MIKROKOSMS AND ROBOTS[2,3]

by

Leonard Uhr and Manfred Kochen
U. of Wisconsin U.of Michigan
U. S. A.

Abstract

This paper presents a computer program that simulates situations (called "MIKROKOSMS") in which severl entities (called "organisms") wander around in an environment that includes one another plus other, apparently simpler, entities (called "objects"). The program has a rather simple set of "laws" of objects, which can be thought of as the laws of physics of the MIKROKOSM. It also has other specifications for organisms - their input and output functions (called "perception" and "response"), their reward functions (called "motivation" or "needs"), and their mechanisms for building up internal memories (called "learning" and "hypothesis formation") that will help them to recognize objects in the future, and respond appropriately to them (for example, in order to maximize expected rewards). This program lays bare the processes that are needed to handle interactions among simulated organisms and objects, including the learning of hypotheses that will guide future action. The present program has only the simplest of pattern recognition, hypothesis formation and need-satisfaction capabilities. Its purpose is to make clear and concrete how such things can be interrelated in a complete system.
Descriptors: Robots, mikrokosms, computers, integrative systems, pattern recognition, learning, hypothesis-formation.

Background

Several relatively simple and special-purpose precursors to MIKROKOSMS have been reported in the literature. Toda[16] discussed the decision-making problem of organisms that move at some cost in energy in order to get energy-giving "mushrooms." Doran[1,2] also examines very simple simulated organisms moving through a space. Travis' simulation and discussion of problem-solving in the form of a chess knight moving in order to cover a board can be thought of from this point of view[17]. Indeed, all game-playing and theorem-proving programs can be thought of as simulating the pieces of the game or the expressions of the logistic system as they move about the environment of the gameboard or the proof tree. (For examples, think of Samuel's checker player[14], Greenblatt's chess player[4], Zobrist's GO program[20], Gelernter's geometry theorem prover[3], Newell and Simon's logic theorist[9] . These programs become more pertinent when they attempt to be general over several games, for example, Newell and Ernst's General Problem Solver[8], Pitrats[12] and Newman and Uhr's[10] to-some-extent-general game players.)

But none of the above seem to be exploring the issues for which the MIKROKOSM programs were developed (Kochen and Uhr[5]). They do not try to tear apart what we mean by environments, objects, organisms, hypotheses, learning, and so on. They do not try to vary, and to generalize about, these things. Rather, they look at relatively peculiar and rigid spaces (a checker board, a logistic system), and their "organisms" (which contain little in the way of perceptual systems, memory, or learning abilities) are rather ad hoc to their space (e.g., chess pieces, logical expressions).

During the past few years four large "robot" projects have been developed (Pingle et al[11] ; Minsky & Papert[6] ; Raphael[13]; Sutro & Kilmer[15]) and there turn out to be certain similarities between robots and MIKROKOSMS. Robots are real physical objects that wander around in our real physical world. Movies can be made of them; when they bump into walls or people they can leave scars. The objects in a MIKROKOSM are computer simulations that consist of internal representations of numbers and letters, of bits, or of magnetic or electrical impulses - whichever you prefer. They can indeed be photographed if a program monitors them onto a scope; and still another equally trivial program could monitor them by having plastic and metal physical objects moving around through a room, just as the scope monitor has grades of light moving around over a phosphorescent screen.

The differences of "reality, " of physical "hardness" and ability to bump and scar, are differences that we had best leave to the ontologists. But there are important differences that we can understand. MIKROKOSMS force us to understand, or at least to code, our environments and their interfaces with our organisms (which simulate robots); whereas the robot researchers simply stick their real robots into the real world, and thus

[1] For people partial to acronyms, MIKROKOSM might stand for: "Models of Inductive Knowledge in Responding Organisms Konstructed (from CHAOS by Concept-formation, Heuristics and Adaptation to Organization Sensed) from Only Sensation, Memory, and Sweat.

[2] Robots on their way to the moon might be thought of as Real Orbiting Bits of Ostensible Things.

[3] This research was partially supported by NIH Grant, 12977.

(potentially) get all the advantages of real-world complexity into their environments. The cost they pay is the cost and the trouble of building robots (which is something that has turned out to take several years, several people and several hundreds of thousands of dollars). This cost will go down and disappear. They may also pay the more important costs of reduced complexity of the organisms they can handle, and rigid behavior on the part of these organisms. Or at least this seems to be the case to date. Finally, they have not yet actually taken advantage of the complexity of the real world into which they can put their robots. We will return to these issues in the discussion at the end of this paper.

MIKROKOSMS Described

We will now give a brief description of MIKROKOSMS in the following way: I) An actual running computer program (called "MIKRO-I") will be discussed. This program is presented in the Appendix, along with an English-language "Precis" that gives a detailed statement-by-statement description of its processes and its flow of control. 2) Comments will be made about the general function embodied in each section of the program, with special emphasis on how it might be simplified to a bare-bones minimum. 3) We will discuss and suggest ways in which the function might be made more powerful.

The MIKROKOSM programs, at least as we conceive of them at present, break down into the following sections:

I. Initialize parameters.
II. Generate the initial state of the MIKROKOSM.
    A. Generate objects, and put them into the space.
    B. Generate organisms, and put them into the space.
III. Compute environment-organism interaction (for each organism).
    A. Print out the present state of the MIKROKOSM.
    B. Present its updated view to the organism.
    C. Let the organism process this view:
        1. Recognize objects .
        2. Conjecture pertinent hypotheses.
        3. Choose the most highly valued hypothesis.
IV. Update the MIKROKOSM to advance to the next time interval.
    A. Compute and effect physical law changes.
    B. Compute and effect changes resulting from organisms' acts.
    C. Have the organism learn as a function of feedback from eating
    D. Go to step III, advancing time by 1 unit.

Initialization of Parameters

The program must first be given a picture of the kind of mikrokosm space it should generate. This includes the dimensions of the space, the number of organisms and of objects in the space, and the characteristics of the organisms and the objects. This might be done in a very general way, by parametrizing all of these characteristics and having the program choose random values for the parameters. Or it might be done with precise descriptions of all the pertinent details.

MIKRO-I is a simple mixture of easily parametrized and built-in characteristics. The basic parameters (PARAMS) for a space are taken to be its row and column size (all spaces are 2-dimensional), the number of organisms and the number of objects in the total space, and the distance that each organism can see. The program is given a set of standard values for these parameters (7 rows, 20 columns, 3 organisms each seeing 5 columns in either direction, and 4 objects). But the program will attempt to read a data card at run time, on which new parameter values can be given to it.

The objects that will be placed into the space will be exact copies of the general object types (GENOBJS). These are descriptions of the set of points, (that is, the shape) of the objects, and of the physical qualities (such things as color, breakability, motion, weight, and caloric value).

Whereas information about the space is parametrized, and information about objects is stored in tables, information about organisms is embodied in code. [This probably reflects the relative simplicity of the space, and the relative complexity of organisms. *E*< t it probably also reflects the primitive stage of this program: a better program would have better reasons for using parameters, tables, or code.] Essentially, an organism is some calls to subroutines that a) take a look, b) pattern recognize, c) test hypotheses, d) choose acts, and e) generate and learn new hypotheses.

Generation of the Initial State of the MIKROKOSM

The program uses ihe parameters that define the space to compute the average distance between objects and between organisms, and then generates and places the specified number of each. MIKRO-I makes several simplifications to shorten the code and speed up processing time. Some of these lead to peculiar characteristics, but all could be coded properly - in most cases with very little trouble.

Objects. Objects are generated first, by making one specific object from each general object, in turn, until the specified total number of objects has been made. To avoid any need to check whether several objects are put into the same place (though in fact in MIKRO-I space this will happen later, causing peculiarities but no fata] problems), all objects are put into the top row of the space, at distances computed so that the specified total number of objects will be equally distributed. Initially, all specific objects of each general type will be identical, except for position in the space. But as time passes objects may move, bounce, break, and in other ways change in value.

