CS-TR-4182 **UMIACS-TR-2000-64**

A SIMULATION ENVIRONMENT FOR EVOLVING MULTIAGENT COMMUNICATION

September 2000

James A. Reggia[1,2], Reiner Schulz[2], Juan Uriagereka[3], and Jerry Wilkinson[4]

[1] Institute for Advanced Computer Studies
[2] Department of Computer Science
[3] Linguistics Department
[4] Biology Department

University of Maryland
College Park, MD 20742 USA

Abstract: A simulation environment has been created to support study of emergent communication. Multiple agents exist in a two-dimensional world where they must find food and avoid predators. While non-communicating agents may survive, the world is configured so that survival and fitness can be enhanced through the use of inter-agent communication. The goal with this version of the simulator is to determine conditions under which simple communication (signaling) emerges and persists during an evolutionary process.

INTRODUCTION

The simulation environment described here is based on a two-dimensional rectangular space of discrete cells referred to as the *world*. The cells form a square-tessellation of this space. The world size is arbitrary but the boundaries are absolute: anything attempting to move out of the existing world is reflected back in. Time is also discrete, starting at zero and being incremented by one on each iteration of a simulation.

A number of entities (objects) are initially randomly placed in the world during a simulation. There are three types of objects: food, predators and agents. It is permissible for multiple entities of multiple types, or of the same type, to occupy the same cell simultaneously. In this sense each cell can be viewed as a small region of finite extent accommodating multiple entities that can interact directly when they occupy the same cell[1]. Agents search for stationary food sources and consume them when found. Agents flee from predators when detected. If detected in time, an agent can escape from a predator. If not detected, a predator will approach and "kill" an agent. Agents can only "see" a small region in front of (and to the sides of) the cell they occupy in the direction they are oriented. Thus predators can approach from other directions undetected, or an agent may not observe a nearby source of food that it passes. However, agents detecting a predator or food site could signal their existence, thus alerting other agents that might not otherwise detect them. Communication thus conveys the potential advantage of alerting agents to nearby danger or food. Agent "fitness" is anticipated in general to be a reflection of success in avoiding predators and obtaining food.

There are four classes or types of agents. These are designated NC for non-communicating agents, FO for agents that communicate about food only and not about predators, OP for agents that communicate about predators only and not food, and FP for agents that communicate about both food and predators. Often in simulations, one is interested in starting with 100% NC agents and determining what fraction of the agent population eventually evolves over time to communicate about food, predators or both.

Instructions on starting the current version of the simulator are given later in this report. Results following a simulation will be found in the following three files in the same directory as the one in which the simulator is started:

[1] For example, an agent can only consume food when it is in the same cell as the food, and a predator can only kill an agent when they are in the same cell. On the other hand, objects also interact when they are in different cells in the sense that an agent/predator may see an object in another cell and change its behavior based on that, and agents may communicate with other agents in other cells via messages.

File	Contents
res.stat	1. parameter settings during the simulation
	2. agent genome/chromosome counts, collected periodically during the simulation
	3. tallies of "statistics" collected during the simulation
res.disp	display of world state, collected periodically during a simulation
res.log	a record of events occurring during a simulation (empty if tracing is turned off)

Numerous parameters/variables in the simulator file can be modified to affect the details of a simulation, and many of these are listed at various places in this report. For example, the parameter *tmax* specifies how many iterations to run a simulation, so a simulation will stop when the current iteration number *tick* equals *tmax*. All global parameters and global variables are declared and assigned a value at the beginning of the system file.[2]

General Simulator Parameters/Variables:
world = two-dimensional space of cells
xdim, ydim = x and y dimensions of *world*
tmax = number of time steps to run simulation
mode = mode of operation
tick = current time/iteration
trace = level of tracing (0 to 2); turns on output to simulation log file
errlist = list of fatal error(s) detected causing simulator termination prematurely
gentime = display gene counts in res.stat every *gentime* iterations
disptime = display world state in res.disp every *disptime* iterations

