AN EXPERIMENTAL COLLECTIVE INTELLIGENCE RESEARCH TOOL

Bei Wang[1], Dung Hoang, Idris Daiz, Chiedu Okpala and Tarek M. Sobh

Department of Computer Science, University of Bridgeport
Bridgeport, CT 06601U.S.A

Abstract: The Collective Intelligence Research Tool (CIRT) is an experimental software and hardware research tool. It provides an inexpensive and efficient alternative research implementation that demonstrates simulations of the collective behaviour of self-organized systems, primarily social insects. The software focuses on 2D simulations of the woodchip-collecting behaviour of termites and 3D simulations of the building behaviour of wasps. The hardware simulation employs a Boe-Bot robot, which has the potential of simulating simple movements of a social insect, by extending its functionality through adding sensors and integrating a control chip.

Keywords: Artificial Life, Intelligent Agents and Multi Agent Systems.

1. INTRODUCTION

Social insects are known to be capable of producing complicated colony patterns [1]. Our first project objective is to simulate self-organized systems using social robots. We have implemented a robotic termite agent, which is able to simulate the wood-chip collecting behaviour of the termite. By defining the behaviour for one robotic agent, we could potentially observe the collective building activity of a group of robots. From a software viewpoint, our goal is to simulate and visualize the collective building of complex architectures for termites in 2D space and social wasps in 3D space. In addition to simulating self-organized systems by changing variables such as the population and obstacle density, the software provides an artificial life environment for observation of the emergent behaviour of autonomous agents (in our case, termites and wasps).

Current research on simulation of self-organized systems and swarm intelligence have a shared underlying idea that the key feature of all nature's patterns is that they are "self-organized" – there is no guiding hand [1-9]. Existing research projects include StarLogo, StarLogoT, NASA COIN project, Repast, AgentSheets, Ascape and SWARM [10-17]. StarLogo is a programmable modelling environment for "exploring the workings of decentralized systems, such as bird flocks, traffic jams, and market economies" [10]. RePast is a software framework for creating agent-based simulations, which provides a library of Java classes for creating and running agent based simulations [11]. SWARM is a software package for multi-agent simulation of complex systems, originally developed at the Santa Fe Institute [20].

We have implemented the simulation of collective intelligence systems from both software and hardware perspectives as a complete experimental experience. The 2D simulation of termites' behaviour employs methodology found in the Starlogo project demonstration and biological observations [10]. 3D simulation in our project focused on the building behaviour of social wasps, using the methodology found in the work of Eric Bonabeau et al. [1].

Our hardware simulation draws idea from research done by Krieger M. J. [6]: given robots with the ability to perform simple object removal tasks, researchers are able to simulate collective behaviour among cooperative robots (in our case, termite agents) [6].

2. SOFTWARE SPECIFICATIONS

Our simulation software is built around the Repast framework. The software adopts the Repast graphic user interface (GUI). The Repast GUI is able to initialize, start, pause and stop a simulation. It also enables user to alter some of the simulation variables, such as the size of the display surface (sample space) and number of agents [10].

Repast is able to handle 2D termites' simulation but has no built-in 3D visualization functionalities. However, its pure Java implementations enable Java 3D API integration.

2.1 2D Termites Simulation

2D termites simulation is a common practice for self-organized system research. It is included in our software as a sample project. We based our development on the simulation of collective building of 2D termites' colony, which involves two major objects: termites and their woodchips [10]. The termites gather wood chips into piles following a set of simple rules demonstrated in the StarLogo project [10]:

1. Each termite walks around randomly in the sample space.

[1] Contact author: beiwang@bridgeport.edu

2. An empty-handed termite picks up a randomly distributed wood chip if it comes across one.
3. The termite continues to walk around randomly.
4. When the termite comes across another wood chip, it finds a nearby empty space and puts its wood chip down and becomes empty-handed again [10].

CIRT simulates and visualizes in a 2D space the termites gathering wood chips into piles based on the initial behaviour definition. It also observes and predicts possible outcome by redefining the number of termite and environmental variables, such as the woodchip density. As the simulation progresses, the randomly distributed woodchips would end up in a single large pile, as shown in similar simulations [10].

