CELLULAR AUTOMATA RULES GENERATOR FOR MICROBIAL COMMUNITIES

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Melissa Marie Quintana

December 2010

CELLULAR AUTOMATA RULES GENERATOR FOR MICROBIAL COMMUNITIES

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Melissa Marie Quintana

December 2010

Approved by:

Keith Evan Schubert, Advisor, School of Computer Science and Engineering	Date
Ernesto Gomez	
Richard Botting	

ABSTRACT

Cellular automata were first used for work on self-reproducing automata which is ideal for the goal of simulating microbial communities . In addition, cellular automata have the benefit of being discrete, which enables the simulation of discrete-time signals representing the time intervals of the changing of growth that occurs within microbial communities. The cellular automaton called the "Game of Life" is an example that is similar to how cellular automata are used to represent microbial communities . Currently, this concept is ideal for representing how each cell within a cellular automaton representing a microbial community should be affected based on the conditions of the surrounding elements found within its natural environment.

The methods provided in this thesis can be used to derive an approximation of the value ranges that represent the rules of cellular automata of a reasonable size through visual identification utilizing image processing. Currently the number ranges of the underlying rules are identifiable through the programs of the simulations and not visibly identifiable through image processing or through patterns. Providing a method that will associate the rules with patterns through image processing will provide an alternative method to the current process of identifying them through expert recognition due to years of experience. The ability to identify an approximation of the number range within cellular automata will aid in the process of associating them with microbial communities of similar growth patterns. A new method is needed because expert recognition is not a practical solution. This thesis project aims to promote the understanding of the underlying rules and allow for visual identification that will aid in the process of better

understanding microbial communities . Overall this project will provide a new method that will enable a visible approximation of the number ranges and radius involved within the cellular automata simulations.

ACKNOWLEDGEMENTS

I would like to acknowledge everyone who supported my success with this thesis project. My deepest gratitude goes out to my advisor, Dr. Keith Schubert, for the guidance that he provided and for the time that he devoted to assist me. I would also like to express my gratitude towards the other committee members, Dr. Ernesto Gomez, Dr. Richard Botting, and my graduate coordinator, Dr. Josephine Mendoza, for their assistance and availability when needed throughout the project. Lastly, I want to thank Jane Curnutt for guiding me in the right direction and Dr. Penelope Boston for being such an inspiration.

DEDICATION

I dedicate this project to my family who has always believed in me and whose unconditional love has been my strength. My father, Richard Quintana, taught me to dream big, just as he did. My mother, Nita Halcomb, and sister, Melody Quintana, inspired me and instilled in me the courage to achieve my education. My Grandparents, Celia and Ignacio Quintana, truly supported my goals and made me promise to never give up. And, Lisa Rojas, provided me with support in every way to ensure that I accomplished my dream of achieving a Master of Science in Computer Science.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Background

Conway's cellular automaton, "Game of Life", is a simple comparison to the concept of growth within microbial communities. The cell occupancy in the "Game of Life" is determined by specified conditions known as the rules. The behavior of the cellular automaton is specified in terms of each cell's local relation to its surrounding neighbors. The conditions determining the rules of the "Game of Live" are as follows. A live cell can only stay alive if it is surrounded by two or three live neighbors. If the cell is dead and is surrounded by three live neighbors, then it becomes alive. And lastly, a live cell will die if it has fewer than two or greater than three live neighbors [7].

Just as the "Game of Life" has rules that act as conditions for the reaction of the individual cells within the grid, so does cellular automata that resemble microbial communities. Although the conditions may vary, the concept is still the same [9]. A live cell will die due to under-population if it has less than a specified number of neighbors or die due to overcrowding if it has more than another specified number of neighbors. For example, a garden of plants most likely wouldn't die if they were provided with enough water which would result in what is known as stabilization. But as the plant colony grows and the water resources become fewer, some of the

garden will die due to overcrowding. If the remaining plants left in the garden are too few to absorb water and reproduce, they would die due to under-population. The conditional rules of cellular automata, that resemble the microbial communities, represent the essential elements of its natural environment.

1.2 Statement of the Problem

Currently, a good guess of the number range can be made through expert analysis. Years of experience is needed to best determine the value range within a cellular automaton. A process that allows an approximation of the number ranges within a cellular automaton utilizing image processing or pattern association would provide a more practical method. My thesis provides a form of image processing as program output that will enable an attempt to approximate the number range of cellular automata and the radius of effect. This is a more practical method than the current method of expert analysis.

1.3 Purpose

Extreme environments that were thought to be void of life due to the presence of hostile elements have proven otherwise. After much exploration, it has become apparent that life forms can and do exist in environments that would be detrimental to most living species [2]. The methodological process of learning about microbial communities lies within the scientific objective to learn about life in space or in areas of our planet heavily affected by climate change. Cellular automata have been the focus of much of the scientific study in the quest to understand microbial communities

[4]. Currently, there is a need for a method that extracts the cellular automata rules which simulate the growth patterns of microbial communities found within extreme environments [5]. The purpose of this study is to provide a visual representation as program output so that the rules and the radius of effect can be estimated.

1.3.1 Significance

The significance of this project is that it provides a method that derives an approximation of the underlying values of the rules and the radius of effect within cellular automata of a reasonable size through visual identification utilizing image processing. Previously the number ranges of the underlying rules were identifiable withih the program code and not visibly identifiable through image processing. The methods provided promote the understanding of the underlying rules and allow for visual identification aiding in the process of better understanding microbial communities [4]. The ability to identify an approximation of the number range will also aid in the process of associating rules of cellular automata with microbial communities of similar growth patterns. The method overall provides a new understanding of how to visibly approximate the number ranges involved within the cellular automata simulations.

1.3.2 Contributions

The sample cellular automata simulations that I used were created by Dr. Keith Schubert and are general to any biological system and appear to resemble bio-patterns related to microbial communities. Each simulation incorporates varying amounts of environmental elements and random instances of life and death resulting in similar

patterns of those found in microbial communities. The random instances of life and death are considered to be the white noise and can range from interference that is natural in nature, caused by human interference or simply a random occurrence. In addition to the white noise, there are values incorporated into the code that represent the conditions of how each individual cell should react in relation to its neighbors. The final results of the simulations are contributed to these values assigned to enforce growth, stabilization, and death. Each program was created utilizing SciLab and was designed to run on a time series of twenty iterations, which is a considerable amount to produce a reasonable size cellular automaton. The values are visible within the code and indicate what surrounding conditions of each cell will cause it to live, die or remain stable. My project output uniquely aims to expose the internal value ranges of dynamic simulations. It contributes to ongoing studies by providing a means of identifying an approximation of the values of a cellular automaton of reasonable size through image processing utilizing histogram analysis.

I produced a method to derive an approximation of the number ranges from a cellular automaton. I produced several programs that provide histogram analysis as the program output in hopes to encourage future research that would eventually associate the value ranges with patterns produced by cellular automata or microbial communities. The process consisted of creating four programs utilizing SciLab, three of which Dr. Schubert's simulations were run on. The programs utilities extract the values based on specified radii of each simulation for image processing. The results of the image processing enables histogram analysis so an attempt can be made to determine an approximation of the rule sets and the radius of effect. Now that an

appropriate approximation of the rules and the radius of effect can be identified to support current and ongoing research, I encourage future studies to be conducted to associate the rules with patterns.

1.4 Theoretical Bases and Organization

I expect that my thesis will assist with active research collaborated with universities and government labs. Jane Curnutt, a student of California State University, has been an active participant in the subject matter of learning about microbial communities. Her presentation titled "Patterned Growth in Extreme Environments" was held in Pasadena, CA. for the Third IEEE International Conference on Space Mission Challenges for Information Technology. She concluded her presentation by stating that it would be useful for biologists if a method could be developed that would enable the underlying rules of cellular automata to be recognized within the growth patterns found in photographs of microbial communities [5]. My thesis project is the start of an area of research which aims to associate the rules to growth patterns found in microbial communities by first attempting to approximate them through image processing.

The programs I developed produce output which enables a visual approximation of the rules within cellular automata to be determined. Each were developed to account for every cell within a cellular automaton and extract the values based on specified radii for image processing. The output of each program allows for histogram analysis in an attempt to determine if the number ranges or the radius of effect can be approximated. The program provides a visual representation of the data,

so that an attempt of approximating the number range of cellular automata can be performed resulting in a new method of approximation over the current method of expert recognition which requires years of experience.

I expect that my thesis will benefit others in the current process of understanding microbial communities within extreme environments. The project is expected to promote the understanding of the underlying rules and promote the continued research of visual identification through a means of image processing. It is likely that the underlying values of the rules may be approximated through visual identification of patterns within microbial communities now that methods are available that approximate the rules in reference to patterns found within cellular automata. Overall, the project will aid in the current study of understanding microbial communities by allowing for a better understanding of the underlying rules.

1.4.1 Findings

The findings of this thesis are as follows:

- Cellular automata are credited for the benefit of being discrete [9], which enables the simulation of discrete-time signals representing the time intervals of the changing of growth that occurs within microbial communities. This is explained in section 2.1.

- The images of microbial communities can be converted into matrix format enabling image processing for analysis. Images were prepared using the Scilab Image Processing tool. In the third phase, which involved using images to estimate the rules, I discovered that the images would have to be prepared for

the program. This consisted of analytically determining what points in the photograph would be used as reference points for clipping the picture to provide consistency and eliminate unnecessary imagery. In addition, the size was determined based on the size of the radius used for calculations. A size was chosen that would enable the radius to be evenly distributed to prevent inconsistent calculations. This is explained in sections 3.2 and 3.5.

- Several algorithms served as a form of treatment and were needed for the success of the program. In particular, each of the four programs needed an algorithm that would perform calculations that would iterate through a matrix based on a specified radius or radii. The programs developed within the second and third phase needed an algorithm that would store all cell state values and summations. This algorithm would also have to strategically compare the cell states and restore specific summed values. This is explained in sections 3, 3.2, 3.3, 3.4, and 3.6.

- Segmentation image processing was needed in the second and third phase of the project. This was accomplished by developing algorithms to perform the necessary calculations within the program. The pixel values were not large enough to produce ones and zeros within the segmentation algorithm of the third phase. Modifications were made to the algorithm so that the largest pixel value was identified and used as a multiplication factor within the calculations so that when the summed values were rounded a matrix of ones and zeros was produced. This is explained in sections 3, 3.2, 3.3, 3.4, and 3.6.

- Histogram-based image processing was needed as output for all of the programs.

This enabled a visual analysis of the data to be analyzed. This is explained in sections 1.3.2, 3.3, 3.5, 3.6, 4.1.

- The first phase of the project produced results that would identify the radius of effect. This is explained in sections 4, 4.1, 4.2, 5.
- The second phase of the project produced results that would identify an estimation of the rules from cellular automata. This is explained in sections 4, 4.1, 4.2, 5.
- The Third phase of the project produced results that would identify an estimation of the rules from pictures of a microbial community. This is explained in sections 4, 4.1, 4.2, 5.

