9-29-2005

Expanding the Use of Cellular Automata

Joseph Bertles
Trinity University

Expanding the Use of Cellular Automata

Joseph Bertles

A departmental senior thesis submitted to the

Department of Computer Science at Trinity University

in partial fulfillment of the requirements for Graduation with departmental honors.

April 19, 2004

Thesis Advisor	Department Chair

Associate Vice President

for

Academic Affairs

Expanding the Use of Cellular Automata

Joseph Bertles

Abstract

Cellular automata are a type of simulation based upon dividing space into cells. More specifically, cellular automata are characterized by parallelism, locality, and homogeneity. A simulation is run by conducting a series of updates, consisting of running a set of rules that all cells follow. The rules typically consist of looking at a cell's immediate neighbors and/or itself to determine what will be in the cell at the next step. The rules are applied to all the cells at exactly the same time in exactly the same manner.

The use of cellular automata has been limited to computer scientists, those who can write code, and people who understand the traditional nomenclature. Physicists, mathematicians, or even those who are just interested in different types of simulations should be able to fully explore the full potential of cellular automata. In order to expand the use of cellular automata to additional fields, my research has led to the creation of a program that allows users to easily create cellular automata without having to have foreknowledge of cellular automata terminology. The user is taken through a series of steps where they can control the size of the system, the number and speed of iterations, define the system variables in their own terms, populate the system however they wish, and make their own rules. The program also allows extreme flexibility so that non-traditional simulations can potentially be explored.

Acknowledgments

I would first like to thank my professors - Dr. Eggen for his support and guidance, Dr. Lewis for putting up with all my questions, Dr. Massingill for her help with LaTex, and Dr. Hicks for all his help over the years. My family, especially my parents, deserve all the gratitude I can give. I also want to thank Todd for putting up with me and to the Ninjas for being ninjas.

Expanding the Use of Cellular Automata

Joseph Bertles

A departmental senior thesis submitted to the

Department of Computer Science at Trinity University

in partial fulfillment of the requirements for Graduation with departmental honors.

TRINITY UNIVERSITY

MAY 2005

Contents

1 Introduction **1**

1.1 The History of Cellular Automata 1

1.2 A Modern Definition 3

2 Real World Applications **7**

2.1 The Natural Sciences 8

2.1.1 *N*-body Simulations 8

2.1.2 Myxobacteria 9

2.1.3 Lattice Gas Cellular Automata 10

2.2 Simulating Human Interactions 11

2.2.1 The Stock Market 11

2.2.2 Traffic Lights 12

2.3 Computer Science 13

2.3.1 Image Analysis and Processing 14

3 Other Contributions **17**

3.1 Cellular 17

3.2 CASim 18

3.3 CelLab 18

3.4 Mirek's Cellebration 19

3.5 Mathematica 20

4 Jackal **21**

4.1 Describing the System 22

4.2 Creating the Cell Structure 24

4.3 The Initial Configuration 25

4.4 Defining the Transition Function 26

4.5 Creating the Program 29

4.6 Summary and Conclusion 30

4.7 Further Research 31

A Source Code **34**

A.1 AddCellVariable.java 34

A.2 AddCondition.java 36

A.3 AddResult.java 37

A.4 AfterAddCellVariables.java 38

A.5 AfterAddRule.java 39

A.6 AfterPopulateSystem.java 41

A.7 CellularAutomataWriter.java 42

A.8 CellWriter.java 48

A.9 CreateCellularAutomata.java 52

A.10 CreateExecutable.java 53

A.11 CustomCellularAutomata.java 55

A.12 CustomCellularAutomataStart.java 56

A.13 DisplayCellularAutomataWriter.java 57

A.14 ManifestWriter.java 59

A.15 PopulatedCellsDataFileWriter.java 60

A.16 PopulateSystem.java 61

A.17 RemoveCellVariable.java 63

A.18 RemovePopulatedCellAction.java 64

A.19 RemoveRule.java 65
A.20 VariableChecker.java 66
A.21 WriteToFile.java 68

B Web Pages **70**

B.1 addCellVariables.jsp 70
B.2 customStart.jsp 72
B.3 makeRules.jsp 74
B.4 populateSystem.jsp 78

List of Figures

1.1 Conway's Game of Life. 4

1.2 von Neumann and Moore neighborhoods. 5

2.1 *N*-body cellular structure. 9

2.2 Stock market simulation results. 12

2.3 Examples of skeletonization. 15

3.1 Mirek's Cellebration. 20

4.1 Describing the system. 22

4.2 CustomBean class diagram. 23

4.3 Creating the cell structure. 24

4.4 VariableBean class diagram. 25

4.5 PopulateSystemBean class diagram. 26

4.6 Populating the system. 27

4.7 Creating a transition function. 28

4.8 The transition function. 29

Chapter 1

Introduction

Understanding the world around us has always been a human quest. Cellular automata provide a tool for simulating discrete phenomena in a way that can be analyzed more readily than otherwise possible. More specifically, cellular automata can provide a visual representation of events that may not be otherwise seen or understood. Although there is tremendous potential for the use of cellular automata, the terminology and development of such a system can be intimidating; as a result, the world of cellular automata is closed to many who could benefit from it.

1.1 The History of Cellular Automata

Also referred to as "tessellation automata", "homogeneous structures", "cellular structures", "tessellation structures", "iterative arrays", or "cellular space", the concept of cellular automata was developed by John von Neumann in the 1950's [17, p. 6]. Driven by an interest in the general problems associated with the behavior of computing structures, von Neumann began to appreciate the complexity and performance of the human brain as a model for designing automatic computing machines. However, with the complexity that such a computing structure brings, a greater need for reliability evolves. Von Neumann thought this could be solved by developing a self-repairing mechanism within the computing

structure itself. As a result, he became increasingly interested in self-reproducing computer systems. Initially, he strove to produce computer systems with the physical property of self-reproduction, but later this became more theoretical [10, p. ix].

John von Neumann believed that this cellular space should not only be able to reproduce an independent version itself, but also be able to produce an independent automaton that differs from itself [4, p. 2-3]. For the most part, reproduction was assumed to be asexual; however, there has been some discussion on sexual reproduction where the code in the "offspring" is not identical to either of its parents [13, p. 83].

In addition to this system being self-reproducing, von Neumann wanted his creation to be nontrivial [13, p. 83]. This means that his computer system should be able to handle any instruction set given to it, in other words it was to be a universal computing system. Operating as a Turing Machine is the only way to demonstrate this property [10, p. 7].

As a result of his demanding requirements, his original construct consisted of nearly 200,000 computers [10, p. 7]. John von Neumann described his new system as having all of the following characteristics [4, p. 2]:

1. An infinite plane is divided up into squares.
2. Each square contains a copy of the same finite automaton (the combination of the square with its automaton is called a cell).
3. A cell's neighborhood consists of itself along with its four immediate, non-diagonal neighbors.
4. The state of a cell at time $t + 1$ is dependent on the state of its neighborhood at time t and the transition function f of the finite automaton in each cell.
5. The finite automaton in each cell has a start state that is the same as every other finite automaton's start state.
6. Only a finite number of cells are not in the start state.
7. There are 29 distinct states that each cell can be in.
8. The transition function f is the same for each cellular automaton and yields a consistent result.

After von Neumann developed the principles of cellular automata, other scientists began

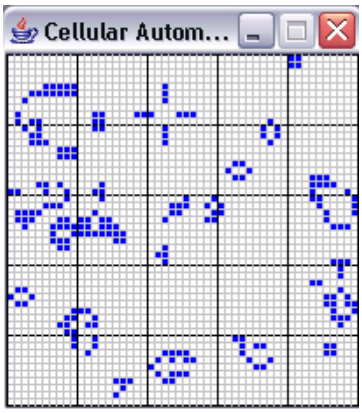
to experiment with different configurations. While maintaining the ability to serve as a universal machine and a self-reproducing system, a balance between neighborhoods and number of possible states emerges. This is an important investigation because the number of possible states in a cell has a dramatic effect on computational speed; the larger the neighborhood, the more computation that is required. In order to reduce the size of the neighborhood, the number of possible states must increase; unfortunately, the limit to this tradeoff has not been determined [13, p. 85].

While studying the performance of these massive computer systems, Stanislaw Ulam and Schrandt became interested by the patterns the automaton's states produced. Different computational rules produced different patterns, some that produced patterns of mainly state 0, others that produced patterns with mainly state 1. Conway, however, discovered a simple rule that produced a somewhat stable system that exhibits oscillation [10, p. 7]. In Conway's system, a cell's neighborhood consists of the 8 surrounding cells. This differed from von Neumann's systems that did not include diagonal neighbors. The rule set he discovered is now known as *Conway's Game of Life*, or *Conway's Life* (See Figure 1.1). The development of Conway's computational rules and those like it were called Cellular Automata Games. These games are what many envision when they think of cellular automata [10, p. 10].

1.2 A Modern Definition

The term cellular automata in modern computer science represents a variety of different meanings. Since cellular automata are being used for simulation purposes, the line between a true cellular automata and a generic simulation is blurring. There are, however, several characteristics that all modern cellular automata should possess. Although most modern cellular automata are not universal machines, they do still retain von Neumann's most general ideals.

Unlike von Neumann's constructs, modern cellular automata are typically not physical



(a)

1. Every organism with two or three neighboring organisms survives for the next generation.
2. Every organism with four or more neighbors dies from over population.
3. Every organism with one or fewer neighbors dies from isolation.
4. Each empty cell with exactly three occupied neighbors gives birth to an organism.

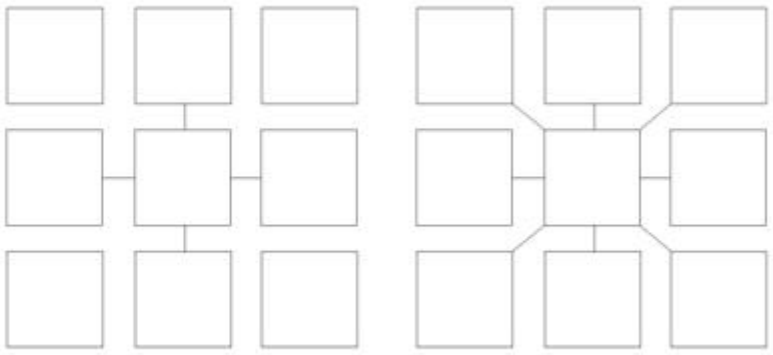
(b)

Figure 1.1: **(a) Picture of** *Conway's Game of Life***. (b) Rules for** *Conway's Game of Life***.**

computer systems, but rather computer programs. These programs are often simulations of real world phenomena in which space is divided into cells, typically squares. Each cell possesses a property referred to as a cell's state. For each cellular automaton, there are, traditionally, a finite number of possible states. The cell's state is determined by a rule or set of rules, called the transition function. The transition function is applied at each time step[15, p. 188].

The space a cellular automaton represents is described in terms of its dimensions. Cellular automata can be 1-dimensional, 2-dimensional, 3-dimensional, or greater. The simplest cellular automata are clearly 1-dimensional, consisting of a single array of cells. Similarly, a 2-dimensional cellular automaton is an array of arrays of cells, or a series of arrays side-by-side. According to von Neumann's definition, a cellular automaton should be infinite in all dimensions. In other words, all arrays in the above descriptions would be of infinite length in a von Neumann cellular automaton [13, p. 82,86].

Another important aspect of cellular automata is the neighborhood that is used in the local rule that is applied at each time step. There are several different types of neighborhoods and is dependent on the number of dimensions in the cellular automaton and the shape of its cells. In a 1-dimensional cellular automaton, a neighborhood is typically the cells to its left, right, or both. Occasionally, a larger, disconnected neighborhood is used. In a 2-dimensional cellular automaton, there are two widely used types of neighborhoods: the von Neumann neighborhood and the Moore neighborhood. The von Neumann neighborhood, or orthogonal neighborhood, consists of a cell and its immediate neighbors to the right and left, as well as those above and below the cell (See Figure 1.2(a)). The Moore neighborhood, or unit cube, on the other hand, includes the cells immediately to the northwest, north, northeast, east, southeast, south, southwest, and west of the cell in question (See Figure 1.2(b)). There are many other types of neighborhoods in a 2-dimensional cellular automaton; some even have a circular or hexagonal shape [13, p. 82,86].



(a) (b)

Figure 1.2: **(a) John von Neumann's neighborhood. (b) Moore's neighborhood.**

At each time step, a cell and its neighbors are inspected to determine the state of that cell for the next time period. A rule or set of rules are used to change the cell's state. This is referred to as the local rule or transition function. The same local rule is applied to all the cells at time t so that the state of all cells for time $t + 1$ are determined in parallel. This creates a homogeneous system in a sense that no cells have a transition function that

is different from any other cell [17, p. 5].

Chapter 2

Real World Applications

The flexible nature of cellular automata lends itself to use in a wide variety of study. Stephen Wolfram, one of the leading figures in cellular automata research, has said:

> Physical systems containing many discrete elements with local interactions are often conveniently modeled as cellular automata. Any physical system satisfying differential equations may be approximated as cellular automaton by introducing finite differences and discrete variables[17, p. 6].

From the effects of gravity to the reaction between chemicals, the world around us is governed by various rules and/or laws. Cellular automata can be used to simulate a wide variety of physical phenomena. In addition to representing the physical world, cellular automata have also been used within the field of computer science.

In order to expand the use of cellular automata, one must first come to appreciate the wide variety of manners in which cellular automata can and have been used. The following are by no means a comprehensive discussion of the utility cellular automata posses; there are virtually endless possible applications for cellular automata. The examples have been chosen to exemplify the diversity of cellular automata both in function and in terms of the fields in which it they are being employed.

2.1 The Natural Sciences

Biology, Chemistry, and Physics all have one thing in common - they all convert the complex physical world around us into formulas, rules, and laws. As a result, large portions of the research done in these three fields can be simulated with cellular automata. Biology, for instance, is easily portrayed by cellular automata since biological organisms are composed of cells and follow certain patterns. Similarly, chemical reactions can be simulated with cellular automata because the reactions follow a specific sequence of events. The portions of Physics that break down physical phenomena into discrete units can also be easily simulated with cellular automata.

2.1.1 *N*-body Simulations

The potential use of cellular automata within the field of physics is enormous. Once a phenomenon's processes are broken down into discrete activities, it can be represented with a cellular automaton. Simulating *N*-body interactions can be done with a high degree of accuracy using cellular automata. In fact, some cellular automata have produced results nearly identical to laboratory experiments [9, p. 11]. *N*-body simulations mimic the actions of N bodies of mass. These bodies are typically referred to as particles. In order to limit computational complexity and to make analysis easier, many *N*-body simulations are done in 2-d rather than 3-d [9, p. 1].

In the system developed by Lejeune, Perdang, and Richert, a specialized cellular automaton was developed. Instead of using square shaped cells, space was divided into hexagon shaped cells. A neighborhood containing two rings of surrounding cells was used. The six cells in immediate contact with the cell in question compose the first ring, with the 12 cells surrounding that ring forming the second (See Figure 2.1). The cells themselves do not represent the particles being simulated, but rather have seven possible "momentum states." Each of these states is a separate variable. If no particle is in that state the state has a value of 0, if a particle within the space of that cell is in that momentum state it has a value

of 1. If a particle is in a momentum state of 0, it is at rest. The other six states represent the direction in which the particles will move [9, p. 1-2].

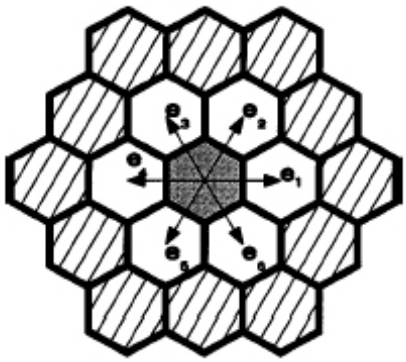


Figure 2.1: **A hexagonal cellular automata structure. The cell's neighborhood consists of the two rings of cells shown (the inner white ring and the striped outer ring). The arrows demonstrate six possible particle positions within each cell. The seventh particle position is at the center of the cell.**

The transition from one time-step to another is composed of several phases. The first phase handles collision detection within the cell and between particles of different cells. If more than one momentum state has a value of 1, possible collisions will be investigated. After particle collisions within the cell have been analyzed, then collisions between particles in the cell with particles from neighboring cells are addressed. The second phase "moves" the particles from one location to another (in a true cellular automata there are no actual entities that move from cell to cell, but rather entities are represented by variables stored in each cell) [9, p. 3-4]. Many aspects of this cellular automaton are similar to a lattice gas cellular automaton; however, a few of its characteristics (i.e. the ability of a particle to be in a stationary position) prevent it from being classified as such.