2.2 3D Lattice Swarms Simulation

The architectural patterns grown by "artificial agents moving and acting in a virtual space" (in our case, artificial wasps) are based on biological data provided by observations of nests built by social wasps [1]. We based our development on the simulation of the collective building of 3D wasps' colony, which involves two major objects: wasps and their bricks. According to Eric Bonabeau et al.'s research, using stigmergic algorithms, these agents move and act in a 3D lattice and are able to "deposit bricks according to their local neighbourhood configurations (26 neighbouring cells for 3D lattice swarms) using a look-up table" [1].

In the stigmertic mode of construction, each swarm insect automatically responds (dropping bricks) when it meets any local configuration. As explained by Bonabeau et al., the regulation of the building activity is mainly achieved by the nest structure, instead of depending on the workers themselves [1].

The wasps put bricks into a 3D structure with the following behaviours [1]:

1. Each wasp is born at a random location in the 3D space.
2. The wasp observes its local configuration with 26 neighbouring cells.
3. If the local configuration applies to one of the pre-defined patterns, the wasp drops a corresponding brick at that location and then moves to another random location.
4. If the local configuration doesn't apply to any of the patterns, the wasp does nothing and moves to another random location.
5. The result of the construction eventually produces certain architectures that can be found in nature.

CIRT simulates and visualizes in 3D space the growth of the colony. Social wasps act in 3D space and drops bricks based on pre-defined behaviour rules. The software observes the outcome by redefining the number of wasps.

According to Bonabeau's research, the neighbourhood of the wasp is composed of the 26 cells surrounding the central cell it occupies. [1] This neighbourhood consists of 3 3X3 layers along the y-axis (see figure 1) [1]. When the wasp occupies the central position of the layer y (marked in black), it follows certain rule to produce our 3D architecture [1]. For example, when there is no brick in the central cell, a wasp puts down a brick of type 1 in the case of configuration 1 in figure 1 (9 cells above are all already filled with type 2 bricks) and type 2 in the case of figure 2 – 4 [1]. There are 9 configurations in total. Detailed rules can be found in [1]. Furthermore, taking symmetries into account, each rule expands further. For example, configuration in figure 2 expands to more configurations as shown in figure 3 and 4.

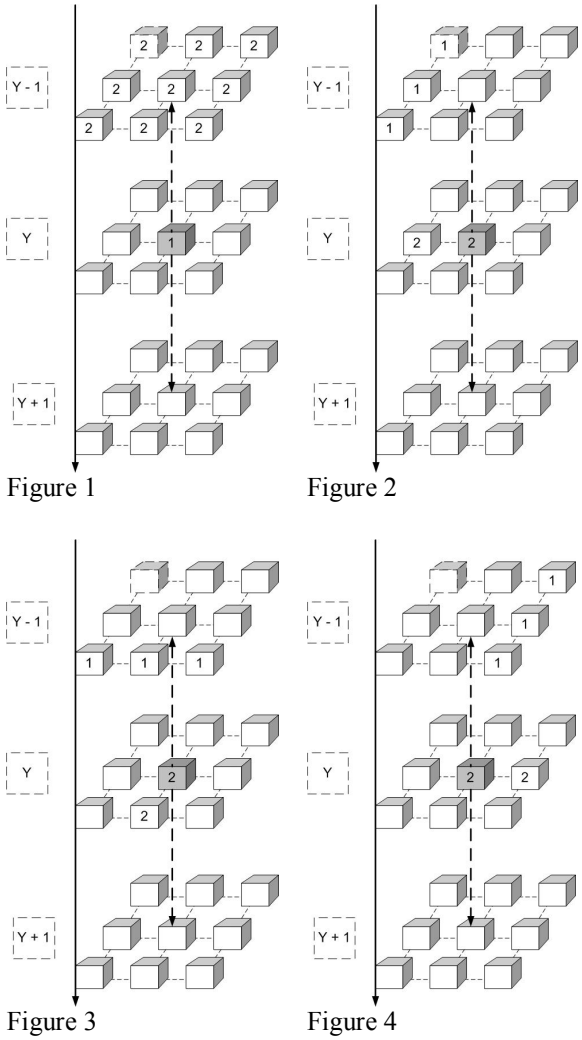

Figure 1 Figure 2

Figure 3 Figure 4

Figure 1-4: Local neighbourhood in 3D lattice swarm [1]

3. SOFTWARE IMPLEMENTATION

Our simulation uses the Repast framework, an agent based modelling toolkit for java. It has three major classes: agent, space and model. Employing the Repast software architecture, an agent class describes how an agent interacts with the environment and moves around the space [1]. A model class coordinates the setup and running of the model. A space class defines the

environment, such as the distribution of woodchips for the termites' 2D simulation and the coordinates of wasps and bricks in swarm 3D simulations [10].