Organisms_, MIKRO-I now generates the specified number of organisms and distributes them uniformly along the bottom row of the space. An organism is simply a name that contains a set of lists with that organism's qualities (energy level, location, direction and force of motion), its hypotheses (after it has learned some), its present view of the space, and some memory as to its own acts and changes it has noted in its environment.

Organisms resemble objects in that they both have physical characteristics; but they differ in that only organisms store and learn hypotheses, view the environment, and have some memory. At present these differences reflect our own primitive thinking on the subject; it will be of great interest to see what differences remain when we have tried to force as much similarity as possible.

There are several major peculiarities to the space of objects and organisms generated up to this point. There is insufficient space to describe discuss, and justify them all in detail, but briefly: Although objects have shapes in 2-dimensions, they are stored as though at points. This avoids the necessity of detecting and computing overlap between objects, which would either force us into 3-dimensional space (which would be expensive of memory and processing time) or a peculiar physics of objects occupying the same point at the same time. The proper way to handle the space would be for all objects to reside, properly positioned, in a background of emptiness or noise (chaos). But this would introduce a number of processing steps that would make this aspect of the program far more sophisticated than needed for the low level of sophistication of the rest. So we store objects and organisms on lists, and designate their positions explicitly by the coordinate numbers, rather than implicitly by their actual positions in the framework of the space.

Computation of Environment-Organism Interactions

Printout. The program outputs the present state of the mikrokosm before it begins the set of object-organisms interactions that will lead to change, and before each subsequent time period. This printout draws a picture of the current space, with its organisms and objects, and also exhibits the present state and innards of all objects and organisms.

Organism Views its Environment. That part of the total space that lies within its field of view is presented to each organism, in turn. Only the externally-visible shape of objects (including the viewing organism) can be seen, and the position of each object is computed and given to the organism relative to its own position. Changes in these relative positions are also detected and given to the organism.

Pattern Recognition and Hypothesis-Testing. The organism now begins to apply its set of hypotheses to its seen view. [Specific hypotheses might be coded in advance into each organism's hypotheses table, but the alternative procedure that we prefer is to rely upon the organism's learning abilities to form these hypotheses through experiences gained in interaction with the environment.] Tor each object in its view, the organism checks to see if there is an hypothesis in its memory with that object as one of its premises. It is here that the program does a very primitive kind of "pattern recognition, " in which the description of the object must match a template representation stored in the organism's memory.

The interesting thing about this process is the structure of an hypothesis, even in its overly-simplc present form. An hypothesis is a statement about what the organism should do, and subsequently expect, when a certain thing(s) is recognized in the environment and the organism is in a certain state(s). At present the thing recognized is one of our simple and undeviating objects, recognition is effected by a perfect template, the acts are simple built-in sequences of code, and the expectations are of a certain amount of positive or negative change ("pleasure" or "pain"). But we feel justified in suggesting that it is fine to keep pattern recognition as simple as possible, when we have so many other problems in the total system, and we know a great deal about how to make more sophisticated pattern recognizers. An especially appropriate type of pattern recognizer will be the sort of flexible heirarchical compound learning program discussed in Uhr[19] . We should also note that the recognition of more complex compounds of varying internal states within the organism can be handled in exactly the same way as external recognition, for these are merely two (sets of) premises of an hypothesis. The issue of "planning" is captured in the setting up and learning ot sequences of acts for the organism to do, which form the consequences of an hypothesis, and also sequences of hypotheses that the program might act upon over time. It is in the individual hypothesis and the methods for choosing among hypotheses, that much of what we call "pattern recognition, " "planning, " and "problem-solving" lie. MIKRO-I handles these problems in the grossest and most primitive ways, but it also makes painfully and hopefully clear how our mechanisms for hypothesis-application and hypothesis-formation can be improved.

An hypothesis is conjectured to apply because something in the environment was found to be one of its premises, and hence implied it, and then the organism found that its own present state satisfied the internal state premised by the hypothesis, and finally computed a positive expected value of this hypothesis (which is a function of expected return from the object involved, the object's distance: a rough measure of difficulty of applying the hypothesis, and the weight of assurance with which this hypothesis is held). After the program has found all such pertinent hypotheses, it chooses the one with the highest positive weight. [A better program would

allow an organism to do more than one thing at a time, to decide what to do before it had spent all the time needed to examine everything, and to make decisions as a function of sets of mutually complementary hypotheses. A simpler program might ignore internal states, and merely choose a single act as a function of a single recognized object. Such a program would be doing straightforward pattern recognition. The program might merely have one act, rather than sequences of acts, stored as the consequence of each hypothesis. It would then be incapable of building up coherent strategies or plans. It is at this point, when a program is given the ability to build up sequences and compounds of characterizers of environments and of internal states, and of acts and expectations, that it begins to build up the capability of doing interesting pattern recognition, concept formation, serial prediction, problem-solving, and planning.]

Up-dating the MIKROKOSM to Advance to the Next Time Interval

After all organisms have been given the chance to interact with their perceived environments, the UPDATE routines change the MIKROKOSM, as indicated by two lists: a) the physical qualities of all the organisms and objects, and b) the UPDATE list of the organisms' chosen acts.

Physical Changes. The only physical quality that is examined by MIKRO-i is the object's motion. The object is moved as indicated and, if it has hit a border of the space, is kept at that border . [A better program would allow objects to accelerate in their motion, and to change directioa for example after hitting one another. The program should also compute other changes in objects, for example decay and cracking over time, and changes that are a function of interactions between objects, for example breaking or fading.]

Organisms' Acts. The acts that the organisms chose to put into UPDATE are now carried out. An act is a sequence of one or more things to do, and each such thing names a subroutine that does it. At present, the program has subroutines that allow the organism to: 1) GET (move l step toward the specified object), 2) DESTROY (wipe out the specified object if it is at the same location as the organism), 3) FLAIL (move l step in a randomly chosen direction), and 4) EAT (wipe out the specified object, and also add the object's specified caloric content to its own internal energy level). This is a rather arbitrary set of primitive acts, chosen primarily to get MIKROs started doing something. But note that some are binary (DESTROY, EAT) and some unary (FLAIL, GET), and that string of acts can be compounded together.

Learning. EAT is of special interest because only here is there real feedback to the organism, when its energy level is changed as a function of the object's caloric value (which can be negative, denoting a noxious object). It is therefore here, and here only, that the organisms in MIKRO-1 "learn." [A more sophisticated program would have learning occur from a wider variety of feedback information - for example, from noting that objects are getting closer, or that a certain sequence of acts (as stored in the organism's running memory of past-done things) resulted in desirable or undesirable consequences. In MIKRO-i there is only one overall "energy level, " which is changed when an object is eaten and its "caloric content" adds to or subtracts from that level. There might be several components of proper functioning - eg. protein, fat, water, oxygen, touching - and there might now be a need to develop characterizers about the self's patterning of these components, and changes in this patterning.]

When the organism's energy level is changed as a function of eating an object, the program hunts through the past done acts of that organism to find an hypothesis that led to acts with the eaten object as their object. Two types of hypothesis are stored on the organism's past-done list: the single hypothesis chosen at each time interval, and the rest of the hypotheses conjectured. If no chosen hypothesis turns out to be about the object eaten, the organism goes through its conjectured hypotheses and, if one is found about that object, raises the value of that hypothesis, so that, since it is pertinent, it will more likely be chosen the next time. [This is a rather arbitrary thing to do, and should not be taken very seriously; it merely demonstrates how easy it is to program in variant types of learning.]

