UNIVERSITY HONORS PROGRAM

SENIOR PROJECT - APPROVAL

Name: James Davis

College: Arts & Sciences Department: Computer Science

Faculty Mentor: Bruce MacLennan

PROJECT TITLE: Creating Worlds - The Design and Implementation of an Artificial Environment Meta-tool

I have reviewed this completed senior honors thesis with this student and certify that it is a project commensurate with honors level undergraduate research in this field.

Signed: _____, Faculty Mentor

Date: May 13, 1998

Comments (Optional):

Creating Worlds

The Design and Implementation

of an Artificial Environment Meta-Tool

James Davis

May 13, 1998

Abstract

This project is the beginning of an attempt to design, construct and document a computer based artificial environment creation and manipulation tool. Of the few existing computer based artificial environments, most were constructed to study a specific phenomenon and are inflexible in use. My tool is an attempt to create a structure and a core set of objects and functions upon which others can build. The tool is simple enough that it can be used for educational purposes, but still complex enough to be applied to more academic and scientific problems. The tool and its uses cross a number of disciplines including all of the life sciences, sociology, psychology, and to a lesser degree economics and political science–it can be used to study the evolving and adaptive nature of all of these fields.

1 Executive Summary

Computer based artificial environments are excellent places to create and study evolutionary systems of all kinds. They have been used to model environments from simple predator-prey scenarios to complex multi-species systems. From simple educational entertainment to "hard-core" science, artificial environments present a forum in which the elusive concepts and complex nature of evolving systems can be tested, studied, and better understood. However, depending upon the needs of the creator, artificial environments have been primarily hand crafted to fit a particular set of parameters. This specialization limits the new environment to the scope for which it was written. Another problem is the limited number of publicly available "pre-packaged" environments for the casual consumer. Not only are they difficult to find and compile, but they often have difficult interfaces that are not intuitive and are therefore restricted to more advanced users.

What I have attempted to do is create a generic artificial environment meta-tool that can be used to compose and view experiments using a simple text interface and a graphical interface. Although the basic use of the tool is designed to be simple, the tool itself can be expanded to allow more complex use by way of more advanced features, objects and functions. The simplicity of the interface lends itself to being used in an educational environment by non computer experts, but the package itself will be expandable to accommodate advanced use as well. The construction of this tool is modularized so that code can be removed and inserted for the creation of more specialized environments. When refined to its final state, this code will be put in the public domain in order to serve as a starting point for anyone who wants to extend it.

2 Previous Work

Since this was more of a creative endeavor than a reasearch project and because I wanted to create something largely original before I poured over the code of others, I kept my literature review to a minimum. The amount of literature was very small anyway–artificial life (a-life) and the creation of artificial environments are relatively new fields. Most of the work on artificial life has been carried out at the Santa Fe Institute [1] and by the universities and departments associated with it. A number of environments exist from upon which mine will

[1] http://alife.santafe.edu

seek, in some way, to improve. Two of the most well known examples are explained here along with their strengths and shortcomings. A third, which is currently under development, looks to be a more professional version of what I hope to accomplish in my project.

2.1 MANTA

"Ants are to artificial life as *Drosophila* are to genetics."[2] It is with this in mind that Alexis Drogoul[3] created **MANTA**– **M**odeling and **ANT**nest **A**ctivity. This simulator focuses solely on ants and the emergent behavior demonstrated by their societies[2]. Although this program is complex in form and is a splendid example of modeling a biological phenomenon in software, it is extremely limited in its functionality by the rigidness of its agenda. In this case, the simulation is restricted to ants, and, with some reworking of the code, other socially oriented creatures. The simulation only studies social behavior of *reactive agents*, or agents that do not learn or think, but simply react to their environment.

2.2 LEE

LEE, or **L**atent **E**nergy **E**nvironments, is a package developed by Fillipo Menzcer and Rik Belew [4]. It is a very simple single-purpose environment in which agents move about their environment collecting "atoms" and learning what atoms can be combined with others in order to produce energy [5]. The results of each action and its effect are propagated through the agent's neural net influencing the future behaviors of the creature. Here again we see an artificial environment that was designed, essentially for a single experiment or set of experiments. **LEE** was not designed as a general artificial environment.

2.3 SWARM

SWARM is a project that is being carried out at the Santa Fe Institute. While I have not seen it in run, the documentation [6] for it is impressive. While my project will, if the

[2] Artificial Life Simulators and their Applications Gutowitz, Howard
[3]University of Paris
[4]University of California, San Diego
[5]Latent energy environments: A tool for artificial life simulations, Menzcer and Belew
[6]http://www.santafe.edu/projects/swarm/intro-material.html

comparison is ever made, most likely fall under the shadow of SWARM, I hope to provide a more open basis. I plan on writing my code in such a way that adding new code or modifying the code are non-prohibitive undertakings. Thus, the basic structure can be tailored to a specific problem, as well as new objects and environments created through simpler tools.

2.4 Summary

The work done thus far in the creation of artificial life environments has been primarily problem specific. The environments all concentrate on creating a specific world with specific organisms, rules, surroundings, and goals. These environments limit the scope of what can be performed and simulated. Other environments, while being, perhaps, a bit more expandable, are unwieldy and difficult to use. In addition, most of these environments have been crafted for research and were not very adaptable to educational purposes. Only one package that I could find seemed to be made for the same purpose as my tool, and even it seemed to be restrictive in allowing creation of new environments.

3 Purpose

My purpose was to design, construct, and document the base structure for an artificial environment development tool. In the limited time available, I realized that the completion of such a task would be difficult, and therefore I planned to complete the basis for a system that could be built upon later in my academic career. I outlined a list of tasks that are detailed in my Statement of Work section. When I began the project, I had a list of properties which the tool should have:

- It must be modular–this would be the key to it being a general purpose tool. Instead of having a rigid structure that would be difficult to change, I wanted my tool to be easy to mold to the needs of the user.
- I wanted it to be useful enough that the academic or hacker community would create libraries for it–increasing the scope and capabilities the tool had in its basic package. This would mean a wider range of capabilities for those who are not programmers.
- It must be user friendly. Instead of being a strictly academic tool, I wanted mine to appeal to educators with its ease of use and gently sloping learning curve.

- While being user friendly, the tool should also be expandable to a complex and powerful command set. This command set would have a much steeper learning curve, but would only be needed by those using the tool in a much more in depth manner.
- My desire for two tiered use lead me to envision a multiple interface model: a graphical user interface which would display the results of experiments and give the user a view into the artificial world, and a text interface which would allow the user (with or without the hindrance of the GUI) to use a large and powerful set of commands and macros. Only six to ten commands would be needed for the most basic experiments to be run, but the user would have several dozen to several hundred commands available if he/she needed them.
- The GUI was designed to follow the fairly standard window template, with input taking the form of buttons, menus, and simple text boxes.
- The text interface was designed after the Matlab[7] model in which command lines are entered at a Unix-like prompt, results of the command are output, and the prompt allows the user to enter the next command. From the command line, the user would be able to access information about the state of the environment, to look "inside" the agents to examine its genetic code, and to generate statistical data about the environment and about the agents in the environment.

4 Statement of Work Plan

This is a brief summary of the design and implementation steps I planned out in the early stages of my project. I was not sure, at that time, how far I would be able to progress in one semester. An asterisk (*) indicates points that I did not complete during this phase of development. This section is largely for completeness and may not interest all readers.

- Gather information about previous projects of a similar nature. Study shortcomings of previous work and consider improvements as well as aspects that are desirable in my product.
- Find the most basic structural components and determine the best method by which to divide the environment into objects.

[7] http://www.mathworks.com/

- Lay out the program on paper and determine what functions need to be created and run within the program.
- Design the functions so that they can be replaced by other functions with the same variables and information being passed between the functions.
- Determine what modules need to be created and what functions will comprise which module. The modules are the strength of this product. Unlike other products of a similar nature, mine will be entirely modular. This will allow other programmers to build on the basic structure *ad infinitum*. It also allows the environment to be customized to fit a number of needs, thereby ending the need for an application to be designed and written for each new experiment or need that arises.
- Code the modules using the paper structure as a blue-print.
- Compile and test the most basic modules. Modify structure based on any problems that are encountered. Rework the paper model to reflect changes that were made to the plan. Verify that the updated paper model still meets all the original criteria.
- * Create alternate modules for different environments. Fine-tune the interchangeability of the modules. The interchangeability of modules will need to be seamless in order for the product's goal to be achieved. Creation of several modules will allow demonstration of this feature as well as increasing the range of performance.
- Design the user interface. This interface will be text based. The interface will be the user's tool box for the creation of environments. It will need to be powerful enough to accomplish all the desires of the user. Continuous updating of the interface in the initial testing stages is inevitable and will allow it to evolve to meet the needs of the user.
- Create a user interface command set that will allow all the desired activities to be performed by the user.
- Code the user interface and the parser that will interpret the user's commands and call the corresponding function.
- Test the entire product thus far.
- * Add libraries and modules to increase the functionality of the product.

- Write documentation for the interface and the code.
- * Allow users to test the product using the documentation for instruction.
- * Revise interface, product, and documentation based on user response.
- Design graphical user interface (GUI) on paper. Get user feedback on initial sketches. This step is also key as the GUI will be the primary means by which non-experts will be accessing the system. The creation of the GUI will allow the product to be used in classroom environments and in other educational, non-research applications.
- Code the graphical user interface.
- Create documentation for GUI.
- * Test product and graphical interface on users. Modify product, GUI, an documentation based on user input.
- * Announce product. Allow free distribution and manage creation of new objects, modules, and libraries.

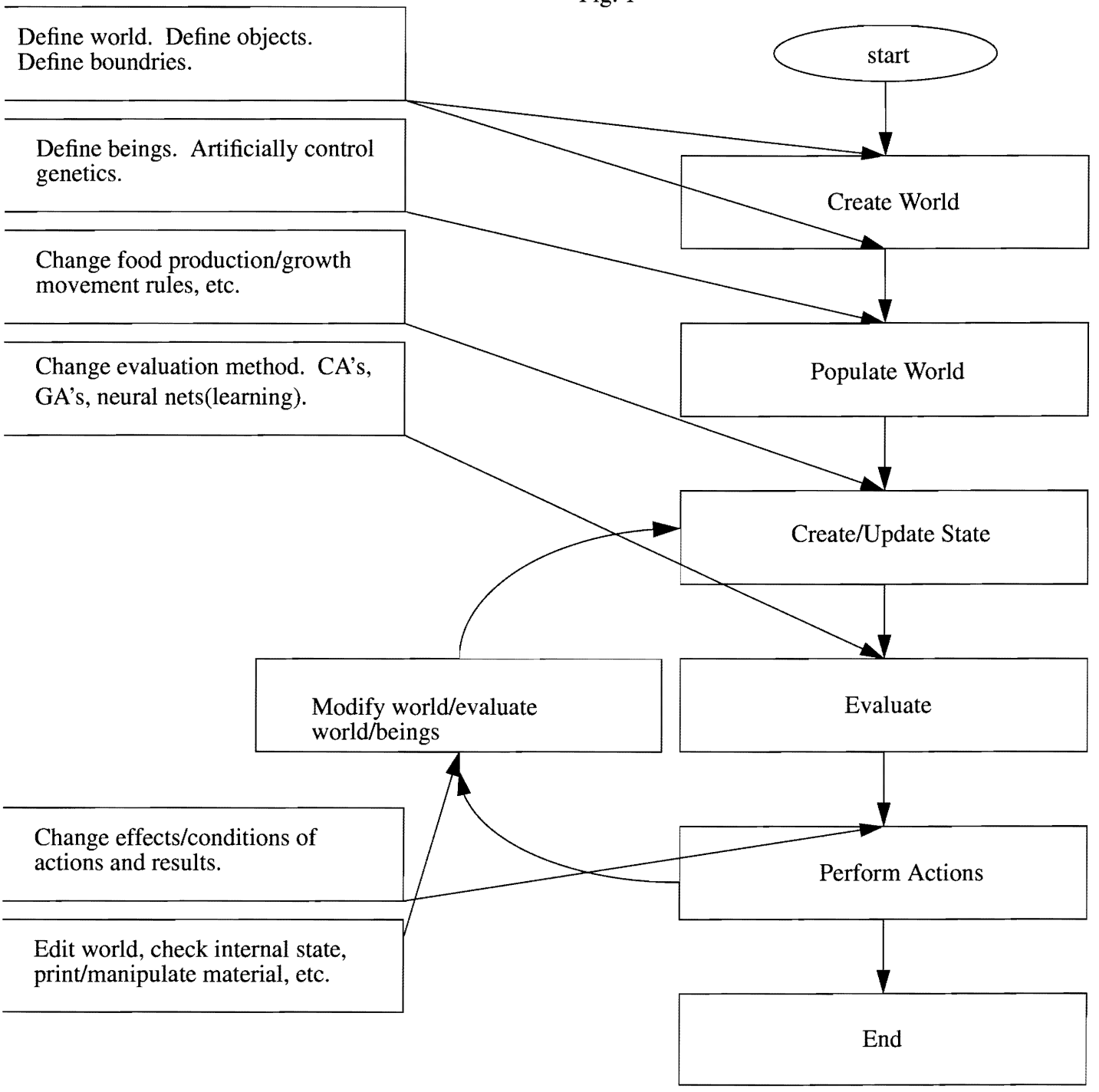
5 Results

I was very pleased with the results. I was able to get a good structure down and was even able to begin on tricky parts like the GUI and making it interact with the program. The only parts of my work plan that were not reached were the creation of extra modules (which would have demonstrated the adaptability of the tool) and the testing on users, which, of course, is essential for creating a user friendly product. Nevertheless, it was, I think, quite a first step given the time constraints.

