Lints, T. (2008). FlockHeadz: Virtual flock in a room used as a controller. In *IEEE Swarm Intelligence Symposium, 2008. SIS 2008*. IEEE. 5 pages.

T. Lints, "FlockHeadz: Virtual flock in a room used as a controller," in *IEEE Swarm Intelligence Symposium, 2008. SIS 2008*, IEEE, 2008. 5 pages.

```
@inproceedings{Lints08_Flock,
  author = {Taivo Lints},
  title = {Flock{H}eadz: Virtual Flock in a Room Used as a
Controller},
  year = {2008},
  booktitle = {IEEE Swarm Intelligence Symposium, 2008. SIS
2008},
  publisher = {IEEE},
  note = {5 pages}
}
```

2008 IEEE Swarm Intelligence Symposium
St. Louis MO USA, September 21-23, 2008

FlockHeadz: Virtual Flock in a Room Used as a Controller

Taivo Lints

Abstract—**The paper calls for further innovations in the field of Swarm Intelligence, and presents an idea of using virtual flock(s) in a virtual room as a controller. Then, to demonstrate the viability of the idea, a working prototype is constructed where the controller is used to guide a "creature" through a nontrivial corridor.**

I. Introduction

SWARM Intelligence (SI) is a branch of Artificial Intelligence and / or Artificial Life where the main source of inspiration is the behavior of flocks of birds, swarms of insects, shoals of fish, herds of animals. One research direction of SI has the goal of making artificial objects behave as a swarm. Examples of this direction are swarms of cooperating robots, and animated flocks of birds or herds of animals in computer games and movies. Another direction of SI has a more abstract approach: using an artificial swarm as a problem solver, without trying to implement the swarm in hardware or even without visualizing it on a screen (except maybe for debugging or educational purposes). The main examples of this direction are Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO).

ACO, inspired by the behavior of ant colonies, is a stochastic optimization algorithm where the optimization problem is transformed into the problem of finding the best path on a weighed graph, and a set of software agents incrementally builds solutions by moving on this graph [1].

In PSO, a swarm of virtual particles is moving around in the search space, cooperatively trying to find the point(s) where the function being optimized (fitness function) has most suitable values (as defined by given optimization problem).

Both ACO and PSO are also applicable in dynamically changing situations. In case of ACO, the typical application example is routing and load balancing in changing networks (see, e.g., [2]). The practical use of dynamical PSO's is apparently less common, but at least the theoretical work is in progress ([3] and [4] being just a few of the many examples).

However, there seems to be a lot of untapped potential in this more abstract direction of SI. Apart from ACO and PSO, there are not many remarkable developments, and even these two concentrate mostly on industrial optimization problems (surely there are several other well-established related methods like Evolutionary Algorithms and Stochastic Diffusion Search [5], but these do *not* draw inspiration directly from typical spatially moving swarms and thus would be more accurately described as *population*-based methods). While it is perfectly reasonable to put a considerable effort on making ACO and PSO increasingly useful in highly practical situations, it might be a good idea to simultaneously keep the more open-ended innovation process more active. To facilitate and inspire that process, this paper proposes a (to my knowledge) new method of Swarm Intelligence.

Manuscript received May 11, 2008. This work was supported in part by Research Laboratory for Proactive Technologies in Tallinn University of Technology, Department of Computer Control in Tallinn University of Technology, Estonian Information Technology Foundation, Estonian Doctoral School in ICT, Estonian Ministry of Education and Research (grants SF0142509s03 and SF0140113As08), and Estonian Science Foundation (grant ETF6182).

Taivo Lints is with the Research Laboratory for Proactive Technologies, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia (phone: +372 56 625 777; fax: +372 6202 101; e-mail: taivo@taivo.net).

II. The General Idea

The idea is to take an artificial swarm, put it in a virtual room, create input and output mappings of necessary parameters to and from the room, and use this system as a controller (Fig. 1).

First of all, there has to be a swarm. In principle, it can be almost any kind of swarm. But to get started, we can use the quite well-known basic rules of boids from [6]. Each boid (*bird-oid*) has three simple simultaneously active steering behaviors:

1) *separation* – boid steers clear from local flockmates to avoid crowding;
2) *alignment* – boid steers towards the average heading of local flockmates trying to maintain the general moving direction of the flock;
3) *cohesion* – boid steers to move towards the center of the local subflock (consisting of boid's local flockmates) to avoid leaving the flock.

