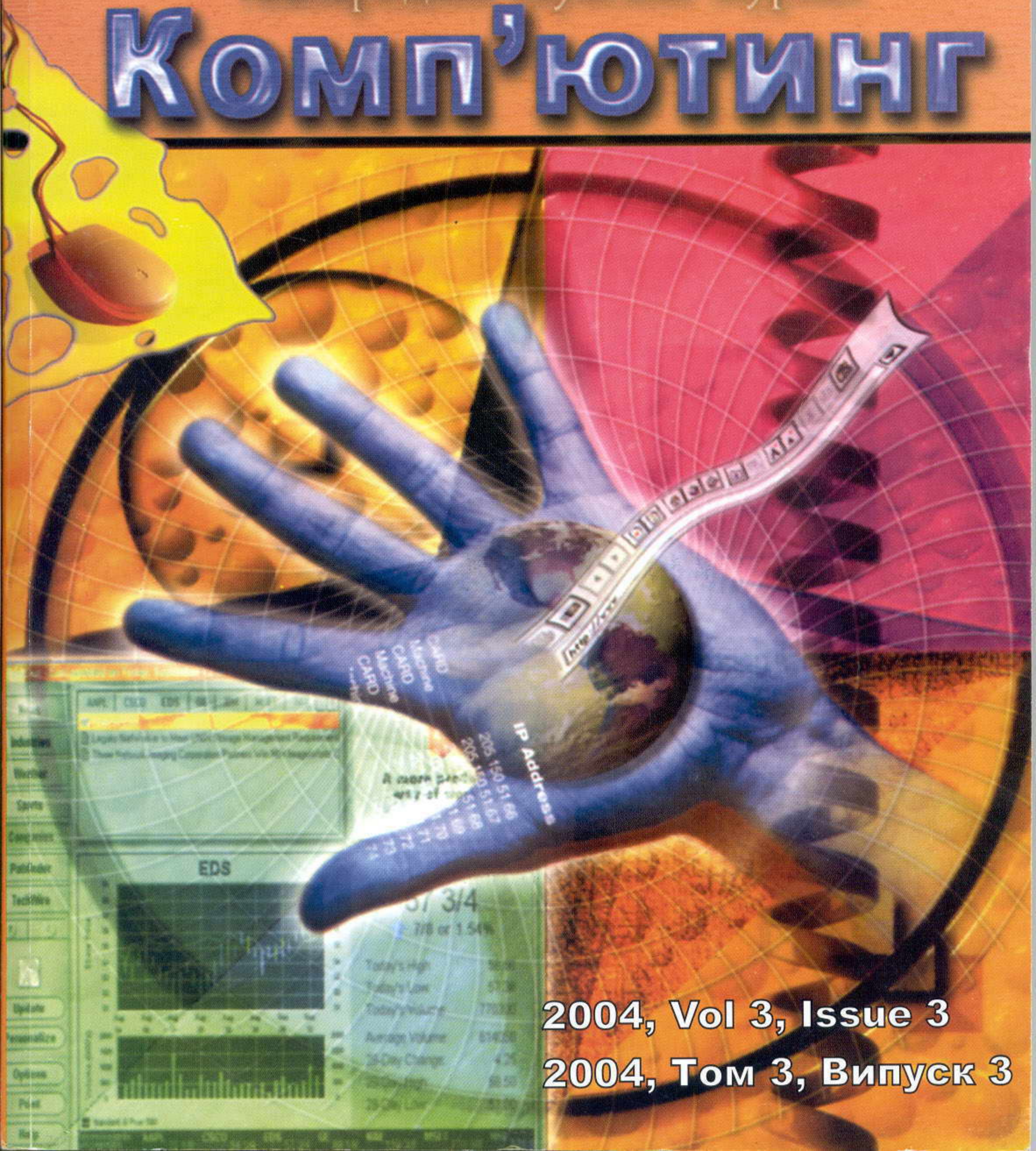
ISSN 1727-6209

International Scientific Journal of

Computing

Міжнародний Науковий Журнал

Комп'ютинг



2004, Vol 3, Issue 3

2004, Том 3, Випуск 3

computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

AN EXPERIMENTAL COLLECTIVE INTELLIGENCE RESEARCH TOOL

Bei Wang, Dung Hoang, Idris Daiz, Chiedu Okpala, Tarek M. Sobh

Department of Computer Science, University of Bridgeport
Bridgeport, CT 06601, U.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 behavior of self-organized systems, primarily social insects. The software focuses on 2D simulations of the woodchip-collecting behavior of termites and 3D simulations of the building behavior 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 behavior of the termite. By defining the behavior 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 behavior 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 modeling 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 using the Java language, which provides a library of classes for creating, running, displaying and collecting data from an agent based simulation [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' behavior employs methodology found in the Starlogo project demonstration and biological observations [10]. 3D simulation in our project focused on the building behavior 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 behavior 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 is also able to run a simulation step by step by clicking the step button. It also enables user to alter some of the simulation variables, such as the size of the display surface 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

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 starts wandering randomly.
2. If it bumps into a wood chip, it picks the chip up, and continues to wander randomly.
3. When it bumps into another wood chip, it finds a nearby empty space and puts its wood chip down.
4. The wood chips eventually end up in a single pile.

CIRT simulates and visualizes in a 2D space the termites gathering wood chips into piles based on the initial behavior definition. It also observes and predicts possible outcome by redefining a 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 [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. Using stigmergic algorithms, these agents move and act in a 3D lattice and are able to deposit bricks according to their local neighborhood configurations (26 neighboring 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 does not depend on the workers themselves, but is mainly achieved by the nest structure [1].

The wasps put bricks into a 3D structure using the following behaviors [1]:

1. Each wasp is born at a random location in the 3D space.
2. The wasp observes its local configuration with 26 neighboring 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 behavior rules. The software observes the outcome by redefining the number of wasps.

We based our 3D simulation on the exploratory study of architectural patterns grown by artificial agents moving and acting in a virtual space [1]. According to research, the neighborhood of the artificial swarm is composed of the 26 first cells surrounding the cell it occupies. This neighborhood is represented with 3 slices along the y-axis (see figure 1). Based on Bonabeau's research below (Table I), are the rules used to produce our simulated 3D architecture when the wasp occupies the central position of the slice Y (marked as "*"). The upper and lower matrices in the 3-D lattice are marked as Y-1 and Y+1. The numbers in tables I and II, indicate the brick type. When there is no brick in that cell, it will put down a brick of type 1 in the case of configuration 1 and type 2 in the case of 2 – 9 [1].

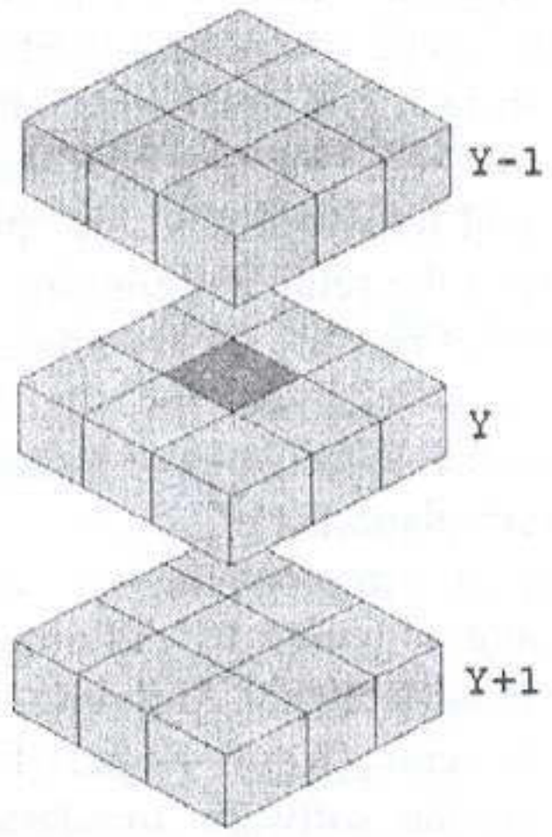


Fig. 1 - Local neighborhood in 3D lattice swarm [1]

