The University of Birmingham

School of Computer Science

MSc in Natural Computation

Mini-project

Studying the Emergence of Multicellular Organisms in Artificial Life

J. Rothermich

Supervisor: J. Miller

April 2002

ABSTRACT AND KEYWORDS

Traditional systems building often consists of using a top-down approach. That is, conceptualizing the overall organization for how a system should operate and then breaking it down so that it fits into logical pieces. Each of these pieces is then divided again and again until manageable units are developed in detail. There is some belief that in order to build very complex systems, such as applications that are able to learn and repair themselves, a bottom-up approach must be taken to systems design. System can be built with small simple components that together organize to function as a complex system. This is similar to how nature creates complex organisms out of cells. Each cell is a relatively simple unit with a minimal instruction set, which is capable of organizing with other cells to create a large complex organism.

This paper studies the emergent properties of single celled organisms and how in an evolutionary simulation, multicelled organisms might originate. This study may give insights into emergent systems, and will hopefully aid in understanding how these systems can be harnessed and created. Experiments are performed in an artificial life simulation on a computer using Cartesian Genetic Programming. These experiments produced populations of cells, which exhibited novel behavior. Sometimes, interesting group behaviors emerged which showed properties of multicellularity.

Keywords

Multicellularity; Genetic Programming; Artificial Life; Emergent Behavior.

TABLE OF CONTENTS:

LIST OF FIGURES:

1 Introduction

The study of emergent decentralized systems is currently a hot topic for many interest groups (Johnson, 2001). An understanding of the properties of emergence would give humans insight into biology, economics, evolution, and computer science. This might help us not only to understand these systems but also to harness and create them.

The origin of multicellular organisms in nature is one of the important mysteries currently being investigated in science (Bonner, 2000). Discovering how single-celled organisms could have evolved into complex beings could be of great value in the understanding of emergence. Insight into this area might help biologists better understand developmental biology. It might also help computer scientists learn how to build bottom-up decentralized systems.

This research project creates an Artificial Life environment for the simulation and exploration of emergent behavior and multicellularity. A form of genetic programming called Cartesian Genetic Programming is used in order to evolve cell behavior. A simulation of cellular slime mold aggregation is created in addition. The implementation described will provide a solid base for future experiments in this area.

According to Bonner, "there are three ways of looking at the origin of multicellular development: the straightforward, descriptive biological way; the way of molecular biology; and the use of mathematical models to search for insights." In the same manner that he proposes the use of models, the experiments performed in this research attempt to give insights to this study.

The organization of this paper is as follows: first a background is given in the topic of emergent systems and computation. A history of the multicellular development in nature is also described. The next section (2.3) gives an overview to the field of Artificial Life (ALife) and discusses previous work relevant to this research. Section 3 describes the experiments and has two main parts. The first part describes the non-evolving portion of the experiments including the slime mold simulation. After, a description of the evolving experiments is made and explains the use of Cartesian Genetic Programming. Finally, the last section forms conclusions on the research and discusses possible improvements.

2 Background

2.1 Multicellularity in Computation

2.1.1 Increasingly Paralleled and Decentralized Computing

As the limits of traditional systems building begin to be reached, a new paradigm of computing must be realized in order for continued progress to be possible (Bentley, 2001). Many say that the next wave of computing will be decentralized and focus on emergent behavior (Ray, 1994; Johnson, 2001, Bentley, 2001). This form of computation is becoming increasingly relevant as networking and parallel processing hardware are becoming more powerful and more readily available. One problem being confronted now is that methodologies and logic for building traditional programs do not work for building decentralized systems.

The majority of computer applications running today have been written using serial processing which is often called the 'von Neumann style' architecture (Mitchell, 1996). A program has a set of tasks, which are executed in a specific order. Programs are often decomposed into functions or objects, but the logic of the program remains serial. Some types of algorithms can be easily converted into parallel tasks. One example is splitting up a large amount of data that needs processing. As long as the results of processing one part of data do not impact the processing of the others, then it is possible to have several programs working on the sections of data independently. It is when a problem's tasks cannot be easily divided that difficulty is encountered when designing an algorithm. It is very difficult to create multiple programs that collectively work on a single set of data. This is especially true if the there is no central coordinator of the different programs.

Computer systems were first introduced as being heavily centralized. Originally they were very expensive and a company would have a central mainframe that did all of the processing for the company. Some time later, personal computers were introduced which allowed processing to take place at each user's desktop. Even then there were intermediate steps. Some applications would still have all of the logic run on a central computer and the PC's would only process the displaying of the information. Other programs allowed business rules and calculations to be processed on the PC's but all of the data was stored centrally. These two examples are how most business applications operate today. There are some applications that also allow data to be stored locally. This is not just until the central server is synchronized, but the data remains distributed on individual computers. This is much harder to control for a company.

In the same way that enterprise software has gradually changed from centralized processing to decentralized processing of transactions, the same type of movement is happening with the processing of a single transaction. Instead of a single PC or processor performing a transaction, it might be split over several PC's or several independent programs. Each program would take a different role in the transaction.

The human mind has a hard time thinking in terms of decentralized parallel processing (Resnick, 1994; Johnson, 2001). Resnick calls this the 'decentralize mindset'. Most people tend to look for a centralized hierarchy when observing a system. They have a hard time when trying to imagine a process with no leaders or central organization.

There are some applications that are ideal for taking advantage of parallel processing improvements in hardware, processors and networking. Neural Network applications that model the behavior of the brain in order to perform a task are well suited for this. They consist of individual neurons that together produce an output based on the networks inputs. These neurons can all be processed independently from one another as long as the flow of the information is still maintained. Evolutionary Algorithms such as Genetic Algorithms are also ideal for this type of technology. These algorithms take a population of many programs and let them compete against each other for fitness. The fitness trial can often be performed independently on separate environments. Problems that were once too large to be solved using traditional serial computers are now solvable due to the possibility of massively paralleled programs.

2.1.2 Emergent Behavior in Systems

Defining emergence

The term emergence is used in many different fields with just as many usages. The word emergence will be used in this paper to describe a system whose behavior cannot be described by only describing the rules of the system's components. This is similar to the idea that the whole is greater than the sum of its parts. Each part may have a set of possible actions, but the combinations of the parts create a new set of actions that are in some way surprising.

Various notions of the concept of surprise have been discussed in terms of emergence. If a system can be created in a way that the combined behavior of its members is predicted, its behavior is not really emerging. Langton (1989) separates emergence into several categories. He describes a system as non-emergent if its behavior is deducible by inspecting the system's rules. A weakly emergent system is defined as such if the behavior is deducible from the rules with hindsight. He defines strongly emergent systems at those that the behavior can be worked out in theory but it would be prohibitively difficult. Finally, he defines maximally emergent systems as those where behavior is impossible to deduce from the system's specification.

Langton goes on to describe what can emerge. Three possibilities are given. Structural Emergence is the emergence of form such as flocking and gliders in the Game of Life (Berlekamp, Conway and Guy, 1985). Computational Emergence gives the system a way to compute or handle logic, and he describes Functional Emergence as occurring when actions that are functional are beneficial to the members of the system. In these categories, Functional Emergence is a sub-category of Computational Emergence.

Possible criteria for emergence may be that a system is made of simple parts that together build a complex whole. This does not have to be true. It could be that complex parts combine to form something even greater in complexity. So relativity is needed when considering the complexity of emergent systems.

Examples of emergent systems are readily available in nature and societies. The studies of flocking birds and behaviors of ants and their colonies are investigations into some emergent systems in nature. When observing flocking, it is often assumed that there must be a leader of the flock (Johnson, 2001). But these studies have shown that coordination among the members of the flock is what keeps them together. In the studies of ant colonies, the queen ant is shown to have no control over the actions of individual ants. Instead, each ant goes about its tasks using a simple set of rules. As a group they perform complex actions such as collection and sorting of food, piling of dead ants, and feeding the queen ant.

Economics and social systems provide other examples of emergent systems. Studies have been performed to learn how markets and pricing work (Cliff and Bruten, 1997), how cities are formed (Krugman, 1996; Shelling 1978), and even how arms races can be won (Dawkins, 1987). Shelling studied the dynamics of the effects of social classes on neighborhood migration. He showed that patterns of social behavior could emerge from individuals following simple rules. He also discussed how positive feedback in a system such could lead to these behaviors. Each member of a population has thresholds for whether or not they will act, as more people begin to act, then those thresholds are met and new people act. In his book "The Self-Organizing Economy", Krugman (1996) described a model of how businesses in a city would group together in an orderly way based on the competition factors and benefits of proximity.

Why Study Emergence

There are many studies on how to create and observe emergence (Resnick, 1994; Johnson, 2001). A better understanding of emergent behavior and how it originates can lead to understanding biology, economics, and even traffic jams better. If models can be made of decentralized emergent systems, perhaps predictions can be made concerning their behavior. And if predictive models are possible, it might also be feasible to learn how to influence a system to achieve a desired behavior.

Besides the goal of learning about the behavior of complex systems, another reason to study emergence is learning how to actually create them. It was mentioned that humans have a hard time thinking in terms of decentralized systems. It is therefore, hard for humans to create algorithms that work in a decentralized way. Once an understanding of decentralized emergent systems is achieved, perhaps it will be easier to create these systems.

2.1.3 Multicellular Computation

In the study of natural organisms, the question of how multicellularity came into existence is one of the most profound questions being studied today (Bonner, 2000). Knowing the answer to this question would reveal how the existence of complex organisms could have arisen from a world of simple single celled organisms. When the idea of a computer program is associated with the notion of a cell, and a complex decentralized system expressed as the interaction of cells, the question of the origins of multicellularity becomes interesting to the field of Computer Science as well.

A cell can be conceptually thought of as a simple program. It receives input and gives output. The input it receives can be environmental factors such as temperature, light, and surrounding chemicals. It processes these inputs and provides output. Output for a cell might consist of an action such as movement or mitosis, or the release of chemicals into the environment. Although the inputs and outputs might be different for a typical computer program, the concept is really the same.

A population of cells can be thought of as a collection of programs. When the cells act together in some way without a leader, the collection of programs becomes a decentralized system. There are no central controllers telling a cell what to do. Each cell or program is created with an instruction set and acts according to those instructions. Cells do not have to be connected to perform a global task. For example, removing chemical waste from an area is a global task that could be performed by a group of cells. However, this is not necessarily emergent behavior since it can easily be predicted if it is known that a cell absorbs a certain chemical and produces a non-waste chemical in return. The aggregated behavior of the cells is just a massively paralleled implementation of that cells program and the results are predictable.

