LATENT ENERGY ENVIRONMENTS: A Tool for Artificial Life Simulations

F. Menczer and R. K. Belew

Cognitive Computer Science Research Group
Computer Science and Engineering Department
University of California, San Diego
La Jolla, CA 92093-0114 USA
E-mail: fil@cs.ucsd.edu, rik@cs.ucsd.edu

Technical Report CS93-301, July 1993

Abstract[1]**.** This report serves as documentation for a new ALife simulation tool, called LEE. The motivation behind the LEE model lies in the need for a measure of complexity across different ALife experiments. This goal is achieved through a careful characterization of environments in which different forms of energy are well-defined. Here we describe the main features of the program in its version 1.0. LEE is an open software engineering tool for the development of computational simulations. It allows experiments to be performed in an interactive, user friendly way; and user-defined data to be collected for statistical analysis. The implementation is written in C and runs on both UNIX and Macintosh platforms. Flexibility has guided the program design, making LEE applicable to the study of a variety of problems related to complex evolutionary systems.

[1] **Acknowledgments.** This program originated from conversations with Stefano Nolfi and Domenico Parisi at the Institute of Psychology of the National Research Center in Rome, Italy. Stefano Nolfi and Jeff Elman have assisted in writing the code. Greg Linden has written the Macintosh interactive version. We also thank everyone in the Cognitive Computer Science Research Group, in the CSE Department at UCSD, for helpful discussions on many issues of the model.

1. Introduction

LEE (Latent Energy Environments) is both an ALife model and a software tool to be used for simulations within the framework of that model. We hope that LEE will help us understand a broad range of issues in theoretical, behavioral, and evolutionary biology. The LEE tool described here consists of approximately 7,000 lines of C code and runs on both UNIX and Macintosh platforms.

The persons who have assisted the authors in writing the LEE software are, in alphabetical order: Jeff Elman and Greg Linden from the University of California, San Diego, USA; and Stefano Nolfi from the Institute of Psychology of the National Research Council, Rome, Italy.

In the future we might put the code in an anonymous ftp site. For inquires, please email either one of the authors. Here is a list of the LEE source files with short descriptions:

FILE	IS
lee.doc	documentation
Makefile	for compiling with make (UNIX)
LEE.proj.rsrc.hqx	Mac resources (BinHex4 format)
defs.h	header: includes, defines, typedefs, externs
fworld.c	world handling routines
globals.c	global variables declarations
gworld.c	world handling routines
include.c	random generation, options routines
interact.c	system-dependent routines
io.c	file input/output
main.c	main function, startup, usage
net.c	neural nets handling routines
populati.c	population handling routines
macdisplay.c	display routines (Mac only)
macinterface.c	interface routines (Mac only)
config.c	utility for creating .cf files
table.c	utility for creating .tb files

2. Model

It is not the purpose of this document to describe details of the LEE model[2].

[2]see F. Menczer and R.K. Belew, *Latent Energy Environments: A Model for Artificial Life Complexity* - Technical Report CS93-298, U.C. San Diego, July 1993

Therefore we will just introduce some general points and refer the interested reader to other sources of information regarding the theoretical issues.

2.1. Motivation

The modeling of environmental complexity across different ALife experiments is perhaps the main motivation behind this project. LEE allows the specification of environments of graduated complexity. A spatially distributed series of "atomic elements" must be combined to transform their "latent potential energy" into "work" necessary for survival. Behavioral strategies must be evolved by the population in order to allow an efficient exploitation of the available energy. This latent energy can be used to measure the environment complexity with respect to the survival task. See the appendix for an example of how latent energy and optimality of behavior can be linked quantitatively through the definition of carrying capacity.

2.2. GA

The genetic algorithm implemented in LEE is a steady-state model rather then a lock-step generational one. The progression of the adaptive process is measured in terms of time rather than generations. At any one time step possibly all the organisms in the population may live, use and/or acquire energy, and reproduce or die. Consequently, the size of the population varies with time. If latent energy is not made available at a rate sufficient to support the energy expense of the population, extinction may occur.

The notion of fitness is replaced by that of energy. In the current version, organisms are created initially with random energy distributed uniformly in the interval [0,ALPHA]. When the energy of an organism reaches 0, it dies. When its energy passes the ALPHA threshold, it reproduces and gives half of its energy to the offspring.

