise after each individual step, as opposed to every simulation step. As a consequence the heatscape is more in turmoil than in the collective update case. And the consequence of this is that an individual agent is more easily isolated in the individual update case. The term isolated refers to the situation in which an agent cannot see other agents (implicitly through higher temperatures in compartments).
The implementation uses individual update. As a result there is more chaotic collective behavior. The reason for the choice of individual instead of collective update is that individual update is more realistic. Also, individual update is obviously more up to date with the actual or current world situation.

6.2.2 Chance

Whenever simulations are run of artificial life it is reported that chance needs to be incorporated. Especially in the first implementations of the 'particle-approach', we had results which emphasize the use of chance. Groups of agents can show cyclic behavior in isolated parts of the world. There is always a logical best compartment to move to. However, during simulations it is observed that it is better to take into account the second best compartment also. This can even be stated more strongly, by stating that chance is a *necessary* factor in our type of simulation work. Furthermore, when one programs an agent without using chance, one defines exactly in what order an agent performs actions. This order returns in what is seen in the resultant simulation[17]. The only way to remedy this is to incorporate chance. Yes, chaos is a good thing.
In the current implementation we have incorporated chance at two point in our program: the decision 'if an agent tries to move' and the decision 'what compartment to move to'. These decision are made on the basis of, respectively, the agent's velocity and the agent's heading. The simulation does not exhibit

[17] During the first 'particle' simulations the large clusters of agents always had a distribution according to the order in which moves were tried. E.g, if the order stated that agents tried to move to the upper-left compartment first, the collective would be stretched to the upper-left.

cyclic sub-group behavior and the programed order in the program-code is not apparent in the simulation.

6.2.3 Shape of world

Agents move around in the simulation world. An individual agent is limited in the possible moves it can make. An agent can move to all eight neighboring compartments as depicted in figure 4. This does have some implications on the shape of the world and one needs to be conscious of this. In typical artificial life simulations agents can move to the compartments next to it and above and below it; horizontally and vertically. If an agent can also walk diagonally, as is the case in our simulations, the actual shape of the world is twisted. On the computer screen, during the simulation, a diagonal step has a distance of $\sqrt{horizontal^2 + vertical^2}$. For the actual agent this is not so. In the simulation the time taken for an agent to walk diagonally across the world is the same as the time taken for a horizontal walk. The diagonal distance is the same as the horizontal or vertical distance.
In the literature we have not found an elegant and/or computationally simple solution for the world-shape distorting implications. To sum up: if the programmer decides to allow diagonal movement as we have done, he or she must be aware of the distorting implications.

6.3 Simulation results of the final implementation

The results of our final behavior based implementation and its resultant simulation are described in this section. First we describe the penguin behaviors individually. It must be taken into account that all the behaviors are designed and parameterized in accordance with the role and interaction the specific behavior has within the whole of behaviors. In our discussion of the results we give two descriptions of the individual behavior, namely, in the situation that the behavior is the sole behavior of all agents and in the situation that the behavior is a part of the set of agent behaviors.
During our discussion of the results we give examples. All examples are based upon a simulation world of size 100×100 with about 107 penguin agents. The wind is blowing from the top to bottom, vector $\{0, 1\}$. The vision radius is 7 compartments, or $3\frac{1}{2}$ body lengths.

6.3.1 Grouping

Grouping and repulsion are directly related to each other. Repulsion is designed as a force to counter grouping and by doing so, avoid the emergence of a steady state. When the grouping behavior is solely active we see as a result a simulation which is quickly stuck. After 20 simulation time steps, agents are already tightly packed within groups with size varying from 2 to 9, aligned towards the center of the local group. About $\frac{2}{3}$ of the agents is not moving any more; stuck. Agents do not see any other agents to initiate grouping behavior or agents are standing

with their heads together. Quickly their is no movement to be seen any more. When grouping is an active behavior amongst the other behaviors agents do not become trapped in a steady state. To give an indication of the grouping behavior we give the distribution of agents over time in table 1. We conclude

Table 1: Simulation of grouping behavior (107 agents).

Time-step	*groups*	*group size*	*solo agents*
0	-	-	all
20	13	3 to 12	20
50	9	7 to 28	some solo
100	5	10 to 46	some solo

that the grouping behavior does what is required of it. However, the balance between importance of this behavior, which is directly related to the activation the behavior gets, *and* repulsion *and* alignment *and* to a lesser degree peeling off is key. The grouping behavior in our simulation is implemented as being highly active when the vector $(\overline{x}, \overline{y})$ is large and thus the agent is standing in relative isolation, and dropping when $(\overline{x}, \overline{y})$ is small. The small vector for $(\overline{x}, \overline{y})$ occurs when the agent is surrounded by others. The resultant standardized grouping vectors range correctly with the situation an agent is in. When the clusters have formed the standardized vectors for grouping range in length from 0.35 to 0.7. These values are stated to give the reader an indication of the resultant importance and variance in resultant importance of this particular behavior in the simulation.

6.3.2 Repulsion

When we run the simulation with only one behavior, namely our repulsive behavior, not much can be observed. Initially the agents are randomly distributed on the world. Occasionally agents will stand too close to each other and consequently are a repulsive force to each other. In under 10 simulation time-steps all agents have adjusted their initial position and have moved away from one another when too close. All agents have a minimal gap between them and others the size of one body length.
If all four behaviors are active, we cannot see the effect of repulsion directly. The importance of this behavior becomes apparent when *no* repulsion is included into the behavioral repertoire. After about one hundred simulation time-steps the simulation is rather static. The agents are still shifting with their alignment, but more importantly, static clusters have emerged with all agents with their heads turned towards the center. The agents cannot align altogether, because the grouping force pulls too much towards the center. The emerged clusters have maximum density and are not moving.
We have fine-tuned our repulsive force to the grouping force. This is demonstrated nicely when these two behaviors are the only active behaviors. The

agents are pulled towards each other and when too close they are repulsed: a continuous push away, followed by a pull back. In the beginning phase, the first hundred simulated time-steps, the agents are clustered in small groups of ± 8 agents. As time progresses the clusters become larger when occasionally a collision between clusters happens. As the clusters reaches a size of $+20$ agents, a stable core emerges with a turbulent rim. The agents at the center have equally large repulsive forces coming from all directions, resulting in a sum repulsive force of zero. This happens after 400 simulation time-steps.
If all four behaviors are active the resultant macroscopic behavior with regard to repulsive force is as described in the previous paragraph. There emerge cores in clusters which are relatively stable, non static, and more distance in between agents towards the rim. The repulsive force experienced by individual agents ranges from 0.00 to 0.70. And again, the standardized repulsive force, in other words, the result repulsion vector length, experienced by an agent closely packed at the core, is close to zero.
We have implemented the repulsive force to be a quadratic function related to closeness of other agents. The quadratic function was introduced to make the agent respond to one agent in particular, namely, the closest one. However in the simulation world there occur situations where the closest agents have the same distance to the current agent, and cancel each other out. This happens at the cores of large groups.

6.3.3 Alignment

In our current implemented simulation alignment does not have an effect on the velocity of the agent. The solo behavior of alignment run in the simulation does not have spectacular effects. The agents do not move from their spots and align to the local neighborhood.
In a previous simulation the agents always had a velocity of unity. In this simulation the sole activation of alignment could be witnessed more beautifully. In this simulation the agents point their heads towards the same local heading. The agents move into that direction, and so does their local neighborhood. There are a number of these groups, or more appropriately stated, there are a number of local headings. It takes some time, but in the end the differently orientated groups come in vision range of another group. The collision of these groups result in a new collective heading and the two headings merge into one. In the end all agents walk in *the same* direction and the agents have the same heading.
Alignment as one of the active four behaviors, has the effect we designed it to have. The agents do align to their local neighborhoods. In the dense cores of the clusters alignment is also apparent, seen in simulations as local areas with the same coloring, which represents the heading an agent has.
When the simulation is run, alignment usually does not have a large influence initially. The standardized vector size of the resultant alignment vector in such a case is 0.1. Quite quickly groups are formed and alignment becomes more important. This is also seen in the size of the resultant vector, namely, the

vector grows parallel to a size of about 0.55.

6.3.4 Peeling off

The peeling off behavior is currently implemented to result, when active, in a *random* result vector with a length related to the importance of the peeling off behavior in the current situation. The simulation with the sole behavior of peeling off results in randomly rotating and moving individuals. When observing this simulation peeling off seems to be too active too often. This is misleading because we have implemented the 'random walk' in the peeling off class. When the agent does not see any other agents in its vision range this 'random walk is active'. Even when this 'random walk' influence has been explicitly taken out of the implementation, we can still see a highly active peeling off behavior and again this is misleading. The vector length of peeling off, and therefore the importance of peeling off in these situations, is in all cases minimal. But because of the fact that this behavior is the only influence in this simulation, the behavioral influence is quite obviously seen. The fact that the actual peeling off influence is minimal can be observed in the fact that movement of agents is minimal, only the agent's heading is sufficiently changed to be visible in the simulation.
When peeling off is active in combination with the other four behaviors, the behavioral influence is apparent at the appropriate times. For example, when the wind is blowing over the world from the top to the bottom, 'irratic' or random individual behavior is observed at the top of the clustered groups. All other regions of the cluster are dominated by the other behaviors.
The resultant vector length due to peeling off ranges between lengths of 0.10 and 0.95.

