College of Saint Benedict and Saint John's University
DigitalCommons@CSB/SJU

Honors Theses, 1963-2015 Honors Program

1997

Simulation of Living Processes Utilizing Concurrency and Object-Oriented Programming

Aaron Ziegler
College of Saint Benedict/Saint John's University

Follow this and additional works at: https://digitalcommons.csbsju.edu/honors_theses

Part of the Computer Sciences Commons

Recommended Citation
Ziegler, Aaron, "Simulation of Living Processes Utilizing Concurrency and Object-Oriented Programming" (1997). *Honors Theses, 1963-2015*. 637.
https://digitalcommons.csbsju.edu/honors_theses/637

Available by permission of the author. Reproduction or retransmission of this material in any form is prohibited without expressed written permission of the author.

Simulation of Living Processes Utilizing Concurrency and Object-Oriented Programming

A THESIS
The Honors Program
College of St. Benedict/St. John's University

In Partial Fulfillment
of the Requirements for "Departmental Distinction"
and the Degree Bachelor of Arts
in the Department of Computer Science

Advisor
Dr. J. Andrew Holey

by
Aaron L. Ziegler
May 3, 1997

PROJECT TITLE: Simulation of Living Processes Utilizing Concurrency and Object-Oriented Programming.

Approved by:

J. Andrew Holey — Associate Professor, Computer Science

James Schnepf — Assistant Professor, Computer Science

John Miller — Instructor, Computer Science

Noreen Herzfeld — Chair, Computer Science

Margaret Cook — Director, Honors Thesis Program

Anthony Cunningham — Director, Honors Program

Simulation of Living Processes Utilizing Concurrency and Object-Oriented Programming
A Thesis by Aaron Ziegler

Section 1: Introduction....................................Page 3

Section 2: The Definition of Life..........................Page 6

Section 3: The History of Artificial Life................Page 8
--Von Neumann
--Horton Conway's "Game of Life"
--Viruses
--Thomas Ray's 'Tierra'

Section 4: Object-Oriented Programming....................Page 13
--Objects, methods, and messages
--Why A-Life forms should be treated as objects
--Java

Section 5: Concurrency: A Vital Element..................Page 21

Section 6: My Own Attempt at A-Life.......................Page 23
--Description of my a-life
--A few scenarios, and my observations
--Improvements that could be made

Section 7: My Conclusions..................................Page 34

Bibliography...Page 35

Appendix A: Von Neumann's Self-Replicating Automaton.....Page 36

Appendix B: A Few 'Game of Life' Structures.............Page 37

Appendix C: Java Code.....................................Page 38
--LifeGrid class
--gridSquare class

Appendix D: HTML Code.....................................Page 78
--Main page
--Scenerio pages

Introduction

Legend and literature is filled with stories of people who try to create life. In the stories of Prague's famous Rabbi Löw, with God's help, lifeless clay is brought to life to create a Golem capable of serving and defending Prague's oppressed Jewish population. In Mary Shelley's popular Frankenstein, a scientist, obsessed with learning the secrets of life and death, animates a horrifying monster from the remnants of dead corpses. With the advent of science fiction, the world became flooded with stories of artificial people, some good, existing alongside humankind, others bad, running amok and turning on their creators. It would seem clear that the idea of creating artificial life is a long debated one, with many questions unanswered and many more still unasked. One thing is certain, however: There is a certain attractiveness in the idea of creating a new form of life.

So, I decided to give it a shot.

As a first step, I needed to find a medium to work with. If literature was to be believed, life didn't necessarily have to be found in fleshy, organic shells; rather, a life-form could be crafted from any of a number of materials. As my Hebrew was decidedly rusty, the Golem would seem not to be an option, and I lacked the cadavers necessary for the construction of a Monster. It would seem that machines would be my best route. My creatures

would be electronic programs, interacting with each other within a world of silicon and circuitry. But were electronic life forms really alive? What criteria did something have to meet to be considered 'alive' anyway? Was it possible that computers would never be good for more than merely simulating life, that they would never be able to synthesize it? Before I could answer these questions, I needed to find a reasonable definition of life, one that wouldn't lead me off on dozens of philosophical tangents. (I was, of course, unable to find one, but I did manage to politely ignore the philosophical tangents associated with the definition I found.)

My next step was to choose a programming language to write in. What sort of language would be best able to produce the results I was looking for? This was a question I had answered before, in my own research paper, "Simulation in Object-Oriented Programming Languages". Object-Oriented programming languages, by their very structure contained numerous parallels to reality, making them ideal for simulating--or synthesizing--life. Naturally, there are many Object-Oriented programming languages, but I had many reasons for choosing Java, including its clear structure, easy graphical interface, and, most of all, its convenient routines for implementing concurrency.

What, you may ask, is concurrency, then? And how would it be useful here? Concurrency allows one to run several 'threads' or

independent processes at the same time in a multiprocessing environment. Non-concurrent processes must wait for one process to finish before taking their own turn. In reality, creatures all live, breathe, and interact with one another constantly. If real creatures don't take turns, neither should artificial creatures, after all.

Now, I had my tools. However, I was not so foolish as to believe that I was the first programmer to try to create artificial life. My research was to begin by exploring the advances computer science had already made in the field of artificial life. Along the way, many of my preconceived notions were crushed, while others were upheld--and I learned a few things about A-Life (or artificial life) that surprised even me. In the end, I had become convinced that computers could do more than merely simulate life. They could synthesize it as well.

The Definition of Life

Before proceeding further, it would be wise to define life. This is never an easy task. What qualities do such diverse objects as puppies, paramecia, bread mold, mandrake, and anabacteria all have in common, anyway? The best answer I could find was in Claus Emmeche's The Garden in the Machine, where he himself refers to J. Doyne Farmer and Aletta d'A. Belin. Life has the following properties:

> 1. Life is a pattern in space/time (rather than a specific material object). In other words, life is a distinct form of organization. We are, after all (and fortunately), more than what we eat. The molecules in our bodies and the cells in our tissues are renewed and exchanged innumerable times during our lifetimes.
>
> 2. Life loves self-reproduction (even mules, which are sterile, are created via a reproductive process).
>
> 3. Life is associated with information storage of a self-representation; that is, a partial description of itself (or of certain components necessary for production of the remainder under the system's continual self-organization).
>
> 4. Life thrives with the aid of metabolism; [it is distinct from its external environment and exchanges materials with its surroundings].
>
> 5. Life enters into functional interactions with the environment. (That is, organisms can adapt, but they also create and control their respective local environments.) Organisms have the ability to selectively respond to external stimuli (what the old physiologists called 'irritability').
>
> 6. Parts of living things have a critical internal dependency on each other (which means that organisms can die).

> 7. Life exhibits a dynamic stability in the face of perturbations (it can maintain form and organization up to a certain limit).
>
> 8. Life, not the individual but its lineage, has the ability to evolve. (Emmeche, p.38)

This definition, while very exacting, leaves a lot of leeway as to what can be alive. It doesn't demand that the creature be carbon-based, for example. It doesn't even require that the creature be physical, in any real sense, as long as it has an environment to exist in and interact with. This was a very convenient definition. Faced with the possibility that it was, perhaps, a little too convenient, I questioned whether or not an object's construction should be taken into account. After all, all the life on Earth seems to be carbon based. Could that be necessary for life? However, the other definitions I came across (in a variety of texts, including Games of Life, Artificial Life: Explorer's Kit, and a variety of different definitions from The Garden in the Machine) seemed to agree that the materials didn't matter. Just because life on Earth is carbon-based doesn't mean that other types of life are impossible.

The History of Artificial Life

Definition in hand, I looked to the past, to previous attempts to create life. One of the earlier names I came across was John von Neumann, who, in the 1940's, devised an algorithm describing an artificial life-form (see Appendix A for diagram). This creature would consist of four parts: First, the 'A' general constructor, which is a "factory" that must be given instructions from which it will fabricate some object using components from its environment. Next, the 'B' duplicator which takes an data input and returns two data outputs, each identical to the input. Then, 'C', the controller, which sends data to 'B', sends one of the two copies thus received to 'A', and the other to whatever 'A' constructs using that data, while keeping the original data itself. Lastly, and perhaps most importantly, is 'D' which is a piece of data which, if fed to 'A', would enable 'A' to create a copy of 'A', 'B', and 'C'--so 'D' is a self-description of the total machine. So, if 'D' is fed to 'C', the machine will replicate itself, and send a copy of 'D' to the replica, enabling the copy to replicate itself, and so on. This machine satisfies all but a few of the requirements for life, and, with the addition of a fifth, mutatable component 'E', can even evolve.

But von Neumann was not satisfied with this device. In itself, it needed too many extra components (such as sensors, to detect

the proper materials 'A' needs to do its construction). With the help of Stanislaw L. Ulam, he devised a new, simpler type of algorithm: Cellular Automata. Cellular Automata are composed of cells, each of which determines its state by observing the states of cells around them. This discovery was to prove pivotal in the development of artificial life, as it set the stage for Horton Conway to develop his famous "Game of Life".

The "Game of Life" is deceptively simple. Start with an (ideally) infinite grid of square cells, each having eight neighbors (one adjacent to each edge, and one adjacent to each corner). Each cell can either be on (alive) or off (dead). The game proceeds in generations, with each cell determining its state by the states of its neighbors in the previous generation. If a cell has two neighbors that are alive, its state remains the same, either alive or dead. If it has three neighbors that are alive, it comes alive if it was dead, or remains alive if it was already alive. In all other cases, the cell either remains dead or dies.

Experimenting with this simple cellular automaton, Conway soon discovered something interesting. While cells would often turn on and off in wild, random patterns, there were a few stable patterns that caught his attention (see Appendix B for diagrams). The most basic, a square block of four adjacent living cells, was a stable pattern. A row of three living cells was somewhat

stable; it switched from a horizontal row to a vertical in one generation, and back the next. Also fascinating was a group of four living cells which he referred to as a 'glider', which, over a period of four generations, would move southeast by one cell.

Conway was not satisfied, however. He unleashed his idea onto the programming world, asking people to find new patterns. And patterns were found: The 'glider gun', which spits out a glider every thirtieth generation, without being altered itself. The 'Eater' which can destroy or reverse the course of a glider approaching it suitably. The 'Puffer Train', which moves steadily east, leaving a messy trail of blocks, gliders, and random clouds of living cells in its wake. As more new patterns were discovered, those in the computer world became more and more enthralled by Conway's simple game.

Despite its name, Conway's "Game of Life" was not artificial life itself. Neither were any of the patterns it produced. (Some of the patterns did emulate a few of the requirements; the 'glider' moved through its environment, while the 'glider gun' produced offspring--or, perhaps, wastes--in the form of gliders.) But many in the scientific community thought that maybe his game, or something like it, might be a basis for real artificial life. Just as an organic creature's living cells are composed of nonliving components, so might an artificial life form be composed of the cells of a cellular automaton.

While Conway was poking around with his "Game of Life", another, far more sinister type of artificial life was innocently being created by unsuspecting programmers at AT&T Bell Laboratories and Xerox Corporation. The name of the game was "Core Wars", and programmers would craft special programs to destroy the programs created by their opponents. In short, they had created the first computer viruses. Innocuous enough at first, "these simple viruses soon displayed their unpredictable nature when one strain went out of control on a Xerox 530, forcing management to call a halt to the programmer's games." (Walnum, p.71) In subsequent years, computer users everywhere would come to know and loathe those diabolical programs known as computer viruses. Ironically, viruses might be the first computer programs to meet every requirement of the definition of life given above. They maintain their form to a limit. they reproduce, and can die. They metabolize (the same as any program, they utilize system resources and require electricity to maintain themselves), and can alter their environment. As that fateful event in the "Core Wars" showed, they can also evolve.

Probably the best attempt to create 'natural' (not created for a purpose, as viruses are) artificial life was made by Thomas Ray. His "Tierra" project was deliberately designed to allow a few simple electronic life forms the chance to evolve on their own (the programs evolved by virtue of certain routines designed

to add an occasional random element when a creature reproduced). After several generations, a multitude of different types of creatures developed (all originating with the simple, original type of creature). There were 'parasites', which depended on other creatures to do their reproducing for them, and 'hyperparasites', which did the same to parasites. 'Hyperparasites' often gathered in social groups to reproduce, which gave small 'cheater' programs a chance to slip in and trick the 'hyperparasites' into reproducing them for them. As CPU time was at a premium for these creatures, there were even some small programs, called 'liars' that tricked the CPU scheduler into thinking that they were larger than they were, earning themselves extra CPU time. The original creature had "evolved to 3,280 separate genotypes, though some of them may not have reproduced." (Thro, p.97) This impressive bit of programming was far beyond anything I had thought possible, and is probably the most advanced type of artificial life in existence.

Object-Oriented Programming

Suitably awed by the achievements of those that had gone before, I proceeded to take my turn. My tool of choice was Java, an Object-Oriented programming language. Why Object-Oriented? My reasons were many.