The first pertinent hypothesis that is found is then up-weighted or down-weighted, depending upon whether the object eaten was positive or negative (noxious) in caloric content. [A better program might change the weight as a function of the actual caloric value, as well as its sign.] If down-weighted, the new weight is examined to see if it has fallen below some acceptable minimum. If it has, the hypothesis is discarded. [MIKRO-1 arbitrarily sets the initial weight of an hypothesis at 5, and discards the hypothesis when its weight goes below l. A more sophisticated program might keep a better record of the good and bad consequences of an hypothesis, and run more sensitive statistical tests to see whether the hypothesis has proved itself good or bad over a sufficiently large sample.]

MIKRO-l generates a new hypothesis a) when an hypothesis is discarded and b) after each time period. [It would be better to keep some count of the amount of space available for the organism's hypotheses - its memory size - and then eliminate the least valued hypotheses when there was not enough room for everything. Alternately, the

organism's level of functioning and improvement rate on this level might be stored, with hypotheses being discarded and generated as deemed best to improve further. A number of extremely interesting aspects of learning pop up at this pointj

The variety of new hypotheses MIKRO-I can generate is limited. If the caloric content of the eaten object was positive, the consequent acts of the hypothesis will be "get, eat:" if negative, they will be "get, destroy." The object's description is stored as one premise, so that the simplest kind of whole-template pattern recognition is learned and used. The initial expectation and weight of all hypotheses are set at the same level-90 and 5. The program checks whether a newly-generated hypothesis has already been discarded in the past; if it has, it doesn't bother to generate it again. [This is a subtle issue; if the mikrokosm were noisy or changed over time, organisms should be able to try hypotheses again. On the other hand, if there is a large set of possible hypotheses, learning might be slowed dangerously if the organism tried again hypotheses' it had already found worthless.]

Printout

The program listing ends with the code that prints out the state of the mikrokosm after each time period. This printout is cumbersome, and need not be described in detail. Essentially, the actual Z-dimensional representation of the mikrokosm is printed as a matrix, all organisms and objects that still exist are listed, and the contents (the internal states) of these organisms and objects are given.

Discussion

MIKRO-I is already too complex to lay bare the minimal structure of a mikrokosm, yet it is far too simple to be very convincing or interesting. Our intent is that this is the first in a continuing series of more complex and more sophisticated programs. These will look into each of the aspects of a mikrokosm separately (pattern recognition, hypothesis formation, heirarchical compounding, planning, inductive learning, discovery, and so on). It should also be instructive to simplify, to try to get at the essence of, mikrokosms by: a) getting at their bare bones, and b) generalizing, eliminating as much as possible of the built-in.

Motivation

There are a number of reasons why MIKROKOSMs seem interesting things to worry about and to simulate.

A) They force those of us interested in modeling intelligent processes (whether "artificial" and/or "natural") to take into account all aspects of our problem, and to contend with the central issue of integrating into a visible whole the various functions that we typically study separately. On the one hand many of us (including the authors) argue that our problem is far too big, so that we should simplify as much as possible; and that, further, not until we have far larger computers will our models exhibit diverse, flexible, and interesting behavior, which we feel may to a great extent be a function of size of memory. So we separately study "pattern recognition" or "concept formation" or "verbal learning" or "problem solving" (e.g., "game playing," "theorem proving," "serial prediction") or "decision functions" or "responding" (moving a "hand" or an "arm" or a "robot"). This is, we hope, right and proper. But we should also begin to combine these various functions. They are merely pieces of a total program, and we had better start worrying about whether we can ever get those pieces working efficiently together.

B) By putting the different functions together at the precise level of computer code, we gain the opportunity to generalize across them. We should force ourselves to do so. One gets the impression from papers and talks about some of the "robot" projects that they intend to take a pattern recognition program, and a question-answering program, and a concept formation program, and a theorem proving program, and maybe a few other programs, and put them together into a Frankenstein monster. On the contrary, we should use this opportunity to tear apart and try to understand our code and the functions we are trying to compute, so that general-purpose routines are achieved. For example, almost certainly most if not all of "pattern recognition" and "concept formation" should be performed by a single subroutine. Those aspects of pattern recognition that involve deductive inference should have subroutines in common with problem-solving. And so on. We must confront ourselves with general criteria for theory-building: keep the set of constructs as simple, elegant, non-redundant, powerful, and insight-producing as possible.

C) The MIKROKOSM situation raises some extremely interesting new questions of generality. Tor example, when we work with typical pattern recognition or artificial intelligence programs the patterns or other external objects with which they interact are merely presented to them. But now we must describe objects in the same kind of computer code with which we describe our perceiving, problem-solving organisms. We can now try to describe both objects and organisms with the same subroutines and the same tables, and we can begin to ask how they are similar, and how they differ. This focuses us on the fascinating issue of what turns an object into an organism. It prepares us to wonder about how a self-organizing system of objects might begin to evolve some organisms. It forces us to think about what must minimally be given to a MIKROKOSM for it to contain, or evolve, objects

among which will be organisms.

D) We can hold one or more parts of the total MIKROKOSM constant or trivially simple, vary the rest, and examine the resulting behavior.

E) We can ask a number of questions about learning that cannot be so clearly asked in any other situation. We can ask what is a minimal organism that will learn; what kinds of environmental experiences must this organism undergo; what kinds of information must this organism store. Since we have close control over both environment and organism (at least until they start to interact and to learn), we can constantly think about one with respect to the other, and we can continually try to simplify both.

F) We can ask two closely related questions that have simply not been asked before at the precise level of computer modelling: 1) How do two or more organisms learn to talk, including the development of vocabulary, grammar, and semantic reference, and the development of the mutually understood convention that these things should be developed, and should have common meaning to all organisms belonging to that linguistic community? 2) How do two or more organisms come to compete, and to cooperate; and what is the relation of such social behavior to a) their basic needs and ways in which they can be satisfied on the one hand, and b) their development of language on the other hand?

G) We can try to simplify to the point where we may pinpoint what is absolutely essential. For example, it is not clear whether such a MIKROKOSM must have a motivating force in the form of the organisms' internal needs. We keep being forced back to such a beginning, or some close equivalent, such as a diffuse curiosity or a neural itch, despite the fact that this somehow, in a very vague way, feels ad hoc and intuitively unsatisfying.

MIKROKOSMS and Robots

The Stanford University robot project (Pingle et al, ") has developed a robot that (at least so far) can do the following: if several large wooden blocks are scattered on a table, a robot hand with several fingers will zero in to hover above each block, close its fingers around that block, pick it up, and put it on top of a tower of like-sized blocks it is building. It thus perceptually sorts out the blocks by size, positions itself to pick up a block, and then places the block very carefully, so that it sits square enough on top of its tower so that the tower does not too often come tumbling down. This involves some extremely complex matters of precise positioning of fingers around blocks, and of blocks on top of other blocks. These problems may well be horrendous, if not impossible; but unfortunately this may be the case because they are being handled with several built-in strikes against the robot. For it gets no feedback about, and has no ability to adapt to, slight variations in position; it is in the same unfortunate position as a guidance system that must compute its trajectory to hit the moon without any opportunity for subsequent self-correction from feedback as to its deviations.

The Stanford Research Institute robot (Rosen, Nillson, Raphael; see Raphael[13]) can wander through a room that has irregularly shaped objects placed in it, and learn through experiences of bumping into objects how to avoid them, and finally wander through the room without bumping into these objects. But its methods appear to be rather rigid and ad hoc - essentially, it stores a graph-paper representation of the room, and it fills in squares as "containing object, so avoid" whenever either its TV camera "eye" or its bumpers find a block covering the corresponding section of the room.

Both of these robots do things that MIKROKOSM organisms do riot do, and they force the researcher to contend with problems that MIKROKOSMs avoid. It is up to the individual researcher's interests and tastes which he decides to be the more important problems, or the problems most central to modelling of intelligence. Robot projects must contend with the noises introduced by TV camera inputs and by mechanical contraptions that can't move around without joggling their TV eyes, stop suddenly without randomly overstepping their intended stopping-point, or place one finger on an object without moving that object slightly so that the computed position for the next finger is no longer correct. New and interesting pattern recognition problems are confronted when the images of possibly interposed 3-dimensional objects on a table or in a room, with their shadows and gradations of intensity, must be recognized. These are problems the real world environment forces upon one, and it is good to be confronted by the real world.