2.3. Neural nets

Each organism is implemented by a feed-forward neural network plus a sensory-motor system and a gut, i.e., a reservoir for energy, both in work (usable) and latent (atomic elements) form. The sensory system consists of a user-specified set of sensors that are mapped onto the network input units. The network may have as many hidden layers as desired. The output layer maps its activation values onto the motor system, made of a set of user-specified motors. Learning can occur in the current version by means of standard back-propagation of error. The error is computed on an input prediction task.

2.4. Life

Organisms live by moving in a finite world consisting of a rectangular grid with toroidal edge conditions. Each basic life cycle (called sweep) consists of 5 steps:

1. Gather information about the surrounding world by means of its set of sensors.
2. Elaborate the sensory information to produce a motor action.
3. Make a movement in the world by means of its set o motors.

4. Use the new sensory information as teaching input for a prediction task learned during its lifetime on a subset of the neural net. (Learning is optional.)
5. Consequences of the movement: there is an energy cost, there may be an energy increase or decrease (depending on the contents of the new world position and the reactions caused by the acquisition of such contents), and finally these energy changes may result in death or reproduction.

3. Sensory-motor system

This can be made arbitrarily simple or complex depending on the object of the simulation/experiment. The numbers of sensors and motors are fixed at compile time (NMOTORS, NSENSORS parameters in defs.h) and the types of sensors and motors (systems) must be specified in the configuration (.cf) file. All the routines relative to the sensory-motor system are in the files fworld.c and gworld.c.

3.1. Sensors

Different sensor systems are numbered 0,1,2,... and named in defs.h. Each sensor can be of any system. Different systems map onto different numbers of input units. There are two sensor systems implemented in the current version: GUT and CONTACT. The former senses elements present in an organism's own gut; the latter senses those present in the world cell in front of the position currently occupied by the organism. New sensor systems can be implemented in a straightforward way. Each sensor also has a "complex" that identifies which elements can be sensed by it.

3.2. Motors

Different motor systems are numbered 0,1,2,... and named in defs.h. Each motor can be of any system. Different systems are mapped onto by different numbers of output units. There is one motor system implemented in the current version: BINARY. It allows the organism to make one of four possible moves: stay still, turn left or right 90 degrees, or move ahead. New motor systems can be implemented in a straightforward way. Each motor also has a "power" that determines its range. All moves have the same energy cost.

Note that, because of the prediction learning task, the number of output units of the network must be larger than required by the motor system, namely, it must include an additional number of units equal to the number of inputs.

4. Code description

The source code is commented at function level. A general, non-exhaustive description of the program structure follows.

4.1 Data structures

All the global variables that should be under the experimenter's control are initialized to a default value and allowed to be changed without recompiling the code by command-line options. Default values are in defs.h. All the simulation parameters and structure declarations are also in defs.h. Part of this file is shown in the table below. Global variables are declared in globals.c.

DEFINE	VALUE	COMMENT
/*************** SIMULATION PARAMETERS ****************/		
NINPUTS	4	max number of inputs units
NHIDDENS	5	max number of hidden units
NOUTPUTS	6	max number of output units
XMAX	25	world x dimension (max=25 for Mac!)
YMAX	25	world y dimension (max=25 for Mac!)
TYPES	3	number of different food types
NSENSORS	4	total number of sensors
NMOTORS	1	total number of motors
ALPHA	100	reproduction threshold
INIT_ABUNDANCE	0.4	for initial food amount
WEIGHT_LIMIT	1.0	for weight initialization
/****************** DEFAULT OPTIONS *****************/		
WMUPE	0.1	percentage of mutated weights
BMUPE	0.1	percentage of mutated biases
INIT_MUTATION_RANGE	0.2	default mutation range
INIT_LEARN_RATE	0.2	default learning rate
INIT_GENERATIONS	500	default generations
INIT_LIFE_CYCLES	50	default sweeps
INIT_POP	100	default population size
FILE_ROOT	"test"	default root for input files
/************* SENSORY-MOTOR PARAMETERS *************/		
MU_SENSOR_PROB	0.5	prob. of mutating sensory system
COMPLEX_SIZE	1	number of atoms in complex
MU_MOTOR_PROB	0.0	prob. of mutating motor system
BINARY_ENERGY_USE	0.1	energy cost per move
BIN_POWER	1	binary motor moves by one cell

The world is implemented by a 2-D matrix where each element points to a cell, which is a linked list of elements and pointers to organisms. The population is a linked list of organisms. Each of them is a structure that contains a description of its genotype and phenotype. In the current version, the genotype contains the network weights and the sensory-motor system specifications. The phenotype contains a copy of the network weights (modifiable during life by learning) and a number of other features such as gut content, position, orientation, age, etc.