I designed a structure based on other models that I had seen. This structure (after initial setup steps) consists of an evaluation step during which each agent evaluates its surroundings and, based on a decision algorithm, decides upon an action; an action step during which each agent passes the results of its evaluation to the world's action "resolver" to determine the effects of the action upon the agent and its environment; and an update step during which ages are incremented, energy levels adjusted, offspring created, and other general bookkeeping resolved. In *fig. 1*, I have the major modules (conceptual) mapped out showing the

general flow of the program (right). I've also shown the insertion points for where future modules will be able to affect the make-up of the world (left).

Fig. 1


Define world. Define objects. Define boundries.
Define beings. Artificially control genetics.
Change food production/growth movement rules, etc.
Change evaluation method. CA's, GA's, neural nets(learning).
Change effects/conditions of actions and results.
Edit world, check internal state, print/manipulate material, etc.
start
Create World
Populate World
Create/Update State
Evaluate
Perform Actions
End
Modify world/evaluate world/beings

Along with the basic structure, I fashioned a simple experiment using agents that could perform five actions: turn left, turn right, move forward, eat, and mate. The creatures use a simple state machine lookup table (encoded in their genetic material) to determine their actions based on the contents of their current location and the location directly ahead of them. I ran two variants of the experiment, one in which the creatures were allowed to eat deceased comrades (appendix A), and the other where cannibalism was strictly prohibited and the creatures were forced to live upon the food that grew naturally in the environment (appendix B). The results of the experiment were extremely interesting (especially visually): Whereas the cannibalistic group grew in increasingly large clusters (where the deceased provided abundant food), the foragers were forced into traveling herds and singletons who would constantly move in search of food.

6 Recommendations

My recommendations in this case will be largely targeted at myself. There are a number of improvements that need to be made in future generations of this program. Some of these are large, but I think the product will be in an impressive and releasable form within 12 months (before Summer 1999).

- The most basic improvement is that it needs to be rewritten in a more object-oriented language (C++ most likely). While this sounds like a large setback, it really just involves a few minor conceptual changes and a couple of days of code writing. This rewrite will allow me to repair some of the rougher portions of my code (inefficient, unintuitive or overly obfuscated), and better document the code.

- Relating to the rewrite and to the object-oriented nature of the next generation–I need to make the modularization stronger. In the current version there are still a few dependencies between modules which must be severed to achieve the initial goal of modularity.

- The GUI needs to be enhanced. The prototype is very useful and provides a very effective visual tool for watching the development of the environment, but it still is not able to send signals *back* to the program. This is essential in the finished product.

- More libraries and objects need to be designed and created. For this product to be of initial interest to the artificial life community or any of its member fields, it needs to

have a large number of interesting demonstration before individuals will be willing to contribute to its growth.

- A large task that will be worked on in the future is the creation of more and better statistical and reporting utilities. In its current form, the program is able to report on the current state of the environments and its agents, but not on the history or the trends that the environment has experienced. These tools are essential for anything more than entertainment or basic educational use.

7 Summary

In summary:

- Artificial environments provide an excellent opportunity to research complex adaptive phenomenon. They are also an effective teaching tool, giving a visually instructive examples of evolutionary activity.

- Despite the many uses of artificial environments, most that are created are designed to accomplish only a narrow band of tasks. The structure of these programs is too restricted to allow them to be expanded to more general tasks. While each program is excellent for the purpose it was created, it is not as useful as a general platform for experimentation.

- As a result of the above fact, each new experiment or model that needs to be created necessitates the recreation of the entire environmental structure. Most components of that structure are common to all such applications. If the common structure and functionality were built into a platform that supported limitless expansion, tremendous amounts of time and energy could be saved in the development of the experiment.

- What I have designed, programmed, tested, and documented is the beginnings of a multi-purpose artificial environment meta-tool that will be capable of creating any type of artificial environment populated by any type of agent(s) for research, educational, or entertainment purposes.

- This tool is largely modular and will, in its next version, be completely modular. This means that later developers will be able to add an even broader variety of decision

making and evaluation algorithms, maps, terrain features, agents, reproduction methods and scenarios without reworking the basic structure of the program. This increased capability will allow a body of interested developers to continuously add new features that can be used by the larger community.

- There is a dual interface. The graphical interface allows the user to visually inspect the progress and results of the experiment and will, in the future, allow limited control over the environment. The text interface allows complete control over the environment and should include a number of statistic compiling and reporting devices in the next development.

- The entire product will be put in the public domain upon completion allowing developers to begin the process of creating additional functionality for it.

Appendices

Appendix A

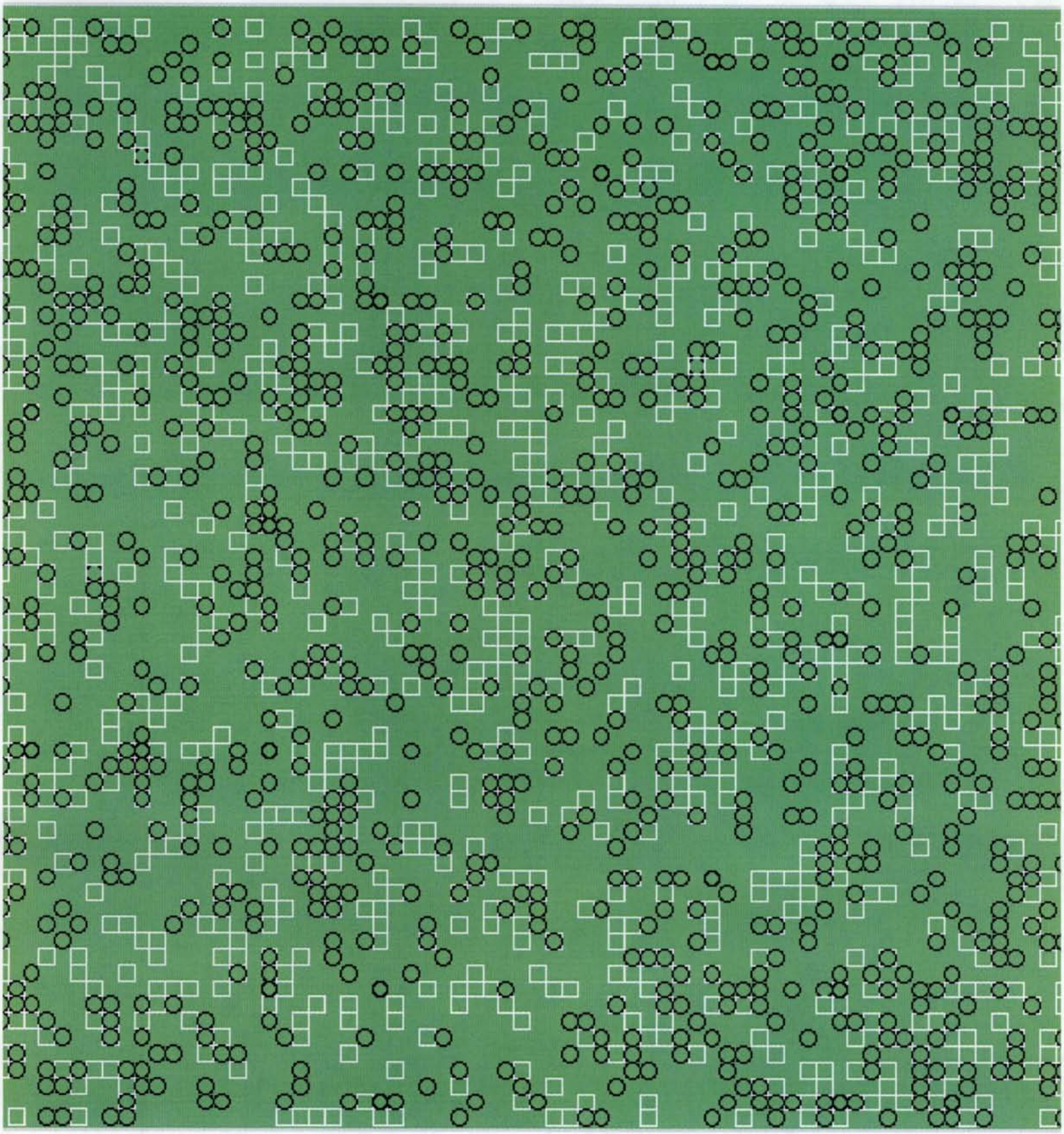
1. After 1 generation: In the first screenshot, the environment is initialized to a random state with 1000 food and 1000 agents. Each agent has a small amount of initial energy, and in this experiment, dead agents can be used by others for food. In this initial state, the majority of the creatures are unfit to survive.

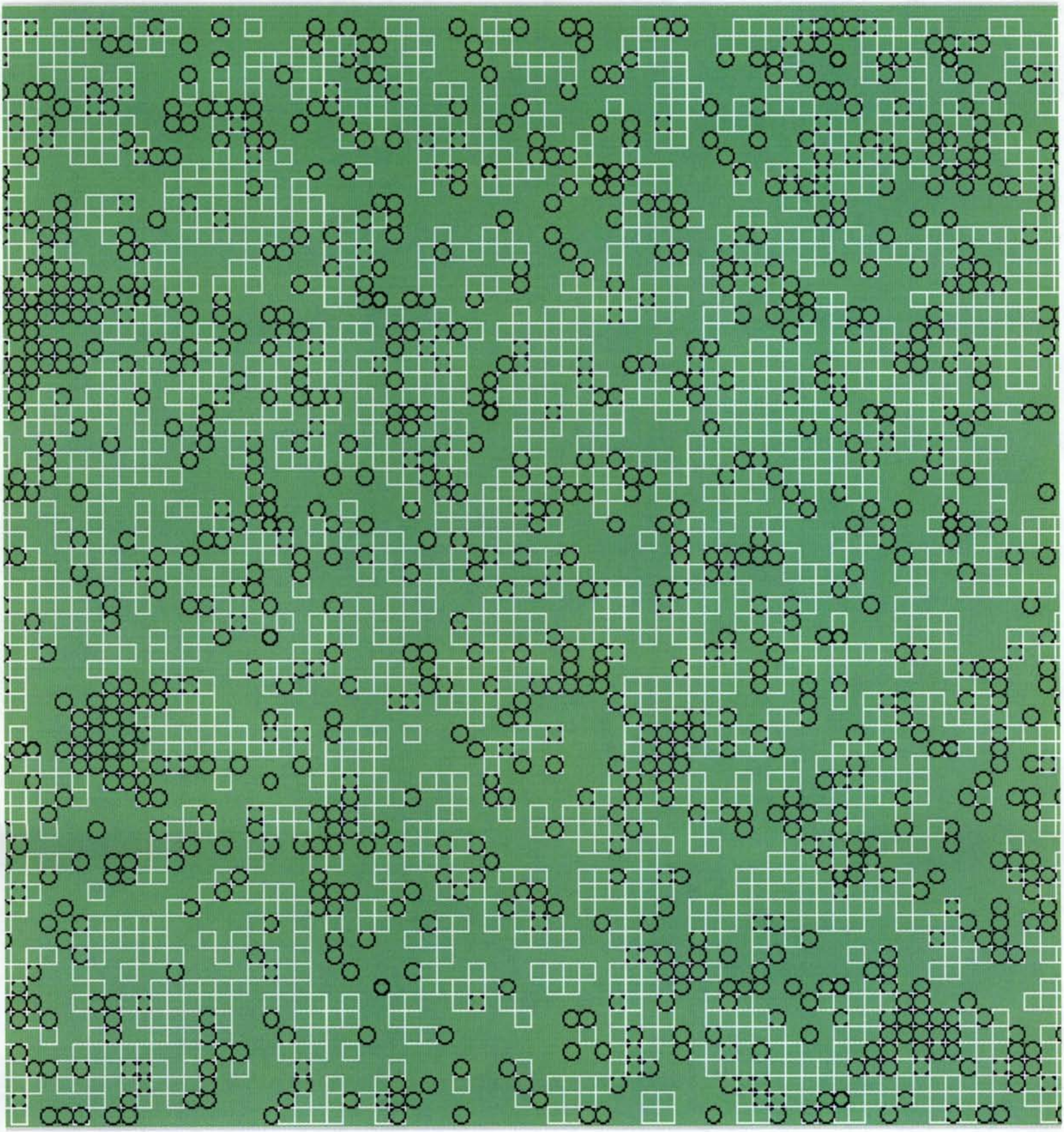
2. After 50 generations: Notice the clusters of reproduction on the left and the one in the bottom right corner. At this point creatures who have reproduce and haven't eaten are dying and creating food for the others, but the majority of the agents haven't used up their initial energy.