6.3.5 Macroscopic behavior

The goal of the implementation and simulation is to reproduce the macroscopic behavior of the emperor penguin, as depicted in figure 1. The desired collective behavior can be described as a stable non moving core with concurrent layers peeling off along the side in full wind. Additionally the individual agent should be aligned to the macroscopic movement pattern. We can observe that:

- A *cohesive* collective emerges. Once individuals have found a group, the agent will not likely loose the group and wander off.
- A stable non moving core emerges. The center of the resultant cluster has a maximum density of agents and moves minimally.
- A non stable rim of the cluster. At the edges of the cluster the simulation shows much more turbulent individual behavior and agents show a minimal distance in between them an others.
- Highly active agents in full wind. The individuals not standing in lee of others are more active, resulting in a turbulent wind-side of the cluster.

- No 'natural emperor penguin' macroscopic pattern. We have not observed the described macroscopic pattern as depicted in figure 1.

To illustrate the described behavior we have included table 2.
The cohesiveness of the emerging groups can be contributed to the appropriate parameterization of vision range and the grouping behavior. Grouping becomes too active when an agent wanders off. As a result the agent is pulled back before it moves out of vision range.
The emergence of a barely moving and dense core came as a surprise. The repulsive force has been incorporated to *always* guarantee a minimal distance between agents, to give space to move in for an agent. Repulsive force is designed to nullify grouping behavior. But there is no grouping behavior active when surrounded by others. Therefore we expected a minimal distance. However, this is not seen in our simulation. Since natural penguins are reported to have high densities of penguins at the center of the cluster, we did not disapprove of this emerging characteristic.
At the rim of the cluster the repulsive force always comes from the direction of the cluster. In this situation there are no, or at least less, possibility for two repulsive forces to cancel each other out: no opposing repulsive vector of equal size effectively canceling out the repulsive behavior totally. Consequently the agents at the rim do keep a minimal distance.
The agents appear to be highly active when standing in full wind. This can be contributed entirely to the chosen implementation of the peeling off behavior. The resultant vector of this behavior is random and therefore erratic behavior is observed. It was hoped that the agent would be influenced by the other behaviors to a higher degree. The interaction with the alignment behavior and grouping behavior, it was hoped for, would make the peeling off of agents emerge. Then the turbulent rim could only 'bend' towards the sides in combination with grouping behavior, where there would be less chaotic influence from to peeling off activity and agents would become more steady. The turbulent wind-side would have to splice in two, even more so, due to the amplifying alignment force. This, however, is not the case.
The macroscopic pattern observed in the simulation can be described as a stable aligned core, with high activity, meaning movement, towards the edges. There is considerable activity at the windward side of the clusters. This can occasionally result in agents walking into the wind and creating room to move to for the agents towards the more stable core. The core agents move into this space and the whole cluster walks *into the wind*. A second occasionally observed unfortunate variant on this is that a certain alignment can become dominant in the turbulent wind-side. This results in a movement of a large part of the wind-side cluster at the wind-side, 'peeling off' as a whole. However, the agents do not make it to the lee side. These agents gang up at a side of the cluster, and nothing more.

A remarkable observation is the importance of correct parameterization. It is quite time consuming to make adjustments to a certain behavior, because

the other three behaviors have to be adjusted accordingly. What the values of the parameters need to be has to be determined experimentally. It is a relative insightful process because one can determine if a value needs to be increased or lowered. The exact values however cannot be determined beforehand.
We have currently set our behavioral parameters to the values as described in section 5.3. These parameters set to their current values through a process trial and error.

Table 2: Simulation of penguin behavior (107 agents).

Time-step	*groups*	*group size*	*solo agents*
0	-	-	all
40	11	4 to 15	20
80	7	6 to 25	3
120	6	9 to 28	1
160	6	9 to 28	1
200	4	9 to 44	1
240	4	9 to 44	1
280	4	9 to 44	1
320	3	10 to 76	1
360	3	10 to 76	1
400	3	10 to 76	1

6.3.6 Activity and behavioral ordering

The resultant standardized vectors of the individual behaviors represent the importance of the particular behavior in the current situation. The activity levels of the behaviors have all been set to unity, meaning that the importance or length of the behavioral vector is directly related to the activity the particular behavior gets. The total sum of available activity is set to the value of 3. How this exactly works is described elaborately in section 5.3.3.
The ordering of behavior can have an impact on behaviors. We have chosen for the ordering of behaviors in:

1. grouping,
2. repulsive,
3. alignment and
4. peeling off.

The first behavior is most important and designed to be active all the time. The last behavior is least important. As can be seen in the previous sections, 6.3.1, 6.3.2, 6.3.3 and 6.3.4, the individual behaviors often do not result in the

maximum standardized vector length, which is a length of unity. The result of this is that the available activity is often not used up entirely. This has the result that the least important behavior is also always taken into account.
The behaviors of agents are ordered in importance of the behaviors. In the theoretical situation in which all behaviors give maximum standardized vector lengths, behaviors can be excluded from having an influence on agent behavior. The maximum activity has been set to 3. This entails that in the stated theoretical case the first three behaviors are taken into account *always* and the last behavior not. The order in the behaviors for the first three behaviors has no influence on the outcome. This is dissimilar to the stated model. In effect there is only a division between important and less important in our current simulation. We will discuss possible solutions to this problem in section 7.3.2
The usage of activation levels and ordering has not been tested enough for appropriate parameterization. In our simulation the values for activity levels have all been set to unity. In our simulation it has not been observed that a behavior is excluded from having an influence.

6.3.7 Conclusion

The model for our simulation stated a number of desired universal characteristics and stated behaviors for the agents and the expected macroscopic effect. The first desired general characteristic is *no chaotic individual behavior*, as compared to our earlier simulations and implementations. It can be concluded that this problem has been remedied through the introduction of a *heading* for our agents. Agents are restricted in the rotation which can occur per simulation time-step to 45 degrees and the heading of an agent restricts possible positions to move to. This works appropriately.
We have discussed the individual behaviors in the previous sections, and can conclude with regard to the behaviors:

- Grouping: the agents cluster together into coherent and cohesive groups.
- Repulsive: the agents keep a minimal distance to others at the rim area of the clusters. At the center of clusters agents do not keep a minimal distance. This is unexpected, but not undesirable.
  Repulsive force is designed to be a counter force for grouping. It functions accordingly.
- Alignment: the agents do align towards the same local heading. This is however less apparent in the turbulent wind-side of the cluster.
- Peeling off: the agents do not move consistently towards the sides of the cluster. Agents do become highly active at the appropriate time, meaning, when standing in wind. However, this behavior is designed to be the initiating force for the *emergence* of the macroscopic pattern *in combination with the other behaviors.* The current implementation of the peeling off behavior failed to be that initiating force.

During modeling we hypothesized that a stable core of agents would and could emerge without direct behavioral influence on velocity. We expected agents to obstruct each others' movement towards the center of the group and eventually the densities of agents rises to create a stable core. However, this was not the case. During implementation and parameterization it became clear that direct influence on velocity is necessary and adjustments were made. With the introduction of a variable velocity, as opposed to a velocity of unity all the time, and behavioral influence on it, stable cores now do emerge.

6.4 Results and goal definition

In section 2 we stated the *global* goal of our research project:

> The topic of this research project is the modeling and the implementation of swarm behavior. A group of simple agents interact with an environment, and show macroscopic behavior, which the individual parts do not possess. The main goal of this study is to gain insight into 'emergence' of such macroscopic behaviors. Thus, it will become clear how simplistic parts make a whole with desired complex characteristics.

In general we can state that we have been able to create a large sum of the desired required behaviors as stated in our model. This has been summarized in section 6.3.7. There are some additional results which must be stressed though. This is the process of trial and error during implementation, simulation and new parameterization, discussed in section 6.4.1. Another complicating matter concerns the impact of the number of agents involved in the simulation. This is elaborated in section 6.4.2. A final general realization concerning the simulation environment and our developmental process is stated in section 6.4.3.

6.4.1 Parameters

Concerning our results we must point out that the process of parameterizing for models is tedious and time consuming. As time progressed, behaviors were introduced, activation functions and importance functions designed, heading and velocity were incorporated. All these aspects have parameters assigned to them which have to be set. For the behavioral functions the global shape of the function is already known beforehand. During simulation a particular individual behavior is observed, analysed and adjustments are made.
Whenever a new behavior or variable is added, all other set parameters need adjustment. This is a process of changing a parameter, running the simulation for an amount of time, observing the result and adjusting again. This is highly time consuming. For adjustment of a certain parameter, its output values in particular situations must be known. Additional functions need to be created to generate the required output to be able to make correct adjustments.