Finally, the reaction table is a symmetric 2-D matrix of structures, one for each couple of different element types. A reaction structure contains a bit to determine whether it is possible or impossible. In the former case it may be exothermic (with positive gain of energy for the organism) or endothermic (negative energy) and it may have by-products. The reaction table represents in the LEE model the set of all the environment features that are not under the control of the evolutionary process. It could be seen as all the (non-adaptive) "physical-chemical laws".

4.2. Compiling LEE and setting up a simulation

4.2.1. UNIX

Make sure all the UNIX .c files, defs.h, and Mekefile, are in the current directory. Then compile using "make all" on the command line. The UNIX compiler used in Makefile is cc. The UNIX make utility creates the object (.o) files, an archive (lee.a), and the executable (lee).

Similarly, the commands "make table" and "make config" should be used to compile the two LEE utility programs.

4.2.2. Macintosh

The Macintosh compiler we refer to is Think-C. The resource file is in BinHex'ed format (.hqx) and must be decoded using Compactor Pro, Unstuffit, or one of many other Mac and UNIX utilities. This file must have the same name as the project (with the .rsrc extension added) for the resources to be loaded properly. Naming the project "LEE.proj" and the resource file "LEE.proj.rsrc" is recommended.

Using a new project, add all the source files (except config.c and table.c) to the project using the "Add..." command under the Source menu. Add the libraries ANSI and unix from the C Libraries folder and MacTraps from the Mac Libraries folder.

The Macintosh loads the code in 32k segments, swapping them in and out of memory when necessary. This means that the LEE code (which is longer than 32k) must be segmented. This is done by clicking and dragging the filenames in the project window to the very bottom (below all the others) and releasing. We recommend segmenting into three or more segments by moving the ANSI library into its own segment, moving MacTraps, unix, macinterface.c, macdisplay.c, and interact.c into a second segment, and the rest of the files in the remaining ones. When a file or group of files is in a separate segment, a dotted line will appear between them and the others.

Under the Project menu, choose "Set Project Type...". Set the partition to 500k, change the creator to "GLEE" and change the MF (Multifinder) option to "Background NULL events". Make sure that the code generation option <MacHeaders> (under the Edit menu) is checked.

Analogously, new projects must be created to compile the utility programs (config.c and table.c files) separately.

4.3. Utility programs

Two utility programs, config and table, make it easier to prepare the two files (named <fileroot>.cf and <fileroot>.tb) that are necessary to set up and run a simulation.

4.3.1. config

The file config.c contains the source code for generating configuration (.cf) files. The configuration for a simulation consists of a description of the neural net's architecture (number of layers including input, number of units per layer, etc.) plus other individual features such as the size of the gut (max number of elements it may contain) and the types of sensors and motors.

In UNIX, to create a configuration file, type "config" and then follow the instructions. For the Macintosh, just double-click on the config icon. The .cf extension is added automatically, while the fileroot must be given: "test" is the LEE default, and if a different fileroot is chosen then LEE must be told upon execution by using the -f options. This helps in distinguishing different experiments.

4.3.2. table

The file table.c contains the source code for generating table (.tb) files. The table for a simulation consists, first, of a description of the spatial and temporal distributions for atomic elements. Elements of each type are generated by a replenishment function according to a 2-D pseudo-normal distribution for which the experimenter must specify center coordinates, peakedness, and magnitude. Second, the table contains the entries for all the reactions as described in section 4.1.

In UNIX, to create a table file, type "table" and then follow the instructions. For the Macintosh, just double-click on the table icon. The .cf extension is added automatically, while the fileroot (which must coincide with the .cf one) must be given: the name convention for the .cf file applies for the .tb file as well.

4.4. Brief guide to the LEE source code

- defs.h and globals.c contain all the simulation parameters. Many of the parameters can be set at run time with command-line options. On the other hand, changes in the files will require recompilation. However, users should only need to change values of constants in defs.h. Most of the variables and constants should be sufficiently commented to understand their usage.