Object-Oriented programming, by its very structure, is organized in much the same way as reality. The ways that creatures interact with their environment and one another can be very similar to the ways that objects interact with one another in a programming environment. There are many parallels. Objects consist of bits of data and 'methods', or functions, for manipulating that data, and for interacting with other objects. Live creatures, too, have methods and data. The data could be the state of each nerve in the body, for example, or the pattern of the DNA. The methods would be how the creature responds to various internal and external stimuli. As a creature becomes hungry, it activates methods for seeking out and consuming food. If a creature's environment becomes too warm, it activates methods to move it to another, cooler location (provided that it is a mobile creature). If another creature attacks it, it responds with its own methods for defense and counterattack. All these stimuli could be considered 'messages', the means by which objects communicate with themselves and each other. An object

uses methods to manipulate its own data, to send signals to itself and other objects, and occasionally to manipulate the data of other objects. In the case of living creatures, manipulating data might include physically sinking one's teeth into another creature (altering the state of its skin), but the parallel remains.

Another very convenient parallel is the class hierarchy of objects in an Object-Oriented language. Any object is an implementation of a class, which defines all the characteristics shared by objects of that class. A class 'List', might allow one to create several different List objects, but all would have certain things in common, such as the ability to add new items to the list, or to remove old ones. But there might also exist a class called OrderedList, which creates objects that have all the characteristics of a List, but add the additional feature that any items on the list are kept in some sort of order. OrderedList could 'inherit' all of the methods and data necessary to be a list from the List class, as well as add a few new methods of its own, such as a routine for ordering data. So, every OrderedList would be a List, but not every List would be an OrderedList. The List class would be higher on the class hierarchy than the OrderedList class.

It is easy to see how a class hierarchy can be applied to living creatures. One could imagine a general class Animal, which

has several of the characteristics necessary to differentiate Animals from other living creatures, such as the ability to move, and to gather nourishment by eating other creatures. Mammal might be a subclass of Animal. All Mammals share the characteristics necessary to be an Animal, but have a few more requirements as well, such as producing live young, and nourishing them with milk. Rabbit might be a subclass of Mammal, including small creatures with large ears that breed often and eat only members of the Plant class. Rabbits would, of course, have all the traits shared by Mammals and Animals as well. CuteFuzzyWuzzyBunnies might be an even more specific subclass of Rabbit, including only those Rabbits that are cute, fuzzy, and undoubtedly wuzzy as well. One could proceed to subdivide ad nauseum, but I have made my point.

However, there are glaring exceptions to this hierarchy in nature. In an inheritance hierarchy in Timothy Budd's An Introduction to Object-Oriented Programming, Mammal has a subclass Platypus. Now, a Platypus is a Mammal, but it fails one of the determining factors (bearing live young). So, Platypus has an "exception" to this factor. So, how does an Object-Oriented programming language handle exceptions like this? In most O-O languages, a characteristic of an inherited class can often be "overridden" by writing "a method in a subclass having the same name as a method in the superclass, combined with a rule for how

the search for a method to match a specific message is conducted." [Budd, p. 8] Many systems, when given a message, will start with the lowest, most specific, class, and try to match the message to one of the methods there, if that fails the system goes up the hierarchy a step and tries to match it there. When an 'exception' is written, a method will stop there before it has a chance to step up the hierarchy. So, if a test was made to determine if a Platypus is a Mammal, the Platypus would have an exception stating that, despite the fact that Platypi lay eggs, a Platypus is a Mammal.

It would seem, then, that the Object-Oriented approach is the best way to create artificial life. As real life forms behave like objects, artificial life forms should as well. An argument might be raised at this point. O-O programming might be a wonderful tool for simulating life, but surely the mechanics behind the methods and the passing of messages must make some difference. If one has a robot that is designed to eat and digest cat food, and a cat that does the same, the robot cannot be said to be alive, can it?

The response to this dilemma is a hazy one. It depends on the situation. For example, if the robot is merely a construct designed to eat cat food without purpose, it could hardly be said to be alive. One might as well say that crystals are alive because they grow and retain their pattern. But consider a cat

robot that was designed to respond just as a cat would under a given stimulus (allowing for the fact that two real cats might not necessarily respond the same way to that stimulus), including the ability to become hungry, the ability to reproduce, the ability to respond to painful sensations or pleasurable ones, and anything else that a cat might be able to do. Perhaps this artificial kitty could be indistinguishable from other real cats to everyone but (possibly) its creator. Is this cat alive? Many would agree that it would be, in spite of the fact that it behaves the way it does because it was programmed to. In fact, it might be said to seem even more lifelike than other real creatures that are unconditionally categorized as life, such as your average hunk of coral. So, whether a creature acts a certain way because of billions of lines of code or because of an arrangement of billions of specialized cells has little bearing on whether a creature is alive or not.

Convinced of the superiority of the Object-Oriented paradigm for the creation of artificial life, I was then presented with another choice. There are many Object-Oriented programming languages available. So why Java?

Java, unlike some Object-Oriented programming languages has a clear Object-Oriented structure. Just about everything in Java is an object, with the exception of some of the primitive data types, like integers and characters (though special classes are

provided that allow you to treat even these as objects when necessary). Every function must be a method of some object, which in turn must be an instance of some class. When an object is passed by a function call, or message, one doesn't need to worry about allocating pointers (variables containing the address of an object in memory) or dereferencing variables. Whenever an object is passed, the object received by the function called is always the same object that had been passed, rather than a copy. Any changes made to that object within the function affect the object that was passed. In one sense, it could be said that a pointer to the object has been passed, though that terminology is not quite accurate (technically, the address contained within a pointer can be read as a value in such languages as C++; this is not the case in Java). As a veteran C++ programmer, I found this to be refreshing, rather than confining. On those rare occasions when I need to create a copy of an object, rather than alter the original, I can easily create routines to do so--and I have never come across a situation where I needed to know the address contained within a pointer.

Another advantage to Java that some might consider oppressive was the sensible class hierarchy. Any given class 'extends' (inherits from) exactly one superclass (the class that it is a subclass of). This is a sharp contrast to the multiple inheritance found in such languages as C++, where a given class

can have several superclasses. Multiple inheritance can be a powerful tool if used properly, but is difficult to implement and can cause many problems if implemented improperly. To make up for the lack of power that multiple inheritance could provide, Java provides 'implementation'. When a class implements another class, it becomes able to use any of the available functions of that class, but is not considered to be of that type. To take a mythical example, a Griffin class might extend the Lion class and implement the Eagle class. In that case, a Griffin would be a type of lion, but able to call upon Eagle functions like Squawk, or Fly, or MoltFeathers. However, it would not be a type of Eagle. A class can extend only one other class, but it can implement many (a class Manticore would, again, extend Lion, while also implementing Scorpion, and HumanBeing). The advantage to this is that a class has exactly one lineage, and overridden functions can be easily called from superclasses, a task which is somewhat trickier under multiple inheritance.

One of the most attractive features Java had for me in this case, however, were easy, built-in routines for generating graphics and for implementing concurrency. As Java was a language originally designed to create spiffy graphics for Internet web sites easily, it came equipped with a number of routines for drawing images and displaying them. This saved me the effort of calling upon some independent graphics tool, like OpenGL to draw

my pictures. (On a pertinent note, it must be realized that graphical images are not at all necessary for artificial life. The important thing is whether or not they behave like life within their own environment. The graphics were one of the last things I implemented when creating my own programs, and only after I was satisfied that my artificial creations were behaving the way I had expected them to.) Even better, by utilizing Java's Applet class, I created a final product that would be viewable on any computer with a web browser with Java capability.

Concurrency: A Vital Element

What about concurrency? What is it, and why is it important to artificial life? Concurrency is the ability to run several 'threads' or independent processes at the same time. Often, it is advantageous for a program to start a process running while continuing to run itself. For instance, a web page might display a bit of animation while still allowing the user to access the other functions of a web page. The animated picture would be an independent thread, running on its own while the web browser continued to run the web page itself (in this way, any Applet could be said to be a thread of a web page).

One distinct trait of living creatures is that they are all living at the same time. An animal doesn't sit and wait while all the other animals take their turns before moving itself. I decided that it would be to my advantage to create my own life under the same conditions, with each of my artificial creatures acting on its own, rather than in sequence.

Oddly, while it is useful for artificial life forms to operate concurrently with other artificial life forms, it is not necessary. Thomas Ray's aforementioned Tierra project was created on a uniprocessor system, without any concurrency at all. Each of his independent program-creatures operated on a turn-based sequence, with each creature utilizing the CPU for a time before

allowing the next creature in line to have a turn. Despite the fact that there is never more than one creature operating at a time, Ray's project still seems to be artificial life.

My Own Attempt at A-Life

It was time to begin my own programming project. Inspired by Conway's 'Game of Life', I decided that my project would be a modification of his. Unlike his, however, the rules that my little cell creatures would follow would be far more complex. Every one of my cells was a thread, running concurrently with all the other cells, and constantly rerunning its own behavior loop. Each cell could either be empty, or it could contain one of four different types of creatures, Veggie, Herbie, Omnie, or Carnie. In any case, the cell would have a value assigned to a variable called scentTraces, which would indicate what type of creature had last occupied that square, and how long ago that had occurred. Certain types of creatures would be able to use the value of scentTraces to track down other types of creatures. Each cell also contains a random number of soilNutrients, which Veggies need to thrive. If a cell is occupied, several other variables are assigned values, including Vitality, which indicates how much life a creature has, and creatureType, which determines what type of creature occupies the cell.

Now it was time to program the really gritty part of my artificial life: the behaviors of the cells under certain conditions. If a cell was empty (or Emptie, as I called them), a cell did very little each 'turn', or cycle through its loop. All

it would do was subtract an amount from its scentTraces, so that the scent would fade over time. The fun part was deciding how a cell would act if it contained a creature. First off, it was important to incorporate death into my project. Each creature starts with a certain amount of Vitality (100 for Veggies, 200 for Herbies, 300 for Omnies, and 400 for Carnies) Each turn, every creature 'dies' by 10 points. If the Vitality of any creature drops below 50, the creature dies and is converted into a certain number of soilNutrients, which are then added to the cell it occupied.

The only way to gain Vitality is by eating, something which each creature does in its own way. A Veggie, being immobile, feeds by subtracting 30 soilNutrients from the cell it occupies and adding 30 to its own Vitality. A Herbie is capable of consuming only Veggies, so it will wander around until it becomes adjacent to a Veggie, at which point it will subtract 20 Vitality from it every turn, increasing its own Vitality by the same amount until the Veggie dies. A Carnie can only eat Herbies or Omnies. By observing the scentTraces of the squares around it, the Carnie would move in the direction of the Herbie or Omnie that had most recently been nearby. Once it finds its prey, it subtracts an amount from its prey's Vitality (the amount is equal to one-tenth of the Carnie's Vitality, so the healthier the Carnie, the more damage it can do), and adds that amount to its

own Vitality. Omnies eat in much the same fashion, but are less choosy. While they prefer to eat Herbies or Carnies, they will eat Veggies if neither of the former are nearby. (Like the Carnie, the size of the bite that an Omnie can take varies with the size of the Omnie, but, as Omnies are less dependent on meat, they do only one-fifteenth of their Vitality in damage. Taken in conjunction with their lower average Vitality, this means that an Omnie rarely wins a one-on-one fight with a Carnie.)

It is worth mentioning that a Herbie is the only creature with a sense of self-preservation. If it finds itself next to a Carnie or an Omnie, it will flee in a direction away from its predator. If it can't flee, it will stand and fight feebly, taking 5 Vitality from its attacker each turn. This is pitifully inadequate, unless the Carnie or Omnie happens to be very weak, and the predator will usually win.

For any creature, if it eats enough so that it doubles its initial Vitality, the creature will spawn a new creature into an adjacent cell. This new creature will be of the same creatureType as the parent creature, and will have half of its parent's Vitality. The parent, accordingly, will lose half of its Vitality. Any creature will spawn to any unoccupied adjacent cell, except for the Veggie, which is a little more selective. The Veggie will look for the adjacent cell with the largest number of soilNutrients, and will place its offspring there,

ensuring that the child has a better chance of breeding on its own.

I mentioned that several creatures have the ability to move from place to place (whether it be to hunt for food or to flee predators). The mechanics behind this troubled me, at first. The best way to move a creature was to have a cell copy its creature into an adjacent cell, and then to erase the creature from itself. But was this really lifelike? In reality, creatures don't move from place to place by copying themselves into each new location, and erasing the old. Or do they? It might make for an interesting philosophical discussion to theorize on the real mechanics of movement (when someone moves, is the person they are in the new location the same person as was in the old?), but such a theoretical discussion is beyond the scope of this paper. I decided that, as the new creature was identical in all discernable ways to the old, it would not be unreasonable to consider it to be the same creature. After all, they do it all the time on Star Trek.