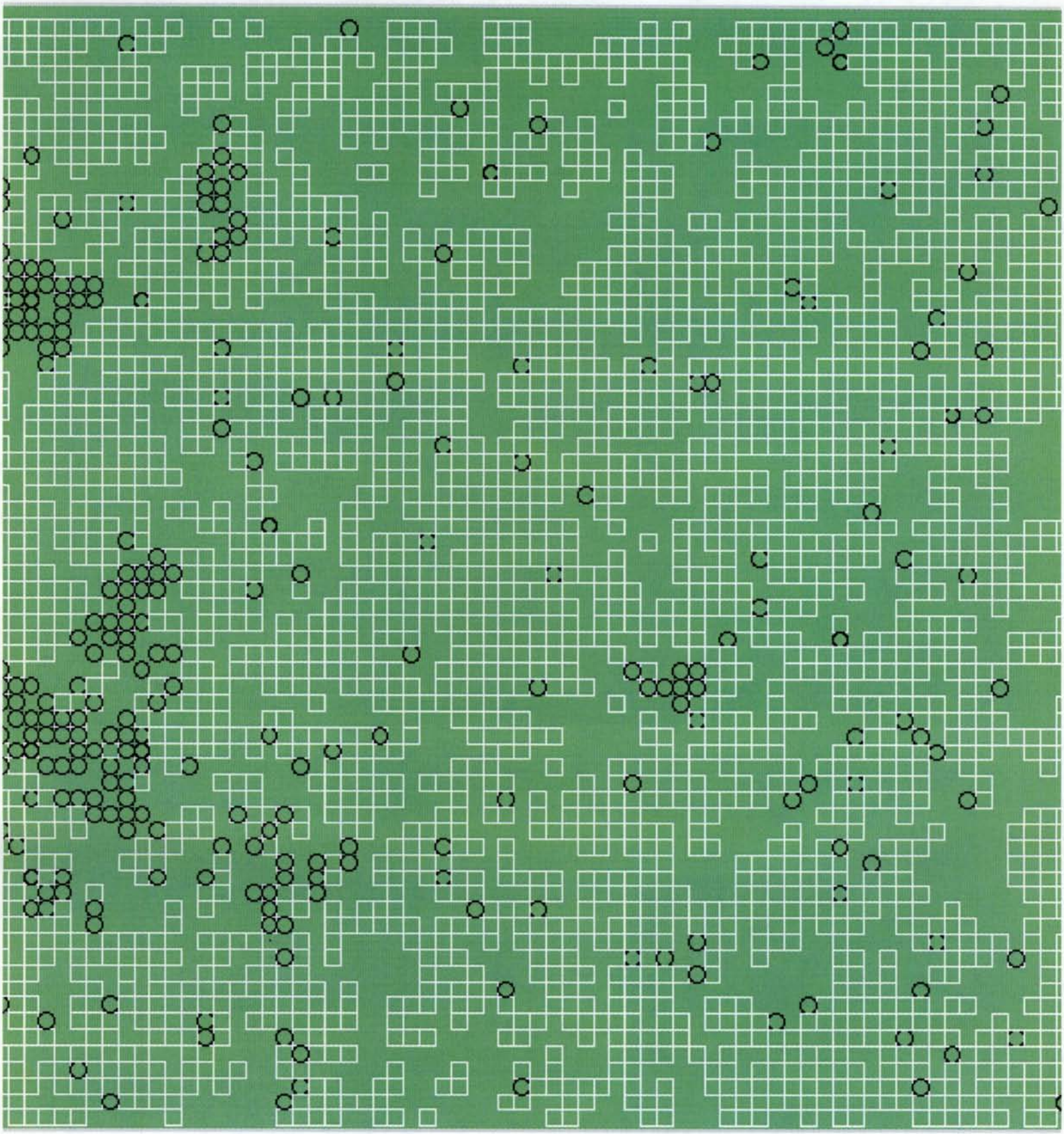
3. After 101 generations: The Third shot is taken immediately after the initial energy has been used up by the creatures. Notice the sudden abundance of food. All the creatures who are still alive have, at the very least, the proper instincts to eat sometimes. Notice the clusters of agents that are successfully mating (some creatures will not mate, they're genetic codes won't be reproduced.

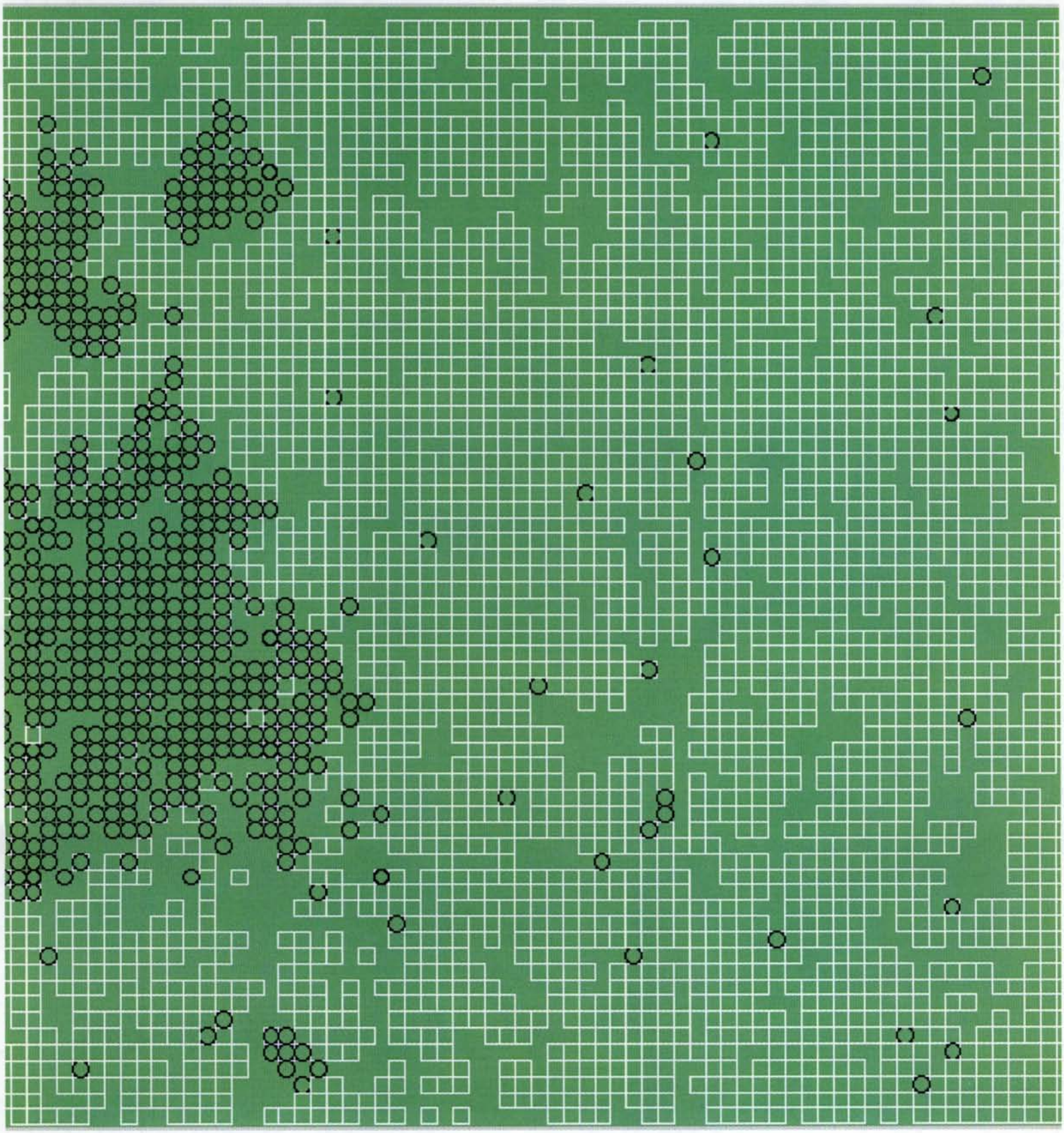
4. After 500 generations: Here the surviving creature are successful survivors, now we know the species will likely never die off (unless we introduce a new element into the environment).

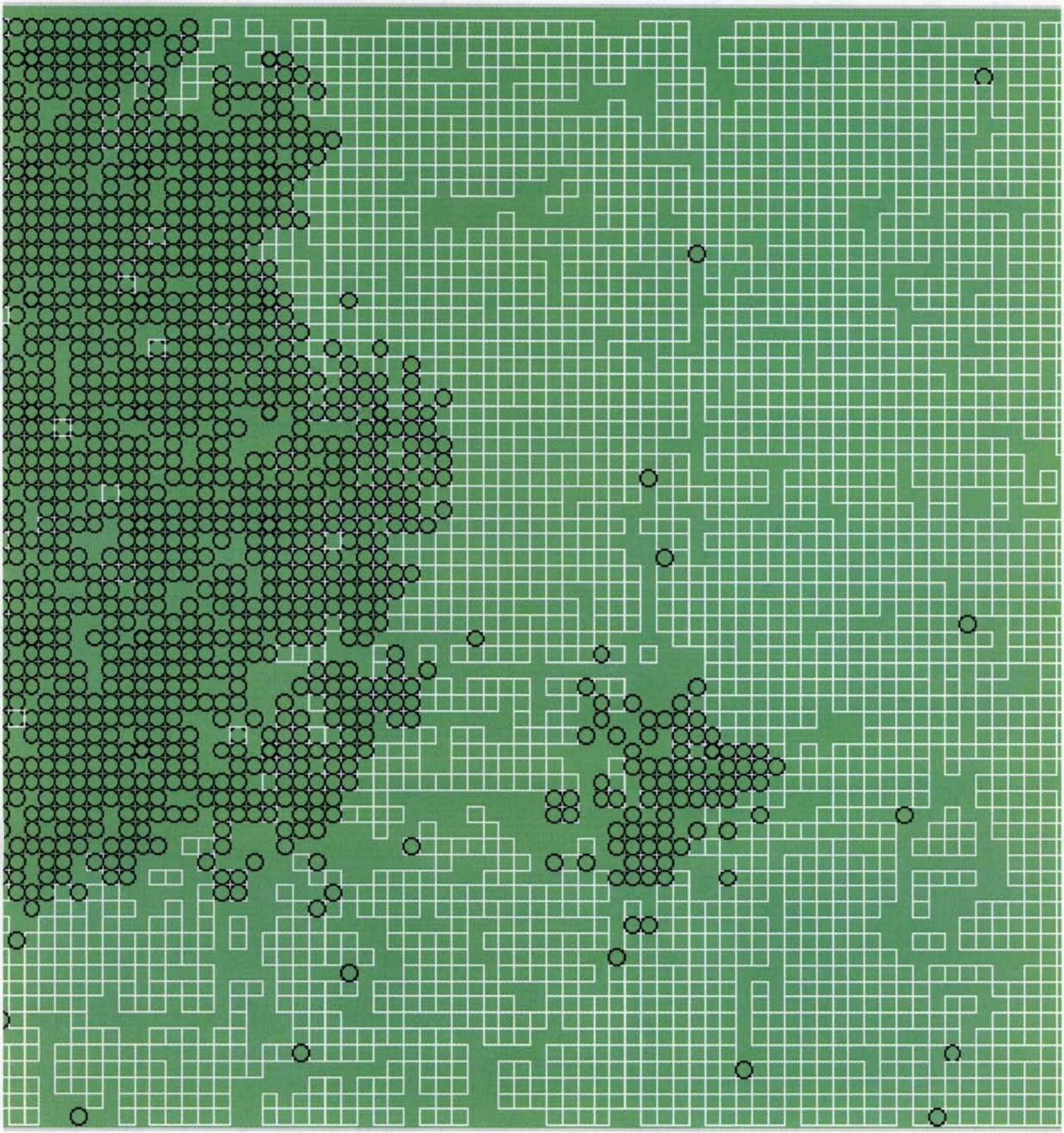
5. After 1000 generations: Not much new in this slide, the left two groups have grown together. Notice the sudden growth from just a few parent agents in the middle-lower-right section.