SAMPLE OUTPUT FILES

Sample contents from the result files are given below. First, the contents of *res.stat* are shown. The values of parameters appear first, indicating (among other things) that a population of 200 agents are present in a 60 x 60 world. This is followed by a table where, every 100 iterations/ticks, a variety of information is given for each of four classes of agents: number in existence, their average age, number currently fleeing/avoiding predators, and their average fitness. For example, at iteration 1000, there are 193 non-communicating agents (NC), 2 agents that communicate about food only (FO), 4 that communicate about predators only (OP), and none that communicate about both food and predators (FP). Finally, at the end of the file after termination at iteration 100,000, various information collected during the simulation is displayed, including that 77108 total agents died due to starvation while 190,722 died due to predators, and that a total of 37,018 messages were

[2]Global parameters/variables all have names starting with an asterisk in the actual code; local variables never do. Thus, a parameter such as *tmax* indicated here is actually named $*tmax$ in the code.

posted by communicating agents. Similar information is given for the last 30,000 iterations. The contents of *res.stat* are:

```
Fri May  5 11:55:43 EDT 2000
60 x 60 world, max ticks 100000, mode EVOLVE
toursize 10, locnpars SPATIAL, pmf 0.003, pmp 0.003, pc 0.0
  locnkids 3, maxrad 6, reprocost 0, reproage 4, follow parents NIL
200 agents (vf ctr 2, hear range 6, max beliefs 15, prob continues 0.9)
  (avoid time 4, food capac 30, initfoodstore 25, max age 1000)
20 preds (see range 3, hear range 6, quiescent 2, prob continues 0.9)
  (max pursuit 7)
32 initial food sites (size 50, target amount 1600, reaptimes 3)

Population genome data:
          #agents            avg. age          #agents in
                                              FLEE/AVOID state   avg. fitness
  Tick   NC FO OP FP TOT   NC FO OP FP TOT   NC FO OP FP TOT   NC FO OP FP TOT
     0  200  0  0  0 200    0  0  0  0   0    0  0  0  0  0     0  0  0  0  0
   100  196  0  1  0 197   59  0 47  0  59   30  0  0  0 30    16  0 10  0 16
   200  195  1  1  0 197   60 49  1  0  59   27  0  0  0 27    16  9  0  0 16
   300  196  0  0  0 196   59  0  0  0  59   41  0  0  0 41    15  0  0  0 15
   400  197  0  1  0 198   48  0  0  0  48   36  0  0  0 36    14  0  0  0 14
   500  192  1  5  0 198   50  5 23  0  50   25  0  2  0 27    16 23 18  0 16
   600  190  2  4  0 196   55 22 65  0  55   45  2  1  0 48    16  6 11  0 16
   700  197  1  1  0 199   64 70 48  0  64   27  0  0  0 27    18 14 25  0 18
   800  197  3  0  0 200   65 79  0  0  65   27  0  0  0 27    15 18  0  0 15
   900  184  5  6  0 195   48 69 31  0  48   44  0  2  0 46    14 15 20  0 14
  1000  193  2  4  0 199   48 62 74  0  49   28  0  0  0 28    16 23 14  0 16
  1100  176 25  0  0 201   52 40  0  0  51   28  5  0  0 33    18 13  0  0 17
                                  .
                                  .
                                  .

                           (material deleted)
                                  .
                                  .
                                  .

 99700  180 18  1  0 199   73 72 39  0  72   23  1  0  0 24    17 15 16  0 17
 99800  173 26  0  1 200   62 60  0 51  62   12  5  0  0 17    16 12  0 17 15
 99900  194  0  0  0 194   38  0  0  0  38   29  0  0  0 29    16  0  0  0 16
100000  197  0  1  0 198   64  0 59  0  64   28  0  0  0 28    18  0 28  0 18
```

```
Termination at tick 100000
Total agents 268028, Total preds 20, Total food units 3029050, Total msgs 37018
Current agents 198, Current food units 1538
Deaths: starvation, old age, predation, total
NC: 69953, 0, 175078, 245031
FO: 4454, 0, 10359, 14813
OP: 2518, 0, 4904, 7422
FP: 183, 0, 381, 564
TOT: 77108, 0, 190722, 267830
Life expectancy: NC 75, FO 71, OP 75, FP 71, TOT 74
Statistics at equilibrium state (starting at tick 70000):
Deaths: starvation, old age, predation, total
_NC: 21824, 0, 51827, 73651
_FO: 1619, 0, 3657, 5276
_OP: 512, 0, 1031, 1543
_FP: 33, 0, 78, 111
_TOT: 23988, 0, 56593, 80581
_Life expectancy: NC 74, FO 71, OP 74, FP 73, TOT 74
```