1.4.2 A Limitations of the Study

Observing the life of a live microbial community over any period of time throughout its life cycle was a limitation of the study. There was nothing at my disposal that would provide a consistent record of the growth of a microbial community for analysis. Several methods were utilized to overcome this limitation. First, a Cellular Automaton was used to represent microbial communities. Cellular Automata are known to generate patterns that are similar to those generated by microbial communities. Second, iterations of a Cellular Automaton that changed cell state based on specified rules were used in the first and second program to represent the possible growth of a microbial community over time. This method simulates what would be expected of the growth of a microbial community and allows for consistent monitoring throughout its growth over time which is a specified period of twenty iterations. The

same limitation occurred when the project progressed into analyzing pictures using the fourth program developed. There was nothing at my disposal that would provide a consistent record of the growth of a microbial community for analysis. To overcome this limitation, two pictures of a cave wall were used to represent two periods in time that were taken over a four year period.

2. LITERATURE REVIEW

2.1 Background

The 1940s brought about the invention and introduction of cellular automata. The invention of the discrete dynamic system known as cellular automaton is attributed to mathematician Staislaw Marcin Ulam [8]. Using a modeled lattice network, he applied a mathematical abstraction to study the growth of crystals [6]. Cellular automata were introduced by John Von Newman as he used it to work on self-reproducing automata [8]. Ulam suggested to Newman that he use a mathematical abstraction when it became evident that he was having difficulty designing a self-replicating robot. This resulted in the first cellular automaton system [6]. Mathematician John Conway popularized cellular automaton with his development of the "Game of Life" in 1970 [8]. It soon became evident that cellular automata were simple models that were useful in the study of biological processes [6].

Pattern growth is visually evident amongst the different types of microbial communities and even microscopic biology, such as grasses in Negev and Australia and tree clumping at the edge of forests in dry areas. The environmental elements available to the microbial communities are a significant factor of its ability to grow and are considered to be the determining rules associated with their unique growth patterns. The level of accessibility of a particular environmental element within the microbial

community determines whether the microbial community or a subset of it experiences growth, die-out, or stabilization. The differing elemental accessibility factors contribute to sub-communities reacting differently within each microbial community resulting in unique growth patterns.

My thesis topic is at the center of an active research collaborated with other universities and government labs. Dr. Keith Schubert, a Computer Science professor of California State University of San Bernardino, has encouraged fellow professors and students to take an active interest in the study of cellular automata. He has developed cellular automata in an effort to better understand how the underlying rules influence the structure of microbial communities. His simulations incorporate values representing varying amounts of environmental elements and random instances of life and death through time series. The results of his simulations produce patterns similar to those found in microbial communities [4].

Another collaborator, Dr. Penelope Boston is a microbiologist who studies microbial life within extreme environments. Her explorations of extreme environments have provided evidence that microbial organisms are capable of thriving in unlikely places here on earth. Bacterial strings called Snotties and other microbial life forms have been discovered through her explorations [2]. Dr. Penelope Boston is also one of the coauthors of the case for mars [2] and is actively involved in the search for extraterrestrial life [1]. The search for life in Extreme Environments here on earth is gained experience preparing scientists for future searches of life on extraterrestrial bodies when the opportunities should arise [2].

2.2 Similarities

My thesis is similar to Dr. Schubert's and Dr. Boston's approaches to the study of microbial communities because it incorporates the importance of using both cellular automata and onsite samples within the study to be used as a comparison. Cellular automata were used to attempt an estimation of the rules. Test site field sample images were used as the advancement of the project progressed into estimating rules based on the actual microbial images in which the cellular automata resembled.

2.3 Differences

Individually, Dr. Schubert's and Dr. Boston's approach to the study of microbial communities are equally important. The obvious difference was that my project combined both approaches to the study of microbial communities. It was evident through the progression of my thesis project that both methods would be of similar use to develop a confirmation of my thesis findings. In addition, Dr. Schubert studied the rules based on a static image, whereas I studied the rules based on a dynamic image.

3. METHODOLOGY

This study resulted in four separate programs, each of which played an important part of the project as it progressed into three different phases. The first phase of the project consisted of developing the first program to test the accuracy of the algorithm that was used to iteratively calculate through a matrix of known values. Once the accuracy was determined to be correct, the algorithm was used to run through series of a cellular automaton, which resulted in the second program. The second Phase of the project consisted of developing a third program using a similar algorithm in which the estimated output values were tested against the estimated output values of another program developed by Dr. Schubert. An additional algorithm was used to store the calculated values bases on a comparison of the cell states. The third phase of the project consisted of developing a fourth program that would analyze the estimated output values of actual images of a microbial community.

3.1 Samples

The program samples used were two cellular automata programs that were developed by Dr. Keith Schubert. The first of the Cellular Automata, KeithGrowthSimulator[1], was chosen for the second program because the output resembled maize like patterns that are formed by microbial communities, such as desert soil crusts [4]. The cellular

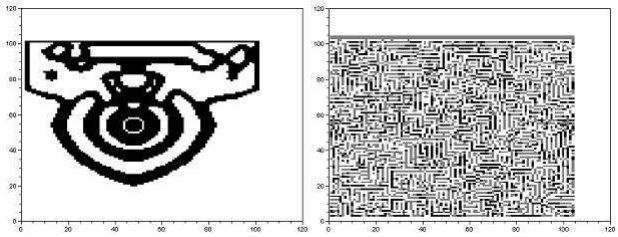

Fig. 3.1: These are the cellular automaton samples that were provided by Dr. Keith Schubert. [3] P. Boston, J. Curnutt, E. Gomez, and B. Strader. To live and die in ca. Retrieved August 20, 2010.

automaton, ExtremeEnvironmentPlantGrowthSimulator[1], was chosen for the third program because it was used by Dr. Keith Schubert to produce estimated values based on a static image. His output was compared to output produced by a program I developed that estimated values based on a dynamic image. The cellular automata output are shown on page 14.

In addition to the cellular automata, two picture samples were used to identify rules. They were taken of a cave wall, in the Cueva De Villa Luz cave near Tabasco, Mexico, in which microbial life was removed and then later observed. They were made available by geologist Louise Hose and are a result of a re-growth experiment which she conducted. The pictures of the cave wall are shown on page 15.

3.2 Treatment

There are two treatments in particular that were needed for preparation. The first treatment that took place involved installing additional software that would be needed to complete the project. This consisted of downloading Scilab 4.1.2, and the Scilab SIP Toolbox. The second treatment involved preparing the images to be inserted into

Fig. 3.2: This is the picture of the Cueva de Villa Luz cave regrowth experiment taken in 1999. [3] P. Boston, J. Curnutt, E. Gomez, and B. Strader. To live and die in ca. Retrieved August 20, 2010.

Fig. 3.3: This is the picture of the Cueva de Villa Luz cave regrowth experiment taken in 2003. [3] P. Boston, J. Curnutt, E. Gomez, and B. Strader. To live and die in ca. Retrieved August 20, 2010.

the program developed during the third phase of the project. Each image was clipped and made to be the same size using the Microsoft Paint program. Additionally, each image was converted into a matrix using the Scilab SIP Toolbox. Any other treatments used throughout the project would be in the form of developing algorithms to be used within the programs.

3.3 Identifying the Radius of Effect

The first phase of the project consisted of developing a program in an effort to identify the radius of effect. The first program was developed to test the accuracy of the calculations that would be used on specific areas of interest within a matrix. The algorithms developed strategically iterate through the cells of a predefined matrix while calculating, storing and comparing the values of radii one, two, and three. A predefined matrix was used so that the accuracy of the calculations could be verified using the program output. The algorithms are designed to start at the center cell of each radius and iterate across each row until the last center cell of the radius is reached within the matrix. It is important to start and end at the center of the cell within the radius so that the entire radius is accounted for. Otherwise, it would extend outside of the matrix boundary and an estimation of the cell values exceeding the boundary would have to be made. This option was not considered because it would have reduced the accuracy of the program output.

As each center cell of the radii are accessed, the values within the entire radius are summed and stored within the index of a vector that matches the number summed. For example, a radius of one would have values stored within each cell of either one

or zero. This meant that it contained a total of nine cells and at the most could sum to the value of nine if each cell contained the value of one. If the total sum was nine, the index of nine within a vector would have a one added to it indicating that one value of nine was found. The algorithm would run for each radii specified within the program and for every series specified within the cellular Automaton. Each series represents a change in time where the rules determine the state of each cell based on the current state of the surrounding neighbor cells within the radius. In addition, the output was designed to be in the form of histograms so that visually identify the radius of effect could be attempted.

Because a matrix of predefined cell states was used, thorough testing for calculation accuracy was possible. Each cell value within the radius that iterated through the matrix was displayed as output. This output was then compared to a printout of the matrix being used to determine that each radius within the matrix was accounted for. A summed value for each radius was also displayed in the output and was compared to a manual count performed using the printed matrix to insure the summations were correct within the program. Histograms were also included in the program output so that a visual identification of the radius of effect could be attempted. Lastly, the total count of each particular summed value was determined from the hard copy of the matrix and compared to total values indexed within the program vector, which was also displayed in the program output. These values were also visually evident in the sequence of the histograms. The code is shown in A.1, starting on page 37.

Once thorough testing was completed, verifying the accuracy of the program calculations, the program was turned into a function and called at every time series within

Dr. Schubert's growth simulator cellular automaton program. This resulted in a second program in which the calculations were performed using a cellular automaton rather than a predefined matrix. The calculations were performed on a total of twenty matrices because there was a time series of twenty specified within the program. The goal was to generate histograms as output for each radius in each series that would provide information about each radius of effect that could be interpreted by others who may not be familiar with reading and understanding Scilab code. The algorithm containing the conditions that would call the function with the cellular automaton named KeithgrowthSimulator[1] is shown in A.2, starting on page 76.

3.4 Identifying the Rules of Cellular Automata

The second phase of the program was inspired by an estimation of the rules contributing to growth, die-out and stabilization within the cellular automaton. The estimated rules were identified by Dr. Keith Schubert of California State University using a program in which he developed named ExtremeEnvironmentPlantGrowthSimulation[1]. His program was designed to provide the estimations based on a cellular automaton simulating iterations that would represent a static image. I developed algorithms that were inserted into Dr. Schubert's cellular automaton that would represent a dynamic image. The algorithms were developed to process calculations and analyze each series with the cellular automaton. Each series represents a phase of growth over time of a microbial community. Analyzing iterations, each as a series of time within the cellular automaton simulation, enabled me to study the results dynamically.

Similar to the first program, the value resulting from the sum of the radius is

indexed into a new matrix. But unlike the others, it is indexed into separate sequenced indices and not accumulatively assigned to the index matching the summed value. This method allows the cell states of two separate matrices to be compared. It also enabled me to compare any two consecutive series of twenty series specified within the program. Although the summed values of the radius of both series are indexed to be referenced, it is the sum of the first series that determines the rule depending on the state of the following series. The actual content of the cell, one or zero, was also noted and indexed sequentially within a vector. A cell state of zero could represent death or stability and a cell state of one could represent life or stability. The comparison of these two values between the first and second series determines whether the state of the cell resulted in life, death or stability. For example, if the state of a particular cell in the first series was zero and then one in the next series, this would determine that the total sum of the radius within the first series would be a rule for life or stability.