A MIKROKOSM simulation could be expanded to handle 3-dimensional partially-viewed, shaded patterns, and these are large problems of pattern recognition on which research should be done. But most pattern recognition researchers would, we think, agree that such research should be done with computer programs and special-purpose pattern recognition computers, not with robots. In fact the actual pattern recognition programs incorporated into these robots are at a very low level of sophistication compared with existing pattern recognition research (see, e.g., Uhr[19], Nagy[7]).

It would be more difficult to simulate problems of bouncing objects and falling towers; but it is hard to judge their centrality.

The above is intended to suggest that robot researchers may be tackling difficult problems that are not really their central problems - the mechan-

ics of putting a robot together; the issues of illumination, contrast, color, and noisiness of patterns input; and the mechanical problems of exact motion and placement - andare simplifying the artificial intelligence aspects of their robots to the point where they are actually less sophisticated than existing computer simulations, including functions, that have been put into our simple MIKROKOSM organisms. But it is not at all intended to criticize the intent of robot research, which we take to be to confront artificial intelligences with the enormous and, intriguingly, inexhaustible and infinite complexity of the real world.

On the contrary, it seems to us crucial to at some point put our simulated organisms into the real world. The complexity of the real world may be necessary for adequate learning; there may even be something to the infinite variety of the real world that is fundamentally unprogrammabie, and we will never know and benefit from this until our learning programs are hooked into the real world. (We assume that good artificial intelligences will be learning programs - they must be able to adapt, and they will be far too complex for anyone to succeed in formulating, or programming.)

But there is a lot to discover about our organisms before they are ready to be put into the real world. People who don't have the mechanical bent skills, or money to build robots can still play an important role in this aspect of robot research. The interfaces between our organisms and the real world almost certainly should be richer than those developed so far in the robot projects . For example, the lack of ability to continually monitor feedbacks and modify behavior accordingly puts existing robots at a tremendous disadvantage. And the ""real world"" has to be an interesting part of the "real world." To what extent is a robot confronting the real world when the room it is in contains two sizes of blocks, sufficiently illuminated so that they can be psrfectly resolved through the input device used? This is merely a world of blocks and positions, even simpler than the simulated worlds of MIKROKOSMs .

Summary

This paper describes and discusses a simple program, "MIKRO-L," that simulates the interaction between a set of "objects, " some of which are "organisms, " wandering around within a little "mikrokosm" space'. This is the first of what we hope will be a series of programs to explore complex intellectual processes by combining the various functions (pattern recognition, concept formation, problem solving, decision making, remembering, learning, and hypothesis formation) that have typically been studied separately.

The basic structure of a mikrokosm appears to be the following: it must include a set of objects, some of which are organisms, that interact within some space. There must, therefore, be ways of describing or generating objects and organisms, and placing them in interrelations. Both objects and organisms must have qualities that are interpreted according to "physical laws" of the mikrokosm; among these will be n-ary qualities that are functions of interactions (e.g., "breaking," "eating "). Organisms must be able to generate acts as a function of interactions. In particular, sensory interaction (seeing at a distance) and hypothesizing and learning (generating and storing sets of premises as to the states of the external environment and of the internal characteristics of the organism) seem necessary for any interesting variety of organism behavior. The mikrokosm must be able to collate all the changes of organisms and objects, and up-date itself to the next time period.

MIKRO-L contains a special kind of object, a "shout, " that is located everywhere. This we take to be the basis of future language learning. But MIKRO-I makes no use of shouts. Nor does it have a very sophisticated set of basic acts (eat, get, flail, destroy), reasons for learning (the positive or negative caloric content of an eaten object), or pattern recognition ability (whole-template matching). It has rigid methods for forming hypotheses, and it has no ability to put anything - hypotheses, pattern characterizes, or acts - into interesting compounds or heirarchies . But mikrokosms make painfully apparent the need for such improvements, and we think that they are an especially good test-bed within which to construct and examine more sophisticated processors.

APPENDIX

^ABSTRACT FOR MIKRO-I .

A Get PARAMS: ROWS, COLS, NORGS, VIEWORG, NOBJS, OBJDESCRIPTIONS.
GENERATE SPECIFIED NUMBER OF OBJECTS, SPREAD ACROSS ROW L .
GENERATE SPECIFIED NUMBER OF ORGANISMS, SPREAD ACROSS ROW N.

B PRINT OUT THE MIKRO, CHANGES, AND THE ORGANISM'S INTERNAL STATES.

C PRESENT THE NEXT ORGANISM WITH THE VIEW OF WHAT IT CAN SENSE, BUILDING UP A CHANGES LIST FROM LAST TIME.
WHEN NO MORE OGRANISMS, GO TO D.
THIS ORGANISM FINDS ALL HYPOTHESES IMPLIED BY WHAT'S IN VIEW, COMPUTES A VALUE FOR EACH HYPOTHESIS, CHOOSES THE SINGLE MOST HIGHLY VALUED HYPOTHESIS, AND PLACES IT IN UPDATE AND IN ITS OWN PASTDO LIST.
GO TO C.

D I/PDATE THE MIKRO FOR THE NEXT TIME PERIOD:
MOVE EACH OBJECT (INCLUDING THE ORGANISMS) AS SPECIFIED IN THEIR DXY (CHANGE* OF-LOCATION) VALUE.

```
    FOR EACH ORGANISM, DO THE STRING OF ACTS
      SPECIFIED: (INCLUDE MOVING - FLAILING,
      GETTING - , EATING, DESTROYING.)
        REWEIGHT THE ACTED-UPON HYPOTHESIS
          UP OR DOWN.
            IF WEIGHTED DOWN TEST WHETHER IT
              SHOULD BE THROWN OUT.
                IF YES, PUT IT ON DISCARDS LIST,
                  AND TRY TO GENERATE A NEW
                  HYPOTHESIS FOR THIS ORGANISM.
                    WHEN ALL ORGANISMS ARE DONE,
                      GO TO B.
END GO TO A.
```

*PRECIS MIKRO-I. SIMPLEST 2-DIM. ENVIRON. OBJS PUT IN ROW L .

**INITIALIZE THE PROGRAM'S PARAMETERS.

	Statement Number
GO	0
Let GENerateOBJects (GENOBJS) contain a list of the prototype objects and their descriptions.	
Let each prototype object contain a list of its physical qualities.	1-5
DEFINE the function BORD (which keeps objects within borders).	5
DEFINE the function PUTOUT (which outputs information about the state of the mikrokosm after each time period).	6
Let PARAMeterS contain a standard set of parameters.	
READ in a different set of parameters for this run (if given)	7
Get from PARAMS the individual parameters: RWS (RoWS of space);	8
CLS (CoLumnS of space); NORG (Number of ORGanisms); OSEE (Organism's-SEEn-view); NOBJ (Number of OBJects).	9
Let RandomBACKground (RBACK) contain a random list of symbols.	10
Let RANDom contain a random list of I, 0, - I .	I I
Let SHOUT contain the physical qualities of shouts.	12
Let OBJectTYPES contain the names of the prototype objects .	13
Let PRIMDO contain the primitive acts an organism can do before learning.	14

**BEGIN TO GENERATE OBJECTS.

BEGIN	15
Let the average DISTance-between-Objects (DISTO) equal the number of CoLumnS (CLS) divided by the Number-of-OBJects.	
B6	
Is the Number-of-Objects-generated (NO) GreaterThan the Number-of-OBJects-to-be-generated (NOBJ)?	16
Yes - Go to BI.	
No - Let 12 equal I. (puts all objects in row 1).	17
From GENOBJS, get the first TyPe of object and its DEScription, and put it after the REST, at the end of GENOBJS (so will generate one example of each type of object in turn).	18
Add 1 to the Number-of-Objects-generated (NO).	19
Let the column where the next object will be PUT equal the present PUT (which is 0 initially) plus DISTO (the DISTance-between-Objects)	20

*NOTE THAT HERE AND ELSEWHERE THE RANDOM NUMBERS CAN BE USED TO VARY POSITIONS.