Appendix B

1. After 1 and 50 generations: Map looks much the same as the first two snapshots from the last appendix (not included in this appendix). This experiment's agents are not allowed to eat deceased agents. That is a large disadvantage.

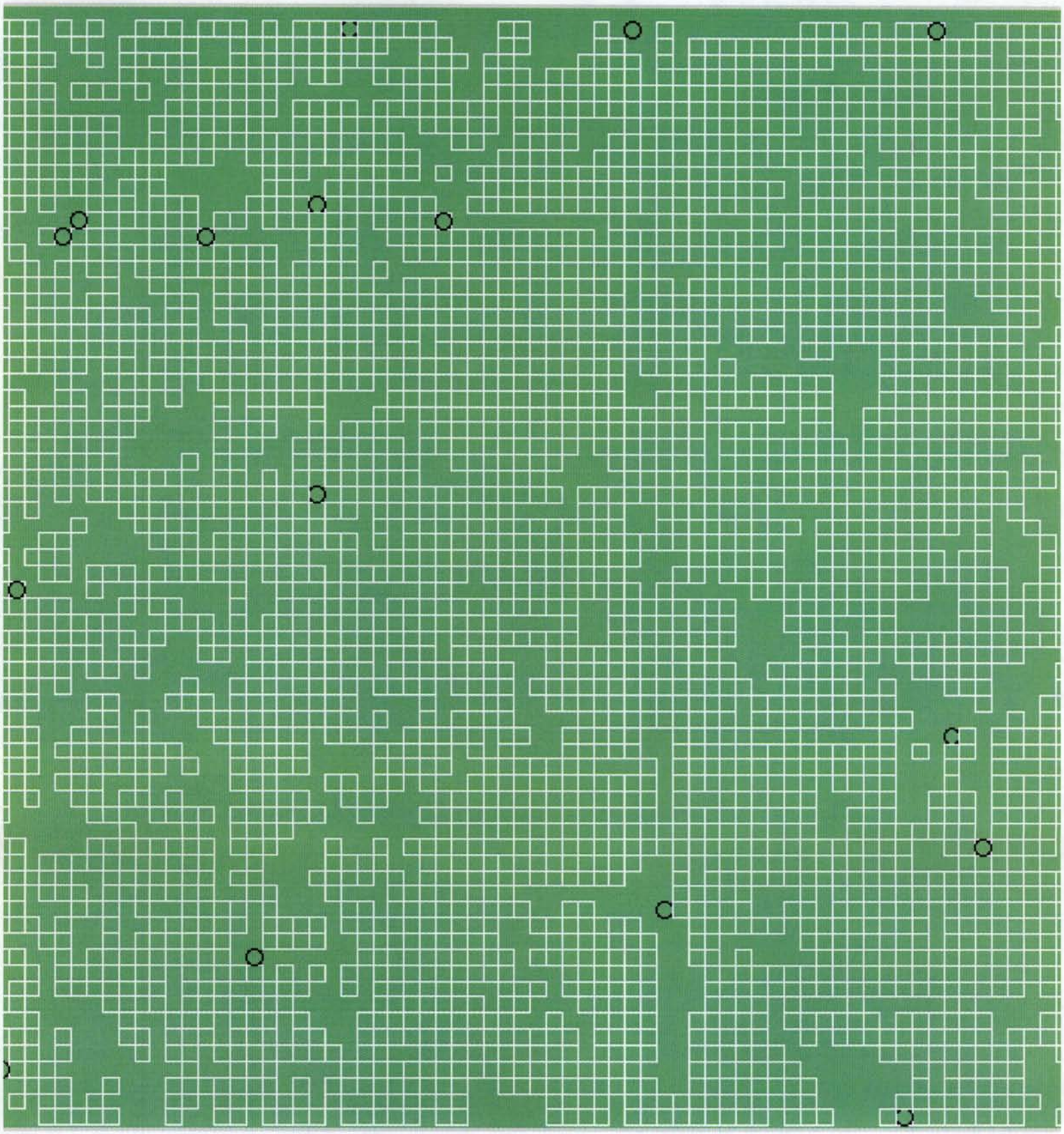
2. After 101 generations: The first snapshot in the appendix is just after the initial energy supply ran out. There are about 14 surviving agents out of the original 1000. At this point it seems our species will fail.

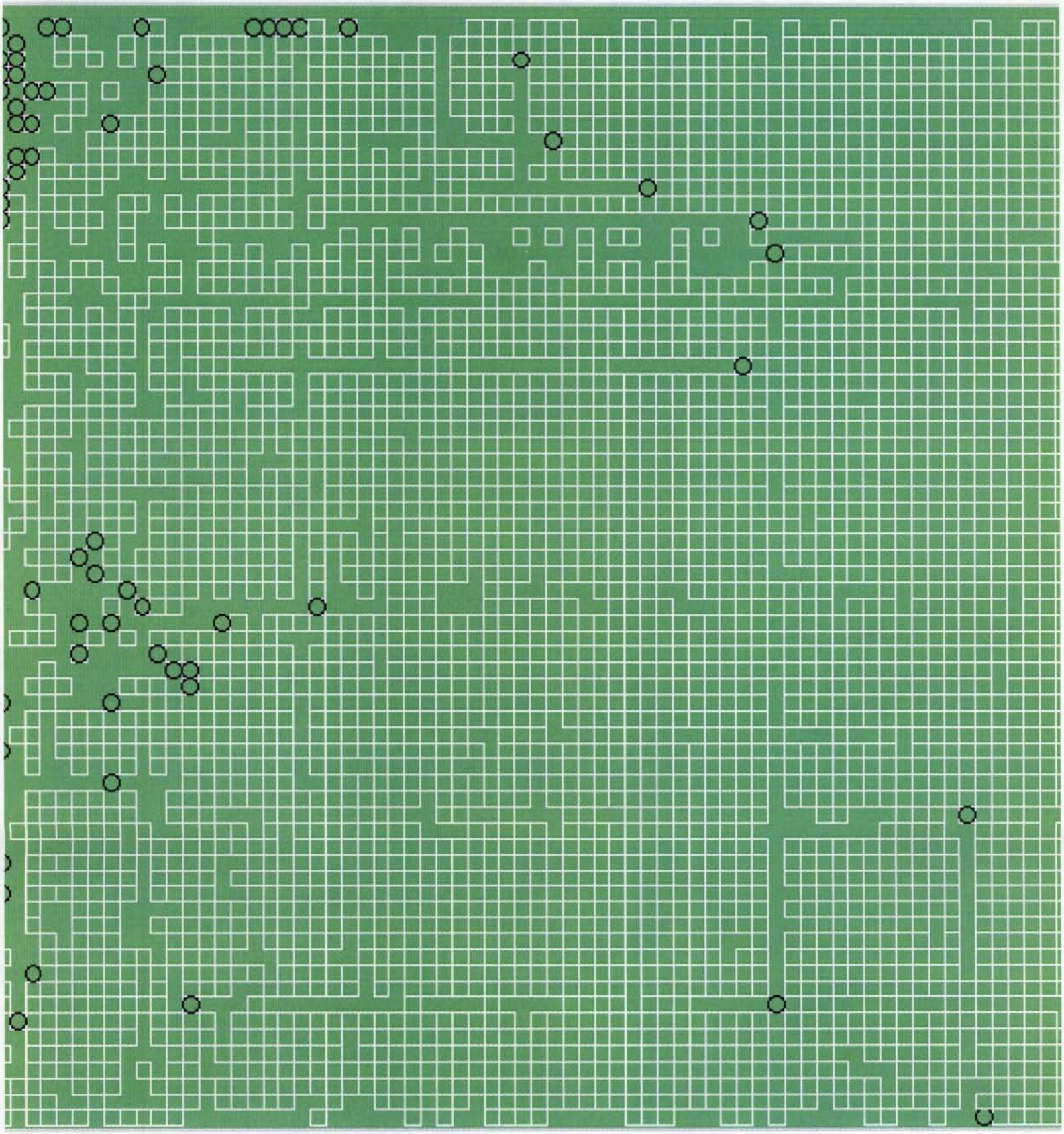
3. After 200 generations: A couple of wandering agents have met up on the left and have had a number of offspring which all started moving away from the parent agents. Notice the trails of eaten food behind the agents. These are the agents who have evolved the behaviors of eating and moving.

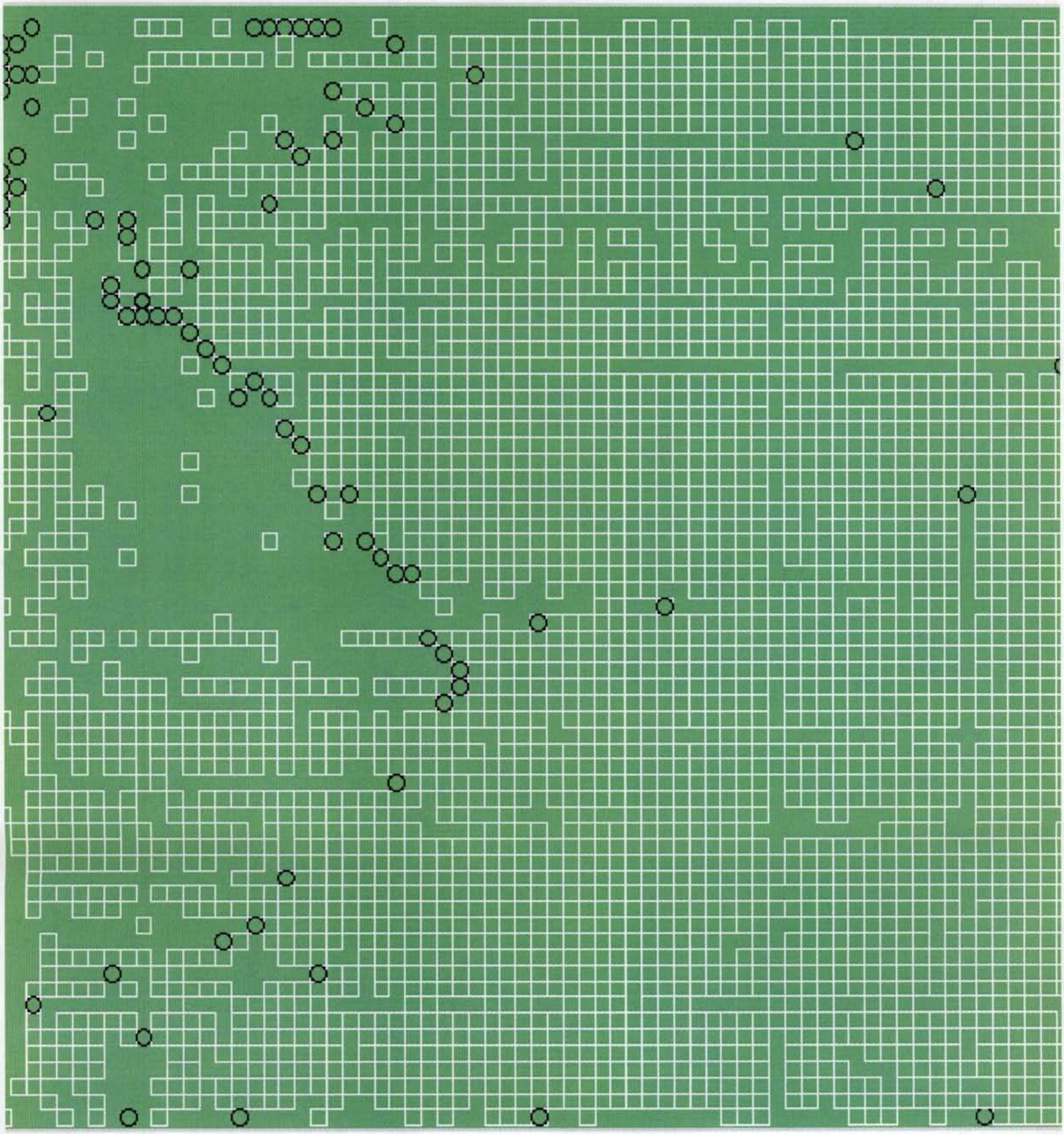
4. After 225 generations: Only a couple of generations later, there's a mass of reproduction in the upper left and middle left of the screen.