The values, known as rules, contributing to whether the cell state resulted in life, death, or stability is written in the cellular automaton code. The overall goal was to generate output that would provide an estimation of the rules that could be interpreted by others that may not be familiar with reading and understanding Scilab code. This program was used to identify whether the estimated rules of a Cellular Automaton representing a static image would be the same or different from a Cellular Automaton representing dynamic series of time. The results were compared to Dr. Schubert's to determine whether there would be a difference between the two or whether the results would prove to be the same. The code is shown in A.3, starting on page 80. The successful completion of this program and analysis lead me to the

next phase of the project in which I determined whether an estimation of the rules can be identified from pictures.

3.5 Identifying the Rules from Pictures

After researching the approximations of the rules within Cellular Automata, I developed a fourth program in an attempt to approximate the rules using pictures. The pictures that were used for this portion of the project were provided by Penny Boston and are of a re-growth experiment that was performed by Louise Hose. Louise Hose has a PH. D. in geology and has actively studied the caves of Mexico for many years. The first picture was taken in April of 1999 and consisted of two portions of a wall, within Cueva De Villa Luz cave near Tabasco, Mexico, that was scraped so that the microbial life was removed. The second picture used was taken at the same test site in September of 2003 and reveals a pattern of microbial life that developed over the years from the areas in which it was first scraped. The pictures are shown on page 15.

The pictures consisted of additional aspects of the cave that were not needed for the project. Microsoft Windows Paint program was used to clip and resize the areas of interest from the pictures. To ensure that picture regions were proportionally equal when clipped, specific features of the cave wall were identified and used as a reference for the top most left corner and bottom most right corner of the clip process. Each picture was resized to ensure that the length and width of the pixels in each were consistent.

When determining the size of the pixels, I considered that I would be using a radius

of one which would be an area containing three rows and three columns. I chose a size of 106 in width and 103 in height for each. This would ensure that the radius of one would distribute evenly from left to right and top to bottom as the iterative calculation is performed. Sizing the image so that the radius would distribute evenly throughout the program was a method used to ensure that the calculations were correct.

Each picture was converted into a matrix using the Scilab Image Processing (SIP) tool. In order for the images to be recognized by the tool throughout the conversion process, the images had to be stored in a specific path within the "Scilab 4.2.1" folder which was created with the install of Scilab. The specific path is Scilab 4.2.1/contrib/siptoolbox/images. After storing the images accordingly, the SIP toolbox was compiled within Scilab and the command, imread, was used within the Scilab command prompt to convert the images. Converting the pictures into matrix form provided a format that enabled my program to run calculations on the images for the purpose of image processing.

After converting the pictures into matrix format, they were ready to be called by the program so that the image processing could be performed. The last program that I developed reads the pixel values and performs image processing. Each process of segmentation image processing used in this phase of the project plays an important role as the final output for analysis is reached. The two forms of image processing that were incorporated into the program are thresholding and histogram-based.

Thresholding is used to create a new matrix containing binary values of one's or zeros after the program reads in each image. Thresholding is done by utilizing an

algorithm that sums the area of a radius, divides it by the maximum cell value found times the radius squared, and rounds the resulting value. The greatest cell value was found to be 0.5411765, which initially produced of matrix of all zeros when rounded. Incorporating the method of multiplying the max cell value into the thresholding algorithum provided values of ones and zeros when rounded. So that one represented live and zero represented death, the resulting values had to be inverted within the program.

The new matrices were sorted and compared, using the algorithms, so that histogram-based segmentation could be developed as output for analysis. After the thresholding was performed, the comparisons were performed using the earliest image taken in April of 1999 as the first series to represent time within the program and the latest image taken in September of 2003 as the second series to represent time. The algorithm compares each cell value of the first series to the same cell value of the second series and determines if the value contributes to life, death or stabilization. As previously mentioned, it is the calculated value within the first series that is of interest and that is stored into a vector that will be used to produce the final histogram. For example, if the cell in the first series is zero and the same cell position in the second series is one, then the summed total of the specified radius within the first series is summed and stored in a vector that is developed to hold all values that are shown to be a rule contributing to life. This examples summed radius value of the first series is determined to contribute to live because the initial cell state of the first series was zero which represents no life and the following cell state of the second series was one which represents life. The four final histograms produced as output separately store

accumulated values that relate to growth, death, and stabilization for analysis.

3.6 Data Analysis Procedures

A visual analysis of simulated histograms were used to analyze the output of the first phase of the project. The program itself did not produce a simulation of the histograms. The output provided four histograms that were labeled life, death, stable-life, and stable-death for each series that was specified within the program. The summed values were stored in a histogram labeled life if the cell state in the first series was zero and the cell state in the second series was one. The summed values were stored in a histogram labeled death if the cell state in the first series was one and the cell state in the second series was zero. The summed values were stored in a histogram labeled stable-life if the cell state in the first series was one and the cell state in the second series was one. The summed values were stored in a histogram labeled stable-death if the cell state in the first series was zero and the cell state in the second series was zero. This was performed on each of the twenty iterative time series specified and resulted in a total of eighty histograms. These were then categorized by the title of the histogram and placed within Microsoft Power-point to simulate from the first to last as they were produced by the program for a simulated visual analysis.

Normalization was used to analyze the output of the second and third phase of the project. Similar to the first phase of the project, the output provided histograms that were labeled life, death, stable-life, and stable-death. The difference would be that the code used in the second and third phase produced four histogram rather than a histogram for each time series. The summed values of the first series were stored

accumulatively in an index that matched the summed value of a histogram labeled life if the cell state in the first series was zero and the cell state in the second series was one. The summed values of the first series were stored accumulatively in an index that matched the summed value of a histogram labeled death if the cell state in the first series was one and the cell state in the second series was zero. The summed values of the first series were stored accumulatively in an index that matched the summed value of a histogram labeled stable-life if the cell state in the first series was one and the cell state in the second series was one. The summed values of the first series were stored accumulatively in an index that matched the summed value of a histogram labeled stable-death if the cell state in the first series was zero and the cell state in the second series was zero. The normalization process was manually performed as each visible value range was noted and compared from the four resulting histograms. The values that were found in more than one histogram were eliminated according to the histograms being compared. Specifically, the histograms labeled die and stable-die were compared. If a particular value was found to be in both histograms, the value was eliminated from the stable-die. Similarly, the histograms labeled life and stable-life were compared. If a particular value was found to be in both histograms, the value was eliminated from the stable-life. The remaining values are the estimated value ranges representing the rules that contribute to life, death or stability. The output results for the second phase can be seen on pages 25-26. The output results for the third phase can be seen on pages 31-32.

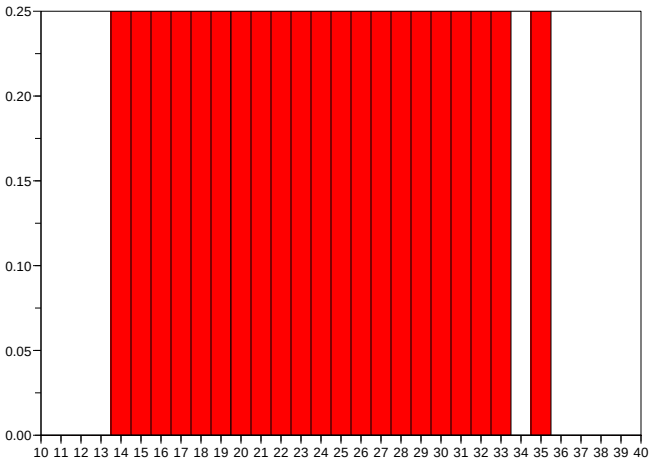

Fig. 3.4: This is the life histogram output of the second phase that was produced from cellular automata.

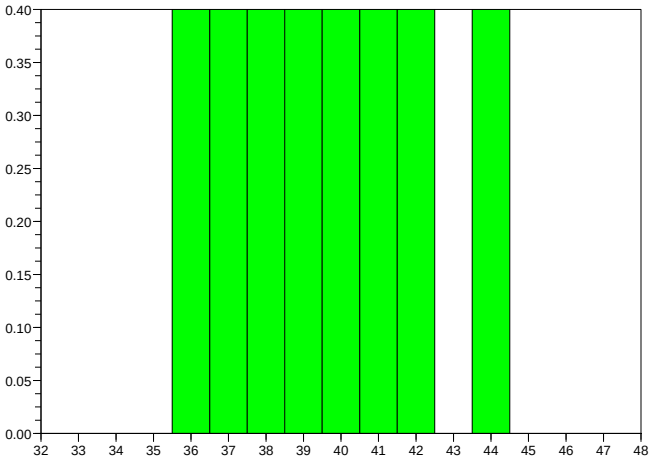

Fig. 3.5: This is the death histogram output of the second phase that was produced from cellular automata.

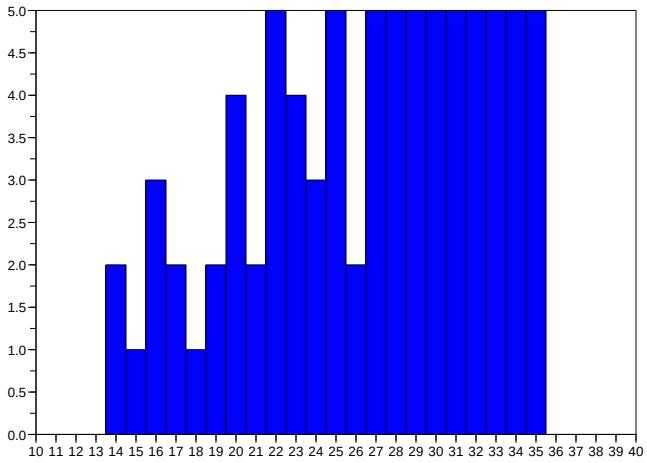

Fig. 3.6: This is the stable life histogram output of the second phase produced from cellular automata.

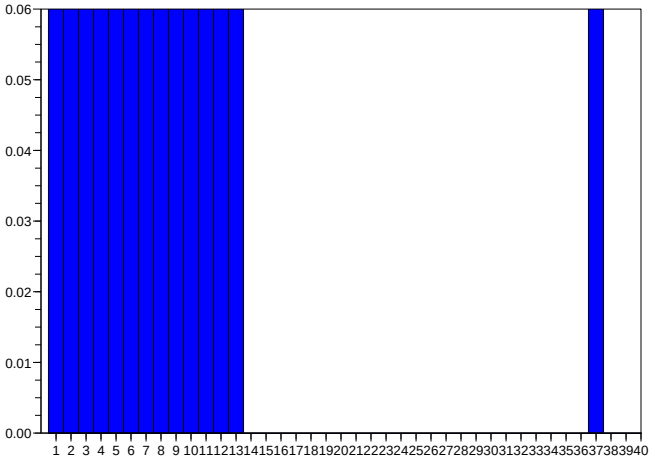

Fig. 3.7: This is the stable death histogram output of the second phase produced from cellular automata.