Table 1. - Rules

Rule No.	Y-1	Y	Y+1	Brick Type	Rule No.	Y-1	Y	Y+1	Brick Type
1	2 2 2 2 2 2 2 2 2	0 0 0 0 * 0 0 0 0	0 0 0 0 0 0 0 0 0	1	6	0 0 0 0 0 0 0 0 0	2 2 2 0 * 0 0 0 0	0 0 0 0 0 0 0 0 0	2
2	0 0 0 0 1 0 0 0 0	0 0 0 0 * 0 0 0 0	0 0 0 0 0 0 0 0 0	2	7	0 0 0 0 0 0 0 0 0	2 2 2 2 * 0 2 0 0	0 0 0 0 0 0 0 0 0	2
3	0 1 0 0 0 0 0 0 0	0 2 0 0 * 0 0 0 0	0 0 0 0 0 0 0 0 0	2	8	0 0 0 0 0 0 0 0 0	2 2 0 2 * 0 0 0 0	0 0 0 0 0 0 0 0 0	2
4	0 1 0 0 0 0 0 0 0	2 2 2 0 * 0 0 0 0	0 0 0 0 0 0 0 0 0	2	9	0 0 0 0 0 0 0 0 0	2 2 2 2 * 2 0 0 0	0 0 0 0 0 0 0 0 0	2
5	1 0 0 0 0 0 0 0 0	2 2 0 2 * 0 0 0 0	0 0 0 0 0 0 0 0 0	2					

On the other hand, taking symmetries into account, some of the above rules expand further. For example, rule 3 expands to three more configurations as shown in table 2:

Table 2: Rule extensions

Rule No.	Y-1	Y	Y+1	Brick Type
3.1	0 0 0 1 0 0 0 0 0	0 0 0 2 * 0 0 0 0	0 0 0 0 0 0 0 0 0	2
3.2	0 0 0 0 0 0 0 1 0	0 0 0 0 * 0 0 2 0	0 0 0 0 0 0 0 0 0	2
3.3	0 0 0 0 0 1 0 0 0	0 0 0 0 * 2 0 0 0	0 0 0 0 0 0 0 0 0	2

3. SOFTWARE IMPLEMENTATION

Our simulation uses the Repast framework, an agent based modeling 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. 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 swarms 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 2 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.

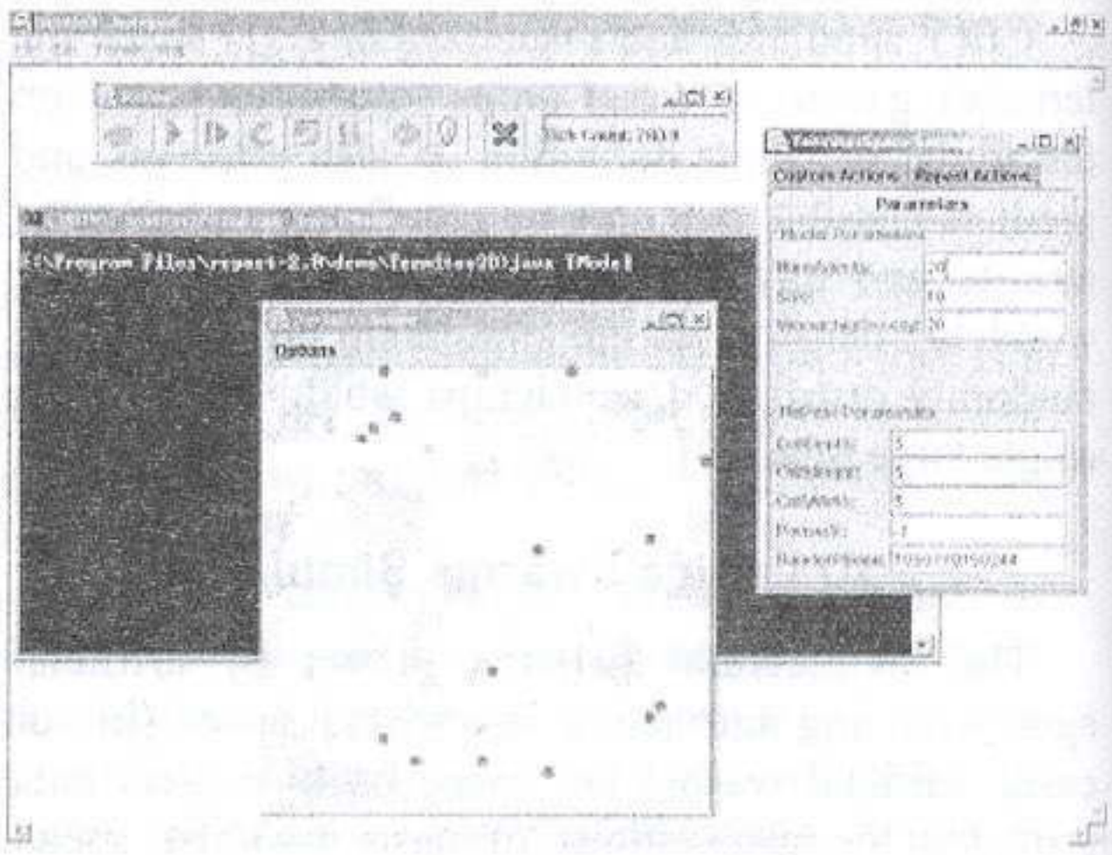


Fig. 2 - 2D termite simulation

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 3, 4).

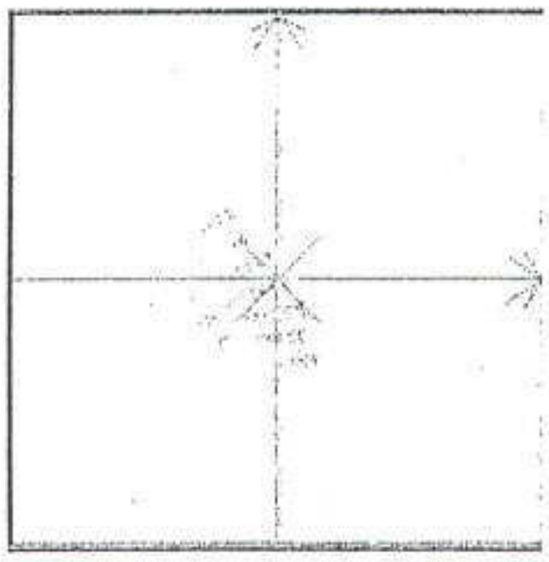


Fig. 3

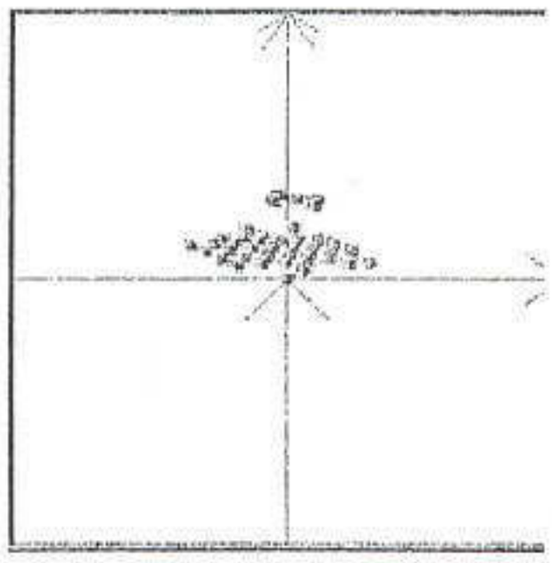


Fig. 4

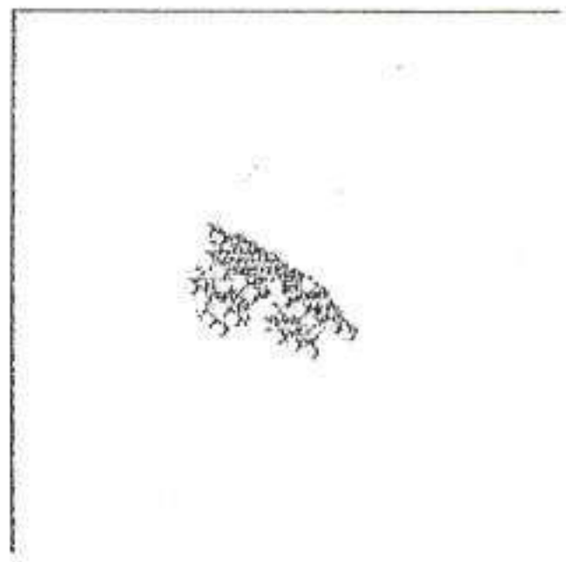


Fig. 5

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 5.