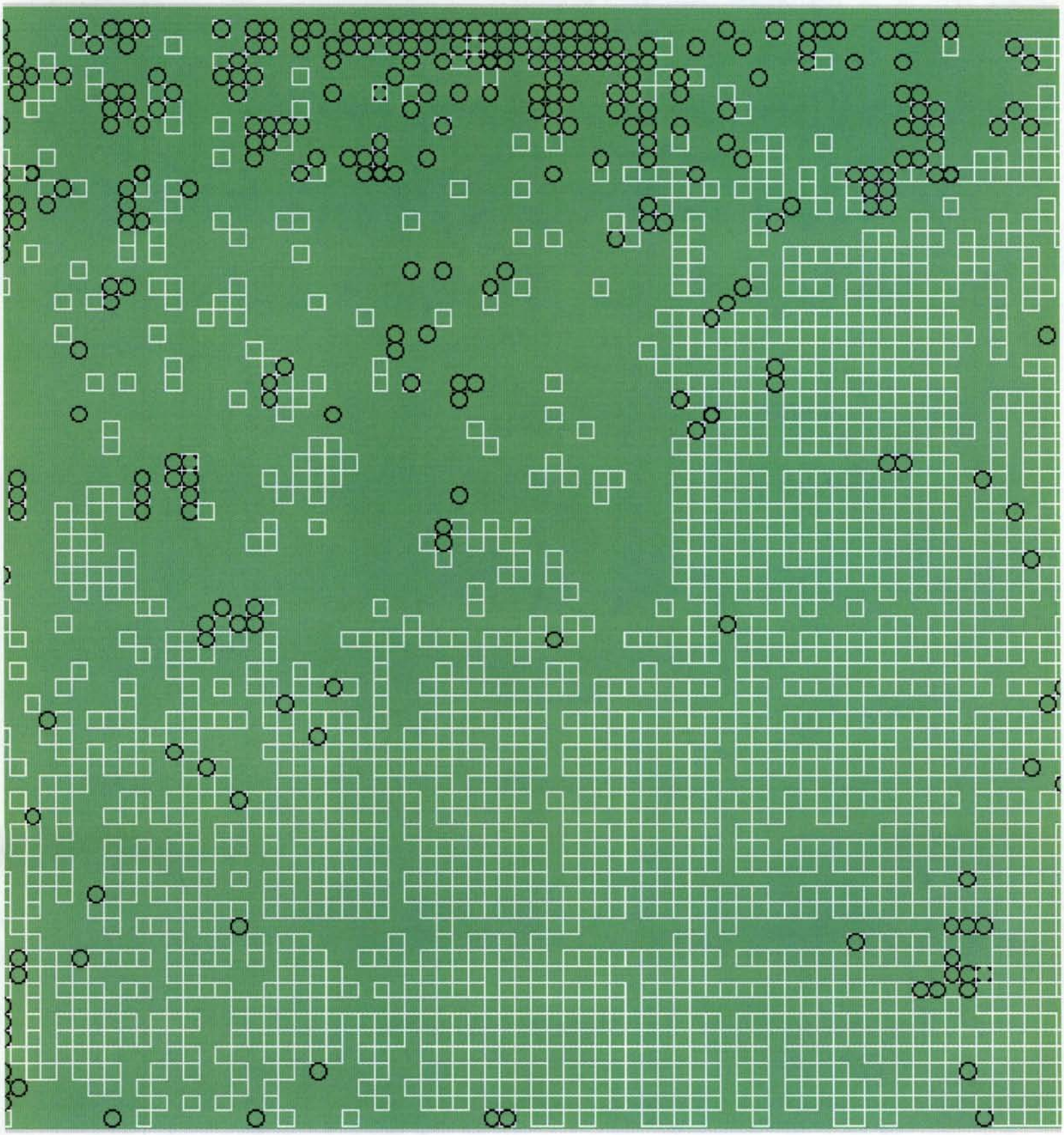
5. After 325 generations: After a period of little activity a large number of offspring turn left and start eating their way across the screen (the formation is the result of inbreeding, under similar conditions these creatures that respond based on their genetic code are all responding the same way).

6. After 500 generations: Back from near extinction, the agents have made a full recover. Food is slow to grow back and so moving creatures have the upper hand. Over the next 10000 generations, population and food enter a basic periodic function of growth and decline.