We have eight parameters determining the importance for every particular behavior in their current situation. The implementer can also adjust the importance of every behavior relative to the other behaviors. This has led to a parameter for total activation and four for every behavior in particular. The basic number of parameters is thus thirteen parameters. The correct balance needs to found for these parameters, which is highly time consuming.

6.4.2 Number of agents

In 'Swarm' [19] it is easy to create a smaller or larger world. The number of agents in the world can be set by use of a chance factor, which sets the chance factor for an agent to be created in a particular world compartment. We have experimented with these factors quite often and became aware of a complication matter.
During simulation and parameter setting we often used small worlds. This is done for the simple reason to speed up the developmental process. These small worlds, for example of size 50×50, consequently have less agents inhabiting them resulting in less computation time. The correct parameter values are determined, resulting in correct behavior in the 'small world'. When we use the larger, 'normal' world the size of 100×100, the same (collective) behavior will not necessarily emerge. We experienced this during fine-tuning for every behavior. To give an example, with the appropriate parameter values groups of forty penguins would emerge in the small world, behaving and moving appropriately, as we intended. The same settings are transported to the large world. In the larger world there are more agents. It is observed that now the clusters can become stuck in a steady state. This is due to the *number of agents* involved in the simulation. Typically the simulation shows correct behavior at the beginning of the simulation. As time progresses clusters collide, exceeding a critical mass, and in effect the clusters are now trapped in a steady state not experienced before. The parameters need to be adjusted some more.
The resultant behavior of the collective is set by parameterizing individual agents. The individuals are influenced by the agents in their vision range. Apparently the limit to the exhibited collective behavior is not determined by the vision range of an individual agent, but also by the vision range of the 'seen agents', whose behavior in turn in influenced by their 'seen agents'. Consequently the *scaling up* of the number of agents is a determining factor for the resultant macroscopic behavior to an even larger extent than expected beforehand.

6.4.3 As real as possible

A main thread through our research has been the ongoing shift towards more and more continuous representations. This is seen in the developmental process and also in the implementation of our agents. Implementations with *discrete* internal representations result in *deterministic behavior*, as seen in the first behavior based simulation (see section 6.1.2). Implementations with no internal

representation result in chaotic behavior, as seen in the particle simulation (see section 6.1.1).
An always present discretizing factor is the simulation world. The simulation world is defined in the used software as a large grid. Take for example the repulsive force experienced by agents. If there are agents located, one to each side of an agent, both result in a repulsive force and so the resultant repulsive force is zero. In such a situation the agent is repulsed heavily, but not responding. The same hold to some degree for all behaviors. We have tried to overcome the difficulty with regard to repulsion by making repulsive force quadratically related to closeness. In the current world this proved ineffective.
The problems arising from the discreteness of the current simulation world can be overcome, of course, by creating agents of considerable size and in doing so making distances in between agents more variable. In practice it proves difficult to make an agent with the size of four compartments, as in our simulations, but it can be done. More severely limiting is the consequent increase in needed vision range and world compartment checking for other agents in vision range. Also, the needed simulation world for these agents is humongous. For all practical purposes this proves impossible in the current simulation environment.
As said before, there has been an ongoing shift towards more and more continuous representations. This is also the case for agents and their internal state. The agents have become much more complex than originally imagined, especially compared to the first particle approach. The deterministic behavior exhibited by the agents has been overcome by usage of analogous values for heading. In the discrete case the heading of an agent always had a direction pointing towards one of the eight possible directions. The difference between continuous valued heading and discrete valued heading is mainly due to the increase in 'memory' an agent has. The situation the agent is located in is stored to some extent in the resultant vector. This vector manipulates the current heading and in doing so stores the situation in the agent's memory. Obviously the discreetly implemented heading has less expressive power to 'represent' situations. The now more expressive heading has more discriminatory ability in different situations, resulting in more realistic individual behavior. We therefore approve and advocate the used implementation of analogous resultant vectors for behaviors, velocity and heading. To conclude: make simulations as continuous as possible on the discrete machines.

7 Discussion and conclusions

7.1 Main goals

As the global goal we have stated in section 2 that we wish to gain insight into the emergence of complex phenomena through modeling, implementation and simulation of simplistic parts. In this section we will reflect on this and conclude if or to what extent we have succeeded.

7.1.1 Emergent phenomena modeling

We have tried to reproduce the behavioral pattern shown by groups of penguins. We have succeeded to model, implement and simulate a lot of designed and desired characteristics. We have been able to simulate a lot of aspects we wanted the simulation to exhibit. This has been elaborately discussed in section 6.3.7. Have we gained knowledge of into the emergence of macroscopic patterns?
The construction and implementation has been a creative process of consecutive models and realized implementations and simulations. Every new simulation answered questions and raised new ones. We observed that the resultant simulation can always be *explained* by the underlaying simplistic rules, however not always predicted. Quite often the implementations and models have been adjusted to remove an undesired macroscopic characteristic in the simulation and often we were successful in the end. However, there is no clear developmental tool kit to approach such phenomena. The key to success lies in choosing *the correct analogy* and *levels of description*, in other words the *level of detail needed*. This is of course the very nature of the kind of research we have chosen. Therefore the research modeling emergent phenomena will always have an experimental character.
We can conclude that the process of design has been, to a large extent, common sensical. The data available for our project has been limited and the chosen behavioral based design did not allow the full usage of the scarcely available data. Still, the successes gained in the course of this project do justify the claim that we have gained knowledge and invaluable experience into the emergence of macroscopic patterns. Limiting factors concerning our project have been the already named limited amount of data concerning the emperor penguin, but most importantly the time consuming construction of implementations and testing and refinement of the resulting simulation.
A final note must be given. Research of emergent phenomena is a highly creative process. The success of a project depends greatly on the ingenuity of the particular researcher. He has to 'see' what the underlying characteristics of the parts are, or he has to guess. Furthermore, it is a highly time consuming process.

7.2 Penguin case study and other case studies

We will compare our used techniques in implementation and simulation with two well known implementations briefly. These two implementations are the

BOIDS implementation and implementation of nest cleaning of ants. This is a general discussion.

7.2.1 BOIDS implementation

The basis for our simulation is made up of four behavioral rules. Three of these behavioral rules are analogous to the BOIDS [24] rules. These rules are named somewhat differently in our model, but are basically similar.
A less refined earlier implementation did not incorporate velocity as a variable value. In this implementation an agent had a constant speed of unity and peeling off was not active, in effect only the BOIDS rules are active. The simulation of this implementation resulted in 'flocking penguins'. The agents move aligned to each other into the direction determined by the local heading. The agents have a minimal distance in between them determined by the repulsive force.
As in the BOIDS simulations, the flocking groups come into contact with other flocking groups. Some groups of agents merge into a larger group. When these groups become sufficiently large, the group breaks into smaller groups and both subgroups move into different directions.
The implementation of the penguin agents is similar to the implementation of the BOIDS. For a large part the same behavioral functions are used. The parameterisation has been done experimentally. The similar behavior, with appropriate parameters set, does not come as a surprise. This was seen as a confirmation that we were on the right track.
To conclude, the implementation of the BOIDS and the penguin agents is similar in design, implementation and results with respect to the flocking behavior. The emergent flocking behavior is reproduced by implementation of three simple rules.

7.2.2 Ants implementation

Ants are collectively capable of a number of things. Among them is the ability to clean up their nest, neatly piling up bodies of dead ants. When we compare the implementation of simulated ants piling up dead bodies to our implementation, there are obvious differences. The simulated ants work on highly localized rules such as:

- 'if I carry a dead body and I see a dead body, then drop this body'
- 'if I do not have a dead body and I see a dead body, then I pick it up'

The second huge difference is that the simulated ants usually walk randomly in the simulation world. The ants simulation works with rules which are roughly taken somewhat similar to cellular automata. Cellular automata base the current state of a certain cell on the local situation, meaning the cells around them. Our simulated penguins compared to the simulated ants have large differences. The most obvious of these is the larger vision range in simulated penguins. This vision range is the basis for behavioral decision making. The area influencing

the individual agent is considerably larger than for the ants. The second major difference is the incorporation of a heading into our penguins for the creation of 'less chaotic' individual behavior. This is opposite from the random behavior used in simulated ants. A third difference is the binary character of the ants agents in this particular case. There is a random walk, creating deterministic behavior for agents but the behavioral rules are *true* or *false*. The simulated penguin agents use behavioral vectors which all influence the behavior of the agents at every simulated time-step.
The approach used in the described ants simulation is similar to our first 'particle' simulation. We have stated the results and the conclusions from this simulation in section 6.1.1. The conclusion we made from our 'particle' approach was that we needed a more behavior based approach. This does not mean that the approach used in the ants simulation is inappropriate for applicability to the simulated penguins. It does mean that the analogy *we* used of heat seeking and producing particles was an inappropriate analogy for the simulated penguins. After all, the basic behavior of peeling off was observed in our 'particle' simulation. However, the specific improvements we wanted to make to individual agents was not allowed in our used model and the model's used analogy.