Aside from the philosophical difficulties, I ran into some physical difficulties that I had expected, but not looked forward to. Concurrency can be dangerous to handle. I discovered that some of my creatures were doing odd things, like two creatures would each try to move to the same cell, destroying one, or a creature would spawn an offspring on top of another creature. My

problem arose from the fact that a creature checked once to see if a cell is occupied before filling it (with itself or a spawn). As the cells were running concurrently, two creatures could decide upon the same cell as a target area before either had the chance to move in. So, both would be running their moving or spawning routines, and whoever occupied the cell second would overwrite the one that moved in first. I solved this dilemma by noting that certain routines (notably, those dealing with moving or spawning) needed to obtain a mutually exclusive hold on the target cell--a lock. So, I added a new boolean variable to my class of cells, one called Locked. Whenever Locked was true, that thread would cease all behavioral activity until it became unlocked, and no creature could try to occupy it. A cell could activate the lock on another cell when it wanted to move in or spawn to it, achieving the sole permission to do so. After it was finished, the cell would release its lock, allowing the now-occupied cell to begin running again.

My concurrency problems were the last problems I needed to solve before finishing my artificial life program. Now, with my critters eagerly waiting to be unleashed, I needed only to test them to ensure that they behaved in the ways that I expected, and, hopefully, in some valid ways that I didn't quite expect.

To test them, I devised a number of test scenarios. The first I entitled 'The Kudzu Factor', which unleashed a few Veggies on

an otherwise empty 10X10 grid of cells. As I expected, with no Herbies to eat them, the Veggies quickly spawned and filled every available square. The apparently random choices they made in choosing a square led me to believe that they were, indeed, choosing the most soilNutritious neighboring squares to spawn to, as I had intended.

The next scenario, which I called 'Eden', placed three Herbies in a 5X5 grid with five Veggies. The Herbies would wander around aimlessly (Veggies have no scent, and Herbies can't smell anyway, so Herbies are forced to move in random directions and hope that they stumble across food) until they came adjacent to some Veggie. The Veggies, in the meantime would spawn rapidly, just as in 'The Kudzu Factor'. Often, they would cover every square except for those occupied by Herbies. After a while, though, the Herbies would eat their way to an area that they could spawn to, at which point the Herbie population would begin to grow. When there were enough Herbies, they would invariably push back the Veggies, eventually consuming every last one. When the last Veggie was gone, the Herbies would stagger blindly around until they all starved to death. Perhaps 'Eden' was a poor choice for a title. Obviously, Herbies need some sort of predator to hold their population in check.

And so, I created 'Bambi Vs Godzilla' to introduce the Carnie. This scenario consisted of one Carnie and one Herbie, on a

variable sized grid. The Herbie was a special one that began with 399 Vitality rather than the usual 200, to prevent it from starving to death before the Carnie had a chance to catch it. In this scenario, the Herbie would flee the Carnie, which would pursue by following its scentTraces. I intended this scenario merely to be a demonstration of the Carnie's ability to track by scent. The outcome of a Carnie battling a Herbie, no matter how big the Herbie was initially, was never in question.

My fourth scenario, 'The Great Hunt' was designed to discover how superior Carnies were to Omnies as far as meat-eating was concerned. Six Omnies were pitted against two Carnies on a 5X5 grid. It's a pretty even fight. About half the time, most of the Omnies will concentrate on one Carnie before attacking the other, and so will overwhelm the Carnies by sheer numbers. At other times, they will attack in waves of two or three Omnies, which the Carnies can usually defeat. Occasionally, a Carnie will even gorge itself on enough Omnies to spawn a third Carnie, at which point the battle is hopeless for the remaining Omnies. This was probably the most entertaining scenario I set up.

My fifth scenario was entitled 'Pit Fighter', and I created it more or less in a fit of whimsy. In a 5X5 grid, I placed a Veggie, a Herbie, an Omnie, and a Carnie, to see which would be the victor. Surprisingly, the winner usually turned out to be the Veggie. If the Veggie could survive until the Carnie had finished

off the Omnie and the Herbie, it was free to spawn until every cell but the Carnie's was occupied. It could take the Carnie's cell after it starved to death. If the Herbie and the Omnie finished off the Veggie, the Carnie usually won. But on rare occasions, if the Carnie was really unlucky in finding the scents of the Omnie and the Herbie, the Omnie could eat the Herbie and then feed on Veggies (which have usually reproduced by now) until there were enough Omnies to overwhelm the Carnie. The Herbie never won, though I hypothesized that it might if both the Omnie and the Carnie died before picking up its scent. All in all, an interesting experiment.

Lastly, I perfected the general random creature generator that I had done most of my initial experiments with, and entitled it 'Random Genesis'. This can take any size grid, and fills it with creatures at random. Aware of the dangers of having too many Carnies, or too few Veggies, I made sure that a cell had a greater chance of being one type of creature than another (a cell has a 20% chance of being a Veggie, 15% of being a Herbie, 10% of being an Omnie, and 5% of being a Carnie). It was with this scenario that I really began to notice imbalances in my creatures. Normally, Carnies, no matter how few of them there were, would rapidly finish off the Omnies and Herbies, dooming themselves to starve among the Veggies. If the Carnies were unable to finish off the Omnies, the Omnies would usually do the

finishing, and then proceed to eat all of the Veggies until they were left on a foodless grid. Sometimes, if the Veggies spawned rapidly enough and the Carnies ate all the Omnies, the Herbies would sit and eat peacefully as the few remaining trapped Carnies starved to death. Then, the Herbies would gradually overwhelm the Veggies until they, too, were left without food.

So, how do my creatures stack up against the eight rules of life? Each creature does retain a specific, if simplistic, pattern. It has the same traits and general qualities whether it is newborn or several cycles old. These creatures do reproduce--in fact, it's the primary goal of their lives. Each creature has a single data item explaining what it is (creatureType), and this data is used to define itself and its functions. My life does metabolize. They eat whatever they eat, and eventually produce offspring as a result. My creatures interact with their environment. Most of them move around, and even Veggies search around for fertile squares when reproducing. My organisms can die. They are dependent on the value of their Vitality for continued survival. Unless they reach a low enough Vitality value, my creatures retain their identity as represented by their creatureType. The last rule of life is the only one where I fall short. My creatures have no method by which they can evolve. No matter how many times it spawns, a Herbie is a Herbie is a Herbie. So, while my program does come very close to qualifying

as artificial life, it still falls short.

There are many ways that I could improve my program. First, and more importantly, I could give it the status of real artificial life by giving it some kind of factor by which its creatures could evolve--perhaps by introducing a random element that allows some Herbies to develop better defenses. These Herbies could fight off Carnies and Omnies more efficiently, and so survive to produce more powerful offspring, as the less defensive Herbies die. Or, perhaps I could give some Carnies the ability to eat Veggies if no other food can be found. It would be difficult, but, as Thomas Ray showed, hardly impossible.

Another improvement that might be beneficial would be to create subclasses of my single gridSquare class for each type of creature. As matters stand, the defining characteristics of every type of creature is contained in the same class. To create a Veggie class, a Herbie Class, an Omnie class, and a Carnie class would make my program much more readable, and make its structure more similar to reality. I would also be able to add new creature types much more easily, like, perhaps, a Scavie, which is a mobile creature with the ability to consume soilNutrients.

Another improvement I could make is to try to find a way to make my random scenario a little more balanced. As Carnies so often seem to eat everyone else too quickly, perhaps I could make them a little less voracious, or allow other creatures to spawn

more quickly. Veggies seem to spread too quickly, so, perhaps I need to reduce the number of soilNutrients that are added to a cell when a creature dies. There are a number of ways I could improve things merely by tinkering with a few variables.

This is the real value of my program. Artificial life, or merely simulated life, it can still be used in a variety of ways. With the proper adjustments, it could be used to simulate the balance of predators, herbivores, and plants in nature. With a few other adjustments it could make an interesting game, if not quite on par with the popularity of Conway's 'Game of Life'.

My Conclusions

In researching this project, I discovered many things about artificial life. Probably most importantly, I learned that it wasn't very difficult to create. "In fact, it's quite easy to create life," (Emmeche, p.3) as Thomas Ray once noted in a lecture. Quite easy, but only after one has decided just what life is. The definition I used, created by Farmer and Belin, is very clinical and precise. But is it really enough? I believe so, but I also believe that belief is not enough to make a thing so.

However, there seems to be little doubt that artificial life can be at least as lively as much of the life that we take for granted every day. It may be a long time in the future before we see artificial life of a complexity even nearing that of, for instance, a cockroach. But in the meantime, Ray's creations are still evolving. Who knows what they may someday become?

Bibliography:

Budd, Timothy. An Introduction to Object-Oriented Programming. Reading, Mass.: Addison-Westley, 1991.

Cornell, Gary, Cay S. Horstmann. Core Java. Mountain View, CA: Sunsoft Press, 1996.

Emmeche, Claus. (Trans. by Steven Sampson). The Garden in the Machine. Princeton, NJ: Princeton University Press, 1994.

Flanagan, David. Java in a Nutshell. Bonn: O'Reilly & Associates, Inc., 1996.

Mcintosh, Harry. Talk Java to Me. Corte Madera, CA: Waite Group Press, 1996.

Sigmund, Karl. Games of Life. Oxford: Oxford University Press, 1993.

Simons, Geoff. Are Computers Alive. Thetford, Great Britian: Thetford Press Ltd., 1983.

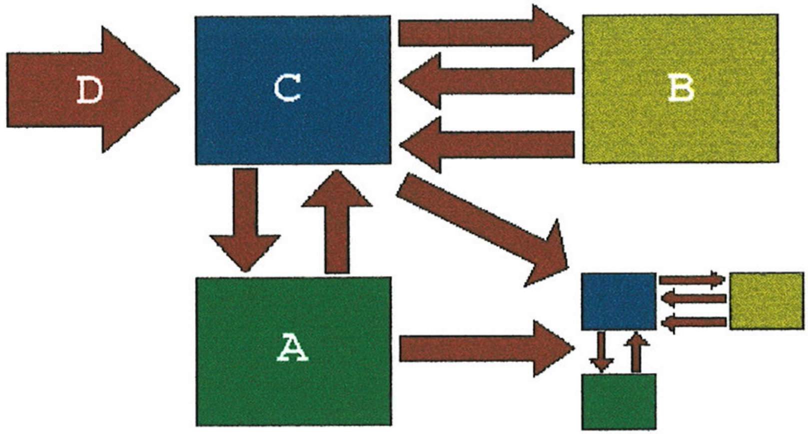
Thro, Ellen. Artificial Life Explorer's Kit. Carmel, IN: Sams Publishing, 1993.

Walnum, Clayton. Adventures in Artificial Life. Carmel, IN: Que Corporation, 1993.

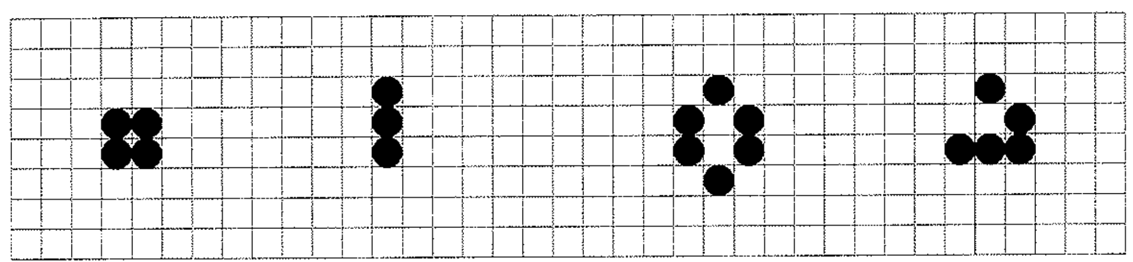
Special thanks to Lynn Ziegler, whose 'testGrid' and 'gridThing' programs were pivotal in helping me to learn to program Cellular Automata in Java.