Displays of the current world state can be done in one of two ways. The first way is to simply indicate which sites are occupied by agents, food and/or predators. For example, two consecutive maps of the world appearing in the file *res.disp* are shown. These maps provide a coarse indication of the world state at the ends of ticks 14 and 15. A dot (.) indicates an empty cell, a circle (o) an agent, an asterisk (*) a predator, and a plus sign (+) a food site. A predator and agent are generally not at the same site (if so the predator captures the agent and it disappears). If an agent is at a food site, only the circle and no + appears. Object/cell coordinates (x,y) are measured from an origin at the upper left corner. The positive x-coordinate is vertical increasing downwards with the first row's index being zero. The positive y-coordinate is horizontal increasing to the right with the first column being zero. Thus the uppermost food site + at tick/iteration 14 is at (5,38). The predator located at (7,19) at tick 14 is seen to move up one cell to (6,19) at tick 15, capturing the agent at that location. The excerpt from *res.disp* is:

```
Tick = 14
........................................
........................................
........................................
........................................
....o........o................o.........
.o....................................+.
..o................o....................
...................*....*...............
........................................
......................o.................
..............+.........................
........................................
........................................
........................................
..........o.............................
.....o..................................
.....................*..................
........................................
........................................
......o.................................
```

```
Tick = 15
........................................
..+.....................................
........................................
........................................
o............o..........................
...o...........................o......+.
...................*....................
........................................
.......................*................
........................................
..............+......o..................
........................................
........................................
........................................
........................................
..........o.............................
......o..............*..................
.............o..........................
........................................
.......o................................
```

At present, the code implementing the above display method is "commented out" in the simulator source file. The second display method, described in the following, is used by default. With this second method, priority is given to displaying agents and further details about them in *res.disp* as follows. If one or more agents are present in a cell, only their type is indicated, and any food or predators in the same cell cannot be seen. When a single agent is present in a cell, its type is indicated as:

n	NC agent
f	FO agent
p	PO agent
b	FP agent

When multiple agents are present in the same cell, their types are:

N	NC agents
F	FO agents
P	PO agents
B	FP agents
M	mixture of different types

Symbols used to designate cell contents when no agents are present in a cell are:

. cell is empty
$*$ predator present, but no agents (food cannot be seen)
+ food present only

Here is an example of how the default *res.disp* file typically appears:

```
Tick = 0
................p.p.....f.......f....pM..fb......bf.......b.
.....f...bn.f.n...b...n.b...n....fp..n.P............p.b.....
..p....+..p..f.n....p.b.......+..p.....b.............n.b.b.b
.....Mf....nM.....Mfn...........Mnf.p..f......f...f....b....
..f..........f.....fpn.....f.....p...f.F....n....M..........
....b.fb.....p......n.n.fpp.........p..............nnp..p...
..p..p.p............................................p..pn..p
..f.........n...f..........f...ppp.b.b*.....................
........M..fb.......pn......p...M.....p....M...f...f..nf.p.n
....b.....n...p....n.......b.....b..........b.....n.........
.............b..b..p...........p...................M........
....n......p.b.........p.n.f.............F...........b......
..p.pp..........p.p........ffb.........f......N...........b.
..M....................*n....n......f..........+....M.....b.
b..M.b..bn......p.........F..p...nP........f.b.....ppnb.....
p..........................Mpb.M..M.f..p.n.......pf........b
...f...b.....f......Mpf....n.n.....b........bp...nnP.p.np...
..M......b........p.......+......n...Fppb....M...f..n.......
f....p............f......M....bn.....................n......
.f....B...n...b...................n.Mf....P.........f..+..M.
..........f.f....Mbp....f..........*...n...nf...fM....p...bn
....nf........n..p........b............f..fb.........f......
.M..............b.......b..np..f.....p......B......M........
.......M...f..n..fp.nb....n.......n....n......pf...*n.b*bf..
f............f...b.n.+b...b.p...b............M..........n...
..n..n....p...*..b.b.fM.b...f...pp.....b.n.......f.......M..
n..M...b..n.b...M...n...........+.....n......p....f.....M..p
p..n.....n.......n...bf.......fn.b..n...f...pnp.n...n.......
.bnb.p.b..n.....f.f.fbM.............n..........p..b..n..n..b
p...f......f....nf....ff..............nb........b......nb..b
```