Add to OBJS (the list of all OBJectS-generated) this object's name (OBI, OB2, . . .OB(NO)) followed by its TyPe, DEScription, and row (12) and column (PUT) locations.	21
Let this object's name (OB(NO)) contain its LOCation, (I2*PUT) and a description of its physical qualities (stored in TyPe). Go to B6, to generate the next object.	22

**PLACE ORGANISMS.

BI	23
Let DISTance-between-Organisms (DISTO) equal the number-of-CoLumnS divided by the Number-of-ORGanisms .	
Let the column where the first organism will be PUT equal i DISTO.	24
If PUT equals 0, set it equal to I (so 1st organism will be inside).	25
B8	26
Add I to NO.	
Add to ORGS (the list of all ORGanismS-generated) this organism's name (ORG (NO)), where NO are the integers that follow the integers used for objects), the number of this organism (NORG), this organism's name again, and row (RWS, the last row) and column (PUT) locations. (== indicates organism; =* indicates object.)	27
Let this organism's name (ORG(NO)) contain its attributes (ENergy, initial location (RWS*PUT), and an initial description of its physical qualities and its past memory (initially blank)).	28
Subtract I from the Number-of-ORGanisms (NORG)-to-be-generated.	29
Is NORG LessThan I ? Yes - Go to B9.	30
No -Add the DISTance-between-Organisms (DISTO) to PUT, for placing the next organism. Go to B8.	31

**PRESENT ITS VIEW TO EACH ORGANISM.

	Statement Number
B9	32
Add the list of ORGanismS to the end of the list of OBJectS.	
TNEXT	33
(after each Time-NEXT) PUTOUT the present state of the mikrokosm.	
Advance TIME by I .	34
Is TIME GreaterThan 5 ? Yes - END this run.	35
TNI	36
No - Make a Copy-ORGanismS (CORGS) of ORGanismS (ORGS) list.	
B7	37
Get the next Name-of-ORGanism from CORGS. (If no more, Go to UPDATER)	
From NORG, get this organism's STATE, HYPotheses, PASTDOne acts, VIEW, and OLDCHANGES.	38
From STATE, get this organism's LOCation= (ROW, COL).	39
Let the LEFT border of this organism's view be its COLumn minus OSEE (dis tance -in-one -direction-of Organism's -SEEn-view).	40
Let the RIGHT border be its COLumn plus OSEE.	41
BIO	42
Make a Copy-OBJectS (COBJS) of OBJects (OBJS) list,	
BLANK out (erase) LVIEW and CHANGES.	43
♦LIST FOR THIS ORGANISM THE OBJECTS IN ITS VIEW, INCLUDING ITSELF.	
BI 1	44
From COBJS, get the next Name-of-Object (NO), its TyPe (TP), and DEScription. (If no more, Fail to B12).	
For this Named-Object (NO), get its LOCation (ROw * COlumn).	45
Is this object of TyPe 'SHOUT' ? Yes - Go to B17.	46
Is this object's COlumn GreaterThan the LEFTmost column viewed? No - Go to BI 1 (it's not in this organism's field of view).	47
Yes - Is CO LessThan the RIGHTmost column viewed? No - Go to BI 1.	48
♦GET AND USE DISTANCES FROM ORGANISM TO OBJECTS.	
Let RO (Row-of-Object) equal ROW (ROW-of-organism) minus RO (to get row distance from organism to object).	49
Let CO (Column-of-Object) equal COL (COLumn-of-organism) minus CO.	50
BLANK out ROA, COA.	51
B17	52
From this organism's VIEW, get the TyPe (TP), DEScription, Name-of-Object (NO), and its location (ROA * COA). (If no more, Fail to B18)	
Add this TyPe, DEScription, Name-of-OBJect and its location (RO * CO) to LVIEW (the Left-VIEW).	53
Do the old locations (ROA, COA) of this object EQUALS the new, computed locations (RO, CO) ? Yes - Go to BI 1 (no changes).	54
Is this object of TyPe "SHOUT* (which is located everywhere) ? Yes - Go to BI I .	55
B20	56
To CHANGES, add this object (its TyPe, DEScription, Name-Object, and its changes in location (RO-ROA, CO-COA)). Go to B11 .	
BL8	57
To the beginning of LVIEW, add this new object (its TyPe, DEScription, Name-of-Object, and location (RO* CO). Goto B20.	
♦♦THIS ORGANISM NOW APPLIES ITS LEARNED HYPOTHESES TO THE VIEW PRESENTED IT.	
BI 2	58
Let VIEW contain what LVIEW contains.	
Let MAYDO equal PRIMDO (those acts PRIMitively-DOne).	59
BI 6	60
From LVIEW, get the next object's TyPe, DEScription, Name-of-Object, and location (RO ♦CO). (If no more, Fail to B13.)	
If this TyPe and its DEScription are found on the list oi this organism's HYPotheses, get the acts-to-DO, state-of-SELF, EXPECTations, and WeighT. (If not found, Fail to B16.)	6 I
Does this organism's STATE contain the SELF-state specified for this hypothesis? (No - Go to BI 6.)	62
Let VALUE equal the EXPECTed-need-change times the WeighT of this hypothesis, divided by the ROw plus COlumn distance plus 10.	63
Get the absolute VALUE (delete any minus sign (-)).	64
Add this hypothesis to the list of things the organism MAYDO: VALUE, acts-to-DQ Name-of-Object, location (RO * CO), what to EXPECT, the object's TyPe and DEScription, and the SELF state found. *Go* to B16.	65
♦♦CHOOSES TO DO AS MOST HIGHLY VALUED HYPOTHESIS ON MAYDO SUGGESTS.	
BI 3	66
BLANK out VALA, XXA, LM.	
From MAYDO, get the first VALueA and the rest of the information (XXA) about this hypothesis. (If none Fail to B15.)	67

```
                                                    Statement
                                                      Number
G1                                                      105
    Does the organism's COL EQual the
      object's COlumn? Yes - Go to G2.
      No - If the organism's COL is                     106
        GreaterThan the object's CO 1, let
        DY (change in the Y direction) equal
        -1 and Go to G2.
          Or let DY equal +1 .                          107
G2                                                      108
    From OBJectS, get this NORG's ROW
      and COL.
    BORDer this organism (calling the                   109
      function BORD, which will move it
      and keep it within the borders).
    On OBJS, find NORG, and add its new                 I 10
      location (NEWRC).
    Add this location (NEWRC) and Name-                 1 I 1
      of-ORGanism to REPL.
    Change the LOCation stored under                    112
      this Name-of-ORGanism to NEWRC.
      Go to U2, to get and do the next act.
DESTROY                                                 L 13
    Get the NEXT object from OBJectS. (If
      no more, Fail to U2.)
    Add this object to Left-OBJectS.                    I 14
    From NEXT, if the location is NEWRC,                I 15
      immediately following ' =*', get
      Name-Object, TyPe, and DEScription.
      (If fail, go to DESTROY.)
      Succeed - Add this Name-of-Object,                1 16
        as 'DESTROYEDBY' the Name-of-
        ORGanism, to GONEOBJS. Go to U6.
FLAIL                                                   I 17
    From RANDom, get the first two (pseudo-
      random) numbers (called DX and DY),
      and put them at the end of RANDom.
      Go to G2, where these will be used to
      randomly move and flail the organism.
EAT                                                     I 18
    Get the NEXT object from OBJectS. (If
      no more, Fail to U2.)
    Add this object to Left-OBJectS.                    1 19
    From NEXT, if the location is NEWRC,                120
      immediately following the ' =*', get
      Name-of-Object, TyPe, and
      DEScription. (If fail, Go to EAT)
      Succeed - PRINT out '///EAT///'.                  121
    For this Name-of-Object, get its
      CALoric VALue.                                    122
    From this Name-of-ORGanism, get its                 123
      ENergy level, and add the object's
      CALoric value to this ENergy level.
    Add this Name-of-Object as 'ATEBY'                  124
      this Name-of-ORGanism to GONEOBJS.
**RE-WEIGHTS LEARN HYPOTHESES AS A
  FUNCTION OF FEEDBACK FROM EATING.
**FINDS AN HYPOTHESES ABOUT THIS EATEN
  OBJECT.
```

```
                                                    Statement
                                                      Number
    BL^NK out PAST, OPAST, and DEX                      125
    From this Named-ORGanism, get its                   126
      STATE, HYPotheses, PASTDOne,
      VIEW, and OCHANGES (get, but don't
      erase them.)
L3                                                      127
    From PASTDOne, get the next chosen-
      act's VALue, DZ, Name-of-Object-I
      (NOI), location (ROI*COI), Expectations,
      and the REST of the conjectured hypotheses
      for that time period. (If no more, Fail to
      L6.)
    From EXpect, get EXPECT, TyPeI, DES-                128
      criptionI, and SELF.
    Add 1 to PAST.                                      129
    Add the REST to Other-PAST                          1 30
    Does the Named-Object EQUALS the                    131
      Named-ObjectI? No - Fail to L3.
**RE-WEIGHTS AN HYPOTHESIS ABOUT THE
  OBJECT EATEN.
L7                                                      I 32
    Yes - On the organism's HYPotheses,
      if TyPeI and DEScriptionI are found,
      get and remove DoY, SELF, EXPECT,
      and WeighT.
    Add Delta-Expectation (DEX) to EXPECT.              I 33
    WASH (erase) DEX.                                   134
    Is the CALoric content negative (-)?                135
      Yes - Go to L4.
      No - Add 1 to WeighT.                             136
L5                                                      137
    Put back on HYPothesis the components
      of the hypothesis that was taken off by
      statement L7 and reweighted. Go to U6.
L4                                                      138
    Lower WeighT by 1 (this was a noxious
      object that the organism should learn
      not to eat).
    Is the WeighT now LessThan I ? No - Go              139
      to L5 (to put this down-weighted
      hypothesis back onto HYPotheses).
**DISCARD AN HYPOTHESIS WHOSE WEIGHT HAS
  GONE BELOW 1.
    Add to the list of DISCARDS the TyPeI,              140
      DEScriptionI, DoY, SELF, and EXPECT
      of this discarded hypothesis. Go to L8.
*IF NONE OF THE HYPOTHESES ACTED UPON WAS
  PERTINENT, CHECK ALL OTHERS ON OPAST.
L6                                                      141
    From Other-PAST, get the next hypothesis
      (VALue, DoY, Name-of-ObjectI, location
      (ROwI*COH), and Expectation). (If no
      more, Fail to L8)
    From EXpect, get EXPECT, TyPeI,                     142
      DEScription, and SELF.
    Does Namea-Object EQual Named*                      143
      ObjectI? No - Fail to L6.
      Yes - Let Delta-Expectation (DEX)                 144
```