Appendix A:

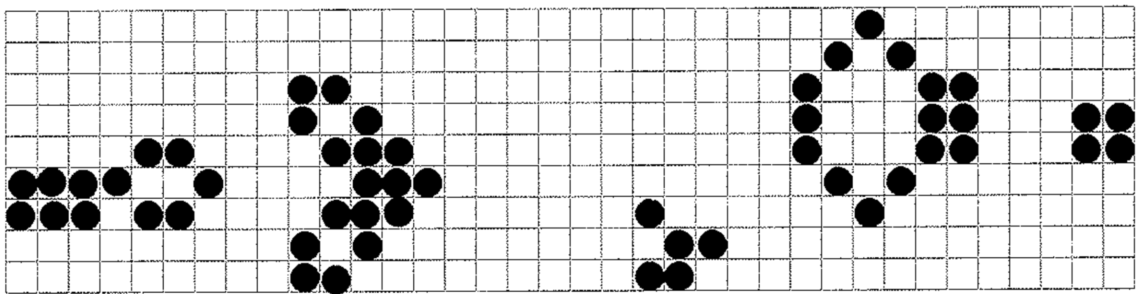
Von Neumann's Self-Replicating Automaton



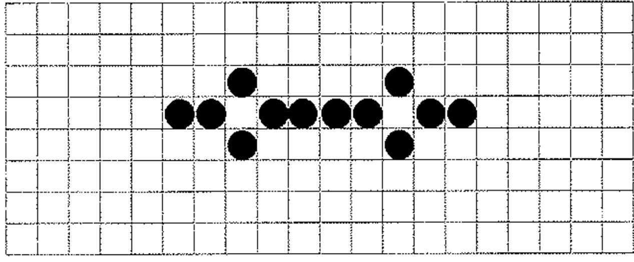
Appendix B: A Few 'Game of Life' Structures



Block Stoplight Ship Glider



Glider Gun (with Glider)



Eater (or Pentadecathlon)

Appendix C: Java Code

LifeGrid.java

```
import java.awt.*;
import java.util.*;
import java.applet.Applet;

public class LifeGrid extends Applet
{
    static Color darkGreen = new Color(0, 122, 0);
    static Color darkOrange = new Color(122, 61, 0);
    public int size = 10;
    public gridSquare myLifeGrid[][]=
                           new gridSquare[size][size];
    public int running = 0;
    public int scenerio = 6;
    Graphics offScreenGraphics;
    Image offScreenImage;

    public void init()
    {
        offScreenImage = createImage(size().width,
                                     size().height);
        offScreenGraphics = offScreenImage.getGraphics();
        scenerio = Integer.parseInt(getParameter("scenerio"));
        size = Integer.parseInt(getParameter("gridsize"));
        for(int i = 0; i < size; i++)
        {
            for(int j = 0; j < size; j++)
            {
                myLifeGrid[i][j]=
                      new gridSquare(i, j, size, size,
                                    size().width,
                                    size().height,
                                    scenerio,
                                    myLifeGrid, this);
            }
        }
    }

    public void update(Graphics g)
    {
        paint(offScreenGraphics);
        g.drawImage(offScreenImage, 0, 0, null);
    }

    public void paint(Graphics g)
```

```
    {
        g.setColor(darkOrange);
        g.fillRect(0, 0, size().width, size().height);
        g.setColor(Color.black);
        g.drawRect(0, 0, size().width, size().height);
        g.drawRect(1, 1, size().width - 1, size().height - 1);
        g.drawRect(2, 2, size().width - 2, size().height - 2);
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                int x = 0;
                int y = 0;
// Choose the color (dependant on the creature type).
                if (myLifeGrid[i][j].creatureType ==
                    myLifeGrid[i][j].emptie)
                    g.setColor(darkOrange);
                else if (myLifeGrid[i][j].creatureType
                         == myLifeGrid[i][j].veggie)
                    g.setColor(darkGreen);
                else if (myLifeGrid[i][j].creatureType
                         == myLifeGrid[i][j].herbie)
                    g.setColor(Color.yellow);
                else if (myLifeGrid[i][j].creatureType
                         == myLifeGrid[i][j].omnie)
                    g.setColor(Color.blue);
                else
                    g.setColor(Color.red);
// Draw a circle in the appropriate place, or abort the
// program if the drawing area is too small.
                if (((size().width / size) < 20) ||
                    ((size().height / size) < 20))
                {
                    System.exit(0);
                }
                else
                {
                    x = (((2 * i) + 1) * (size().width
                         / size) / 2) - 10;
                    y = (((2 * j) + 1) * (size().height
                         / size) / 2) - 10;
                    g.fillOval(x, y, 20, 20);
                    if (myLifeGrid[i][j].creatureType != 0)
                    {
                        g.setColor(Color.black);
                        g.drawOval(x, y, 20, 20);
                    }
                }
```

```
            }
        }
    }

    public boolean mouseDown(Event e, int x, int y)
    {
        if (running == 0)
        {

            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    myLifeGrid[i][j].start();
                }
            }
            running = 1;
        }
        else if (running == 1)
        {
            for (int i = 0; i < size; i++)
            {
                for (int j = 0; j < size; j++)
                {
                    myLifeGrid[i][j].stop();
                }
            }
            destroy();
            running = 2;
        }
        else
        {
            running = 0;
            init();
        }
        return true;
    }
     //{{DECLARE_CONTROLS
     //}}
}
```