- main.c has the main() function, initialization procedures, printing information to stdout, options processing, usage, etc.

- include.c has code that was taken from other sources. Routines in this file deal with command-line option processing and pseudo-random numbers. A portable random number generator was taken from the GAucsd package.

- io.c contains most of the routines that deal with input/output, either from/to standard input and output, or from/to files. For example all the checkpointing is done by these routines in order to save/load the current state of a simulation.

- net.c contains all the routines relative to the neural nets' management, such as initialization, memory allocation, spreading of activation, logistic function computation, back-propagation, etc. For time-efficiency, the logistic function is stored in a table. Note that in the current version the phenotype weights are simply copied by the genotype ones. In future versions, we expect that more complex mechanisms may guide this developmental process.

- populati.c contains the function that performs the main generational loop: for each sweep, each organism is given a stochastic chance at a "life cycle". This stochastic mechanism is an unbiased way to implement serially the parallel life process. An outer generational loop is maintained mainly for checkpointing and data collection purposes. Other functions in this file deal with population initialization, reproduction, death, mutation, and other population handling routines. Note that in the current version mutation is the only genetic operator available. Uniform (float) deviate mutation is applied to the genotype weights and biases of the network at reproduction, and other types of mutation are devised ad-hoc for the sensory-motor system.

- fworld.c and gworld.c contain most of the routines that handle the world: initialization, interaction with the organisms, reactions, sensory-motor apparatus and their mutation operators, and replenishment of atomic elements. The former file also contains a function, save_dat(), which the experimenter is supposed to tailor to his/her needs for performing data collection.

- interact.c contains all the routines with system-dependencies, i.e., interactive vs. non-interactive version compatibility differences. All the routines for handling the current graphical interface are in the macdisplay.c and macinterface.c files. These two files are for the Macintosh version only and not necessary for the UNIX one. In fact, currently, the only interactive platform is the Macintosh, where a graphical representation of the world and the life process can be observed on the screen during simulations. The Macintosh version, however, can also work in non-interactive mode. In future versions, we expect that there will also be an interactive mode in the UNIX version, to be implemented via the X interface.

5. Running the program

As explained above, we will refer to the interactive version meaning, for the current version, the Macintosh version.

5.1. Input

The .cf and .tb files created with the config and table utilities must me in the same simulation directory/folder along with the lee executable/application.

5.1.1. Non-interactive

Run lee by typing "lee" alone or followed by command-line options. Type "lee -u" on UNIX machines for a list of the available options with a brief explanation of their function, as shown below.