Secondly, the swarm is put into a closed virtual room. The boids can be made to actively avoid the walls, or just to bounce back from, or slam into, them. For simplicity, we take a circular room.

For this system to act as a controller (not necessarily in an industrial sense), it has to be supplied with input and output channels. Thus, the third step is creating the means of interfacing with the flock.

The boids should be given some sensors, say "light sensors" (being a virtual flock it really does not matter how the sensors are called, but currently this naming makes it easier for a human to get hold of the idea). In the room there should be sources of information corresponding to the sensors ("lights" in this case). Some additional behaviors are then added to boids, so that they will react to the sensed information.

The input signals to the controller are mapped to information sources in the virtual room. For example the input signals can affect the position, intensity, color, etc. of the

978-1-4244-2705-5/08/$25.00 ©2008 IEEE

"lights". Outputs of the controller are formed by monitoring the parameters of the flock – the position and speed of flock center, flock density, etc. – and combining / processing these parameters to match the requirements of the controlled object.

To make the point clearer, here is a concrete simple example of a possible use of the system, in the form of a thought experiment (i.e., not implemented). Imagine we have a virtual creature in a computer game, who has to avoid other "bad" creatures (as shown on Fig. 1). As a "brain" our creature has the previously described system of a flock in room (hence the name of this project: FlockHeadz). The creature is able to detect the direction and distance (or maliciousness) of enemies, and this information is relayed to the controller where each enemy is represented as a light source on the wall of the virtual room. The closer or more malicious the "bad" creature, the more intensive the light. Direction of the enemy, relative to our creature's heading, is mapped directly to the direction of light source from room center (e.g., an enemy who is in front of the creature is represented by a light source on the Northern wall of the room; an enemy behind the creature by a light source on the Southern wall, etc.). Boids in the flock are added a rule which makes them move away from light sources with a speed depending on light intensity. The output of the controller is derived from flock center's position relative to room center. E.g., if the flock is in the Northern / upper part of the virtual room, our creature will move forward with a speed depending on how far the flock is from the room center. If the flock is in the Eastern / right part of the virtual room, our creature will move sidewise right (or maybe turn right), etc.

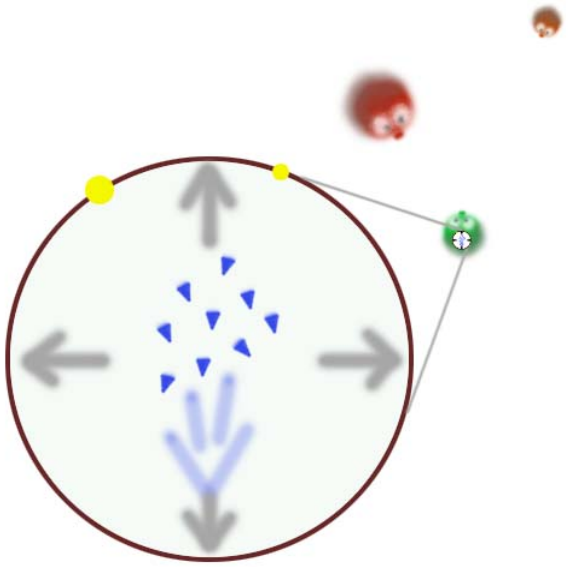

Fig. 1. Simple flock-based control system for avoiding "enemies". In the upper right quarter is shown a situation where our controllable creature (facing upwards) is confronted by two bad creatures (facing downwards). Big circle on the left is a virtual room that acts as the "brain" of our creature (i.e., it can be thought of as situated inside the creature's head, not next to it as shown here for visualization purposes). Inside the room there is a flock of boids (small triangles). At the walls of the room there are two "lights" (circles) that represent enemies. Boids move away from lights (big sketched arrow downwards). When flock reaches bottom area of the circle, the controlled creature will start moving backwards (four smaller grey arrows represent the mapping between flock position and control signal sent to the creature).