Our software implementation observes and predicts possible outcomes by defining a number of termite and environmental variables, such as the density of obstacles (wood chips). Figure 5 is a screen shot for the software simulation in action. The red rectangle represents termite carrying no woodchip; orange rectangle represents termite carrying one woodchip; yellow rectangle represents woodchip. During the simulation, the user observes the movement of red and orange "termites", picking-up or dropping woodchips in the simulation space.

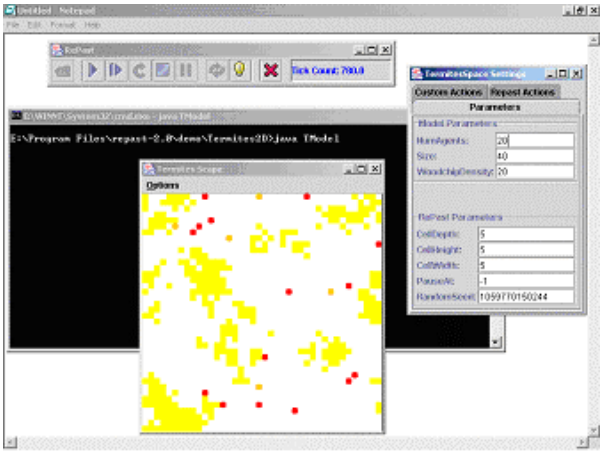

Figure 5: 2D termite simulation

In order to visualize the 3D colony, we use Java 3D API. Java 3D defines the concept of a virtual universe as a three-dimensional space with an associated set of objects [7]. Since Repast doesn't come with 3D visualization support, we separately programmed a set of Java classes to be integrated with Repast to realize and illustrate the 3D colony architecture. Our 3D integration works with the Repast original GUI in a way that the display surface corresponds to commands sent from various buttons, such as setup, step, pause and stop. With proper time-delay between each clock tick, the users can observe the growth of the artificial colony architecture.

Furthermore, we incorporate mouse rotation functionality into the 3D visualization. At each clock tick, users are able to rotate the visual colony by left-mouse click so that the colony is clearly-viewed from different angles (figure 6, 7).

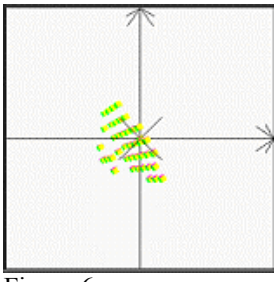

Figure 6

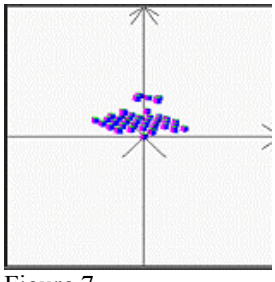

Figure 7

We present a simple example of architectures grown by artificial agents moving randomly in the 3D space and performing simple "asynchronous actions with purely local information" [1], as shown in figure 8.

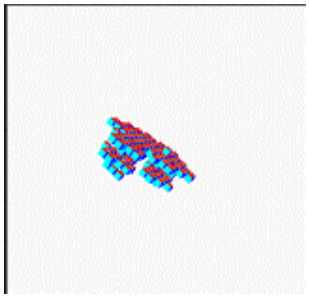

Figure 8

4. HARDWARE SPECIFICATIONS AND IMPLEMENTATION

4.1 Hardware Specifications

In order to achieve the same results from Krieger's research [6], we needed to construct a small-scale and low-cost robot that can perform simple object removal tasks. These tasks including moving on smooth surfaces, detecting new objects (woodchips in our case), picking up an encountered new object and dropping the woodchip it carries when encountering another object. The robot should come with sensors that are sensitive enough to detect objects within 20-30 cm range. Because the final simulation requires a relatively large number of robot agents, the robot should be easy and fast to assemble. Since it moves around randomly, it should be using batteries as its primary power supply (a power cord will provide an extra obstacle). For more economical reasons, the robot toolkit should be reusable, reprogrammable and consume as little power as possible. It should be easy to connect to other devices. We chose the Boe-Bot Tool Kit from Parallax Inc. and the Board of Education featured BASIC Stamp embedded microcontroller [18].