Multicelled organisms, however, perform tasks that the individual cells could not do in a smaller way on their own. These systems have truly emergent behavior. Learning how a cell and its program can be combined into a multicellular-like program could lead to learning how to build decentralized computer systems with emergent behavior. This method of building a system would be considered a bottom-up approach, as opposed to the usual top-down approach taken in software development.

The object of this research is to explore how multicellular-like functionality can emerge from single-celled programs. This study can hopefully be a small step in the understanding of creating emergent systems. Currently, the process of creating an emergent system is like putting a lot of ingredients into a pot, stirring it and seeing what happens. Once a better understanding of how this process can occur on its own, either in natural life or in artificial life systems, the controlling of the creation process to achieve certain results will be more feasible.

2.2 Multicellularity In Nature

This section provides a background for the research that is discussed later in the paper. In order to learn about the simulation of multicelled organisms in an artificial ecology, it is helpful to first study how multicellularity might have originated in nature. This section does not aim to propose any new ideas on the topic, or to describe it completely. Instead it offers a high level overview to the study and some of the ideas that might help recreate this behavior in simulation.

2.2.1 A Brief History of Multicelled Organisms in Evolution

Nobody knows exactly how multicelled organisms came into existence. There is proof however, that the event occurred at least three times during the course of evolution (Bonner, 2000), indicating that it was not an event that happened by chance.

It took one billion years for the first cells to form. Then after 3 billion years, the first metazoans came into existence. (Taylor and Jefferson, 1995). About 540 million years ago, a huge diversity of multicellular life was created in a relatively short period of time. Much of the diversity apparent in organisms today was created during this period. This time is called the Cambrian Period and its events are often referred to as the Cambrian Explosion or Biology's Big Bang. During the Cambrian Explosions, massive extinctions and creative evolutionary changes occurred. There is much controversy surrounding the events in the Cambrian Period. Research by Evans, Ripperdan, and Kirschvink (1998) has suggested that continental drift during this time created large-scale climatic changes. Possibly the changes in environmental conditions are responsible for the enormous activity in evolution during this period (Kirschvink, Ripperdan, and Evans, 1997).

2.2.2 Possible Explanations for the Origin of Multicelled Organisms

There are many features of a multicelled organism that can seem advantageous to all of the cells cooperating within it. It is in the move from cells that compete to cells that cooperate that the first multicellular organisms may have evolved (Bonner, 2000; Michod and Roze, 1999). By being formed into a cooperative unit, the cells might have increased their collective fitness and increased their chances for survival. Three possible properties that might have been important in the evolution of multicellularity are size, adhesiveness, and specialization.

Size

The benefits of size in the struggle for survival can often be intuitive. One aspect to size in evolution is that there is always the opportunity to be larger than other members of the population. The distinction of being the largest in a population is always at risk since a larger specimen might appear. An example is given by Dawkins (1987) that the trees in a rainforest do not necessarily gain anything from just being tall. However, they do benefit from being taller than the other trees in the forest. They are evolving to become taller in order to compete for the sunlight. Dawkins and Bonner both discuss the negative connotations of discussing size related to evolution in that size is often associated with 'progress'.

Size may give a cell or group of cells several benefits. It is possible that larger organisms can move faster, giving them an advantage at finding food. Cells that are able to use sunlight to produce energy could perhaps produce more energy when light is available. In same way trees compete for sunlight by being tall, cells could have an advantage being larger and being able to monopolize resources. This type of advantage could be applied to sunlight or the absorption of chemicals that are used to produce energy. Size can also be a factor in that larger organisms are less likely to be candidates for predators. Sometimes the prey is just too large to be eaten or swept into the mouth of a feeder. All of these benefits might encourage cells to join together at some point in their existence or remain joined immediately after cell division.

Adhesiveness

The property of adhesion might be an explanation for the development of multicellularity. A cell with an adhesive membrane would tend to stick to other cells. If cells with that acquired adhesive membranes through mutations were more successful than others, that property could have been retained in the evolutionary process.

Adhesion could provide several advantages to cells. Cells with a sticky membrane could to attach to other cells by coming into contact with them or directly after cell division. Size, as discussed above, can be advantageous to cells. A group of attached cells could benefit from being a part of the larger group.

It is possible that adhesion first provided a benefit to cells that adhered to fixed objects. Instead of being swept away with the current or wind, the cells were able to stay in one place. These cells might be able to profit from staying in a fertile environment. They also might be able to benefit from absorbing chemicals or other cells that pass by them. This might be the primary reason for the evolution of adhesiveness. The adherence to other cells could have been a useful long-term side effect.

Specialization

In a complex system, the components often take on specific roles or function. This can be witnessed in a multicellular organism as cell differentiation. Even though each cell has the same DNA, the cells perform a different role depending on where it is in the organism's body. The cells only express certain genes if their role requires it. The chemicals that surround a cell determine its role. A cell found in a human's skin will be surrounded by different chemicals than a cell in a human's heart. When these cells are created, different genes are expressed based on where they are in the body because of the chemicals found in each environment.

Specialization of cells may provide a benefit that helped evolve multicellularity. Cells that are allowed to perform a specific task such as locomotion, energy absorption, or division might be able to perform those roles better than a general-purpose cell. The increased efficiency provided by specialization might give the group of cells a competitive edge.

2.2.3 An Example: Cellular Slime Molds

Cellular Slime Molds (Dictyostelium discoideum) are one of the prime examples in nature available for the study of multicellularity (Bonner, 2000;

Johnson, 2001). They are organisms that bridge the gap between single and multi-celled organisms. The slime mold cells spend most of their life as single celled organisms. Once the cells start to run out of a food source, they begin to aggregate together forming a slug-like creature. In this form, the cells move together as a group towards a new area, which hopefully contains more sources of food. After a new food source is found, the 'slug' forms a fruiting body and the cells return to living a single-celled life. The cycle restarts when food sources again become scarce.

For a long time it was assumed that the cells aggregated under the command of a leader cell (Bonner, 2000). It was thought the leader cell would signal the other cells and organize them into forming the collective organism. It was later discovered by Keller and Segel (1970) in their study of slime mold chemotaxis that the cells had no leader. Once some cells started to run out of their energy supply, they would begin signaling other cells around them. This set off a chain reaction that caused all of the cells in the area to begin signaling. The cells followed the gradient of the signal, a chemical cyclic AMP (cAMP) signal, and would thus all move towards gathering points of highest concentrations. This movement is known as chemotaxis.

The cell-cell interaction shown in signaling with cAMP signaling is a clue to how cells can communicate and form a cooperative group. Their behavior shows an example of how the development of multicelled organisms can occur in nature. The ability to form a slug-like creature allows the cells to move rapidly to a possibly more resource-rich environment.

2.3 Artificial Life

2.3.1 A Brief Introduction to Artificial Life

At the time this paper was written, the web site for the International Society for Artificial Life at http://www.alife.org had a survey posted. The question was "What is ALife". Six possible answers were available as responses in a multiple-choice format. The list of choices were:

a. Life as it could be
b. Life made by man
c. Synthetic biology
d. Recreating Life from scratch
e. Understanding biology
f. A cool scientific topic

So far only 70 votes were cast, but all of the choices had a decent proportion of the vote with (a) and (b) slightly in the lead. One might argue that the survey wasn't quite fair because all of the choices could be argued to be correct. From the perspective of each participant, the correct answer might have been whichever most applied to his or her field or interest. Many visitors to the site might also have been disappointed by a lack of a correct, or relevant, answer for them. The question "What is Artificial Life" might only be correctly addressed by acknowledging that it is a multidisciplinary field. There is interest from biologists, computer scientists, mathematics and physics researchers, economists, linguists (Wheeler, Bullock, et. al, 2002) and others. Each of these parties has different goals that they wish to accomplish through studying Artificial Life (ALife).

Even though there is a broad range of motivations for studying Artificial Life, it is still helpful to state a definition. A first definition is given by Chris Langton, an important researcher in the field and known to be the first person to use the term 'Artificial Life':

> "Artificial Life is a field of study devoted to understanding life by attempting to abstract the fundamental dynamical principles underlying biological phenomena, and recreating these dynamics in other physical media -- such as computers -- making them accessible to new kinds of experimental manipulation and testing. (Langton, 1992)

A second definition is given by T. S. Ray, the creator of the Tierra Simulator:

> "Artificial Life (AL) is the enterprise of understanding biology by constructing biological phenomena out of artificial components, rather than breaking natural life forms down into their component parts. It is the synthetic rather than the reductionist approach." (Ray, 1994)

Ray's definition takes the approach of using ALife as a way to study biology. However, his Tierra Simulator is used to study the possible uses of Artificial Life to help in the building of parallel processing applications in computers. Langton's definition is less specific and is not restricted to the study of biology. He discusses the use of computers but does not make them a prerequisite (Von Neumann's work, which is now considered part of ALife, was done without the use of any computers). He emphasizes the 'dynamical principles underlying biological phenomena'. This phenomena is closely linked with the discussions of emergence in the previous sections of this report. He said, "Emergent behavior is one of the fundamental characteristics of an ALife system". (Langton, 1989)

2.3.2 Previous Work in ALife and Evolving Multicellularity

Although Artificial Life is a relatively new field, there have been a large number of implementations. A simple search on the Internet will return a seemingly endless amount of programs and papers related to the field. It can become difficult to sort through all of the sites to find specific simulations that are of interest, so it was necessary to limit the search. Given the topic of this research, the following areas were investigated to find relevant implementations:

1. Emergent and cellular computation
2. Multicellular development
3. Programming for the control of autonomous agents

Artificial Life and the idea of computing with cells can be traced back to the work of John von Neumann in the 1940's. He worked with Cellular Automata, which can be explained as a collection of cells whose state depends on the states of its neighbors. As a result of each cell's local behavior, global behavior starts to emerge. Cellular Automata was popularized by John Conway's Game of Life (Berlekamp, Conway and Guy, 1985). This program made experiments easy to perform with different patterns resulting in emergent behavior such as 'gliders' that would move across the screen. Grammar-based Lindenmayer Systems (L-Systems) have been used to model plant growth and other forms of development such as neural network development and cell development (Lindenmayer, 1968).

In 1952, Turing created a mathematical model of cell morphogenesis in which cell's interact via chemical substances, or morphogens. He introduced reaction-diffusion systems that help to explain cell differentiation and pattern formation in development (Turing, 1952). The slime mold chemotaxis models created by Keller and Segel (1970) used the reaction-diffusion equations of Turing.