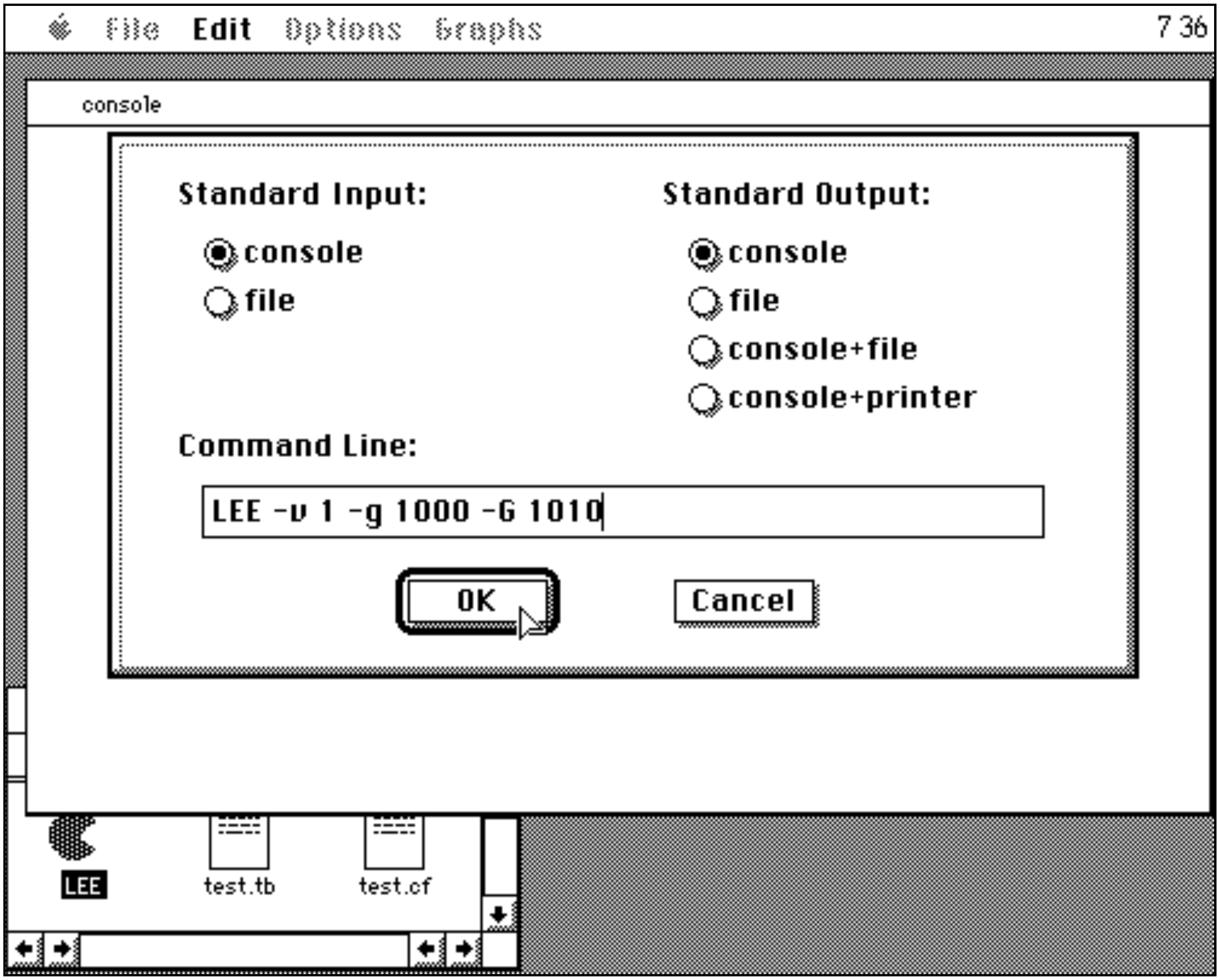
```
example% lee -u

COMMAND-LINE OPTIONS:

-R #  use # to seed random generator
-E #  record error in .err file every # sweeps
-N    no learning
-f #  specify <fileroot> for .cf and .tb files
-s #  use # sweeps per generation
-po   print information each cycle
-r #  use learning rate #
-G #  stop at generation #
-P #  size of initial population
-g #  start by loading generation #
-h #  mutations range
-n #  percentage of weights mutated (0-100)
-o #  percentage of biases mutated  (0-100)
-b    do not mutate biases
-v #  verbose mode (0-5)
-d #  save best # organisms of generation
-e #  save every # generations
-u    usage

example%
```

5.1.2. Interactive

Run LEE by double-clicking on the LEE application icon; a window will pop-up to allow UNIX-like command-line options, as shown in the example of the figure below.

5.1.2.1. Menus

The menus allow user interface with the simulation in the Macintosh application. There are three menus: File, Options, and Graphs.



- File Menu: allows the user to save the current state of the simulation, load an old saved state, or quit the application. [Save/load functions are not yet implemented but can be obtained through command-line options as in UNIX.]

- Options Menu: allows the user to disable the interactive mode, select the lasso option, and pause a simulation. Unselecting the interactive option will cause the Mac to emulate the UNIX version of the simulation. Selecting it allows a graphical representation of the LEE environment. Clicking on any cell in the world causes the content of that cell to be shown. Simulations run considerably faster with interactive mode turned off. Since the default value of verbose on the Mac is 0 (off), the command line option to set verbose to 1 or higher must be given to cause any output to the console window. The lasso option causes a selected organism to be tracked with the zoomWindow when it moves. If this option is off, the zoomWindow will switch to a different item in the cell if the organism moves out of the cell. Pause is useful for examining organisms.

- [Graphs Menu: not implemented.]