4. RESULTS AND DISCUSSION

The results in each phase of the project were analyzed as a consequence of the programs. The two things that were kept in mind throughout the analysis process were that I was attempting to visually identify the radius of effect and an estimation of the rules. All three phases produced results that, combined, provide a small addition to the infinite process of learning about how environmental factors affect microbial communities.

4.1 Presentation of the Findings

The first phase of the project produced a visual representation of the radius of effect that was specified within the cellular automaton that was used. A visual analysis of the simulated histograms used to analyze the output revealed that in some of the histograms, the columns nearest to the left most and right most portions of the histogram did not fluctuate enough or in some cases at all to the bottom of the histogram. This would indicate that the histograms were not fully revealing the rules and that the radius used was not the radius specified within the program. One particular simulation that held the results of the calculations, using a radius of three, showed that all of the columns nearest to the left most and right most portions of the histogram did fluctuate all the way to the bottom. This visual representation in

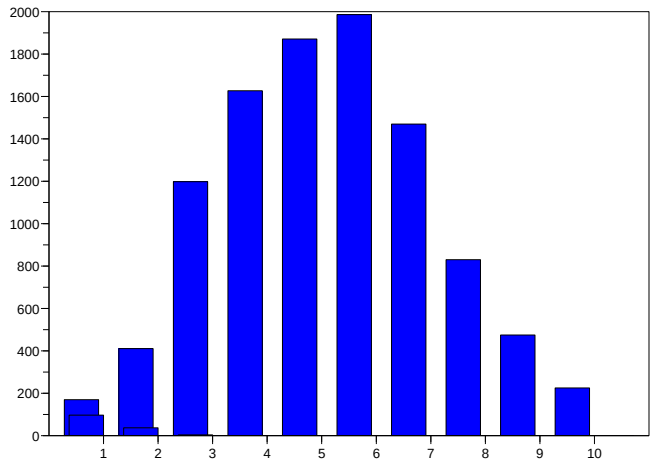

Fig. 4.1: The outer edges of the histogram do not fluctuate to zero indicating that not all of the rules are visible.

particular indicated that all of the rules specified within the program were identified and visually represented within the program output image analysis. The results can be viewed on pages 28-30.

The second phase of the project produced a method that can be used to determine the rules based on the comparison of the value ranges provided through histogram outputs. A histogram-based analysis was conducted by eliminating shared values from the stable-die and stable-live histograms. The results of the manual assessment of the values revealed that there were values specific to each histogram that weren't found in the others. The remaining range of values found within the histograms labeled live and dead are the estimated value ranges representing the rules that result in each of the cell states. The remaining range of values found in the stable-live and the stable-death combined are the estimated value ranges representing the rules that

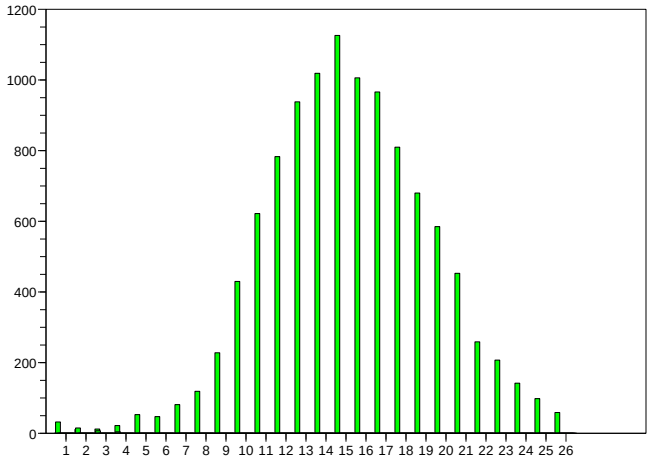

Fig. 4.2: The outer edges of the histogram do not fluctuate to zero indicating that not all of the rules are visible.

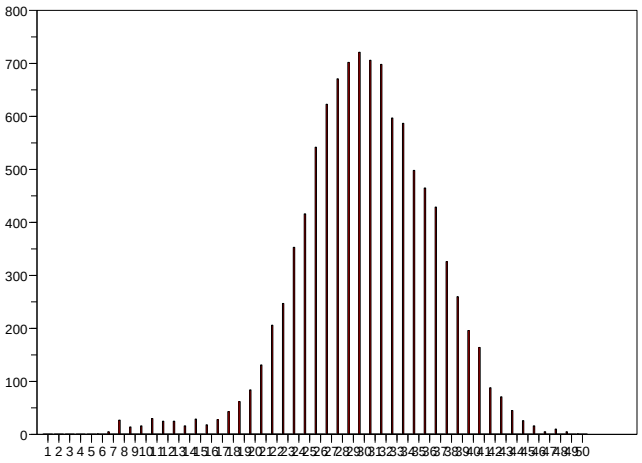

Fig. 4.3: The outer left edge of the histogram do fluctuate to zero indicating that all of the rules are visible.

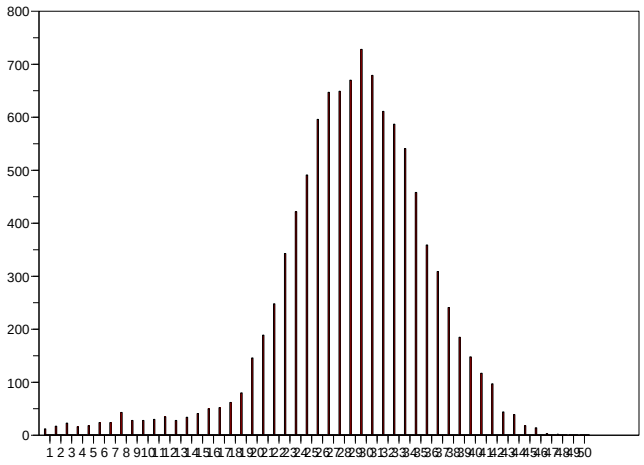

Fig. 4.4: The outer right edge of the histogram do fluctuate to zero indicating that all of the rules are visible.

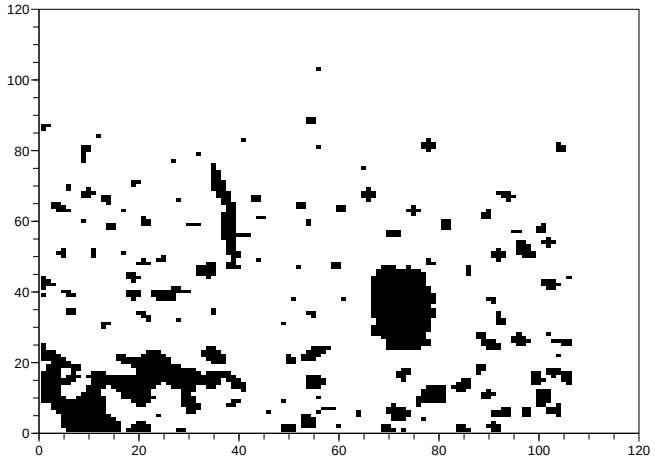

Fig. 4.5: The output of the picture taken in 1999 of the cave wall shows the initial starting state.

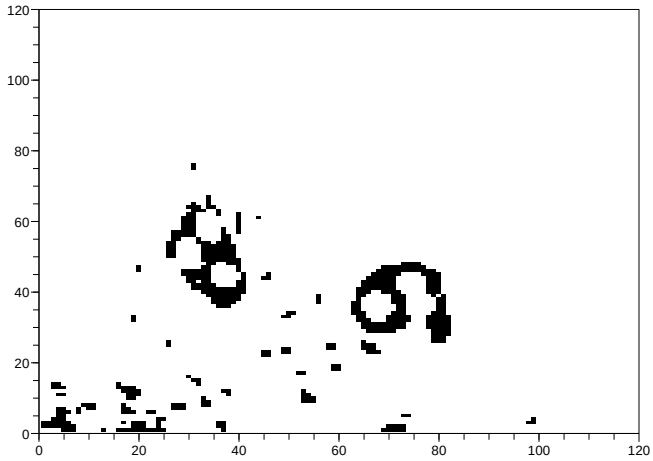

Fig. 4.6: The output of the picture taken in 2003 of the cave wall shows the resulting growth pattern that produced high ranges contributing to life and death due to the extended length of time in which the picture was taken.

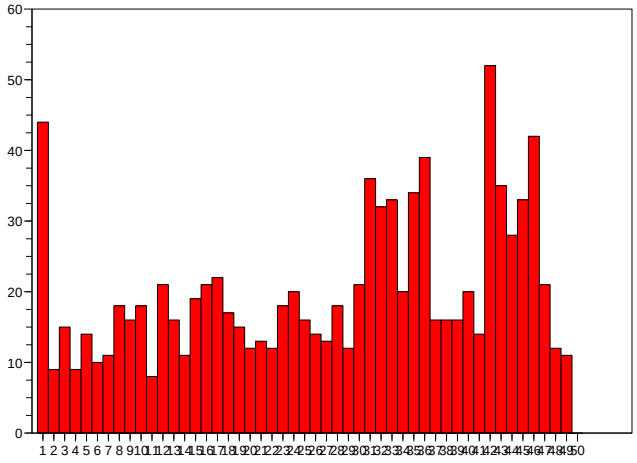

Fig. 4.7: This is the stable death histogram output of the third phase produced from the 1999 picture.

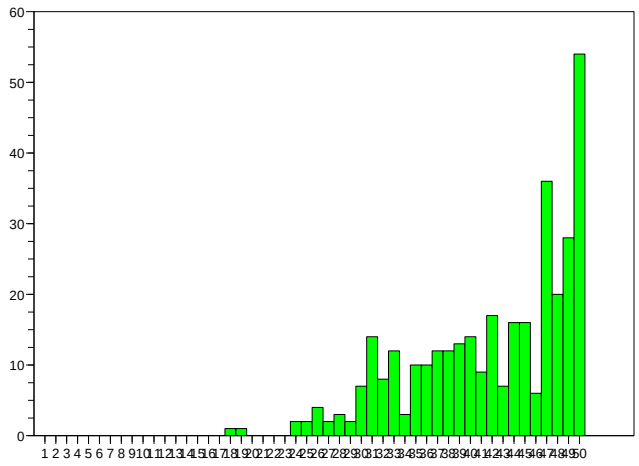

Fig. 4.8: This is the stable death histogram output of the third phase produced from the 2003 Picture.

result in a cell state of stability.

The third phase of the project reused the method produced in the second phase, which determined the rules based on a comparison of the value ranges provided through histogram outputs. From the output, I found that I would not be able to eliminate the shared values from the stable-die and stable-live histograms because the life and death histograms shared the same values in the higher ranges that were summed within the program. After developing several programs that produced the same results, it became evident that the results were accurate and the time series was not. There are portions of the cave in which a portion with no life, surround entirely by life, becomes alive. And, there are portions of the cave in which a portion with life, surround entirely by life, becomes dead. These pictures can be viewed on pages 30-31. This resulted in life and death values falling into the same higher value range. The time series should be close enough so that the life and death value ranges can

be identified uniquely as either a low or high value range. The pictures used as time series were taken around four years apart. A smaller time series would have to be used in order for the rules to be adequately identified. The program output results can be viewed on pages 31-32.