Fleischer and Barr (1992) have created developmental models "for the long range goal to create artificial neural networks to solve problems in perception and control". They have also applied their models of morphogenesis to the creation of computer graphics.

The study of flocking behavior is a well-known implementation of Artificial Life. Reynold's program, Boids (1987), showed how the seemingly complex flocking behavior of birds can be implemented using the simple local rules of separation, alignment and cohesion. This success is useful to computer graphics and has been used in generating computer animation sequences for movies.

The following are some packages and toolkits that have been created to promote the study of emergence and agent-based programs:

StarLogo

StarLogo is the creation of Mitchell Resnick (1994) of the Epistemology and Learning Group in the Media Laboratory at the Massachusetts Institute of Technology. This is a language based on the Logo programming language from the 1960's. Logo was a language that made computer programming accessible to children and non-programmers. The language allowed for the control of 'turtles' that would drag a virtual pen across the screen and create graphics. Resnick extended the idea of this language so that the programs could control multiple turtles at once. The turtles each acted as agents, or autonomous programs, that would interact. StarLogo was used to study the ideas of decentralized systems and the 'decentralized mindset' of people. Resnick spent a lot of time working with children in schools implementing new systems and noting the assumptions and observations of the children. Some of the systems modeled using StarLogo were traffic jams, forest fires, and cellular slime molds.

The slime mold implementation in StarLogo consisted of a population of turtles representing slime mold cells. The cells would move around in the artificial StarLogo world and aggregate as a result of their programs and some randomness. The logic behind the cell's program was very simple. Each cell released a constant amount of a cyclic AMP-like signal. The cells would 'sniff' straight ahead, 45 degrees to the right, and 45 degrees to the left and then move towards the strongest concentration of the signal. The population was initialized with cells created in random locations. After a few time-steps the cells would start to move towards each other, forming something similar to the multicelled organisms that occur naturally with slime molds.

This StarLogo Slime Mold simulation has been recreated in Java as a part of this research. It provided a basis for exploring multicellularity and a good way to test the environment before evolving more complex cells.

Swarm

Members of the Santa Fe Institute in New Mexico have created a development tool called Swarm (Burkhart, 1994). It is described by the Swarm Development Group as:

> "Swarm is a software package for multi-agent simulation of complex systems, originally developed at the Santa Fe Institute. Swarm is intended to be a useful tool for researchers in a variety of disciplines. The basic architecture of Swarm is the simulation of collections of concurrently interacting agents: with this architecture, we can implement a large variety of agent based models"

This development toolkit supports the Objective C and Java programming languages. The kit is very well documented and has been used to implement models of termites, 'heatbugs', and social dynamics.

The Cell Programming Language

Work was performed by Pankaj Agarwal (1995) who implemented a language called the Cell Programming Language (CPL). Agarwal used CPL to create cell-based computer models for the study of developmental biology. He used the language to model slime molds aggregation, limb skeleton formation, and sponge reconstitution. This work modeled cell behavior and development but did not use evolution to generate the cell genotypes in simulations. Cells existed in grid locations and executed functions in discrete time steps.

3 Experiment

3.1 Goal of Experiments

The goal of the experiments was to study the ways in which multicelled organisms can emerge in an Artificial Life setting. The motivation for studying this is that it may contribute to the much larger goal of understanding emergent behavior, including its evolution, and computation made possible by it.

The plan for these experiments was to create an artificial world in which cells struggle for survival. The cells are controlled by the expression of their genes, so a cell's genes determine a cell's actions and how successful it is going to be in its environment.

The final goal is the creation of an evolving population of cells that demonstrate success by survival. Survival is defined not by the age that a cell reaches, but by the length of time that a cell's family tree survives.

As the program construction was being designed, several milestones were planned. Each milestone consisted of a sub-experiment that might provide interesting results while at the same time provide a useful checkpoint for the program development. This allowed for the monitoring of progress during the lifecycle of the research project.

3.2 Overview of Phases

Slime Mold Simulation

A logical first step in the study of Multicellularity is the simulation of

Cellular Slime Molds. This was also a convenient first step in the development of the program since it required that a program architecture, user interface and objects be built, but did not require any evolution or genome descriptions. Instead the simulation used hard coded rules that each cell follows. At a high level, all cells release a signal and are attracted to the highest concentration of that signal. The signals are a chemical that is diffused in the environment.

Basic Symbiotic Relationship between cells

The next step for the program was including the ability to have multiple types of cells, each with their own chemical requirements and signals. This experiment showed how different cell types can work together in a symbiotic-like relationship. In some ways, this could also be seen as a demonstration of differentiation since the cells behave differently according the chemical around them.

Introduce Survival Criteria for cells

Once the program included functionality to display cells and chemicals in the environment and the management of cell-cell interactions, the next step in the program development was the introduction of cell health and survival requirements. Cells require a chemical in order to produce energy. Their energy is decreased each time step with additional amounts deducted for movement. If a cell is not able to find the chemical that it needs for energy production, it will die. Cell division was also created during this phase.

Evolving Cell Functionality

The final phase of the research experiments is the evolution of cells. This phase build upon the previous experiments. New functionality added in this phase is the creation of the cell genotype and it's resulting phenotype. Cell division includes possible mutations, which make the population change and hopefully improve over time.

3.3 Implementation Details

3.3.1 Choice of Java vs. Other Languages and Packages

Java was chosen as the implementation language because of the ease of graphically depicting cell activity. Displaying a population graphically is not critical for the program to be able to control cells and chemicals. It would be possible to just print out a log of each cell's actions, coordinates, and statistics. However, it would be very difficult to witness emergent behavior at a global level without being able to view the interactions of the cells. Performance is a concern with Java. In order to improve performance, the graphic display of program activity can be temporarily suppressed.

Some of the packages described in previous sections such as Swarm and StarLogo were considered. However, the ability to design and develop an implementation from scratch was considered a valuable experience. Also, since the duration of the project was relatively short, it was considered a risk to begin using a package without having an appropriate amount of time to study its limitations beforehand.

3.3.2 Summary of Classes and Objects in the Program

With the intent of creating a flexible program that can be used for future experiments, the design of the program was kept as simple as possible. There is one class that manages the user interface, one overall class that controls the artificial world's time steps and a few classes that control the objects in the program. The classes are outlined below with a brief description. For more information, see the API documentation in the Appendix.

NWPanel is a class that controls the user interface of the program. This class is an extension of the Java Applet class. The main functions of the class are to receive input from the associated applet and to call the application. The application is called when a thread should be started, paused, resumed or stopped. Also, the program parameters that are accessible from the applet may be changed and a method must be called in the application to change these values if they are modified.

NWWorld is a class that controls the overall execution of the Artificial World. This class is constructed when the simulation is started from the applet. When this happens, a thread is started. Once a thread is operating, this class creates all of the objects in the environment such as the chemicals and cell populations. Time steps are executed which makes calls to each of the objects. The NWWorld class also controls the painting of the graphical display of the artificial world.

The **Cell** class allows for the creation of individual cells in a population. Cells can be created at random or specific positions. They can also be created as a result of cell division, in which case they inherit the genes from their parent with possible mutations. If a cell is created from scratch, then the cell's genotype is randomly created. The cell class has methods that control the cell's movements, energy management, and sensing of its surroundings.

CellPopulation is a class that manages a population of individual cells. This class creates a population, calls each cell in the population when a time step should be executed, and manages the death of a cell if it has run out of energy. This class also controls the printing of cell descriptions and statistics.

The **Chemical** class is used to construct an object for each substance in the environment. A chemical object is created that describes how much of the chemical is present in each location of a two by two grid. There are methods available for adding amounts of the chemical to a location (e.g., when cells release the chemical or a mouse click on the applet is used to insert chemicals). Chemical levels are reduced through diffusion occurring at each time step and by the absorption of the chemical by a cell.

3.4 Non-Evolving Experiments

3.4.1 Slime Mold Experiment

Overview

The purpose of the Slime Mold experiment was to observe the aggregation behavior of simple cells and see how they could function as a more complex individual. As described in the earlier section on Slime Molds, the cells spend most of their life as single celled organisms. When they are close to running out of energy and need more food sources, they begin to signal each other and join to form a multicelled organism. The slug-like organism is then able to move rapidly, delivering the cells to a new area where food might be available. The experiment performed here focuses on this aggregation phase.

Other than observing slime mold behavior, a second reason for performing the Slime Mold experiment was that the functionality required for Slime Mold simulations was also needed as a base for the following experiments. Thus the experiment provided an appropriate milestone to manage project status and ensure that the environment and graphical capabilities of the program were on track.

The functional requirements necessary for the Slime Mold experiment were:

1. Applet and User Interface for program control
2. Chemical and Cell Objects
3. Diffusion of chemicals
4. Cell Movement
5. Display of the 2D grid environment
6. Display of chemicals and cells
7. Management of time (program threading)

Experiment Setup

Two different strategies for cell movement were used in the creation of the Slime Mold experiment. The first strategy allowed the cells to decide their movements based on the gradient of the signaling chemical. The cell was allowed to move directly towards that gradient. This cell movement strategy shared some similarities with cells that move with cilia.

The second method for cell movement was designed using polarity for cells.

This type of cell movement was more like cells that have flagella. Instead of being able to move directly towards a gradient, the cells were given the freedom to move forward while turning a certain degree to the right or left. These cells were also restricted in the way they read the chemical strength in the environment. Instead of being able to directly interpret the gradient of the signaling chemical, they were only able to sense the amount of chemical present straight ahead, 45 degrees to the right and 45 degrees to the left. The cell would then choose to move towards the direction with the strongest concentration of the chemical.

The second setup described followed a design similar to Resnick's StarLogo implementation (1994). Even though the eventual evolutionary and genetic programming aspects of this experiment would be very different then the StarLogo implementation, the functionality of this first experiment matched StarLogo well. Although the design of cell functionality was similar, the results were expected to be different since the underlying programs controlling the cells were different. Also, the StarLogo programming language created an abstraction level, which restricted the understanding of what functions were doing at a detailed level.

In the StarLogo experiment, cells were created with polarity. They sniffed ahead straight and to the right and left and moved towards the strongest concentration. A random component to cell movement was also added so that the cells would not always move directly towards the strongest signal. After sniffing and determining how a cell would turn, randomness was added. The direction in degrees was increased by a random number between zero and 40 and then decreased by a random number between zero and 40. The resulting random change averaged zero. The random movement component was similar to that used in StarLogo.