4.2 Boe-bot Description

The Boe-Bot is built on a high quality brushed aluminum chassis that provides a sturdy platform for the servomotors and printed circuit board (figure 9) [18]. Mounting holes and slots may be used to add custom robotic equipment [18]. "The rear wheel is a drilled polyethylene ball held in place with a cotter pin… Wheels are machined to fit precisely on the servo spine and held in place with a small screw. " [18] In our case, to simulate the termite's woodpile building process, each Boe-Bot needs a gripper.

The main controller of Boe-Bot is a BS2-IC (BASIC Stamp 2), which is a customized chip from Microchip PIC 16C57C. The BASIC Stamp 2 has 16 I/O pins, 2 dedicated serial port pins (1 input, 1 output), and room

for 500 to 600 lines of code. Detailed technical description for BS2-IC as well as its schematic can be found in Boe-Bot User Manual.

4.3 Termite Description

In the following section, we define the term "termite" as a Boe-Bot with a pair of whiskers and gripper (figure 9). The hardware design methodology divides the implementation into two major components: whiskers module and gripper Module.

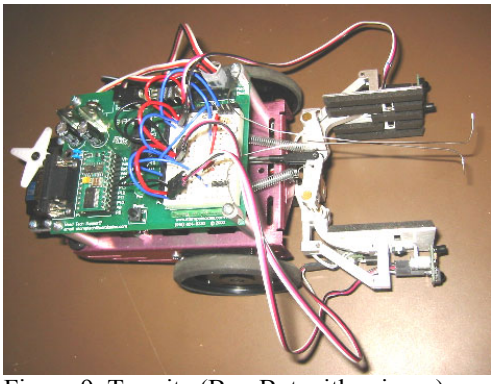

Figure 9: Termite (Boe-Bot with gripper)

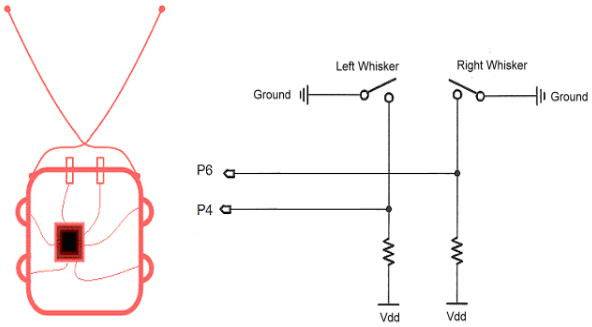

Figure 10: Whiskers schematic

4.3.1Whiskers Module

The Whiskers are used as object detectors since the BASIC Stamp can be programmed to detect when a whisker is pressed. Once the termite touches a new object by it whiskers, it will release the object it is holding (if there is one). Here, pin 4 and pin 6 connected to each switch circuit monitor the voltage at the 10 kΩ pull-up resistor. When a given whisker is not pressed, the voltage at the pin connected to that whisker is 5 V (logic 1). When a whisker is pressed, the I/O line is shorted to ground, and the pin sees 0 V (logic 0). See figure 10 for details.

A program will keep checking whether the logic from pin 4 and pin 6 is changed. If there is a change, a corresponding subroutine will be called to react to the change, by either releasing the object it carries or avoiding the object it touches.

4.3.2 Gripper Module

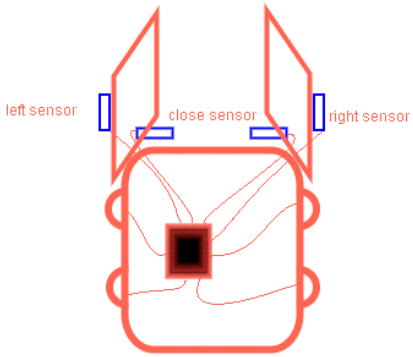

Figure 11: gripper

The gripper (figure 11) has 3 pairs of IR sensors used for object detection. They are used to control the movement of the termite as well as the gripper. The IR unit incorporates a standard IR LED with a 40 kHz IR receiver. The IR specification as well as it schematic is given below (figures 12 and 13):

Size: Width = 15.8 mm, Length = 18.2 mm
Power Requirements: + 5vdc, 2.6 mA
Vdd = +5vdc
Signal = I/O pin 0