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 following information, and figures 6-8, describing the Boe-bot are available on the parallax web site http://www.parallax.com/html_pages/-robotics/boebot/boebot.asp:

"The Boe-Bot is built on a high quality brushed aluminum chassis that provides a sturdy platform for the servomotors and printed circuit board. Mounting holes and slots may be used to add custom robotic equipment. 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." In our case, to simulate the termite's woodpile building process, each Boe-Bot needs a gripper.

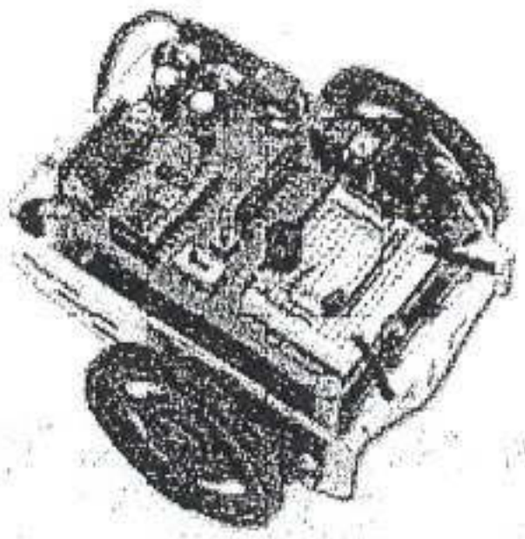


Fig. 6 - Boe-Bot

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. Figure 7 is a detailed technical description for BS2-IC as well as its schematic (figure 8).

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.

4.3.1 Whiskers Module

The Whiskers will be used as object detectors since the BASIC Stamp can be programmed to detect when a whisker is pressed. The whiskers are touch sensors, whose circuitry is depicted in Figure 10. Once the termite touches a new object by it whiskers, it will release the object it is holding (if there is one). If the termite does not carry any object, it will avoid the new object. 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.

BS2-IC SPECIFICATIONS	
Package	24-pin DIP
Package Size (L x W x H)	1.2" x 0.6" x 0.4"
Environment	0° - 70° C (32° - 156° F)
Microcontroller	Microchip PIC16C57c
Processor Speed	20 MHz
Program Execution Speed	~4,000 instructions/sec
RAM Size	32 Bytes (6 I/O, 26 Variable)
Scratch Pad RAM	N/A
EEPROM (Program) Size	2K Bytes, ~500 instructions
Number of I/O pins	16 + 2 Dedicated Serial
Voltage Requirements	5 - 15 vdc
Current Draw @ 5V	8 mA Run / 100 µA Sleep
Source / Sink Current per I/O	20 mA / 25 mA
Source / Sink Current per unit	40 mA / 50 mA per 8 I/O pins
PBASIC Commands	36
PC Programming Interface	Serial Port (9600 baud)
DOS Text Editor	STAMP2.EXE
Windows Text Editor	Stampw.exe (v1.04 and up)

Fig. 7 - BS2-IC specifications

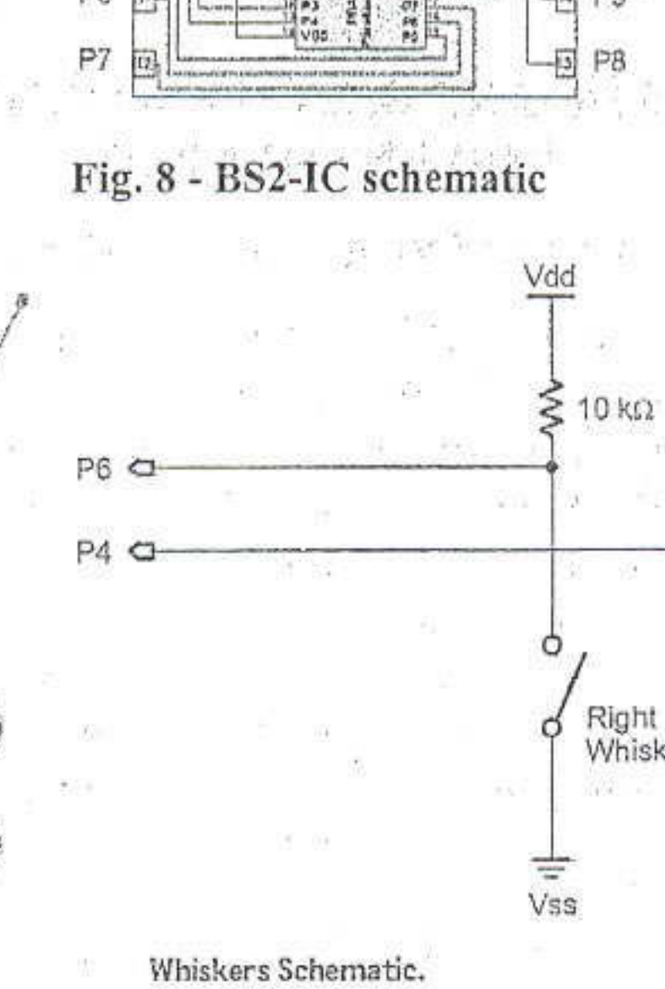


Fig. 8 - BS2-IC schematic

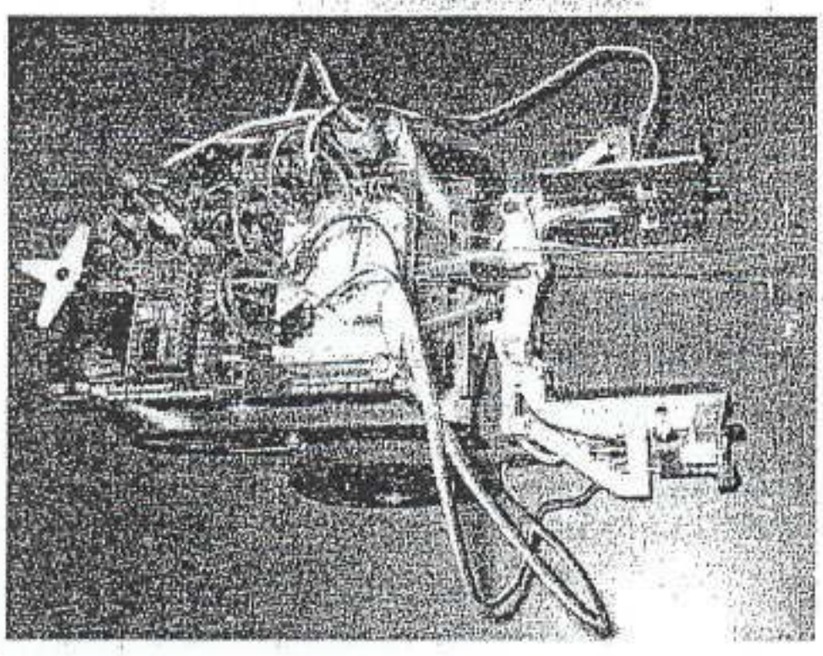


Fig. 9 - Termite (Boe-Bot with gripper)

Whiskers Schematic.

Fig. 10 - Whiskers schematic

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 is holding or avoiding the object it touched.

4.3.2 Gripper Module

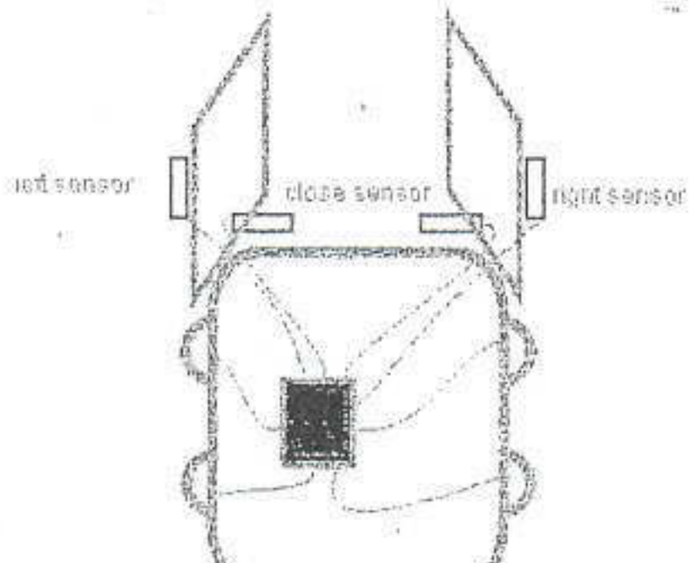


Fig. 11 - gripper

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

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 just passed the object (figure 15).