	Statement Number
equal 3 (so that this hypothesis will have a higher expectation, and will more likely be chosen and acted upon, in the future). Go to L7.	
**A NEW HYPOTHESIS IS GENERATED	
L8	145
Is the CALoric content of the eaten object negative (-)? Yes - Go to L9.	
No - Let New-Do (ND) equal "GET, EAT.". Go to LIO.	146
L9	147
Let the New-Do (ND) equal "GET. DESTROY.".	
LIO	148
See if DISCARDS already has this new hypothesis' TyPe, DEScription, and New-Do. (If yes, Go to U6.)	
Add this newly-generated hypothesis to the organism's HYPotheses: its TyPe, DEScription, New-Do, and initial expectation-level (90), and weight (5).	149
U6	150
Erase information about this eaten or destroyed Named-Object.	
Put the Left-OBJectS back onto the start of the OBJectS list.	151
WASH (erase) Left-OBJectS.	152
Erase this (eaten or destroyed) Named-Object from the OBJectS list. Go to U2A.	153
**FUNCTION FOLLOWS TO PRINT OUT INFORMATION ABOUT MIKRO AT EACH TIME PERIOD.	

Bibliography

1. Doran, J.E., Experiments with a pleasure seeking automaton. In: Machine Intelligence 3, (ed. Michie, D.) Edinburgh: University Press; 1968a, pp. 195-216.
2. Doran, J.E., A simulated robot/environment system: progress and problems, 1968b.
3. Gelernter, H., Realization of a geometry theorem proving machine. Proc. International Conference on Information Processing, Paris: UNESCO, 1959.
4. Greenblatt, R.D., Eastlake, D.E., & Crocker, S.D., The Greenblatt chess program. Proc. Fall Toint Computer Conf., 1967.
5. Kochen, M. and Uhr, L., A model for the process of learning to comprehend. In: Information Science (M. Kochen, Ed.) New York: Scarecrow Press, 1965.
6. Minsky, M. and Papert, S., Discussion of MIT robot project. Presented at Int. Pattern Recognition Conf., Puerto Rico, 1967.
7. Nagy, G., State of the Art in Pattern Recognition, Proceedings of the IEEE, 1968, Vol. 56, no. 5.
8. Newell, A. and Ernst, G., The search for generality. Proc. IFIP Congress 65. New York, 1965, pp. 17-23.
9. Newell, A., & Shaw, J.C., & Simon, H.A., Empirical explorations of the logic theory machine. Proc. West. Joint Comp. Conf., 1957, 218-239.
10. Newman, C. and Uhr, L., BOGART: A discovery and induction program for games. Proc. 20th Annual Meeting of the ACM, 1965.
11. Pingle, K.K., Singer, J.A. & Wichmann, W.M., Computer control of a mechanical arm through visual input. Proc. IFIP Congress 68, Edinburgh, 1968, pp. H140-H147.
12. Pitrat, J., Realization of a general game-playing program. Proc. IFIP Congress 68, Edinburgh, 1968, pp. H120-H124.
13. Raphael, B., Programming a robot. Proc. IFIP Congress 68, Edinburgh, 1968. pp. H135-H140.
14. Samuel, A.L., Some studies in machine learning using the game of checkers. IBM J. Res, and Develop., 1959,1, 210-229.
15. Sutro, L.L. & Kilmer, Assembly of computers to command and control a robot. Proc. Spring Joint Comp. Conf., 1969.
16. Toda, M., The design ot a fungus-eater: a model of human behavior in an unsophisticated environment, Behav. Sci., 1962, *I_t* 164-183.
17. Travis, L.E., Experiments with a theorem-utilizing program. Proc. Spring Joint Computer Conf., 1964.
18. Uhr, L. (Ed.) Pattern Recognition, New York: Wiley, 1967.
19. Uhr, L., Pattern Recognition, Problem Solx/ing and Learning. I969 (in preparation).
20. Zobrist, A., A model of visual organization for the game of *GO.* Proc. Spring Joint Comp. Conf., 1969.

```
                    B 5 5 0 0    S N O B O L 3    S Y S T E M

VERSION 3.0
THE DATE IS 10 MAR 1969
COMPILATION STARTED AT  9:32: 7
        =3600
        *PROGRAM MIKRO1
        * MIK-1 SIMPLEST  2-DIM ENVIRON, OBJS  PUT, NO VARIATION, IN ROW 1
        ** INITIALIZE PARAMETERS
     0  GO      GENOBJS  = "A=1*1,1*1,1*1,/R=1*0,/"
     1      A   =  "COLOR=BLUE,BREAK=NO,HT=9,DXY=1*1,VAL=-6,/"
     2      B   =  "COLOR=RED,DXY=1*0,HT=2,VAL=7,/"
     3      C   =  "BREAK=YES,DXY=0*0,VAL=-4,/"
     4      D   =  "DXY=0*1,/"
     5        DEFINE( "BORD(RW,OB,CL,DC)" , "BOND1" , )
     6        DEFINE( "PUTOUT()" , "PUT2" , "LIST0,I1" )
     7      PARAMS  =  "7,20,3,5,4,"
     8        READ    *PARAMS* "."
     9        PARAMS  *RWS* "," *CLS* "," *NORG* "," *OSEE* "," *NOBJ* ","
    10        RBACK   =  "QLXZMPXUV"
    11        RAND    =  "1,-1,0,0,1,0,-1,0,-1,-1,0,1,"
    12        SHOUT   =  "LOC=0*0,/"
    13        OBJTYPES  =  "ABCD"
    14        PRIMOD  =  ",FLAIL,EAT,,WASH2,*()"
        **GEN OBJS STARTING IN ROW1 COL1,  IGNORES OVERLAP (ASSUMES DONT)
    15  BEGIN DISTO   =  CLS / NOBJ
    16  B6    .GT( NO , NOBJ  )  /S(B1)
    17        I2  =  "1"
    18        GENOBJS  *TP* "=" *DES* "/" *REST* = REST TP "=" DES "/"
    19        NO  =  NO  + "1"