Results

Once the control functions of the slime mold cells were programmed, the program was executed and observed. As hoped, the cells began to aggregate in a manner similar to real cellular slime mold. One difference between this simulation and real life was that the cells did not follow a spiraling path during aggregation. This result was similar to that of the StarLogo experiment and can possibly be explained by the constant signaling of the cells. In real life, the cells emit a signal pulse instead of a constant amount. This may affect the motion of the cells that are following the gradient of the signal.

A population of cells was created by the initiation of the program. Cells were created in evenly distributed random locations throughout the grid. Within a few time steps, it was possible to observe the cells beginning to join together in groups. After several more time steps, most of the cells had found groups. Due to the random element in the cell movements, cells occasionally jumped out of the groups and would appear to be exploring. Sometimes the cells would simply turn around and rejoin the group. However, if another group was nearby, the cell might join that group. An interesting effect of the exploration of cells was that the cell would occasionally pull two groups together. Small movements of other cells caused this. If a single cell that had temporarily left its group came near a second group, a couple of cells in the second group might be near its edge and be attracted to the wandering cell. They might move slightly toward that cell and attract other members of the second group to do the same. The small movements received positive reinforcement, which eventually caused the entire second group to move towards the first group. In the end the groups of cells would be merged together forming one cluster of cells.

The same effect was possible for divisions of groups. Sometimes a wandering cell took other cells with it. In some cases, the movement of those cells carried a momentum strong enough to split the original group in two.

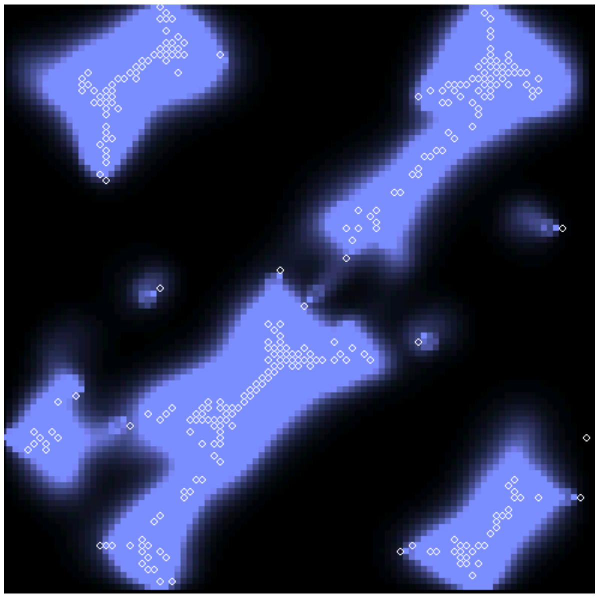


Figure 1: Slime Mold with 45 Degrees of Movement (500 Cells)

Analysis of parameter settings

Population Size

Several population sizes were tried to see if the number of cells had an effect on cell behavior. These experiments were carried out with cells that have polarity and which were limited to turns up to 45 degrees. Very small population sizes, e.g. ten cells, had a harder time finding other cells with which to join. Eventually the cells in these populations would find others and form small groups.

Sometimes these cells would end up 'following their tails' since they were attracted to their own signal. However, since the cells had polarity and could only turn at a certain degree, each cell's signal was only present behind it. A cell would have to turn back towards where it had just been in order to sense it's own signal. Due to the random component of cell movement, this situation did occur often. The randomness also ensured that the cells did not always get stuck following their own signal. Eventually, cells came across other cells and would begin to follow each other, almost as if chasing the other cells. Cells eventually found larger and larger groups. The population never completely stabilized. Groups of cells would form, exist for a while and then split off into new groups. The signals from the cells were diffused far enough into the environment to attract the cells into forming one cell group.

Larger populations of cells (e.g., 100 or 500) found groups much faster. The groups formed were also larger since greater amounts of the signaling chemical could be concentrated in certain areas. Just as with smaller populations though, the cell groups would continue to drift, break apart, and form new groups. The state of the population was constantly changing.

When populations of size 1000 or greater were used, the cells would form similar patterns. However, the concentration of chemicals became so great that the majority of the environment had a relatively high amount of the chemical. Since the cells followed the greatest concentration of chemicals though, the patterns formed by the cells themselves were roughly the same as with smaller population sizes.

Polarity and Random Movement

As described in the setup detail, two methods of cell movement were tested. The first method resembled cells with cilia. These cells had no polarity and could directly follow the gradient of a chemical. The second method tested used was closer to cells with flagella. These cells had polarity and could only move ahead straight or in a certain range of degrees.

Cells that were allowed to move in any direction exhibited the behavior that was expected. They quickly moved toward each other and formed groups. Random movements had little effect in their overall behavior. The problem encountered with this experiment was that collision detection had not yet been implemented in the program so the cells would all eventually occupy the same point on the 2D grid.

When polarity and limited turning was introduced, the behavior of the cells became much more interesting and life-like. Cells would move toward each other and would have to keep moving past each other. They would usually turn around and pass each other again or form a circular or figure eight type of pattern. Randomness made sure that the patterns changed over time. These cells could only move forward. They could either move straight ahead or forward to the left or right. With this control rule in place, the cells never stopped moving. Once groups of cells formed, the cells would continue to move around inside the group.

Each cell maintained it's current direction as a local variable. Usually during the experiments, the cells moved one step forward in the direction they started in plus an amount to turn: negative 45, 0, or positive 45 degrees. The random component would then be added after the cell decided the turning amount. When smaller degrees of freedom were given to the cell, the behavior was much more erratic. With 20 degrees of freedom allowed, cells would still form into groups. However, since cells could not turn around to rejoin the group as easily as it could when it was allowed 45 degree turns, the cell would continue past the group and then rejoin after making a wide turn. This had less effect when the cell groups were larger and the cells could still turn while close to the group. However, smaller sized groups would take on a spider shape as cells passed through them and rejoined or continued on to find other groups. This behavior resulted in much more exploration by the cells and eventually resulted in less groups being formed overall with larger group sizes. Populations of 100 or more cells tended to eventually form a single group with occasional cells wandering away and rejoining again after a while. Figure 2 shows an example of this behavior.

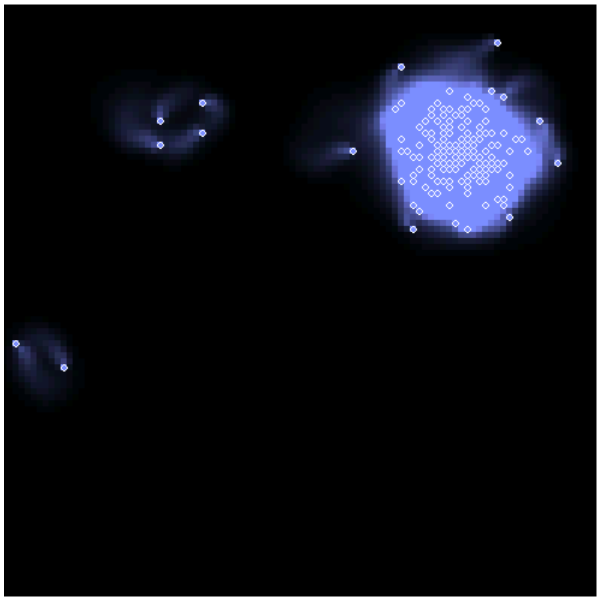


Figure 2: Slime Mold with 20 Degrees of Movement Allowed (200 Cells)

When small population sizes were used in combination with limiting the cell's turning ability, the cells did not form into groups as easily. This was probably because they could not form a critical mass of chemical significant enough to attract each other. Their turning radius was too large for this.

It was interesting to observe that some cells would orbit around groups. The cells would then be constantly turning towards the group, but only at the turning degree allowed so it would never turn enough to enter the cluster. This behavior was rare since both the group size and turning radius had to be in the right proportion. The random behavior of the cells prohibited this from happening for a long period of time. Another interesting pattern were cells that formed a large ring. Multiple cells followed each other forming a ring of five or six cells that all followed each other.

Cells that were given the ability to move with a wider turning radius than 45 degrees were able to be much more responsive to the chemical gradients.

Turning amounts of up to 90 degrees were allowed. Smaller clusters of cells resulted from this setting. Since the cells would form small groups and could turn quickly enough to stay in the groups, the cells performed little exploration. For this reason, the cells tended to stay in many small groups.

3.4.2 Symbiotic and Survival Criteria Experiments

Two additional mini-experiments were carried out as milestones before the evolving experiments were begun. These experiments provided a chance to test that the program's functionality was working correctly during development.

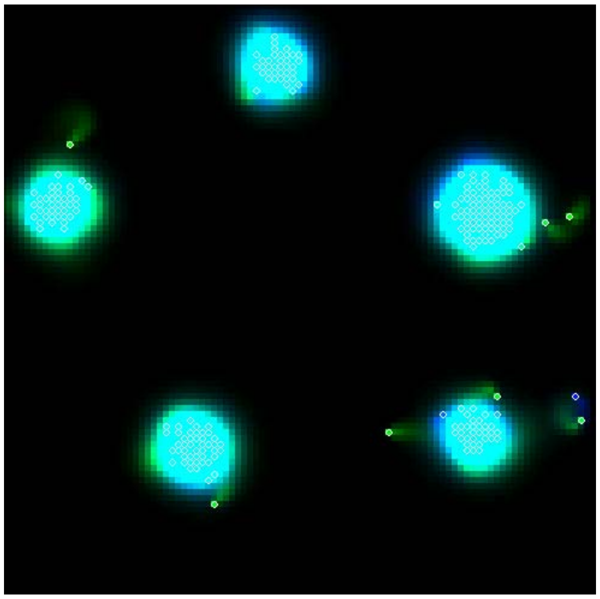


Figure 3: Symbiotic Relationship (500 Cells / Population)

Symbiotic Relationships

The first experiment consisted of creating two populations that were attracted to each other's output chemical. This resulted in a symbiotic-like relationship between the two cells. The cells in both populations were created in random positions. After a few time steps, the cells would begin to move towards the other population creating symbiotic groups of cells. A screen capture of this behavior is shown in Figure 3.

The additional functional requirements necessary for the symbiotic experiments:

1. Management of multiple populations of cells
2. Parameterization of each cell's chemical requirements and emission
3. Management of multiple chemicals in the environment

Survival Criteria

The second mini-experiment in this phase was testing the functionality of survival criteria. For this experiment to be run, cells needed to have a quantity of energy that decreased over time. Cells were given the ability to absorb chemicals around them to increase their energy level. Once chemical amounts were absorbed, it was removed from the environment. This created a competition for resources among the cells. If a cell was not able to maintain an energy level above zero, it was considered dead and was removed from the population.