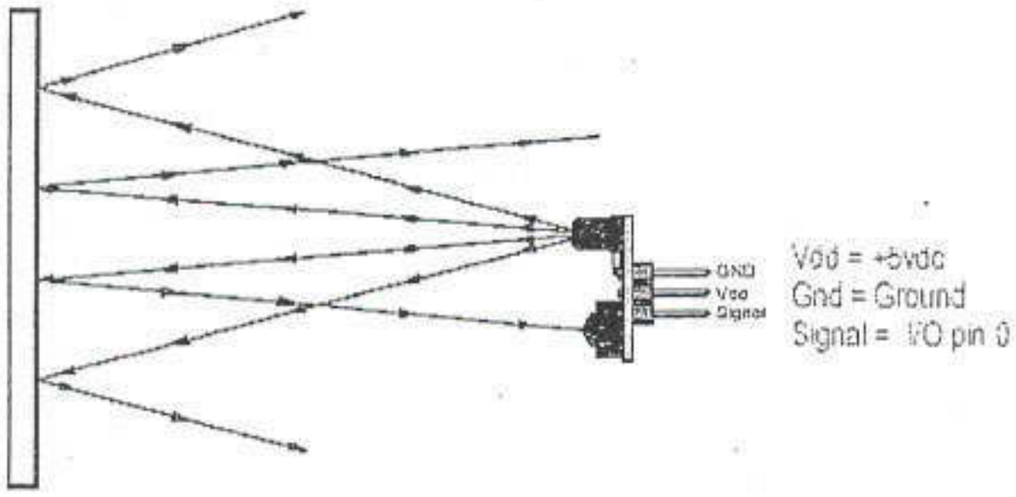


Fig. 12 - IR specification

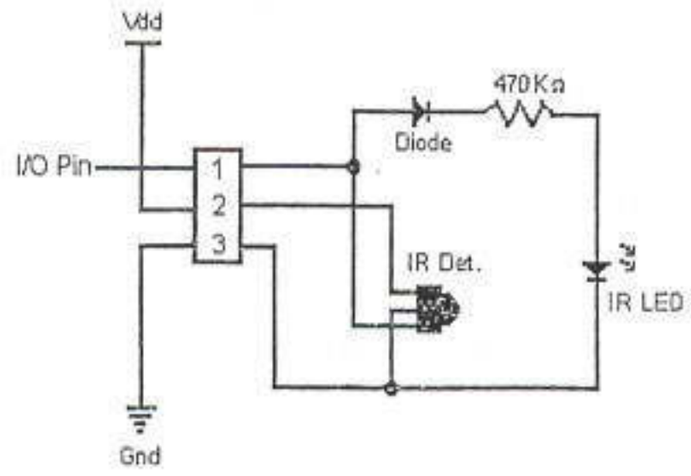


Fig. 13 - IR schematic

Thus, we will let the robot turn right for a little (an angle of around 3 degrees) to center 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 either of it 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.

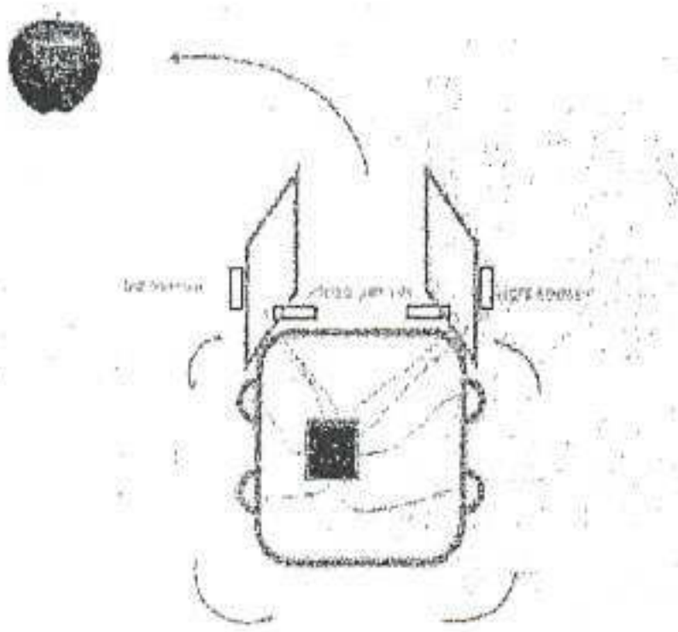


Fig. 14

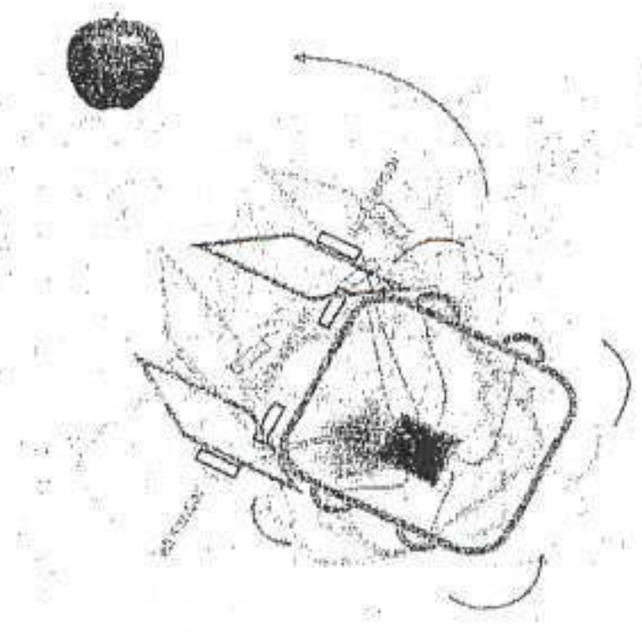


Fig. 15

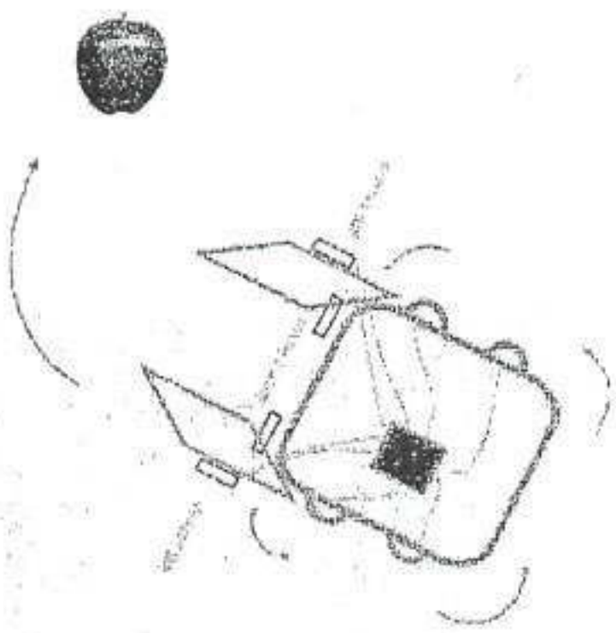


Fig. 16

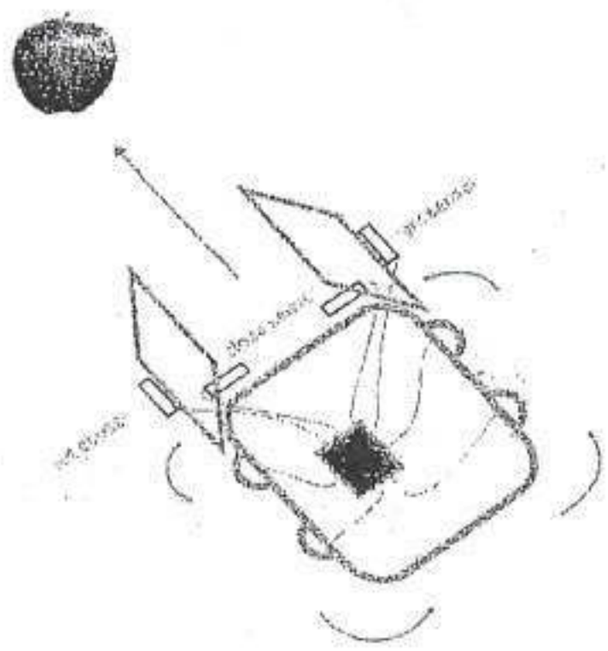


Fig. 17

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.