7.2.3 Conclusion

We have discussed two different implementations simulating emergent behavior or creating a self organising system. The used implementations vary considerably in specifics. Both systems use local situation to determine an individual's actions, however, what exactly is defined as local is rather different. Additionally the representation of situations and the possible actions to be done by an agent varies largely. Why have we used the behavioral approach and not the cellular automata styled approach?
We believe that both approaches are valid to the modeling of penguin behavior. We can imagine a totally different implementation of our penguin agents as cellular automata styled agents. We do not see any restricting characteristics of natural penguins to make this impossible. However, during development certain considerations emerged, namely, concerning vision range and cohesion of groups, which made the behavior based frame a more obvious and logical frame to work with.
In previous sections we discussed the first simulation, the particle approach. This simulation actually used the highly localized 'cellular automata' approach, and did show many of the desired characteristics of the macroscopic pattern. It must be realized that there are likely to be numerous possible implementations to realize the same macroscopically desired behavior. Our developmental course made the behavior based style of implementation the logical path to go along.

7.3 The simulation and implementation

7.3.1 Vectors

In our implementation we have used vector summation to determine the influence of a particular behavior. This does have the negative effect that vectors can cancel each other out. This became apparent in the summation of opposing repulsive forces produced by nearby visible agents. The result of this is that the agent does not respond in a situation in which it does need to respond.
In the case of repulsive force there is a possible solution to this problem. This solution is to use the cross-product of the repulsive vectors. When opposing repulsive forces are experienced, for example from the east and west, this results in a behavioral vector in a direction which is not occupied by other agents, in the example this would be north or south. We have not investigated this possible solution and the effect of it in simulations. In this situation the resultant repulsive vector can be oriented into two direction and a decision has to be made which direction is the obvious choice. A determining factor on which to base this choice could be the heading the agent now has. The agent is more likely to move into the direction determined by its heading.
More generally speaking, is the usage of vectors in determining behavioral influence adequate? We think it is. Simple vector operations have resulted in quite successful simulations, as seen in our research. Furthermore, the toolkit for vector manipulation is large. The expressive power of vectors in the implementation and simulation can be enhanced by adequate and smart usage of vector arithmetic.

7.3.2 Behavioral ordering

As stated in section 6.3.6, the order of behavioral influence in our model, is not apparent in the implementation. The order is not divided in four behaviors from important to less important, but in a division of always active and sometimes active behaviors. A solution to this problem lies in the appropriate setting of the activation values for the individual behaviors and the total available activation. Currently this is not parameterized correctly resulting in a sluggish division in two types of behaviors, namely, always active behaviors which are the first three behaviors and sometimes active behaviors which is the peeling off behavior.
We have experimented with the lowering of overall available activity to make behaviors cancel each other out more often. The result is a new division of behaviors in:

1. Behaviors still always fully active, with a minimum of one behavior. These behaviors are the first behaviors in the order. These behaviors always fall within the bounds set by the maximum available activation.

2. Behaviors which are sometimes active. These behaviors consume the 'left-over' activation in the order stated.

If the maximum available activity is lowered enough the behaviors do have an order of importance. However, in such an implementation often only one or two

behaviors are active and we want behavioral influences from all behaviors to some extent all the time (See section 6.1).
There is no obvious solution to this problem. An option could be to construct a *minimal activation for every behavior* according to the behavioral order, with a limited amount of *additional activation consumed according to the behavioral order*. We do not approve of such a solution, as it will introduce additional parameters which have to be set experimentally. Furthermore, it is not an elegant solution.

7.3.3 Macroscopic pattern

In our final simulation we have not been successful in simulating the desired macroscopic pattern. We have been able to implement a lot of the modeled characteristics, such as cohesion of groups, stable core clustering and a turbulent outer rim of the agent clusters. Individual agents generally do not exhibit chaotic and implausible behavior. However, we have not been able to implement the peeling off behavior successfully, which seems to be the initiating force for emergence of the desired global behavioral pattern. Now we ask ourselves three questions:

1. Are the used behaviors sufficient for a successful simulation?
2. Is the implementational shift towards continuous individual agents the way to go?
3. Is the macroscopic pattern emergent?

We have used four basic agent behaviors. The results we have had so far are encouraging. We, as implementers, feel anxiety when we look at the resultant simulation. It is as if we are climbing a foggy mountain of which we cannot see the peak, the successful simulation. We think the peak is some meters away, however, when we have traveled this distance we still cannot see the peak. In the end we have traveled kilometers and still believe that the peak is just some meters away.
Concerning the used behaviors we do feel that these behaviors are sufficient. When the decision was made to apply the behavior based approach, three behavioral forces were recognized: grouping, alignment and peeling off. Later a forth behavioral force was introduced, namely, repulsion. The core of the model has maintained these behaviors throughout the project since then. Refinements have been made on all aspects of the implementation, and more can be done in the future. We do believe that the core behaviors must be sufficient. These behaviors are the obvious result from the translation of 'observed penguin behavior' to 'simulation behaviors'. There seems to be no other obvious or plausible translation to behaviors possible.
Is the shift towards continuous implementation successful? Due to this continuous shift we have been able to eliminate the deterministic behavior experienced in earlier implementations and have given more discriminatory expressiveness to distinguish different situations. However, the agents used in the simulations

have become more complex and computationally heavy, which goes against the idea of simple agents collectively producing complex behavior. We have emphasized the construction of more realistic individual behavior and this has led to more continuous implementations. The implementer must take responsibility for the course taken. However, it is likely that there are more solutions to construction of the desired pattern than only one, and the 'continuous shift' is only a little part of it.
Our simulation does not reproduce the macroscopic pattern. Is the pattern emergent? Are penguins explicitly or 'consciously' walking according to the global pattern? We cannot answer this question from the available data. It *is* likely that the global pattern is not entirely conscious in the natural penguins, since the penguins also exhibit the pattern in minimal vision conditions such as blizzards. Still we cannot be certain that maybe the penguins are actually aware of the global pattern. The most likely explanation is that it is a combination the two, namely, that the penguins are explicitly peeling off towards the lee-sides and consequently the rotating global pattern emerges. In our implementations we did not implement explicit peeling off in such a way and we did not want to implement it in such a way. We have used a stronger 'emergence' definition concerning this particular pattern and did not want to deviate from it. We feel this would make our fundamental hypothesis crumble, that the behavior *is* emergent. This is a matter of definition.

7.4 AI and Emperor Penguins

What is the relevance of the specific research we conducted concerning Emperor Penguins? This is not directly obvious. The research we have conducted has been an attempt to apply techniques of AI, mainly the behavior based approach, to agents and self organisation to the case study of the Emperor Penguin. If we are capable of simulating aspects of life, this is relevance enough. Then we have created a form of, or an expression of, artificial life.
However, we will discuss the applicability to other fields of our research and techniques used in section 7.4.1. In section 7.4.2 the general philosophical importance of the research is given.

7.4.1 Applicability

The techniques used in our project can be applied to countless other research topics as well. The first type of topic concerns 'simple' social life forms. There are many 'simple' social life forms on our planet interacting with each other. It is likely that these creatures use simple cognitive heuristics in their small brains waiting to be discovered and simulated. There is a lot of research going on in this field. We know of the existence of simulations and models for flocks of birds, schools of fish, route finding of ants, nest cleaning of wasps and ants, herd movement, and many many more. Less obvious application is seen in the modeling and simulation of social structure in groups of apes [13]. Self organisation techniques are also used in modeling of other situations, such as, flow modeling

of humans in crowded situations [12] and flow modeling of cars in traffic.
There are a number of fields which also have phenomena similar to the Emperor Penguin's macroscopic pattern, the second type of topics. The consecutive peeling off of layers in our penguin agents can also be observed, with some imagination, in for example 'two-dimensional particle flow' in fluids[18]. Other fields also have obvious self organising phenomena. The exact macroscopic pattern is different, but techniques of self organisation could still be used. Examples are high and low pressure areas produced by molecules in our atmosphere, galaxy shapes in our universe, the black and white stripes on zebra skins[19], the shape of snowflakes under a microscope.
For the modeling and simulation of our Emperor Penguins we have used a behavior based approach. The first type of topic given, natural living organisms and flow models of parts with typical behaviors, use a similar approach. Our used approach is thus shown to be applicable to these fields. However, we can not use our approach in the examples of inanimate mater, because these parts do not have 'explicit' behaviors. This will remain so, until a researcher comes up with, for example, a correct translation from molecule movement to molecule behavior. It seems key that the to be modeled and simulated, a phenomenon has to have *behaviors* or can be expressed as if the parts had behaviors.