    20        PUT  =  PUT  +  DISTO
    21        OBJS  =  OBJS  "OB"  NO "," TP "," DES "=*" I2 "*" PUT "/"
    22        $( "OB"  NO )  =  "LOC=" I2 "*" PUT "," STP  /(B6)
        **PLACE ORGS (AT BOTTOM ROW)
    23  B1    DISTO  =  CLS  / NORG
    24        PUT  =  DISTO / "2"
    25        PUT  =  .EQ( PUT , "0" ) "1"
    26  B8    NO  =  NO  +  "1"
    27      ORGS = ORGS "ORG" NO "," NORG ",ORG" NO "==" RWS "*" PUT "/"
        *     ONLY HYP INDIRECT NOW, AS GET BIGGER, MAKE OTHERS TOO
    28    $("ORG" NO) ="EN=100,LOC=" RWS "*" PUT  ",DXY=0*0,$HY" NORG "$//$$"
    29        NORG  =  NORG  - "1"
    30        .LT( NORG , "0" )  /S(B9)
    31        PUT  =  PUT  +  DISTO  /(B8)
        **DONE OBJS,  ORGS, PRESENT VIEW TO EACH ORG.
    32  B9    OBJS  =  OBJS  ORGS
    33  TNEXT  PUTOUT()
    34        TIME  =  TIME + "1"
    35        .GT( TIME , "5" )  /S(END)
    36  TN1   CORGS  =  ORGS
    37  B7    CORGS  *NORG* "," *XY* "/" =  /F(UPDATER)
    38        $NORG  *STATE* "$" *HYP* "$" *PASTOO* "$" *VIEW* "$" *OLDCHANGES*
    39        STATE  "LOC="  *ROW*  "*"  *COL*  ","
    40        LEFT  =  COL  - OSEE
    41        RIGHT  =  COL  +  OSEE
    42  B10   CNBJS  =  OBJS
    43        BLANK  *LVIEW*  *CHANGES*
        * LISTS OBJS VIEWED BY THIS ORG INCLUDING SELF
    44  B11   CNBJS  *NO*  ","  *TP*  ","  *DES* "=" *XX* "/" =  /F(B12)
    45        $NO  "LOC=" *RO* "*"  *CO*  ","
    46        TP  "SHOUT"  /S(B17)
    47        .GT( CO , LEFT )  /F(B11)
    48        .LT( CO , RIGHT )  /F(B11)
        * GET AND USE DISTANCES TO OBJECTS(MOVES ONLY UP-DOWN, RIGHT-LEFT)
    49        RO  =  ROW  - RO
    50        CO  =  COL  - CO
    51        BLANK *ROA*  *COA*
    52  B17   VIEW  TP "," DES "," NO "=" *ROA* "*" *COA* "/" =  /F(B18)
    53        LVIEW  = LVIEW TP "," DES "," NO "=" RO "*" CO "/"
    54        EQUALS( ( ROA "=" COA ) , ( RO "=" CO ) )  /S(B11)
    55        TP  "SHOUT"  /S(B11)
        * A NEW OBJECT IS NOW GIVEN INITIAL CHANGE AS ITS DISTANCE FROM ORG.
    56  B20   CHANGES  =  CHANGES TP "," DES "," NO "=" RO - ROA "*" CO - COA
              "/"  /(B11)
        *PUTS NEW OBJS FIRST
    57  B18 LVIEW =  TP "," DES "," NO "=" RO "*" CO "/"  LVIEW  /(B20)
        **THIS ORG APPLIES ITS HYPOTHESES
    58  B12   VIEW  =  LVIEW
    59        MAYOO  =  PRIMOO
        * OBJS IN VIEW POINT TO HYPS TO ACT ON
    60  B16  LVIEW *TP* "," *DES* "," *NO* "=" *RO* "*" *CO* "/" =  /F(B13)
```

```
61          $HYP  TP ",' DES ",' *DO* ",' *SELF* ",' *EXPECT* ",' *WT* "/"
*             /F(B16)
62            STATE  SELF  ",'  /F(B16)
* VALUE IS FUNCTION OF EXPECTED NEED-SATISFACTION WT AND DISTANCE
63          VALUE = ( EXPECT  * WT ) / ( ( RO + CO ) + "10" )
64            VALUE "=" =
65          MAYDO = MAYDO VALUE ",'  DO ",' NO ",' RO "*" CO "(" EXPECT ","
*           TP  ",' DES ",'  SELF ")"   /(B16)
* CHOOSE TO DO MOST HIGHLY VALUED "MAYDO"
66 B13      BLANK  *VALA* *XXA*  *LM*
67          MAYDO  *VALA* ",' *XXA* ")"  =  /F(B15)
68 B14      MAYDO  *VALB* ",' *XXB* ")"  =  /F(B15)
69            ,GT( VALB , VALA )  /S(B19)
70          LM  = LM VALB ",' XXB ")"  /(B14)
71 B19      LM = LM VALA ",'  XXA  ")"
72          VALA  =  VALB
73          XXA  =  XXB  /(B14)
* PUT ACT WITH HIGHEST EXPECTED VALUE ON UPDATE
74 B15      UPDATE = UPDATE NORG ",' ROW "*" COL "*" XXA ")"
75          PASTDO  *XX* "/" *YY* = YY VALA ",' XXA ")" LM "/"
76          $NORG  = STATE "$" HYP "$" PASTDO "$" VIEW "$"  CHANGES /(B7)
* LETS 2 OBJS OCCUPY SAME SPACE (BETTER TO TEST OVERLAP , BOUNCE THEM)
** UPDATE MIKRO FOR NEXT TIME PERIOD
77 UPDATER    OBJS  *COBJS*  =
** UPDATE FROM OBJECTS' QUALITIES
78 U3       COBJS  *NO* ",' *TP* ",' *DES*  "=" *OX/"1"* *YY* "/" = /F(U1)
79          $NO  "DXY=" *DX* "*" *DY* ",'
80          $NO "LOC=" *RO* "*" *CO* ",'  =
*WILL STICK TO BORDER
81          BORD( RO , DX , CO , DY )
82          $NO = "LOC="  NEWRC  ",' $NO
83          OBJS = OBJS NO ",' TP ",' DES "="  OX  NEWRC "/"
84          REPL = REPL RO "*" CO "=" TP ",'  /(U3)
85 BORD1     NRO = RW + DR
86          NCO = CL + DC
87          NRO = .LT( NRO , "1" ) "1"
88          NRO = .GT( NRO , RWS ) RWS
89          NCO = .LT( NCO , "1" ) "1"
90          NCO = .GT( NCO , CLS ) CLS
91          CL = CL - "1"
92          $( "P" RW ) *XX/CL* *SS/"1"* = XX ","
93          NEWRC =  NRO "*" NCO  /(RETURN)
** UPDATE FROM ORGANISMS CHOSEN ACTS
94 U1       UPDATE *NORG* ",' *ROW* "*" *COL* "=" *DO* ",' *NO* ",' *XX* ")"
*             /F(TNEXT)
95          $NO "LOC=" *RO* "*" *CO* ",'
96          NO  "WASH2" =
97 U2       OBJS =  LOBJS  OBJS
98          WASH  *LOBJS*