gridSquare.java

```
import java.awt.*;
import java.util.*;
import java.lang.Math;

class gridSquare extends Thread
{
// Creature types:
    final int emptie = 0;
    final int veggie = 1;
    final int herbie = 2;
    final int omnie = 3;
    final int carnie = 4;
// The grid:
    private LifeGrid BigDaddyMain;
    private gridSquare grid[][];
    private int xSize;
    private int ySize;
    private int appXSize;
    private int appYSize;
    public int scenerio;
// Stats for this gridSquare;
    public int gridX;
    public int gridY;
    public int creatureType;
    public int vitality;
    public int soilNutrients;
    public int scentTraces;
    public int biteSize = 0;
    public int trail = 0;
    public int predLocY = 0;
    public int predLocX = 0;
    public int preyLocX = 0;
    public int preyLocY = 0;
    public boolean nextToPred = false;
    public boolean nextToPrey = false;
// A lock variable, for mutual exclusion. Whenever a
// thread wishes to alter this thread in such a way that
// a third thread might interfere harmfully, Locked will
// become true, disallowing any other thread access to
// this one.
    public boolean Locked = false;

    public gridSquare(int xLoc, int yLoc, int sizeX, int
                      sizeY, int appX, int appY, int s,
                      gridSquare ingrid[][], LifeGrid dad)
    {
```

```
// First, set a few variables that the gridSquare needs to
// know, like where it is and how big the grid is.
        gridX = xLoc;
        gridY = yLoc;
        xSize = sizeX;
        ySize = sizeY;
        appXSize = appX;
        appYSize = appY;
        grid = ingrid;
        BigDaddyMain = dad;
        scenerio = s;
// Next, decide what creature, if any, is initially in
// the gridSquare. Each creature type has it's own
// initial vitality level. Several scenerioes are
// available.
        switch (scenerio)
        {
            case (0):
// THE KUDZU FACTOR
// This shows the affect of unrestrained Veggies. It
// assumes a 5X5 or larger grid.
            {
                if (((gridX == 0) && (gridY == 4)) ||
                    ((gridX == 1) && (gridY == 0)) ||
                    ((gridX == 2) && (gridY == 1)) ||
                    ((gridX == 3) && (gridY == 0)) ||
                    ((gridX == 4) && (gridY == 1)))
                {
                    creatureType = veggie;
                    vitality = 100;
                    scentTraces = 0;
                }
                break;
            }
            case (1):
            {
// EDEN
// An idyllic scenerio, where Herbies romp and play among
// the Veggies, doing their best to keep the plant
// population down. Again, a 5X5 minimum.
                if (((gridX == 0) && (gridY == 0)) ||
                    ((gridX == 0) && (gridY == 1)) ||
                    ((gridX == 1) && (gridY == 0)) ||
                    ((gridX == 1) && (gridY == 1)) ||
                    ((gridX == 4) && (gridY == 0)))
                {
                    creatureType = veggie;
                    vitality = 100;
```

```
                    scentTraces = 0;
                }
                if (((gridX == 4) && (gridY == 4)) ||
                    ((gridX == 2) && (gridY == 4)) ||
                    ((gridX == 3) && (gridY == 3)))
                {
                    creatureType = herbie;
                    vitality = 200;
                    scentTraces = 102;
                }
                break;
            }
// BAMBI VS. GODZILLA
// Watch a Carnie track down and kill a few Herbies.
// Notice how it follows the Herbie's path by following
// its scent traces. Requires 3X5 grid minimum.
            case (2):
            {
                if ((gridX == 0) && (gridY == 0))
                {
                    creatureType = carnie;
                    vitality = 400;
                    scentTraces = 104;
                }
                if ((gridX == 1) && (gridY == 1))
                {
                    creatureType = herbie;
                    vitality = 399;
                    scentTraces = 102;
                }
                break;
            }
// THE GREAT HUNT
// Can six Omnies take down two Carnies?
// 5X5 grid minimum.
            case (3):
            {
                if (((gridX == 0) && (gridY == 0)) ||
                    ((gridX == 2) && (gridY == 0)) ||
                    ((gridX == 4) && (gridY == 0)) ||
                    ((gridX == 0) && (gridY == 4)) ||
                    ((gridX == 2) && (gridY == 4)) ||
                    ((gridX == 4) && (gridY == 4)))
                {
                    creatureType = omnie;
                    vitality = 300;
                    scentTraces = 103;
                }
```

```
                    if (((gridX == 1) && (gridY == 2)) ||
                        ((gridX == 3) && (gridY == 2)))
                    {
                        creatureType = carnie;
                        vitality = 400;
                        scentTraces = 104;
                    }
                    break;
                }
// PIT FIGHTER
// A Veggie, a Herbie, an Omnie, and a Carnie. Who will
// be the victor? 5X5.
                case (4):
                {
                    if ((gridX == 2) && (gridY == 2))
                    {
                        creatureType = veggie;
                        vitality = 100;
                        scentTraces = 0;
                    }
                    if ((gridX == 2) && (gridY == 0))
                    {
                        creatureType = herbie;
                        vitality = 200;
                        scentTraces = 102;
                    }
                    if ((gridX == 0) && (gridY == 4))
                    {
                        creatureType = omnie;
                        vitality = 300;
                        scentTraces = 103;
                    }
                    if ((gridX == 4) && (gridY == 4))
                    {
                        creatureType = carnie;
                        vitality = 400;
                        scentTraces = 104;
                    }
                    break;
                }
// RANDOM GENESIS
// This is the official, random creature generator. It
// works well with any grid size.
// Statistics:
// There is a 50% chance that the square will be Emptie.
// There is a 20% chance that it will be a Veggie.
// There is a 15% chance that it will be a Herbie.
// There is a 10% chance that it will be an Omnie.
```

```
// There is a 5% chance that it will be a Carnie.
            default:
            {
                double creatureChoice;
                creatureChoice = Math.random();
                if (creatureChoice < 0.5)
                {
                    creatureType = emptie;
                    vitality = 0;
                    scentTraces = 0;
                }
                else if (creatureChoice < 0.7)
                {
                    creatureType = veggie;
                    vitality = 100;
                    scentTraces = 0;
                }
                else if (creatureChoice < 0.85)
                {
                    creatureType = herbie;
                    vitality = 200;
                    scentTraces = 102;
                }
                else if (creatureChoice < 0.95)
                {
                    creatureType = omnie;
                    vitality = 300;
                    scentTraces = 103;
                }
                else
                {
                    creatureType = carnie;
                    vitality = 400;
                    scentTraces = 104;
                }
                break;
            }
    }
// Then, decide how nutritious the soil for the
// gridSquare is.
        soilNutrients = (int)(1000 * Math.random());
// and draw it.
        update();
    }

// My implimentation of the update() function, which calls
// the main class to activate its paint() function, and
// redraw the picture.
```

```
    public void update ()
    {
        BigDaddyMain.repaint();
    }

// And now, the Run routine.

    public void run()
    {
        while (true)
        {
        try{sleep(500);}
            catch(InterruptedException e) {}
        while (Locked) {}
// Here's where the behavior of each type of creature is
// executed. First, if a square is Emptie:
        if (creatureType == emptie)
        {
            scentTraces = scentTraces - 5;
            if (scentTraces < 0)
                scentTraces = 0;
        }
// Next, if a square contains a veggie:
        else if (creatureType == veggie)
        {
            vitality = vitality - 10;
// If the Veggie is dead, kill it.
            if (vitality < 50)
            {
                vitality = 0;
                creatureType = emptie;
                soilNutrients = soilNutrients + 300;
                update();
            }
// Otherwise, feed it.
            else if (soilNutrients == 0)
            {
                vitality = vitality + 0;
            }
            else if (soilNutrients < 30)
            {
                vitality = vitality + soilNutrients;
                soilNutrients = 0;
            }
            else
            {
                vitality = vitality + 30;
                soilNutrients = soilNutrients - 30;
```

```
            }
// This unpleasant bit of code allows a Veggie to spawn,
// if there is a square available to spawn to.
            if (vitality >= 200)
            {
                int spawnX = gridX;
                int spawnY = gridY;
                int fruitful = 0;
                for (int i = -1; i < 2; i++)
                {
                    for (int j = -1; j < 2; j++)
                    {
                        if (((gridY + j) >= 0) && ((gridX
                            + i) >= 0) && ((gridY + j) <
                            ySize) && ((gridX + i) <
                            xSize) && ((j != 0) ||
                            (i != 0)))
                        {
                            if ((grid[gridX + i][gridY +
                                j].creatureType == emptie)
                                && (grid[gridX + i][gridY
                                + j].soilNutrients >
                                fruitful) && !grid[gridX +
                                i][gridY + j].Locked)
                            {
                                fruitful = grid[gridX +
                                i][gridY +
                                j].soilNutrients;
                                spawnX = gridX + i;
                                spawnY = gridY + j;
                            }
                        }
                    }
                }
                if ((spawnX != gridX) || (spawnY !=
                    gridY) && !grid[spawnX][spawnY].Locked)
                {
                    grid[spawnX][spawnY].Locked = true;
                    grid[spawnX][spawnY].creatureType =
                        veggie;
                    grid[spawnX][spawnY].vitality =
                                vitality / 2;
                    vitality = vitality / 2;
                    grid[spawnX][spawnY].Locked = false;
                    update();
                }
            }
        }
```

```
// Now, the instructions for the Herbie:
        else if (creatureType == herbie)
        {
            scentTraces = 102;
            vitality = vitality - 10;
// Is the Herbie dead? If so, kill it and make the square
// Emptie. Skip the rest.
            if (vitality < 50)
            {
                creatureType = emptie;
                vitality = 0;
                soilNutrients = soilNutrients + 267;
                update();
            }
            else
            {
// First, the Herbie scans the area for food or foes:
                nextToPred = false;
                for (int i = -1; i < 2; i++)
                {
                    for (int j = -1; j < 2; j++)
                    {
                        if (((gridX + i) >= 0) && (gridX +
                            i < xSize) && ((gridY + j) >=
                            0) && ((gridY + j) < ySize) &&
                            ((grid[gridX + i][gridY +
                            j].creatureType == omnie) ||
                            (grid[gridX + i][gridY +
                            j].creatureType == carnie)))
                        {
                            nextToPred = true;
                            predLocX = i;
                            predLocY = j;
                        }
                    }
                }
                nextToPrey = false;
                for (int i = -1; i < 2; i++)
                {
                    for (int j = -1; j < 2; j++)
                    {
                        if (((gridX + i) >= 0) && ((gridX
                            + i) < xSize) && ((gridY + j)
                            >= 0) && ((gridY + j) < ySize)
                            && (grid[gridX + i][gridY +
                            j].creatureType == veggie))
                        {
                            nextToPrey = true;
```

```
                                        preyLocX = i;
                                        preyLocY = j;
                                    }
                                }
                            }
// If there is something about to chew on it, the Herbie
// tries to flee. If there's nowhere to run, it makes a
// feeble attack.
                            if (nextToPred)
                            {
                                if (predLocX == 0)
                                {
                                    if (((gridY - predLocY) >= 0) &&
                                        ((gridY - predLocY) < ySize)
                                        && (grid[gridX][gridY -
                                        predLocY].creatureType ==
                                        emptie) && !grid[gridX][gridY
                                        - predLocY].Locked)
                                    {
                                        grid[gridX][gridY -
                                            predLocY].Locked = true;
                                        grid[gridX][gridY -
                                            predLocY].creatureType =
                                            herbie;
                                        grid[gridX][gridY -
                                            predLocY].vitality =
                                            vitality;
                                        grid[gridX][gridY -
                                        predLocY].scentTraces = 102;
                                        creatureType = emptie;
                                        vitality = 0;
                                        grid[gridX][gridY -
                                            predLocY].Locked = false;
                                        update();
                                    }
                                    else if (((gridY - predLocY) >= 0)
                                             && ((gridY - predLocY) <
                                             ySize) && ((gridX - 1) >=
                                             0) && (grid[gridX -
                                             1][gridY -
                                             predLocY].creatureType
                                             == emptie) && !grid[gridX -
                                             1][gridY -
                                             predLocY].Locked)
                                    {
                                        grid[gridX - 1][gridY -
                                            predLocY].Locked = true;
                                        grid[gridX - 1][gridY -
```

```
        predLocY].creatureType =
        herbie;
    grid[gridX - 1][gridY -
        predLocY].vitality =
        vitality;
    grid[gridX - 1][gridY -
        predLocY].scentTraces =
        102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX - 1][gridY -
        predLocY].Locked = false;
    update();
}
else if (((gridY - predLocY) >= 0)
         && ((gridY - predLocY) <
         ySize) && ((gridX + 1) <
         xSize) && (grid[gridX +
         1][gridY -
         predLocY].creatureType ==
         emptie) && !grid[gridX +
         1][gridY
         - predLocY].Locked)
{
    grid[gridX + 1][gridY -
        predLocY].Locked = true;
    grid[gridX + 1][gridY -
        predLocY].creatureType =
        herbie;
    grid[gridX + 1][gridY -
        predLocY].vitality =
        vitality;
    grid[gridX + 1][gridY -
        predLocY].scentTraces =
        102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX + 1][gridY -
        predLocY].Locked = false;
    update();
}
else
{
    grid[gridX + predLocX][gridY +
        predLocY].vitality =
        grid[gridX +
        predLocX][gridY +
        predLocY].vitality - 5;
```

```
    }
}
else if (predLocY == 0)
{
    if (((gridX - predLocX) >= 0) &&
        ((gridX - predLocX) < xSize)
        && (grid[gridX -
        predLocX][gridY].creatureType
        == emptie) && !grid[gridX -
        predLocX][gridY].Locked)
    {
        grid[gridX -
            predLocX][gridY].Locked
            = true;
        grid[gridX -
            predLocX][gridY].creatureType
            = herbie;
        grid[gridX -
            predLocX][gridY].vitality
            = vitality;
        grid[gridX -
            predLocX][gridY].scentTraces
            = 102;
        creatureType = emptie;
        vitality = 0;
        grid[gridX -
            predLocX][gridY].Locked
            = false;
        update();
    }
    else if (((gridX - predLocX) >= 0)
             && ((gridX - predLocX) <
             xSize) && ((gridY - 1) >=
             0) && (grid[gridX -
             predLocX][gridY -
             1].creatureType ==
             emptie) && !grid[gridX -
             predLocX][gridY
             - 1].Locked)
    {
        grid[gridX - predLocX][gridY -
            1].Locked = true;
        grid[gridX - predLocX][gridY -
            1].creatureType = herbie;
        grid[gridX - predLocX][gridY -
            1].vitality = vitality;
        grid[gridX - predLocX][gridY -
            1].scentTraces = 102;
```

```
            creatureType = emptie;
            vitality = 0;
            grid[gridX - predLocX][gridY -
                1].Locked = false;
            update();
        }
        else if (((gridX - predLocX) >= 0)
                 && ((gridX - predLocX) <
                 xSize) && ((gridY + 1) <
                 ySize) && (grid[gridX -
                 predLocX][gridY +
                 1].creatureType ==
                 emptie) && !grid[gridX -
                 predLocX][gridY +
                 1].Locked)
        {
            grid[gridX - predLocX][gridY +
                1].Locked = true;
            grid[gridX - predLocX][gridY +
                1].creatureType = herbie;
            grid[gridX - predLocX][gridY +
                1].vitality = vitality;
            grid[gridX - predLocX][gridY +
                1].scentTraces = 102;
            creatureType = emptie;
            vitality = 0;
            grid[gridX - predLocX][gridY +
                1].Locked = false;
            update();
        }
        else
        {
            grid[gridX + predLocX][gridY +
            predLocY].vitality =
            grid[gridX + predLocX][gridY +
            predLocY].vitality - 5;
        }
    }
    else
    {
        if (((gridX - predLocX) >= 0) &&
            ((gridX - predLocX) < xSize)
            && ((gridY - predLocY) >= 0)
            && ((gridY - predLocY) < ySize)
            && (grid[gridX - predLocX][gridY
            - predLocY].creatureType ==
            emptie) && !grid[gridX -
            predLocX][gridY -
```

```
    predLocY].Locked)
{
    grid[gridX - predLocX][gridY -
        predLocY].Locked = true;
    grid[gridX - predLocX][gridY -
        predLocY].creatureType =
        herbie;
    grid[gridX - predLocX][gridY -
        predLocY].vitality =
        vitality;
    grid[gridX - predLocX][gridY -
        predLocY].scentTraces =
        102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX - predLocX][gridY -
        predLocY].Locked = false;
    update();
}
else if (((gridX - predLocX) >= 0)
         && ((gridX - predLocX) <
         xSize) && (grid[gridX -
         predLocX][gridY].creatureType
         == emptie) && !grid[gridX -
         predLocX][gridY].Locked)
{
    grid[gridX -
        predLocX][gridY].Locked
        = true;
    grid[gridX -
        predLocX][gridY].creatureType
        = herbie;
    grid[gridX -
        predLocX][gridY].vitality
        = vitality;
    grid[gridX -
        predLocX][gridY].scentTraces
        = 102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX -
        predLocX][gridY].Locked
        = false;
    update();
}
else if (((gridY - predLocY) >= 0)
         && ((gridY - predLocY) <
         ySize) && (grid[gridX][gridY
```

```
         - predLocY].creatureType
         == emptie) &&
         !grid[gridX][gridY
         - predLocY].Locked)
{
    grid[gridX][gridY -
        predLocY].Locked = true;
    grid[gridX][gridY -
        predLocY].creatureType =
        herbie;
    grid[gridX][gridY -
        predLocY].vitality =
        vitality;
    grid[gridX][gridY -
        predLocY].scentTraces =
        102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX][gridY -
        predLocY].Locked = false;
    update();
}
else if (((gridX - predLocX) >= 0) &&
    ((gridX - predLocX) < xSize)
    && (grid[gridX - predLocX][gridY
    + predLocY].creatureType ==
    emptie) && !grid[gridX -
    predLocX][gridY +
    predLocY].Locked)
{
    grid[gridX - predLocX][gridY +
        predLocY].Locked = true;
    grid[gridX - predLocX][gridY +
        predLocY].creatureType =
        herbie;
    grid[gridX - predLocX][gridY +
        predLocY].vitality =
        vitality;
    grid[gridX - predLocX][gridY +
        predLocY].scentTraces =
        102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX - predLocX][gridY +
        predLocY].Locked = false;
    update();
}
else if (((gridY - predLocY) >= 0)
```

```
                                        && ((gridY - predLocY) < ySize)
                                        && (grid[gridX + predLocX][gridY
                                        - predLocY].creatureType ==