All the described applications in this section have used the same heuristics, namely, modeling of simple agents with certain properties. These agents are let loose in a simulation world. If the observed collective behavior is similar to the behavior observed in 'nature', the implementation is considered to correctly model and implement the 'natural' counterpart in question. However, it is only shown that the used model and implementation is a *possible* realization of the 'natural' macroscopic behavior.
A final note on the applicability to or usability in the field of AI of our specific research is that an application is almost always found after the research has been done. For example, first the 'shortest-route finding' ability was shown in simulated ants, then a different researcher applied this simulated ants ability to 'the traveling sales-man problem' [17] and finally a researcher came with the idea to use this technique in mobile telephones [23].

7.4.2 Philosophical

A goal of our research has been the simulation of an *emergent* phenomena. When studying AI this is an often encountered term. One can say that AI is trying to bridge the gap between humans and machines and show that it is possible to create artificial intelligence just as keen as natural intelligence. In the field of AI there is the always present question if this goal is actually possible. In discussions an often heard topic is 'the emergence of intelligence'. The emergence term is used in discussion of phenomena which are obviously present in humans, such as consciousness and intelligence, but cannot be explained by

[18] website: www.fluid.tue.nl/WDY/vort/ntvn/zelforg.html
[19] website: www.ifi.unizh.ch/ailab/people/thomas/

looking at the brain. A large portion of researchers in the field of AI believe that intelligence and consciousness are emergent phenomena. Also see the illustrative quote on page 11 from Brooks about 'thinking in living systems' and our ability to reconstruct them artificially.
Therefore we believe that research concerning emergent phenomena are of great importance to the field of AI. More importantly, now we look with awe and amazement to the high levels of intelligence seen in (some) human beings. We had the same awe and amazement for the intelligently flocking birds. We have seen how that intelligent behavior can be reproduced by usage of three simple rules. Insights gained from modeling, implementing and simulating emergent phenomena are an inspiring and motivating force, a force capable of pushing AI forward.

8 Recommendations

To conclude we will give additional recommendations for future work to be conducted on our model and implementation.

8.1 Data collection

It would be recommended to take out a reasonable amount of time for the collection of penguin behavior data. Video tapes would be the preferred form for the collected data. In our research we became aware of the existence of such tapes, possessed by the 'Centre d'Ecologie et Physiologie Energetiques[20] research group in Straatsburg. These biologists have made field observations of the Emperor Penguin. At that stage in our research we did not conceive this as a necessity, especially in comparison to the needed investment in time for acquiring the tapes.
A note must be made that the literature available concerning Emperor Penguins is limited. Furthermore, the biologists usually do not stress the macroscopic behavioral pattern of the group as a whole. As a result the collected data's applicability is limited.
To illustrate: In the stage of exploring for a possible research project we came across our Emperor Penguin project. At that time the macroscopic pattern exhibited by the penguins was portrayed quite differently than we later read in the literature. We make this point to illustrate the limited knowledge about the exact pattern exhibited by the Emperor Penguin.

8.2 Future work

For future work on the simulation of the Emperor Penguin we recommend two possibilities, both equally valid. Either

1. the researcher implements penguin behavior in a more continuous world than the currently used world provided by the Swarm software. This will give more freedom and expressiveness for a behavior based approach. Or,

2. the researcher abandons this approach and tries to implement a much more localized and simple model, similar to the simulations of ants. We termed this the 'cellular automata style'.

Still, the currently implemented model can be enhanced and prove sufficient for simulating the Emperor Penguins. The implementational parts which need refinement is stated in section 8.2.1.

[20] Centre d'Ecologie et Physiologie Energetiques
Centre National de la Recherche Scientifique
23 rue Becquerel
67087 Strasbourg Cedex 02 - FRANCE

8.2.1 The implementation

The resultant implementation created is by no means totally finished. The implementation phase of our project has taken quite some time and towards the end time became a pressing matter. We have made a number of different 'peeling off' functions. We give a number of the tried functions:

- Peeling off implemented as a random walk, only active when compartments in wind are not occupied by other agents.
- Peeling off as only active when the agent is actually with its head pointing in wind.
- Peeling off resulting in a resultant vector perpendicular to the wind direction.
- Peeling off as implemented in the current simulation.

The last implementation we used was the most successful. There is a reasonable amount of time needed to correctly parameterize this behavior in the context of the other three. We have not had the time to try all configurations.
The design of peeling off functions must not use explicit behaviors. Take for example the peeling off function resulting in perpendicular movement relative to the wind. This function states a little part of the macroscopic pattern explicitly. This can result in additional functions, all defining a little part of the desired pattern. This is an obvious flaw in design, since we wish to reproduce *emergent* behavior, and do not wish to 'guide' the penguin agent along the desired pattern we wish to produce.
Additional work can be done on the activation functions for individual behaviors and the appropriate ordering of the behaviors. It is however not clear cut if this is actually necessary for the emergence of the desired macroscopic pattern. As can be seen in our discussion of the developmental process, the results for the 'particle approach' concerning the macroscopic pattern were already considerable and peeling off behavior was observed in the simulation. The 'particle approach' and the 'behavior based' approach are totally different in the modeling and implementation of individual agent. However, concerning the global desired pattern exhibited by the Emperor Penguin the results in the simulation are similar.

We believe that the simulation of the macroscopic behavioral pattern of the Emperor Penguin can be realized by implementation of simple agents. Furthermore, we believe that it can be simulated as an *emergent* property. However, we cannot state a path for design and implementational which will guarentee success. Fundamentally different models have been constructed and implemented, and simulation have been run and observed. The resultant simulations have had varying degrees of success concerning individual agent behavior and macroscopic behavior, every implementations with its own advantages and drawbacks. The succes in simulating emergent phenomena is determined by the ability of

the researcher to utilize the knowledge of succesfull implementations in the literature and the determination if this knowledge is relevant for the current research topic. Additionally it must be realized that there are likely to be a number of different individual agent implementations resulting in the same global emergent behavior or pattern.

Summary

The goal of the research project is gaining insight into the emergence of macroscopic behavioral patterns. To accomplish this goal we have studied a specific macroscopic pattern, namely, the 'huddling' behavior of the Emperor penguins. The Emperor Penguin lives in an extreme environment and has adapted appropriately. Individual penguins group together allowing the population as a whole to survive. Characteristic of this pattern is the rotation of positions of penguins, thereby distributing exposure to the cold over all individuals. We have hypothesized that the behavioral pattern of the Emperor Penguin is an emergent phenomenon, and have attempted to model, implement and simulate it as such.
We have constructed subsequent models, implementations and simulations, using the software Swarm. Swarm is a toolkit for simulation of multi-agent systems, allowing us to construct simulations in the Java programming language.

The developmental process can be described as a subsequent revision and adjustment of previous models, implementations and resultant simulations. The first simulation constructed is based on the view of individual penguin agents as heat seekers. A heat world is created by implementation of agents as heat emanating particles. The resultant heat landscape and the penguin agents positions are in constant discrepancy, namely, the heat landscape is positioned more downwind relative to the agents.
The particle approach has led to a new approach to the simulation of the desired macroscopic pattern, namely, a behavior based approach. This new approach first resulted in a simulation which showed deterministic individual behavior and undesired macroscopic phenomena. A refinement of the behavior based implementation is discussed, which is the basis for the final behavior based model, implementation and simulation.

The final implementation is a behavior based one and has four behaviors, namely, grouping, alignment, repulsion and peeling off. The individual agents are implemented with a continuously valued heading and have a velocity or speed. The local situation of an agent results in behavioral vectors, which are all used in the determination of the resultant agent behavior. The size of agents is increased from occupying one compartment to four, as opposed to the previous implementations. This is done for the possibility of 'emergence' of more obstructing dynamics in clusters and is required for a more natural simulation of certain parts of the model.
The resultant simulation shows a stable core cluster with a turbulent rim, with high turbulence at the cluster side in wind. Individual agents behave 'naturally' with respect to quick clustering of agents and do not behave deterministically. The individual agents align to their neighborhood.
Concerning the macroscopic behavioral pattern of the Emperor Penguin we can conclude that the final implementation succeeds to a high degree in the simulation of the modeled key aspect of this behavior. Alignment, repulsion and

grouping result in appropriate macroscopic clusters. The peeling off of agents is however not shown in the simulation, although the collective is highly active at the wind-side. Collective rotation and collective movement down-wind had not been reproduced in the constructed simulations.
We can conclude that we have been successful in the construction of a considerable part of the desired macroscopic pattern shown by Emperor Penguins. Individual implementational aspects can be adjusted and parameterized appropriately and result in simulations with intended emergent phenomena. However, the interaction of behaviors and individuals exhibiting those behaviors and parameterizing these aspects remains highly experimental. When one adopts a behavior based approach one has to use a minimal number of behaviors and parameters for setting these behaviors. The time needed for appropriate parameter setting increases exponentially with the number parameters used. An additional conclusion concerning the modeling and implementation is that the resultant simulation can always be explained, however this does not entail that the result is as it was predicted.
We have simulated the same behavioral repertoire with different numbers of agents in the simulation world. It was expected that the determining factor for resultant macroscopic behavior in the simulation is limited by the number of agents which can possibly be in the vision field of an individual. However, during simulation different macroscopic patterns are observed for different numbers of agents, which both exceed the maximum possible agent in vision range.
The course of the project has involved different approaches with fundamentally different models. We conclude that if one chooses a behavior based approach to the implementation and simulation of the desired macroscopic pattern, one must construct continuous representations whenever possible to avoid deterministic behavior in the simulations. We question the appropriateness of the Swarm toolkit for the implementation of such a behavior based approach.