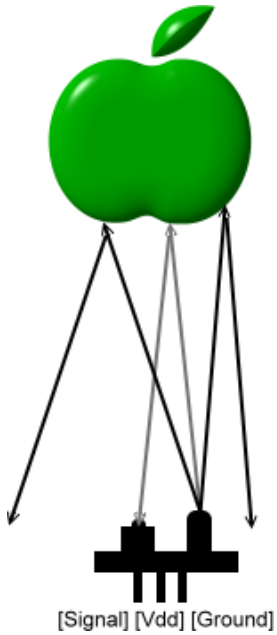

Figure 12: IR specification

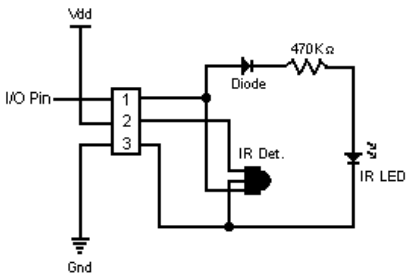

Figure 13: IR schematic

Note that the IR LED (emitter) and IR Detector are both connected to the same I/O pin.

4.3.3 Robotic Termite Working Scenario

The termite works as follows: first, we let the robot spin left (360 degrees), and keep detecting the signals sent by both the left sensor and right sensor of the gripper (figure 14). Second, if the left senor signal is on, meaning that the robot detects an object from the left, we let the robot turn left until the right sensor turns on. This indicates that the robot has just passed the object (figure 15). Thus, we will let the robot turn right for a little (an angle of around 3 degrees) to centre the object into the gripper (figure 16). Now the robot can keep moving straight until the close sensor is on (this means the object is inside the robot), and grips the object (figure 17). Afterwards, the robot starts searching for a new object. When it hits the new object by its whisker, it releases the object it is carrying. After releasing the object, the robot moves backward, turns an angle of 45 degrees, and the same procedure is repeated.

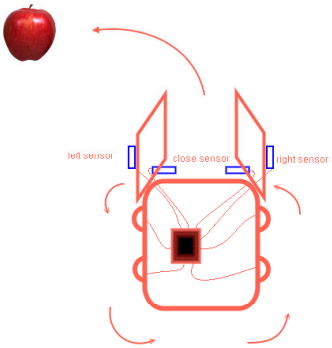

Figure 14

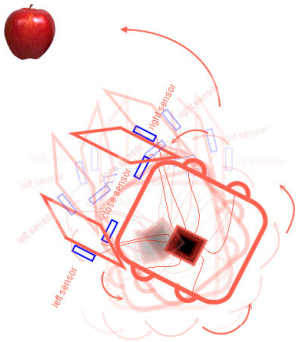

Figure 15

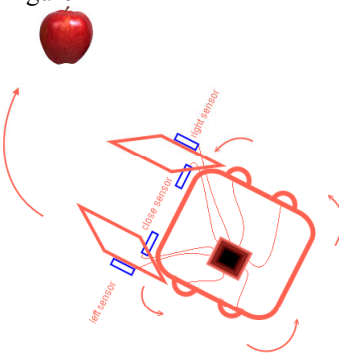

Figure 16

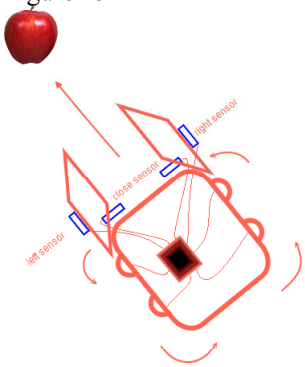

Figure 17

The robot can detect a new object by using its left and right sensors. We use a touch sensor to enable the detection of the second object (in order to release the one it carries). However, there is one drawback: if the new object the robot encounters is too big, it could activate the touch sensor and the robot would release the object right after gripping it.

5. RESULTS AND CONCLUSIONS

Simulation results are shown below as screen shots for the software implementation. Figure 18, 19 and 20 show 2D termite simulation.