Finally, an excerpt from an example *res.log* file is shown below. This file notes significant events, message postings, and details about agents whose state is other than WANDER. (Much of this information might be better displayed graphically in *res.disp* in a future version of the simulator.) For example, at iteration 15 it is noted that a new food site has been created at (1,2), and that agent a12 at (6,19) dies due to predator p1, as was noted above. The excerpt from file *res.log* is:

```
14: a3 dies, captured by p3 at (16,21)
14: a11(9,22) FLEE heading (1,-1) gazing (1,-1) consumed 0
14: a8(4,13) CONSUME heading (0,-1) gazing (0,-1) consumed 3
      Foods seen: (4,13,14)
```

```
14: a7(4,30) FORAGE heading (1,1) gazing (1,1) consumed 0
      Foods seen: (5,38,2)

15: Creating 1 new agent(s), randomly placed, directed
15: Create 1 food site(s) at (1,2)
15: p1(7,19) begins pursuit of a12(6,19)
15: a12 dies, captured by p1 at (6,19)
15: a11(10,21) FLEE heading (1,-1) gazing (1,-1) consumed 0
15: a8(4,13) CONSUME heading (0,-1) gazing (0,-1) consumed 3
      Foods seen: (4,13,15)
15: a7(5,31) FORAGE heading (1,1) gazing (1,1) consumed 0
      Foods seen: (5,38,2)
```

The posting of messages is also noted in the *res.log* file (not illustrated above). For example, at iteration 17 the two lines

```
17: a2(4,4) sees 10 food units at (1,2)
17: MSGs: (FOOD,4,4,a2)
```

appear signifying that agent a2 emitted a FOOD signal at this time when it first detected the new food site at (1,2). (In Version 2, food signals are emitted when agents first arrive at a food site, not when agents first see the food.)

FOOD

Food is initially placed at random locations throughout the *world.* Food sites are static and passive ("edible plants"). Parameters control the amount of initial food per site and the initial number of sites to be created. A computed value *targetfoods* (product of initial number of food sites and amount of food per site) indicates the target total number of food units to be present. If the amount of food present at any time falls sufficiently below this target, one or more new randomly-located food sites are generated to restore food levels (the total food present may transiently exceed the target). The number of food units at any one location, once created, remains constant unless consumed by an agent at that site. At present, each agent at a site can consume a maximum of one unit of food per iteration.

Food parameters/variables:
numfoods = number of initial food sites (randomly placed)
totfoods = total number of food sites generated
curfoods = all foods currently in the world
foodsize = number of food units placed in each new food site
targetfoods = target total amount of food to be present
foodrange = amount below *targetfoods* at which a new, randomly-placed food site is created

PREDATORS

Predators are simple, eternal, mobile, non-adaptive, non-evolving "machines" that hunt and kill agents. Predators receive no reward/punishment for their success/failure. Predators exist in one of three states: quiescent, searching, and pursuing. Predators enter the quiescent state initially (to allow randomly placed nearby agents a chance to detect them and escape), following a kill, and following an unsuccessful chase. During the quiescent state the predator is completely idle and of no danger to agents. After a predetermined time, the quiescent state ends and predators automatically enter the searching state. Predators in the searching state move around in a quasi-random fashion, searching for agents in nearby cells. On each iteration in this state, the predator moves one cell, in the same direction as previously with a predetermined probability p, or in a new randomly-chosen direction with probability $1 - p$. Predators can see any agents within a certain pre-determined distance *predrange* in any direction (view is blocked only by world boundaries at present). When a predator first sees one or more agents, it randomly selects one of the closest ones and enters the pursuit state with the selected agent as a target. In that state the predator repeatedly moves directly towards the selected agent until either it catches the agent (by landing on the cell occupied by the agent), it catches another agent inadvertently (by entering the cell of that other agent while pursuing a different one), or its pursuit time exceeds a pre-determined maximum (the agent escapes). In all of these cases the predator subsequently enters the quiescent state. Predators can also hear messages issued by communicating agents within a distance *predhearrange* (usually set to be greater than *predrange*, the distance a predator can see). If a searching predator hears a message, it changes its direction to be towards the source of the message. In this sense, communication has a cost for agents in that it can attract predators.