In the developmental process of the project we have used different analogies and models to accomplish our goal, every model with its successes and failures in resultant simulations. To a large degree the success of a particular endeavor concerning 'emergent phenomena simulation' lies in the ability of the researcher to apply the correct level of description, in other words, appliance of the correct level of detail. We discuss other 'emergent phenomena' implementations in comparison to our implementation and conclude that we can not determine what the appropriate course would be concerning the correct level of description for this particular project.

References

[1] Tucker Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8:209–238, 2000.

[2] John S. Bay. Design of the 'army-ant' cooperative lifting robot. *IEEE Robotics and Automation Magazine*, 2, 1995.

[3] John S. Bay. Behavior learning in large homogeneous populations of robots. In *IASTED International Conference on Artificial Intelligence and Soft Computing*, pages 137–140, 1997.

[4] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*, pages 1–24. Oxford University Press, 1999.

[5] R. A. Brooks. Integrated systems based on behaviors. *SIGART Bulletin*, 2:46–50, 1991.

[6] R. A. Brooks. Intelligence without reason. In *Proceedings of 12th Int. Joint Conf. on Artificial Intelligence, Sydney, Australia, August 1991*, pages 569–595, 1991.

[7] R. A. Brooks. Intelligence without representation. *Artificial Intelligence Journal*, 47:139–159, 1991.

[8] R. A. Brooks. The relationship between matter and life. *Nature*, 409:409–411, 2001.

[9] M Clerc and J. Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6:58–73, 2002.

[10] R.I. Damper. Emergence and levels of description. *International Journal of Systems Science*, 31:811–818, 2000.

[11] Richard Durrett and David Griffeath. Asymptotic behavior of excitable cellular automata. *Experimental Mathematics*, 2:183–208, 1993.

[12] Dirk Helbing, Illes J. Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[13] C. K. Hemelrijk. Dominance interactions, spatial dynamics and emergent reciprocity in a virtual world. *In Proceedings of the fourth international conference on simulation of adaptive behavior, vol. 4 (ed. P. Maes, M. J. Mataric, J-A Meyer, J Pollack & S. W. Wilson)*, pages 545–552, 1996.

[14] Dunin Keplicz and R. Verbrugge. A reconfiguration algorithm for distrubuted problem solving. *Engineering Simulation*, 18:227–246, 2001.

[15] R. Kirkwood. Emperor penguin (aptenodytes fosteri) foraging ecology. Technical Report 144, ANARE Report, 2001.

[16] Kristina Lerman, Aram Galstyan, Alcherio Martinoli, and Auke Jan Ijspeert. A macroscopic analytical model of collaboration in distributed robotic systems. *Artificial Life*, 7:375–393, 2001.

[17] Dorigo M. and L.M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 1997.

[18] J. McCarthy. Recursive functions of symbolic expressions. *CACM*, 3:184–195, 1960.

[19] Nelson Minar, Rogert Burkhart, Chris Langton, and Manor Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. sfi wp " 96-06-042", 1996.

[20] Nils J. Nillson. Shakey the robot. *SRI A.I. Center Technical Note 323*, April, 1984.

[21] J. Préfost. *Actualités scientifiques et industrielles*, chapter Ecologie du manchot empereur Aptenodytes forsteri Grey. Hermann Press. Paris, France, 1961.

[22] J. Préfost and F. Bourliére. Vie sociale et thermorégulation chez le manchot empereur. *Aptenodytes forsteri. Aulauda*, 25:167–173, 1957.

[23] Schoonderwoerd R., Holland O., Bruten J., and Rothkrantz L. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5:169–207, 1996.

[24] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21:25–34, 1987.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Processing*, chapter Learning Internal Representations by Error Propagation, pages 318–364. MIT Press, Cambridge, MA, 1986.

[26] Nicholas J. Savill and Paulien Hogeweg. Competition and dispersal in predator-prey waves. *Th. Pop. Biology*, 56:243–263, 1999.

[27] L. Steels. Towards a theory of emergent functionality. *From Animals to Animats, Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 451–461, 1991.

[28] L. Steels and R. Brooks. *The 'artificial life' route to 'artificial intelligence'. Building situated embodied agents*, chapter Building Agents with Autonomous Behavior Systems. Lawrence Erlbaum Associates, New Haven, 1994.

[29] A. M. Turing. Computing machinery and intelligence. *Mind*, 49:433–460, 1950.

[30] Stephen Young, Penelope J. Watt, James P. Grover, and David Thomas. The unselfish swarm? *J. Animal Ecology*, 63:611–618, 1994.