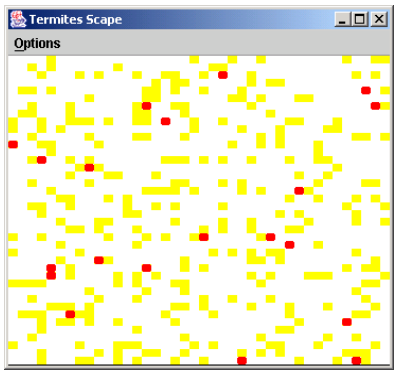

Figure 18: initial stage

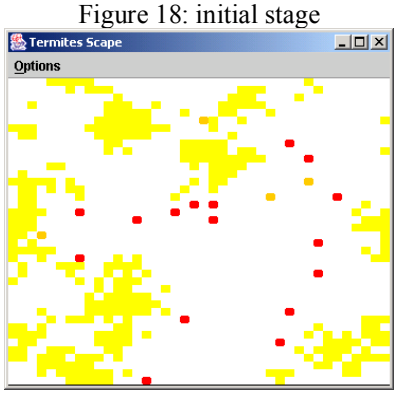

Figure 19: in progress

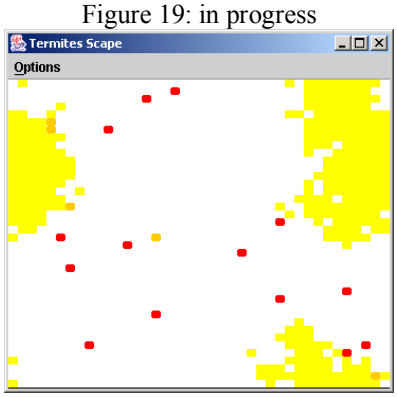

Figure 20: final stage

Figure 21 shows a lattice swarm simulation done within a 20X20X20 3D space with 315 tick counts (note there is an X-Y axis displayed in this simulation).

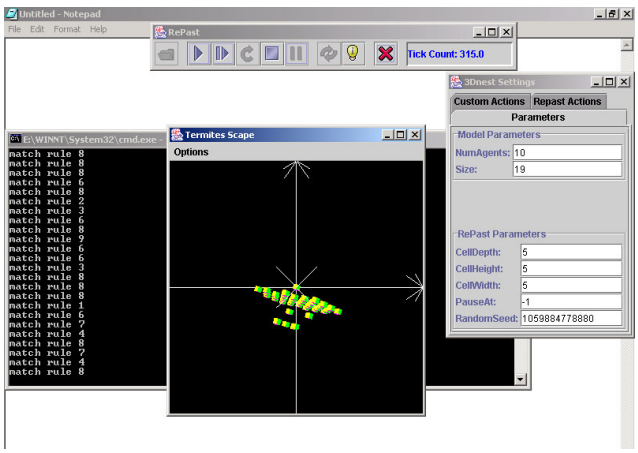

Figure 21

Figure 22 - 25 shows a lattice swarm simulation done within a 20X20X20 3D space with 38142 tick counts.

Figure 26 shows a simulation done within a 40X40X40 3D space with 91306 tick counts. Due to the huge size of the simulation space, the resulting structure remains relatively small-scale even after a large number of tick counts.