99 U2A      DO *DOX* "," =  /S($DOX)F(U1)
**CODE FOR THE SPECIFIC ACTS FOLLOWS
100 GET     BLANK *DX* *DY*
101         SYSPOT = " ///GETS///"
102           EQ( ROW , RO )  /S(G1)
103         DX  =  GT( ROW , RO ) "-1"  /S(G2)
104         DX  =  "1" /(G2)
* ALREADY GOT " RETURN
105 G1      EQ( COL , CO )  /S(G2)
106         DY  =  GT( COL , CO ) "-1"  /S(G2)
107         DY  =  "1"
108 G2       OBJS NORG "==" *ROW* "*" *COL* "/" = NORG "=="
109         BORD( RW , DX , COL , DY )
110          OBJS  NORG "==" = NORG "==" NEWRC "/"
111          REPL = REPL NEWRC "=" NORG ",'
112          $NORG "LOC=" *XX* ",' = "LOC=" NEWRC ",'  /(U2)
113 DESTROY   OBJS *NEXT* "/" =  /F(U2)
114          LOBJS =  LOBJS NEXT "/"
115          NEXT  NO  ",' *TP* ",' *DES* "==" NEWRC /F(DESTROY)
116         GONEOBJS  =  GONEOBJS NO "=DESTROYEDBY " NORG ",'  /(U6)
* FOR RANDOM MOTION (ALSO WANT BABBLE)
117 FLAIL   RAND  *DX* ",' *DY* ",' *XX*  = XX DY "," DX ",'  /(G2)
*     BEGINS DEFINED FUNCTION (RIGHT BEFORE TNEXT)
118 EAT     OBJS *NEXT* "/" =  /F(U2)
119          LOBJS =  LOBJS NEXT "/"
120          NEXT  NO  ",' *TP* ",' *DES* "==" NEWRC /F(EAT)
121          SYSPOT  =  "  ////EAT////"
122         $NO  "VAL="  *CAL*  ",'
123         $NORG  "EN=" *EN* ",'  =  "EN=" EN + CAL ",'
124         GONEOBJS = GONEOBJS NO "=ATERY " NORG ",'
*** LEARN NEW HYPS ****
125          BLANK *PAST* *OPAST* *DEX*
```

```
126          $NORG *STATE* "$" *HYP* "$" *PASTDO* "$" *VIEW* "$" *DCHANGES*
127 L3       PASTDO *VAL* "," *DZ* "," *NOI* "," *ROI* "*" *COI* "("
             *EX* ")" *REST* "/" =  /F(L6)
128 *  EX     *EXPECT* "," *TPI* "," *DESI* "," *SELF*
*       DONT YET USE THIS COUNT OF HOW FAR PAST
129          PAST =  PAST + "1"
130          OPAST = OPAST REST
131     EQUALS( NO , NOI )   /F(L3)
132 L7  $HYP =  TPI "," DESI   "," *DY* "," *SELF* "," *EXPECT* "," *WT* "/" =
133          EXPECT =  EXPECT + DEX
134          WASH *DEX*
135          CAL "=" /S(L4)
136          WT  =  WT + "1"
137 L5   $HYP = TPI "," DESI   "," DY "," SELF "," EXPECT "," WT "/" $HYP
*          /(U6)
138 L4       WT  =  WT - "1"
139          .LT( WT , "1" )  /F(L5)
140     DISCARDS = DISCARDS TPI "," DESI    "," DY "," SELF "," EXPECT "/"
*          /(L6)
*       NONE OF THE HYP ACTED ON CHOSEN, CHECK THE REST ON OPAST
*       IGNORES "$" WHICH TELLS ABOUT TIME PERIOD
141 L6       OPAST *VAL* "," *DY* "," *NOI* "," *ROI* "*" *COI* "("
             *EX* ")" =  /F(L8)
142 *  EX   *EXPECT* "," *TPI* "," *DES* "," *SELF*
143          .EQ( NO , NOI )   /F(L6)
144          DEX  =  "3"  /(L7)
*       FORMS NEW HYP
145 L8       CAL "=" /S(L9)
146          NO  =  "GET.EAT,"   /(L10)
147 L9       NO  =  "GET.DESTROY,"
148 L10      DISCARDS TP "," DES "," NO "," /S(U6)
149          $HYP  =  $HYP TP "," DES "," NO ",;90;5/"
150 U6       $NO  =
151          OBJS =  LOBJS  OBJS
152          WASH  *LOBJS*
153          OBJS  NO "," *XX* "/" =  /(U2A)
154 PUT2     SYSPOT = "  AT TIME = " TIME " , MIKRO IS=  "
*    REPL GETS ONLY SOME
155 PUT1     BLANK *JJ*   *OUTI* *II*  *LISTO*
156 P1       JJ =  JJ + "1"
157          .GT( JJ , CLS )  /S(P2)
158          OUTI =  OUTI "*"  /(P1)
159 P2       II =  II + "1"
160          .GT( II , RWS )  /S(P3A)
161          $( "P" II ) =  OUTI  /(P2)
162 P3A      COBJS  =  OBJS
163          WASH *II* *LISTO*
164 P3       COBJS *NO* "," *TP* "," *DES* "=" *XX* "/" =  /F(P4)
165          $NO "LOC=" *RO* "*" *CO* "," *XX* "DXY=" *DX* "*" *DY* ","
166          LISTO = LISTO NO "=" RO "*" CO ",DXY=" DX "*" DY "/"
167          CO  =  CO  - "1"
168          CO "=" *XX* = "0"
169          $( "P" RO ) *XX/CO*  TP  /S(P3)
170          $( "P" RO )  *XX/CO* *SS/"1"* = XX TP /F(P7)
171          REPL = REPL RO "*" CO "=" SS "," /(P3)
172 P7       $( "P" RO ) = $( "P" RO ) "+" TP  /(P3)
173 P4       II = II  + "1"
174          .GT( II , RWS )  /F(P4)
175 P5       SYSPOT =  "  REPLACED = " REPL
176          WASH *REPL*
177          SYSPOT =  " OBJ (NAME=LOC,MOVEMENT) = " LISTO
178          CORGS  =  ORGS
179          SYSPOT =  " ORGS AND THEIR INFO ARE= "
180 P6       CORGS  *NORG* "," *XY* "/"  =  /F(RETURN)
181          SYSPOT =  "     ORG " NORG " = "
182          $NORG  *STATE* "$" *HYP* "$" *PASTDO*  "$" *VIEW* "$" *CHANGES*
183          SYSPOT =  "  STATE =  "  STATE
184          SYSPOT =  "  HYP =  "  $HYP
185          SYSPOT =  "  PASTDO =  "  PASTDO
186          SYSPOT =  "  VIEW =  "  VIEW
187          SYSPOT =  "  CHANGES =  " CHANGES  /(P6)
188 END    GO
0 SYNTAX ERRORS DETECTED

COMPILATION COMPLETED AT  9:34:36

CPU TIME =       58.9 SEC.
I/O TIME =       82.1 SEC.

EXECUTION STARTED AT  9:34:36
```