Predator parameters/variables:
numpreds = number of initial predators (randomly placed)
totpreds = total number of predators created
(should be same as *numpreds* at end of simulation)
predators = all predators currently in the world (list)
predrange = distance a predator can "see"
(must be less than half the smallest world dimension)
predhearrange = distance a predator can "hear"
predrest = time predator stays in quiescent state
pcont = probability of continuing searching in same direction
max-pursuit = maximum time an agent can be pursued before resting

AGENTS

The population of mobile agents are able to "see" a limited region in their neighborhood and may also be able to send/receive messages to/from other nearby agents. From such information each agent constructs a very limited "internal model" of the world in which it exists. This internal model represents the existence and location of nearby predators and food (and later perhaps other agents), but is potentially limited by memory capacity and may contain inaccurate information (for example, the location of a potential food source that was consumed by another agent since it was discovered; or, the location of a predator that has moved).

Agents in our model start where many past modeling efforts have ended: they have built-in, preprogrammed behaviors for avoiding predators, for seeking food or mates, and for other non-communication tasks. Such behaviors can lead to successful survival in the absence of communication. Our focus is on emergent communication that supplements this preprogrammed behavioral repertoire, leading to increased survival and fitness due to a better internal world model and cooperative actions. For example, as noted above, since agents can only "see" a small region in front of their location, the ability of agents to warn other nearby agents of the presence of predators could give communicating agents an increased chance of survival and hence reproduction. Communication could also convey a selective advantage in obtaining food/mates.

At the beginning of a simulation, a prespecified number of agents of prespecified types are initially placed in random locations throughout the world. As agents "die" during a simulation, they are generally replaced so that the population size is roughly constant. Details of how replacement is done can be varied from simulation to simulation, and is described further below.

Non-communicating (NC) agents essentially serve as controls for comparison with communicating agents. Each NC agent is represented as a table of information containing the agent's location, direction of movement, direction of gaze, its current "food stores", the total amount of food it has consumed since its creation, memory of recent predators/food sites seen, etc. Food stores represent the current food reserves accumulated by an agent. Each time an agent consumes a unit of food, its food stores increase by 1. On the other hand, all agent food stores are decremented by 1 periodically, so an agent's food stores may rise and fall during a simulation. An agent will die if: 1. its food stores reach zero; 2. it is captured by a predator; or 3. it reaches a predetermined maximum age. Prior to starting a simulation, many parameters concerning agents can be adjusted to determine the agent population size, the distance an agent can "see", the maximum food stores an agent can accumulate, how frequently food stores are decremented, the maximum age an agent can have, memory capacity, etc.

Communicating agents include all of the features of NC agents plus the ability to issue/receive messages. A single message consists of a label or signal, either FOOD or PRED, that is issued when an agent first sees a nearby predator or arrives at a food site. The only other information associated with the message is the location of the agent issuing the message. A message may be received only by other agents within a prespecified distance. This distance may be made any desired value prior to starting a simulation, e.g., it can be made so large that all agents can potentially communicate with all others.

Communicating agents come in three varieties. A communicating agent may send/receive messages about food-only (FO agents), predators-only (OP agents), or both (FP agents). Whether an agent object's class is NC, FO, OP or FP is determined by its *chromosome* as described later in this report.

An agent's behavior is governed by the *state* it is in. There are six possible states as listed below. Any type of agent can be in any of these states unless explicitly indicated otherwise.

WANDER: In this state an agent has no current goal: it knows of no food sites or predators. It therefore wanders aimlessly in the hope of discovering a food source. All agents start in this state, and may revert to it later. If the agent sees a food site/predator (or if a communicating agent processes a message about food or a predator), or has a memory of food sites or predators, the agent leaves this state to FLEE, AVOID, FORAGE or SEARCH states as appropriate

FLEE: An agent will immediately enter this state if it sees a predator regardless of all other information (highest priority state). The agent selects the closest predator it sees if there is more than one, and for a prespecified time moves directly away from that predator. It then checks whether it is sufficiently far from the pursuing predator before terminating this state. This behavior is currently configured so that the alerted fleeing agent will escape, unless it runs into a boundary ("cornered") or is within the range of another undetected predator that captures it. After flight, the agent switches to the WANDER state, where its next course of action is determined as above.