2.1.2 Myxobacteria

Myxobacteria, a social bacteria that swarm, feed, and develop cooperatively, group together to form what is called a *fruiting body* when there is a lack of food. While in *fruiting body* form, the bacteria is non-mobile. This is a way of conserving energy. The manner in which

this type of bacteria move and form the *fruiting body* is extremely rule based, which lends itself to simulation through cellular automata [2, p. 1].

Since the bacteria stack on top one another, a special kind of cellular automata is needed. In the program developed by Alber, Jiang and Kiskowski, a lattice gas cellular automaton was used to model the bacteria's behavior. This type of cellular automata has a two-part transition function: an interaction step and a transport step. In the interaction step, the state of the particle at each lattice site is updated. The cells move in the direction and in the distance specified in the velocity state during the transport step [2, p. 4-8].

2.1.3 Lattice Gas Cellular Automata

Lattice gas cellular automata are an integral part to simulating moving bodies with cellular automata. This type of cellular automata was developed in order to simulate hydrodynamics and "fluid" dynamics. There are two subdivisions within the lattice gas genre: the HPP model and the FHP model. The main difference between the two are its cell shape; the HHP model has square cells, while FHP model, which was developed later, uses hexagonal cells [14, 7].

A central area of concern while researching lattice gas cellular automata are lattice points. Simply stated, lattice points are no different than the cells of a normal cellular automata, they just have a different name; the structure of the cell, however is extremely important. The cell's shape is either square or hexagonal; a hexagonal shape tends to be more accurate. The shape of the cell determines how many neighbors, n, the cell has. This is important because n particles can occupy the cell at any given time. Each of these particles can have n velocities; a particle must have a velocity of 1 in the direction of one of the neighboring cells. No two particles in a cell, however, may have the same velocity [14, 7, 11].

The second important feature, and the one which requires a different construction than conventional cellular automata, is the transition function. Within a lattice gas cellular automata, a transition function is broken down into two parts: propagation and collision.

In the propagation portion, the particles "move" from one cell to another. In a true cellular automat there are no entities that move from cell to cell, rather the values representing the entity are moved from one cell to the next. Collision detection is handled by a collision rule, often in the form of a lookup table [14, 7, 11].

2.2 Simulating Human Interactions

Although human behavior can never be simulated with complete accuracy, there are methods to approximate human decision making. Using stochastic decision making enables scientists to model human interactions with acceptable precision. Stochastic decision making involves using probability and/or random variables. This is a popular method of simulating human behavior because humans do not necessarily follow set rules.

2.2.1 The Stock Market

One area within the arena of human interactions that has lent itself to be simulated through cellular automata is the stock market. Each cell within the given space represents a person. A person's state can either be a trader (1), a seller (-1), or inactive (0). At each time-step the state is reevaluated using its neighbor's as an influence. This simulates the group mentality characteristic of human behavior mimicking the influence large or prestigious trading corporations have over the market [3, p. 1-3].

In order to simulate this group mentality, features of cellular automata were used alongside three parameters used for probability purposes. The notion of a neighborhood within cellular automata was quite useful in simulating spheres of influence. In this particular case, the von Neumann neighborhood was used. The first parameter, P_h, represents the influence a trader or seller has on an inactive person becoming either a trader or seller in the next time-step. The second parameter, P_d, is the likelihood that a trader or seller will turn inactive in the next time-step if at least one of its neighbors is inactive. The final parameter, P_e, signifies the odds of an inactive cell to spontaneously become active in the

next time-step [3, p. 1-3].

$$P_h : \sigma_i(t) = 0 \rightarrow \sigma_i(t+1) = \pm 1$$
$$P_d : \sigma_i(t) = \pm 1 \rightarrow \sigma_i(t+1) = 0$$
$$P_e : \sigma_i(t) = 0 \rightarrow \sigma_i(t+1) = \pm 1$$

The parameters P_d and P_e were kept constant for the simulation. P_h, however, changes depending on the situation. Despite the fact that this cellular automata is attempting to approximate human behavior, the results from the simulation have a striking resemblance to the S&P500 over the past fifty years (See Figure 2.2). Although the results from the cellular automata were not an exact match for the S&P500, it did exhibit the same types of trends. For instance, the cellular automaton is able to simulate market crashes as well as bubbles because of its use of clustering to approximate group mentality [3, p. 3-8].

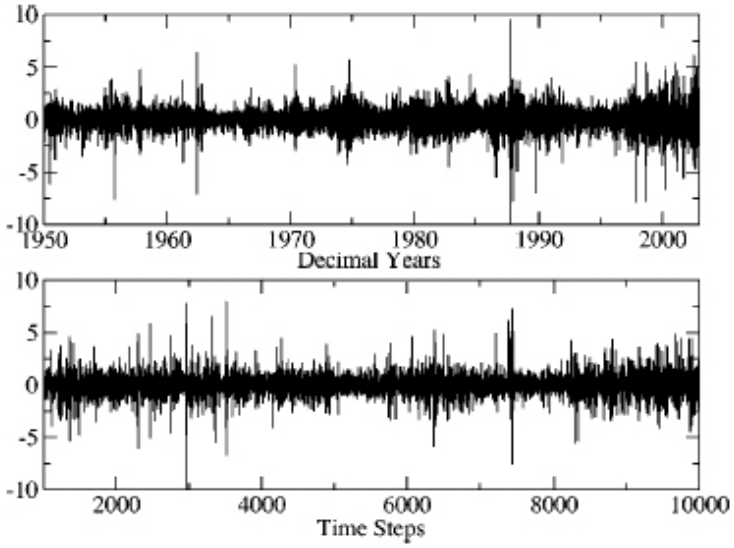


Figure 2.2: **Comparison between the stock market (*top*) and the results of the simulation(*bottom*).**

2.2.2 Traffic Lights

Another area of inquiry within the realm of human interactions is the issue of traffic flow. More specifically, there has been a large amount of research done on the benefits of syn-

chronizing traffic lights. The research has been centered maximizing the amount of total throughput over a given area. Until recently, most of this research has been done solely with mathematics; however, improvements in the field of cellular automata have enabled researchers to more accurately simulate traffic flow [8, p. 1].

In the model designed by Ding-wei and Wei-neng, a one lane, signal direction road was simulated. Lights were spaced at equal distances and given the same signal period (the length of time a traffic light takes to cycle from green to red and back again). The simulations were run under three levels of traffic flow: low-density, saturated, and over-saturated. Interestingly, synchronizing traffic lights has its most dramatic effect on low-density traffic. Over-saturated conditions can be slightly improved by using a negative delay time (this effectively mores the traffic jam further away from the root, dispersing the effects). During the transition between low-density and over-saturated driving conditions, the saturated stage, synchronizing traffic lights has little to no effect. In this situation, the probability in the stochastic decision making algorithm has a much greater influence on the flow of traffic [8, p. 1-7].

2.3 Computer Science

The use of cellular automata within the realm of Computer Science has a long and rich history. Since cellular automata spawned from research on parallel processing, there it's not surprising that cellular automata are still being used in the field. Cellular automata have been used in load balancing algorithms. Analyzing and processing digital images has been enabled because of cellular automata. Aside from the screen savers and other novelty products, cellular automata prove themselves quite useful within the field of Computer Science.

2.3.1 Image Analysis and Processing

One of the first uses of cellular automata was in the general field of image processing. This is not too surprising considering that digital images are already divided into a cellular form as most digital images are represented by pixels. While cellular automata were being developed, scientists were beginning to research ways to allow automated systems to replace human observers, specifically in character recognition [10, p. 3-4]. The discussion on the use of cellular automata in image processing is long and requires great detail. There are, however, several important contributions that deserve to be noted.

Segmentation and Propagation

Segmentation deals with taking a digitized image and dividing the image into subgroups where a region is characterized by likeness or homogeneity. Before segmentation can take place, noise removal must first take place. This process strives to eliminate non-uniform illumination, shadowing, vignetting, and other such imperfections. After noise removal takes place, boundaries between regions are sought. This is commonly referred to as *edge detection* or *boundary detection*. There are many different methods for detecting boundaries, each of which requires many intricate details [10, p. 103-111].

After the image has been divided into different regions, cellular automata perform a process called *propagation*. Since an object in an image can contain several regions, it is important to determine how connected adjacent regions are. The process of *propagation* serves to identify and label objects in the image. There are various methods of determining the connectedness of regions, including binary propagation and gray propagation. While binary propagation works as is, gray propagation needs to be refined for each individual case [10, p. 111-116]. The result of the segmentation and propagation processes could be described as a contour drawing of the original image.

Skeletonization

The fields of pattern and character recognition are dependent upon finding the essence of an object within an image. Skeletonization is the process by which the interior skeleton of an object or group of objects within an image is derived. This process can be executed on a two or three dimensional image (See Figure 2.3). There are several methods used to obtain the inner structure of an object, but the basic idea centers on connectivity. The general idea is to first find the object, and then reduce the shape to its underlying structure [10, p. 117-136].

Figure 2.3: **Examples of skeletonization.**

Cellular Filtering

Filters are used to extract portions of an image for measurement or to make the image more meaningful, often in the form of eliminating noise. Applying a filter often requires, at least in part, some degree of pattern recognition. A process that utilizes matching finds the edges of objects in an image. There are several types of methods used to match patterns - binary pattern primitives, generalized logical filtering, and numerical filtering. Binary pattern primitives are used when one is searching for a specific pattern. Generalized logical filtering, however is used when the object is given and its pattern or shape is in question. While generalized logical filtering first converts the image into its binary form through a threshold transformation, numerical filtering applies local rules to the images themselves through the use of cellular automata [10, p. 137-172].

Chapter 3

Other Contributions

Clearly, cellular automata have a wide variety of uses that go well beyond the realm of pure computer science. As a result, there is a need for computer scientists to open the world of cellular automata to others. Since there are such a wide variety of fields that can potentially use cellular automata, it is necessary to create an easy-to-use, flexible environment in which to develop simulations. There have been several attempts to create development environments specifically tailored to making cellular automata.

3.1 Cellular

Cellular is an entire set of programs to develop, view, and analyze cellular automata. The system is divided into four components: a programming language, compiler, virtual machine, and viewer. *Cellang 2.0*, the programming language component, allows the programmer to enter a description of the system (including the variables within in each cell) and a set of statements. The language does not have specific neighborhoods; the user must "create" neighborhoods in the conditional statements through index manipulation. After the program is written, it must be compiled, and then run. The viewer, *cellview*, reads data from a file written by the execution of a *Cellang 2.0* program [5, p. 99-102].

Although this comprehensive system has several useful features, it still requires users to

learn a programming language and write code. Compiling and running the program must be done on the command line and requires the use of various flags and parameters. This can be a daunting task for a non-computer scientist. Also, the system is extremely limited because it only allows the user to store integers in the cells. This places boundaries on the types of computations that can be done, thus not providing enough flexibility [5, p. 99-102].

3.2 CASim

Like *Cellular*, *CASim* is an entire system used to create and view cellular automata. *CASim* has only three components: a stand alone graphical user interface, a applet with a graphical user interface, and a command line execution that prints output to file. *CASimFrame*, the stand alone application, provides three methods for creating rules for a cellular automata as well as a user interface to display the simulation. The user can either create rules interactively, programming a java class, or by describing the cellular automata in the computer language CDL (the program then translates the specifications into java). *CASimApplet* enables a user to incorporate a cellular automaton into their web page. For simulations that require large amounts of computations, a regular *CASim* program, in which a java class is written by the user, can be run without a graphical display [6].

Despite the fact that providing a method for the user to incorporate a cellular automaton in the form of an applet into their web pages is a nice feature, the manner in which the user must create their own system specifications is not user friendly. In order to create a complex cellular automaton, the user must program a java class. The manual suggests that the user is only required to write a few methods, however, a non-computer scientists should not be forced to write code to experience all that cellular automata have to offer [6].

3.3 CelLab

Despite the fact that *CelLab* was developed in the late 1980's, the program surprisingly provides a descent amount of flexibility. *CelLab* provides several different ways to define

a cellular automaton. The system allows users to define rules by writing code in Java, Turbo Pascal, C, Basic, and even Assembly. Unfortunately, this flexibility is not enough to overcome its rather large faults. Obviously, the system's flexibility stems from the fact that the user is required to write code, which should be kept to a minimum, if not eliminated. In addition, the system does not allow for complex cellular automata. The cells are limited to having one variable, *state*, with integer values ranging from 0 to 256. This is extremely limiting and does not even come close to allowing the computational complexity some cellular automata can require[12].

3.4 Mirek's Cellebration

There are several applications that allow the user to view classic cellular automata. These include *Conway's Game of Life*, Gnarl, Maze and many others. Often, these programs allow the user to "edit" the rules. This means that the user is able select when a cell will die, give birth, or survive. These cellular automata only contain one variable in each cell that is either 0 or 1. Death refers to a cell changing from a state of 1 to 0, while the term birth is used to describe a cell changing from a state of 0 to a state of 1. Survival refers to a cell with a state of 1 remaining in that state for the next time step. *Mirek's Cellebration v.4.20.0.500* contains all the standards, as well as many others. The cellular automata re divided up into categories based upon their rule structure (i.e. Life, which has birth and death rules). Within the category are folders representing each rule. Each of these folders contains different pre-defined starting configurations [16].

Although this program is extremely limited in terms of real-world applications, it does have one extremely useful feature: the way it allows the user to pre-populate the system. The program provides a large screen that is divided up into cells. The user is able to use drawing tools to designate the cells that will have a value of 1 (See Figure 3.1). The tools include a pencil tool which allows the user to select individual cells, a line tool, a rectangle tool, a circle tool, and a paint bucket tool to fill in surrounded areas. The user is able to

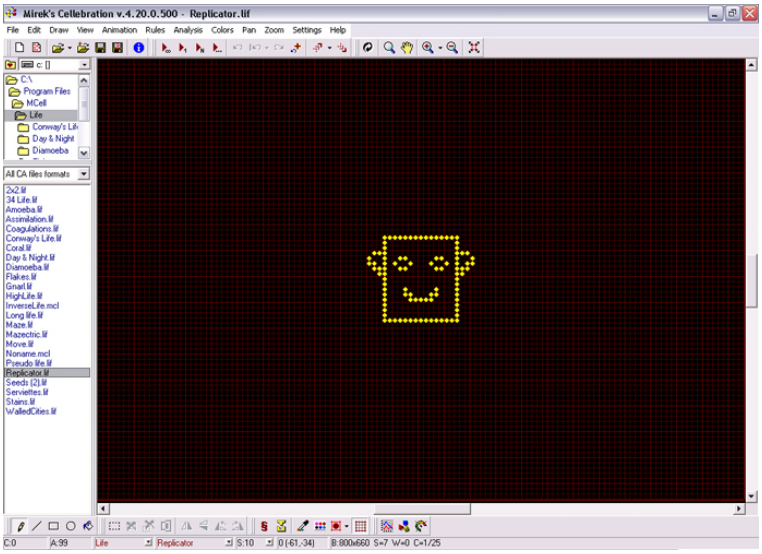


Figure 3.1: **A screen shot of Mirek's Cellebration.**

use these tools even while the cellular automata is running [16].

3.5 Mathematica

Mathematica is a comprehensive computer program developed by Wolfram Research. This system incorporates numeric calculations with graphical representations and its own programming language. One of the possible uses of *Mathematica* is to create cellular automata. In order to do this, the user must first learn the language of *Mathematica*, which changes depending on the style you choose. The language styles vary from a procedural language style similar to C++ to a more functional style similar to Lisp [1].

Even though *Mathematica* is a very powerful tool, there are several rather large drawbacks. Similar to many other powerful cellular automata development tools, creating a cellular automata with *Mathematica* requires that the user program, and in this case learn a specialized language. Another problem is the cost of the system. The program is rather expensive, even for a student license. If the one intends on using *Mathematica* for personal or commercial purposes, the program will cost over $1,800.00 [1].

Chapter 4

Jackal

Despite the fact that cellular automata are an extremely useful tool for multiple fields of research, its full potential is only available to those with the ability to write computer programs. Several computer systems have been developed that generalize the construction of a cellular automaton so that the user does not need to create an entire program on their own. Unfortunately, most of these systems either limit the capabilities of cellular automata created with their program, require the user to learn a specialized computer language, or both. The goal of Jackal is to provide the user with an easy way to create complex cellular automata.

The idea of the project is to create a user interface that is as user friendly as possible without sacrificing the power of cellular automata. The user is taken through a series of steps in which they are asked to describe the cellular space, the contents of each cell, the initial configuration, and to define the transition function. The cellular automata created by the program are written in Java to enable portability. This was done so the user's operating system should not determine their ability to use Jackal.