While the given example might be helpful for understanding the basic idea of FlockHeadz, it does not seem to be very practical in its extreme simplicity. Consider the case where a few enemies have surrounded our creature. The flock will most likely move to the center of the virtual room, and our creature will stay still, without trying to escape (though, in certain cases it might even be a good strategy). A notably more serious problem is: boids in the controller are explicitly designed to behave analogously to the expected behavior of controlled creature – they just move away from light sources that represent enemies. It raises the question: why not apply the same rules directly to our creature and avoid the computational cost and complexity of the flock-based controller? In such a simple example the use of the flock-based controller might indeed not make much difference, maybe only adds a small element of unexpectedness and suprise (from the viewpoint of human observer) to the movement of the creature. This is not to play down the value of unexpectedness – unlike in common engineering, surprises have a very high value in Artificial Life and Artificial Intelligence – it is just that the magnitude of surprise here is somewhat small, especially compared to the added development and computational costs.

Getting more interesting, more complex and more useful behavior requires, in most cases, fine-tuning the myriad of parameters already implicitly present in given system, choosing suitable rules for boids, suitable mappings for controller inputs and outputs, etc. This is a task not suitable for human designers. Thus a reasonable approach would be to construct a test environment relevant to the problem in hand (or, if possible and reasonable, use the "real" problem environment itself), and apply artificial evolution to the system. In the aforementioned simplistic example, the fitness of the creature could be defined as surviving time on a flat surface populated with enemies, or the speed of getting from one point to another on that surface without being eliminated in between.

More complex tasks may require the addition of new features to the controller. Room shape can be made more complex, and a third dimension (or even more) can be added. Room can be made to contain a (possibly nonhomogeneous) internal environment that affects boids' capabilities. There could be several species of boids, each species equipped with different sensors and possibly being affected by the movements of other species. Also, the possible ways of how controller's inputs are made to affect boids are nearly endless, from directly influencing boids' parameters, to all kinds of mappings of inputs to information sources in room that are sensed by boids through advanced sensors. But of course the addition of new features causes huge expansion of evolution's search space, which in many cases makes the finding of useful solutions a lot harder, and thus practical considerations may suggest keeping the controller relatively simple.

III. A Simple Prototype

To test whether the idea of using a flock in a virtual room as a controller is at all feasible, I made a simple prototype. As deciding whether a creature is successful in avoiding enemies is somewhat arbitrary, the test case is taken to be the passage of a nontrivial corridor, where success is quite clearly defined as getting from one end of the corridor to another as fast as possible. Please note, however, that the main contribution of this paper is the *idea* of FlockHeadz and, more generally, the facilitation of innovation in the field of Swarm Intelligence. The controller presented in this section is only a *demonstration* – a very primitive instantiation of the much more general and complex FlockHeadz idea – and as such does not even attempt to match the performance of proper navigational algorithms, neither in accuracy nor in applicability to previously unencountered environments. Developing practically useful controllers that behave well in large sets of diverse environments is a direction to pursue in future research.

The corridor to be passed is shown on Fig. 2. The "creature" (just a dot on the screen) starts from the lower central part and has to finally reach the far right end of the corridor. The creature is able to sense the closest points of each of corridor's wall (including those out of sight). This ability is obviously not realistic, but the test case is not intended to be a simulation of a real robot. Rather, it is an artificial object in an artificial environment, where it happens to have an easy access to a function that returns the positions of such points (because I happened to have at hand a fast software implementation of given ability, and developing a fast realistic one, compatible with the used system, would have delayed the implementation of this prototype). However unrealistic, for our current purposes the test case is perfectly suitable, as it is not trivially solvable by a random controller.

The prototype is implemented in Python programming language. For human input handling and visualization, packages pygame and PyOpenGL are used. Controller, controllable creature, and management console are all running as separate processes, communicating mostly by asynchronous message passing through Spread daemon (www.spread.org). The main motivation for using message passing was to allow the prototype to be later easily extended and distributed over several computers for larger evolution runs, and to make it easier to visualize controller and controllable system in separate windows. However, for this simple test case the distribution was not necessary – one computer provided enough computing power. Also, the separation of controller and controllable system processes by asynchronous messaging makes system behavior dependent on nonessential variables like the relative running speed of the processes and the general state of operating system environment (e.g., visualization of the controller may be computationally more expensive than visualization of the controlled system, thus switching on the visualization causes a larger slowdown of the controller process compared to the slowdown of controllable system, which changes system behavior). Therefore in most cases it might be a good idea to avoid such separation.