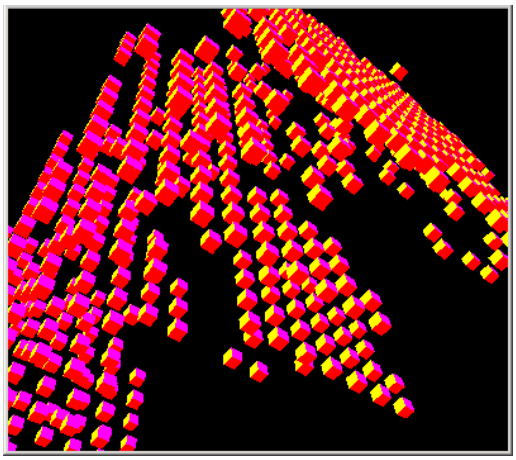

Figure 22

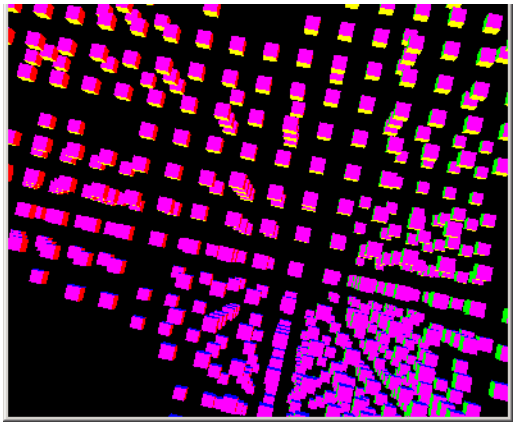

Figure 23

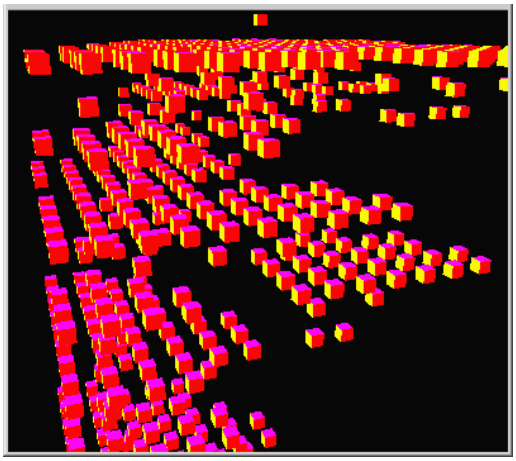

Figure 24

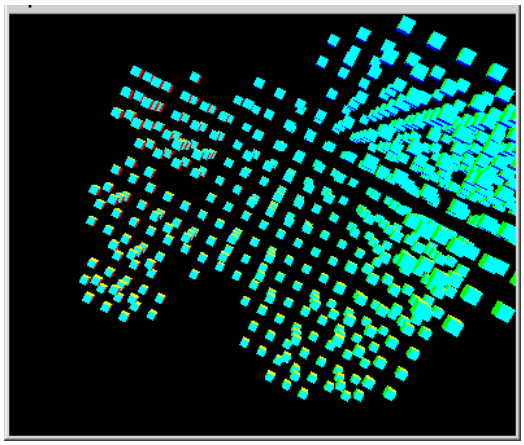

Figure 25

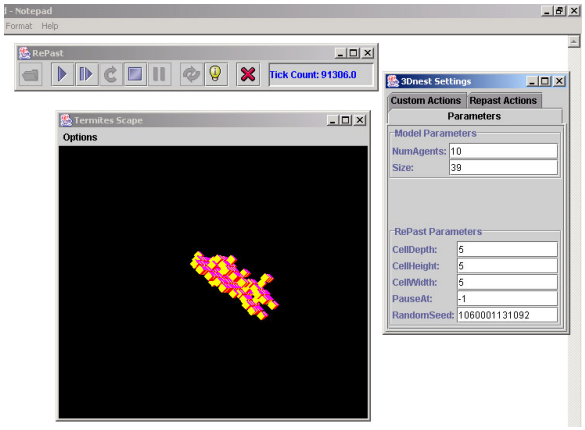

Figure 26

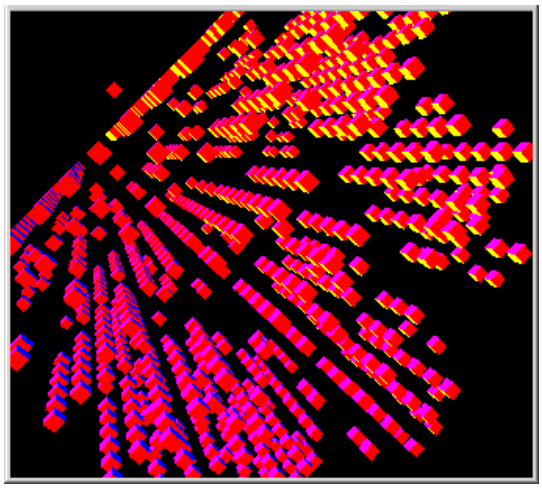

Figure 27: our simulation result

Comparing our simulation results (figure 27) with that of the Wasp Nest Building Simulator, it can be seen that both simulations have resulted in nature-like patterns [1, 19]. Bonabeau believes that "extensive simulations on a powerful computer have to be performed in order to explore the behavioural space in a satisfactory manner, even in this simplest case" [1]. Our software proves that small scale simulation runs smoothly and relatively fast, even on a PC. We offer 3D rotations during simulation; enable users to observe the visualization from various angles. While Wasp Nest Building Simulator is implemented using C/C++ and is to be distributed under GNU License soon, our software is solely implemented

in Java, which is platform independent. Furthermore, our implementation integrates neatly with the Repast toolkit framework and may provide future researchers convenience and user-friendly interface. Video clips showing the actual software and hardware simulations are available upon request by contacting the authors.