Appendix C

```
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<tcl.h>
#include<tk.h>
#include"actions.h"
#include"enviro_make.h"
#include"utilities.h"
#include"create.h"
#include"interface.h"

/*********************************************************************
* Name: enviro_make.c
*
* Purpose: This is the coordinating module.  Ideally, it shouldn't
* do much, but there are a couple of global variables that need to
* be initialized, and in this version, alot of setup of experiments
* is done here that will be moved later into user input and scripts.
* Eventually, all main will do is initialize variables and start
* the interface.
*
* Functions: None
*
*
*********************************************************************/

/*********************************************************************
* function: main()
* purpose: coordinate the activities of the program
* inputs: none (will be command line arguments in later versions)
* returns: no explicit returns
* side affects: N/A
*
*********************************************************************/

main()
{
  time_t t;
  int i, j;
  int decision;
  int cur_x_loc;
  int cur_y_loc;
  int cur_loc;
  int for_loc;
  int facing;


  /**** Initializing global variables ****/
  dead_index = -1;
  removed_index = -1;

  /**** Setting environment variables can be reset through interface ****/

  food_energy_value = 10;
  max_food_growth = 1;
  max_food = 3;
  food_growth_factor = .005;
  create_dat = 1;
  beings = 0;
  food = 0;
  dead_food = 1;
  max_age = 300;

  t = time(NULL);   /** seed random number generator **/
  srand48(t);
  for (i = 0; i < MAX_POP; i++)  /** initialize agent and food lists **/
  {                              /** as well as removed lists **/
    dead_list[i] = -1;
    pop_list[i] = 0;
  }
for (i = 0; i < MAX_FOOD; i++)
  {
    removed_food[i] = -1;
    food_list[i] = 0;
  }


  /***** This section will be removed eventually, it initializes the
         experiments I've been running.  Shortly, this will be handled
         entirely through the interface
                                  *****/
zero_section(0, 0, MAP_SIZE, MAP_SIZE);
build_border(0, 0, MAP_SIZE, MAP_SIZE);
//  print_section(0, 0, MAP_SIZE, MAP_SIZE);
populate_section(1000.0, 4, 0, 0, MAP_SIZE, MAP_SIZE, 1);
populate_section(1000.0, 1, 0, 0, MAP_SIZE, MAP_SIZE, 1);
//print_section(0, 0, MAP_SIZE, MAP_SIZE);

start_interface();

/***** Comments from here down, old code--keeping for learning purposes **/

/* while(1)
  {
    getchar();
    for(i = 0; i < beings; i++)
      {
*/      /*if(pop_list[i] != 0)
        {
          printf("%d %d ", pop_list[i]->x_loc, pop_list[i]->y_loc);
          for(j=0; j < STRING_SIZE; j++)
            {
              printf("%d", pop_list[i]->chromosome[j]);
              if((j + 1)%4 == 0)
              printf(" ");
              }

          printf("\n");
      */
          //printf("%d %d\n", pop_list[i]->x_loc, pop_list[i]->y_loc);
/*      if(pop_list[i] != 0)
        {
          if(update(i) != 0)
            {
              printf("Problem in update\n");
            }
        }
      }

    for(i = 0; i < beings; i++)
      {
        if(pop_list[i] != 0)
          {
            decision = evaluate(i);
*/          //              printf("decision = %d\tmate = %d\tenergy = %d\n", deci
            //                      pop_list[i]->mate, pop_list[i]->energy);
/*          perform_action(decision%5, i);
          }
      }
    print_section(0, 0, MAP_SIZE, MAP_SIZE);
*/  //      printf("x=\ty=\tage=\tenergy=\tmate=\n");
```

```
  /*  for(i = 0; i < beings; i++)
        {
          if(pop_list[i] != 0)
            {
              printf("%d\t%d\t%d\t%d\t%d\n",
             pop_list[i]->x_loc,
             pop_list[i]->y_loc,
             pop_list[i]->age,
             pop_list[i]->energy,
             pop_list[i]->mate
             );

            }
        }

      for(i = 0; i < beings; i++)
        {
          if(pop_list[i] != 0)
            printf("%d is alive\n", i);
        }
      printf("dead = %d\n", dead_index);

  */  /*  for(i = 0; i < beings; i++)
        {
          cur_x_loc = pop_list[i]->x_loc;
          cur_y_loc = pop_list[i]->y_loc;
          cur_loc = pop_list[i]->cur_loc;
          for_loc = pop_list[i]->for_loc;
          facing = pop_list[i]->facing;

          printf("x = %d\ty = %d\tloc = %d\tfor_loc = %d\t fac = %d\n",
                 cur_x_loc, cur_y_loc, cur_loc, for_loc, facing);
        }


        } */
}
```

```
#include<stdio.h>
#define STRING_SIZE 48
#define MAP_SIZE 70
#define MAX_POP 10000
#define MAX_FOOD 10000
struct Individual {
  //int string_size;
  //int POP_size;
  int ma;
  int pa;
  int mate;
  int offspring_chromosome[STRING_SIZE];
  int split_loc;
  int age;
  int facing;
  int x_loc;
  int y_loc;
  int for_loc;
  int cur_loc;
  int energy;
  int offspring;
  int moved;
  int chromosome[STRING_SIZE];
};

struct Food_Source{
  int moved;
  int amount;
  int x_loc;
  int y_loc;
};

int world_map[MAP_SIZE + 1][MAP_SIZE + 1];
struct Individual *pop_list[MAX_POP];
struct Food_Source *food_list[MAX_FOOD];

int beings;
int food;
int dead_list[MAX_POP];
int dead_index;
int removed_food[MAX_POP];
int removed_index;



double drand48();

/****  Here are the environment variables *****/

int food_energy_value;
int max_food_growth;
float food_growth_factor;
int create_dat;
int max_food;
int dead_food;
int max_age;
FILE *fp;
```

```
#include "enviro_make.h"


/*********************************************************************
* Name: evaluation.h
*
* Purpose: This is the decision making module.  Every generation,
* each being does an evaluation and returns a decision based on
* whatever algorithm its using
*
* Functions:
* int evaluate(int)
*
*********************************************************************/


/*********************************************************************
* function: evaluate(int)
*
* purpose: the decision making function.  This is just an example of
* an evaluation function.  What's important is that any evaluation
* that takes the agent and returns it's decision can be substituted
* in here.
*
* inputs: being_number
*
* returns: decision (integer)
*
* side affects: none
*
*********************************************************************/

int evaluate(int being_number)
{
  int cur_loc;
  int cur_x_loc;
  int cur_y_loc;
  int for_loc;
  int decision;
  int facing = pop_list[being_number]->facing;


  cur_x_loc = pop_list[being_number] -> x_loc;
  cur_y_loc = pop_list[being_number] -> y_loc;
  cur_loc = world_map[cur_x_loc][cur_y_loc];
  pop_list[being_number] -> cur_loc = cur_loc;

  switch (facing)
    {
    case 8:
      if(cur_y_loc - 1 < 0)  /*  BIG OL' HACK--GET RID OF  */
        for_loc = 3;
      else
        for_loc = world_map[cur_x_loc][cur_y_loc - 1];
      break;
    case 2:
      if(cur_y_loc + 1 > MAP_SIZE)  /*  BIG OL' HACK--GET RID OF  */
        for_loc = 3;
      else
        for_loc = world_map[cur_x_loc][cur_y_loc + 1];
      break;
    case 4:
      if(cur_x_loc - 1 < 0)  /*  BIG OL' HACK--GET RID OF  */
        for_loc = 3;
      else
        for_loc = world_map[cur_x_loc - 1][cur_y_loc];
      break;
    case 6:
      if(cur_x_loc + 1 > MAP_SIZE)  /*  BIG OL' HACK--GET RID OF  */
        for_loc = 3;
      else
        for_loc = world_map[cur_x_loc + 1][cur_y_loc];
      break;
    default:
      printf("something\'s wrong\n");
    }
pop_list[being_number]->for_loc = for_loc;

//printf("x = %d\ty = %d\tloc = %d\tfor_loc = %d\t fac = %d\n",
//   cur_x_loc, cur_y_loc, cur_loc, for_loc, facing);



if (cur_loc == 4)
  {
    if (for_loc == 0)
      {
        decision = get_decimal(0, 4, being_number);
      }
    else if (for_loc == 1)
      {
        decision = get_decimal(4, 4, being_number);

      }
    else if (for_loc == 2)
      {
        decision = get_decimal(8, 4, being_number);

      }
    else if (for_loc == 3)
      {
        decision = get_decimal(12, 4, being_number);

      }
    else if (for_loc == 4)
      {
        decision = get_decimal(16, 4, being_number);

      }
    else if (for_loc == 5)
      {
        decision = get_decimal(20, 4, being_number);

      }
    else
      {
        printf("something\'s wrong\n");
      }
  }
  else if (cur_loc == 5)
  {
    if (for_loc == 0)
      {
        decision = get_decimal(24, 4, being_number);

      }
    else if (for_loc == 1)
      {
        decision = get_decimal(28, 4, being_number);

      }
    else if (for_loc == 2)
```

```
        {
          decision = get_decimal(32, 4, being_number);

        }
      else if (for_loc == 3)
        {
          decision = get_decimal(36, 4, being_number);
        }
      else if (for_loc == 4)
        {
          decision = get_decimal(40, 4, being_number);

        }
      else if (for_loc == 5)
        {
          decision = get_decimal(44, 4, being_number);

        }
      else
        {
          printf("something\'s wrong\n");
        }
    }
  //printf("unmodified decision = %d\n", decision);
  return (decision);

}
```

```
#include "utilities.h"
#include "enviro_make.h"
#include <ctype.h>
/*********************************************************************
* Name: utilties.c
* Purpose: contains utilities that don't directly correspond to
* any particular stage (evaluation, create, etc).
*
*
*********************************************************************/









/*********************************************************************
* function: get_decimal(int, int, int)
* purpose: this function computes a decimal from genetic material
* (binary) from a creature number.
* inputs: start_where - where to begin the conversion, how_many - how
* many digits to read, being_number - whose material to read
* returns: the integer equivalen to the read string
* side affects: none
*
*********************************************************************/
int get_decimal(int start_where, int how_many, int being_number)
{
  int x, sum = 0;
  int decimal;

  for(x = 0; x < how_many;  x++)
    {
      /*printf("chrom = %d\t ", pop_list[being_number]
             ->chromosome[start_where + how_many - x - 1]);

      printf("sum = %d\n",sum);
      */
      sum = sum +
        pop_list[being_number] ->
        chromosome[start_where + how_many - x - 1] * my_pow(2,x);

    }
    decimal = sum; //pow(sum, 2);
  /*printf("decimal = %d\n", decimal);
   */
  return(decimal);
}

 /*********************************************************************
* function: whos_here(int, int, int)
* purpose: given an x and y location, which creature is in that spot
* inputs: the x and y location to be looked for, and the number of
* beings that exist.
* returns: the being number in that location, or -1 on a failed search
* side affects: none
*
*********************************************************************/

int whos_here(int x_loc, int y_loc, int beings)
{
  int i;
  //printf("x = %d\ty = %d\ti = %d\n",x_loc, y_loc, beings);

  for(i = 0; i < beings; i++)
    {

      //printf("x = %d\ty = %d\ti = %d\n",x_loc, y_loc, beings);
      if(pop_list[i] != 0)
        {
          if(pop_list[i]->x_loc == x_loc)
            if (pop_list[i]->y_loc == y_loc)
              return(i);
        }
    }

  return(-1);
}
/*********************************************************************
* function: which_food(int, int, int)
* purpose: identical to whos_here, but for food
* inputs:
* returns:
* side affects:
*
*********************************************************************/

int which_food(int x_loc, int y_loc, int food_no)
{
  int i;
  //printf("x = %d\ty = %d\ti = %d\n",x_loc, y_loc, beings);

  for(i = 0; i < food_no; i++)
    {
      //printf("x = %d\ty = %d\ti = %d\n",x_loc, y_loc, beings);
      if(food_list[i] != 0)
        {
          if(food_list[i]->x_loc == x_loc)
            if (food_list[i]->y_loc == y_loc)
              return(i);
        }
    }
  return(-1);
}

/*********************************************************************
* function: print_section(int, int, int, int)
* purpose: prints a section of the map (in ascii) to text interface
* inputs: the upper left corner x and y, and the lower right corner
* x and y
* returns: none
* side affects: prints out the contents of the map for all x y
* locations between the corners (given by inputs)
*
*********************************************************************/

void print_section(int up_x, int up_y, int low_x, int low_y)
{
  int x, y;
  printf("%d %d %d %d\n", up_x, up_y, low_x, low_y);
  for (y = up_y; y <= low_y; y++)
    {
      for (x = up_x; x <= low_x; x++)
        {
          printf("%d ", world_map[x][y]);
        }
      printf("\n");
    }
}
```

```
/*********************************************************************
* function: zero_section(int, int. int, int)
* purpose: places a zero in all map locations in the given square
* inputs: the upper left and lower right corners of the square
* returns: none
* side affects: changes global variable world_map.
*
*********************************************************************/

void zero_section(int up_x, int up_y, int low_x, int low_y)
{
  int x, y;

  for (x = up_x; x <= low_x; x++)
    {
      for (y = up_y; y <= low_y; y++)
        {
          world_map[x][y] = 0;
        }
    }
}
/*********************************************************************
* function: build_border(int, int, int, int)
* purpose: builds a border around a given square
* inputs: the upper left and lower right x and y coordinates of the
* square
* returns: none
* side affects: places a border around the edge of the designated
* square in the variable "world_map"
*
*********************************************************************/

void build_border(int up_x, int up_y, int low_x, int low_y)
{
  int x, y;

  x = up_x;
  for (y = up_y; y <= low_y; y++)
    {
      world_map[x][y] = 3;
      //printf("%d %d\n", x, y);
    }
  x = low_x;
  for (y = up_y; y <= low_y; y++)
    {
      world_map[x][y] = 3;
      //printf("%d %d\n", x, y);
    }
  y = up_y;
  for (x = up_x; x <= low_x; x++)
    {
      world_map[x][y] = 3;
      //printf("%d %d\n", x, y);
    }
  y = low_y;
  for (x = up_x; x <= low_x; x++)
    {
      world_map[x][y] = 3;
      //printf("%d %d\n", x, y);
    }
}

/*********************************************************************
* function: find_free_space(int, int, int, int *, int *)
* purpose: finds a free space around a given agent (no diagonals)
* used to find open space for offspring
* inputs: the being_number, or the location
* returns: changes the values of two passed variables x, y to indicate
* the coordinates of an ajacent free space.
* side affects: changes two pass-by-value variables.
*
*********************************************************************/

void find_free_space(int being_number, int x_loc, int y_loc, int *x, int *y)
{
  int temp_x, temp_y;


  if(being_number >= 0 ) /* Passing a being number causes the function to
                            be called again using the lookup x and y */
    {
      temp_x = pop_list[being_number]->x_loc;
      temp_y = pop_list[being_number]->y_loc;
      find_free_space(-1, temp_x, temp_y, x, y);
      return;
    }
  if((world_map[x_loc][y_loc +1] == 0) || (world_map[x_loc][y_loc +1] == 1))
    {
      *x = x_loc;
      *y = y_loc +1;
    }
  else if((world_map[x_loc][y_loc -1] == 0)
          || (world_map[x_loc][y_loc -1] == 1))
    {
      *x = x_loc;
      *y = y_loc -1;
    }
  else if((world_map[x_loc +1][y_loc] == 0)
          || (world_map[x_loc +1][y_loc] == 1))
    {
      *x = x_loc +1;
      *y = y_loc;
    }
  else if((world_map[x_loc -1][y_loc] == 0)
     || (world_map[x_loc -1][y_loc] == 1))
    {
      *x = x_loc -1;
      *y = y_loc;
    }
  else
    {
      *x = -1;
      *y = -1;
    }
}
/*********************************************************************