Visualization of the controller is shown and explained on Fig. 3. Input to the controller is the aforementioned closest

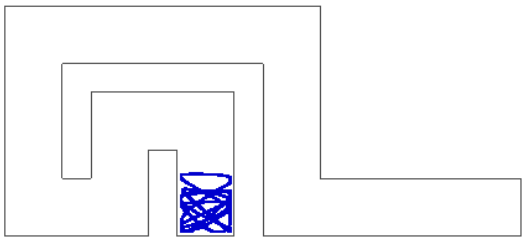

Fig. 2. Traced path of the creature moving in a corridor during an average run with randomly parameterized flock-based controller.

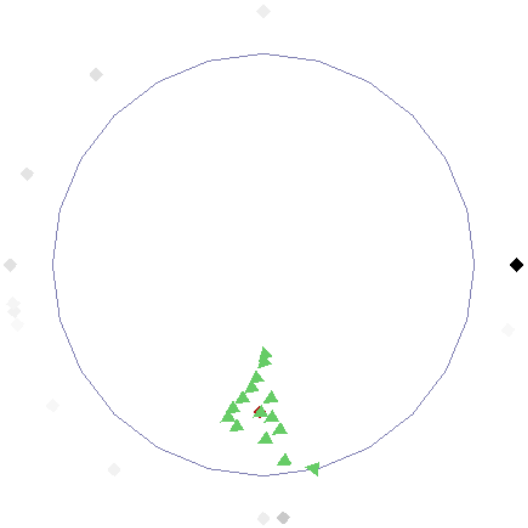

Fig. 3. Visualization of the flock-based controller. Triangles inside the room are boids. Black and grey dots around the circle are information sources that represent nearest points of every wall in the corridor, their intensity depending on how near those corridor points are (see Fig. 4).

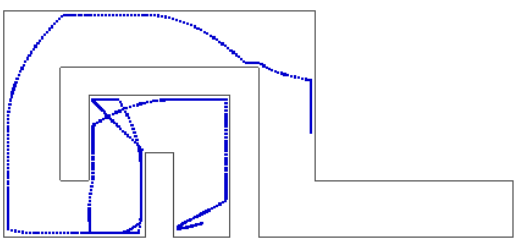

Fig. 4. Traced path of the creature moving in a corridor during a relatively good run with well-evolved controller. Current position of the creature is in the rightmost end of the path, moving down next to a wall. The controller state of this moment is shown on Fig. 3.

points of all walls, represented as "lights" near controller room's walls. Boids can, depending on their parameters, flee or chase the lights (currently all boids have same rules and parameter values, so the situation of some boids avoiding and some chasing the lights is currently not possible). Output of the controller is derived from flock center's position (i.e., the average position of all boids) relative to controller room's center. This output is mapped into controlled creature's acceleration – the further North the flock is, the faster the creature accelerates towards North, etc. (if the creature was moving Southwards beforehand, then Northwards acceleration is synonymous with deceleration for a while).

The main controller-related changeable parameters in given test case are: the number of boids; the neighborhood radiuses around the boid for separation, alignment, and cohesion rules (all three possibly different); the weights of separation, alignment, and cohesion rules (how much each of them affects boid's movement; can be both positive and negative, the latter inversing the behavior, e.g. separation to cohesion or alignment to reverse-alignment); the weight of "light" avoidance rule (also either positive or negative); boid's speed decay factor (basically similar to friction); and output scaling factor, i.e. how strong is the acceleration applied to controlled creature.