4.2 Discussion of the Findings

I feel that the thesis findings are an important step in the direction of continuing to learn more about several aspects pertaining to microbial communities. Identifying the radius of effect may be the start of understanding the range of an area in which microbial communities reside. Identifying an approximation of the rules may be the start of understanding how much and of what type of environmental elements influence the growth of microbial communities. And as a whole, the methods of visual identification provided through image analysis may be a valuable resource in the process of learning to correlate patterns and rules based on visual identification of microbial communities within their environments as we increase our knowledge of them and how the environmental elements affect their growth.

5. CONCLUSION

I feel that this project could reach its fullest potential if there were more picture samples used. For example, the cellular automata uses a series of time that incorporates more than a series of two which enables a study of the growth from the beginning of the growth, to an end point of the growth, and anywhere in between. The two pictures that were used can only represent the beginning of the growth to an end point, but nothing in between. In addition to finding that some of the same rules were found to represent both life and death, there may have been unrecognized rules due to the portion of time represented by the pictures and within the cellular automata because a small sample of only two consecutive time series were used. It is likely that not every rule may have occurred from one series to another that was compared. After analyzing the program output of the fourth phase I determined that an extended comparison of the study is needed using more picture samples because the time series is not a close enough representation of time to produce an accurate estimation of the rules, which resulted in the same rules contributing to life and death, and because there also may have been rules that were not identified. More than two pictures would allow for continued research of estimating the rules using pictures.

I also feel that additional tests using pictures of various time series will aid in understanding what amount of time should represent a time series within cellular

automata. Iterations in cellular automata represents categorized series of time. But what is or should be the span of time that equals a series within cellular automata? For example, the pictures that I used were nearly four years apart. Should each series of growth within cellular automata be based on a day, week, month, year, or years to respond to the surrounding elements? How much time passed should represent a series of cellular automata? Studying the differences in time, in which the pictures are taken, will enable more testing of estimated rules by visually comparing the cellular automata output of each series to the output produced using the pictures taken. A variety of picture samples taken at various times will enable the continued study of estimating rules.

Understanding the spatial properties of microbial communities is of major importance to the scientific community interested in continuing to learn more. Continued testing will increase knowledge and potentially, over time and with experience, will evolve into expert recognition of being able to correlate patterns with rules with an understanding of which and how much of the surrounding environmental factors have contributed to the growth. This would be ideal and beneficial for current earth and future extraterrestrial explorations for new life. My hope is that this project will inspire others to continue researching using the methods provided and lead to a much better understanding of life forms in extreme environments.

APPENDIX A

MY CODE

The First Program was developed to specify a radius of one, two, and three and to perform calculations on each as it iterates through a matrix of a cellular automaton. This program was then turned into a function to be used within the second program developed. The second program consisted of code which calls the function at every time series of the cellular automata so that the calculations can be performed on every grid produced. This resulted in the identification of the radius of effect. The third program resulted in an estimation of the rules of a cellular automaton. The fourth program resulted in an estimation of the rules based on pictures of a microbial community.