* function: my_pow(int, int)
* purpose: somehow, the math library power function returns crap
* sometimes, so this is a simple routine to compute the function
* reliably (my fault, most likely)
* inputs: base and the exponent
* returns: base raised to the exponent power
* side affects: none
*
*********************************************************************/

int my_pow(int base, int exponent)
{
  int sum = 1;

  while(exponent > 0)
    {
```

```
      sum = base * sum;
      exponent --;
    }
  return(sum);
}
/*********************************************************************
* function: isnumber(char *)
* purpose: tests a character string for numberhood
* inputs: character srting
* returns: 1 if string is a number (all characters are digits)
* 0 otherwise.
* side affects: none
*
*********************************************************************/

int isnumber(char *word)  /* Returns 1 if word is a number, 0 otherwise */
{
  int x;


  for(x = 0; x < strlen(word); x++)
    {
      if(isdigit(word[x]))
        {
          x++;
        }
      else
        {
          return(0);
        }
    }
  return(1);
}
```

```
#include "enviro_make.h"
#include "actions.h"

/*********************************************************************
* Name: actions.c
*
* Purpose: actions holds the functions that deal with performing
* the action decided upon by the evaluation section.  It's really
* not much to talk about, mostly a very large case statement to
* determine what happens given an agents decision to perform an
* action.
*
* Functions:
* void perform_action(int, int);
* int mate_with(int, int, int);
* void eat(int);
*
*********************************************************************/

/*********************************************************************
* function: perform_action(int, int)
*
* purpose: calculates the result of an action using a large if-then-
* else tree.
*
* inputs: being_number and decision
*
* returns: nothing
*
* side affects: Numerous possible.  Changes the map depending upon
* agent movement.
*
* NOTE: This really needs to be polished, there must be a better way
* to determine outcomes!
*
*********************************************************************/

void perform_action(int decision, int being_number)
{
  int x_loc = pop_list[being_number]->x_loc;
  int y_loc = pop_list[being_number]->y_loc;
  int facing = pop_list[being_number]->facing;
  int cur_loc = pop_list[being_number]->cur_loc;
  int for_loc = pop_list[being_number]->for_loc;
  int x, y;

  if(decision == 0)
    {
      if(world_map[x_loc][y_loc] == 1)
        {
          pop_list[being_number]->energy +=15;
          world_map[x_loc][y_loc] = 4;
        }
    }
  if(decision == 1)  /*** DECIDE TO MOVE ***/
    {
      if(for_loc == 0) /*** empty ahead ***/
        {
          pop_list[being_number]->moved = 1;

          if(cur_loc == 5)      /*** Change map to reflect move ***/
            world_map[x_loc][y_loc] = 1;
          else
            world_map[x_loc][y_loc] = 0;

          if(facing == 8)
            {

              world_map[x_loc][y_loc - 1] = 4;
              pop_list[being_number]->y_loc = y_loc - 1;
            }
          else if(facing == 2)
            {
              world_map[x_loc][y_loc + 1] = 4;
              pop_list[being_number]->y_loc = y_loc + 1;
            }
          else if(facing == 6)
            {
              world_map[x_loc + 1][y_loc] = 4;
              pop_list[being_number]->x_loc = x_loc + 1;
            }
          else if(facing == 4)
            {
              world_map[x_loc - 1][y_loc] = 4;
              pop_list[being_number]->x_loc = x_loc - 1;
            }
        }
      else if(for_loc == 1)  /*** Food Ahead ***/
        {
          pop_list[being_number]->moved = 1;

          if(cur_loc == 5)      /*** Change map to reflect move ***/
            world_map[x_loc][y_loc] = 1;
          else
            world_map[x_loc][y_loc] = 0;

          if(facing == 8)
            {
              world_map[x_loc][y_loc - 1] = 5;
              pop_list[being_number]->y_loc = y_loc - 1;
            }
          else if(facing == 2)
            {
              world_map[x_loc][y_loc + 1] = 5;
              pop_list[being_number]->y_loc = y_loc + 1;
            }
          else if(facing == 6)
            {
              world_map[x_loc + 1][y_loc] = 5;
              pop_list[being_number]->x_loc = x_loc + 1;
            }
          else if(facing == 4)
            {
              world_map[x_loc - 1][y_loc] = 5;
              pop_list[being_number]->x_loc = x_loc - 1;
            }
        }
    }
  if(decision == 2)  /*** DECIDE TO TURN LEFT ***/
    {
      if(facing == 8)
        {
          facing = 4;
          pop_list[being_number]->facing = 4;
        }
      else if(facing == 2)
        {
          facing = 6;
          pop_list[being_number]->facing = 6;
        }
      else if(facing == 6)
        {
```

```
        facing = 8;
        pop_list[being_number]->facing = 8;
          }
      else if(facing == 4)
        {
          facing = 2;
          pop_list[being_number]->facing = 2;
        }
    }

  if(decision == 3)  /*** DECIDE TO TURN RIGHT ***/
    {
      if(facing == 8)
        {
          facing = 6;
          pop_list[being_number]->facing = 6;
        }
      else if(facing == 2)
        {
          facing = 4;
          pop_list[being_number]->facing = 4;
        }
      else if(facing == 6)
        {
          facing = 2;
          pop_list[being_number]->facing = 2;
        }
      else if(facing == 4)
        {
          facing = 8;
          pop_list[being_number]->facing = 8;
        }
    }


  if(decision == 4)  /*** DECIDE TO MATE ***/
    {
      if(facing == 8 &&
         (for_loc == 4 || for_loc == 5))
        {
          mate_with(x_loc, y_loc - 1, being_number);
          find_free_space(-1, x_loc, y_loc - 1, &x, &y);
          //printf("baby at %d %d\n", x, y);

        }
      else if(facing == 2 &&
              (for_loc== 4
               || for_loc == 5))
        {
          mate_with(x_loc, y_loc + 1, being_number);
          find_free_space(-1, x_loc, y_loc + 1, &x, &y);
          //printf("baby at %d %d\n", x, y);
        }
      else if(facing == 6 &&
              (for_loc == 4
               || for_loc == 5))
        {
          mate_with(x_loc + 1, y_loc, being_number);
          find_free_space(-1, x_loc + 1, y_loc, &x, &y);
          //printf("baby at %d %d\n", x, y);
        }
      else if(facing == 4 &&
              (for_loc == 4
               || for_loc == 5))
        {
          mate_with(x_loc - 1, y_loc, being_number);
          find_free_space(-1, x_loc - 1, y_loc, &x, &y);
          //printf("baby at %d %d\n", x, y);
        }
    }

  if(decision == 5)
    {
      if(cur_loc == 5)
        {
          eat(being_number);
        }

    }
}

/*********************************************************************
* function: mate_with(int, int, int)
*
* purpose: is called when mating occurs (as determined by
* perform_action.  It does the entire genetic process (crossover,
* mutation, etc) and stores the resulting genetic code in the
* "mother" agent structure.
* inputs: the x and y loc of the "mother", and the "father's" being
* number.
*
* returns: returns 0 upon successful mating
*
* side affects: alters the structure of the mother agent (to indicate
* genetic code of offspring and flag the agent as "pregnant").
* also alters the being list and the dead list (potentially) to reflect
* creation of new creature.
*
*********************************************************************/

int mate_with(int x_loc, int y_loc, int father)
{
  int being_number;
  int x, y;
  int being1, being2;
  int insert_point;
  int loc1, loc2;
  int chrom1[STRING_SIZE/2], chrom2[STRING_SIZE/2];


  being_number = whos_here(x_loc, y_loc, beings);
  //printf("SEX!! between %d & %d\n", being_number, father);
  if (being_number == -1)
    return(-1);
  if (pop_list[being_number] -> mate != -1)
    {
      //printf("already mated\n");
      return(0);
    }
  pop_list[being_number]->mate = father;


  if(drand48() > .5)
    {
      being1 = being_number;
      being2 = father;
    }
  else
    {
      being1 = father;
      being2 = being_number;
    }
```

```
  insert_point = rint(drand48() * STRING_SIZE);
  //  printf("insert point = %d\n", insert_point);
  for(y = 0; y < insert_point; y++)
    {
      pop_list[being_number]->offspring_chromosome[y] =
        pop_list[being1]->chromosome[y];
    }
  for(y = insert_point; y <= STRING_SIZE; y++)
    {
      pop_list[being_number]->offspring_chromosome[y] =
        pop_list[being2]->chromosome[y];
    }
  return(0);
}

/*********************************************************************
* function: eat(int)
*
* purpose: calculate the effects of eating
*
* inputs: location of being eating (also location of food)
*
* returns: none
*
* side affects: affects map to reflect decrease in amount of food,
* affects structure of eating agent to
* reflect energy increase etc.
*
*********************************************************************/

void eat(int being_number)
{
  int x_loc, y_loc;
  int food_number;

  x_loc = pop_list[being_number]->x_loc;
  y_loc = pop_list[being_number]->y_loc;

  food_number = which_food(x_loc, y_loc, food);

  if (food_number == -1)
    {
      printf("food error at %d %d\n", x_loc, y_loc);
      return;
    }
  //printf("food number = %d %d %d\n", food_number, food, removed_index);
        //food_list[food_number]->amount);
  food_list[food_number]->amount = food_list[food_number]->amount - 1;

  pop_list[being_number] -> energy = pop_list[being_number] -> energy + food_energy_value;
  if(food_list[food_number]->amount <= 0)
    {
      //printf("removing food\n");
      remove_food(food_number);
    }
}
```

```
#include "enviro_make.h"
#include "create.h"

/*********************************************************************
* Name: create.c
*
* Purpose: This is used in the initialization phase and throughout
* the simulation to create and destroy the agents, food supplies, and
* to update (which is all the book-keeping) the environment and the
* agents.
*
* Functions:
*  void populate_section(double, int, int, int, int, int, int);
* void kill(int);
* int update(int);
* struct Food_Source *make_food(int, int, int);
* struct Individual *make_life(int, int, int, int, int);
* void remove_food(int);
*
*
*********************************************************************/

/*********************************************************************
* function: populate_section(double, int, int, int, int, int, int)
*
* purpose: This is the all purpose populator/placer of items on the
* map.  Anytime anything needs to be added to the world, it should
* be (and be able to be) done through this function.
*
* inputs: number - how many to place, code - the numeric code of what
* to place, the coordinates of the upper-right and lower-left corners
* of the placement square.  new indicates if the agent (if an agent
* is being placed) is new or offspring from other agents
*
* returns: nothing
*
* side affects: writes to the world_map global variable
*
*********************************************************************/

void populate_section(double number, int code, int up_x,
                      int up_y, int low_x, int low_y, int new)
{
  double placement_odds;
V  int x = up_x, y = up_y, z = 0;
  int x_diff, y_diff;
  int total_free = 0;
  int i, j;
  int food_number;

  for(i = up_x; i <= low_x; i++)
    {
      for(j = up_y; j <= low_y; j++)
        {
          //printf("%d ", world_map[i][j]);

          if (code == 4 && (world_map[i][j] == 1 || world_map[i][j] == 0))
            {
              total_free++;
            }
          else if(code == 1 && (world_map[i][j] != 3))
            {
              total_free++; /*** The number of candidate spots **/
            }
        }
    }
//printf("tot_free = %d\n", total_free);
x_diff = low_x - up_x;
y_diff = low_y - up_y;

placement_odds = number/(total_free);
//printf("%f %f\n", number, placement_odds);
while(z < number)   /****** PLACEMENT LOOP ******/
  {

    if ((drand48() < placement_odds) && (world_map[x][y] != 3))
      {
          //printf("b_no %d\tx %d\ty %d\tmap %d\n", beings, x, y,
                      //world_map[x][y]);


        if (code == 4 && world_map[x][y] == 1)
          {
            world_map[x][y] = 5;
          }
        else if(code == 1)
          {

            if(world_map[x][y] == 1 || world_map[x][y] == 5)
              {
                food_number = which_food(x, y, food);
                if(food_list[food_number] == 0)
                  {
                    printf("food error %d %d %d\n", x, y, world_map[x][y]);
                    show_food();
                  }
                else
                  {
                    food_list[food_number]->amount = food_list[food_number]->amoun
                  }
              }

            if(world_map[x][y] == 4 || world_map[x][y] == 0)
              {

                if (removed_index != -1)
                  {
                    food_list[removed_food[removed_index]] =
                      make_food(x, y, 1);
                    removed_food[removed_index] = -1;
                    removed_index --;
                  }
                else
                  {
                    food_list[food] = make_food(x, y, 1);
                    food++;
                  }
                if(world_map[x][y] == 4)
                  world_map[x][y] = 5;
                else
                  world_map[x][y] = 1;
              }
          }
        else
          {
            world_map[x][y] = code;
          }

        if (code == 4 && new == 1)
          {
            pop_list[beings] = make_life(1, 0, 0, x, y);
```

```
            beings = beings + 1;
          }
        z++;
      }


    //      else
    //printf("denied at %d %d\n",x, y);
    //printf("xdiff = %d\tydiff = %d\tx = %d\ty = %d\t\n",x_diff, y_diff,
//        x, y);
    if (y == y_diff)
      {
        x = (x+1)%(x_diff + 1);
      }
    y = (y+1)%(y_diff + 1);
  }
}


/*********************************************************************
* function: make_food(int, int, int)
*
* purpose: creates a new food source.  This is needed when food
* grows, or spreads between map sections.
*
* inputs: x and y locations, and amount of food in the new source
*
* returns: a food structure holding the information about the food
* source
*
* side affects: mallocs memory for the new food source.  Modifies
* agent structure.
*
*********************************************************************/

struct Food_Source *make_food(int x, int y, int amount)
{
  struct Food_Source *new_food;
  int i;

  new_food = (struct Food_Source *)malloc(sizeof(struct Food_Source));
  new_food->moved = 1;
  new_food->x_loc = x;
  new_food->y_loc = y;
  new_food->amount = amount;

  return new_food;
}


/*********************************************************************
* function: make_life(int, int, int, int, int)
*
* purpose: creates a new life, whether from nothing or from
* a previous generation
*
* inputs: new - indicates that the agent has no parents, ma & pa are
* the progenitors, x & y_loc is the location of the new agent
*
* returns: returns new agent structure with all genetic info, energy,
* location, age, etc.
*
* side affects: writes to agent structure, mallocs memory to create
* agent.
*
*********************************************************************/

struct Individual *make_life(int new, int ma, int pa, int x_loc, int y_loc)

{
  struct Individual *new_being;
  int i;

  new_being = (struct Individual *)malloc(sizeof(struct Individual));


  if(new == 1)
    {
      new_being->moved = 1;
      new_being->age = 0;
      new_being->ma = 0;
      new_being->pa = 0;