Cell division was also implemented for these tests. When the tests were executed, the cell populations would often die out quickly or explode, reaching large numbers. It was hard to properly balance the amount of energy resources available in order to achieve a stable population. A maximum and minimum population constraint had to be created in the program in order to control population size. If the population fell below a certain level, new cells would be randomly added. If the population reached the maximum, no more cell division would be allowed.

If two symbiotic populations were used in conjunction with cell division, the population would often grow too quickly. This is because the cells had a variable energy supply—dependent only on the size of the other population. Population sizes were much more stable once a limited amount of energy was available in the environment. This was implemented by creating random energy sources that would last for a period of time and then run out. Usually one or two energy supplies were available in the environment at the same time.

The functional requirements for cell survival experiments:

1. Energy tracking
2. Energy decreasing at each time step
3. Cell death
4. Population size management
5. Energy sources
6. Cell division
7. Collision detection

3.5 Evolving Multicellular Experiments

In the previous non-evolving experiments, it was shown that very simple cells could produce interesting behavior when combined together in an environment. They joined together into groups, rings, and other patterns and in the final experiment either died or survived based on this behavior. One lesson learned in the experiment that contained survival criteria was that it was hard to find a balance between cell rules, population size, and the survival requirements. Either the environment was too challenging and all of the cells died off, or too easy and populations had no problem surviving. When the balance of the environment was right and cells only survived if their control rules were appropriate, then more interesting behavior emerged. But explicitly programming a complex system like this was difficult.

The goal of the evolving experiment was to try to automate this programming. If cell control rules could be evolved, then achieving appropriate cell interactions and emergent behavior could occur in a more natural way. Whereas it was hard to force emergence, the goal of this experiment was to see if it could occur by itself. In this setting, the study of multicellular activity and self-organization of cells could hopefully be observed.

The requirements for creating an evolving population of cells needed to be identified. The following sections describe the inputs that each cell needs from its environment. The functions available to each cell are also described. Once cells are defined by their inputs and outputs (i.e., actions), the representation that describes the cell genotype must be defined. In these experiments, a form of genetic programming was used called Cartesian Genetic Programming (CGP). This method will be described in the following sections.

The additional functional requirements necessary for the evolving experiment were:

1. CGP genotype creation
2. CGP phenotype decoding
3. Cell division
4. Mutation operators
5. Absorption of a chemical (food) by a cell, decreasing the level of the chemical in the location of the cell
6. Ability to select a cell using the mouse to view its phenotype
7. Ability to trace the energy level and chemical inputs for a selected cell over time
8. Printing the phenotypes and statistics of the entire cell population
9. Additional controls and modifiable parameters on the user interface
10. Ability to turn graphics on and off to improve performance
11. Local variables for lower and upper energy thresholds

3.5.1 Cell Inputs

Cells need to be able to decide their actions with a realistic knowledge of their environment. Real cells do not know anything about their location or the activities of other cells outside of their immediate area. If they are part of an emergent complex system, they do not understand their role in it. Instead, they only know what they do, which is based on their surroundings and have no idea of their action's impacts on the greater population. It is therefore important to decide what inputs the cells will receive from their environment. The hope is that these inputs will be realistically limited and at the same time provide adequate information for the cells to decide their actions.

External Inputs

The cells in this experiment are not able to see or feel anything. The only external input that they have is the level of chemicals present around them. This is similar to nature in the way that cells absorb chemicals. A cell directly absorbs some chemicals through its membrane. Other chemicals do not get absorbed immediately through a cell's membrane but are instead limited by receptors on the cell's exterior. Receptors work as a threshold function that only allows chemicals to be absorbed once a high enough concentration of the chemical has been collected by the receptor. The cell can base its actions on the level of chemicals absorbed from its environment. Since a cell cannot see or feel a neighbor cell, it only knows of its neighbors by the chemical signaling from the cells around it.

A cell can sense different types of chemicals from the environment. The types of chemicals are the signals from other cells, waste from cells, and chemicals that a cell needs to produce energy.

Internal Inputs

The cells in this experiment also have internal inputs. The cells know what their current energy level is and how old they are. This type of information is useful when a cell determines whether or not it should divide.

3.5.2 Cell Functions

One of the goals of this experiment was to explore the possibilities of emergent behavior and multicellular functions based on combined actions of very simple units. Although the behavior exhibited as a group may be complex, the actions of individual units should not be. Therefore, a simple set of functions was given to the cells. The idea was to give the cells a limited set of actions, but not so limited that surprising actions would not be possible.

The set of actions available to a cell is given below. In a single time step, the actions are mutually exclusive with the exception of signaling. This means that a cell cannot move and divide at the same time. A cell can, however, signal and perform other actions at the same time.

Cell Movement

One basic function of a cell is movement. The cells receives its inputs from its surrounding and can choose to move or not move based on those inputs. A simple way to describe cell movement was desired so that it would be easy to represent in a genotype. Instead of enabling a cell to move left, right straight, turn around, etc., it was decided that cells would only be able to move towards or away from chemicals. In biology, this is called chemotaxis. Using this kind of movement allowed for simple representation and at the same time kept the experiment close to the way cells behave in nature.

Cell Division

Cells also have the ability to divide. Once a cell divides, the cell itself no longer exists. Instead, two offspring are created. The offspring each have half of the energy of the original cell minus an energy cost for the division process. The offspring inherit the genes of the parent cell plus some possible mutation.

Chemical Signaling

Cells have the ability to emit a chemical into the environment. This action gives the cell a way to communicate with other cells in a population. An example of signaling is seen in Cellular Slime Molds when they use cyclic AMP signals during their aggregation phase. When a cell emits a signal, it is still able to perform the other actions listed above.

3.5.3 Survival Criteria and Fitness Evaluation

A cell's fitness is not measured explicitly during the program execution. Instead, the ability to survive is the measurement of fitness. The length of an individual's lifespan often describes how successful an individual is at survival. Another way to describe fitness based on survival is not the length of the individual's life, but the time period during which a cell's family DNA survives. In this scenario, the only thing of importance is the ability of a cell to pass on its DNA to the next generation. This is the definition of fitness taken for the experiments here.

A cell will survive as long as it has energy. Since energy must be sourced from chemicals in the environment, a cell must either be positioned near an energy source, or move towards it. The balancing of incoming energy and expenditures of energy (e.g., division and movement) is required for a cell to survive.

3.5.4 Cartesian Genetic Programming

In this section, a brief introduction to Cartesian Genetic Programming (CGP) is given. The paper (Miller and Thomson, 2000) should be referenced for more details on the topic.

In order to control the actions of the cells, some form of instruction must be encoded to serve as the cell's genotype. While Neural Networks and other types of Machine Learning techniques were considered, it was decided that some form of Genetic Programming (GP) would be best suited for this experiment. This would allow a direct encoding of the inputs and actions to be taken by the cells into the cells genotypes. Since the cells would not be learning during their lifetime some advantages of other Machine Learning representations (e.g., learning ability of Neural Networks) would not be helpful.

Genetic Programming is a method of combining functions and terminals so that programs can be randomly generated and evolved. Most implementations of GP, for example the type of GP described by Koza, creates its representation as a parse tree. This can also be represented as a string that resembles LISP programming language syntax. This type of

Genetic Programming has been proven effective in many applications (Koza, 1992) such as digital circuit design, pattern recognition, and other tasks. Tree-based genetic programming has proven to be a powerful method of evolving programs. Some drawbacks to this technique are that the use operators such as crossover and mutations can be difficult to employ. In some cases, the tree must be repaired after operations have been performed. The parse-tree representation also often leads to a one to one relationship between the genotype and the phenotype.

Cartesian Genetic Programming (Miller and Thomson, 2000) shares some similarities to the GP techniques described. However, the CGP genotype is formed by a directed graph instead of being tree based. In this representation, a string of numbers is used to represent a fixed length genome. However, the mapping from the genotype to the phenotype is not one to one, which results in a variable length phenotype. One benefit of this is that the genotypes in members of the populations can be different, although the phenotypes of those individuals are the same. The mapping of the genotype allows for a high degree of diversity of the genotype among equally fit individuals. This permits a much greater search space from crossover and mutation operations in a single generation.

As described in (Miller and Thomson, 2000), the redundancy inherent in CGP allows for neutrality, the genotypic diversity of equally fit phenotypes. Three forms of redundancy are described: node, functional, and input redundancy. Node redundancy consists of nodes existing in the genotype that are not part of the phenotype. Functional redundancy is the repetition of parts of the program and causes bloat. Input redundancy is when some of the inputs to the program are not connected to functions. Redundancy is thought to be a useful property during the evolutionary process by promoting diversity in the population.

Typically a genotype in CGP consists of a series of nodes. The nodes are connected to other nodes forming a graph. The genotype can be described as a series of columns with multiple rows or as a single row. In this research a single row CGP representation was used.

In the paper (Miller and Thomson, 2000), a detailed example is described using Cartesian Genetic Programming to solve the Santa Fe Ant Trail problem and to solve for a sixth order polynomial function. The use of representations in CGP was adapted for this research. An example of this is given in the next section.

3.5.5 Chromosomes – Representation

Explanation of Genotype Representation

The representation used for this CGP implementation consisted of functions and inputs. However, the inputs to the genotype were not represented as nodes as in typical CGP programs. Instead all of the nodes in the genotype represent a function.

As in standard CGP, the genome consists of a number of genes that are linked together to form a graph. The right-most gene is executed first. Starting from the right, the genes are linked to any cell on the left. The last node processed is always set to the function 'do nothing'. During the decoding of the phenotype, if no action has been reached while traversing the graph of genes, then eventually the right-most node will be called. In that case, there will be no action for the cell. As larger sizes of fixed length genotypes are used, this case occurs less frequently.

Each node, or gene, consists of an array of four numbers. The numbers respectively represent the function number, an external input, and two connections inputs. Note that examples given in (Miller and Thomson, 2000) are in a different order with the function in the last position. The representation of a single gene is shown in Figure 4.