The first step in creating such a dynamic program was to create a cellular automata program in a way that allows for customization and expansion. The program was then broken down into components that could be defined by the user (each component will be described thoroughly in its own section). After the components of a cellular automata program were

established, a user interface was developed in the form of a JSP web-application. Finally, a system was developed to generate the code based on user input, compile the code, and create an executable jar file.

4.1 Describing the System

Jackal opens with a definition of cellular automata and an overall description of the program. The user is able to view classic examples of cellular automata, view the help page, or create a custom cellular automata program. The first thing a user does when creating a cellular automaton is describe the system. This consists of naming your cellular automata, choosing the size of the system, the size of each cell, the number of iterations, and the speed of each time step (See Figure 4.1).

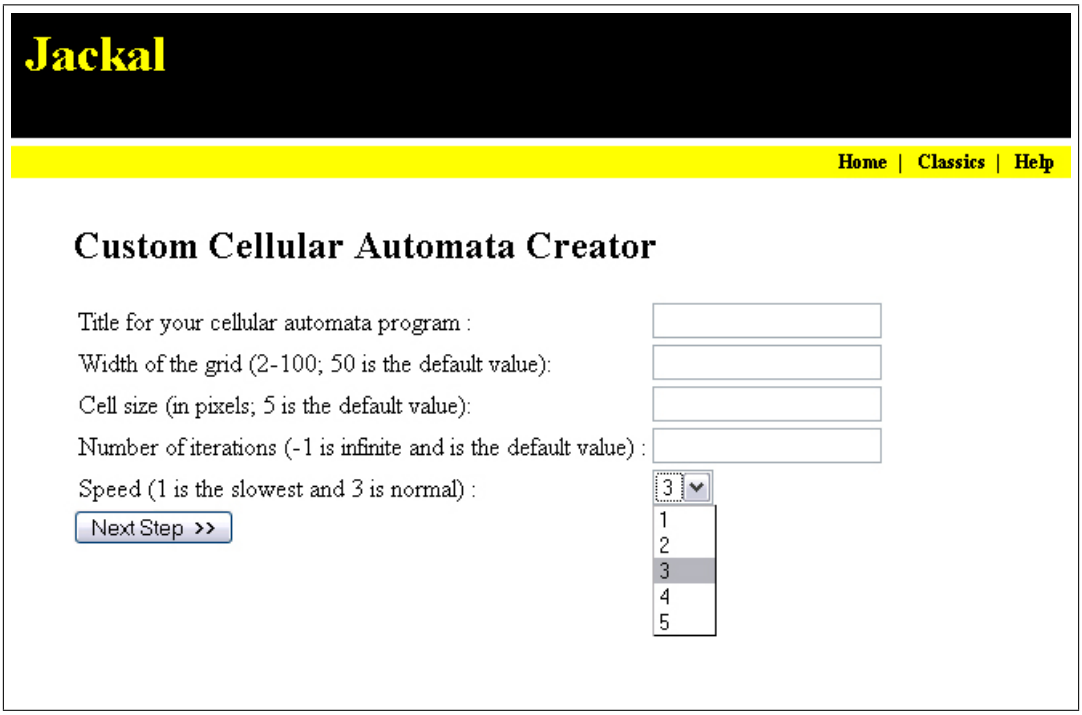


Figure 4.1: **A screen shot of describing a cellular automata in Jackal.**

The cellular automata program the Jackal creates is given the name that the user enters. The size of the system is chosen by defining how many cells wide the simulation will be.

All programs generated have equal height and width dimensions. The system must have between two and one hundred cells in width. Obviously, there are reasons for having a lower limit on the size of the system; placing a ceiling on the number of cells on the system is a result of several concerns. First, the manner in which the cellular automaton is displayed to the user necessitates an upper limit to the number of cells in the system. The amount of data that is generated by a large system leads to concerns about performance in terms of Jackal as well as the program that is produced.

The size of each cell is strictly used to adjust the visual appearance of the cellular automata program. The user is allowed to specify the number of iterations they would like the program to run through. The user can either allow the simulation to run indefinitely, or choose to end the simulation after any given time step. The user is also able to choose how quickly the program runs through the time steps. This effect is accomplished by placing a sleep timer between each time step.

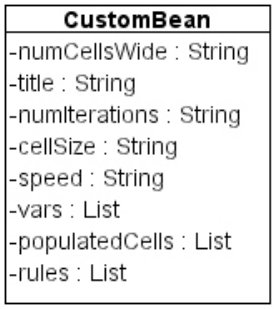


Figure 4.2: **Class diagram for CustomBean.**

When the user submits this information, a *CustomBean* is created. The *CustomBean* is the data structure that retains the overall system information. This data structure also houses a list of the variables contained within each cell, the initial configuration of the system, as well as the rules for the transition function (See Figure 4.2). This data structure is passed from page to page carrying all of the information entered by the user.

4.2 Creating the Cell Structure

After the overall features of the system have been decided, the user then creates the cell structure. This aspect of Jackal is one of the reasons the system is so flexible. The manner in which the user is able to add variables allows the user to harness the full potential of cellular automata. Creating the cell structure consists of adding variables to the cell and assigning the initial value for that variable.

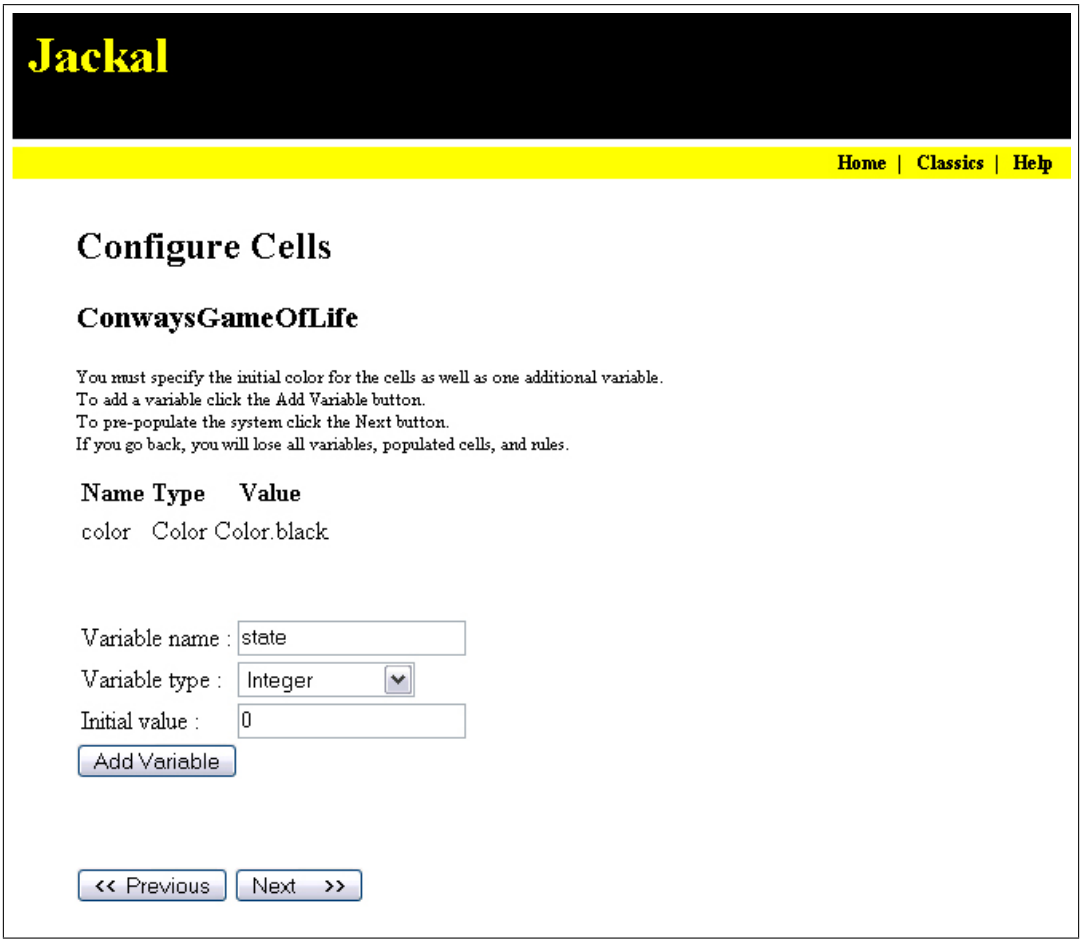


Figure 4.3: **A screen shot of creating the cell structure in Jackal.**

The user creates a variable by assigning the variable a name, choosing the variable type, and entering its initial value. The variable name is checked to make sure that it is not a keyword used by Java, used within the program, or if the user has already used that

variable name. The user chooses the variable type from a drop down menu (See Figure 4.3). Currently only two data types (integers and real numbers) have been implemented, but the system architecture of Jackal was designed to allow additional data types to be added. The initial value of the variable is checked to make sure that it is valid for the variable's type.

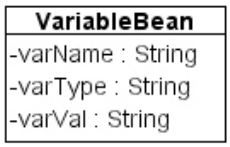


Figure 4.4: **Class diagram for VariableBean.**

The user is required to add at least two variables to the cell structure. Since the cellular automata program has a graphical output, the first variable is *color* and is of type *java.awt.Color*. The initial value for *color* is chosen from a drop down list. Before the user can continue, at least one additional variable must be chosen. This is required for computational purposes; a cell's color is not allowed to be referenced while defining conditions within the transition function. After a variable is added to the cell, it appears in a list above the user input area (See Figure 4.3). Users are allowed to remove any variable from the cell aside from *color*. When the user decides to continue, the variables, in the form of a *java.util.List* containing *VariableBeans*, are added to the *CustomBean* (See Figure 4.4).

4.3 The Initial Configuration

At this point, every cell within the system is exactly the same. Since there would be little point in having a cellular automaton where all the cells have an identical initial configuration, the initial configuration is an integral part of the system. Implementing a configuration system like that found in *Mirek's Cellebration v.4.20.0.500* would have been extremely user friendly (See Section 3.4). However, the fact that multiple variables, each with a wide range of possible values, can be housed within each cell does not allow such an interface.

In order to populate the system, a user can either select a specific cell or populate the

system randomly. If the user chooses to configure a specific cell, they select x and y values from drop down boxes. In order to populate the system randomly, they must choose the number of cells they wish to randomly populate and select the random value from the x and y drop down boxes. Once the user has selected how they wish to populate the system, they are able to choose the values for each of the variables contained in the cell. When the user submits the populated cell(s), the values are checked to make sure they are of the appropriate type.

If the cell(s) pass validation, they are displayed below the user input area (See Figure 4.6). Each of the cells listed can be removed. Each cell in the system can be configured, however, it is not necessary to change the initial configuration of any cells. When the user decides to continue, the populated cells, in the form of a *java.util.List* containing *PopulateSystemBeans*, are added to the *CustomBean* (See Figure 4.5).

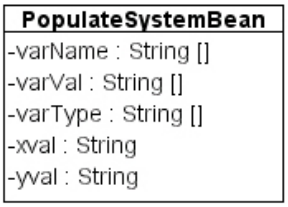


Figure 4.5: **Class diagram for PopulateSystemBean.**

4.4 Defining the Transition Function

Between each time step of a cellular automata, a transition function determines the value of the variables for the upcoming time step and is performed on every cell within the system in "parallel". The transition function consists of one or more conditions under which a variable, or variables, will change. The conditions can be based upon values of variables within the cell, those of surrounding cells, or some combination there of.

Developing a way for the user to create a transition function without foreknowledge of what the variables are or what the possible values while still being easy to use was a

daunting task. Enabling the user to have the capabilities of writing code themselves without forcing them to learn how to do so lead me to create an interface in which the user selects options from a series of drop down lists (See Figure 4.7).

The first option for the user is to decide if the condition is based upon a variable within the cell or the surrounding cells. Next the user selects a variable and an operator. All of the standard comparators are available - less than, less than or equal, equal, greater than or equal, greater than, or not equal. The user then enters a value they wish to compare the variable to. This may either be in the form of an actual value or a complex statement. This statement can refer to any of the variables within the cell the transition function is being

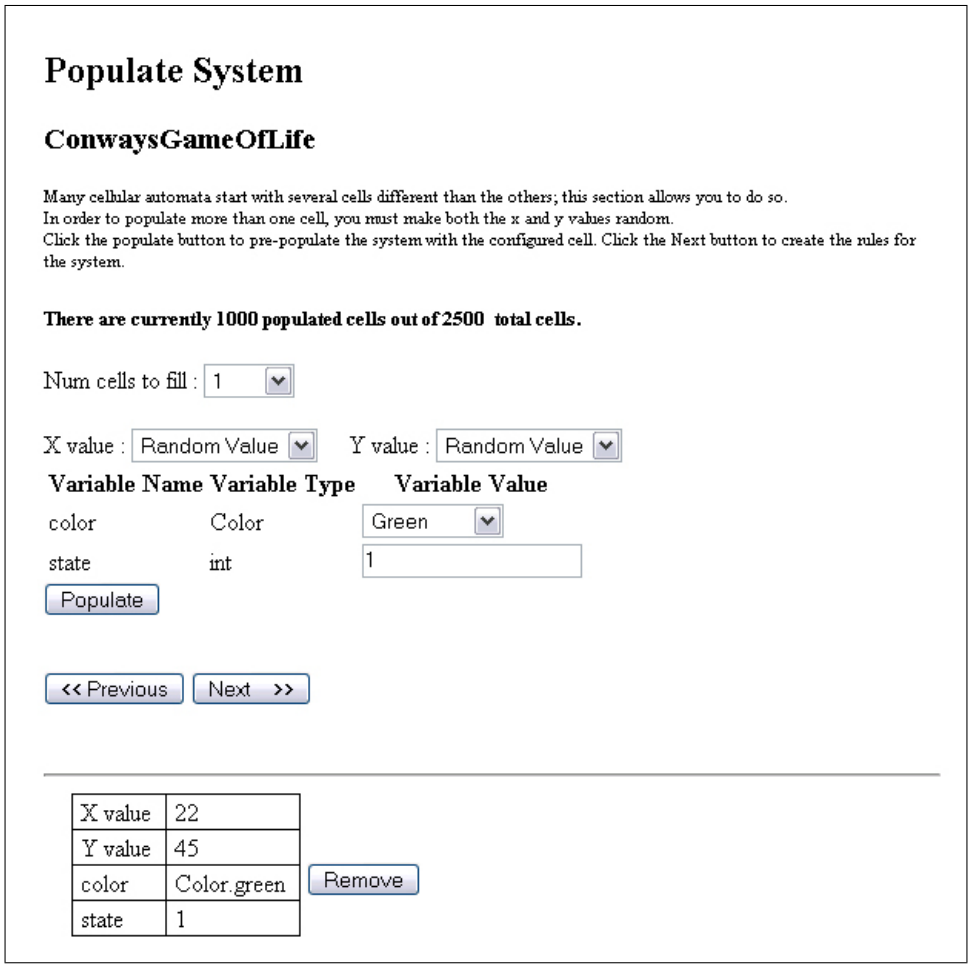


Figure 4.6: **A screen shot of populating the system in Jackal.**

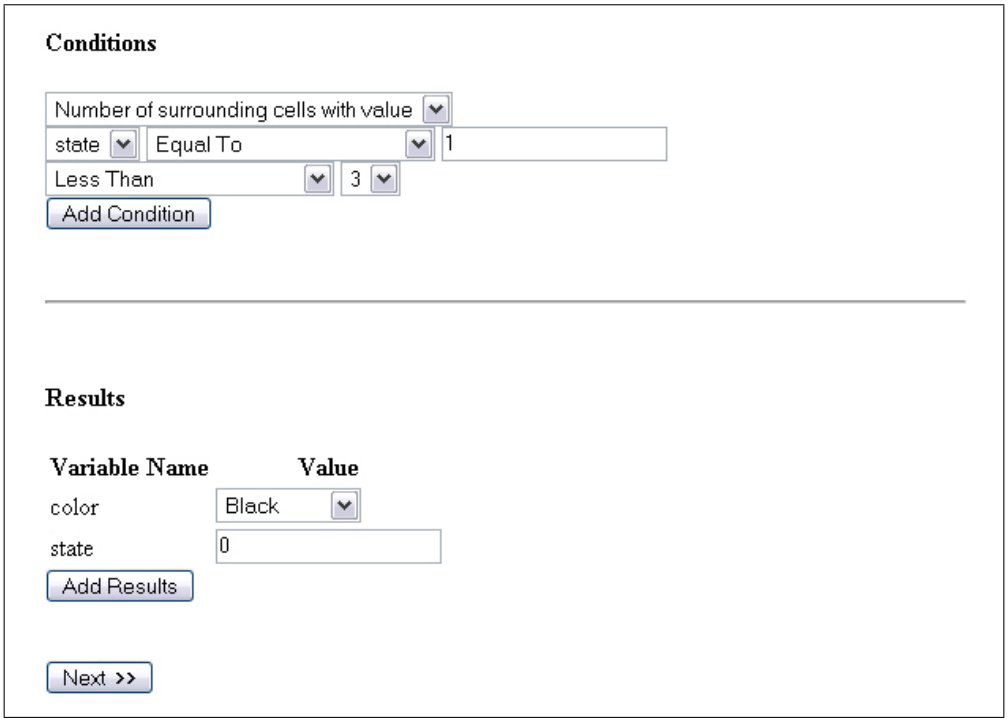


Figure 4.7: **A screen shot of creating a transition function in Jackal.**

run, literal values, and any appropriate Java operator.