5.2. Output

5.2.1. Non-interactive

5.2.1.1. Standard Output

The verbose level determines the amount of information printed to the standard output in the non-interactive version or to the console in non-interactive mode. Level 0 means nothing is printed on stdout and no checkpointing or error files are saved. Levels 1 and up send increasing amounts of information to stdout (redirection to a file can be used); and checkpointing files are saved according to the -e option (default is saving only at the end of the simulation). The following figure is the typical screen for a simulation with the default UNIX verbose level, 1.

```
example% ls

exp1.cf        exp1.tb        lee

example% lee -f exp1 -P 100 -G 5 -R 1234

Config and table files          : exp1
Initial pop_size                : 100
Generations                     : 5
Life's sweeps                   : 50
Net's (first) 3 layers          : 4 -> 3 -> 6
Gutsize of the starting gen.    : 1
Learning rate                   : 0.20
Random seed                     : 1234
Number of weight mutations      : 5
Number of bias mutations        : 1
Range of mutations              : 0.20

Running generation: 0,  Pop: 100,       Time: 0

Running generation: 1,  Pop: 105,       Time: 2

Running generation: 2,  Pop: 101,       Time: 7

Running generation: 3,  Pop: 105,       Time: 11

Running generation: 4,  Pop: 104,       Time: 17

example% ls

1234.dat        4.wld         exp1.cf         lee
1234.gen        4.ind         exp1.tb

example%
```

5.2.1.2. Files

Most non-interactive output is done through files. The figure above also shows the created data and checkpointing files for a typical lee run.

The <fileroot>.err file contains errors (<fileroot> is the same as for the .cf and .tb files). The <gen>.wld and <gen>.ind files contain information needed for checkpointing (<gen> is the generation number). The former refers to world data and the latter to state and population data. Checkpointing is determined by the -e (for saving) and -g (for loading) options. The super<n>.ind files (according to the -d command line option) contain descriptions of the best organisms, ranked by <n>.

The files <seed>.gen and <seed>.dat contain, for each generation, data relative to the simulation (<seed> is the random generator seed number, either generated by a time routine or set with the -R option). The .gen file has the energies of all the organisms (it can be used for, e.g., energy histograms).

The .dat file contains, for each line, generation number, population size, and energy average, separated by tab characters. These data can be used to analyze/monitor the simulation results using any plotting program. Additionally, the experimenter may want to keep track of other data and/or statistical measures computed throughout the simulation. This can be done most easily following the template in the function save_dat() in the fworld.c file. In the current version, this function measures the distributions of different atom types in sensor complexes across the population. The function save_dat() is called once for each generation, and each datum should be printed on the .dat file followed by a tab separator, on the same line (new-line characters are inserted automatically).