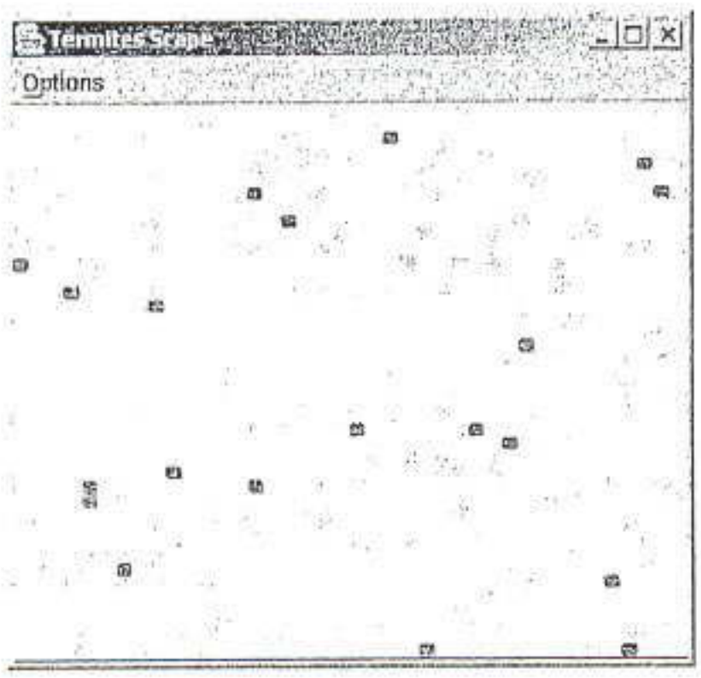


Fig. 18 - initial stage

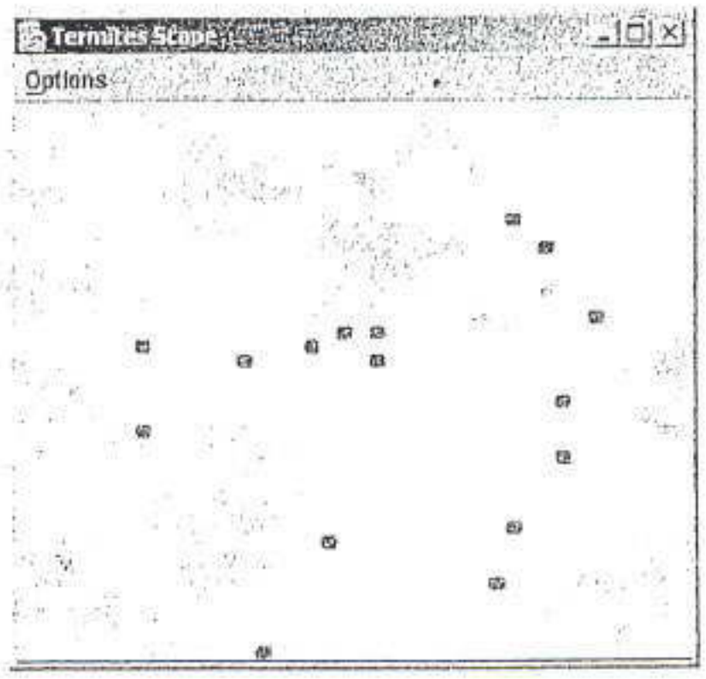


Fig. 19 - in progress

Fig. 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).