                                        emptie) && !grid[gridX +
                                        predLocX][gridY -
                                        predLocY].Locked)
                                    {
                                        grid[gridX + predLocX][gridY -
                                            predLocY].Locked = true;
                                        grid[gridX + predLocX][gridY -
                                            predLocY].creatureType =
                                            herbie;
                                        grid[gridX + predLocX][gridY -
                                            predLocY].vitality =
                                            vitality;
                                        grid[gridX + predLocX][gridY -
                                            predLocY].scentTraces =
                                            102;
                                        creatureType = emptie;
                                        vitality = 0;
                                        grid[gridX + predLocX][gridY -
                                            predLocY].Locked = false;
                                        update();
                                    }
                                    else
                                    {
                                        grid[gridX + predLocX][gridY +
                                        predLocY].vitality =
                                        grid[gridX + predLocX][gridY +
                                        predLocY].vitality - 5;
                                    }
                                }
                            }
// If no predators are present, the Herbie nibbles on any
// nearby Veggies.
                            else if (nextToPrey)
                            {
                                if (grid[gridX + preyLocX][gridY +
                                    preyLocY].vitality < 20)
                                {
                                    vitality = vitality +
                                        grid[gridX + preyLocX][gridY +
                                        preyLocY].vitality;
                                    grid[gridX + preyLocX][gridY +
                                        preyLocY].vitality = 0;
                                }
                                else
                                {
```

```
                        vitality = vitality + 20;
                        grid[gridX + preyLocX][gridY +
                            preyLocY].vitality =
                            grid[gridX + preyLocX][gridY +
                            preyLocY].vitality - 20;
                    }
// If, after eating, the Herbie is lively enough, and there
// is enough space, it spawns.
                    if (vitality >= 400)
                    {
                        int spawnX = gridX;
                        int spawnY = gridY;
                        for (int i = -1; i < 2; i++)
                        {
                            for (int j = -1; j < 2; j++)
                            {
                                if (((gridY + j) >= 0) &&
                                    ((gridX + i) >= 0) &&
                                    ((gridY + j) < ySize) &&
                                    ((gridX + i) < xSize) &&
                                    ((j != 0) || (i != 0)))
                                {
                                    if ((grid[gridX + i][gridY
                                    + j].creatureType ==
                                    emptie) && !grid[gridX +
                                    i][gridY + j].Locked)
                                    {
                                        spawnX = gridX + i;
                                        spawnY = gridY + j;
                                    }
                                }
                            }
                        }
                        if ((spawnX != gridX) || (spawnY !=
                            gridY) &&
!grid[spawnX][spawnY].Locked)
                        {
                            grid[spawnX][spawnY].Locked = true;
                            grid[spawnX][spawnY].creatureType =
                                herbie;
                            grid[spawnX][spawnY].vitality =
                                         vitality / 2;
                            grid[spawnX][spawnY].scentTraces =
102;
                            vitality = vitality / 2;
                            grid[spawnX][spawnY].Locked = false;
                            update();
                        }
```

```
                    }
                }
// If neither prey nor predators are around, the Herbie
// moves in a random direction, assuming there's anyplace
// to go. But first, one more chance to spawn.
                else
                {
// Normally, a Herbie will only spawn after eating. But it
// is technically possible for a Herbie to become
// surrounded while having more than 400 life. So, we need
// to repeat the spawning process here.
                    if (vitality >= 400)
                    {
                        int spawnX = gridX;
                        int spawnY = gridY;
                        for (int i = -1; i < 2; i++)
                        {
                            for (int j = -1; j < 2; j++)
                            {
                                if (((gridY + j) >= 0) &&
                                    ((gridX + i) >= 0) &&
                                    ((gridY + j) < ySize) &&
                                    ((gridX + i) < xSize) &&
                                    ((j != 0) || (i != 0)))
                                {
                                    if ((grid[gridX + i][gridY
                                    + j].creatureType ==
                                    emptie) && !grid[gridX +
                                    i][gridY + j].Locked)
                                    {
                                        spawnX = gridX + i;
                                        spawnY = gridY + j;
                                    }
                                }
                            }
                        }
                        if ((spawnX != gridX) || (spawnY !=
                            gridY) &&
!grid[spawnX][spawnY].Locked)
                        {
                            grid[spawnX][spawnY].Locked = true;
                            grid[spawnX][spawnY].creatureType =
                                herbie;
                            grid[spawnX][spawnY].vitality =
                                      vitality / 2;
                            grid[spawnX][spawnY].scentTraces =
102;

                            vitality = vitality / 2;
```

```
                        grid[spawnX][spawnY].Locked = false;
                        update();
                    }
                }
// Now let's move it!
                double temp = Math.random();
                if ((temp < 0.125) && ((gridX - 1) >=
                    0) && ((gridY - 1) >= 0) &&
                    (grid[gridX - 1][gridY -
                    1].creatureType == emptie) &&
                    !grid[gridX - 1][gridY -
                    1].Locked)
                {
                    grid[gridX - 1][gridY -
                        1].Locked = true;
                    grid[gridX - 1][gridY -
                        1].creatureType = herbie;
                    grid[gridX - 1][gridY -
                        1].vitality = vitality;
                    grid[gridX - 1][gridY -
                        1].scentTraces = 102;
                    creatureType = emptie;
                    vitality = 0;
                    grid[gridX - 1][gridY -
                        1].Locked = false;
                    update();
                }
                else if ((temp < 0.25) && ((gridY - 1)
                         >= 0) && (grid[gridX][gridY -
                         1].creatureType == emptie) &&
                         !grid[gridX][gridY - 1].Locked)
                {
                    grid[gridX][gridY -
                        1].Locked = true;
                    grid[gridX][gridY -
                        1].creatureType = herbie;
                    grid[gridX][gridY -
                        1].vitality = vitality;
                    grid[gridX][gridY -
                        1].scentTraces = 102;
                    creatureType = emptie;
                    vitality = 0;
                    grid[gridX][gridY -
                        1].Locked = false;
                    update();
                }
                else if ((temp < 0.375) && ((gridX + 1)
                         < xSize) && ((gridY - 1) >= 0)
```

```
        && (grid[gridX + 1][gridY -
        1].creatureType == emptie) &&
        !grid[gridX + 1][gridY -
        1].Locked)
{
    grid[gridX + 1][gridY -
        1].Locked = true;
    grid[gridX + 1][gridY -
        1].creatureType = herbie;
    grid[gridX + 1][gridY -
        1].vitality = vitality;
    grid[gridX + 1][gridY -
        1].scentTraces = 102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX + 1][gridY -
        1].Locked = false;
    update();
}
else if ((temp < 0.5) && ((gridX - 1)
        >= 0) && (grid[gridX -
        1][gridY].creatureType ==
        emptie) && !grid[gridX -
        1][gridY].Locked)
{
    grid[gridX -
        1][gridY].Locked = true;
    grid[gridX -
        1][gridY].creatureType
        = herbie;
    grid[gridX -
        1][gridY].vitality = vitality;
    grid[gridX -
        1][gridY].scentTraces = 102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX -
        1][gridY].Locked = false;
    update();
}
else if ((temp < 0.625) && ((gridX + 1)
        < xSize) && (grid[gridX +
        1][gridY].creatureType ==
        emptie) && !grid[gridX +
        1][gridY].Locked)
{
    grid[gridX +
        1][gridY].Locked = true;
```

```
    grid[gridX +
        1][gridY].creatureType
        = herbie;
    grid[gridX + 1][gridY].vitality
        = vitality;
    grid[gridX +
        1][gridY].scentTraces = 102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX +
        1][gridY].Locked = false;
    update();
}
else if ((temp < 0.75) && ((gridX - 1)
         >= 0) && ((gridY + 1) < ySize)
         && (grid[gridX - 1][gridY +
         1].creatureType == emptie) &&
         !grid[gridX - 1][gridY +
         1].Locked)
{
    grid[gridX - 1][gridY +
        1].Locked = true;
    grid[gridX - 1][gridY +
        1].creatureType = herbie;
    grid[gridX - 1][gridY +
        1].vitality = vitality;
    grid[gridX - 1][gridY +
        1].scentTraces = 102;
    creatureType = emptie;
    vitality = 0;
    grid[gridX - 1][gridY +
        1].Locked = false;
    update();
}
else if ((temp < 0.875) && ((gridY + 1)
         < ySize) && (grid[gridX][gridY
         + 1].creatureType == emptie)
         && !grid[gridX][gridY +
         1].Locked)
{
    grid[gridX][gridY +
        1].Locked = true;
    grid[gridX][gridY +
        1].creatureType = herbie;
    grid[gridX][gridY +
        1].vitality = vitality;
    grid[gridX][gridY +
        1].scentTraces = 102;
```

```
                        creatureType = emptie;
                        vitality = 0;
                        grid[gridX][gridY +
                            1].Locked = false;
                        update();
                    }
                    else if ((temp <= 1.0) && ((gridX + 1)
                             < xSize) && ((gridY + 1) <
                             ySize) && (grid[gridX +
                             1][gridY + 1].creatureType ==
                             emptie) && !grid[gridX +
                             1][gridY + 1].Locked)
                    {
                        grid[gridX + 1][gridY +
                            1].Locked = true;
                        grid[gridX + 1][gridY +
                            1].creatureType = herbie;
                        grid[gridX + 1][gridY +
                            1].vitality = vitality;
                        grid[gridX + 1][gridY +
                            1].scentTraces = 102;
                        creatureType = emptie;
                        vitality = 0;
                        grid[gridX + 1][gridY +
                            1].Locked = false;
                        update();
                    }
                }
            }
        }
// Next, we handle the case of the omnie:
        else if (creatureType == omnie)
        {
            scentTraces = 103;
            vitality = vitality - 10;
// If the Omnie snuffs it, wipe it out.
            if (vitality < 50)
            {
                creatureType = emptie;
                vitality = 0;
                soilNutrients = soilNutrients + 233;
                update();
            }
            else
            {
// Though they prefer to eat meat, Omnies eat just about
// anything but other Omnies. Look in the immediate area
// for Herbies and Carnies. If neither are available, look
```

```
// for Veggies.
                nextToPrey = false;
                for (int i = -1; i < 2; i++)
                {
                    for (int j = -1; j < 2; j++)
                    {
                        if (((gridX + i) >= 0) && ((gridX
                            + i) < xSize) && ((gridY + j)
                            >= 0) && ((gridY + j) < ySize)
                            && ((grid[gridX + i][gridY +
                            j].creatureType == herbie) ||
                            (grid[gridX + i][gridY +
                            j].creatureType == carnie)))
                        {
                            nextToPrey = true;
                            preyLocX = i;
                            preyLocY = j;
                        }
                    }
                }
                if (!nextToPrey)
                {
                    for (int i = -1; i < 2; i++)
                    {
                        for (int j = -1; j < 2; j++)
                        {
                            if (((gridX + i) >= 0) &&
                                ((gridX + i) < xSize) &&
                                ((gridY + j) >= 0) &&
                                ((gridY + j) < ySize) &&
                                (grid[gridX + i][gridY +
                                j].creatureType == veggie))
                            {
                                nextToPrey = true;
                                preyLocX = i;
                                preyLocY = j;
                            }
                        }
                    }
                }
// The Omnie tears a hunk out of its prey. The bigger the
// Omnie, the bigger the hunk.
                biteSize = vitality / 15;
                if (nextToPrey)
                {
                    if (grid[gridX + preyLocX][gridY +
                        preyLocY].vitality < biteSize)
                    {
```

```
                            vitality = vitality +
                                grid[gridX + preyLocX][gridY +
                                preyLocY].vitality;
                            grid[gridX + preyLocX][gridY +
                                preyLocY].vitality = 0;
                        }
                        else
                        {
                            vitality = vitality + biteSize;
                            grid[gridX + preyLocX][gridY +
                                preyLocY].vitality =
                                grid[gridX + preyLocX][gridY +
                                preyLocY].vitality - biteSize;
                        }
// Once an Omnie bloats to the proper size, it spawns.
                        if (vitality >= 600)
                        {
                            int spawnX = gridX;
                            int spawnY = gridY;
                            for (int i = -1; i < 2; i++)
                            {
                                for (int j = -1; j < 2; j++)
                                {
                                    if (((gridY + j) >= 0) &&
                                        ((gridX + i) >= 0) &&
                                        ((gridY + j) < ySize) &&
                                        ((gridX + i) < xSize) &&
                                        ((j != 0) || (i != 0)))
                                    {
                                        if ((grid[gridX + i][gridY
                                        + j].creatureType ==
                                        emptie) && !grid[gridX +
                                        i][gridY + j].Locked)
                                        {
                                            spawnX = gridX + i;
                                            spawnY = gridY + j;
                                        }
                                    }
                                }
                            }
                            if ((spawnX != gridX) || (spawnY !=
                                gridY) &&
!grid[spawnX][spawnY].Locked)
                            {
                                grid[spawnX][spawnY].Locked = true;
                                grid[spawnX][spawnY].creatureType =
                                    omnie;
                                grid[spawnX][spawnY].vitality =
```

```
                                        vitality / 2;
                                grid[spawnX][spawnY].scentTraces =
                                    103;
                                vitality = vitality / 2;
                                grid[spawnX][spawnY].Locked = false;
                                update();
                            }
                        }
                    }
// The Omnie takes a whiff. If it smells any Herbies or
// Carnies nearby, it heads towards the strongest scent.
// Otherwise, it moves in some random direction.
                    else
                    {
// Like the Herbie, the Omnie might need to spawn before
// it begins moving.
                        if (vitality >= 600)
                        {
                            int spawnX = gridX;
                            int spawnY = gridY;
                            for (int i = -1; i < 2; i++)
                            {
                                for (int j = -1; j < 2; j++)
                                {
                                    if (((gridY + j) >= 0) &&
                                        ((gridX + i) >= 0) &&
                                        ((gridY + j) < ySize) &&
                                        ((gridX + i) < xSize) &&
                                        ((j != 0) || (i != 0)))
                                    {
                                        if ((grid[gridX + i][gridY
                                        + j].creatureType ==
                                        emptie) && !grid[gridX +
                                        i][gridY + j].Locked)
                                        {
                                            spawnX = gridX + i;
                                            spawnY = gridY + j;
                                        }
                                    }
                                }
                            }
                            if ((spawnX != gridX) || (spawnY !=
                                gridY) &&
!grid[spawnX][spawnY].Locked)
                            {
                                grid[spawnX][spawnY].Locked = true;
                                grid[spawnX][spawnY].creatureType =
                                    omnie;
```

```
                                    grid[spawnX][spawnY].vitality =
                                                 vitality / 2;
                                    grid[spawnX][spawnY].scentTraces =
                                        103;
                                    vitality = vitality / 2;
                                    grid[spawnX][spawnY].Locked = false;
                                    update();
                                }
                            }
// Now let's move it!
                            preyLocX = gridX;
                            preyLocY = gridY;
                            for (int i = -1; i < 2; i++)
                            {
                                for (int j = -1; j < 2; j++)
                                {
                                    if (((gridX + i) >= 0) &&
                                        ((gridY + j) >= 0) &&
                                        ((gridX + i) < xSize) &&
                                        ((gridY + j) < ySize) &&
                                        (grid[gridX + i][gridY +
                                        j].scentTraces > trail) &&
                                        ((grid[gridX + i][gridY +
                                        j].scentTraces % 5 == 2) ||
                                        (grid[gridX + i][gridY +
                                        j].scentTraces % 5 == 4))
                                        && (grid[gridX + i][gridY +
                                        j].creatureType == emptie)
                                        && !grid[gridX + i][gridY +
                                        j].Locked)
                                    {
                                        trail = grid[gridX +
                                            i][gridY + j].scentTraces;
                                        preyLocX = gridX + i;
                                        preyLocY = gridY + j;
                                    }
                                }
                            }
                            if (((preyLocX != gridX) || (preyLocY
                                != gridY)) &&
                                !grid[preyLocX][preyLocY].Locked)
                            {
                                grid[preyLocX][preyLocY].Locked
                                    = true;
                                grid[preyLocX][preyLocY].creatureType
                                    = omnie;
                                grid[preyLocX][preyLocY].vitality
                                    = vitality;
```

```