5.2.2. Interactive

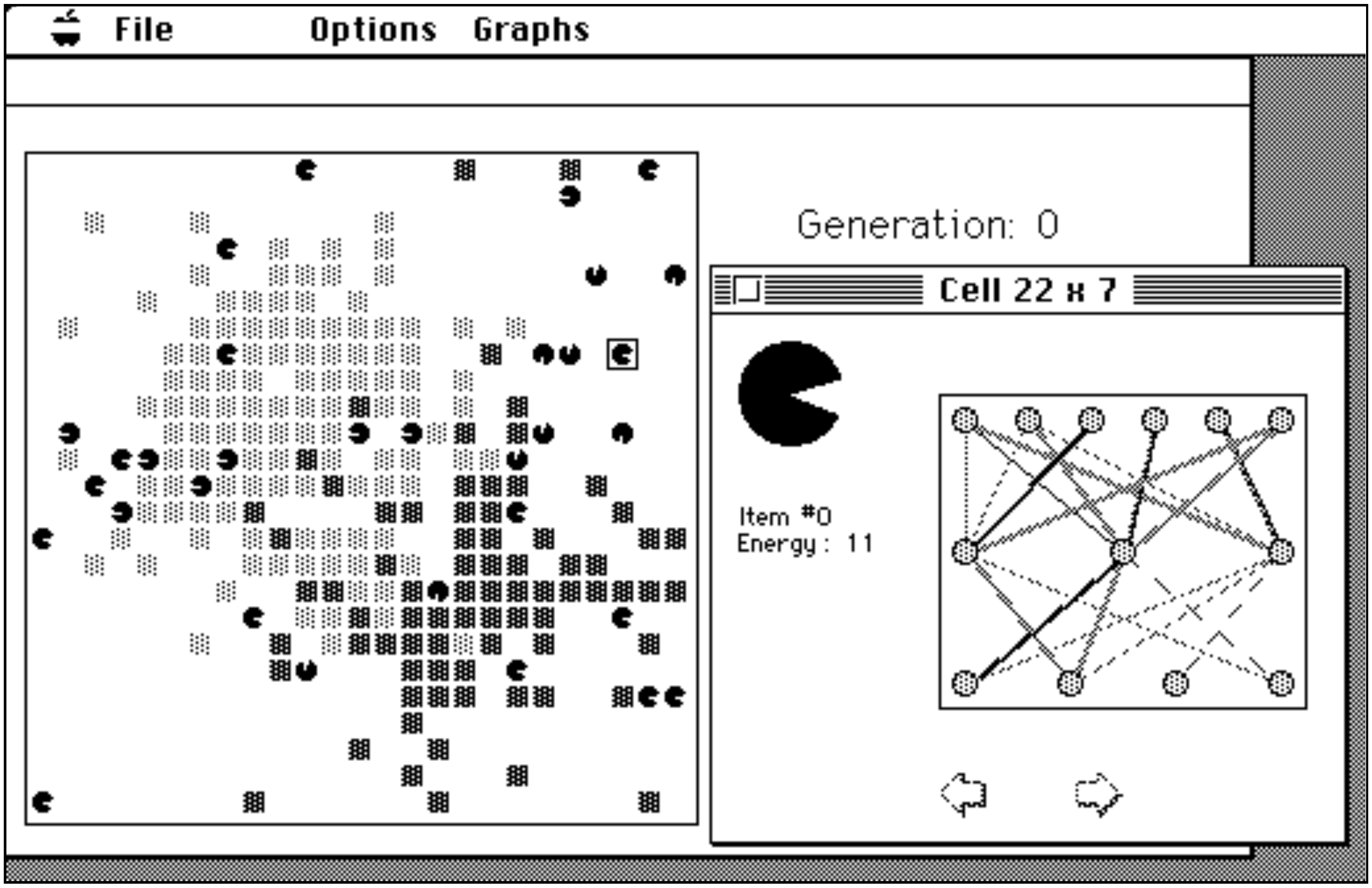
A typical Macintosh screen during a LEE run is shown in the figure below.

5.2.2.1. Windows

- console: this is the UNIX emulation window and can be used as stdout by the Macintosh version of LEE, when the verbose level is set to a value greater than zero. Until interactive is unselected in the Options menu, this window is hidden behind the other windows.

- LEE: this window shows the toroidal world, with all the organisms and food elements. Food is represented by three shades of gray (more than three types of food will cause the additional types to be indistinguishable without zooming). Organisms and their facings are represented by a circle with a wedge missing in the direction of facing. In the case of multiple items in a cell, the last organism in the cell is displayed if there is an organism, otherwise the last food item in the cell is displayed.

- zoomWindow: allows the user to examine the contents of a specific cell. By clicking on a cell in the LEE window, the zoomWindow is brought up for that cell. When multiple items are present in a selected cell, the previous and next arrows are activated, allowing the user to scroll through the items in the cell. For food items, the number of the item (starting from 0) and the type of food is displayed. For organisms, the item number is displayed, the energy of the organism given, and the neural network of the organism is shown. The network is represented by drawing negative-weight connections as gray lines (thin for medium values and thick for high values). Positive-weight connections are drawn as black lines (thin and thick for high and medium values, respectively). Weights near zero are not drawn. The bias of each node is shown as a gray-filled circle for a bias of 0, a light gray or white circle for a negative bias (depending on the value), and a dark gray or black circle for a positive bias. The rightmost output nodes (as many as the number of input nodes) are those used to predict the input at the next time step, used to train the net.



6. Conclusion

The LEE model and program have been designed in close association: we like to think that they are in the same relationship as a set of theories and an experimental laboratory. In order for this analogy to be reasonably actual, we have strived to make LEE a flexible tool. As a result, LEE is applicable to the study of a variety of problems related to complex evolutionary systems. These include, among others: co-evolution of sensory-motor system; development and morphology; co-evolution of soma with nervous system; evolution of environments; migration; evolution of memory; aging; evolution of learning; social interaction; cultural interaction; evolution of reproductive maturity; sexual reproduction; emergence of speciation; multiple species ecologies; and predator-prey population dynamics. It is our hope, to conclude, that LEE will represent both a rich theoretical framework and a useful simulation tool for the ALife community.