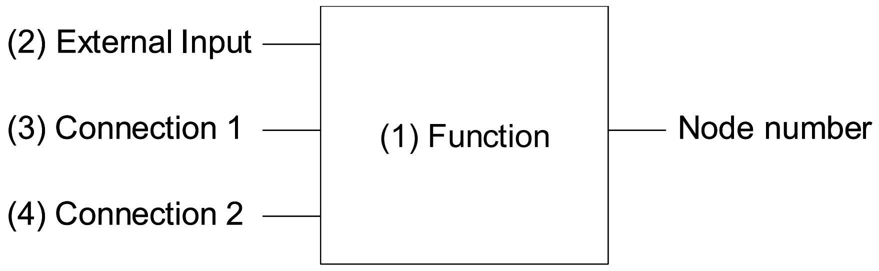


Figure 4: A Single Node in the CGP Representation

The first part of the gene contains the function, marked as (1). The functions are linked to the remaining positions in the gene during the decoding of the phenotype. Below is a list of the functions used in the creation of the genotype. In the list, each number in parenthesis refers to a position in the array of the four numbers making up the gene.

0. Do nothing
1. Move towards (2)
2. Move away from (2)
3. If (2) is present do (3), else do (4)
4. If (2) is present do (4), else do (3)
5. If energy is above UPPER_THRESHOLD do (3), else do (4)
6. If energy is below LOWER_THRESHOLD do (3), else do (4)
7. Perform (3) then divide
8. Perform (3) then release chemical signal

Figure 5: Function used in CGP Representation

The second position in the gene refers to the external input (2). This is an array of elements from the cell's environment. In this implementation, the external inputs are a list of chemicals. The number zero represents "Chemical A" and the number one represents "Chemical B". Chemical A serves as food for the cell population, and Chemical B is what is released as a chemical signal by the cells. This is true for the main population of cells. Other populations are possible that have different uses for the chemicals. If function 1 is called, the value of the external input in (2) is evaluated. If the value is zero, the cell moves towards Chemical A. If the value is one, the cell moves toward Chemical B.

Functions 3-8 use the values in the third and fourth positions on the gene.

These positions store links to other genes. If function 7 is called for example, the cell will decode and perform the instructions in the gene referenced in position (3) in the array of the current gene. Then the cell will perform cell division. Function 8 operates in the same manner except that it instructs the cell to signal instead of divide.

Functions 3 and 4 are IF statements that depend on the presence of the chemical identified by the external input in position (2). Presence is a Boolean value determined by the level of the chemical in the x, y coordinate occupied by the cell. This is currently a fixed threshold shared by all cells. In future implementations, this could be made a local variable that is subject to mutation. This would have the effect of evolving the cell's membrane, making cells more or not as sensitive to the chemicals in their environment.

The variables UPPER_THRESHOLD and LOWER_THRESHOLD used in functions 5 and 6 are local variables of each cell. This permits each cell to have different threshold limits that control its actions. The values for these variables are created randomly for each cell. These values are not encoded in the CGP string, but can be considered part of the cell's genotype since they are used during the decoding of the phenotype. The values for these variables are subject to evolution just as is the rest of the cell's genotype.

In summary, the cells have the ability to do nothing, move towards or away from a chemical, signal and divide. Their genome contains functions so that when decoded, these actions are either performed directly or as a result of the chemicals around them and their current energy levels.

The size of the genome was a program parameter. Genomes of lengths five to fifty genes were tested during initial experiments. Longer genomes often resulted in more complex phenotypes. The length of the genome had an impact on the amount of genetic drift and neutrality that was possible. Since only one gene was mutated during cell division, a longer genome meant a lower probability that the mutated gene was expressed in the phenotype. Shorter genomes were more likely to reach the left-most gene, which was coded with the 'do-nothing' function during the decoding of the phenotype. To balance the complexity of the phenotype and the amount of neutrality, a genome of length 20 was used during the experiments.

Genotype-Phenotype Mapping Example

In order to fully explain the decoding process for a phenotype, it is helpful to give an example. This example shows a cell's genotype and its corresponding phenotype. A sample cell genotype is given in Figure 6. The positions containing functions are underlined.

```
       0         1         2         3         4         5
  | 0 0 0 0 | 7 1 0 0 | 1 0 1 0 | 1 0 1 2 | 3 1 0 0 | 3 0 2 4 |
```

Figure 6: Example Cell Genotype in CGP

The genotype described in Figure 6 can be represented pictorially as shown in Figure 7.

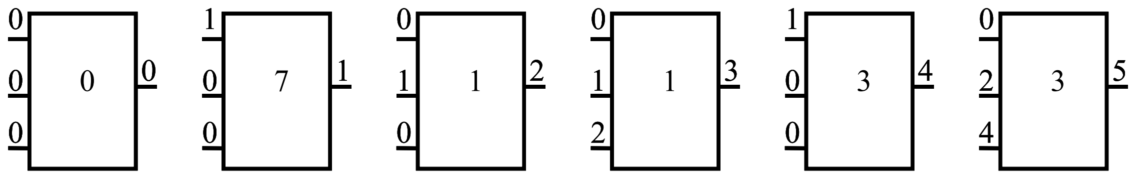


Figure 7: Pictorial Representation of a Cell Genotype in CGP

To map the genotype to a phenotype, the right-most gene is assessed first. In the example, this is gene number 5 and the function in position (1) is function 3: *If (2) is present, do (3) else do (4)*. The value in position (2) is zero, which represents Chemical A. If Chemical A is present around the cell (over a fixed threshold), then decode and perform the gene identified by position (3): which is gene 2. Otherwise perform the gene identified in position (4): gene 4. This process continues until an action is found.

The fully decoded phenotype for this example is shown in Figure 8. The resulting phenotype describes the behavior of the cell and is a series of IF-ELSE statements and cell actions.

```
If Chemical 0 is present
     Move towards Chemical 0
Else
     If Chemical 1 is present
          Do nothing
     Else
          Do nothing
```

Figure 8: Example Decoded CGP Cell Phenotype

3.5.6 Genetic Operators

The only operator used in this implementation is mutation. The cells are replicated only by cell division so sexual reproduction and crossover is not relevant. An elitist form of evolution is used so that after cell division, one cell remains the same so the parent. The second cell is always mutated. There are two steps in cell mutation: mutating the CGP string, and mutating the cell's threshold values.

The first step in mutation consists of randomly changing one of the integer values in the genotype's CGP string. Only one value is changed per mutation. Possible changes that can result from mutation include changing a function, a chemical input, or a node connection within a gene.

A second step is modifying the cell's threshold values. In this implementation each cell has two local variables that contain and upper and a lower energy threshold value. This permits the cells to act according to their current amount of energy. These variables contain a real number, which is mutated in a manner similar to the mutation performed in Evolution Strategies (Schwefel, 1981). Each threshold variable is increased or decreased by adding to it a random number. The random number is similar but not equal to a normal distribution. It is actually a random number added to the variable and then a second random number in the same range subtracted from the variable. This type of random function in Java was used instead of a normal distribution so that the applets would still be compatible with most Internet browsers. The range of the random numbers was equal to 20% of the maximum energy value allowed for a cell.

3.5.7 Population Size and Fitness Evaluation

An initial population of cells was created for each execution. The number of initial cells is a parameter modifiable in the user interface. All of the cells in the population were generated with a random set of genes. If the population size decreased over time, the program would generate new cells randomly placed in the environment. This kept the population at the minimum size required by a parameter controllable from the user interface. A maximum size parameter was also available to the user to control how many cells could be created through cell division. Once the maximum limit was met, the cells would not be able to perform cell division.

A steady state algorithm was used in which cells would continue living or dividing until they no longer had energy and died. When a cell divided, the rest of the population would remain as-is. Fitness was not explicitly measured, but instead was a result of the cells being able to perpetuate the existence of their DNA. Cells that had a long life and did not self-replicate were considered equally fit to cells that divided often as long as both strands of DNA continued remain active in the environment.

It was decided that a generational algorithm would not produce the desired results. By selecting the cells with the most energy and allowing them to divide and continue on to the next generation, it would have been possible to implement a generational algorithm. However, this was not desired since cells that could continue to live or divide successfully with a small energy amount were considered just as fit as others with large amounts of energy. This would have given cells incentive to evolve to search out energy, not to pass on their DNA. A generational algorithm would also have required significantly more processing time and would have been less similar to nature.

3.5.8 Results

The evolving cell experiment produced cells that exhibited interesting behaviors. It was entertaining to witness an initial population of cells, each with their own, often odd, behavior. Some cells would circle around, others would do nothing and signal, and others would move quickly and efficiently towards energy sources. As time steps passed, the cells with losing strategies would die off and cells with better fitness would either continue to live or produce offspring. After some time went by, it was possible to observe recurring strategies and even some emergent group behavior.

A family of cells can be considered to be all of the cells in a population that share a common lineage and therefore DNA. Even though many of the cells had mutated over several generations, often the behavior (the phenotype) of the cells would remain the same. Observing families of cells often revealed group behavior that could be thought of as the actions of a multicelled organism. Observations of experiments and the effects of different settings are documented in the following section.

3.5.9 Analysis

General Observations

The populations of cells during the experiments usually tended to find a working strategy and use it for a while. Occasionally, an improved mutated phenotype would evolve and eventually dominate the population. Sometimes, even though a population consisted mostly of a successful breeding cell family, a few older cells would exist. These cells did not divide but merely had the instruction to move towards energy sources. They had been lucky enough to continue finding new sources after each old source ran out. Since they did not divide and split their energy in half, they were able to keep more energy for themselves. This store of energy gave them a longer time to search out a new energy source before running out of energy. In the long term, however, these cells almost always died off eventually. Either they would ultimately not find a new energy source or cells that had already found the new source would block them out.

Families of cells that divided showed much better success at long-term survival. By accepting the energy cost of cell division, they were weakened individually. In spite of this, as a group they were stronger. The families of cells were able to distribute their stored energy into more cells so that a greater area of the environment could be explored for new food sources. It didn’t matter that a large percentage of the cell family died by not finding sources. The probability of at least a few of the cells finding an energy source was high and those cells would absorb energy and divide to create the next generation. This was an example of cell cooperation. The cells were sacrificing their own fitness for the fitness of the group. As mentioned in Section 2.2, the movement from competing cells to cooperating cells might be one of the first requirements for the development of multicellularity.

Genetic Neutrality and the Cambrian Period

The rapidly changing environmental conditions during the Cambrian Period are possibly the cause of the genetic diversity and progress for which the period is known (Kirschvink, Ripperdan, and Evans, 1997). In the spirit of this time period, the environment's energy sources were increased or decreased over time. This made it possible to observe the robustness of individuals and watch the phenotypes adapt to new conditions. Changing the diffusion and evaporation parameters was also an effective way to change the environment during experiments.