If the user has chosen the number of surrounding cells from the first drop down box, they must also select how many cells that will meet that condition as well as another conditional. In other words, the user is allowed to make a conditional statement like the following:

If the number of surrounding cells with state = 0 is less than 4 then...

After entering a conditional statement, the user is able to create a compound conditional statement using the *AND* and *OR* conjunctions. All previously entered statements appear above the user input section (See Figure 4.7).

Below the conditions section the user is able to enter the results of the condition. The user is able to change the values of each variable within the cell, including *color* whose value is selected from a drop down box. As with the conditional statement, the user can enter a value or a complex statement. If a value is left blank, its contents are not changed by the transition function.

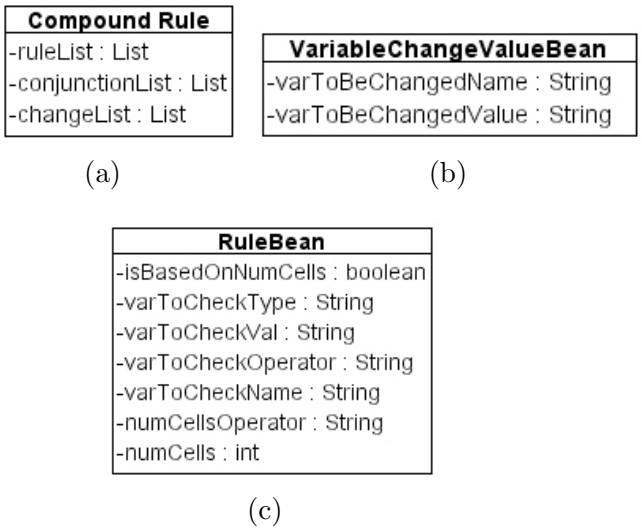


Figure 4.8: **Class diagrams of (a)** *CompoundRuleBean*, **(b)** *VariableChangeValueBean*, **and (c)** *RuleBean*

When the user submits the rule, it is added to the list of *CompoundRules* held within the *CustomBean*. A *CompoundRule* contains Lists of *RuleBeans*, *VariableChangeValueBeans*, and conjunctions in the form of a String (See Figure 4.8). The user is then taken to a page that shows all the rules of the transition function as well as options to remove any of the rules, create another rule, or generate the cellular automata program.

4.5 Creating the Program

Once the user has decided they are finished creating their cellular automaton, Jackal creates a program in the form of an executable *jar* file. The *jar* file contains the manifest and the data file containing populated cells. The CellularAutomata, Cell, and DisplayCellularAutomata classes are also included. Both the source and object files for the classes are contained in the *jar* file in case the user wishes to further customize the program.

The manifest is used to locate the main function; this enables the *jar* file to be executable. Without the manifest, a *jar* file is essentially the same as a *zip* file. The manifest must be within a folder labelled "META-INF" at the root of the *jar* file. The source code and

populated cells are held in a folder at the root level of the *jar* file called "cellularautomata". The Cell class contains the variables within the cell, get methods, and a generic set method to which the variable's name and new value are *String* parameters. The function determines what type the variable is, and assigns its value appropriately.

The CellularAutomata class creates the cellular space in the form of a two dimensional array. The initial configuration is then read from file. This was done so the class would not become too large to compile. The populated cells are written to file in the form:

x y variable value

CellularAutomata also contains the transition function and a procedure that calculates the number of surrounding cells with a specific qualification. The DisplayCellularAutomata class simply instantiates the cellular automata, and displays the contents of the cellular space at each time step.

4.6 Summary and Conclusion

Used in Biology, Physics, Economics, Computer Science, and many other fields, cellular automata have proved themselves to be an extremely flexible and powerful tool. Unfortunately, their potential has been wasted because there was not an easy way for researchers in fields other than computer science to create cellular automata. Although there is a plethora of free programs available, none of them can create complex programs.

The first goal of Jackal is to provide the most user friendly interface possible while not limiting the potential of the cellular automata. Although creating a user friendly interface is the main goal of this project, the capabilities of the programs created should not be sacrificed. A balance must be reached; cellular automata lose their effectiveness if their complexity is limited, but they will not be utilized if they are too difficult to create. One of the most difficult aspects of creating Jackal was developing a method of initializing the cellular automaton. Since there are so many possible configurations for each cell, a simple interface could not be used. However, the fact that the interface is not simple does not

mean that it is not easy to use.

Another goal of Jackal was to allow as many people as possible to have access to the program. This was done by writing the code in Java, allowing the automata program to be run on any operating system. Since Java programs are not actually compiled into machine code, the programs are portable across platforms. The user is able to further customize their cellular automata, since the source code for the program is included within the *jar* file.

4.7 Further Research

As with any academic pursuit, there are areas that could be researched further. One area of cellular automata simulations that was not covered by the research is data analysis. Although Jackal does provide a graphical display of the cellular automaton, there is not currently a way of analyzing the values within the cells. This would most likely need to be done after the simulation is run. The values of the cells could be written to file, similar to the way the cells are populated. A separate program could then be run on the data file.

The process of creating the initial configuration of the cellular automaton could also be improved. Developing a method to populate the system using a mathematical equation could be extremely useful for research in Mathematics and Physics. Allowing the user to specify what type of neighborhood the transition function will use would also be a useful feature.

Bibliography

[1] Wolfram research. Internet. www.wolfram.com/.

[2] Mark S. Alber, Yi Jiang, and Maria Kiskowski. Lattice gas cellular automata model for rippling and aggregation in myxobacteria. *Elsevier Science*, 2004.

[3] M. Bartolozzi and A. W. Thomas. Stochastic cellular automata model for stock market dynamics. *Physical Review E*, 2004.

[4] F. C. Codd. *Cellular Automata*. Academic Press, 1968.

[5] J. Dana Eckart. A cellular automata simulation system: Version 2.0. *ACM SIGPLAN Notices*, 27(8), 1992.

[6] Uwe Freiwald and Jorg Weimar. Manual to casim java-environment for simulating cellular automata. Internet, 1999. www.jweimar.de/jcasim/casimhand.

[7] David Hannon and Olivier Tribel. Lga: introduction to lattice gas automata. Internet. poseidon.ulb.ac.be/simulations/intro en.html.

[8] Ding-wei Huang and Wei-neng Huang. Traffic signal synchronization. *Physical Review E*, 2003.

[9] A. Lejeune, J. Perdang, and J. Richert. Application of cellular automata to n-body systems. *Physical Review E*, 60(3), 1999.

[10] Kendall Preston and Michael J.B. Duff. *Modern Cellular Automata: Theory and Application*. Plenum Press, 1984.

[11] Daniel Rothman and Stephane Zaleski. Simple models of complex hydrodynamics. Internet, 1997. cscs.umich.edu/ crshalizi/reviews/rothman-zaleski-on-lga/.

[12] Rudy Rucker and John Walker. Cellab user guide exploring cellular automata. Internet. www.fourmilab.ch/cellab/manual/.

[13] Palash Sarkar. A brief history of cellular automata. *ACM Computing Surveys*, 32(1), March 2000.

[14] Franz J. Vesely. 8.3.1 lattice gas cellular automata. Internet, October 2001. ap.univie.ac.at/users/ves/cp0102/dx/node126.html.

[15] Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice Hall, 1999.

[16] Mirek Wojtowicz. Mirek's cellebration - 1d and 2d cellular automata explorer. Internet. www.mirwoj.opus.chelm.pl/ca/index.html.

[17] Stephen Wolfram. *CellularAutomata and Complexity.* Addison-Wesley Publishing Company, 1994.

Appendix A

Source Code

A.1 AddCellVariable.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.commons.*;
import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.*;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class AddCellVariable
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.AddCellVariable.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {

    logger.debug("Entering AddCellVariable action");

    VariableBean bean = (VariableBean) actionForm;
    ActionErrors errors = new ActionErrors();

    logger.info("Retreiving attributes");
    //retreiving the necessary attributes, if not found send user to index
    //if found, remove them from the session
    List vars = (List) request.getSession().getAttribute("vars");
    if (vars == null) {
      response.sendRedirect("index");
      logger.info("Vars = null");
    }
    else {
      request.getSession().removeAttribute("vars");
    }
```

```
  CustomBean cb = (CustomBean) request.getSession().getAttribute(
      "customBean");
  if (cb == null) {
    response.sendRedirect("/index.jsp");
    logger.info("cb = null");
  }
  else {
    request.getSession().removeAttribute("customBean");
  }

  logger.info("Verifying that the form was filled out");
  //checking to see if the form was filled out
  if (bean.getVarName() == null || bean.getVarType() == null ||
      bean.getVarVal() == null) {
    errors.add("addCellVariable", new ActionError("form.empty.field.all"));
    this.saveErrors(request, errors);
    return actionMapping.findForward("failure");
  }
  if (bean.getVarName().trim().equals("") || bean.getVarType().equals("") ||
      bean.getVarVal().equals("")) {
    errors.add("addCellVariable", new ActionError("form.empty.field.all"));
    this.saveErrors(request, errors);
    return actionMapping.findForward("failure");
  }

  logger.info("Adjusting varName and varVal to remove all spaces");
  //adjusting varName and varVal to remove all spaces
  bean.setVarName(bean.getVarName().trim().replaceAll(" ", "_"));
  bean.setVarVal(bean.getVarVal().trim().replaceAll(" ", "_"));

  logger.info("Validating variable");
  //checking to see if the variable is valid
  if ( (! (vars.size() == 0)) && containsVariable(vars, bean.getVarName())) {
    request.setAttribute("vars", vars);
    request.setAttribute("customBean", cb);
    errors.add("addCellVariable", new ActionError("variable.exists"));
    this.saveErrors(request, errors);
    return actionMapping.findForward("failure");
  }
  if (!VariableChecker.checkName(bean.getVarName()) && ! (vars.size() == 0)) {
    request.setAttribute("vars", vars);
    request.setAttribute("customBean", cb);
    errors.add("addCellVariable", new ActionError("variable.invalid"));
    this.saveErrors(request, errors);
    return actionMapping.findForward("failure");
  }
  if (!VariableChecker.checkTypeValuePair(bean.getVarType(), bean.getVarVal())) {
    request.setAttribute("vars", vars);
    request.setAttribute("customBean", cb);
    errors.add("addCellVariable", new ActionError("variable.type.value.error"));
    this.saveErrors(request, errors);
    return actionMapping.findForward("failure");
  }

  logger.info("Adding variable to list");
  //if the varible passes the checks, it is added to the list
  vars.add(bean);
  request.setAttribute("vars", vars);
  request.setAttribute("customBean", cb);
  return actionMapping.findForward("success");
}

private boolean containsVariable(List vars, String varName) {
```

```
    logger.debug("Entering containsVariable");
    logger.info("NumVars = " + vars.size());
    logger.info("VarName = " + varName);

    for (int i = 0; i < vars.size(); i++) {
      if ( ( (VariableBean) vars.get(i)).getVarName().equals(varName)) {
        return true;
      }
    }
    return false;
  }
}
```

A.2 AddCondition.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.List;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class AddCondition
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.AddCondition.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {
    logger.debug("Entering AddCondition action");

    AddConditionBean bean = (AddConditionBean) actionForm;

    logger.info("Getting necessary attributes");
    List list = (List) request.getSession().getAttribute("conditions");
    CustomBean cb = (CustomBean) request.getSession().getAttribute(
        "customBean");
    AddResultBean results = (AddResultBean) request.getSession().getAttribute(
        "results");
    if (list == null || cb == null) {
      response.sendRedirect("index");
    }
    if (results != null) {
      request.getSession().removeAttribute("results");
      request.setAttribute("results", results);
    }
    request.getSession().removeAttribute("conditions");
    request.getSession().removeAttribute("customBean");
    request.setAttribute("customBean", cb);

    if (bean.getVarValue().equals("") || (bean.getNumCellsBased().equals("true") &&
                                          (bean.getNumCells().equals("") ||
                                           bean.getNumCellsOperator().equals("")))) {
      request.setAttribute("conditions", list);
      ActionErrors errors = new ActionErrors();
      errors.add("addCondition", new ActionError("form.field.empty.all"));
```

```
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }
    logger.info("Adding condition to the list");
    list.add(bean);

    request.setAttribute("conditions", list);

    return actionMapping.findForward("success");
  }
}
```

A.3 AddResult.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.*;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class AddResult
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.AddResult.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {
    logger.debug("Entering AddResult action");

    logger.info("Retrieving necessary attributes");
    AddResultBean addResultBean = (AddResultBean) actionForm;
    CustomBean cb = (CustomBean) request.getSession().getAttribute("customBean");
    List conditions = (List) request.getSession().getAttribute("conditions");
    AddResultBean results = (AddResultBean) request.getSession().getAttribute(
        "results");
    if (conditions != null) {
      request.getSession().removeAttribute("conditions");
    }
    else {
      conditions = new ArrayList();
    }
    request.setAttribute("conditions", conditions);
    if (results != null) {
      request.getSession().removeAttribute("results");
    }
    else {
      response.sendRedirect("index");
    }
    if (cb != null) {
      request.getSession().removeAttribute("customBean");
      request.setAttribute("customBean", cb);
    }
    else {
      response.sendRedirect("index");
    }
```

```
    ArrayList names = new ArrayList();
    ArrayList values = new ArrayList();
    String[] name = addResultBean.getVarName();
    String[] value = addResultBean.getVarValue();

    logger.info("Harvesting only the changes that the user entered");

    if (name != null && value != null) {
      for (int i = 0; i < value.length; i++) {
        if (value[i] != null) {
          if (!value[i].equals("")) {
            names.add(name[i]);
            values.add(value[i]);
          }
        }
      }
    }

    if (names.size() == 0) {
      ActionErrors errors = new ActionErrors();
      errors.add("addResult", new ActionError("form.value.empty"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }

    name = new String[names.size()];
    value = new String[values.size()];
    for (int i = 0; i < names.size(); i++) {
      name[i] = (String) names.get(i);
      value[i] = (String) values.get(i);
    }
    addResultBean.setVarName(name);
    addResultBean.setVarValue(value);
    request.setAttribute("results", addResultBean);

    return actionMapping.findForward("success");
  }
}
```

A.4 AfterAddCellVariables.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.List;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class AfterAddCellVariables
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.AfterAddCellVariables.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {
    logger.debug("Entering AfterAddCellVariables action");
```

```
    logger.info("Retrieving necessary attributes");
    List vars = (List) request.getSession().getAttribute("vars");

    if (vars == null) {
      response.sendRedirect("index");
    }
    request.getSession().removeAttribute("vars");
    CustomBean bean = (CustomBean) request.getSession().getAttribute(
        "customBean");
    if (bean == null) {
      response.sendRedirect("index");
    }
    request.getSession().removeAttribute("customBean");

    String str = (String) request.getParameter("Submit");
    if (str.startsWith("Next")) {

      logger.info(
          "Verifying that the user has entered a sufficient amount of variables");
      if (vars.size() == 0) {
        ActionErrors errors = new ActionErrors();
        errors.add("afterAddCellVariables", new ActionError("no.cell.variables"));
        this.saveErrors(request, errors);
        request.setAttribute("customBean", bean);
        return actionMapping.findForward("failure");
      }
      if (vars.size() == 1) {
        ActionErrors errors = new ActionErrors();
        errors.add("afterAddCellVariables", new ActionError("one.cell.variable"));
        this.saveErrors(request, errors);
        request.setAttribute("customBean", bean);
        request.setAttribute("vars", vars);
        return actionMapping.findForward("failure");
      }
      logger.info("Saving the variables");
      bean.setVars(vars);
      List list = bean.getPopulatedCells();
      if (list != null) {
        logger.info("There are " + list.size() + " populated cells");
      }
      else {
        logger.info("There are no populated cells");
      }

      request.setAttribute("customBean", bean);
      return actionMapping.findForward("next");
    }
    else {
      return actionMapping.findForward("back");
    }

  }
}
```