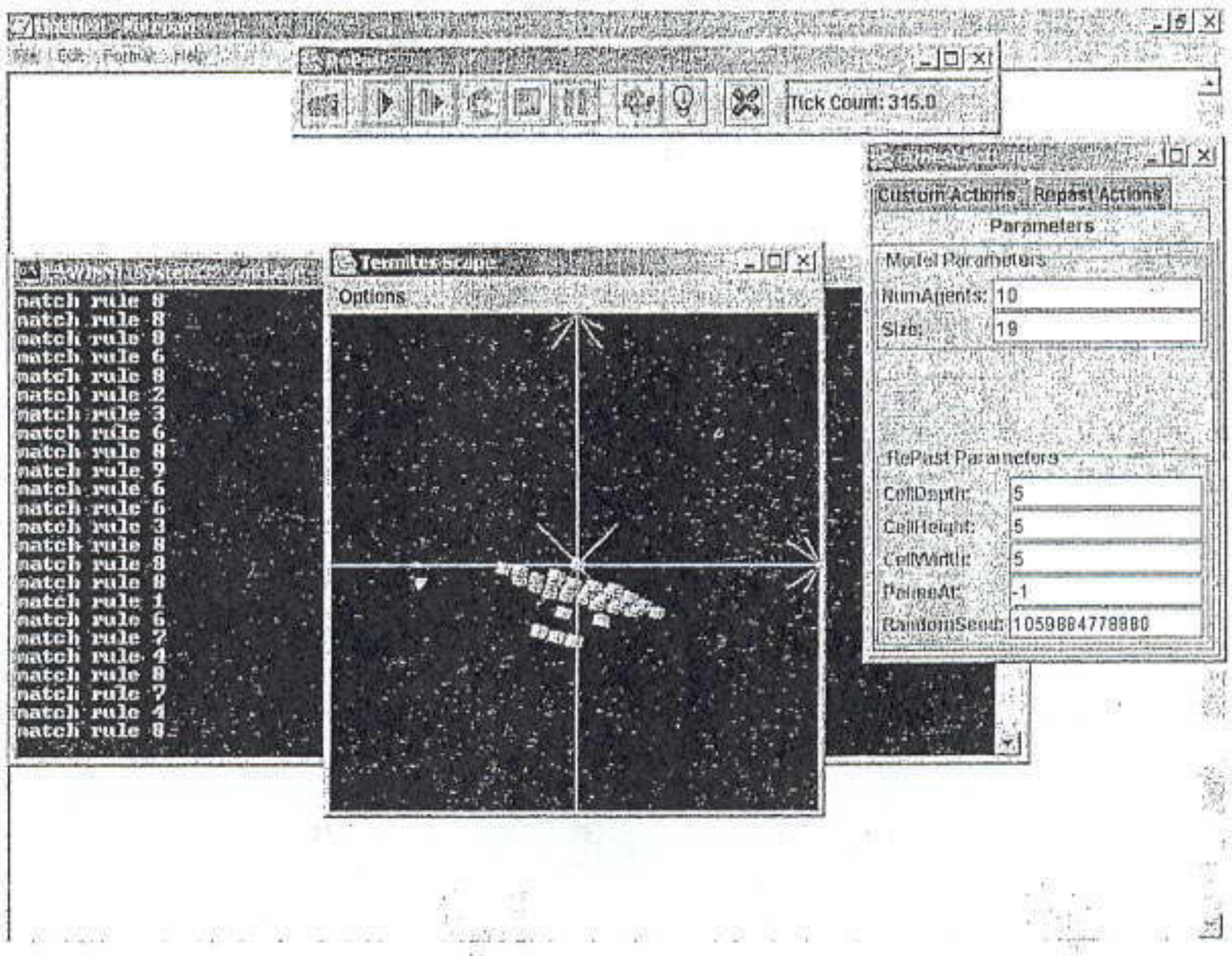


Fig. 21

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

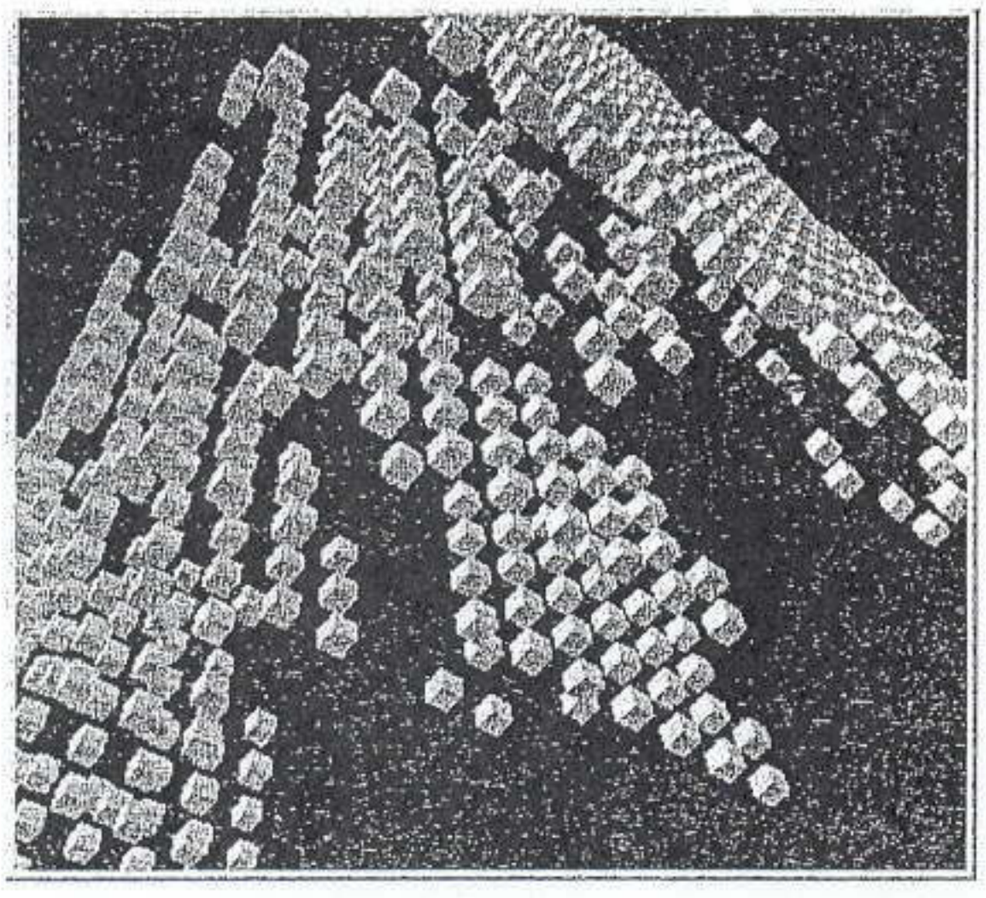


Fig. 22

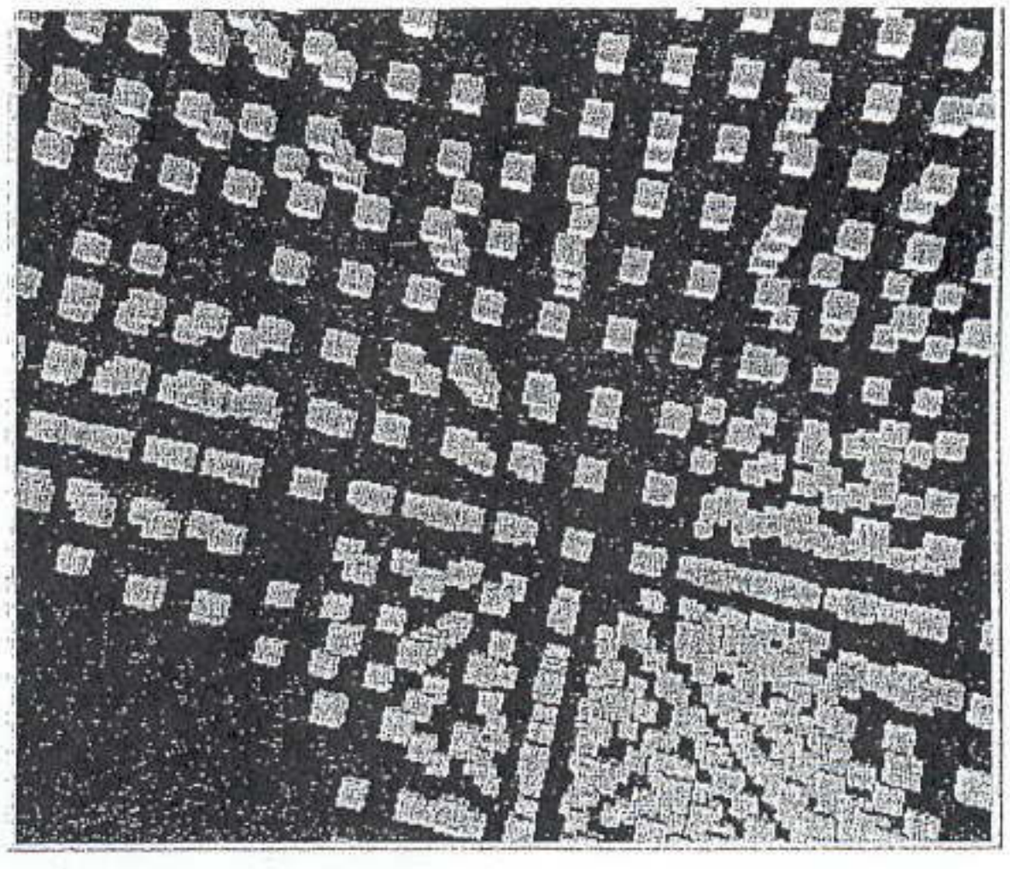


Fig. 23

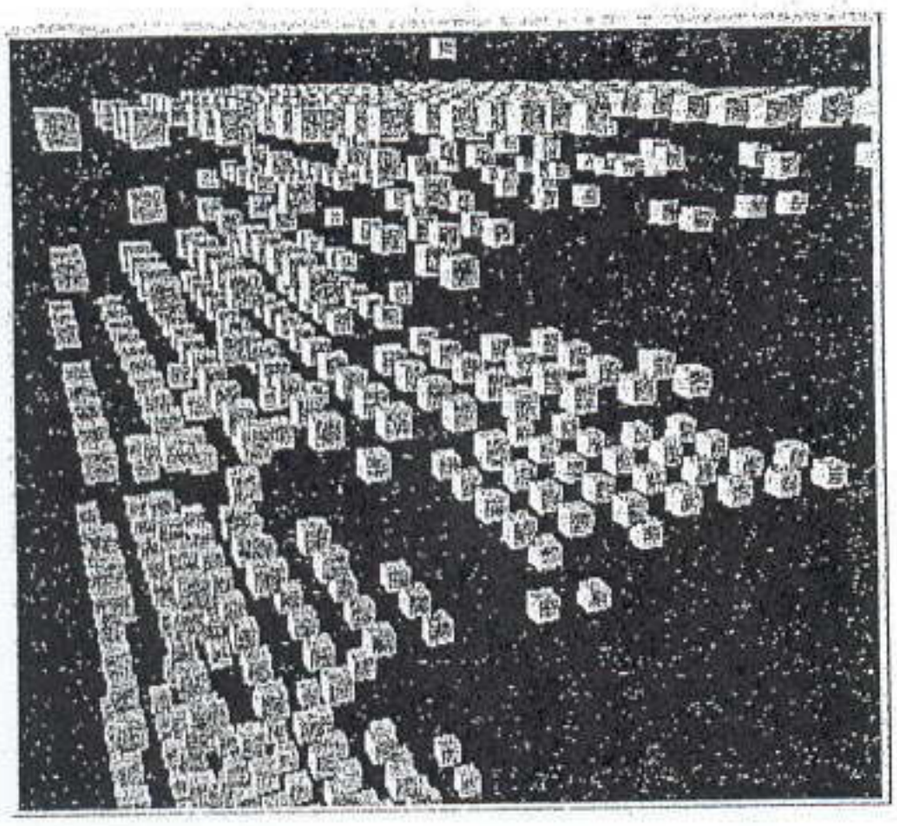


Fig. 24

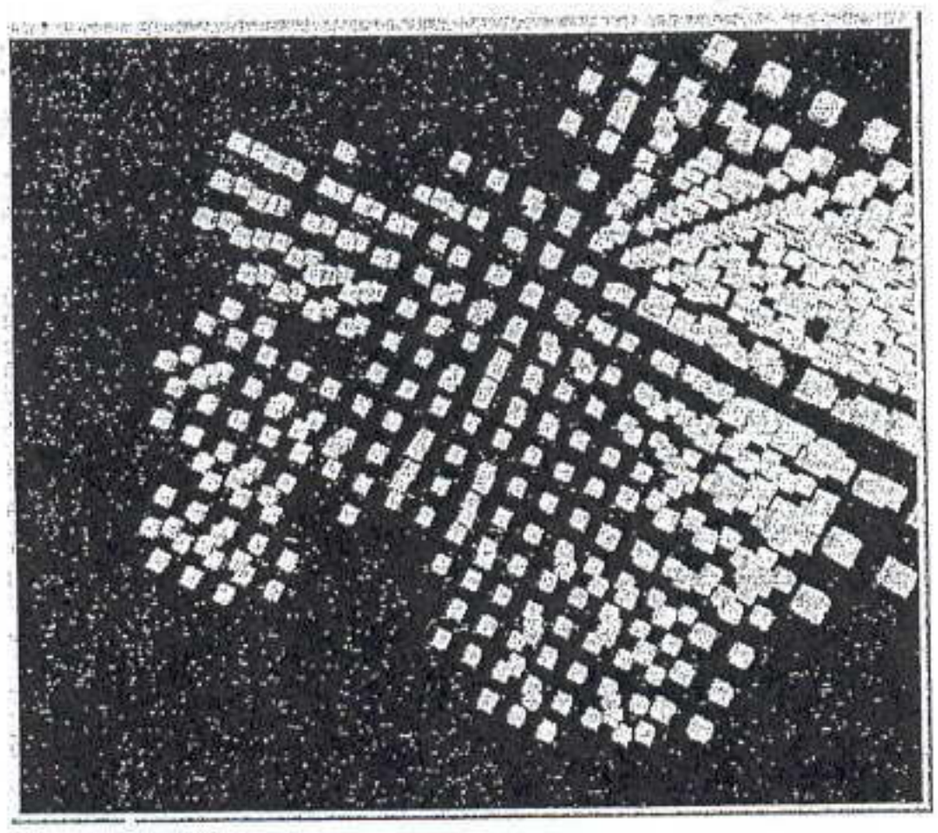


Fig. 25

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.

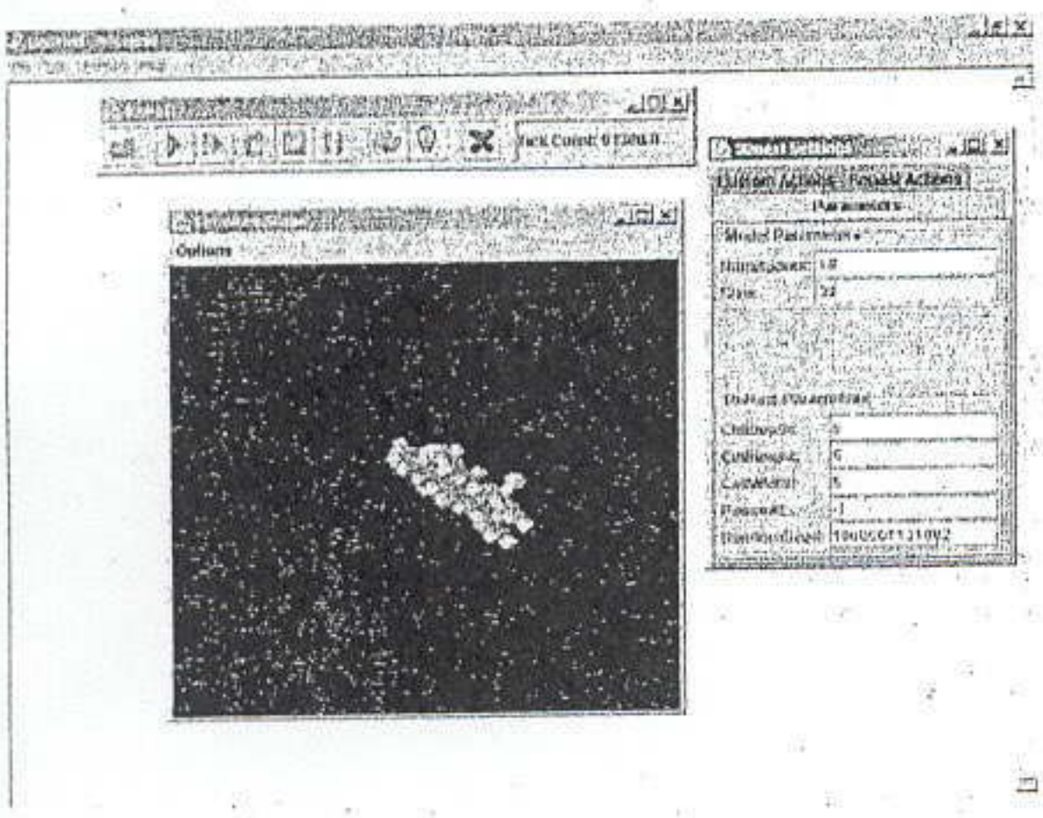


Fig. 26

Comparing our simulation results with that of the Wasp Nest Building Simulator, as shown in figure 27 and figure 28, it can be seen that both simulations have resulted in interesting patterns which closely match those found in nature [1, 19]. On the other hand, comparing our simulation results with these of Bonabeau's, one can see that our software runs smoothly and relatively fast, even on a PC, despite that Bonabeau believes that "extensive simulations on a powerful computer have to be performed in order to explore the behavioral space in a satisfactory manner, even in this simplest case"[1]. However, one should not underestimate the dramatic capabilities of PC's in this time and age. 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.