A.1 First Phase - Part One

```
A=[1 1 0 1 0 0 1 1 1;
1 0 0 0 0 0 1 0 0;
1 1 1 0 0 1 0 0 1;
1 1 0 1 1 0 0 0 0;
1 1 0 1 0 0 1 1 0;
0 0 1 0 1 1 0 0 1;
1 0 0 0 1 0 0 1 1;
1 1 1 1 0 0 0 1 0;
0 0 0 1 0 0 0 0 0];

'CENTER CALCULATIONS OF 3X3 RADIUS'
```

```
q=10;
Rad3Vector=zeros(q,q);


Rad3ColStart=1;
Rad3ColEnd=3;


Rad3f=1;
Rad3g=3;


for Rad3col=A(1:$-2,Rad3ColStart:$-2)
  Rad3x=A(1:3,Rad3ColStart:Rad3ColEnd);
  Rad3ColStart = Rad3ColStart + 1;
  Rad3ColEnd = Rad3ColEnd + 1;


  Rad3RowStart=1;
  Rad3RowEnd=3;


   for Rad3row=A(:,1:$-2)
       Rad3x=A(Rad3RowStart:Rad3RowEnd,Rad3f:Rad3g)
       Rad3b=sum(Rad3x)
       if Rad3b == 0 then
         Rad3b = 10;
       end
```

```
        Rad3RowStart = Rad3RowStart + 1;
        Rad3RowEnd = Rad3RowEnd + 1;
        Rad3Vector(Rad3b)= Rad3Vector(Rad3b)+1;

    end
  Rad3f = Rad3f + 1;
  Rad3g = Rad3g + 1;
end

'CORNER CALCULATIONS OF 3X3 RADIUS'
Rad2x2=A(1:2,1:2)
Rad3b = sum(Rad2x2)
if Rad3b == 0 then
    Rad3b = 10;
end
Rad3Vector(Rad3b)= Rad3Vector(Rad3b)+1;

Rad2x2=A($-1:$,1:2)
Rad3b=sum(Rad2x2)
if Rad3b == 0 then
    Rad3b = 10;
end
Rad3Vector(Rad3b)= Rad3Vector(Rad3b)+1;
```

```
Rad2x2=A(1:2,$-1:$)
Rad3b=sum(Rad2x2)
if Rad3b == 0 then
    Rad3b = 10;
end
Rad3Vector(Rad3b)= Rad3Vector(Rad3b)+1;

Rad2x2=A($-1:$,$-1:$)
Rad3b=sum(Rad2x2)
if Rad3b == 0 then
    Rad3b = 10;
end
Rad3Vector(Rad3b)= Rad3Vector(Rad3b)+1;

'TOP EDGE OF 3X3 RADIUS-including right edge'
Radius3ColStart = 1;
Radius3ColEnd = 3;

Radius3RowStart=1;
Radius3RowEnd=3;

'TOP EDGE OF 3X3 RADIUS'
```

```
for Radius3col=A(2:$-1,Radius3ColStart:$-2)
  Radius3UpperEdge=A(1:2,Radius3ColStart:Radius3ColEnd)
  Radius3ColStart = Radius3ColStart + 1;
  Radius3ColEnd = Radius3ColEnd + 1;
  Rad3b=sum(Radius3UpperEdge)
  if Rad3b == 0 then
    Rad3b = 10;
  end
  Rad3Vector(Rad3b)= Rad3Vector(Rad3b)+1;

'RIGHT EDGE OF 3X3 RADIUS**************'
       Radius3x=A(Radius3RowStart:Radius3RowEnd,$-1:$)
       Radius3RowStart = Radius3RowStart + 1;
       Radius3RowEnd = Radius3RowEnd + 1;
       Radius3b=sum(Radius3x)
       if Radius3b == 0 then
         Radius3b = 10;
       end
       Rad3Vector(Radius3b)= Rad3Vector(Radius3b)+1;
end

'BOTTOM EDGE OF 3X3 RADIUS -including left edge'
```

```
Radius3ColStart = 1;
Radius3ColEnd = 3;


Radius3RowStart=1;
Radius3RowEnd=3;


'BOTTOM_EDGE_OF_3X3_RADIUS'
for Radius3col=A(2:$-1,Radius3ColStart:$-2)
  Radius3x=A($-1:$,Radius3ColStart:Radius3ColEnd)
  Radius3ColStart = Radius3ColStart + 1;
  Radius3ColEnd = Radius3ColEnd + 1;
  Rad3b=sum(Radius3x)
  if Rad3b == 0 then
    Rad3b = 10;
  end
  Rad3Vector(Rad3b)= Rad3Vector(Rad3b)+1;


'LEFT_EDGE_OF_3X3_RADIUS'
  Radius3x=A(Radius3RowStart:Radius3RowEnd,1:2)
  Radius3RowStart = Radius3RowStart + 1;
  Radius3RowEnd = Radius3RowEnd + 1;
  Radius3b=sum(Radius3x)
  if Radius3b == 0 then
```

```
    Radius3b = 10;
  end
  Rad3Vector(Radius3b)= Rad3Vector(Radius3b)+1;
end


Rad3Vector(:,1)
subplot(3,2,1)
bar(Rad3Vector,7,'Blue')
xtitle("3X3 Radius")


'CENTER CALCULATIONS OF 5X5 RADIUS'


q=26;
Rad5Vector=zeros(q,q);


Rad5ColStart=1;
Rad5ColEnd=5;


Rad5f=1;
Rad5g=5;


for Rad5col=A(1:$-4,Rad5ColStart:$-4)
  Rad5x=A(1:5,Rad5ColStart:Rad5ColEnd);
```

```
  Rad5ColStart = Rad5ColStart + 1;
  Rad5ColEnd = Rad5ColEnd + 1;


  Rad5RowStart=1;
  Rad5RowEnd=5;


     for Rad5row=A(:,1:$-4)
       Rad5x=A(Rad5RowStart:Rad5RowEnd,Rad5f:Rad5g)
       Rad5b=sum(Rad5x)
       if Rad5b == 0 then
         Rad5b = 26;
       end
       Rad5RowStart = Rad5RowStart + 1;
       Rad5RowEnd = Rad5RowEnd + 1;
       Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


     end
  Rad5f = Rad5f + 1;
  Rad5g = Rad5g + 1;
end


'CORNER CALCULATIONS OF 5X5 RADIUS'
```

```
Rad3x3=A(1:3,1:3)
Rad5b = sum(Rad3x3)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;

Rad3x4=A(1:3,1:4)
Rad5b = sum(Rad3x4)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;

Rad4x3=A(1:4,1:3)
Rad5b = sum(Rad4x3)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;

Rad4x4=A(1:4,1:4)
Rad5b = sum(Rad4x4)
```

```
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad3x3=A($-2:$,1:3)
Rad5b=sum(Rad3x3)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad3x4=A($-2:$,1:4)
Rad5b=sum(Rad3x4)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad4x3=A($-3:$,1:3)
Rad5b=sum(Rad4x3)
if Rad5b == 0 then
    Rad5b = 26;
```

```
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad4x4=A($-3:$,1:4)
Rad5b=sum(Rad4x4)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad3x3=A(1:3,$-2:$)
Rad5b=sum(Rad3x3)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad3x4=A(1:3,$-3:$)
Rad5b=sum(Rad3x4)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
```

```
Rad4x3=A(1:4,$-2:$)
Rad5b=sum(Rad4x3)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;

Rad4x4=A(1:4,$-3:$)
Rad5b=sum(Rad4x4)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;

Rad3x3=A($-2:$,$-2:$)
Rad5b=sum(Rad3x3)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;

Rad3x4=A($-2:$,$-3:$)
```

```
Rad5b=sum(Rad3x4)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad4x3=A($-3:$,$-2:$)
Rad5b=sum(Rad4x3)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


Rad4x4=A($-3:$,$-3:$)
Rad5b=sum(Rad4x4)
if Rad5b == 0 then
    Rad5b = 26;
end
Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;


'TOP OUTER EDGE USING 3X5 RADIUS'


Radius5ColStart = 1;
```

```
Radius5ColEnd = 5;


for Radius5col=A(3:$-2,Radius5ColStart:$-4)
  Radius5UpperEdge=A(1:3,Radius5ColStart:Radius5ColEnd)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5UpperEdge)
  if Rad5b == 0 then
    Rad5b = 26;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TOP_INNER_EDGE_USING_4X5_RADIUS'


Radius5ColStart = 1;
Radius5ColEnd = 5;


for Radius5col=A(3:$-2,Radius5ColStart:$-4)
  Radius5UpperEdge=A(1:4,Radius5ColStart:Radius5ColEnd)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5UpperEdge)
```

```
  if Rad5b == 0 then
    Rad5b = 26;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'BOTTOM_OUTER_EDGE_USING_3X5_RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 5;


for Radius5col=A(3:$-2,Radius5ColStart:$-4)
  Radius5x=A($-2:$,Radius5ColStart:Radius5ColEnd)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 26;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'BOTTOM_INNER_EDGE_USING_4X5_RADIUS'
```

```
Radius5ColStart = 1;
Radius5ColEnd = 5;

for Radius5col=A(3:$-2,Radius5ColStart:$-4)
  Radius5x=A($-3:$,Radius5ColStart:Radius5ColEnd)
  Radius5ColStart = Radius5ColStart + 1;
    Radius5ColEnd = Radius5ColEnd + 1;
    Rad5b=sum(Radius5x)
    if Rad5b == 0 then
      Rad5b = 26;
    end
    Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end

'TEST FOR RIGHT OUTER EDGE USING 5x3 RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 5;

for Radius5col=A(Radius5ColStart:$-2,3:$-2)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,$-2:$)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
```

```
  if Rad5b == 0 then
    Rad5b = 26;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TEST FOR RIGHT INNER EDGE USING 5x4 RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 5;


for Radius5col=A(Radius5ColStart:$-2,3:$-2)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,$-3:$)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 26;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TEST FOR LEFT OUTER EDGE USING 5x3 RADIUS'
Radius5ColStart = 1;
```

```
Radius5ColEnd = 5;


for Radius5col=A(Radius5ColStart:$-2,3:$-2)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,1:3)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 26;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TEST_FOR_LEFT_INNER_EDGE_USING_5x4_RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 5;


for Radius5col=A(Radius5ColStart:$-2,3:$-2)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,1:4)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
```

```
    Rad5b = 26;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


Rad5Vector(:,1)
subplot(3,2,2)
bar(Rad5Vector,7,'Green')
xtitle("5X5 Radius")


'CENTER CALCULATIONS OF 7X7 RADIUS'


q=50;
Rad7Vector=zeros(q,q);


Rad7ColStart=1;
Rad7ColEnd=7;


Rad7f=1;
Rad7g=7;


for Rad7col=A(1:$-6,Rad7ColStart:$-6)
  Rad7x=A(1:7,Rad7ColStart:Rad7ColEnd);
```

```
  Rad7ColStart = Rad7ColStart + 1;
  Rad7ColEnd = Rad7ColEnd + 1;


  Rad7RowStart=1;
  Rad7RowEnd=7;


     for Rad7row=A(:,1:$-6)
       Rad7x=A(Rad7RowStart:Rad7RowEnd,Rad7f:Rad7g)
       Rad7b=sum(Rad7x)
       if Rad7b == 0 then
         Rad7b = 50;
       end
       Rad7RowStart = Rad7RowStart + 1;
       Rad7RowEnd = Rad7RowEnd + 1;
       Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
     end
  Rad7f = Rad7f + 1;
  Rad7g = Rad7g + 1;
end


'CORNER CALCULATIONS OF 7X7 RADIUS'


'Top Left Corner'
```

```
Rad4x4=A(1:4,1:4)
Rad7b = sum(Rad4x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

Rad4x5=A(1:4,1:5)
Rad7b = sum(Rad4x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

Rad4x6=A(1:4,1:6)
Rad7b = sum(Rad4x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

Rad5x4=A(1:5,1:4)
```

```
Rad7b = sum(Rad5x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x5=A(1:5,1:5)
Rad7b = sum(Rad5x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x6=A(1:5,1:6)
Rad7b = sum(Rad5x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x4=A(1:6,1:4)
Rad7b = sum(Rad6x4)
if Rad7b == 0 then
```

```
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x5=A(1:6,1:5)
Rad7b = sum(Rad6x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x6=A(1:6,1:6)
Rad7b = sum(Rad6x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


'Bottom Left Corner'


Rad4x4=A($-3:$,1:4)
Rad7b=sum(Rad4x4)
if Rad7b == 0 then
```

```
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad4x5=A($-3:$,1:5)
Rad7b=sum(Rad4x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad4x6=A($-3:$,1:6)
Rad7b=sum(Rad4x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x4=A($-4:$,1:4)
Rad7b=sum(Rad5x4)
if Rad7b == 0 then
    Rad7b = 50;
end
```

```
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x5=A($-4:$,1:5)
Rad7b=sum(Rad5x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x6=A($-4:$,1:6)
Rad7b=sum(Rad5x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x4=A($-5:$,1:4)
Rad7b=sum(Rad6x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
```

```
Rad6x5=A($-5:$,1:5)
Rad7b=sum(Rad6x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

Rad6x6=A($-5:$,1:6)
Rad7b=sum(Rad6x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

'Right Top Corner'

Rad4x4=A(1:4,$-3:$)
Rad7b=sum(Rad4x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
```

```
Rad4x5=A(1:4,$-4:$)
Rad7b=sum(Rad4x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

Rad4x6=A(1:4,$-5:$)
Rad7b=sum(Rad4x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

Rad5x4=A(1:5,$-3:$)
Rad7b=sum(Rad5x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;

Rad5x5=A(1:5,$-4:$)
Rad7b=sum(Rad5x5)
```

```
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x6=A(1:5,$-5:$)
Rad7b=sum(Rad5x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x4=A(1:6,$-3:$)
Rad7b=sum(Rad6x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x5=A(1:6,$-4:$)
Rad7b=sum(Rad6x5)
if Rad7b == 0 then
    Rad7b = 50;
```

```
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x6=A(1:6,$-5:$)
Rad7b=sum(Rad6x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;



'Right Bottom Corner'


Rad4x4=A($-3:$,$-3:$)
Rad7b=sum(Rad4x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad4x5=A($-3:$,$-4:$)
Rad7b=sum(Rad4x5)
if Rad7b == 0 then
```

```
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad4x6=A($-3:$,$-5:$)
Rad7b=sum(Rad4x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x4=A($-4:$,$-3:$)
Rad7b=sum(Rad5x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x5=A($-4:$,$-4:$)
Rad7b=sum(Rad5x5)
if Rad7b == 0 then
    Rad7b = 50;
end
```

```
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad5x6=A($-4:$,$-5:$)
Rad7b=sum(Rad5x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x4=A($-5:$,$-3:$)
Rad7b=sum(Rad6x4)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


Rad6x5=A($-5:$,$-4:$)
Rad7b=sum(Rad6x5)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
```

```
Rad6x6=A($-5:$,$-5:$)
Rad7b=sum(Rad6x6)
if Rad7b == 0 then
    Rad7b = 50;
end
Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;


'TOP OUTER EDGE USING 4x7 RADIUS'


Radius7ColStart = 1;
Radius7ColEnd = 7;


for Radius7col=A(:,Radius7ColStart:$-6)
  Radius7UpperEdge=A(1:4,Radius7ColStart:Radius7ColEnd)
  Radius7ColStart = Radius7ColStart + 1;
  Radius7ColEnd = Radius7ColEnd + 1;
  Rad7b=sum(Radius7UpperEdge)
  if Rad7b == 0 then
    Rad7b = 50;
  end
  Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
end
```

```
'TOP CENTER EDGE USING 5X7 RADIUS'


Radius7ColStart = 1;
Radius7ColEnd = 7;


for Radius7col=A(:,Radius7ColStart:$-6)
  Radius7UpperEdge=A(1:5,Radius7ColStart:Radius7ColEnd)
  Radius7ColStart = Radius7ColStart + 1;
  Radius7ColEnd = Radius7ColEnd + 1;
  Rad7b=sum(Radius7UpperEdge)
  if Rad7b == 0 then
    Rad7b = 50;
  end
  Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
end


'TOP INNER EDGE USING 6X7 RADIUS'


Radius7ColStart = 1;
Radius7ColEnd = 7;


for Radius7col=A(:,Radius7ColStart:$-6)
  Radius7UpperEdge=A(1:6,Radius7ColStart:Radius7ColEnd)
```

```
  Radius7ColStart = Radius7ColStart + 1;
  Radius7ColEnd = Radius7ColEnd + 1;
  Rad7b=sum(Radius7UpperEdge)
  if Rad7b == 0 then
    Rad7b = 50;
  end
  Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
end


'BOTTOM_OUTER_EDGE_USING_4X7_RADIUS'


Radius7ColStart = 1;
Radius7ColEnd = 7;


for Radius7col=A(:,Radius7ColStart:$-6)
  Radius7x=A($-3:$,Radius7ColStart:Radius7ColEnd)
  Radius7ColStart = Radius7ColStart + 1;
  Radius7ColEnd = Radius7ColEnd + 1;
  Rad7b=sum(Radius7x)
  if Rad7b == 0 then
    Rad7b = 50;
  end
  Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
```

```
end

'BOTTOM_CENTER_EDGE_USING_5X7_RADIUS'

Radius7ColStart = 1;
Radius7ColEnd = 7;

for Radius7col=A(:,Radius7ColStart:$-6)
  Radius7x=A($-4:$,Radius7ColStart:Radius7ColEnd)
  Radius7ColStart = Radius7ColStart + 1;
    Radius7ColEnd = Radius7ColEnd + 1;
    Rad7b=sum(Radius7x)
    if Rad7b == 0 then
      Rad7b = 50;
    end
    Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
  end

'BOTTOM_INNER_EDGE_USING_6X7_RADIUS'

Radius7ColStart = 1;
Radius7ColEnd = 7;
```

```
for Radius7col=A(:,Radius7ColStart:$-6)
  Radius7x=A($-5:$,Radius7ColStart:Radius7ColEnd)
  Radius7ColStart = Radius7ColStart + 1;
    Radius7ColEnd = Radius7ColEnd + 1;
    Rad7b=sum(Radius7x)
    if Rad7b == 0 then
      Rad7b = 50;
    end
    Rad7Vector(Rad7b)= Rad7Vector(Rad7b)+1;
end


'TEST FOR RIGHT OUTER EDGE USING 7x4 RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 7;


for Radius5col=A(Radius5ColStart:$-2,4:$-3)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,$-3:$)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 50;
  end
```

```
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TEST FOR RIGHT CENTER EDGE USING 7x5 RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 7;


for Radius5col=A(Radius5ColStart:$-2,4:$-3)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,$-4:$)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 50;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TEST FOR RIGHT INNER EDGE USING 7x6 RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 7;


for Radius5col=A(Radius5ColStart:$-2,4:$-3)
```

```
  Radius5x=A(Radius5ColStart:Radius5ColEnd,$-5:$)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 50;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TEST FOR LEFT OUTER EDGE USING 7x4 RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 7;


for Radius5col=A(Radius5ColStart:$-2,4:$-3)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,1:4)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 50;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
```

```
end


'TEST FOR LEFT CENTER EDGE USING 7x5 RADIU'
Radius5ColStart = 1;
Radius5ColEnd = 7;


for Radius5col=A(Radius5ColStart:$-2,4:$-3)
  Radius5x=A(Radius5ColStart:Radius5ColEnd,1:5)
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 50;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


'TEST FOR LEFT INNER EDGE USING 7x6 RADIUS'
Radius5ColStart = 1;
Radius5ColEnd = 7;//5


for Radius5col=A(Radius5ColStart:$-2,4:$-3)//3:$-2
  Radius5x=A(Radius5ColStart:Radius5ColEnd,1:6)
```

```
  Radius5ColStart = Radius5ColStart + 1;
  Radius5ColEnd = Radius5ColEnd + 1;
  Rad5b=sum(Radius5x)
  if Rad5b == 0 then
    Rad5b = 50;
  end
  Rad5Vector(Rad5b)= Rad5Vector(Rad5b)+1;
end


Rad7Vector(:,1)
//subplot(3,2,3)
f3=scf(3);
bar(Rad7Vector,7,'Red')
xtitle("7X7 Radius")
```