Program code

File: Penguin.java

```
import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;
import swarm.space.Discrete2dImpl;
import swarm.space.Grid2dImpl;
import swarm.collections.ListImpl;
import swarm.gui.Raster;
import swarm.gui.ZoomRasterImpl;
import swarm.collections.ListImpl;

public class Penguin extends SwarmObjectImpl {
  public int ID;                  public int reporterID = 10;
  public double x;                public double y;
  public double headx;            public double heady;
  public double velocity;
  private ModelSwarm model;
  private Tool tool;
  private Look look;
  public byte pColor;

  private Group group;            private Align align;
  private Repulse repulse;        private Peeloff peeloff;
  public BodyPart BP1;            public BodyPart BP2;
  public BodyPart BP3;            public BodyPart BP4;

  public double wind_x;           public double wind_y;
  public double visionRadius;     public double repulsiveRadius;
  public double numSeen;

  public double max_activity = 3;
  public double available_activity;
  public double used_activity;
  public double max_activity_peeloff   = 1;
  public double max_activity_grouping  = 1;
  public double max_activity_aligning  = 1;
  public double max_activity_repulsive = 1;
  public double f_1_grouping  = 0; public double f_2_grouping  = 6;
  public double f_1_aligning  = 3; public double f_2_aligning  = 0;
  public double f_1_repulsive = 0; public double f_2_repulsive = 0;
  public double f_1_peeloff   = 0; public double f_2_peeloff   = 0;

  public double new_heading_x;     public double new_heading_y;
```

```
public double new_velocity_x;    public double new_velocity_y;

//-----------------------------------------------
// Constructor:
//-----------------------------------------------
public Penguin(Zone aZone, double xPos, double yPos,
                int num, ModelSwarm m) {
  model = m; ID = num;
  tool = model.getTool(); look = new Look(model);
  group = new Group(    f_1_grouping , f_2_grouping,
                        max_activity_grouping , tool);
  align = new Align(    f_1_aligning , f_2_aligning ,
                        max_activity_aligning , tool);
  repulse = new Repulse(f_1_repulsive, f_2_repulsive,
                        max_activity_repulsive, tool);
  peeloff = new Peeloff(f_1_peeloff  , f_2_peeloff  ,
                        max_activity_peeloff  , tool);
  wind_x = model.getWindX(); wind_y = model.getWindY();
  peeloff.setWind(wind_x, wind_y);
  visionRadius = model.getVisionRadius();
  repulsiveRadius = model.getRepulsiveRadius();
  do {
    headx = (model.getTool()).randomDbl(-1,1);
    heady = (model.getTool()).randomDbl(-1,1);
  } while ((headx == 0) && (heady == 0));
  double angle = (model.getTool()).vectToDeg(headx,heady);
  headx = (model.getTool()).x_vectToSize(angle,1);
  heady = (model.getTool()).y_vectToSize(angle,1);
  x = xPos; y = yPos;
  BP1 = new BodyPart(this, (int)X(x - 0.5), (int)Y(y - 0.5));
  BP2 = new BodyPart(this, (int)X(x - 0.5), (int)Y(y + 0.5));
  BP3 = new BodyPart(this, (int)X(x + 0.5), (int)Y(y - 0.5));
  BP4 = new BodyPart(this, (int)X(x + 0.5), (int)Y(y + 0.5));
  (model.getWorld()).putObject$atX$Y(BP1, BP1.getX(), BP1.getY());
  (model.getWorld()).putObject$atX$Y(BP2, BP2.getX(), BP2.getY());
  (model.getWorld()).putObject$atX$Y(BP3, BP3.getX(), BP3.getY());
  (model.getWorld()).putObject$atX$Y(BP4, BP4.getX(), BP4.getY());
}

//-----------------------------------------------
// Movement:
//-----------------------------------------------
private void newVelocity() {
  double angle = tool.vectToDeg(headx, heady);
  double tmpx = (tool.x_vectToSize(angle, velocity) +
                new_velocity_x)/2;
```

```
  double tmpy = (tool.y_vectToSize(angle, velocity) +
                new_velocity_y)/2;
  velocity = tool.vectToAmpl(tmpx, tmpy);
}
private void newHeading() {
  if(tool.vectToAmpl(new_heading_x, new_heading_y) > 0) {
    double angle = tool.vectToDeg(headx,heady,new_heading_x,
                                  new_heading_y);
    if(angle > 45) { angle = 45; }
    if(angle < -45) { angle = -45; }
    angle = angle + tool.vectToDeg(headx, heady);
    headx = tool.x_vectToSize(angle,1);
    heady = tool.y_vectToSize(angle,1);
  }
}
private boolean move() {
  if(tool.randomDbl(0,1) < velocity) { return true; }
  else { return false; }
}
private void leftOrRight() {
  int left = 0; int right;
  double angle = tool.vectToDeg(headx,heady);
  for(int i = -180;i < angle; i += 45) { left = i; }
  right = left + 45;
  if(tool.randomDbl(0,1) > ((right - angle)/45)) { go(right); }
  else { go(left); }
}
private void go(int angle) {
  Grid2dImpl world = model.getWorld();
  int add_x = 0; int add_y = 0;
  if((angle == 180)||(angle == -180)) { add_x = -1; add_y =  0; }
  else if(angle == -135) { add_x = -1; add_y = -1; }
  else if(angle ==  -90) { add_x =  0; add_y = -1; }
  else if(angle ==  -45) { add_x =  1; add_y = -1; }
  else if(angle ==    0) { add_x =  1; add_y =  0; }
  else if(angle ==   45) { add_x =  1; add_y =  1; }
  else if(angle ==   90) { add_x =  0; add_y =  1; }
  else if(angle ==  135) { add_x = -1; add_y =  1; }
  else { System.out.println("no usable angle"); }
  BodyPart tmp;
  if(((tmp = (BodyPart)world.getObjectAtX$Y(
     (int)X(BP1.x + add_x),(int)Y(BP1.y + add_y))) != null) &&
     (tmp.getPenguin() != this)) {  }
  else if(((tmp = (BodyPart)world.getObjectAtX$Y(
          (int)X(BP2.x + add_x), (int)Y(BP2.y + add_y))) != null)
          && (tmp.getPenguin() != this)) {  }
```

```
  else if(((tmp = (BodyPart)world.getObjectAtX$Y(
          (int)X(BP3.x + add_x), (int)Y(BP3.y + add_y))) != null)
          && (tmp.getPenguin() != this)) {  }
  else if(((tmp = (BodyPart)world.getObjectAtX$Y(
          (int)X(BP4.x + add_x), (int)Y(BP4.y + add_y))) != null)
          && (tmp.getPenguin() != this)) {  }
  else {
    world.putObject$atX$Y(null, BP1.x, BP1.y);
    world.putObject$atX$Y(null, BP2.x, BP2.y);
    world.putObject$atX$Y(null, BP3.x, BP3.y);
    world.putObject$atX$Y(null, BP4.x, BP4.y);
    BP1.x = (int)X(BP1.x + add_x); BP1.y = (int)Y(BP1.y + add_y);
    BP2.x = (int)X(BP2.x + add_x); BP2.y = (int)Y(BP2.y + add_y);
    BP3.x = (int)X(BP3.x + add_x); BP3.y = (int)Y(BP3.y + add_y);
    BP4.x = (int)X(BP4.x + add_x); BP4.y = (int)Y(BP4.y + add_y);
    world.putObject$atX$Y(BP1, BP1.x, BP1.y);
    world.putObject$atX$Y(BP2, BP2.x, BP2.y);
    world.putObject$atX$Y(BP3, BP3.x, BP3.y);
    world.putObject$atX$Y(BP4, BP4.x, BP4.y);
    x = X(x + add_x); y = Y(y + add_y);
  }
}

//-----------------------------------------------
// Bound checking:
//-----------------------------------------------
private double X(double x) {
  while(x < 0) { x+= model.getXSize(); }
  while(x >= model.getXSize()) { x-= model.getXSize(); }
  return x;
}
private double Y(double y) {
  while(y < 0) { y+= model.getYSize(); }
  while(y >= model.getYSize()) { y-= model.getYSize(); }
  return y;
}

//-----------------------------------------------
// Getting needed base variables for further calculations:
//-----------------------------------------------
public double deltaX(Penguin other) {
  double delta = other.getX() - x;
  if(delta < (0.5 * model.getXSize())) { delta += model.getXSize(); }
  if(delta > (0.5 * model.getXSize())) { delta -= model.getXSize(); }
  return delta;
}
```

```
public double deltaY(Penguin other) {
  double delta = other.getY() - y;
  if(delta < (0.5 * model.getYSize())) { delta += model.getYSize(); }
  if(delta > (0.5 * model.getYSize())) { delta -= model.getYSize(); }
  return delta;
}

//-------------------------------------------------------
// calculation of variables.
//-------------------------------------------------------
private void resetVar() {
  group.reset();
  align.reset();
  repulse.reset();
  peeloff.reset();
  new_velocity_x = 0; new_velocity_y = 0;
  new_heading_x = 0;  new_heading_y = 0;
  numSeen = 0;
}
private double getActivity(double x, double y) {
  double length = 0;
  if(available_activity > 0) {
    length = tool.vectToAmpl(x,y);
    if(available_activity < length) {
      length = available_activity;
    }
    available_activity -= length;
  }
  return length;
}
private void resultVelocity(double x, double y,
                            double velocity) {
  double angle = tool.vectToDeg(x,y);
  new_velocity_x += tool.x_vectToSize(angle, velocity);
  new_velocity_y += tool.y_vectToSize(angle, velocity);
}
private void resultHeading(double x, double y, double activity) {
  if(activity > 0) {
    double angle = tool.vectToDeg(x,y);
    new_heading_x += tool.x_vectToSize(angle, activity);
    new_heading_y += tool.y_vectToSize(angle, activity);
  }
}
private void addVar(Penguin other) {
  numSeen++;
  double deltax = deltaX(other); double deltay = deltaY(other);
```

```
  group.add(deltax, deltay);
  align.add(deltax, deltay, other.headx, other.heady,
            other.velocity);
  repulse.add(deltax, deltay); peeloff.add(deltax, deltay);
}
private void calcVar() {
  group.calc();   align.calc();
  repulse.calc(); peeloff.calc();
  available_activity = max_activity;
  used_activity = 0;
  if(group.anySeen()) {
    double activity = getActivity(group.getX(), group.getY());
    resultHeading(group.getX(), group.getY(), activity);
    resultVelocity(group.getX(), group.getY(), group.getV());
  }
  if(align.anySeen()) {
    double activity = getActivity(align.getX(), align.getY());
    resultHeading(align.getX(), align.getY(), activity);
    resultVelocity(align.getX(), align.getY(), align.getV());
  }
  if(repulse.anySeen()) {
    double activity = getActivity(repulse.getX(), repulse.getY());
    resultHeading(repulse.getX(), repulse.getY(), activity);
    resultVelocity(repulse.getX(), repulse.getY(), repulse.getV());
  }
  double activity = getActivity(peeloff.getX(), peeloff.getY());
  resultHeading(peeloff.getX(), peeloff.getY(), activity);
  resultVelocity(peeloff.getX(), peeloff.getY(), peeloff.getV());
}

//-----------------------------------------------
// Main routine:
//-----------------------------------------------
public void walk() {
  Penguin other;
  resetVar();
  while((other = look.next(this)) != null) {
    addVar(other);
  }
  calcVar();
  newVelocity();
  newHeading();
  if(move()) { leftOrRight(); };
}
```

```
  //-----------------------------------------------
  // Debugging functions.
  //-----------------------------------------------
  public void setPColor(byte i) {
    pColor = (byte) i;
  }
  public Object drawSelfOn (Raster r) {
    if(ID == reporterID) { pColor = (byte) 9; }
    else {
      pColor = (byte)Math.round((8 * tool.vectToDeg(headx,heady) /
                                360) + 4.5) ;
    }
    r.drawPointX$Y$Color (BP1.getX(), BP1.getY(), pColor);
    r.drawPointX$Y$Color (BP2.getX(), BP2.getY(), pColor);
    r.drawPointX$Y$Color (BP3.getX(), BP3.getY(), pColor);
    r.drawPointX$Y$Color (BP4.getX(), BP4.getY(), pColor);
    return this;
  }
  public byte getPColor() {
    return pColor;
  }
  public int getID() { return ID; }
  public double getX() { return x; }
  public double getY() { return y; }
  public double getHeadX() { return headx; }
  public double getHeadY() { return heady; }
  public double getWindX() { return wind_x; }
  public double getWindY() { return wind_y; }
  public double getVisionRadius() { return visionRadius; }
  public double getRepulsiveRadius() { return repulsiveRadius; }
  public double getTotalSeen() { return numSeen; }
  public double getMaxActivity() { return max_activity; }
  public double getResHeadX() { return new_heading_x; }
  public double getResHeadY() { return new_heading_y; }
}
```