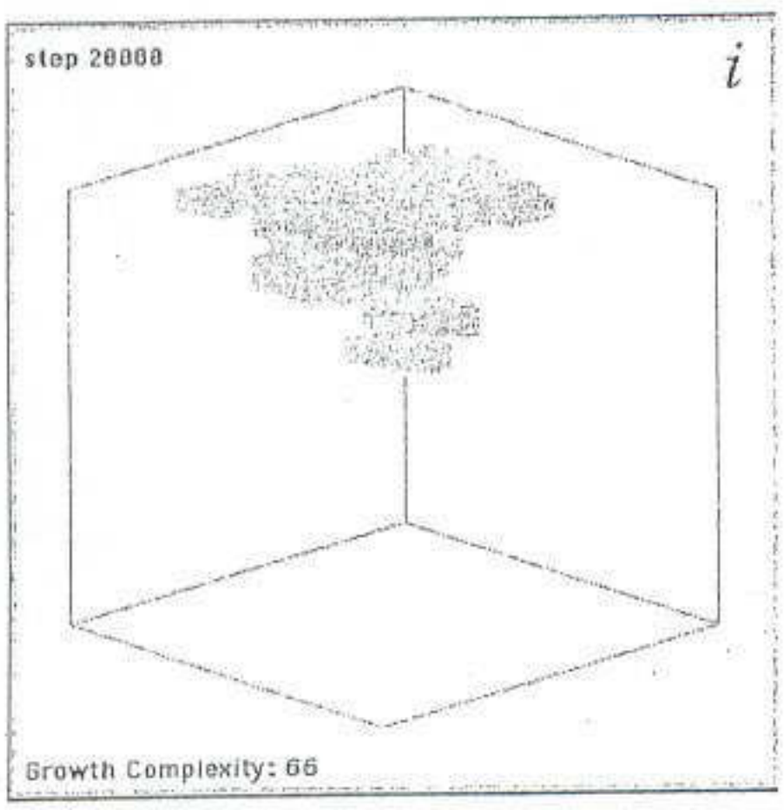


Fig. 27: result from Wasp Nest Building Simulator

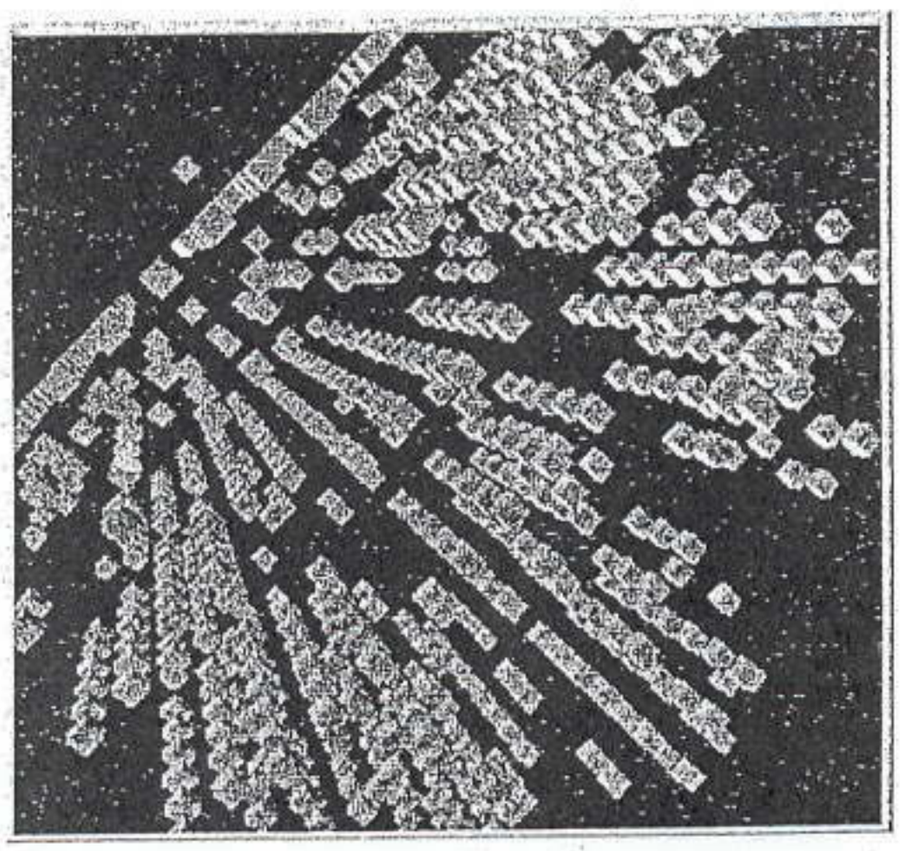


Fig. 28: our simulation result

Laude, Highest Honors in Computer Science. She is also the recipient of President's Award and Dean's Award from University of Bridgeport.

She is a member of IEEE, ACM, Phi Kappa Phi (The Honor Society) and Upsilon Pi Epsilon (The Computing Honor Society). She is also the president of Upsilon Pi Epsilon University of Bridgeport chapter and Vice-president of Phi Kappa Phi University of Bridgeport chapter.

Bei Wang is a 2003 Phi Kappa Phi award of excellence recipient, 2002 Upsilon Pi Epsilon Microsoft Scholarship Award recipient and 2002 Sigma Xi Grant-in-Aid of Research Recipient. She also receives Full tuition Academic Scholarship from University of Bridgeport.

Bei Wang's major research interests are Artificial Intelligence/Robotics and Computational Biology.

Dung Hoang is a senior at University of Bridgeport, pursuing a BS in Computer Engineering. He was invited to be a member of Phi Kappa Phi (The Honor Society) and Upsilon Pi Epsilon (The Computing Honor Society) in the year of 2003. He is elected for inclusion in the 2003 edition of WHO'S WHO AMONG STUDENTS IN AMERICAN UNIVERSITIES AND COLLEGES. Dung Hoang's major research interests are E-commerce and Robotics.

Idris Daiz is a senior at the University of Bridgeport. He is majoring in Computer Science with a minor in mathematics. Idris chose to study Computer Science because he feels that programming is a form of art. Programming allows individuals to create or even invent new software that they could manipulate artistically to their own desires. He personally likes to think of programming as a fun hobby. "It is exciting to see your personal creation to work properly and be utilized by other people".
Idris Daiz receives a Full Tuition Academic Scholarship from the University of Bridgeport. He is also a member of Mu Alpha Theta, which is the National Math Honor Society in the U.S.

Chiedu Okpala is a senior at University of Bridgeport, pursuing a B.S in computer science.

Professor Tarek M. Sobh received the Ph.D. and M.S. degrees in Computer and Information Science from the School of Engineering, University of Pennsylvania in 1991 and 1989, respectively, and the B.Sc. in Engineering degree with honors in Computer Science and Automatic Control from the Faculty of Engineering, Alexandria University, in 1988. He is currently the Dean of the School of Engineering at the University of Bridgeport, Connecticut, U.S.A.; the Founding Director of the Interdisciplinary Robotics, Intelligent Sensing, and Control (RISC) laboratory and a Professor of Computer Science, Computer Engineering, Mechanical Engineering and Electrical Engineering.

He was the Interim Chairman of Computer Science and Computer Engineering and the Director of External Engineering Programs at the University of Bridgeport. He was an Associate Professor of Computer Science and Computer Engineering at the University of Bridgeport from 1995 -- 1999, a Research Assistant Professor of Computer Science at the Department of Computer Science, University of Utah from 1992 -- 1995, and a Research Fellow at the General Robotics and Active Sensory Perception (GRASP) Laboratory of the University of Pennsylvania from 1989 -- 1991. He was the Chairman of the Discrete Event and Hybrid Systems Technical Committee of the IEEE Robotics and Automation Society from 1992- 1999, and the Chairman of the Prototyping Technical Committee of the IEEE Robotics and Automation Society from 1999-2001. His background is in the fields of computer science and engineering, control theory, robotics, automation, manufacturing, AI, computer vision and signal processing.

Dr. Sobh current research interests include reverse engineering and industrial inspection, CAD/CAM, active sensing/imaging under uncertainty, robots and electromechanical systems prototyping, sensor-based distributed control schemes, unifying tolerances across sensing, design, and manufacturing, hybrid and discrete event control, modeling, and applications, and mobile robotic manipulation. He has published over 100 journal and conference papers, and book chapters in these and other areas. Dr. Sobh edited or co-edited issues of several international research Journals in these areas. He has been on the program committees of several international conferences and has chaired and organized several conferences, sessions, workshops, and tracks in Robotics, Automation, and Sensing meetings and has made many presentations, invited talks, invited lectures and colloquia, seminars, and panel participations, at research meetings, University departments, research centers, and companies.

Dr. Sobh is active in consulting and providing service to many industrial organizations and companies. He has consulted for many companies in the U.S., Switzerland, India, Malaysia, Dubai, and Egypt, to support projects in robotics, automation, manufacturing, sensing, numerical analysis, and control. He has also worked at Philips Laboratories in New York, and a number of companies in Egypt.

Dr. Sobh has been awarded many grants to pursue his work in robotics, automation,

manufacturing, and sensing. Dr. Sobh is a Licensed Professional Electrical Engineer (P.E.), a Certified Manufacturing Engineer (CMfgE) by the Society of Manufacturing Engineers, a Certified Professional Manager (C.M.) by the Institute of Certified Professional Managers at James Madison University, a Certified Reliability Engineer (C.R.E.) by the American Society for Quality Control, a member of Tau Beta Pi (The Engineering Honor Society), Sigma Xi (The Scientific Research Society), Phi Beta Delta (The International Honor Society), and Upsilon Pi Epsilon (The Computing Honor Society). Dr. Sobh was the recipient of the Best Research Award by the World Automation Congress in 1998.

Dr. Sobh is a member or senior member of several professional organizations including; ACM, IEEE, IEEE Computer Society, IEEE Robotics and Automation Society, IEEE Computer Society Technical Committee on Pattern Analysis and Machine Intelligence (PAMI), the International Society for Optical Engineering (SPIE), the National Society of Professional Engineers (NSPE), the New York Academy of Sciences, the American Society of Engineering Education (ASEE), the American Society of Quality (ASQ), the American Association for the Advancement of Science (AAAS), Society of Manufacturing Engineers (SME), and a founding member of the Society for Industrial Computing.