As a conclusion, for the software simulation, as indicated by Bonabeau et al., we have restricted our attention to "very simple individual algorithms, where information is processed locally, in space as well as in time" [1]. We hope the simulation results, which closely resemble these found in [1], can be of some value to the understanding of biological collective systems [1]. Though our software - so far - is modelled after the research done by Bonabeau et al, it provides an inexpensive tool, and a fast and platform independent implementation, for researchers seeking new ways to implement these systems.

6. FUTURE DEVELOPMENT

Where can we go from here? From a software perspective, as Repast is an open-source software framework for creating agent-based simulations using the Java programming language and it is a more sophisticated development tool, further developments might include developing a whole 3D visualization library that can be integrated into Repast, for realizing powerful simulations.

From hardware perspective, we might be able to get more robots to form a group of termite agents for better simulation results. Meanwhile, we may also install an extra pair of sensors for each robot to detect new objects. Moreover, it is also possible to bring in interaction between simulation software and hardware, such as programming robotic agents behaviors through a software terminal. The experimental robots may also lead to further research in the area of cataglyphis fortis [20, 21]

In addition, biologists are concerned with the individual behavioral algorithms that allow a society to build its nest [1]. Researchers have looked into genetic algorithms to implement nest construction algorithms [16]. Therefore, it is possible to implement an algorithm generation component or more precisely, a "rule generator" for our software and integrate it into Repast, to enhance its simulation possibilities.

REFERENCES

[1] Bonabeau, E. Théraulaz, G., Arpin, E. and Sardet, E. The building behavior of lattice swarms. In: Artificial Life IV, Brooks, R. and Maes, P. eds., pp. 307-312, MIT Press (1994).

[2] Ramos V. On the Implicit and on the Artificial: Morphogenesis and Emergent Aesthetics in Autonomous Collective Systems. In: ARCHITOPIA Book / Catalogue, Art, Architecture and Science, J.L. Maubant and L. Moura (Eds.), pp. 25-57, Ministério da Ciência e Tecnologia, Feb. 2002.

[3] Langham, A. E. and Grant P. W. Evolving the Building Activity of a Termite Colony for Finite Element Mesh Generation. Department of Computer Science, University of Wales Swansea Research Report CSR1 4-99 (1999).

[4] A Cooperative Multi-Robot Control Architecture. Dynamic Concepts, Inc. Technical Report (2002).

[5] Fong, T., Nourbakhsh, I., and Dautenhahn, K., A Survey of Socially Interactive Robots. In: Robotics and Autonomous Systems, vol. 42(3-4), March 2003.

[6] Krieger, M. J., Billeter, J.B., Keller, L. 2000. Ant-like task allocation and recruitment in co-operative robots. In: Nature, 406, 992-995. 2000.

[7] Bonebeau E., Dirigo M.and Theraulaz G. Inspiration for Optimization from Social Insect Behavior. In: Nature 406, 39-42, 2000.

[8] Bonebeau E., Theraulazx G. and Cogne F., The Design of Complex Architectures by Simple Agents. In: Sante Fe Institute Working Paper 98-01-005.

[9] Resnick, M. Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds. Cambridge, Ma: MIT Press (1994).

[10] StarLogo. (http://education.mit.edu/starlogo/)

[11] Collier, N. Repast: An extensible framework for agent simulation (2002). (http://repast.sourceforge.net/)

[12] NASA COIN Project. (http://is.arc.nasa.gov/AR/projects/ColInt.html)

[13] SWARM. (http://www.swarm.org/)

[14] StarLogoT. (http://ccl.sesp.northwestern.edu/cm/starlogoT/)

[15] NetLogo. (http://ccl.sesp.northwestern.edu/netlogo/)

[16] AgentSheets. (http://agentsheets.com/)

[17] Ascape. (http://www.brook.edu/es/dynamics/models/ascape/)

[18] Boe-Bot Specifications. (http://www.parallax.com/html_pages/robotics/boebot/)

[19] Wasp Nest Building Simulator. (http://www-iasc.enst-bretagne.fr/PROJECTS/SWARM/nest.html)

[20]Roumeliotis S.I., Pirjanian P. and Mataric M.J. Ant-Inspired Navigation in Unknown Environments. In Proc. 2000 AAAI International Conference on Autonomous Agents, Barcelona, Spain, June 3-7, pp. 25-26.

[21]Kuipers B. and Byun Y.T. A robot exploration and mapping strategy based on semantic hierarchy of spatial representations. In: Robotic and Autonomous Systems, 8(1-2):47-63, Nov. 1991.