File: BodyPart.java

```
import swarm.defobj.Zone;
import java.lang.reflect.Array;
import swarm.space.Grid2dImpl;
import swarm.gui.Raster;

public class BodyPart {
  public int x;
  public int y;
```

```
  public Penguin p;

  public BodyPart(Penguin penguin, int i, int j) {
    x = i; y = j; p = penguin;
  }
  public Penguin getPenguin() { return p; }
  public int getX() { return x; }
  public int getY() { return y; }
}
```

File: Tool.java

```
public class Tool {
  public Tool() { }
  // returns the theta component of the point (r, theta)
  // in polar coordinates that corresponds to the
  // point (x, y) in Cartesian coordinates.
  public double vectToPolar(double x, double y) {
    return Math.atan2((double)y ,(double)x);
  }
  // calculate angle of vector.
  public double vectToDeg(double x, double y) {
    return ( vectToPolar(x,y) * 360 / (2 * Math.PI) );
  }
  // Calculate angle between two vectors. Used for alignment.
  // (x,y) == heading of agent.
  // (i,j) == distance away of seen-agent, relative to agent.
  // return the angle to turn to.
  public double vectToDeg(double x, double y, double i, double j) {
    double tmp = ( (vectToPolar(i,j) - vectToPolar(x,y)) * 360 /
                   (2 * Math.PI) ) ;
    if(tmp < -180) { tmp += 360; }
    if(tmp > 180) { tmp -= 360; }
    return tmp;
  }
  // Calculate the length or amplitude of a vector
  public double vectToAmpl(double x, double y) {
    return( Math.sqrt( x*x + y*y ) );
  }

  // ------------------------------------------------
  // Resize vectors to a certain length.
  // ------------------------------------------------
  public double x_vectToSize(double angle, double size) {
    return( Math.cos((angle / 360) * 2 * Math.PI) * size );
  }
```

```
  public double y_vectToSize(double angle, double size) {
    return( Math.sin((angle / 360) * 2 * Math.PI) * size );
  }
  // ----------------------- random numbers -------------
  public int randomInt(int min, int max) {
    return (int)Math.round(((max - min) * Math.random()) + min);
  }
  public double randomDbl(double min, double max) {
    return (((max - min) * Math.random()) + min);
  }
}
```

File: Group.java

```
public class Group {
  private double x;        private double y;
  private double velocity; private int numSeen;
  private double f_1;      private double f_2;
  private double max_vect; private Tool tool;

  public Group (double f1, double f2, double activity, Tool t) {
    f_1 = f1; f_2 = f2; max_vect = activity; tool = t;
  }
  public void reset() {
    x = 0; y = 0; velocity = 1; numSeen = 0;
  }
  public void add(double deltax, double deltay) {
    numSeen++; x += deltax; y += deltay;
  }
  public void calc() {
    if((numSeen == 0) || ((x == 0) && (y == 0))) {
      x = 0.0001; y = 0.0001;
    }
    else {
      x = x / numSeen; y = y / numSeen;
      velocity = tool.vectToAmpl(x,y);
      double a = 1 / (f_2 - f_1); double b = (-a * f_1);
      velocity = Math.max(0, Math.min(1, ((velocity * a)
                                          + b)));
    }
  }
  public double getX() {
    double angle = tool.vectToDeg(x,y);
    return tool.x_vectToSize(angle, velocity*max_vect);
  }
  public double getY() {
```

```
    double angle = tool.vectToDeg(x,y);
    return tool.y_vectToSize(angle, velocity*max_vect);
  }
  public double getV() { return (velocity * max_vect); }
  public boolean anySeen() { return (numSeen != 0); }
}
```

File: Align.java

```
public class Align {
  private double x;          private double y;
  private double velocity_x; private double velocity_y;
  private int numSeen;       private int vision = 7;
  private double f_1;        private double f_2;
  private double max_vect;   private Tool tool;

  public Align (double f1, double f2, double activity, Tool t) {
    f_1 = f1; f_2 = f2; max_vect = activity; tool = t;
  }
  public void reset() {
    x = 0; y = 0; velocity_x = 0; velocity_y = 0; numSeen = 0;
  }
  public void add(double deltax, double deltay, double head_x,
                  double head_y, double velocity) {
    numSeen++;
    double a = 1 / (f_1 - vision); double b = -a * vision;
    double importance = Math.min(1, (tool.vectToAmpl(deltax,
                        deltay) * a + b));
    double angle = tool.vectToDeg(head_x, head_y);
    x += tool.x_vectToSize(angle, importance);
    y += tool.y_vectToSize(angle, importance);
    velocity_x += tool.x_vectToSize(angle, velocity*importance);
    velocity_y += tool.y_vectToSize(angle, velocity*importance);
  }
  public void calc() {
    if((numSeen == 0) || ((x == 0) && (y == 0))) {
      x = 0.0001; y = 0.0001;
    }
    else {
      x = (x / numSeen) * max_vect;
      y = (y / numSeen) * max_vect;
      velocity_x = velocity_x / numSeen;
      velocity_y = velocity_y / numSeen;
    }
  }
  public double getX() { return x; }
```

```
  public double getY() { return y; }
  public double getV() {
    return tool.vectToAmpl(velocity_x,velocity_y);
  }
  public boolean anySeen() { return (numSeen != 0); }
}
```

File: Repulse.java

```
public class Repulse {
  private double x;            private double y;
  private double velocity;     private double sumImportance;
  private double vision = 4.5;
  private double f_1;          private double f_2;
  private double max_vect;     private Tool tool;

  public Repulse (double f1, double f2, double activity, Tool t) {
    f_1 = f1; f_2 = f2; max_vect = activity; tool = t;
  }
  public void reset() {
    x = 0; y = 0; velocity = 0; sumImportance = 0;
  }
  public void add(double deltax, double deltay) {
    if(tool.vectToAmpl(deltax,deltay) <= vision) {
      double importance = tool.vectToAmpl(deltax,deltay);
      importance = ( 2 * (importance - vision)) / vision;
      importance = importance * importance;
      double angle = tool.vectToDeg(deltax, deltay);
      sumImportance += importance;
      x -= tool.x_vectToSize(angle,importance);
      y -= tool.y_vectToSize(angle,importance);
    }
  }
  public void calc() {
    if((sumImportance == 0) || ((x == 0) && (y == 0))) {
      x = 0.0001; y = 0.0001;
    }
    else {
      sumImportance = Math.max(1, sumImportance);
      x = (x / sumImportance) * max_vect;
      y = (y / sumImportance) * max_vect;
    }
  }
  public double getX() { return x; }
  public double getY() { return y; }
  public double getV() { return tool.vectToAmpl(x,y); }
```

```
  public boolean anySeen() { return (sumImportance != 0); }
}
```

File: Peeloff.java

```
public class Peeloff {
  private double x;        private double y;
  private double velocity; private int numSeen;
  private double wind_x;   private double wind_y;
  private double f_1;      private double f_2;
  private double max_vect; private Tool tool;
  private double importance = 0;
  private double importance1 = 0;
  private double importance2 = 0;

  public Peeloff (double f1, double f2, double max_activity, Tool t) {
    f_1 = f1; f_2 = f2; max_vect = max_activity; tool = t;
  }
  public void reset() {
    x = 0; y = 0; numSeen = 0;
    importance = 0; importance1 = 0; importance2 = 0;
  }
  public void setWind(double wx, double wy) {
    wind_x = wx; wind_y = wy;
  }
  public void add(double deltax, double deltay) {
    numSeen++; x += deltax; y += deltay;
  }
  public void calc() {
    if(numSeen <= 0) {
      double angle = tool.randomDbl(-180,180);
      x = tool.x_vectToSize(angle, max_vect);
      y = tool.y_vectToSize(angle, max_vect);
    }
    else {
      x = x / numSeen; y = y / numSeen;
      if((x == 0) && (y == 0)) { x = 0.001; y = 0.001; }
      double angle = Math.min(120, Math.abs(tool.vectToDeg(x,y,
                     wind_x, wind_y)));
      importance1 = Math.abs(angle - 120) / 120;
      double length = tool.vectToAmpl(x,y);
      if(length <= 2) { importance2 = length / 2; }
      else if(length <= 3) { importance2 = 1; }
      else { importance2 = Math.max(0, 2.5 - (length / 2));}
      importance = importance1 * importance2;
      angle = tool.randomDbl(-180,180);
```

```
      x = tool.x_vectToSize(angle, Math.max(0.001, importance*
                                                    max_vect));
      y = tool.y_vectToSize(angle, Math.max(0.001, importance*
                                                    max_vect));
    }
  }
  public double getX() { return x; }
  public double getY() { return y; }
  public double getV() { return tool.vectToAmpl(x,y); }
  public boolean anySeen() { return (numSeen != 0); }
}
```