A.5 AfterAddRule.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
```

```
import java.io.IOException;
import java.util.*;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class AfterAddRule
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.AfterAddRule.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {
    logger.debug("Entering afterAddRule");

    logger.info("Retreiving necessary attributes");
    CustomBean cb = (CustomBean) request.getSession().getAttribute(
        "customBean");
    if (cb == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("customBean");
    }

    List conditions = (List) request.getSession().getAttribute("conditions");
    if (conditions == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("conditions");
    }

    AddResultBean results = (AddResultBean) request.getSession().getAttribute(
        "results");
    if (results == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("results");
    }
    logger.info("Verifying that the rule meets the correct specifications");
    if (conditions.size() < 1 || results.getVarName() == null ||
        results.getVarName().length < 1) {
      request.setAttribute("customBean", cb);
      request.setAttribute("results", results);
      request.setAttribute("conditions", conditions);
      ActionErrors errors = new ActionErrors();
      errors.add("afterAddRule", new ActionError("not.valid.rule"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }

    logger.info("Converting rule into the correct format");

    List vars = cb.getVars();
    CompoundRule cr = new CompoundRule();
    for (int i = 0; i < conditions.size(); i++) {
      AddConditionBean acb = (AddConditionBean) conditions.get(i);
      String varType = "";
      for (int j = 0; j < vars.size(); j++) {
        VariableBean var = (VariableBean) vars.get(j);
```

```
        if (var.getVarName().equals(acb.getVarName())) {
          varType = var.getVarType();
        }
      }
      boolean isNumCellsBased = false;
      String NumCellsBased = acb.getNumCellsBased();
      if (NumCellsBased != null) {
        isNumCellsBased = NumCellsBased.equals("true");
      }
      int numCells = 0;
      if (acb.getNumCells() != null) {
        numCells = Integer.parseInt(acb.getNumCells());
      }
      RuleBean rule = new RuleBean(isNumCellsBased,
                                   varType, acb.getVarValue(), acb.getOperator(),
                                   acb.getVarName(),
                                   numCells, acb.getNumCellsOperator());
      String con = acb.getConjunction();

      if (con == null || con.equals("||")) {
        cr.addOr(rule);
      }
      else {
        cr.addAnd(rule);
      }
    }
    String[] names = results.getVarName();
    String[] values = results.getVarValue();
    for (int i = 0; i < names.length; i++) {
      VariableChangeValueBean vcvb = new VariableChangeValueBean(names[i],
          values[i]);
      cr.addChange(vcvb);
    }
    List r = cb.getRules();
    if (r == null) {
      r = new ArrayList();
    }
    r.add(cr);
    logger.info("Adding rule");
    cb.setRules(r);
    request.setAttribute("customBean", cb);
    return actionMapping.findForward("success");
  }
}
```

A.6 AfterPopulateSystem.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.List;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class AfterPopulateSystem
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.AfterPopulateSystem.class);
```

```
  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {
    logger.debug("Enter AfterPopulateSystem action");
    logger.info("Retrieving necessary attributes");

    CustomBean cb = (CustomBean) request.getSession().getAttribute("customBean");
    if (cb == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("customBean");
    }
    List list = (List) request.getSession().getAttribute("populatedCells");
    if (list == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("populatedCells");
    }
    logger.info("Saving " + list.size() + " populated cells");
    cb.setPopulatedCells(list);
    request.setAttribute("customBean", cb);
    String buttonPressed = (String) request.getParameter("Submit");
    if (buttonPressed.startsWith("Next")) {
      List lst = cb.getRules();
      if (lst == null) {
        logger.info("Forwarding to makeRule");
        return actionMapping.findForward("makeRule");
      }
      logger.info("Forwarding to listRules");
      return actionMapping.findForward("listRules");
    }
    else {
      logger.info("Forwarding to back");
      return actionMapping.findForward("back");
    }

  }
}
```

A.7 CellularAutomataWriter.java

```
package edu.trinity.Jackal.writers;

import edu.trinity.Jackal.commons.*;
import edu.trinity.Jackal.structures.*;
import java.io.*;
import java.util.*;
import org.apache.log4j.Logger;

public class CellularAutomataWriter {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      writers.CellularAutomataWriter.class);

  public CellularAutomataWriter() {
    logger.debug("Entering CellularAutomataWriter constructor");
  }
```

```
public void writeCellularAutomataClassToFile(List populatedCells, List vars,
                                             List compoundRules,
                                             int size,
                                             String path) throws IOException {
  logger.debug("Entering writeCellularAutomataClassToFile");
  logger.info("NumPopulatedCells = " + populatedCells.size());
  logger.info("NumVars = " + vars.size());
  logger.info("NumCompoundRules = " + compoundRules.size());
  logger.info("Size = " + size);
  logger.info("Path = " + path);

  List list = this.makeCellularAutomataClass(populatedCells, vars, size,
                                             compoundRules);
  WriteToFile.writeToFile(list,
                          path + "cellularautomata\\CellularAutomata.java");
}

private List makeCellularAutomataClass(List populatedCells, List vars,
                                       int size,
                                       List compoundRules) {
  logger.debug("Entering makeCellularAutomataClass");

  List list = new ArrayList();
  list.add("package cellularautomata;");
  list.add("");
  list.addAll(this.makeImportStatements(vars));
  list.add("");
  list.add("public class CellularAutomata{");
  list.add("");
  list.addAll(this.makeClassVariables(size));
  list.add("");
  list.addAll(this.makeConstructor(populatedCells));
  list.add("");
  list.addAll(this.makeGetCells());
  list.add("");
  list.addAll(this.makeUpdate(vars, compoundRules));
  list.add("");
  list.addAll(this.makeNumCellsWithValueFunctions(compoundRules));
  list.add("");
  list.addAll(this.makeProcessStringFunction());
  list.add("}");
  return list;
}

private List makeNumCellsWithValueFunctions(List rules) {
  logger.debug("Entering makeNumCellsWithValueFunctions");

  List list = new ArrayList();
  List vars = new ArrayList();
  for (int i = 0; i < rules.size(); i++) {
    CompoundRule cr = (CompoundRule) rules.get(i);
    List ruleList = cr.getRuleList();
    for (int j = 0; j < ruleList.size(); j++) {
      RuleBean rule = (RuleBean) ruleList.get(j);
      if (!vars.contains(rule.getVarToCheckName())) {
        list.addAll(this.makeNumCellsWithValue(rule));
        list.add("");
        vars.add(rule.getVarToCheckName());
      }
    }
  }
  return list;
}
```

```
private List makeNumCellsWithValue(RuleBean rule) {
  logger.debug("Entering makeNumCellsWithValue");

  List list = new ArrayList();
  list.add("  public int numCellsWith" + rule.getVarToCheckName() +
           "Value(Cell[][] __g, String __operator, " + rule.getVarToCheckType() +
           " __value, int __x, int __y){");
  list.add("     int __minx = __x-1;");
  list.add("     int __miny = __y-1;");
  list.add("     int __maxx = __x+1;");
  list.add("     int __maxy = __y+1;");
  list.add("     int __count = 0;");
  list.add("");
  list.add("     if(__x <= 0)");
  list.add("       __minx = 0;");
  list.add("     if(__y <= 0)");
  list.add("       __miny = 0;");
  list.add("     if(__maxx >= __SIZE-1)");
  list.add("       __maxx = __SIZE-1;");
  list.add("     if(__maxy >= __SIZE-1)");
  list.add("       __maxy = __SIZE-1;");
  list.add("");
  list.add("     if(__maxx <= __minx) return 0;");
  list.add("     if(__maxy <= __miny) return 0;");
  list.add("");
  list.add("      if(__operator.equals(\"<\")){");
  list.add("        for(int __i = __minx; __i <= __maxx; __i++)");
  list.add("         for(int __j = __miny; __j <= __maxy; __j++){");
  list.add("           if(__g[__i][__j].get" + rule.getVarToCheckName() +
           "() < __value && !(__x == __i && __y == __j))");
  list.add("             __count++;");
  list.add("          }");
  list.add("      }");
  list.add("      else if(__operator.equals(\"<=\")){");
  list.add("        for(int __i = __minx; __i <= __maxx; __i++)");
  list.add("         for(int __j = __miny; __j <= __maxy; __j++){");
  list.add("           if(__g[__i][__j].get" + rule.getVarToCheckName() +
           "() <= __value && !(__x == __i && __y == __j))");
  list.add("             __count++;");
  list.add("          }");
  list.add("      }");
  list.add("      else if(__operator.equals(\"==\")){");
  list.add("        for(int __i = __minx; __i <= __maxx; __i++)");
  list.add("         for(int __j = __miny; __j <= __maxy; __j++){");
  list.add("           if(__g[__i][__j].get" + rule.getVarToCheckName() +
           "() == __value && !(__x == __i && __y == __j))");
  list.add("             __count++;");
  list.add("          }");
  list.add("      }");
  list.add("      else if(__operator.equals(\">\")){");
  list.add("        for(int __i = __minx; __i <= __maxx; __i++)");
  list.add("         for(int __j = __miny; __j <= __maxy; __j++){");
  list.add("           if(__g[__i][__j].get" + rule.getVarToCheckName() +
           "() > __value && !(__x == __i && __y == __j))");
  list.add("             __count++;");
  list.add("          }");
  list.add("      }");
  list.add("      else if(__operator.equals(\">=\")){");
  list.add("        for(int __i = __minx; __i <= __maxx; __i++)");
  list.add("         for(int __j = __miny; __j <= __maxy; __j++){");
  list.add("           if(__g[__i][__j].get" + rule.getVarToCheckName() +
           "() >= __value && !(__x == __i && __y == __j))");
  list.add("             __count++;");
```

```
  list.add("          }");
  list.add("      }");
  list.add("      else if(__operator.equals(\"!=\")){");
  list.add("        for(int __i = __minx; __i <= __maxx; __i++)");
  list.add("         for(int __j = __miny; __j <= __maxy; __j++){");
  list.add("           if(__g[__i][__j].get" + rule.getVarToCheckName() +
           "() != __value && !(__x == __i && __y == __j))");
  list.add("            __count++;");
  list.add("          }");
  list.add("      }");

  list.add("      return __count;");
  list.add("  }");
  return list;
}

/**
 * takes in a list VariableBean types, a list of RuleBean types, and a list
 * of CompoundRuleTypes
 *
 * @param vars List
 * @param rules List
 * @return List
 */
private List makeUpdate(List vars, List compoundRules) {
  logger.debug("Entering makeUpdate");

  List list = new ArrayList();
  VariableBean vb;
  list.add("  public void update(){");
  list.add("     Cell[][] __g = new Cell[__SIZE][__SIZE];");
  list.add("");
  list.add(
      "     // coppying the grid so that all updates are made \"simultaneously\"");
  list.add("     for(int __x = 0; __x < __SIZE; __x ++){");
  list.add("        for(int __y = 0; __y < __SIZE; __y++){");
  list.add("           __g[__x][__y] = new Cell();");
  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    list.add("           __g[__x][__y].set" + vb.getVarName() +
             "(__grid[__x][__y].get" + vb.getVarName() + "());");
  }
  list.add("        }");
  list.add("     }");
  list.add("");
  list.add("     for (int __i = 0; __i < __SIZE; __i++) {");
  list.add("        for (int __j = 0; __j < __SIZE; __j++) {");
  list.add("            //making variables available for user rules");
  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    list.add("            " + vb.getVarType() + " " + vb.getVarName() +
             " = __g[__i][__j].get" +
             vb.getVarName() + "();");
  }
  list.add("            //updates go here");

  CompoundRule compoundRule;
  List rules;
  List changes;
  List operators;
  VariableChangeValueBean vcvb;

  for (int q = 0; q < compoundRules.size(); q++) {
    compoundRule = (CompoundRule) compoundRules.get(q);
```

```
    rules = compoundRule.getRuleList();
    changes = compoundRule.getChangeList();
    operators = compoundRule.getConjunctionList();

    list.add("            if(" +
             this.makeVariableComparisons(rules, operators) + "){");

    for (int w = 0; w < changes.size(); w++) {

      vcvb = (VariableChangeValueBean) changes.get(w);
      list.add("               " + this.makeValueChange(vcvb));
    }
    list.add("            }"); //closing if statement
  }

  list.add("        }"); //closing j
  list.add("     }"); //closing i
  list.add("  }"); //closing function
  return list;
}

private String makeValueChange(VariableChangeValueBean vcvb) {
  logger.debug("Entering makeValueChange");

  return "__grid[__i][__j].set" + vcvb.getVarToBeChangedName() + "(" +
      vcvb.getVarToBeChangedValue() + ");";
}

private String makeVariableComparisons(List rules, List operators) {
  logger.debug("Entering makeVariableComparisons");

  String comps = "";
  RuleBean rb;
  for (int i = 0; i < rules.size(); i++) {
    comps += "(";
    rb = (RuleBean) rules.get(i);
    if (rb.isIsBasedOnNumCells()) {
      comps += this.makeNumCellsComparison(rb);
    }
    else {
      comps += this.makeVariableComparison(rb);
    }
    if (i < operators.size()) {
      comps += ") " + (String) operators.get(i) + " ";
    }
    else {
      comps += ")";
    }
  }
  return comps;
}

private String makeVariableComparison(RuleBean rule) {
  logger.debug("Entering makeVariableComparison");
  return "__g[__i][__j].get" + rule.getVarToCheckName() + "() " +
      rule.getVarToCheckOperator() + " " + rule.getVarToCheckVal();
}

private String makeNumCellsComparison(RuleBean rule) {
  logger.debug("Entering makeNumCellsComparison");

  return "numCellsWith" + rule.getVarToCheckName() +
      "Value(__g, \"" + rule.getVarToCheckOperator() + "\" , " +
      rule.getVarToCheckVal() + ", __i, __j) " +
```

```
      rule.getNumCellsOperator() + " " + rule.getNumCells();
}

private List makeGetCells() {
  logger.debug("Entering makeGetCells");

  List list = new ArrayList();
  list.add("  public Cell[][] getCells(){");
  list.add("     return __grid;");
  list.add("  }");
  return list;
}

/**
 * takes in a list of VariableBean types
 *
 * @param vars List
 * @return List
 */
private List makeImportStatements(List vars) {
  logger.debug("Entering makeImportStatements");

  List list = new ArrayList();
  VariableBean vb;
  String imp;
  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    imp = Imports.getImportStatementForType(vb.getVarType());
    if (!imp.equals("") && !list.contains(imp)) {
      list.add(imp);
    }
  }
  list.add("import java.io.*;");
  list.add("import java.util.*;");
  return list;
}

/**
 * takes in a list of VariableBean types
 *
 * @param vars List
 * @return List
 */
private List makeClassVariables(int size) {
  logger.debug("Entering makeClassVariables");

  List list = new ArrayList();
  list.add("  private int __SIZE = " + size + ";");
  list.add("  private Cell[][] __grid = new Cell[__SIZE][__SIZE];");
  return list;
}

/**
 * takes in a list of VariableBean types
 *
 * @param vars List
 * @return List
 */
private List makeConstructor(List populatedCells) {
  logger.debug("Enterint makeConstructor");

  List list = new ArrayList();
  list.add("  public CellularAutomata(){");
  list.add("     //initializing the cells");
```

```
    list.add("     for (int __i = 0; __i < __SIZE; __i++) {");
    list.add("       for (int __j = 0; __j < __SIZE; __j++) {");
    list.add("         __grid[__i][__j] = new Cell();");
    list.add("       }"); //close j
    list.add("     }"); //close i

    list.add("");
    list.add("//reads the populated cells in from datafile within the jar file");
    list.add("     try {");
    list.add("       InputStream fis = CellularAutomata.class.getResourceAsStream(\"populatedcells.dat\");");
    list.add(
        "       BufferedReader in = new BufferedReader(new InputStreamReader(fis));");
    list.add("       String str;");
    list.add("       while ((str = in.readLine()) != null){");
    list.add("           processString(str);");
    list.add("       }");
    list.add("   in.close();");
    list.add("       fis.close();");
    list.add("     }catch (IOException e) {");
    list.add(
        "        System.out.println(\"Error loading populated cells from file\");");
    list.add("     }");

    list.add("  }"); //close constructor
    return list;
  }

  private List makeProcessStringFunction() {
    logger.debug("Entering makeProcessStringFunction");

    List list = new ArrayList();
    list.add("  public void processString(String str){");
    list.add("     StringTokenizer st = new StringTokenizer (str);");
    list.add("     int x = Integer.parseInt(st.nextToken());");
    list.add("     int y = Integer.parseInt(st.nextToken());");
    list.add("     __grid[x][y].set__value(st.nextToken(),st.nextToken());");
    list.add("  }");

    return list;
  }
}
```