When environmental conditions were changed, a stable population started to adapt to the new environment. Slowly new mutant cells would begin to appear that were better positioned to compete for the energy sources. An example seen often was the evolution of a simpler phenotype when resources were plentiful and more complicated phenotype behavior when resources were scarce. A lack of energy source meant that cells needed to be more careful about conserving their energy and could not divide as freely.

A high degree of neutrality was created in the populations over time. Even though one phenotype might begin to dominate the population, mutations would constantly be changing the genotype of the cells. Only rarely did a mutation affect the phenotype behavior. Often when a mutation did manage to change the phenotype, the cell would not behave well and would die quickly, removing it from the population.

After many generations of cells dividing, a population would often consist of a very diverse set of genotypes which all mapped approximately to the same phenotype. When the environmental conditions changed as described above, the neutrality and genetic drift seemed to have a positive effect on the population's ability to adapt to the new conditions. The diversity of genotypes due to neutrality led to a wide range of phenotypes being tested in the new environments. It was much easier for mutations to find a better-suited phenotype as a result of this diversity. Neutrality in this way made the population and families of cells sharing the same DNA origins much more robust.

Collision Detection and the Huddle Strategy

The ability to control whether or not two cells could occupy the same space was available as a checkbox on the user interface. The value of this setting had strong impacts on which cells were successful and how the cells evolved.

In order for a family of cells to be successful, it was important that it reached food sources quickly and then was able to take advantage of them when the cells were there. When collision detection was turned on, it was possible for groups of cells to monopolize the food source and block out other cells trying to reach it.

If the population was a moderate size and there was little competition for the limited supply of food, then usually cells evolved to simply move towards the food. However, when there was strong competition for food and collision detection was enabled, different behavior was more successful. Occasionally, cells evolved to move towards food, but once there, move towards each other's chemical signal. This behavior resembled a huddling action around the food that blocked out competition from entering. Signaling and moving towards the signal concentration kept the group as tight as possible. If instead a cell just moved towards the highest level of food in its immediate environment, the cell might move around a little. Since many cells are quickly absorbing the food coming from the source (the food drop-off point), a cell might sense a temporarily higher level of food in the direction opposite the food drop-off point—just outside of the cluster. While a cell might get a short-term benefit from moving away from the group, it might also lose its position and allow competition to enter. In this case, short-term greedy behavior hurt the cell and the average success of the family of cells.

Stopping Once the Destination Was Reached

When collision detection was turned off (and cell polarity was turned on), different types of strategies evolved. Without collisions, cells wouldn't be stopped just by running into each other, getting stuck, and forming clusters. Instead, they would approach the food source, absorb some of the chemical and then move past it. Then, having passed the source, they would have to turn around and go back.

The most effective behavior in this situation was to have a combination of the functions "move towards the food" and "do nothing" in the cell's phenotype. One successful combination was a group of cells that evolved to move toward the food as long as the cell's energy was low. When a cell moved over a food source, its energy would be quickly increased. The cell used an energy threshold function so that once the energy reached a certain level, the cell would stop moving and 'do nothing'. Often, this resulted in a cell stopping perfectly on the location of the food source, absorbing the maximum amount of chemical producing energy possible until the source was depleted. An example of a phenotype using this behavior is shown in Figure 9. This proved to be an effective strategy but was not necessarily beneficial to the group of cells in a cell family.

```
If Energy is greater than 5 then
     Do Nothing
Else
     Move Towards Chemical 0
```

Figure 9: A Cell Phenotype That Stays Still Once It Has Found Food

Evolution Exploiting the Environment

After the programming for cell division was completed, a few experiments were performed to see what behaviors would evolve and be successful. During this set of experiments, the same sort of phenotype became dominant almost every time. Regardless of how strong the energy cost was set for dividing, cells almost always evolved to divide immediately. They would keep dividing and die off if no food was found. The ones that found food flourished and continued to divide. The population size quickly reached its maximum almost every time.

It was later discovered that a bug was in the program that only applied the energy cost to one of the offspring. Regardless of how the division energy cost parameter was set, the other offspring would always keep the original energy level of the parent. This program bug meant that there was no risk to dividing. Usually a parent cell's energy would be split in half plus an additional amount of energy would be deducted for the effort of dividing. Since there was a bug in the way the environment operated, the process of evolution learned how to exploit it to their advantage.

Dividing Only While Located at an Energy Source

After the bug in the previous example was fixed, cells had to be more careful about whether or not they would divide. While it was important to produce offspring, helping their cell family to explore and increase chances of long-term survival, it was dangerous to divide unless energy was high or a food source was near. Each time a cell divided, its energy was more than cut in half. If a cell divided when its energy was low, it hurt its chances of finding a food source before running out of energy and dying.

When an experiment was run over a large number of time-steps, cells would often evolve to only divide if they had a high amount of energy and were thus probably near a food source. An example of a cell with this behavior is shown in genotypic form in Figure 10 and its phenotype is shown in Figure 11.

$\underline{0}$ 0 0 0	$\underline{7}$ 1 0 0	$\underline{4}$ 1 0 0	$\underline{2}$ 0 0 1	$\underline{5}$ 0 2 0	$\underline{7}$ 0 4 4
$\underline{0}$ 0 3 1	$\underline{8}$ 1 1 1	$\underline{1}$ 0 1 2	$\underline{0}$ 0 7 4	$\underline{1}$ 0 2 5	$\underline{5}$ 1 10 1
$\underline{1}$ 0 1 1	$\underline{4}$ 0 10 12	$\underline{4}$ 0 11 9	$\underline{0}$ 1 6 2	$\underline{6}$ 1 10 7	$\underline{2}$ 0 7 9
$\underline{6}$ 1 2 17	$\underline{6}$ 1 10 7

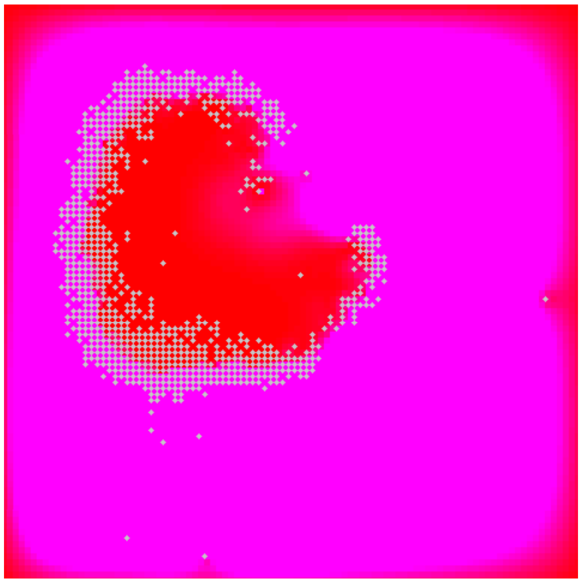
Figure 10: The Genotype of a Cell that Divides Only When It Is Near a Food Source

```
If Energy is lower than 8 then
     Move Towards Chemical 0
Else
     Release Chemical Signal and
     Divide and
     Do Nothing
```

Figure 11: The Phenotype of a Cell that Divides Only When It Is Near a Food Source

This program allowed the cells to divide only when they had a significant level of energy. If they were low on energy, they would move towards food. Once they found food and their energy increased, they would stop moving as the cells did in a previous example. While collecting energy, they would reproduce and not move. This often worked because it meant that they could position themselves directly at an energy source and divide as much as possible making the group strong. If a cell had used another strategy and did not divide, the single cell would grow strong but only up until the maximum energy level—the rest of the energy would have been wasted.

This strategy was very successful so the population was dominated by similar phenotypes. Since collision detection was turned off, almost all of the cells in the population would be located at a single position during their dividing phase. While collecting energy and dividing, the cells would develop a massive population that would stay at a food source and grow. Once the food source was gone, cells would continue to divide until the cells had an energy level below the threshold of 8. Then the cells would all go at the same time to search for a new food source. This mass exodus of leaving looked almost like a tidal wave of cells. Since all of the cells had been occupying the same location, a small amount of the energy-producing chemical was distributed throughout the environment. The cells would take advantage of this residual amount which helped them stay alive long enough to find the next food source.



```
Cell-783.2145 :      Energy: 0.7499999    Genome Created At Time: 2945
```

Figure 12: A Mass Exodus of Cells Looking for a New Energy Source

The mass exodus after an energy source was depleted is shown in Figure 12. The center of the group of cells is where the previous energy source was located. Since the population was large and all cells were releasing signals while dividing, the environment was filled with the red colored signaling chemical. The purple region is a mix between the blue energy-producing chemical and the red signaling chemical. Clicking on one of the cells in the population displayed the phenotype and statistical information. The cell selected had descended from the initial population. It's original parent cell was number 783 and it was the 2145th generation to be created from that cell.

4 Conclusions and Next Steps

The goal of this research was to explore the origins of multicellularity in an artificial life setting, helping to gain insight into emergent behaviors. Two primary experiments were conducted, a simulation of the aggregation of cellular slime molds and an evolving ecology of cells that were programmed with Cartesian Genetic Programming.

The slime mold aggregation experiment showed behavior characteristic of slime mold in nature. Cells were created in random locations and began to aggregate into a single group, resembling a more complex organism. During this experiment, the effects of randomness, polarity and other settings were explored.

The use of Cartesian Genetic Programming was successful in producing cell behaviors that were at the same time novel and successful. The cells evolved several interesting strategies that sometimes benefited from group behavior. Behavior such as huddling and blocking were discovered. Creative uses of evolving thresholds were also used by groups of cells to improve cell division strategies.

The cell rules in the second set of experiments were not carefully handcrafted for each cell and made with an expected behavior in mind. Instead the rules evolved by themselves. And instead of intuitive behavior, what evolved often resulted in surprising strategies that would not have been thought of otherwise.

The implementation provided a good base for the experiments performed in this research and should hopefully provide a good test bed for a wide variety of future experiments.

Improvements and Next Steps

This application was written in an older version of Java so that it would be compatible with most web browsers. For this reason, some sacrifices were made in terms of performance and code simplicity. For example, the Java class Vectors was used instead of some of the newer and more efficient classes. Also some graphical elements were removed once it was realized that Java2D was incompatible with many browsers. Future implantations using this code should take into consideration whether or not browser compatibility is an issue and modify the code accordingly.