AVOID: This state only applies to OP and FP communicating agents, i.e., only to agents that communicate about predators. It is entered if an agent processes a message about the presence of a predator. The alerted agent moves directly away from the message source (not away from the predator's location, which is uncertain but presumably close to the message source). The agent first looks backwards (to localize the predator if possible) and then forwards (to avoid running into another predator) as it moves. As with any other state, if an agent in state AVOID sees a predator, this takes precedence and the agent enters the state FLEE.

FORAGE: An agent will enter this state if it is not fleeing or avoiding a predator, and if it has seen a food location. If so, it knows the exact location of the food site (or sites), selects the closest, and moves towards it, looking in the direction of its movement. It changes to the state CONSUME upon arrival at the target food site. Note that it is possible that food may no longer be present at a site where it was previously observed if it has already been consumed by other agents. If food is still present, the agent issues a message upon arriving at the food site indicating that it has found food (assuming the agent is an FO or FP agent).

CONSUME: Upon arrival at a food site, if an agent's food stores are below their maximum capacity, the agent consumes one unit of food per iteration until full. It will remain at the food site until either all food is gone there or it is forced to flee/avoid a predator. If the agent has filled its internal food store to capacity, it simply remains in the CONSUME state at the food site but does not consume any food until its food stores are decremented. When the food at that site is exhausted, the agent enters the WANDER state to determine its next action.

SEARCH: This state only applies to FO and FP communicating agents, i.e., only to agents that communicate about food. If an agent has not seen food or predators, and is not avoiding a predator it learned of from a message, it will enter the state SEARCH to look for the closest food site about which it has received a message. The agent in this state moves towards the message source (which was the food location) until a food source is seen or the agent arrives at the food location (assuming it does not see or hear of a predator in the mean time). The agent then enters the WANDER state, and other states (FORAGE) as appropriate.

Agent parameters/variables:
numagents = number of initial agents (randomly placed)
totagents = total number of agents ever generated
agents = list of all agents currently in the world
agentrange = distance to center of visual field
agenthearrange = max distance at which messages can be received
maxbeliefs = max number of agent memories
acont = prob. wandering agent continues in same direction
avoidtime = num iterations to spend in avoid state
foodcapac = max amount of food stores
reaptime = interval at which food stores are decremented (also, in current version, this is times at which agents' deaths due to starvation or old age are checked for)
maxage = older agents die of old age

POPULATION ISSUES

Each agent has a two bit "chromosome" where the first bit indicates whether or not the agent sends and receives FOOD messages and the second bit indicates whether or not it sends and receives PRED messages (messages about predators). Thus, an agent with the chromosome 01 processes just PRED messages, i.e., it is an OP agent. Chromosome values are assigned at the time an agent is created and once assigned are fixed.

The number of NC agents that are to be placed in the world at the beginning of a simulation is specified as the value of the parameter *numNC*. For example, if *numNC* is set to 40, then 40 NC agents will be placed at random locations in the world before a simulation begins. The initial numbers of FO, OP and FP agents are determined similarly by the values of *numFO*, *numOP* and *numFP*, respectively. The sum of these initial quantities, $numagents = numNC + numFO + numPO + numFP$, is a target value for the total number of agents to be present throughout a simulation.

If the actual number of agents present during a simulation drops below *numagents* due to death of agents, the simulator automatically creates enough new agents to replace those that have died and places them in the world at the beginning of the next iteration. The class of each new replacement agent is determined depending on the *mode* of operation of the simulator. The mode of a simulation is one of the following, mutually-exclusive possibilities:

NC, *FO*, *OP* or *FP*: All new agents are of the specified type. For example, suppose one wishes to measure how well NC agents survive under a specific set of parameter settings in the simulator. This might be determined by setting *numNC* to 100, thus indicating 100 initial NC agents, and having no other class of agents present initially. If *mode* is set to NC, any replacement agents will also be of the class NC. With such settings, the number of deaths of agents due to starvation, predation and old age over some period of time would form one measure of how successful NC agents are under the given conditions. New agents in any of these four modes are placed randomly in the world.