A.8 CellWriter.java

```
package edu.trinity.Jackal.writers;

import edu.trinity.Jackal.commons.*;
import edu.trinity.Jackal.structures.VariableBean;
import java.io.*;
import java.util.*;
import org.apache.log4j.Logger;

public class CellWriter {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      writers.CellWriter.class);

  public CellWriter() {
    logger.debug("Entering CellWriter constructor");
  }
```

```
public void writeCellClassToFile(List vars, String path) throws IOException {
  logger.debug("Entering writeCellClassToFile");

  List list = this.makeCellClass(vars);
  WriteToFile.writeToFile(list, path + "cellularautomata\\Cell.java");
}

/**
 * takes in a list of VariableBean types
 *
 * @param vars List
 * @throws InvalidVariableNamesException
 * @return List
 */
private List makeCellClass(List vars) {
  logger.debug("Entering makeCellClass");

  List list = new ArrayList();
  VariableBean vb;
  list.add("package cellularautomata;");
  list.add("");
  list.addAll(this.makeImportStatements(vars));
  list.add("");
  list.add("public class Cell {");
  list.addAll(this.makeClassVariables(vars));
  list.add("");
  list.addAll(this.makeConstructor(vars));
  list.add("");
  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    list.addAll(this.makeGetterFunction(vb));
    list.add("");
    list.addAll(this.makeSetterFunction(vb));
    list.add("");
  }
  list.add("");
  list.addAll(this.makeSetValueFunction(vars));
  list.add("}");
  list.add("");
  return list;
}

private List makeSetValueFunction(List vars) {
  logger.debug("Entering makeSetValueFunction");
  List list = new ArrayList();
  VariableBean vb;

  list.add("  public void set__value(String var, String val){");
  list.add("     if(var.equals(\"color\")){");
  list.add("        if(val.equals(\"Color.black\")){");
  list.add("          setcolor(Color.black);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.blue\")){");
  list.add("          setcolor(Color.blue);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.cyan\")){");
  list.add("          setcolor(Color.cyan);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.darkGray\")){");
  list.add("          setcolor(Color.darkGray);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.gray\")){");
  list.add("          setcolor(Color.gray);");
  list.add("        }");
```

```
  list.add("        else if(val.equals(\"Color.green\")){");
  list.add("          setcolor(Color.green);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.lightGray\")){");
  list.add("          setcolor(Color.lightGray);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.magenta\")){");
  list.add("          setcolor(Color.magenta);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.orange\")){");
  list.add("          setcolor(Color.orange);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.pink\")){");
  list.add("          setcolor(Color.pink);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.red\")){");
  list.add("          setcolor(Color.red);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.white\")){");
  list.add("          setcolor(Color.white);");
  list.add("        }");
  list.add("        else if(val.equals(\"Color.yellow\")){");
  list.add("          setcolor(Color.yellow);");
  list.add("        }");
  list.add("     }");

  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    list.add("     else if(var.equals(\"" + vb.getVarName() + "\")){");
    if (vb.getVarType().equals("int")) {
      list.add("        set" + vb.getVarName() + "(Integer.parseInt(val));");
    }
    else if (vb.getVarType().equals("double")) {
      list.add("        set" + vb.getVarName() + "(Double.parseDouble(val));");
    }
    list.add("     }");
  }

  list.add("  }");
  return list;
}

/**
 * takes in a list of VariableBean types
 *
 * @param vars List
 * @return List
 */
private List makeImportStatements(List vars) {
  logger.debug("Entering makeImportStatements");

  List list = new ArrayList();
  VariableBean vb;
  String imp;
  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    imp = Imports.getImportStatementForType(vb.getVarType());
    if (!imp.equals("") && !list.contains(imp)) {
      list.add(imp);
    }
  }
  return list;
}
```

```
/**
 * takes in a list of VariableBean types
 *
 * @param vars List
 * @return List
 */
private List makeClassVariables(List vars) {
  logger.debug("Entering makeClassVariables");

  List list = new ArrayList();
  VariableBean vb;
  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    list.add("  private " + vb.getVarType() + " " + vb.getVarName() + ";");
  }
  return list;
}

/**
 * takes in a list of VariableBean types
 *
 * @param vars List
 * @return List
 */
private List makeConstructor(List vars) {
  logger.debug("Entering makeConstructor");

  List list = new ArrayList();
  list.add("  public Cell(){");
  VariableBean vb;
  for (int i = 0; i < vars.size(); i++) {
    vb = (VariableBean) vars.get(i);
    list.add("     " + vb.getVarName() + " = " + vb.getVarVal() + ";");
  }
  list.add("  }");
  return list;
}

private List makeSetterFunction(VariableBean var) {
  logger.debug("Entering makeSetterFunction");

  List list = new ArrayList();
  String signature = "  public void set" + var.getVarName() + "(" +
      var.getVarType() + " " + var.getVarName() + "){";
  String body = "     this." + var.getVarName() + " = " + var.getVarName() +
      ";";
  String close = "  }";
  list.add(signature);
  list.add(body);
  list.add(close);
  return list;
}

private List makeGetterFunction(VariableBean var) {
  logger.debug("Entering makeGetterFunction");

  List list = new ArrayList();
  String signature = "  public " + var.getVarType() + " get" + var.getVarName() +
      "(){";
  String body = "     return " + var.getVarName() + ";";
  String close = "  }";
  list.add(signature);
  list.add(body);
  list.add(close);
```

```
    return list;
  }
}
```

A.9 CreateCellularAutomata.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.commons.*;
import edu.trinity.Jackal.exceptions.CompileException;
import edu.trinity.Jackal.structures.*;
import edu.trinity.Jackal.writers.*;
import java.io.*;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class CreateCellularAutomata
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.CreateCellularAutomata.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) {
    logger.debug("Entering CreateCellularAutomata action");

    CustomBean cb = (CustomBean) request.getSession().getAttribute(
        "customBean");
    if (cb != null) {
      request.getSession().removeAttribute("customBean");
    }
    request.setAttribute("customBean", cb);
    String str = (String) request.getParameter("Submit");
    if (!str.startsWith("Finish")) {
      return actionMapping.findForward("back");
    }

    String temp = cb.getCellSize();
    int cellSize = 0;
    if (temp != null) {
      cellSize = Integer.parseInt(temp);
    }
    temp = cb.getNumCellsWide();
    int numCellsWide = 0;
    if (temp != null) {
      numCellsWide = Integer.parseInt(temp);
    }
    temp = cb.getNumIterations();
    int numIterations = 0;
    if (temp != null) {
      numIterations = Integer.parseInt(temp);
    }
    temp = cb.getSpeed();
    int speed = Integer.parseInt(temp);
    logger.debug("Creating a downloadable jar file");
    CustomCellularAutomata ca = new CustomCellularAutomata(cb.getVars(),
        cb.getRules(), cb.getPopulatedCells(), "", cellSize, numCellsWide,
        numIterations, speed);
```

```
    try {
      logger.info("Writing code to file");
      ca.createCustomAutomata();
      logger.info("Writing manifest to file");
      ManifestWriter mw = new ManifestWriter();
      mw.writeManifest("DisplayCellularAutomata", "");
      try {
        new Thread().sleep(4000);
        logger.info("Back from sleeping");
        String location = CreateExecutable.CreateJarFile(cb.getTitle());
        logger.info("Created Jar file");
        request.setAttribute("location", location);
      }
      catch (InterruptedException ex1) {
        response.sendRedirect("/index.jsp");
      }
      catch (IOException ex1) {
        response.sendRedirect("/index.jsp");
      }
      catch (CompileException ex1) {
        ActionErrors errors = new ActionErrors();
        errors.add("createCellularAutomata", new ActionError("compile.error"));
        this.saveErrors(request, errors);
        return actionMapping.findForward("failure");
      }
    }
    catch (IOException ex) {
      logger.error("IOException");
      ActionErrors errors = new ActionErrors();
      errors.add("createCellularAutomata", new ActionError("server.error"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }
    return actionMapping.findForward("success");
  }
}
```

A.10 CreateExecutable.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.commons.*;
import edu.trinity.Jackal.exceptions.CompileException;
import edu.trinity.Jackal.structures.*;
import edu.trinity.Jackal.writers.*;
import java.io.*;
import javax.servlet.http.*;
import org.apache.log4j.Logger;
import org.apache.struts.action.*;

public class CreateCellularAutomata
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.CreateCellularAutomata.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) {
    logger.debug("Entering CreateCellularAutomata action");
```

```
CustomBean cb = (CustomBean) request.getSession().getAttribute(
    "customBean");
if (cb != null) {
  request.getSession().removeAttribute("customBean");
}
request.setAttribute("customBean", cb);
String str = (String) request.getParameter("Submit");
if (!str.startsWith("Finish")) {
  return actionMapping.findForward("back");
}

String temp = cb.getCellSize();
int cellSize = 0;
if (temp != null) {
  cellSize = Integer.parseInt(temp);
}
temp = cb.getNumCellsWide();
int numCellsWide = 0;
if (temp != null) {
  numCellsWide = Integer.parseInt(temp);
}
temp = cb.getNumIterations();
int numIterations = 0;
if (temp != null) {
  numIterations = Integer.parseInt(temp);
}
temp = cb.getSpeed();
int speed = Integer.parseInt(temp);
logger.debug("Creating a downloadable jar file");
CustomCellularAutomata ca = new CustomCellularAutomata(cb.getVars(),
    cb.getRules(), cb.getPopulatedCells(), "", cellSize, numCellsWide,
    numIterations, speed);
try {
  logger.info("Writing code to file");
  ca.createCustomAutomata();
  logger.info("Writing manifest to file");
  ManifestWriter mw = new ManifestWriter();
  mw.writeManifest("DisplayCellularAutomata", "");
  try {
    new Thread().sleep(4000);
    logger.info("Back from sleeping");
    String location = CreateExecutable.CreateJarFile(cb.getTitle());
    logger.info("Created Jar file");
    request.setAttribute("location", location);
  }
  catch (InterruptedException ex1) {
    response.sendRedirect("/index.jsp");
  }
  catch (IOException ex1) {
    response.sendRedirect("/index.jsp");
  }
  catch (CompileException ex1) {
    ActionErrors errors = new ActionErrors();
    errors.add("createCellularAutomata", new ActionError("compile.error"));
    this.saveErrors(request, errors);
    return actionMapping.findForward("failure");
  }
}
catch (IOException ex) {
  logger.error("IOException");
  ActionErrors errors = new ActionErrors();
  errors.add("createCellularAutomata", new ActionError("server.error"));
  this.saveErrors(request, errors);
```

```
      return actionMapping.findForward("failure");
    }
    return actionMapping.findForward("success");
  }
}
```

A.11 CustomCellularAutomata.java

```
package edu.trinity.Jackal.writers;

import java.io.IOException;
import java.util.List;
import org.apache.log4j.Logger;

public class CustomCellularAutomata {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      writers.CustomCellularAutomata.class);
  List vars;
  List compoundRules;
  List populatedCells;
  String path;
  int cellSize;
  int numCellsWide;
  int numIterations;
  int speed;

  public CustomCellularAutomata(List vars, List compoundRules,
                                List populatedCells, String path, int cellSize,
                                int numCellsWide, int numIterations, int speed) {
    logger.debug("Entering CellBasedCellularAutomata constructor");
    this.vars = vars;
    this.compoundRules = compoundRules;
    this.populatedCells = populatedCells;
    this.path = path;
    this.cellSize = cellSize;
    this.numCellsWide = numCellsWide;
    this.numIterations = numIterations;
    this.speed = speed;
  }

  public void createCustomAutomata() throws IOException {
    logger.debug("Entering createCellBasedCellularAutomata");
    CellWriter cw = new CellWriter();
    cw.writeCellClassToFile(vars, path);

    CellularAutomataWriter caw = new CellularAutomataWriter();
    caw.writeCellularAutomataClassToFile(populatedCells, vars, compoundRules,
                                         numCellsWide, path);

    DisplayCellularAutomataWriter dcaw = new DisplayCellularAutomataWriter();
    dcaw.writeCellClassToFile(cellSize, numCellsWide, numIterations, speed,
                              path);

    PopulatedCellsDataFileWriter pcdfw = new PopulatedCellsDataFileWriter();
    pcdfw.writePopulatedCellsToFile(populatedCells);
  }
}
```

A.12 CustomCellularAutomataStart.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import javax.servlet.http.*;
import org.apache.log4j.*;
import org.apache.struts.action.*;


public class CustomCellularAutomataStart
    extends Action {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      ui.action.CustomCellularAutomataStart.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) {

    logger.debug("entering createCellBasedAction");

    CustomBean cellbean = (CustomBean)
        actionForm;

    ActionErrors errors = new ActionErrors();

    //New CA must have a title
    if (cellbean.getTitle().equals("")) {
      errors.add("createCellBasedAction",
                 new ActionError("create.empty.title"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }
    cellbean.setTitle(cellbean.getTitle().trim().replaceAll(" ", "_"));

    //setting default values if user failes to input anything
    if (cellbean.getNumCellsWide().equals("") ||
        !this.isNumber(cellbean.getNumCellsWide())) {
      logger.info("invalid system width...replacing value with default");
      cellbean.setNumCellsWide("50");
    }
    if (Integer.parseInt(cellbean.getNumCellsWide()) < 2 ||
        Integer.parseInt(cellbean.getNumCellsWide()) > 100) {
      errors.add("createCellBasedAction", new ActionError("create.invalid.size"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }
    if (cellbean.getCellSize().equals("") ||
        !this.isNumber(cellbean.getCellSize())) {
      logger.info("invalid cellSize...replacing value with default");
      cellbean.setCellSize("5");
    }
    if (cellbean.getNumIterations().equals("") ||
        !this.isNumber(cellbean.getNumIterations())) {
      logger.info("invalid numIterations...replacing value with default");
      cellbean.setNumIterations("-1");
    }
    request.setAttribute("customBean", cellbean);
    return actionMapping.findForward("success");
  }
```

```
  private boolean isNumber(String num) {
    logger.debug("Entering isNumber");
    try {
      Integer.parseInt(num);
      return true;
    }
    catch (NumberFormatException ex) {
      return false;
    }
  }
}
```

A.13 DisplayCellularAutomataWriter.java

```
package edu.trinity.Jackal.writers;

import java.io.*;
import java.util.*;
import org.apache.log4j.Logger;

public class DisplayCellularAutomataWriter {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      writers.DisplayCellularAutomataWriter.class);

  public DisplayCellularAutomataWriter() {
    logger.debug("Entering DisplayCellularAutomataWriter");
  }

  public void writeCellClassToFile(int cellSize, int numCellsWide,
                                   int numIterations, int speed, String path) throws
      IOException {
    logger.debug("Entering writeCellClassToFile");

    List list = this.makeDisplayCellularAutomataClass(cellSize, numCellsWide,
        numIterations, speed);
    WriteToFile.writeToFile(list,
                            path + "cellularautomata\\DisplayCellularAutomata.java");
  }

  private List makeDisplayCellularAutomataClass(int cellSize, int numCellsWide,
                                                int numIterations, int speed) {
    logger.debug("Entering makeDisplayCellularAutomataClass");

    List list = new ArrayList();
    list.add("package cellularautomata;");
    list.add("");
    list.add("import javax.swing.*;");
    list.add("import java.awt.*;");
    list.add("import java.awt.event.*;");
    list.add("");
    list.add("public class DisplayCellularAutomata");
    list.add("   extends JPanel");
    list.add("   implements Runnable {");
    list.add("");
    list.add("  private final int SIZE = " + numCellsWide + ";");
    list.add("  private final int SQUARE_SIZE = " + cellSize + ";");
    list.add("  private final int NUM_ITERATIONS = " + numIterations + ";");
    list.add("  private final int WIDTH = SIZE * (SQUARE_SIZE + 1) + 7;");
    list.add("  private final int HEIGHT = SIZE * (SQUARE_SIZE + 1) + 33;");
    list.add("");
```