A.2 First Phase - Part Two

```
if plotBuildupOrGrowth==0 then
    scf()
    Matplot(time-buildup);
    if t==1 then radius1 = RadiusCalculations(N)
      scf()
      elseif t==2 then radius2 = RadiusCalculations(N)
```

```
    scf()
    elseif t==3 then radius3 = RadiusCalculations(N)
    scf()
    elseif t==4 then radius4 = RadiusCalculations(N)
    scf()
    elseif t==5 then radius5 = RadiusCalculations(N)
    scf()
    elseif t==6 then radius6 = RadiusCalculations(N)
    scf()
    elseif t==7 then radius7 = RadiusCalculations(N)
    scf()
    elseif t==8 then radius8 = RadiusCalculations(N)
    scf()
    elseif t==9 then radius9 = RadiusCalculations(N)
    scf()
    elseif t==10 then radius10 = RadiusCalculations(N)
    scf()
    elseif t==11 then radius11 = RadiusCalculations(N)
    scf()
    elseif t==12 then radius12 = RadiusCalculations(N)
    scf()
    elseif t==13 then radius13 = RadiusCalculations(N)
    scf()
```

```
      elseif t==14 then radius14 = RadiusCalculations(N)
      scf()
      elseif t==15 then radius15 = RadiusCalculations(N)
      scf()
      elseif t==16 then radius16 = RadiusCalculations(N)
      scf()
      elseif t==17 then radius17 = RadiusCalculations(N)
      scf()
      elseif t==18 then radius18 = RadiusCalculations(N)
      scf()
      elseif t==19 then radius19 = RadiusCalculations(N)
      scf()
      elseif t==20 then radius20 = RadiusCalculations(N)
      scf()
      end
else
    scf()
    Matplot((MaxN+1)-N);
    if t==1 then radius1 = RadiusCalculations(N)
      scf()
      elseif t==2 then radius2 = RadiusCalculations(N)
      scf()
      elseif t==3 then radius3 = RadiusCalculations(N)
```

```
    scf()
    elseif t==4 then radius4 = RadiusCalculations(N)
    scf()
    elseif t==5 then radius5 = RadiusCalculations(N)
    scf()
    elseif t==6 then radius6 = RadiusCalculations(N)
    scf()
    elseif t==7 then radius7 = RadiusCalculations(N)
    scf()
    elseif t==8 then radius8 = RadiusCalculations(N)
    scf()
    elseif t==9 then radius9 = RadiusCalculations(N)
    scf()
    elseif t==10 then radius10 = RadiusCalculations(N)
    scf()
    elseif t==11 then radius11 = RadiusCalculations(N)
    scf()
    elseif t==12 then radius12 = RadiusCalculations(N)
    scf()
    elseif t==13 then radius13 = RadiusCalculations(N)
    scf()
    elseif t==14 then radius14 = RadiusCalculations(N)
    scf()
```

```
      elseif t==15 then radius15 = RadiusCalculations(N)
      scf()
      elseif t==16 then radius16 = RadiusCalculations(N)
      scf()
      elseif t==17 then radius17 = RadiusCalculations(N)
      scf()
      elseif t==18 then radius18 = RadiusCalculations(N)
      scf()
      elseif t==19 then radius19 = RadiusCalculations(N)
      scf()
      elseif t==20 then radius20 = RadiusCalculations(N)
      scf()
      end

  end
end

A = N;
radius = RadiusCalculations(N)
```

A.3 Second Phase

```
 if plotBuildupOrGrowth>0 then
```

```
 scf()  //Melissa added
 Matplot((MaxN+1)-N(TopData-1:BottomData,
 LeftData-1:RightData));
end
    if t==1 then
       if(min(rows,cols)>2*fixedRadius) then
         for row=fixedRadius+1:rows-fixedRadius,
           for col=fixedRadius+1:cols-fixedRadius,
             contents=N(row,col);
             index=sum(N(row-fixedRadius:row+fixedRadius,
             col-fixedRadius:col+fixedRadius))-contents;
             a1(row,col) = contents;
             a2(row,col)= index;
           end,
         end
       end
    if t==2 then
      if(min(rows,cols)>2*fixedRadius) then
       for row=fixedRadius+1:rows-fixedRadius
         for col=fixedRadius+1:cols-fixedRadius
           contents=N(row,col);
           index=sum(N(row-fixedRadius:row+fixedRadius,
```

```
                col-fixedRadius:col+fixedRadius))-contents;///
                b1(row,col) = contents;
                b2(row,col) = index;
              end
            end
          end
        end
end


      for i=1:prod(size(a1))
        if (a1(i)==0 & b1(i)==1) then
          Live(a2(i)+1)=Live(a2(i)+1)+1;
        elseif (a1(i) - b1(i)>0) then
          Die(a2(i)+1)=Die(a2(i)+1)+1;
        elseif (a1(i)==1 & b1(i)==1) then
          StableOne(a2(i)+1)=StableOne(a2(i)+1)+1;
        elseif (a1(i)==0 & b1(i)==0) then
          StableTwo(a2(i)+1)=StableTwo(a2(i)+1)+1;
        end
      end


scf()
bar(Live,1,'Red')
```

```
xtitle("Life")


scf()
bar(Die,1,'Green')
xtitle("Death")


scf()
bar(StableOne,1,'Blue')
xtitle("Stable - Life")


scf()
bar(StableTwo,1,'Blue')
xtitle("Stable - Death")
```

A.4 Third Phase

```
OverallWidth = 106;
OverallHeight = 103;


radius = 1;


maxColor = max(max(Apr1999));
[rows, cols]=size(Apr1999);
```

```
for col=1:cols/radius
  for row=1:rows/radius
    ca1(row,col)=round(sum(Apr1999(row*radius-radius
    +1:row*radius, col*radius-radius+1:col*radius))
    /(maxColor*radius^2));
  end
end

maxColor = max(max(Sept2003));
[rows, cols]=size(Sept2003);

for col=1:cols/radius
  for row=1:rows/radius
    ca2(row,col)=round(sum(Sept2003(row*radius-radius
    +1:row*radius, col*radius-radius+1:col*radius))
    /(maxColor*radius^2));
  end
end

[rows, cols]=size(ca1);
radius=3;
```

```
Live=zeros((2*radius+1)^2+1,1);
Die=zeros((2*radius+1)^2+1,1);
StableOne=zeros((2*radius+1)^2+1,1);
StableTwo=zeros((2*radius+1)^2+1,1);

for col=radius+1:cols-radius
  for row=radius+1:rows-radius
    bordersum=sum(ca1(row-radius:row+radius,
    col-radius:col+radius))+1;
        if (ca1(row,col)==0 & ca2(row,col)==1) then
          Live(bordersum)=Live(bordersum)+1;
        elseif (ca1(row,col) - ca2(row,col)>0) then
          Die(bordersum)=Die(bordersum)+1;
        elseif (ca1(row,col)==1 & ca2(row,col)==1) then
          StableOne(bordersum)=StableOne(bordersum)+1;
        elseif (ca1(row,col)==0 & ca2(row,col)==0) then
          StableTwo(bordersum)=StableTwo(bordersum)+1;
        end
  end
end

scf()
bar(Live,1,'Red')
```

```
xtitle("Live Value Range")


scf()
bar(Die,1,'Green')
xtitle("Death Value Range")


scf()
bar(StableOne,1,'Blue')
xtitle("Stable Value Range - Live")


scf()
bar(StableTwo,1,'Blue')
xtitle("Stable Value Range - Death")
```

APPENDIX B

SAMPLE CELLULAR AUTOMATA CODE

The sample code “Keith Growth Simulator” was used in part two of phase one. The sample code “Extreme Environment Plant Growth Simulator” was used in phase two of the thesis project.