      new_being->mate = -1;
      new_being->energy = 100;
      new_being->x_loc = x_loc;
      new_being->y_loc = y_loc;
      new_being->facing = 8;
      for (i = 0; i < STRING_SIZE; i++)
        {
          if (drand48() < .5)
            new_being->chromosome[i] = 0;
          else
            new_being->chromosome[i] = 1;
        }
    }
  else
    {
      new_being->moved = 1;
      new_being->age = 0;
      new_being->ma = ma;
      new_being->pa = pa;
      new_being->mate = -1;
      new_being->energy = (pop_list[ma] -> energy) / 2;
      new_being->x_loc = x_loc;
      new_being->y_loc = y_loc;
      new_being->facing = 8;
      for (i = 0; i < STRING_SIZE; i++)
        {
          new_being->chromosome[i] = pop_list[ma]->offspring_chromosome[i];
        }
      pop_list[ma] -> energy = pop_list[ma] -> energy / 2;
    }
  return new_being;
}

int update(int being_number)
{
  int x, y;
  int x_loc = pop_list[being_number]->x_loc;
  int y_loc = pop_list[being_number]->y_loc;
  int mate = pop_list[being_number]->mate;

  pop_list[being_number]->age = pop_list[being_number]->age + 1;
  pop_list[being_number]->energy = pop_list[being_number]->energy - 1;
  pop_list[being_number]->moved = 0;

  /*  if(drand48() < pop_list[being_number]->age / 300.00)
    {
      printf("%d  died of old age at age %d\n", being_number, pop_list[being_number]
      kill(being_number);
      return(0);
      }*/
```

```
  if(pop_list[being_number]->energy < 0)
    {
      //printf("time to die!\n");
      kill(being_number);

      return(0);
    }

  if(mate >= 0)
    {

      pop_list[being_number] -> mate = -1;
      find_free_space(being_number, 0, 0, &x, &y);

      if(x < 0 || y < 0)
        {
          //printf("Child absorbed\n");
          return(0);
        }
      //printf("This one is pregnant x = %d   y = %d\n", x, y);
      if (dead_index != -1)
        {
          pop_list[dead_list[dead_index]] =
            make_life(0, being_number, mate, x, y);
          dead_list[dead_index] = -1;
          dead_index --;
        }
      else
        {
          pop_list[beings] = make_life(0, being_number, mate, x, y);
          beings++;
          //printf("beings %d\n",beings);
        }
      populate_section(1.0, 4, x, y, x, y, 0);
    }
  return(0);
}

int update_food(int food_number)
{
  int x_loc = food_list[food_number]->x_loc;
  int y_loc = food_list[food_number]->y_loc;
  int amount = food_list[food_number]->amount;

  food_list[food_number]->moved = 0;

  food_list[food_number]->amount = (amount + ((max_food - amount) * max_food_growth * drand4
8()));

  if((world_map[x_loc][y_loc+1] == 0 || world_map[x_loc][y_loc+1] == 4)
     &&drand48() < food_growth_factor)
    {
      populate_section(1, 1, x_loc, y_loc+1, x_loc, y_loc+1, 0);
    }
  if((world_map[x_loc][y_loc-1] == 0 || world_map[x_loc][y_loc-1] == 4)
     && drand48() < food_growth_factor)
    {
      populate_section(1, 1, x_loc, y_loc-1, x_loc, y_loc-1, 0);
    }
  if((world_map[x_loc+1][y_loc] == 0 || world_map[x_loc+1][y_loc] == 4)
     && drand48() < food_growth_factor)
    {
      populate_section(1, 1, x_loc+1, y_loc, x_loc+1, y_loc, 0);
    }
  if((world_map[x_loc-1][y_loc] == 0 || world_map[x_loc-1][y_loc] == 4)
     && drand48() < food_growth_factor)
    {
      populate_section(1, 1, x_loc-1, y_loc, x_loc-1, y_loc, 0);
    }

  return(0);
}


void kill(int being_number)
{
  int x_loc = pop_list[being_number]->x_loc;
  int y_loc = pop_list[being_number]->y_loc;
  char strng[20];

  if(create_dat)
  {
    sprintf(strng, "1 %d -1 -1\n", being_number);
    fputs(strng, fp);
  }
  free(pop_list[being_number]);
  pop_list[being_number] = 0;
  dead_index++;
  dead_list[dead_index] = being_number;

  if(world_map[x_loc][y_loc] == 5)
    {
      world_map[x_loc][y_loc] = 1;
    }
  else
    {
      world_map[x_loc][y_loc] = 0;
    }

  if(dead_food)
    {
      populate_section(1.0, 1, x_loc, y_loc, x_loc, y_loc, 0);
    }

}

void remove_food(int food_number)
{
  int x_loc = food_list[food_number]->x_loc;
  int y_loc = food_list[food_number]->y_loc;
  char strng[20];

  if(create_dat)
    {
      sprintf(strng, "2 %d -1 -1\n", food_number);
      fputs(strng, fp);
    }
  food_list[food_number] = 0;
  removed_index ++;
  removed_food[removed_index] = food_number;


  if(world_map[x_loc][y_loc] == 5)
    {
      world_map[x_loc][y_loc] = 4;
    }
  else if(world_map[x_loc][y_loc] == 1)
    {
      world_map[x_loc][y_loc] = 0;
    }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "utilities.h"
#include "enviro_make.h"

#define SEPARATORS " \t\n(),"



int start_interface()
{
  int command_no = 1;
  char buf[1024];
  char orig_buf[1024];
  char history[100][1024];
  char strng[30];
  int x;
  int command_results = 0;

  //char *word;
  //int num_1, num_2, num_3, num_4;


  fp = fopen("data.dat", "w");


  sprintf(strng, "0 0 %d %d\n", MAP_SIZE, MAP_SIZE);
  fputs(strng, fp);
  while(command_results != -1)
    {
      printf("%d >>> ", command_no);

      fgets(buf, 1024, stdin);

      if(!strcmp(buf, "!!\n"))
        {
          strcpy(buf, history[command_no - 1]);
          strcpy(history[command_no], history[command_no - 1]);
          printf("%s", buf);
        }
      else
        {
          strcpy(history[command_no], buf);
        }
      command_results = command(buf);
          if (command_results && command_results != -1)
            {
              usage(command_results);
            }
          command_no ++;
    }
}



int command(char *buf)
{

  char *word;
  int num_1, num_2, num_3, num_4;
  int x;

  word = (char *)strtok(buf, SEPARATORS);

  if(word == NULL)
    {
      return(6);
    }



if(!strcmp(word, "run"))
  {
    word = (char *)strtok(NULL, SEPARATORS);
    if(word == NULL)
      {
        return(1); /*** not enough arguments ***/
      }
    if(!isnumber(word))
      {
        return(1); /*** argument not a number ***/
      }
    else
      {
        num_1 = atoi(word);
        word = (char *)strtok(NULL, SEPARATORS);
        if(word != NULL)
          {
            return(1); /*** usage (too many arguments); ***/
          }
        else
          {
            printf("running %d\n", num_1);
            run(num_1);
            return(0);
          }
      }
  }

if(!strcmp(word, "step"))
  {
    word = (char *)strtok(NULL, SEPARATORS);
    if(word != NULL)
      {
        return(7); /*** too enough arguments ***/
      }
    run(1);
    print_section(0, 0, MAP_SIZE, MAP_SIZE);
    return(0);
  }


if(!strcmp(word, "animate"))
  {
    word = (char *)strtok(NULL, SEPARATORS);
    if(word == NULL)
      {
        return(8); /*** not enough arguments ***/
      }

    if(!isnumber(word))
      {
        return(1); /*** argument not a number ***/
      }
    else
      {
        num_1 = atoi(word);
        word = (char *)strtok(NULL, SEPARATORS);
        if(word != NULL)
          {
            return(1); /*** usage (too many arguments); ***/
```

```
          }
        else
          {
            for(x = 0; x < num_1; x++)
              {

                //system("clear");
                printf("%d %d\n", x, num_1);
                run(1);
                print_section(0, 0, MAP_SIZE, MAP_SIZE);
                show_organism(0, beings);

              }
          }
      }
  }


else if(!strcmp(word, "place"))
  {
    printf("place ok\n");
  }




else if(!strcmp(word, "place"))
  {
    printf("place ok\n");
  }



else if(!strcmp(word, "show"))
  {
    word = (char *)strtok(NULL, SEPARATORS);
    if(word == NULL)
      {
        return(4);
      }


    if(!strcmp(word, "map"))
      {
        word = (char *)strtok(NULL, SEPARATORS);

        if(word == NULL)
          {
            printf("showing map\n");
            print_section(0, 0, MAP_SIZE, MAP_SIZE);
            return(0);
          }
        return(3); /** show map usage error **/
      }

    if(!strcmp(word, "step"))
      {
        word = (char *)strtok(NULL, SEPARATORS);

        if(word == NULL)
          {
            printf("showing step\n");
            run(1);
            print_section(0, 0, MAP_SIZE, MAP_SIZE);
            show_organism(0, beings);
            return(0);
          }
        return(6); /** show step error**/
      }

    if(!strcmp(word, "food"))
      {
        word = (char *)strtok(NULL, SEPARATORS);

        if(word == NULL)
          {
            printf("showing food\n");
            show_food();
            return(0);
          }
        return(3); /** show map usage error **/
      }

    else if (!strcmp(word, "all"))
      {
        word = (char *)strtok(NULL, SEPARATORS);
        if(word == NULL)
          {
            print_section(0, 0, MAP_SIZE, MAP_SIZE);
            show_organism(0, beings);
            return(0);
          }
        else
          {
            return(5);  /** show all usage error **/
          }
      }

    else if(!isnumber(word))
      {
        return(4); /** show general usage error **/
      }

    else
      {
        num_1 = atoi(word);
        word = (char *)strtok(NULL, SEPARATORS);
        if (word == NULL)
          {
            show_organism(num_1, num_1);
            return(0);
          }
        else if(!isnumber(word))
          {
            return(4); /** show general usage error **/
          }
        else
          {
            num_2 = atoi(word);
            word = (char *)strtok(NULL, SEPARATORS);
            if (word != NULL)
              {
                return(4);  /** show general usage error **/
              }
            else
              {
                show_organism(num_1, num_2);
                return(0);
              }
          }
      }
```

```
    }


  else if(!strcmp(word, "quit"))
    {
      return(-1);
    }

  else
    {
      printf("%s\n", word);

    }
}


int run(int iterations)
{
  int orig_beings, orig_food;
  int x, i;
  int decision;
  char strng[20];
  FILE *junk;



  for(x = 0; x < iterations; x++)
    {
      orig_beings = beings;
      orig_food = food;

      map_food(orig_food);


      for(i = 0; i < orig_beings; i++)
        {
          if(pop_list[i] != 0 && pop_list[i]->age != 0)
            {
              decision = evaluate(i);
              //              printf("decision = %d\tmate = %d\tenergy = %d\n", decision%6,
              //     pop_list[i]->mate, pop_list[i]->energy);
              perform_action(decision%6, i);
            }
        }
      map_beings(orig_beings);

      /*      for(i = 0; i < orig_beings; i++)
        {
          if(pop_list[i] != 0)
            {
              if (pop_list[i]->moved == 1)
                {
                  sprintf(strng, "1 %d %d %d\n", i, pop_list[i] -> x_loc, pop_list[i] -> y_l
oc);

                  fputs(strng, fp);

                }
            }
          else
            {
              sprintf(strng, "1 %d -1 -1\n", i);
              fputs(strng, fp);
            }
          fflush(fp);
        }
      */
      //map_food(orig_food);
      //map_beings(orig_beings);

      for(i = 0; i < orig_beings; i++)
        {
          if(pop_list[i] != 0)
            {
              if(update(i) != 0)
                {
                  printf("Problem in update\n");
                }
            }

        }

      for(i = 0; i < orig_food; i++)
        {
          if(food_list[i] != 0)
            {
              if(update_food(i) != 0)
                {
                  printf("Problem in food\n");
                }
            }
        }

      junk = fopen("flag", "w");
      fclose(junk);
    }
}


int show_organism(int start, int end)
{
  int i;

  printf("#\tx\ty\tage\tenergy\tmate\tmove\n");
  for(i = start; i <= end; i++)
    {
      if(pop_list[i] != 0)
        {
          printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n",
                 i,
                 pop_list[i]->x_loc,
                 pop_list[i]->y_loc,
                 pop_list[i]->age,
                 pop_list[i]->energy,
                 pop_list[i]->mate,
                 pop_list[i]->moved
                 );

        }
      /*else
        {
          printf("%d\t\tempty\n",i);
          }*/
    }
}

int show_food()
{
  int i;

  printf("#\tx\ty\tenergy\n");
  for(i = 0; i <= food; i++)
    {
```

```
      if(food_list[i] != 0)
        {
          printf("%d\t%d\t%d\t%d\t%d\t\n",
                 i,
                 food_list[i]->x_loc,
                 food_list[i]->y_loc,
                 food_list[i]->amount,
                 food_list[i]->moved
                 );

        }
      else
        {
          printf("%d  empty\n", i);
        }
    }
}

int map_beings(int orig_beings)
{
  int i;
  char strng[20];
  for(i = 0; i < orig_beings; i++)
    {
      if(pop_list[i] != 0)
        {
          if (pop_list[i]->moved == 1)
            {
              sprintf(strng, "1 %d %d %d\n", i, pop_list[i] -> x_loc, pop_list[i] -> y_loc);
              fputs(strng, fp);

            }
        }
      fflush(fp);
    }
}

int map_food(int orig_food)
{
  int i;
  char strng[20];

  for(i = 0; i < orig_food; i++)
    {
      if(food_list[i] != 0)
        {
          if (food_list[i]->moved == 1)
            {
              sprintf(strng, "2 %d %d %d\n", i, food_list[i] -> x_loc, food_list[i] -> y_loc
);
              fputs(strng, fp);
            }
        }
      fflush(fp);
    }
}
```

```
#include "usage.h"

void usage(int error_no)
{
  switch (error_no)
    {
    case 1:
      printf("Usage:  run <turns to run>.  Example: run 5\n");
      break;
    case 2:
      printf("something else is wrong\n");
      break;
    case 3:
      printf("Usage: show map");
      break;
    default:
      printf("you need to add this no:  %d\n", error_no);
    }

}
```

```
#!/usr/local/X11R6.3/bin/wish

set f .25
set file [open "data.dat" r]
while {1} {
    set i [file exists flag]
     while {[file exists flag] == 1} {
        puts "its there"

        while {[gets $file line] >= 0} {
            set arg_1 [lindex $line 0]
            set arg_2 [lindex $line 1]
            set arg_3 [lindex $line 2]
            set arg_4 [lindex $line 3]
            if {$arg_1 == 0} {
                canvas .c -background green2 -width [expr $arg_3 * $f]c -height [expr $arg_
4 * $f]c
                pack .c

            } elseif {$arg_1 == 1} {
                if {[array size pop_list] > $arg_2} {
                    .c delete [ set pop_list($arg_2)]
                    if {$arg_3 != -1} {
                        set new [.c create oval [expr $arg_3 * $f]c \
                                [expr $arg_4 * $f]c [expr ($arg_3 + 1) * $f]c \
                                [expr ($arg_4 + 1) * $f]c]
                        set pop_list($arg_2) $new
                    }
                } else {
                    set new [.c create oval [expr $arg_3 * $f]c \
                            [expr $arg_4 * $f]c [expr ($arg_3 + 1) * $f]c \
                            [expr ($arg_4 + 1) * $f]c]
                    set pop_list($arg_2) $new
                }
            } elseif {$arg_1 == 2} {

                if {[array size food_list] > $arg_2} {
                    .c delete [ set food_list($arg_2)]
                    if {$arg_3 != -1} {
                        set new [.c create rectangle [expr $arg_3 * $f]c \
                                [expr $arg_4 * $f]c [expr ($arg_3 + 1) * $f]c \
                                [expr ($arg_4 + 1) * $f]c -outline white]
                        set food_list($arg_2) $new
                    }
                } else {
                    set new [.c create rectangle  [expr $arg_3 * $f]c \
                            [expr $arg_4 * $f]c [expr ($arg_3 + 1) * $f]c \
                            [expr ($arg_4 + 1) * $f]c -outline white]
                    set food_list($arg_2) $new
                }

            }
            update

        }
        exec rm flag
    }
 }
```