First I tested how successful the creature is with controllers that have parameters randomly drawn from reasonable ranges (reasonable in the sense of "not too large to make the system totally dysfunctional, based on system designer's educated guesses"). This is kind of a null hypothesis test, to make sure the desired corridor passing behavior is not human-designed into the controller structure so that most parameter values would give expected behavior. The results demonstrated that creatures with randomly parameterized flock-based controllers tend to stay in the first segment(s) of the corridor for a long time, either crashing into walls and corners or wobbling around (Fig. 2). When parameters were hand-tuned, the system performed considerably better, in some cases even reaching the other end of the corridor. However, in most runs the creature still spent a lot of time moving back and forth in a few corridor segments. Thus, it is not trivial to find parameter values that would make the flock-based controller solve the test case efficiently. At the same time the acceptable solution regions in parameter space ("acceptable" being vaguely defined as the situation where corridor is passed with relatively few backward movements) are not so small and sharp-edged as to render the test case unsuitable for given controller.

Then I applied (a somewhat primitive form of) evolution. The fitness of a controller is calculated based on how far the creature is after a certain number of simulation steps (in terms of "corridor distance", not the "as the crow flies" distance, from starting point). If the creature reaches the other end of corridor before time limit, the simulation run is terminated and extra score assigned depending on how much earlier the creature finished. To lessen fitness distortions by "lucky runs", each controller is tested three times and the scores are summed (each time the initial position of boids in controller room is different – they are placed there randomly).

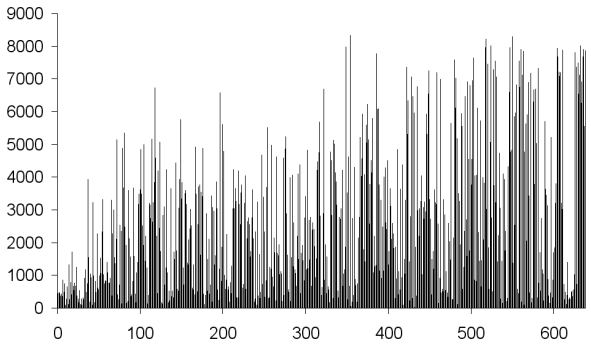

Fig. 5. Fitness scores of each controller in the (primitive) evolution process. The score is given on y-axis (arbitrary units; see text for explanation about what the fitness represents), and x-axis is just the number of each controller (they are generated and tested one at a time, and enumerated in increasing order).

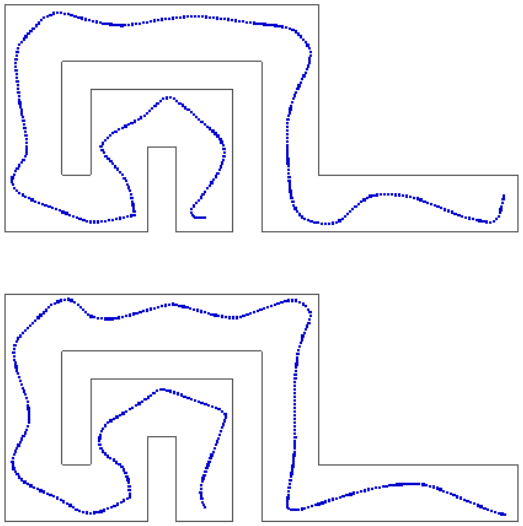

Fig. 6. Traced paths of the creature moving in a corridor during some good runs with a well-evolved controller. Both have the same controller, but initial positions of boids in the controller room are different.

Controllers are tested one at a time, and best 20 are kept as the breeding pool. First 20 controllers are parameterized randomly (within predefined reasonable ranges). After that, there is 0.1 probability of generating random controller, and 0.9 probability of crossing existing parents, who are drawn from the pool of 20 best with probability proportional to fitness (i.e., more fit are selected more often). Crossing procedure goes through each changeable parameter and either, with probability 0.1, generates a random value, or, with probability 0.9, takes the value of that parameter from one of the parents (both parents having equal probability of being the source of that parameter value).

Fig. 5 shows the progress of evolution. Although the used evolution process is quite primitive, it is good enough to generate increasingly fit solutions.