```
list.add("  private Cell[][] cells = null;");
list.add("  private CellularAutomata ca = null;");
list.add("  private int count = 0;");
list.add("");
list.add("  public DisplayCellularAutomata() {");
list.add("     this.setSize(WIDTH, HEIGHT);");
list.add("     ca = new CellularAutomata();");
list.add("     cells = ca.getCells();");
list.add("  }");
list.add("");
list.add("  public void paintComponent(Graphics g) {");
list.add("     draw();");
list.add("     if (count != NUM_ITERATIONS) {");
list.add("        try {");
if (speed == 1) {
  list.add("           Thread.sleep(350);");
}
if (speed == 2) {
  list.add("           Thread.sleep(300);");
}
if (speed == 3) { //normal
  list.add("           Thread.sleep(250);");
}
if (speed == 4) {
  list.add("           Thread.sleep(200);");
}
if (speed == 5) {
  list.add("           Thread.sleep(150);");
}

list.add("        }");
list.add("        catch (InterruptedException ex) {");
list.add("           ex.printStackTrace();");
list.add("        }");
list.add("        count++;");
list.add("        this.repaint();");
list.add("     }");
list.add("  }");
list.add("");
list.add("  public void run() {");
list.add("     draw();");
list.add("  }");
list.add("");
list.add("  private void draw() {");
list.add("     Graphics g = this.getGraphics();");
list.add("     g.setColor(Color.white);");
list.add("     g.fillRect(0, 0, this.WIDTH, this.HEIGHT);");
list.add("     displayGrid(g);");
list.add("");
list.add("     displayCells(g);");
list.add("     ca.update();");
list.add("     cells = ca.getCells();");
list.add("  }");
list.add("");
list.add("  public static void main(String args[]) {");
list.add("     JFrame f = new JFrame(\"Cellular Automata\");");
list.add(
    "     DisplayCellularAutomata display = new DisplayCellularAutomata();");
list.add("");
list.add("     f.getContentPane().add(display);");
list.add("     f.setSize(display.getSize());");
list.add("     f.setLocation(100, 100);");
list.add("     f.setResizable(false);");
list.add("     f.setVisible(true);");
```

```
    list.add("");
    list.add("     f.addWindowListener(new WindowAdapter() {");
    list.add("        public void windowClosing(WindowEvent e) {");
    list.add("           System.exit(0);");
    list.add("        } //windowClosing end");
    list.add("     }); //addWindowListener end");
    list.add("");
    list.add("  }");
    list.add("");
    list.add("  private void displayGrid(Graphics g) {");
    list.add("");
    list.add("     for (int i = 0; i <= this.SIZE; i++) {");
    list.add("        if (i % 10 == 0) {");
    list.add("           g.setColor(Color.black);");
    list.add("        }");
    list.add("        else {");
    list.add("           g.setColor(Color.lightGray);");
    list.add("        }");
    list.add("        g.drawLine(0, i * (this.SQUARE_SIZE + 1), this.WIDTH, i * (this.SQUARE_SIZE + 1));");
    list.add("     }");
    list.add("     for (int j = 0; j <= this.SIZE; j++) {");
    list.add("        if (j % 10 == 0) {");
    list.add("           g.setColor(Color.black);");
    list.add("        }");
    list.add("        else {");
    list.add("           g.setColor(Color.lightGray);");
    list.add("        }");
    list.add("        g.drawLine(j * (this.SQUARE_SIZE + 1), 0, j * (this.SQUARE_SIZE + 1), this.HEIGHT);");
    list.add("     }");
    list.add("  }");
    list.add("");
    list.add("  private void displayCells(Graphics g) {");
    list.add("     for (int i = 0; i < cells.length; i++) {");
    list.add("        for (int j = 0; j < cells[i].length; j++) {");
    list.add("           g.setColor(cells[i][j].getcolor());");
    list.add("           g.fillRect(i * (this.SQUARE_SIZE + 1) + 1, j * (this.SQUARE_SIZE + 1) + 1, this.SQUARE_SIZE, t
    list.add("        } //close j");
    list.add("     } //close i");
    list.add("  }");
    list.add("}");
    return list;

  }

}
```

A.14 ManifestWriter.java

```
package edu.trinity.Jackal.writers;

import java.io.IOException;
import java.util.*;
import org.apache.log4j.Logger;

public class ManifestWriter {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      writers.ManifestWriter.class);

  public ManifestWriter() {
    logger.debug("Entering ManifestWriter constructor");
```

```
  }

  public void writeManifest(String mainClass, String path) throws IOException {
    logger.debug("Entering writeManifest");

    List list = new ArrayList();
    list.add("Manifest-Version: 1.2");
    list.add("Main-Class: cellularautomata." + mainClass);
    list.add("");
    WriteToFile.writeToFile(list, path + "META-INF\\MANIFEST.MF");
  }
}
```

A.15 PopulatedCellsDataFileWriter.java

```
package edu.trinity.Jackal.writers;

import edu.trinity.Jackal.structures.*;
import java.io.*;
import java.util.List;
import org.apache.log4j.Logger;

public class PopulatedCellsDataFileWriter {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      writers.PopulatedCellsDataFileWriter.class);

  public PopulatedCellsDataFileWriter() {
    logger.debug("Entering PopulatedCellsDataFileWriter constructor");
  }

  public void writePopulatedCellsToFile(List cells) {
    logger.debug("Entering writePopulatedCellsToFile");
    logger.info("There are " + cells.size() + " populated cells");

    try {
      BufferedWriter out = new BufferedWriter(new FileWriter(
          "cellularautomata\\populatedcells.dat"));

      for (int i = 0; i < cells.size(); i++) {
        PopulateSystemBean bean = (PopulateSystemBean) cells.get(i);
        String x = bean.getXval();
        String y = bean.getYval();
        String[] names = bean.getVarName();
        String[] values = bean.getVarVal();
        for (int j = 0; j < names.length; j++) {
          out.write(x + " " + y + " " + names[j] + " " + values[j]);
          out.newLine();
        }
      }
      out.close();
    }
    catch (IOException e) {
    }
  }

}
```

A.16 PopulateSystem.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.commons.*;
import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.*;
import javax.servlet.http.*;
import org.apache.log4j.*;
import org.apache.struts.action.*;

public class PopulateSystem
    extends Action {
  private Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.ui.
      action.PopulateSystem.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {
    logger.debug("Entering PopulateSystem action");

    PopulateSystemBean populateSystemBean = (PopulateSystemBean) actionForm;
    logger.info("populateSystemBean.xval = " + populateSystemBean.getXval());
    List list = (List) request.getSession().getAttribute("populatedCells");
    if (list == null) {
      list = new ArrayList();
    }
    else {
      request.getSession().removeAttribute("populatedCells");
    }
    CustomBean cb = (CustomBean) request.getSession().getAttribute(
        "customBean");
    if (cb == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("customBean");
    }
    request.setAttribute("customBean", cb);

    ActionErrors errors = new ActionErrors();

    if (list.size() ==
        Integer.parseInt(cb.getNumCellsWide()) *
        Integer.parseInt(cb.getNumCellsWide())) {
      request.setAttribute("populatedCells", list);
      errors.add("populateSystem", new ActionError("populate.full"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }

    String numToFill = populateSystemBean.getNumToFill();
    int num = Integer.parseInt(numToFill);

    logger.info("Checking to see if the cell has already been populated");
    String xval = populateSystemBean.getXval();
    String yval = populateSystemBean.getYval();
    int size = Integer.parseInt(cb.getNumCellsWide());
    boolean isrand = false;
    if (xval.equals("random") || yval.equals("random")) {
```

```
  isrand = true;
}
if (!isrand) {
  num = 1;
}
logger.info("Num = " + num);
for (int count = 0; count < num; count++) {
  logger.info("Count = " + count);
  boolean ok = false;
  while (!ok) {
    logger.info("In while(!ok) loop");
    if (populateSystemBean.getXval().equals("random")) {
      xval = Integer.toString( (int) ( (Math.random() * 100) % size));
    }
    if (populateSystemBean.getYval().equals("random")) {
      yval = Integer.toString( (int) ( (Math.random() * 100) % size));
    }
    logger.info("xval = " + xval + " ---- yval = " + yval);
    ok = true;
    for (int i = 0; i < list.size(); i++) {
      PopulateSystemBean psb = (PopulateSystemBean) list.get(i);
      if (psb.getXval().equals(xval) && psb.getYval().equals(yval)) {
        if (isrand) {
          ok = false;
        }
        else {
          errors.add("populateSystem", new ActionError("populate.same.cell"));
          this.saveErrors(request, errors);
          request.setAttribute("populatedCells", list);
          return actionMapping.findForward("failure");
        }
      }
    }
  }

  String[] vals = populateSystemBean.getVarVal();
  String[] types = populateSystemBean.getVarType();
  String[] names = populateSystemBean.getVarName();

  ArrayList v = new ArrayList();
  ArrayList t = new ArrayList();
  ArrayList n = new ArrayList();
  logger.info("Harvesting only the data that the users have entered");
  for (int i = 0; i < vals.length; i++) {
    if (!vals[i].trim().equals("")) {
      logger.info("!vals[i].equals(\"\")");
      if (!VariableChecker.checkTypeValuePair(types[i], vals[i])) {
        logger.info("populateSystemBean info was not valid");
        errors.add("populateSystem",
                   new ActionError("variable.type.value.error"));
        this.saveErrors(request, errors);
        request.setAttribute("populatedCells", list);
        return actionMapping.findForward("failure");
      }
      else {
        v.add(vals[i]);
        t.add(types[i]);
        n.add(names[i]);
      }
    }
  }
  PopulateSystemBean __populateSystemBean = new PopulateSystemBean();
  __populateSystemBean.setVarType(t.toArray());
  __populateSystemBean.setVarVal(v.toArray());
```

```
      __populateSystemBean.setVarName(n.toArray());
      __populateSystemBean.setXval(xval);
      __populateSystemBean.setYval(yval);

      list.add(__populateSystemBean);
      logger.info("numPopulatedCells = " + list.size());
    }
    request.setAttribute("populatedCells", list);
    return actionMapping.findForward("success");
  }
}
```

A.17 RemoveCellVariable.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.*;
import javax.servlet.http.*;
import org.apache.log4j.*;
import org.apache.struts.action.*;

public class RemoveCellVariable
    extends Action {
  private Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.ui.
      action.RemovePopulatedCellAction.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {

    logger.debug("Entering RemovePopulatedCellAction");

    IndexBean indexBean = (IndexBean) actionForm;
    List list = (List) request.getSession().getAttribute("vars");
    if (list == null) {
      list = new ArrayList();
    }
    else {
      request.getSession().removeAttribute("vars");
    }
    CustomBean cb = (CustomBean) request.getSession().getAttribute(
        "customBean");
    if (cb == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("customBean");
    }
    request.setAttribute("customBean", cb);

    ActionErrors errors = new ActionErrors();

    int index = 0;
    try {
      index = Integer.parseInt(indexBean.getIndex());
    }
    catch (NumberFormatException ex) {
```

```
      response.sendRedirect("index");
    }
    if (index == 0) {
      request.setAttribute("populatedCells", list);
      errors.add("removeCellVariable", new ActionError("remove.color"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }
    list.remove(index);

    request.setAttribute("vars", list);
    return actionMapping.findForward("success");

  }
}
```

A.18 RemovePopulatedCellAction.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import java.util.*;
import java.io.IOException;
import javax.servlet.http.*;
import org.apache.log4j.*;
import org.apache.struts.action.*;

public class RemovePopulatedCellAction
    extends Action {

  private Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.ui.
      action.RemovePopulatedCellAction.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {

    logger.debug("Entering RemovePopulatedCellAction");

    IndexBean indexBean = (IndexBean) actionForm;
    List list = (List) request.getSession().getAttribute("populatedCells");
    if (list == null) {
      list = new ArrayList();
    }
    else {
      request.getSession().removeAttribute("populatedCells");
    }
    CustomBean cb = (CustomBean) request.getSession().getAttribute(
        "customBean");
    if (cb == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("customBean");
    }
    request.setAttribute("customBean", cb);

    int index = 0;
    try {
```

```
      index = Integer.parseInt(indexBean.getIndex());
    }
    catch (NumberFormatException ex) {
      response.sendRedirect("index");
    }
    list.remove(index);

    request.setAttribute("populatedCells", list);
    return actionMapping.findForward("success");

  }
}
```

A.19 RemoveRule.java

```
package edu.trinity.Jackal.ui.action;

import edu.trinity.Jackal.structures.*;
import java.io.IOException;
import java.util.*;
import javax.servlet.http.*;
import org.apache.log4j.*;
import org.apache.struts.action.*;

public class RemoveRule
    extends Action {

  private Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.ui.
      action.RemoveRule.class);

  public ActionForward execute(ActionMapping actionMapping,
                               ActionForm actionForm,
                               HttpServletRequest request,
                               HttpServletResponse response) throws IOException {

    logger.debug("Entering RemoveRule action");

    IndexBean indexBean = (IndexBean) actionForm;

    CustomBean cb = (CustomBean) request.getSession().getAttribute(
        "customBean");
    if (cb == null) {
      response.sendRedirect("index");
    }
    else {
      request.getSession().removeAttribute("customBean");
    }

    List list = cb.getRules();

    ActionErrors errors = new ActionErrors();

    int index = 0;
    try {
      index = Integer.parseInt(indexBean.getIndex());
    }
    catch (NumberFormatException ex) {
      response.sendRedirect("index");
    }
```

```
    if (list.size() == 1) {
      request.setAttribute("customBean", cb);
      errors.add("removeRule", new ActionError("remove.rule"));
      this.saveErrors(request, errors);
      return actionMapping.findForward("failure");
    }

    list.remove(index);
    cb.setRules(list);
    request.setAttribute("customBean", cb);
    return actionMapping.findForward("success");

  }
}
```

A.20 VariableChecker.java

```
package edu.trinity.Jackal.commons;

import java.util.*;
import org.apache.log4j.Logger;

public class VariableChecker {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.commons.VariableChecker.class);

  /**
   * Checks to make sure that the value of the variable is appropriate for the
   * given datatype.
   *
   * @param type String
   * @param value String
   * @return boolean
   */
  public static boolean checkTypeValuePair(String type, String value) {
    logger.debug("Entering checkTypeValuePair");
    logger.info("Type = " + type);
    logger.info("Value = " + value);

    if (type.equals("int")) {
      try {
        Integer.parseInt(value);
        return true;
      }
      catch (NumberFormatException ex) {
        return false;
      }
    }
    if (type.equals("double")) {
      try {
        Double.parseDouble(value);
        return true;
      }
      catch (NumberFormatException ex1) {
        return false;
      }
    }
    return true;
  }

  /**
```

```
 * Checks to make sure the variable name is not a key word in java,
 * contains any special characters, or is a variable used within the custom
 * cellular automata program.
 *
 * @param name String
 * @return boolean
 */
public static boolean checkName(String name) {
  logger.debug("Entering checkName");
  logger.info("Name = " + name);

  List list = new ArrayList();
  list.add("color");
  list.add("int");
  list.add("double");
  list.add("float");
  list.add("String");
  list.add("char");
  list.add("Color");
  list.add("int");
  list.add("boolean");
  list.add("true");
  list.add("false");
  list.add("private");
  list.add("public");
  list.add("protected");
  list.add("static");
  list.add("import");
  list.add("package");
  list.add("new");
  list.add("void");
  list.add("if");
  list.add("return");
  list.add("for");
  list.add("while");
  list.add("try");
  list.add("catch");
  list.add("do");
  list.add("switch");
  list.add("Cell");
  list.add("CellularAutomata");
  list.add("DisplayCellularAutomata");
  list.add("getCells");
  list.add("update");
  list.add("Integer");
  list.add("Double");

  if (list.contains(name)) {
    return false;
  }
  if (contains(name, "!@#$%^&*()-+={[}]:;\"'\\|<,>.?/~`")) {
    return false;
  }
  if (startswith(name, "1234567890")) {
    return false;
  }
  if (name.startsWith("__")){
    return false;
  }

  return true;
}

/**
```

```
   * Checks to see if the string starts with any potentially harmful characters.
   *
   * @param str String
   * @param nums String
   * @return boolean
   */
  private static boolean startswith(String str, String nums) {
    logger.debug("Entering startswith");
    logger.info("Str = " + str);
    logger.info("Nums = " + nums);

    for (int i = 0; i < nums.length(); i++) {
      if (str.startsWith(nums.substring(i, i + 1))) {
        return true;
      }
    }
    return false;
  }

  /**
   * Checks to see if a string contains any potentially harmful characters.
   *
   * @param str String
   * @param chars String
   * @return boolean
   */
  private static boolean contains(String str, String chars) {
    logger.debug("Entering contains");
    logger.info("Str = " + str);
    logger.info("Chars = " + chars);

    char[] c = chars.toCharArray();
    for (int i = 0; i < c.length; i++) {
      if (str.indexOf(c[i]) > 0) {
        return true;
      }
    }
    return false;
  }
}
```