RAND: All new agents have randomly generated bits forming their chromosome. Thus, roughly one in four new agents falls in each class. New agents in this mode are also placed randomly in the world.

EVOLVE: This is the mode that is used most often with the simulator. New agents are evolved through simulated natural selection, mutation and crossover operations. Simulated evolution is similar to what occurs with traditional genetic algorithms using tournament selection, but involves incremental replacement of dead agents rather than discrete generations, and may take into account spatial relations of parent/child agents.

The last of these possible modes, EVOLVE, deserves further explanation. In this mode, three steps occur: 1. selection of two parents, 2. generation of two children, and 3. placement of children into the world. These steps are repeated as many times as necessary to restore the population to at least the target population size. Each time the steps are repeated, two children are produced and added to the population.

1. *Parent selection*: Tournament selection is used to identify two parent agents to reproduce[3]. In a tournament, a small set of candidate agents are picked, and the two of these candidate agents with the highest fitness are selected for reproduction (ties are resolved arbitrarily). The default explicit fitness measure built into the simulator is the current food stores possessed by an agent[4]. Two variables govern the tournament selection process. Variable *toursize*, usually a small positive integer $k \geq 2$, specifies the number of candidate agents used in a tournament. The larger that k is, the more competitive the selection process (but the more computationally expensive the tournament). The other variable, *locnpars*, determines how the k agents are selected to be candidate parents in a tournament. If $locnpars = RANDOM$, then k candidate agents are selected randomly and independently from the current population of agents. This initial selection of candidates is done *without* regard for agent fitness or the spatial location of agents. Thus, the probability that an agent of class NC, OP, etc, is selected to participate in a tournament is the fraction of the population of that class at the time candidate selection is done. If $locnpars = SPATIAL$, then one candidate agent is picked at random from the current agent population, while the remaining candidate agents are then the $k-1$ closest neighbor agents to the first one chosen (ties resolved arbitrarily). When $locnpars = SPATIAL$, another parameter *maxrad* comes into play. The positive integer *maxrad* determines the maximum distance from the first can-

[3]The term "tournament selection" is used here in the sense that term is usually used in the genetic algorithms and genetic programming literature.

[4]This is readily changed by modifying the single function *fitness* in the code. For example one might alternatively use food consumed divided by an agent's age, or the total number of food units consumed by an agent since birth or the number of times an agent has successfully fled from predators, or many other quantities.

didate agent that can be searched for the other $k-1$ closest agents to be in the tournament. If the needed $k-1$ agents are not within this distance, less than $k-1$ will be used, so when $locnpars = SPATIAL$, fewer than *toursize* agents might be used in some tournaments. If even a second agent cannot be found within distance *maxrad*, this particular tournament is aborted and the original agent is rechosen.

As noted above, selection of the *toursize* agents that are to participate in a tournament is done without regard to their fitness. An agent's explicit fitness measure is only used in subsequently selecting the two most fit of the k candidate agents to be the actual reproducing parents. As a result of the above method of selecting parent agents, if the tournament size $k = 2$ is used, the explicit measured fitness of agents does not enter into parent selection process at all (only the "implicit fitness" of having survived predators, starvation, etc. and thus remaining in the population plays a role).

2. *Generation of children*: From the two parent agents selected as described above, two new children agents are created and added to the population. One new child agent has its genome set to that of one parent, while the other new child agent has its genome set to that of the other parent. The following modifications are then made to the chromosomes of the child agents:

- with probability *pmf* the food bit in each child agent's chromosome is mutated (flipped);
- with probability *pmp* the predator bit in each child agent's chromosome is mutated (flipped); and
- with probability *pc*, crossover occurs (there is only one crossover point possible).

Note that the chromosomes of the two child agents may end up by chance to be equivalent to those of the two parent agents, and will end up with certainty to be equivalent to those of the parents if $pmf = pmp = pc = 0.0$.

3. *Placement of children*: The above process generates two new agents to be placed in the world. These new agents replace others that have died during the last iteration, and not the parent agents which remain in the world. How the new agents are placed in the world depends on the value of parameter *locnkids*. If $locnkids = RANDOM$, the two new agents are placed at random locations in the world. Otherwise, *locnkids* must be a small non-negative integer designating the radius of the neighborhood around the parent agents in which the new agents are to be randomly placed. The child agent derived from each agent is placed within the neighborhood of the parent agent from which it was derived. For example, if $locnkids = 0$, each child agent will be placed within the same cell as the parent agent from which it was derived (it will differ in its orientation, etc.). If $locnkids = 2$, each child agent will be placed at a random location of the 5x5 neighborhood centered on its parent agent. The initial direction of movement of child agents is random.