Even though the initial null hypothesis tests raised suspi-

cions that the proposed flock-based controller may not live up to the hopes, the more fit solutions found by evolution proved the doubts wrong. There exist parameter values which make the controller work reasonably well, as seen in Fig. 6. Detailed description and analysis of the parameter values for this test case is out of the scope of current paper, but at least the number of boids deserves some attention here. Namely, although in found good controllers the number of boids tended to be above 10, a question may arise about whether really a *swarm* is needed, or maybe one boid would be enough for such a simple prototype. To find an answer, I forced the number of boids to be fixed to one in the evolution process. When running the evolution with such a constraint, it can be observed (Fig. 7) that the fitness of the controllers stays quite low. It may well be possible that there does exist a suitable combination of parameter values for the single boid controller to be successful, and that these values are not found within reasonable running time because used evolution algorithm is too primitive. But even if this is the case, it is clear that those values are considerably harder to find than good values for a multiple boid controller. In addition, it is highly unlikely that for more complex tasks one boid would ever suffice (as hypothesized earlier, even one *species* of boids could be not enough in more difficult cases).

IV. Possible Applications

Even with the best found multi-boid controller, the paths of the creature through corridor are definitely not the shortest possible. Therefore, industrial control engineering may consider the flock-based controller not particularly useful, at least for such simpler tasks. However, the movements of the creature are interesting to look at (if not for everyone, then at least for people with ALife background) – it may take observing tens of passes through the same corridor with the same controller (but with different initial positions of boids, which alters the whole route of the creature) before boredom sets in. I consider it a good, though not necessarily scientific, sign of high enough potential of the proposed concept for ALife and suggest further testing and development of the idea.

While not directly applicable to serious problems at this stage of development, the proposed idea has already several possible uses. First of all, it serves as an inspiration calling for bolder explorations in the field of Swarm Intelligence. Secondly, it can be used as a visually appealing and relatively easily understandable illustration of SI in talks for the general public (which is not to say anything is wrong with illustrative applications currently in use, it is just that having a larger base of demonstrations makes it easier to pick suitable presentation tools for specific audiences). Thirdly, it could be used in computer games for generating somewhat lifelike and unexpected behavior.

More practical uses of flock-based control can certainly be found after further research.

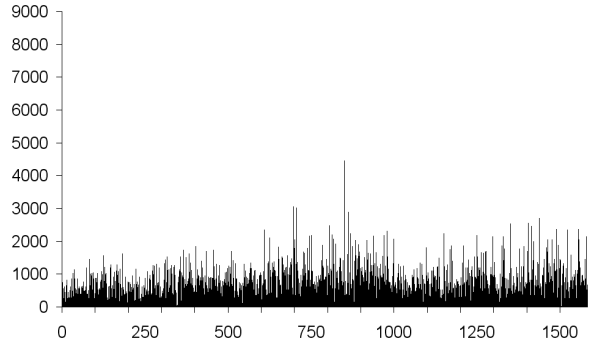

Fig. 7. Fitness scores of each controller in the (primitive) evolution process when the number of boids is fixed to one. Axes same as in Fig. 5.

V. Conclusion

There is plenty of room for innovations in the field of Swarm Intelligence. The proposed FlockHeadz concept is just a small sidestep from the main flourishing paths of SI, but hopefully it inspires further such steps, while also proving to be useful itself in a few specific applications.

Acknowledgment

Thanks to Leo Mõtus for providing me the possibility to do highly unconstrained research. Thanks to Roland Pihlakas, Oleg Davidyuk, René Michelsen, and the two anonymous reviewers, for reading the first versions of this paper and providing constructive criticism.

References

[1] M. Dorigo, "Ant colony optimization," Scholarpedia, 2(3):1461, 2007.
[2] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: survey and new directions," IEEE Transactions on Systems, Man and Cybernetics, Part A, 33(5):560–572, 2003.
[3] X. Hu and R. C. Eberhart, "Adaptive particle swarm optimization: detection and response to dynamic systems," Proceedings of the 2002 Congress on Evolutionary Computation CEC'02, Vol. 2, pp. 1666–1670, IEEE Computer Society, 2002.
[4] D. Parrott and X. Li, "A particle swarm model for tracking multiple peaks in a dynamic environment using speciation," Congress on Evolutionary Computation, CEC 2004, Vol. 1, pp. 98–103, IEEE Computer Society, 2004.
[5] J. M. Bishop, "Stochastic diffusion search," Scholarpedia, 2(8):3101, 2007.
[6] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," Computer Graphics, 21(4):25–34, 1987.