Further experiments in this area might lead to more complex cellular behavior. Additional functionalities that might provide interesting results are cell adhesiveness, growing sizes of cells, and the ability for groups of cells to move at different rates than single cells

A partial design including some code for a Predator / Prey experiments has been done during this research and is documented in the code. In addition to Predator / Prey, the code could also be fairly easily extended to allow parasites, further symbiotic relationships, bottom feeders, etc. Other ideas are the use of a constant or periodic source of energy such as sunlight, the addition of more chemicals, and possibly even a current or stream-like effect. Combining adhesiveness with the effects of a current might show why it is helpful for organisms to attach to something stationary to capture passing food. Adding functions to the cells or the increasing the complexity of the ecology using these ideas or others might lead to more creative group behavior and might show additional insight into multicellular development.

One of the effects of using Cartesian Genetic Programming to evolve cell behavior was the high degree of neutrality and redundancy apparent in the cell's genotypes. It was observed that this feature seemed to have a positive effect when evolving the cells, especially during times of rapid environmental change. More work could be done in studying the specific impacts of neutrality and genetic drift in this context. Perhaps a comparison could be made between a population that allowed neutrality and one that didn't. To prohibit neutrality, genetic mutations could be restricted to those that made a change to the phenotype mapping.

5 References

5.1 Acknowledgements

The author of this paper would like to express gratitude to his project supervisor, Julian Miller, and other anonymous reviewers.

5.2 Bibliography

Agarwal, Pankaj (1995). The cell programming language. Artificial Life, 2 (1):3777.

Bentley, Peter J. (2001). Digital Biology: The creation of life inside computers and how it will affect us. London: Headline Book Publishing.

Berlekamp, E., Conway, J., & Guy, R. (1985). Winning Ways. For your mathematical plays. Vol. 2: Games in Particular. Academic Press.

Bonner, John Tyler. (2001). First Signals: The Evolution of Multicellular Development. Princeton, NJ: Princeton University Press.

Burkhart, Roger. (1994). The Swarm Multi-Agent Simulation System. OOPSLA Workshop on "The Object Engine" 7.

Cliff, Dave, and Bruten, Janet. (1997). Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets, Tech. Rep. no. HPL-97-141, Hewlett-Packard Laboratories.

Dawkins, Richard (1987). The Blind Watchmaker. New York, W. W. Norton.

Epstein, Joshua M. and Axtell, Robert (1996). Growing Artificial Societies:

Social Science from the Bottom Up. Cambridge, MA: The MIT Press.

Evans, D.A., Ripperdan, R.L., and Kirschvink, J.L. (1998) Polar Wander and the Cambrian. Science, 279: 9a-9e.

Fleischer, Kurt and Barr, Alan H. (1992) A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In C. Langton (ed), Artificial Life III, London: Addison-Wesley, pp. 389-416.

Johnson, Steven. (2001). Emergence: The connected lives of ants, brains, cities, and software. London: Penguin Press.

Keller, E. and Segel, L. (1970). Initiation of slime mold aggregation viewed as an instability, J. Theo. Biol. 26, pp. 399-415.

Kirschvink, Joseph L.; Ripperdan, Robert L. and Evans, David A. (1997). Evidence for a Large-Scale Reorganization of Early Cambrian Continental Masses by Inertial Interchange True Polar Wander. Science, 277:541-545.

Koza, John R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: MIT Press.

Krugman, Paul. (1996). The Self-Organizing Economy. Malden: Blackwell Publishers.

Langton, C. G. (1989). Artificial life. In Langton, C. G. (Ed.), Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of

Living Systems, Vol. VI of SFI Studies in the Sciences of Complexity, Redwood City, CA: Addison-Wesley, pp. 1-48.

Langton, C. G. (1992). Preface. In C. G. Langton, C. Taylor, J. D. Farmer and S. Rasmussen (Eds.), Artificial Life II. Redwood City: Addison-Wesley.

Lindenmayr A. (1968). Mathematical models for cellular interaction in development, Parts I and II, Journal of Theoretical Biology, Vol. 18, pp. 280.

Michod, R. E. and Roze, D. (1999). Cooperation and conflict in the evolution of individuality. III. Transitions in the unit of fitness. In: C. L. Nehaniv, editor, Lectures on Mathematics in the Life Sciences, American Mathematical Society, Vol. 26: 47-91.

Miller, Julian and Thomson, Peter (2000). Cartesian Genetic Programming. EuroGP, 1st International Conference on Genetic Programming, pp. 15-17.

Mitchell M. (1996). Computation in cellular automata: a selected review. Santa Fe Institute Working Paper 9609 -074.

Ray, Thomas S. (1991). An approach to the synthesis of life. In Langton, Christopher, Taylor, Charles, Farmer, J. Doyne, and Rasmussen, Steen (editors). Artificial Life II, SFI Studies in the Sciences of Complexity. Volume X. Redwood City, CA: Addison-Wesley, pp. 371-408.

Ray, Thomas S. (1994). An evolutionary approach to synthetic biology: Zen and the art of creating life. Artificial Life, 1(1/2), pp. 179-209.

Resnick, Mitchel (1994). Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds. Cambridge, MA: MIT Press.

Reynolds, Craig W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. In Computer Graphics 21(4) (SIGGRAPH 87 Conference Proceedings), pp. 25-34.

Shelling, Thomas C. (1978). Micromotives and Macrobehavior, New York: W. W. Norton.

Schwefel, H.P. (1981). Numerical optimization of computer models. Chichester: Wiley.

Taylor, C. and Jefferson, D. (1995). Artificial Life as a Tool for Biological Inquiry, in Langton C. (ed.), Artificial Life- An Overview, MIT Press, pp. 1-14.

Turing, Alan (1952). The chemical basis of morphogenesis. Philosophical Transactions of the Royal Society of London, B (237): 37-42.

Wheeler, M., Bullock, S., Di Paolo, E., Noble, J., Bedau, M., Husbands, P., Kirby, S., and Seth, A. (2002). The view from elsewhere: Perspectives on ALife modeling. Artificial Life 8, to appear.

The references have been prepared using:

American Psychological Association (1994) *Publication manual of the American Psychological Association* (4th ed). Washington, DC: American Psychological Association.

Appendix A: Mini-project declaration

Appendix B: Statement of information search strategy

A formal search was planned so that relevant literature could be assessed and reviewed while performing this research. The plan for searching consisted determining types of literature to search and which search tools to use. This statement appears in the format provided by The University of Birmingham guide to writing mini-project and project reports (2001).

Parameters of literature search

Forms of Literature

The types of literature listed below were determined to be the most relevant to this research. The determination was based on performing an informal cited reference search and analyzing a representative sampling of the references retrieved. The most important types of literature are listed in the order of importance:

- Journal Articles
- Conference Papers
- Theses
- Books

The majority of the information reviewed for this research was found in Journals and Conference Papers. Artificial Life is a relatively new field and there are not that many books published on the subject. There were several informative books on evolutionary concepts and multicellular development.

Geographical/Language Coverage

North America and the Western Europe were determined to be the primary locations for sources. English and French languages were both possible with English being the primary language for the search. Other languages would have to be evaluated using abstracts in English or via translations if it was determined to be an important source.

Retrospective coverage and currency

This area of research is fairly new so a search covering the ten last years should be sufficient. This range should be appropriate to find any work using artificial life simulations to research the development of multicellularity.

Appropriate Search Tools

The following indexes were determined to be important search tools: *Engineering Index* (EDINA Ei Compendex), *Science Citation Index, and Internet Searches.*

The above tools provided access to journal articles, conference proceedings, and theses.

Search Statements

The following queries were used in searches

multicell* AND evolution* AND (artificial life OR alife)

genetic programming AND (artificial life OR alife)

These queries will be modified if recall is too high or low. Two different queries have been identified it is necessary to research both multicellular development in artificial life and genetic programming.

Brief evaluation of the search

This search resulted in a broad range of papers that were deemed relevant. The Science Citation Index retrieved the most relevant resources. The Internet also proved to be a valuable tool since many artificial life simulations were available as demonstrations.

Appendix C: Java API Documentation

Appendix D: Java Source Code Excerpt

This appendix shows code for the decoding of the phenotype using Cartesian Genetic Programming. A complete listing of source code will be provided on the Internet on the student web page:

http://studentweb.cs.bham.ac.uk/~msc84jar/

This method decodes the genome to create the phenotype:

```
private void executePhenotype(Chemical[] chemicals, Chemical chemReleased)
  {
  int currentGene = numGenes-1; /* this assigns the entry point for decoding the
genome*/
  boolean actionFound = false;
  while (!actionFound)
  {
    switch (gene[currentGene][0]) /* evaluate the function of the current */
    {
    case 0 :
      /* do nothing */
      actionFound = true;
      break;
    case 1 :
      /* move towards the object given in the first node of the gene */
      {
      move(true, chemicals[gene[currentGene][1]]);
      actionFound = true;
      }
      break;
    case 2 :
      /* move away from object given in the first node of the gene */
      {
      move(false, chemicals[gene[currentGene][1]]);
      actionFound = true;
      }
      break;
    case 3 :
      /* if the substance in node 0 is present, then do node 2 else node 3 */
      {
      if (chemicals[gene[currentGene][1]].getLevel(x,y) > chemThreshold)
        currentGene = gene[currentGene][2];
      else
        currentGene = gene[currentGene][3];
      }
      break;
    case 4 :
      /* if the substance in node 0 is present, then do node 3 else node 2 */
      {
      if (chemicals[gene[currentGene][1]].getLevel(x,y) > chemThreshold)
        currentGene = gene[currentGene][3];
      else
        currentGene = gene[currentGene][2];
      }
      break;
    case 5 :
      /* if the energy level is above upper threshold then do node 2 else node 3 */
      {
      if (energy > energyUpperThreshold)
        {
        currentGene = gene[currentGene][2];
        }
      else
        currentGene = gene[currentGene][3];
      }
    case 6 :
      /* if the energy level is lower than lowerthreshold then do node 2 else node 3
*/
      {
      if (energy < energyLowerThreshold)
        {
        currentGene = gene[currentGene][2];
        }
      else
        currentGene = gene[currentGene][3];
      }
      break;
    case 7 :
      /* perform node 2 and divide afterwards */
      {
      divideNow = true;
      currentGene = gene[currentGene][2];
      }
      break;
    case 8 :
      /* perfrom node 2 and signal */
      {
      release(chemReleased,prevX,prevY);
      currentGene = gene[currentGene][2];
      }
      break;
    default:
      System.out.println("Error: could not decode genome");
      break;
    }
  }
 }
```