B.5 Sample Code for Part 2 of Phase 2

```
width=100;
height=100;
time=100;

MinSame=2;
MaxSame=6;

MinGrow=2;
MaxGrow=3;

MinN=0;
MaxN=2;

RandGrow=0.02;
RandDie=0.12;

WrapTopToBottom=0;
```

```
TopNutrients=1;
BottomNutrients=0;

WrapLeftToRight=1;

LeftNutrients=0;
RightNutrients=0;

InitRow=1;
InitRowLR=2;
InitRowTB=2;

InitSquare=1;
InitSquareLR=1;
InitSquareTB=3;



plotBuildupOrGrowth=1;

rand('uniform');
rand('seed',getdate("s"));

function BorderSum=CalcBorder(N,row,col,radius)
```

```
  BorderSum=N(row-1,col)+N(row-1,col-1)+N(row,col-1)
  +N(row+1,col-1)+N(row+1,col)+N(row+1,col+1)+
  N(row,col+1)+N(row-1,col+1);
  if radius>1 then
    BorderSum=BorderSum+N(row-2,col-2)+N(row-2,col-1)+
    N(row-2,col)+N(row-2,col+1);
    BorderSum=BorderSum+N(row-2,col+2)+N(row-1,col+2)+
    N(row,col+2)+N(row+1,col+2);
    BorderSum=BorderSum+N(row+2,col+2)+N(row+2,col+1)+
    N(row+2,col)+N(row+2,col-1);
    BorderSum=BorderSum+N(row+2,col-2)+N(row+1,col-2)+
    N(row,col-2)+N(row-1,col-2);
  end
endfunction


WrapWidth=MaxN;


LeftData=WrapWidth+1;
RightData=width+WrapWidth;
TopData=WrapWidth+1;
BottomData=height+WrapWidth;


DataRow=LeftData:RightData;
```

```
DataCol=TopData:BottomData;


LeftWrap=1:WrapWidth;
RightWrap=RightData+1:RightData+WrapWidth;
TopWrap=1:WrapWidth;
BottomWrap=BottomData+1:BottomData+WrapWidth;


LeftDataWrap=LeftData:LeftData+WrapWidth-1;
RightDataWrap=RightData-(WrapWidth-1):RightData;
TopDataWrap=TopData:TopData+WrapWidth-1;
BottomDataWrap=BottomData-(WrapWidth-1):BottomData;


OverallWidth=RightData+WrapWidth;
OverallHeight=BottomData+WrapWidth;
FullCol=1:OverallHeight;


LeftN=ones(OverallWidth,WrapWidth).*LeftNutrients;
RightN=ones(OverallWidth,WrapWidth).*RightNutrients;
TopN=ones(WrapWidth,height).*TopNutrients;
BottomN=ones(WrapWidth,height).*BottomNutrients;


MidLR=width/2+1;
MidTB=height/2+1;
```

```
LeftSide=width/4+1;
RightSide=width*3/4+1;
TopSide=height/4+1;
BottomSide=height*3/4+1;
LRPosition=[LeftSide MidLR RightSide];
TBPosition=[TopSide MidTB BottomSide];




N=zeros(OverallHeight,OverallWidth);
W=ones(OverallHeight,OverallWidth).*6;

if plotBuildupOrGrowth==0 then
  xset("colormap",graycolormap(time+1))
else
  xset("colormap",graycolormap(MaxN+1))
end

if InitRow==1 then
  CenterLR=LRPosition(InitRowLR);
  CenterTB=TBPosition(InitRowTB);
  for i=-3:3
```

```
    N(CenterTB,CenterLR+i) = 1;
  end
end

if InitSquare==1 then
  CenterLR=LRPosition(InitSquareLR);
  CenterTB=TBPosition(InitSquareTB);
  for i=-1:1
    for j=-1:1
      N(CenterTB+j,CenterLR+i)=1;
    end
  end
end

TempN=N;
buildup=N;

for t=1:time
 for i=DataCol
    for j=DataRow
      rad=1;
      if N(i,j)>1 then
        rad=2;
```

```
      end
      BorderSum=CalcBorder(N,i,j,rad);
      if ((BorderSum >=MinGrow) & (BorderSum <=
        MaxGrow) | (rand()<RandGrow)) then TempN(i,j)=
        TempN(i,j)+1;
      elseif ((BorderSum >MaxSame) | (BorderSum < MinSame) |
        (rand()<RandDie)) then TempN(i,j)=TempN(i,j)-1;
      end
      if TempN(i,j)<MinN then TempN(i,j)=MinN;end
      if TempN(i,j)>MaxN then TempN(i,j)=MaxN;end
    end
  end

  if WrapTopToBottom>0 then
    TempN(TopWrap,DataRow)=TempN(BottomDataWrap,DataRow);
    TempN(BottomWrap,DataRow)=TempN(TopDataWrap,DataRow);
  else
    TempN(TopWrap,DataRow)=TopN;
    TempN(BottomWrap,DataRow)=BottomN;
  end

  if WrapLeftToRight>0 then
    TempN(FullCol,LeftWrap)=TempN(FullCol,RightDataWrap);
```

```
    TempN(FullCol,RightWrap)=TempN(FullCol,LeftDataWrap);
  else
    TempN(FullCol,LeftWrap)=LeftN;
    TempN(FullCol,RightWrap)=RightN;
  end
  N = TempN;
  buildup=buildup+N;
  if plotBuildupOrGrowth==0 then
    Matplot(time-buildup);
  else
    Matplot((MaxN+1)-N);
  end
end
```

B.6 Sample Code Extreme Environment Plant Growth Simulator

```
width=100;
height=100;
time=20;

fixedRadius=3;

MinN=0;
```

```
MaxN=1;


randgrow=0.00;
randdie=0.05;


RangeTest=1;


Grow=[3 4 5 6 7 8];
Same=[0 1 2 9 10 11 12 13 14];


rangeFactor=[1 3 6];


MinSame=[1        6      8];
MaxSame=[6        22     34];


MinGrow=[3        6      13];
MaxGrow=[3        11     34];


WrapTopToBottom=0;


TopNutrients=MaxN;
BottomNutrients=0;
```

```
WrapLeftToRight=0;


LeftNutrients=0;
RightNutrients=0;


InitRow=1;
InitRowLR=2;
InitRowTB=2;




InitSquare=1;
InitSquareLR=2;
InitSquareTB=2;


plotBuildupOrGrowth=1;


rand('uniform');
rand('seed',getdate("s"));


function [BorderSum]=CalcBorder(N,r,c,radius)


  BorderSum=-N(r,c);
  for row = r-radius:r+radius
```

```
    for col= c-radius:c+radius
      BorderSum=BorderSum+N(row,col);
    end
  end

endfunction

function [NewN]=GrowSameDie(BorderSum,rad,MinGrow,
  MaxGrow,MinSame,MaxSame,randgrow,randdie,N,MinN,
  MaxN,RangeTest,Grow,Same)
  NewN=N;
  if RangeTest==1 then
    if ((BorderSum >= MinGrow(rad))&(BorderSum <=
      MaxGrow(rad))&(rand()>randgrow)) then NewN=N+1;
  elseif ((BorderSum > MaxSame(rad))|(BorderSum
    < MinSame(rad))|(rand()<randdie)) then NewN=N-1;
    end
  else
    if ((or(Grow==BorderSum))&(rand()>randgrow)) then
      NewN=N+1;
  elseif ((~or(Same==BorderSum))|(rand()<randdie)) then
    NewN=N-1;
    end
```

```
  end
  if (NewN<MinN) then NewN=MinN;end
  if (NewN>MaxN) then NewN=MaxN;end
endfunction

WrapWidth=max(MaxN,fixedRadius);

LeftData=WrapWidth+1;
RightData=width+WrapWidth;
TopData=WrapWidth+1;
BottomData=height+WrapWidth;

DataRow=LeftData:RightData;
DataCol=TopData:BottomData;

LeftWrap=1:WrapWidth;
RightWrap=RightData+1:RightData+WrapWidth;
TopWrap=1:WrapWidth;
BottomWrap=BottomData+1:BottomData+WrapWidth;

LeftDataWrap=LeftData:LeftData+WrapWidth-1;
RightDataWrap=RightData-(WrapWidth-1):RightData;
TopDataWrap=TopData:TopData+WrapWidth-1;
```

```
BottomDataWrap=BottomData-(WrapWidth-1):BottomData;


OverallWidth=RightData+WrapWidth;
OverallHeight=BottomData+WrapWidth;
FullCol=1:OverallHeight;


LeftN=ones(OverallWidth,WrapWidth).*LeftNutrients;
RightN=ones(OverallWidth,WrapWidth).*RightNutrients;
TopN=ones(WrapWidth,height).*TopNutrients;
BottomN=ones(WrapWidth,height).*BottomNutrients;


MidLR=ceil(width/2);
MidTB=ceil(height/2);
LeftSide=ceil(width/4);
RightSide=ceil(width*3/4);
TopSide=ceil(height/4);
BottomSide=ceil(height*3/4);
LRPosition=[LeftSide MidLR RightSide];
TBPosition=[TopSide MidTB BottomSide];


N=zeros(OverallHeight,OverallWidth);
W=ones(OverallHeight,OverallWidth).*6;
```

```
if (plotBuildupOrGrowth==0)|(plotBuildupOrGrowth==2) then
  BuildupPlot=scf();
  BuildupPlot.color_map=graycolormap(time+1);
end
if plotBuildupOrGrowth>0 then
  GrowPlot=scf();
  GrowPlot.color_map=graycolormap(MaxN+1);
end

if InitRow==1 then
  CenterLR=LRPosition(InitRowLR);
  CenterTB=TBPosition(InitRowTB);
  for i=-8:8
    N(CenterTB,CenterLR+i) = MaxN;
    N(CenterTB-1,CenterLR+i) = MaxN;
    N(CenterTB+1,CenterLR+i) = MaxN;
  end
end

if InitSquare==1 then
  CenterLR=LRPosition(InitSquareLR);
  CenterTB=TBPosition(InitSquareTB);
  for i=-4:4
```

```
    for j=-4:4
      N(CenterTB+j,CenterLR+i)=MaxN;
    end
  end
end


TempN=N;
buildup=N;


for t=1:time
 for i=DataCol
    for j=DataRow
      if fixedRadius>0 then
        rad=fixedRadius;
      else
        rad=1;
        if N(i,j)>1 then
          rad=N(i,j);
        end
      end
      BorderSum=CalcBorder(N,i,j,rad);
      TempN(i,j)=GrowSameDie(BorderSum,rad,MinGrow,
      MaxGrow,MinSame,MaxSame,randgrow,randdie,
```

```
      TempN(i,j),MinN,MaxN,RangeTest,Grow,Same);
   end
end

if WrapTopToBottom>0 then
   TempN(TopWrap,DataRow)=TempN(BottomDataWrap,DataRow);
   TempN(BottomWrap,DataRow)=TempN(TopDataWrap,DataRow);
else
   TempN(TopWrap,DataRow)=TopN;
   TempN(BottomWrap,DataRow)=BottomN;
end

if WrapLeftToRight>0 then
   TempN(FullCol,LeftWrap)=TempN(FullCol,RightDataWrap);
   TempN(FullCol,RightWrap)=TempN(FullCol,LeftDataWrap);
else
   TempN(FullCol,LeftWrap)=LeftN;
   TempN(FullCol,RightWrap)=RightN;
end
N = TempN;
buildup=buildup+N;
if (plotBuildupOrGrowth==0)|(plotBuildupOrGrowth==2) then
   scf(BuildupPlot);
```

```
    Matplot(time-buildup(TopData:BottomData,
    LeftData:RightData));
  end
  if plotBuildupOrGrowth>0 then
    scf(GrowPlot);

    Matplot((MaxN+1)-N(TopData-1:BottomData,
    LeftData-1:RightData));
  end
end
```

REFERENCES

[1] Earth and environmental science. Internet, New Mexico Tech Web Page, August 1985. Retrieved August 20, 2009.

[2] Life in the extremes: An interview with Dr. Penelope Boston. Internet, Astrobiology, August 2000. Retrieved August 20, 2009.

[3] P. Boston, J. Curnutt, E. Gomez, K. Schubert, and B. Strader. To live and die in ca. Retrieved August 20, 2010.

[4] P. Boston, J. Curnutt, E. Gomez, K. Schubert, and B. Strader. Patterned growth in extreme environments. In *Proceedings of the Third IEEE International Conference on Space Mission Challenges for Information Technology*, pages 221–226. IEEE Press, July 2009.

[5] P. Boston, J. Curnutt, E. Gomez, K. Schubert, and B. Strader. Patterned growth in extreme environments. 2009. Presented at the Third IEEE International Conference on Space Mission Challenges for Information Technology.

[6] E.F. Codd. *Cellular Automata*. Cambridge University Press, 1885.

[7] M. Gardner. The fantastic combinations of John Conway's new solitaire game of life. *Scientific American*, 223:120–123, 1970.

[8] M. A. Nowak. *Evolutionary Dynamics: Exploring the Equations of Life*. Belknap Press of Harvard University Press, 2006.

[9] B. P. Strader. Simulating partial differential equations with cellular automata - an empirical survey. Master's thesis, California State University of San Bernardino, 2009.