Appendix: Latent Energy and Carrying Capacity

Let us analyze an example in order to illustrate how LEE allows to quantitatively estimate optimality of behaviors by monitoring the population size throughout an experiment. We must understand the relationship between rates of world replenishment, latent energy, and carrying capacity.

Consider the following simple reaction table in the presence of two types of atomic elements, a and b:

	a	b
a	E_{aa}	E_{ab}
b	E_{ba}	E_{bb}

where

$$
\begin{aligned}
&E_{aa} = E_{bb} \equiv E_{=} \\
&E_{ab} = E_{ba} \equiv E_{\neq} \\
&E_{=} < 0 < E_{\neq}. \quad (1)
\end{aligned}
$$

The table, along with space and time distributions for elements, determines the environmental complexity of the survival task. The minimal set of elements for alternative behavioral strategies is {a,a,b,b}. In fact, with these elements an organism can catalyze the following sets of reactions:

$$
\begin{aligned}
(a+a),(b+b) &\rightarrow 2E_{=} \quad (2) \\
(a+b),(a+b) &\rightarrow 2E_{\neq} \quad (3)
\end{aligned}
$$

where (3) is the advantageous strategy while (2) is the deleterious one, given condition (1). If r is the rate of replenishment for both a and b elements, i.e.,

$$
\begin{aligned}
&r_a \equiv \# a \,/\, sweep \\
&r_b \equiv \# b \,/\, sweep \\
&r_a = r_b \equiv r
\end{aligned}
$$

then a set {a,a,b,b} is produced every 2/r sweeps. Thus strategies (2) and (3) produce energy changes

$$
\begin{aligned}
\Delta E_{=} &= \frac{2E_{=}}{2/r} = rE_{=} \quad (4) \\
\Delta E_{\neq} &= \frac{2E_{\neq}}{2/r} = rE_{\neq} \quad (5)
\end{aligned}
$$

per unit time (sweep), respectively.

Let us now consider the situation at equilibrium. On average, two conditions are verified: the population size remains constant, and energy is produced at the same rate at which it is consumed by organisms in the population. Using brackets to indicate time averages, we can write:

$$\langle population \rangle = const \equiv p$$
$$\langle \Delta E \rangle = 0. \quad (6)$$

Because energy is conserved at reproduction, the only consumed energy is that lost in the form of work, that is, used up for moving in the world. If we call c the cost per move for any organism, the average energy used by the population per unit time is given by pc.

To calculate how much energy is produced, we must know the strategy used on average by the population for combining elements. In other words, we need to determine how efficiently the latent energy contained in the elements is transformed into usable energy. Let us then introduce a probability distribution over strategies. This is quite simple in the present example, with only two strategies: let us call η the probability of strategy (3), so that (1-η) is that of strategy (2). Then the average energy produced per unit time by the population is, using (4) and (5):

$$\eta \Delta E_{\neq} + (1-\eta)\Delta E_{=} = r\left[\eta E_{\neq} + (1-\eta)E_{=}\right]$$

so that we can finally rewrite (6):

$$\langle \Delta E \rangle = r\left[\eta E_{\neq} + (1-\eta)E_{=}\right] - pc = 0. \quad (7)$$

Equation (7) provides the link between optimality of behavioral strategies, expressed through the probability distribution over catalyzed reactions, and carrying capacity, represented by the population size. If the former is known, we can obtain the expected population size by solving (7) for p:

$$p = \frac{r\left[\eta\left(E_{\neq} - E_{=}\right) + E_{=}\right]}{c}.$$

For example, assuming optimal behavior, η=1, this allows to estimate the maximum population that can be sustained given the parameters of a particular experiment:

$$p_{\max} = \frac{rE_{\neq}}{c}.$$

The converse case is useful for estimating the optimality of a population's behavior at equilibrium, by measuring the population size and solving (7) for η:

$$\langle \eta \rangle = \frac{E_{=} - \frac{pc}{r}}{E_{=} - E_{\neq}}.$$

Of course, such easy linear relation holds only for this simple example: more complex experiments will yield systems of differential equations, which may be too hard to solve analytically. The important point is that carefully designed latent energy environments allow to maintain an accurate connection between ALife models and simulations.