                        grid[preyLocX][preyLocY].scentTraces
                            = 103;
                        creatureType = emptie;
                        vitality = 0;
                        grid[preyLocX][preyLocY].Locked
                            = false;
                        update();
                    }
// If there are no Veggies, Carnies, or Herbies, or even
// the slightest scent of Carnie or Herbie, the Omnie sets
// off in a random direction.
                    else
                    {
                        double temp = Math.random();
                        if ((temp < 0.125) && ((gridX - 1)
                            >= 0) && ((gridY - 1) >= 0) &&
                            (grid[gridX - 1][gridY -
                            1].creatureType == emptie) &&
                            !grid[gridX - 1][gridY -
                            1].Locked)
                        {
                            grid[gridX - 1][gridY -
                                1].Locked = true;
                            grid[gridX - 1][gridY -
                                1].creatureType = omnie;
                            grid[gridX - 1][gridY -
                                1].vitality = vitality;
                            grid[gridX - 1][gridY -
                                1].scentTraces = 103;
                            creatureType = emptie;
                            vitality = 0;
                            grid[gridX - 1][gridY -
                                1].Locked = false;
                            update();
                        }
                        else if ((temp < 0.25) && ((gridY
                                 - 1) >= 0) &&
                                 (grid[gridX][gridY -
                                 1].creatureType == emptie)
                                 && !grid[gridX][gridY -
                                 1].Locked)
                        {
                            grid[gridX][gridY -
                                1].Locked = true;
                            grid[gridX][gridY -
                                1].creatureType = omnie;
                            grid[gridX][gridY -
                                1].vitality = vitality;
```

```
        grid[gridX][gridY -
            1].scentTraces = 103;
        creatureType = emptie;
        vitality = 0;
        grid[gridX][gridY -
            1].Locked = false;
        update();
    }
    else if ((temp < 0.375) && ((gridX
            + 1) < xSize) && ((gridY
            - 1) >= 0) &&
            (grid[gridX + 1][gridY -
            1].creatureType == emptie)
            && !grid[gridX + 1][gridY
            - 1].Locked)
    {
        grid[gridX + 1][gridY -
            1].Locked = true;
        grid[gridX + 1][gridY -
            1].creatureType = omnie;
        grid[gridX + 1][gridY -
            1].vitality = vitality;
        grid[gridX + 1][gridY -
            1].scentTraces = 103;
        creatureType = emptie;
        vitality = 0;
        grid[gridX + 1][gridY -
            1].Locked = false;
        update();
    }
    else if ((temp < 0.5) && ((gridX -
            1) >= 0) && (grid[gridX -
            1][gridY].creatureType ==
            emptie) && !grid[gridX -
            1][gridY].Locked)
    {
        grid[gridX -
            1][gridY].Locked = true;
        grid[gridX -
            1][gridY].creatureType
            = omnie;
        grid[gridX -
            1][gridY].vitality =
            vitality;
        grid[gridX -
            1][gridY].scentTraces = 103;
        creatureType = emptie;
        vitality = 0;
```

```
    grid[gridX -
        1][gridY].Locked = false;
    update();
}
else if ((temp < 0.625) && ((gridX
         + 1) < xSize) &&
         (grid[gridX +
         1][gridY].creatureType ==
         emptie) && !grid[gridX +
         1][gridY].Locked)
{
    grid[gridX +
        1][gridY].Locked = true;
    grid[gridX +
        1][gridY].creatureType
        = omnie;
    grid[gridX + 1][gridY].vitality
        = vitality;
    grid[gridX +
        1][gridY].scentTraces = 103;
    creatureType = emptie;
    vitality = 0;
   grid[gridX +
        1][gridY].Locked = false;
    update();
}
else if ((temp < 0.75) && ((gridX
         - 1) >= 0) && ((gridY + 1)
         < ySize) && (grid[gridX -
         1][gridY +
         1].creatureType == emptie)
         && !grid[gridX - 1][gridY
         + 1].Locked)
{
    grid[gridX - 1][gridY +
        1].Locked = true;
    grid[gridX - 1][gridY +
        1].creatureType = omnie;
    grid[gridX - 1][gridY +
        1].vitality = vitality;
    grid[gridX - 1][gridY +
        1].scentTraces = 103;
    creatureType = emptie;
    vitality = 0;
    grid[gridX - 1][gridY +
        1].Locked = false;
    update();
}
```

```
                    else if ((temp < 0.875) && ((gridY
                            + 1) < ySize) &&
                            (grid[gridX][gridY +
                            1].creatureType == emptie)
                            && !grid[gridX][gridY +
                            1].Locked)
                    {
                        grid[gridX][gridY +
                            1].Locked = true;
                        grid[gridX][gridY +
                            1].creatureType = omnie;
                        grid[gridX][gridY +
                            1].vitality = vitality;
                        grid[gridX][gridY +
                            1].scentTraces = 103;
                        creatureType = emptie;
                        vitality = 0;
                        grid[gridX][gridY +
                            1].Locked = false;
                        update();
                    }
                    else if ((temp <= 1.0) && ((gridX
                            + 1) < xSize) && ((gridY
                            + 1) < ySize) &&
                            (grid[gridX + 1][gridY +
                            1].creatureType == emptie)
                            && !grid[gridX + 1][gridY
                            + 1].Locked)
                    {
                        grid[gridX + 1][gridY +
                            1].Locked = true;
                        grid[gridX + 1][gridY +
                            1].creatureType = omnie;
                        grid[gridX + 1][gridY +
                            1].vitality = vitality;
                        grid[gridX + 1][gridY +
                            1].scentTraces = 103;
                        creatureType = emptie;
                        vitality = 0;
                        grid[gridX + 1][gridY +
                            1].Locked = false;
                        update();
                    }
                }
            }
        }
    }
// And, finally, the code for the common carnie:
```

```
        else if (creatureType == carnie)
        {
            scentTraces = 104;
            vitality = vitality - 10;
// Has the Carnie bought the farm? If so, kill it and make
// the square Emptie. Skip the rest.
            if (vitality < 50)
            {
                creatureType = emptie;
                vitality = 0;
                soilNutrients = soilNutrients + 200;
                update();
            }
            else
            {
// First, the Carnie sniffs around for food, that is,
// Herbies or Omnies. Carnies aren't cannibals.
                nextToPrey = false;
                for (int i = -1; i < 2; i++)
                {
                    for (int j = -1; j < 2; j++)
                    {
                        if (((gridX + i) >= 0) && ((gridX
                            + i) < xSize) && ((gridY + j)
                            >= 0) && ((gridY + j) < ySize)
                            && ((grid[gridX + i][gridY +
                            j].creatureType == herbie) ||
                            grid[gridX + i][gridY +
                            j].creatureType == omnie))
                        {
                            nextToPrey = true;
                            preyLocX = i;
                            preyLocY = j;
                        }
                    }
                }
// Dinnertime! The carnie snarfs some grub. The amount of
// life taken with each bite depends on the size of the
// Carnie.
                biteSize = vitality / 10;
                if (nextToPrey)
                {
                    if (grid[gridX + preyLocX][gridY +
                        preyLocY].vitality < biteSize)
                    {
                        vitality = vitality +
                            grid[gridX + preyLocX][gridY +
                            preyLocY].vitality;
```

```
                        grid[gridX + preyLocX][gridY +
                            preyLocY].vitality = 0;
                    }
                    else
                    {
                        vitality = vitality + biteSize;
                        grid[gridX + preyLocX][gridY +
                            preyLocY].vitality =
                            grid[gridX + preyLocX][gridY +
                            preyLocY].vitality - biteSize;
                    }
// If the Carnie is massive enough, it spawns.
                    if (vitality >= 800)
                    {
                        int spawnX = gridX;
                        int spawnY = gridY;
                        for (int i = -1; i < 2; i++)
                        {
                            for (int j = -1; j < 2; j++)
                            {
                                if (((gridY + j) >= 0) &&
                                    ((gridX + i) >= 0) &&
                                    ((gridY + j) < ySize) &&
                                    ((gridX + i) < xSize) &&
                                    ((j != 0) || (i != 0)))
                                {
                                    if ((grid[gridX + i][gridY
                                    + j].creatureType ==
                                    emptie) && !grid[gridX +
                                    i][gridY + j].Locked)
                                    {
                                        spawnX = gridX + i;
                                        spawnY = gridY + j;
                                    }
                                }
                            }
                        }
                        if ((spawnX != gridX) || (spawnY !=
                            gridY) &&
!grid[spawnX][spawnY].Locked)
                        {
                            grid[spawnX][spawnY].Locked = true;
                            grid[spawnX][spawnY].creatureType =
                                carnie;
                            grid[spawnX][spawnY].vitality =
                                          vitality / 2;
                            grid[spawnX][spawnY].scentTraces =
                                104;
```

```
                                        vitality = vitality / 2;
                                        grid[spawnX][spawnY].Locked = false;
                                        update();
                                    }
                                }
                            }
// If no prey is nearby, the Carnie sniffs at the scent
// traces around its square to find where the nearest
// food might be.
                            else
                            {
// Once more, give the Carnie a chance to spawn if it
// needs to.
                                if (vitality >= 800)
                                {
                                    int spawnX = gridX;
                                    int spawnY = gridY;
                                    for (int i = -1; i < 2; i++)
                                    {
                                        for (int j = -1; j < 2; j++)
                                        {
                                            if (((gridY + j) >= 0) &&
                                                ((gridX + i) >= 0) &&
                                                ((gridY + j) < ySize) &&
                                                ((gridX + i) < xSize) &&
                                                ((j != 0) || (i != 0)))
                                            {
                                                if ((grid[gridX + i][gridY
                                                + j].creatureType ==
                                                emptie) && !grid[gridX +
                                                i][gridY + j].Locked)
                                                {
                                                    spawnX = gridX + i;
                                                    spawnY = gridY + j;
                                                }
                                            }
                                        }
                                    }
                                    if ((spawnX != gridX) || (spawnY !=
                                        gridY) &&
!grid[spawnX][spawnY].Locked)
                                    {
                                        grid[spawnX][spawnY].Locked = true;
                                        grid[spawnX][spawnY].creatureType =
                                            carnie;
                                        grid[spawnX][spawnY].vitality =
                                                    vitality / 2;
                                        grid[spawnX][spawnY].scentTraces =
```

```
                    104;
                vitality = vitality / 2;
                grid[spawnX][spawnY].Locked = false;
                update();
            }
        }
// Now let's move it!
        preyLocX = gridX;
        preyLocY = gridY;
        for (int i = -1; i < 2; i++)
        {
            for (int j = -1; j < 2; j++)
            {
                if (((gridX + i) >= 0) &&
                    ((gridY + j) >= 0) &&
                    ((gridX + i) < xSize) &&
                    ((gridY + j) < ySize) &&
                    (grid[gridX + i][gridY +
                    j].scentTraces > trail) &&
                    ((grid[gridX + i][gridY +
                    j].scentTraces % 5 == 2) ||
                    (grid[gridX + i][gridY +
                    j].scentTraces % 5 == 3))
                    && (grid[gridX + i][gridY +
                    j].creatureType == emptie)
                    && !grid[gridX + i][gridY +
                    j].Locked)
                {
                    trail = grid[gridX +
                        i][gridY + j].scentTraces;
                    preyLocX = gridX + i;
                    preyLocY = gridY + j;
                }
            }
        }
        if (((preyLocX != gridX) || (preyLocY
            != gridY)) &&
            !grid[preyLocX][preyLocY].Locked)
        {
            grid[preyLocX][preyLocY].Locked
                = true;
            grid[preyLocX][preyLocY].creatureType
                = carnie;
            grid[preyLocX][preyLocY].vitality
                = vitality;
            grid[preyLocX][preyLocY].scentTraces
                = 104;
            creatureType = emptie;
```

```
                                vitality = 0;
                                grid[preyLocX][preyLocY].Locked
                                    = false;
                                update();
                            }
// If the Carnie can't smell any prey, it wanders in a
// random direction, hoping to find a scent.
                            else
                            {
                                double temp = Math.random();
                                if ((temp < 0.125) && ((gridX - 1)
                                    >= 0) && ((gridY - 1) >= 0) &&
                                    (grid[gridX - 1][gridY -
                                    1].creatureType == emptie) &&
                                    !grid[gridX - 1][gridY -
                                    1].Locked)
                                {
                                    grid[gridX - 1][gridY -
                                        1].Locked = true;
                                    grid[gridX - 1][gridY -
                                        1].creatureType = carnie;
                                    grid[gridX - 1][gridY -
                                        1].vitality = vitality;
                                    grid[gridX - 1][gridY -
                                        1].scentTraces = 104;
                                    creatureType = emptie;
                                    vitality = 0;
                                    grid[gridX - 1][gridY -
                                        1].Locked = false;
                                    update();
                                }
                                else if ((temp < 0.25) && ((gridY
                                         - 1) >= 0) &&
                                         (grid[gridX][gridY -
                                         1].creatureType == emptie)
                                         && !grid[gridX][gridY -
                                         1].Locked)
                                {
                                    grid[gridX][gridY -
                                        1].Locked = true;
                                    grid[gridX][gridY -
                                        1].creatureType = carnie;
                                    grid[gridX][gridY -
                                        1].vitality = vitality;
                                    grid[gridX][gridY -
                                        1].scentTraces = 104;
                                    creatureType = emptie;
                                    vitality = 0;
```

```
    grid[gridX][gridY -
        1].Locked = false;
    update();
}
else if ((temp < 0.375) && ((gridX
         + 1) < xSize) && ((gridY
         - 1) >= 0) &&
         (grid[gridX + 1][gridY -
         1].creatureType == emptie)
         && !grid[gridX + 1][gridY
         - 1].Locked)
{
    grid[gridX + 1][gridY -
        1].Locked = true;
    grid[gridX + 1][gridY -
        1].creatureType = carnie;
    grid[gridX + 1][gridY -
        1].vitality = vitality;
    grid[gridX + 1][gridY -
        1].scentTraces = 104;
    creatureType = emptie;
    vitality = 0;
    grid[gridX + 1][gridY -
        1].Locked = false;
    update();
}
else if ((temp < 0.5) && ((gridX -
         1) >= 0) && (grid[gridX -
         1][gridY].creatureType ==
         emptie) && !grid[gridX -
         1][gridY].Locked)
{
    grid[gridX -
        1][gridY].Locked = true;
    grid[gridX -
        1][gridY].creatureType
        = carnie;
    grid[gridX -
        1][gridY].vitality =
        vitality;
    grid[gridX -
        1][gridY].scentTraces = 104;
    creatureType = emptie;
    vitality = 0;
    grid[gridX -
        1][gridY].Locked = false;
    update();
}
```

```
else if ((temp < 0.625) && ((gridX
         + 1) < xSize) &&
         (grid[gridX +
         1][gridY].creatureType ==
         emptie) && !grid[gridX +
         1][gridY].Locked)
{
    grid[gridX +
        1][gridY].Locked = true;
    grid[gridX +
        1][gridY].creatureType
        = carnie;
    grid[gridX + 1][gridY].vitality
        = vitality;
    grid[gridX +
        1][gridY].scentTraces = 104;
    creatureType = emptie;
    vitality = 0;
   grid[gridX +
        1][gridY].Locked = false;
    update();
}
else if ((temp < 0.75) && ((gridX
         - 1) >= 0) && ((gridY + 1)
         < ySize) && (grid[gridX -
         1][gridY +
         1].creatureType == emptie)
         && !grid[gridX - 1][gridY
         + 1].Locked)
{
    grid[gridX - 1][gridY +
        1].Locked = true;
    grid[gridX - 1][gridY +
        1].creatureType = carnie;
    grid[gridX - 1][gridY +
        1].vitality = vitality;
    grid[gridX - 1][gridY +
        1].scentTraces = 104;
    creatureType = emptie;
    vitality = 0;
    grid[gridX - 1][gridY +
        1].Locked = false;
    update();
}
else if ((temp < 0.875) && ((gridY
         + 1) < ySize) &&
         (grid[gridX][gridY +
         1].creatureType == emptie)
```

```
                                    && !grid[gridX][gridY +
                                    1].Locked)
                            {
                                grid[gridX][gridY +
                                    1].Locked = true;
                                grid[gridX][gridY +
                                    1].creatureType = carnie;
                                grid[gridX][gridY +
                                    1].vitality = vitality;
                                grid[gridX][gridY +
                                    1].scentTraces = 104;
                                creatureType = emptie;
                                vitality = 0;
                                grid[gridX][gridY +
                                    1].Locked = false;
                                update();
                            }
                            else if ((temp <= 1.0) && ((gridX
                                    + 1) < xSize) && ((gridY
                                    + 1) < ySize) &&
                                    (grid[gridX + 1][gridY +
                                    1].creatureType == emptie)
                                    && !grid[gridX + 1][gridY
                                    + 1].Locked)
                            {
                                grid[gridX + 1][gridY +
                                    1].Locked = true;
                                grid[gridX + 1][gridY +
                                    1].creatureType = carnie;
                                grid[gridX + 1][gridY +
                                    1].vitality = vitality;
                                grid[gridX + 1][gridY +
                                    1].scentTraces = 104;
                                creatureType = emptie;
                                vitality = 0;
                                grid[gridX + 1][gridY +
                                    1].Locked = false;
                                update();
                            }
                        }
                    }
                }
            }
            try{sleep(500);}
                catch(InterruptedException e) {}
            }
        }
    }
```