Parameters/variables related to population evolution:
numNC, *numFO*, *numOP*, *numFP* = initial numbers of NC, FO, OP and FP agents, respectively
genome-size = number of chromosome bits (2 in this version of simulator; do not change)
ncgenome, *fogenome*, *opgenome*, *fpgenome*
= computed number of NC, FO, OP and FP agents generated, respectively

toursize = tournament size during selection (integer ≥ 2)
locnpars = RANDOM or SPATIAL
pmf = probability of mutating a food bit in a chromosome
pmp = probability of mutating a predator bit in a chromosome
pc = probability of crossover
maxrad = maximum radius allowed in spatial selection process
(must be less than half the smallest world dimension)
locnkids = RANDOM, or non-negative integer representing the maximum distance from parent that a child may be placed (initial agents are always placed randomly; only evolved child agents may be placed spatially)

USING THE SIMULATOR

The simulator is implemented in Allegro Common Lisp. Save the simulator in a single file which will be called *main.lsp* here. To start an interpreted simulation, simply sign onto Allegro and enter

(load "main.lsp")

at the prompt in the same directory in which Allegro was started. The simulation is set to run automatically and to terminate after *tmax* iterations. The results will appear in the files res.state, res.disp and res.log as described above. The simulator can be compiled and runs qualitatively faster once compiled. Any non-trivial simulation should be run after compilation. The simulator can be compiled and loaded, for example, after starting lisp, by entering

:cf main.lsp
:ld main

Parameters that can affect a simulation are located at the beginning of the simulator file. These are largely set at arbitrary values used in a recent simulation. Due to the stochastic nature of simulations, precise results may vary upon repeating a simulation, so any specific simulation should be run multiple times to obtain confidence in the results. Note that the tracing and display of world states on every iteration can be a substantial factor in slowing the system.

The last two lines of the simulator file are:

(control)
(exit)

These start a simulation and terminate the lisp session, respectively. Elimination of the call to (*exit*) at the end of the file will leave you talking to the lisp interpreter at the end of a simulation (this might be useful should one want to inspect data structures at that time). Eliminating both (control) and (exit) will load the simulator but not start a simulation.

A Perl script has been written that allows one to run several instances of a simulation (with the same simulation parameters) in parallel on a pool of Unix workstations. Each instance is the same simulation but with a different random seed. After all instances have terminated, the output files of each instance are parsed by the script. Among other things, the script averages over all output values of all instances and displays averaged results.

Distance measures in the simulator are not Euclidean. The distance between two points is the minimum number of cell-to-immediate-neighbor-cell steps required to move between those points, where individual steps are horizontal, vertical, *or diagonal.* Thus all cells equidistant from a specific given cell form a square.

SOME FINAL COMMENTS

The simulator described here has several obvious limitations, including:

1. Not optimized for speed (slow) in that the implementation is done in Lisp
2. No interface for the non-programmer; no graphics interface
3. Rather impoverished world (does not have landmarks, water, prey, sound, speed variations, cover, visual obstacles, more than one predator, noise, more than one food type (none mobile; isolated and not clusters; poisons), seasons; day/night; non-communication interactions between agents (awareness; fight over food; mating; cooperation)

Two issues that may merit further evaluation and modification in future versions of the simulator are:

1. The simple, fast *movetowards* function currently used biases entities to move towards their targets from horizontal/vertical directions (and not diagonal directions) as they get closer to the target. This could be fixed in *movetowards* by computing a normalized direction vector pointing at the target, and using its components as probabilities of moving in the corresponding coordinate directions. Cost: Extra real-valued arithmetic, and complexity.

2. Agent visual scan should create base (width) and scan out from there progressively to permit extension to obstacles by blocking outward progression. Now it just selects a block of cells in the correct direction and scans all of its cells.

At present, the simulator is being used for a series of experiments to determine conditions under which signaling emerges. The results of these experiments will be the subject of a future report.