Appendix D: HTML Code

Alife.HTML

```
<html>
<title>Artificial Life</title>
<body bgcolor="00aaff" text="000000" link="890000"
vlink="898989">
<center><H1>And They Called Me Mad at the University...</H1>
<H3>But what do they know of madness?!?<br>Is it madness to yearn
for the
eternal secrets of life?<br>Is it madness to delve into the
unknown?<br>To seek
that which unenlightened fools would dare suggest humankind was
not meant
to know?<br>IS IT?!?<p>
If madness it is, then it is with pride that I call myself a
madman.<br>Those
so-called scholars were fools to deny my work.<br>But I'll show
them!<br>I'll
show them all!</H3>
<H1>For I have created...LIFE!!!</H1>
<a href="lightning.au">Ominous Thunder</a><p>
<center><img src="spectrum.gif"></center>
<H1>Allow me to introduce you to my creations...</H1></center>
Gridcritters come in four varieties: Veggies, Herbies, Omnies,
and Carnies.
Each critter has a certain amount of life (Vitality), and can
absorb a certian
amount of Vitality from its prey, depending on the amount of
Damage it can
deal. Any gridcritter that manages to double its initial Vitality
will spawn a
new creature into an adjacent grid square. Any creature whose
Vitality falls
below 50 will die, and be absorbed into the grid square as a
certain amount of
soil nutrients. To keep any creature from living forever, I've
added
entropy to the system (every creature loses 10 Vitality each
turn,
whatever else it does). And now: the gridcritters.<p>
<center><img src="Veggie.gif"><H2>VEGGIE</H2></center>
<H3>Statistics</H3>
<li>Color: Green
<li>Vitality: 100
<li>Damage: 30 (only to soil nutrients)
<li>Eats: Soil Nutrients<p>
```

```
Ah, the Veggie, possibly the most insidious of my little toys. It
seems so
harmless, completely unable to move under its own power. Yet a
single veggie can
smother an entire grid in moments. The Veggie breeds faster than
any other
creature--a necessity, considering the fact that a Veggie begins
life with a
pitifully small vitality. The Veggie is also the only gridcritter
that is
choosy about where it spawns to. When spawning, a Veggie will
place its
offspring into the neighboring square that has the most Soil
Nutrients.<p>
<center><img src="Herbie.gif"><H2>HERBIE</H2></center>
<H3>Statistics</H3>
<li>Color: Yellow
<li>Vitality: 200
<li>Damage: 20 (5 against predators, no Vitality gained)
<li>Eats: Veggies<p>
A luckless creature, the Herbie. The weakest of the mobile
gridcritters,
Herbies run from predators when they can, and fight back with a
pitifully
inadequate attack when they cannot. Fortunately, Herbies breed
fairly rapidly,
and can easily overwhelm the defenseless Veggies that they prey
upon. Utterly
blind, Herbies will wander aimlessly if not next to food, and in
time will
starve to death.<p>
<center><img src="Omnie.gif"><H2>OMNIE</H2></center>
<H3>Statistics</H3>
<li>Color: Blue
<li>Vitality: 300
<li>Damage: 1/15 X Vitality
<li>Eats: Veggies, Herbies, Carnies<p>
A very adaptable creature, the Omnie can eat just about anything
but each
other, though they prefer to eat meat, and will attack Herbies
and Carnies
before eating Veggies, if there are Herbies or Carnies in the
immediate area.
In the absence of food, Omnies hunt for Herbies and Carnies by
smell, following
the scent traces left behind by these creatures in the grid
squares they have
occupied.<p>
```

```
<center><img src="Carnie.gif"><H2>CARNIE</H2></center>
<H3>Statistics</H3>
<li>Color: Red
<li>Vitality: 400
<li>Damage: 1/10 X Vitality
<li>Eats: Herbies, Omnies<p>
The bad boys of the grid, Carnies are veritable living tanks. If
their large
starting Vitality wasn't enough, Carnies are also loaded for
bear. A well-fed
Carnie can deal as much as 79 damage per strike, which is enough
to make short
work of just about anything. In the absence of food, Carnies,
like Omnies, hunt
by smell. Their dependence on meat is their only shortcoming, as
a few Carnies
are enough to sweep the board of every non-Veggie in short order,
at which point
the Veggies grow out of control and smother them. No one said
they were bright.
Fortunately, these suckers breed very slowly.<p>
<center><img src="spectrum.gif"></center>
<center><H1>Observe the gridcritters in action...</H1>
<H2><a href="kudzu.html">Scenerio One: The Kudzu
Factor</a></H2><br>
Observe the unimaginable horror of Veggies unleashed!<br>A
handful of Veggies
quickly overwhelm a 10x10 grid!<br>Oh, the humanity!<br>THE
HUMANITY!!!
<H2><a href="eden.html">Scenerio Two: Eden</a></H2><br>
An idyllic paradise...or is it?<br>Three Herbies are allowed to
graze unchecked.
<br>The Veggies initially seem to conquer the 5x5 area, but the
Herbies will not be
denied...
<H2><a href="bambi.html">Scenerio Three: Bambi Vs
Godzilla</a></H2><br>
A lone Herbie flees for its life as a ravenous Carnie chases it
across a 10x10
grid.<br>(Parental discretion advised)
<H2><a href="hunt.html">Scenerio Four: The Great
Hunt</a></H2><br>
Just how tough are Carnies?<br>Their muscle is put to the test,
as six Omnies
square off against two Carnies on a 5x5 grid.
<H2><a href="pit.html">Scenerio Five: Pit Fighter</a></H2><br>
It's Veggie vs Herbie vs Omnie vs Carnie in a one on one on one
on one fight to
```

```
the finish!<br>Who will emerge triumphant from this 5x5 field of
blood?<br>(Hint:
Probably not the Herbie.)
<H2><a href="genesis.html">Scenerio six: Random
Genesis</a></H2><br>
The true goal of my work, Random Genesis generates a random
setting of
creatures on a 10x10 grid.<br>Each grid square has<br>a 20%
chance of being a
Veggie,<br>a 15% chance of being a Herbie,<br>a 10% chance of
being an Omnie,
and<br>a 5% chance of being a Carnie.<br>Start 'er up, and watch
the fun!<p>
<img src="spectrum.gif">
<H4>Return to <a href="../AaronZ.html">Aaron's</a>/<a
href="../DarkId.html">Dark Id's</a> homepage.</center>
</body>
</html>
```

Kudzu.html

```
<html>
<title>THE KUDZU FACTOR</title>
<body bgcolor="00aaff" text="000000">
<center>
<applet code="LifeGrid.class" width=500 height=500>
<param name=scenerio value=0>
<param name=gridsize value=10>
</applet>
<H1>INSTRUCTIONS<H1>
<H3>Click on the grid once to start.<p>
Click on the grid a second time to stop.<p>
Click on the grid a third time to reset.</H3>
</center>
</body>
</html>
```

Eden.html

```
<html>
<title>EDEN</title>
<body bgcolor="00aaff" text="000000">
<center>
<applet code="LifeGrid.class" width=250 height=250>
<param name=scenerio value=1>
<param name=gridsize value=5>
</applet>
<H1>INSTRUCTIONS<H1>
<H3>Click on the grid once to start.<p>
Click on the grid a second time to stop.<p>
Click on the grid a third time to reset.</H3>
</center>
</body>
</html>
```

Bambi.html

```
<html>
<title>BAMBI VS GODZILLA</title>
<body bgcolor="00aaff" text="000000">
<center>
<applet code="LifeGrid.class" width=500 height=500>
<param name=scenerio value=2>
<param name=gridsize value=10>
</applet>
<H1>INSTRUCTIONS<H1>
<H3>Click on the grid once to start.<p>
Click on the grid a second time to stop.<p>
Click on the grid a third time to reset.</H3>
</center>
</body>
</html>
```

Hunt.html

```
<html>
<title>THE GREAT HUNT</title>
<body bgcolor="00aaff" text="000000">
<center>
<applet code="LifeGrid.class" width=250 height=250>
<param name=scenerio value=3>
<param name=gridsize value=5>
</applet>
<H1>INSTRUCTIONS<H1>
<H3>Click on the grid once to start.<p>
Click on the grid a second time to stop.<p>
Click on the grid a third time to reset.</H3>
</center>
</body>
</html>
```

Pit.html

```
<html>
<title>PIT FIGHTER</title>
<body bgcolor="00aaff" text="000000">
<center>
<applet code="LifeGrid.class" width=250 height=250>
<param name=scenerio value=4>
<param name=gridsize value=5>
</applet>
<H1>INSTRUCTIONS<H1>
<H3>Click on the grid once to start.<p>
Click on the grid a second time to stop.<p>
Click on the grid a third time to reset.</H3>
</center>
</body>
</html>
```

Genesis.html

```
<html>
<title>RANDOM GENESIS</title>
<body bgcolor="00aaff" text="000000">
<center>
<applet code="LifeGrid.class" width=500 height=500>
<param name=scenerio value=5>
<param name=gridsize value=10>
</applet>
<H1>INSTRUCTIONS<H1>
<H3>Click on the grid once to start.<p>
Click on the grid a second time to stop.<p>
Click on the grid a third time to reset.</H3>
</center>
</body>
</html>
```