A.21 WriteToFile.java

```
package edu.trinity.Jackal.writers;

import java.io.*;
import java.util.List;
import org.apache.log4j.Logger;

public class WriteToFile {
  private static Logger logger = (Logger) Logger.getInstance(edu.trinity.Jackal.
      writers.WriteToFile.class);

  public static void writeToFile(List lines, String filename) throws
      IOException {
    logger.debug("Entering writeToFile");
    logger.info("NumLines = " + lines.size());
    logger.info("FileName = " + filename);

    File f = new File(filename);
```

```
    if (!f.canRead()) {
      f = new File(f.getParent());
      f.mkdir();
    }
    FileWriter fw = new FileWriter(filename);
    BufferedWriter out = new BufferedWriter(fw);
    for (int i = 0; i < lines.size(); i++) {
      out.write( (String) lines.get(i));
      out.newLine();
    }
    out.close();

  }
}
```

Appendix B

Web Pages

B.1 addCellVariables.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ page import="edu.trinity.Jackal.structures.*, edu.trinity.Jackal.commons.*, java.util.*"%>

<html>
<head>
<title>Configure Cells</title>
<%String url = "http://" + Config.getProperty("IP_ADDRESS") + ":" + Config.getProperty("PORT") + Config.getProperty("ST
<style type="text/css">@import url("<%=url%>");</style>
</head>
<body>
<jsp:include flush="true" page="header.jsp" />

<div id="main">
<h2>Configure Cells</h2>
<%CustomBean bean = (CustomBean)request.getAttribute("customBean"); %>
<h3><%=bean.getTitle()%></h3>
<p>You must specify the initial color for the cells as well as one additional variable.
<br />To add a variable click the Add Variable button.
<br />To pre-populate the system click the Next button.
<br />If you go back, you will lose all variables, populated cells, and rules.</p>
<div id="error">
<html:errors/>
</div>

<%
  List list = bean.getVars();
  if(list == null){
    list = (List)request.getAttribute("vars");
    if(list == null){
        list = new ArrayList();
    }
  }
  if(list.size() > 0){%>
  <table>
    <thead>
      <tr>
```

```
        <th>Name</th>
        <th>Type</th>
        <th>Value</th>
      </tr>
    </thead>
    <%
    for(int i=0; i< list.size();i++){
      %>
      <tr><%
        VariableBean vb = (VariableBean)list.get(i);%>
        <td><%=vb.getVarName()%></td>
        <td><%=vb.getVarType()%></td>
        <td><%=vb.getVarVal()%></td>
        <%if(i != 0){ %>
        <td>
          <html:form action="/removeCellVariable" method="post">
            <html:hidden property="index" value="<%=Integer.toString(i)%>" />
            <html:submit>Remove</html:submit>
          </html:form>
        </td>
        <%}%>
      </tr>
  <%}
  %></table>
  <br />
  <br />
  <%
  }
%>

<html:form action="/addCellVariable" method="post">
  <table>
    <tr>
      <td>
        Variable name :
      </td>
      <td>
        <%if(list.size() == 0){%>
            <html:text property="varName" value="color" readonly="true"/>
        <%}
        else{%>
        <html:text property="varName" value="" />
        <%}%>
      </td>
    </tr>
    <tr>
      <td>
        Variable type :
      </td>
      <td>
        <html:select property="varType">
          <%if(list.size() == 0){ %>
              <html:option value="Color">Color</html:option>
          <%}else{ %>
              <html:option value="int">Integer</html:option>
              <html:option value="double">Real Number</html:option>
          <%}%>
        </html:select>
      </td>
    </tr>
    <tr>
      <td>
        Initial value :
      </td>
```

```
      <td>
        <%if(list.size() == 0){%>
        <html:select property="varVal">
          <html:option value="Color.black">Black</html:option>
          <html:option value="Color.blue">Blue</html:option>
          <html:option value="Color.cyan">Cyan</html:option>
          <html:option value="Color.darkGray">Dark Gray</html:option>
          <html:option value="Color.gray">Gray</html:option>
          <html:option value="Color.green">Green</html:option>
          <html:option value="Color.lightGray">Light Gray</html:option>
          <html:option value="Color.magenta">Magenta</html:option>
          <html:option value="Color.orange">Orange</html:option>
          <html:option value="Color.pink">Pink</html:option>
          <html:option value="Color.red">Red</html:option>
          <html:option value="Color.white">White</html:option>
          <html:option value="Color.yellow">Yellow</html:option>
        </html:select>
        <%}else{%>
        <html:text property="varVal" value="" />
        <%}%>
      </td>
    </tr>
  </table>
  <%request.getSession().setAttribute("vars",list);
    request.getSession().setAttribute("customBean", bean);%>
  <html:submit property="Submit" value="Add Variable"/>
</html:form>
<br />
<br />
<html:form action="/afterAddCellVariables" method="post">
  <html:submit property="Submit" value="<<  Previous" />
  <html:submit property="Submit" value="Next      >>" />
</html:form>

</div>

</body>
</html>
```

B.2 customStart.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ page import="edu.trinity.Jackal.commons.*, java.util.*"%>

<html>
<head>
<title>Custom Cellular Automata Start</title>
<%String url = "http://" + Config.getProperty("IP_ADDRESS") + ":" + Config.getProperty("PORT") + Config.getProperty("ST
<style type="text/css">@import url("<%=url%>");</style>
</head>
<body>
<jsp:include flush="true" page="header.jsp" />

<div id="main">

<h2>Custom Cellular Automata Creator</h2>

<div id="error">
<html:errors/>
</div>
```

```
<html:form action="/customCellularAutomataStart" method="post">
  <table>
        <tr>
          <td>
            Title for your cellular automata program :
          </td>
          <td>
            <html:text property="title" />
          </td>
        </tr>
        <tr>
          <td>
            Width of the grid (2-100; 50 is the default value):
          </td>
          <td>
            <html:text property="numCellsWide" />
          </td>
        </tr>
        <tr>
          <td>
            Cell size (in pixels; 5 is the default value):
          </td>
          <td>
            <html:text property="cellSize" />
          </td>
        </tr>
        <tr>
          <td>
            Number of iterations (-1 is infinite and is the default value) :
          </td>
          <td>
            <html:text property="numIterations" />
          </td>
        </tr>
        <tr>
          <td>
            Speed (1 is the slowest and 3 is normal) :
          </td>
          <td>
            <html:select property="speed" value="3">
              <html:option value="1">1</html:option>
              <html:option value="2">2</html:option>
              <html:option value="3">3</html:option>
              <html:option value="4">4</html:option>
              <html:option value="5">5</html:option>
            </html:select>
          </td>
        </tr>
  </table>
  <html:submit value="Next Step  >>" property="Submit"/>
</html:form>

</div>

</body>
</html>
```

B.3 makeRules.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ page import="edu.trinity.Jackal.commons.*, edu.trinity.Jackal.structures.*, java.util.*"%>

<script language="JavaScript" type="text/javascript">
     <!-- Beginning of JavaScript -->

    function correctInput (select) {
        if(select.options[select.selectedIndex].value == "true"){
          getElement("numCells").style.visibility='visible';
          getElement("numCellsOperator").style.visibility='visible';
          getElement("numCells").disabled="";
          getElement("numCellsOperator").disabled="";
        }
        else{
          getElement("numCells").style.visibility='hidden';
          getElement("numCellsOperator").style.visibility='hidden';
          getElement("numCells").disabled="true";
          getElement("numCellsOperator").disabled="true";
          getElement("numCells").value="";
          getElement("numCellsOperator").value="";
        }
     }

     function getElement(psID) {
        if(document.all) {
           return document.all[psID];
        } else if(document.getElementById) {
           return document.getElementById(psID);
        } else {
           for (iLayer = 1; iLayer < document.layers.length; iLayer++) {
              if(document.layers[iLayer].id == psID)
                 return document.layers[iLayer];
           }

        }

        return Null;
     }
     <!-- End of JavaScript - -->
  </script>

<html>
<head>
<title>Make Cellular Automata Rules</title>
<%String url = "http://" + Config.getProperty("IP_ADDRESS") + ":" + Config.getProperty("PORT") + Config.getProperty("ST
<style type="text/css">@import url("<%=url%>");</style>

</head>
<body>

<jsp:include flush="true" page="header.jsp" />

<div id="main">

<h2>Rules</h2>



<%CustomBean cb;
cb = (CustomBean)request.getSession().getAttribute("customBean");
```

```
if(cb != null){
  request.getSession().removeAttribute("customBean");
}
else{
  cb = (CustomBean) request.getAttribute("customBean");
}
request.getSession().setAttribute("customBean",cb);
List conditions = (List)request.getAttribute("conditions");
if(conditions == null){
  conditions = new ArrayList();
}
request.getSession().setAttribute("conditions",conditions);
AddResultBean results = (AddResultBean)request.getAttribute("results");
if(results == null){
  results = new AddResultBean();
}
request.getSession().setAttribute("results",results);
%>

<h3><%=cb.getTitle()%></h3>
<div id="error">
<html:errors/>
</div>

<h5>
  <%if(conditions.size() > 0){%>
  <h4>Current Rule</h4>
  if 
  <%for(int i = 0; i < conditions.size(); i++){
      AddConditionBean acb = (AddConditionBean) conditions.get(i);
      if(acb.getConjunction() != null){%>
       <%=acb.getConjunction()%><br />   
      <%}
      if(acb.getNumCellsBased().equals("true")){%>
      numCellsWithValue(  
      <%}%>
      <%=acb.getVarName()%>   <%=acb.getOperator()%>   <%=acb.getVarValue()%>
      <%if(acb.getNumCellsBased().equals("true")){%>
          <%=" ) " + acb.getNumCellsOperator() + " " + acb.getNumCells()%>
      <%}%>
  <%}
  }
  String [] names = results.getVarName();
  String [] values = results.getVarValue();
  if(names!=null){
    if(names.length > 0){%>
  <br />then   <%
      for(int j = 0; j < names.length; j++){
    %>
      <%=names[j] + " = " + values[j]%>
      <br />
    <%}
    }
  }%>
</h5>

<br />

<h4>Note:</h4>
<p>Values will not be checked so there is more flexibility. (i.e. x = x+1)
<br />             
  If you did not enter correct information, the code will not compile
<br />If you leave a variable's value blank it will not be changed.
<br />If you click the add results button, it will overwrite previous results</p>
```

```
<p>Click the Next button to submit the rule. You will then be able to create additional rules.</p>


<br />
<h4>Conditions</h4>
<html:form action="/addCondition" method="post">

<% if(conditions.size() > 0){ %>
      <html:select property="conjunction">
        <html:option value="||">Or</html:option>
        <html:option value="&amp;&amp;">And</html:option>
      </html:select>
<%}%>

<html:select property="numCellsBased" onchange="correctInput(this)" styleId="numCellsBased" value="">
  <html:option value="false">Single cell with value</html:option>
  <html:option value="true">Number of surrounding cells with value</html:option>
</html:select>
<br />
<html:select property="varName">
  <%List vars = cb.getVars();
    for(int i = 0; i < vars.size(); i++){
      VariableBean vb = (VariableBean)vars.get(i);
      if(!vb.getVarName().equals("color")){%>
      <html:option value="<%=vb.getVarName()%>"><%=vb.getVarName()%></html:option>
    <%}
    }%>
</html:select>

<html:select property="operator">
  <html:option value="<">Less Than</html:option>
  <html:option value="<=">Less Than or Equal To</html:option>
  <html:option value="==">Equal To</html:option>
  <html:option value="!=">Not Eqaul To</html:option>
  <html:option value=">">Greater Than</html:option>
  <html:option value=">=">Greater Than or Equal To</html:option>
</html:select>

<html:text property="varValue" value="" />
<br />
<html:select property="numCellsOperator" disabled="true" style="visibility='hidden'" styleId="numCellsOperator">
  <html:option value="<">Less Than</html:option>
  <html:option value="<=">Less Than or Equal To</html:option>
  <html:option value="==">Equal To</html:option>
  <html:option value="!=">Not Eqaul To</html:option>
  <html:option value=">">Greater Than</html:option>
  <html:option value=">=">Greater Than or Equal To</html:option>
</html:select>

<html:select property="numCells" disabled="true" style="visibility='hidden'" styleId="numCells">
  <html:option value="<%=Integer.toString(0)%>">0</html:option>
  <html:option value="<%=Integer.toString(1)%>">1</html:option>
  <html:option value="<%=Integer.toString(2)%>">2</html:option>
  <html:option value="<%=Integer.toString(3)%>">3</html:option>
  <html:option value="<%=Integer.toString(4)%>">4</html:option>
  <html:option value="<%=Integer.toString(5)%>">5</html:option>
  <html:option value="<%=Integer.toString(6)%>">6</html:option>
  <html:option value="<%=Integer.toString(7)%>">7</html:option>
  <html:option value="<%=Integer.toString(8)%>">8</html:option>
</html:select>
<br />
<html:submit property="Submit" value="Add Condition" />
</html:form>
```

```
<br />
<hr />
<br />

<h4>Results</h4>


<html:form action="/addResult" method="post">
  <table>
    <thead>
      <tr>
        <th>Variable Name</th>
        <th>Value</th>
      </tr>
    </thead>
    <%List vars = cb.getVars();
   for(int i = 0; i < vars.size(); i++){
     VariableBean vb = (VariableBean)vars.get(i);  %>
    <tr>
      <td><%=vb.getVarName()%></td>
      <html:hidden property="varName" value="<%=vb.getVarName()%>" />
      <%if(vb.getVarName().equals("color")){%>
      <td>
        <html:select property="varValue" value="">
             <html:option value=""></html:option>
             <html:option value="Color.black">Black</html:option>
             <html:option value="Color.blue">Blue</html:option>
             <html:option value="Color.cyan">Cyan</html:option>
             <html:option value="Color.darkGray">Dark Gray</html:option>
             <html:option value="Color.gray">Gray</html:option>
             <html:option value="Color.green">Green</html:option>
             <html:option value="Color.lightGray">Light Gray</html:option>
             <html:option value="Color.magenta">Magenta</html:option>
             <html:option value="Color.orange">Orange</html:option>
             <html:option value="Color.pink">Pink</html:option>
             <html:option value="Color.red">Red</html:option>
             <html:option value="Color.white">White</html:option>
             <html:option value="Color.yellow">Yellow</html:option>
         </html:select>
      </td>
      <%}
      else {%>
      <td>
        <html:text property="varValue" value="" />
      </td>
      <%}%>
    </tr>
    <%}%>
  </table>
 <html:submit value="Add Results" property="Submit" />

</html:form>

<br />
<html:form action="/afterAddRule" method="post">
  <html:submit property="Submit" value="Next  >>" />
</html:form>
</div>
</body>
</html>
```

B.4 populateSystem.jsp

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ page import="edu.trinity.Jackal.structures.*, edu.trinity.Jackal.commons.*, java.util.*"%>

<html>
<head>
<title>Populate System</title>
<%String url = "http://" + Config.getProperty("IP_ADDRESS") + ":" + Config.getProperty("PORT") + Config.getProperty("ST
<style type="text/css">@import url("<%=url%>");</style>
</head>
<body>
<jsp:include flush="true" page="header.jsp" />

<div id="main">

<h2>Populate System</h2>

<% CustomBean cb = (CustomBean)request.getAttribute("customBean");
   if(cb == null){
     response.sendRedirect("/index.jsp");
   }
   request.getSession().setAttribute("customBean",cb);
   List vars = cb.getVars();
   int max = Integer.parseInt(cb.getNumCellsWide()) - 1;
   int __max = Integer.parseInt(cb.getNumCellsWide());
   %>
<h3><%=cb.getTitle()%></h3>

<p>Many cellular automata start with several cells different than the others; this section allows you to do so.<br />In

<div id="error">
<html:errors/>
</div>

<%List list = cb.getPopulatedCells();
  if(list == null){
     list = (List)request.getAttribute("populatedCells");
     if(list == null){
        list = new ArrayList();
      }
  }

 request.getSession().setAttribute("populatedCells",list);
 int numCells = (__max * __max) - list.size();
 %>
<h5>There are currently <%=list.size()%> populated cells out of <%=__max * __max%>  total cells.</h
<html:form action="/populateSystem" method="post">
   <label>Num cells to fill :
     <html:select property="numToFill" value="1">
       <%for(int i = 1; i <= numCells; i++){%>
           <html:option value="<%=Integer.toString(i)%>"><%=i%></html:option>
       <%}%>
     </html:select>
   </label>
   <br />
   <br />
   <label>X value :
   <html:select property="xval" value="random">
     <html:option value="random">Random Value</html:option>
   <%for(int i = 0; i <= max; i++){%>
        <html:option value="<%=Integer.toString(i)%>"><%=i%></html:option>
```

```
    <%}%>
   </html:select>
   </label>
      
   <label>Y value :
   <html:select property="yval" value="random">
     <html:option value="random">Random Value</html:option>
     <%for(int i = 0; i <= max; i++){%>
        <html:option value="<%=Integer.toString(i)%>"><%=i%></html:option>
    <%}%>
   </html:select>
   </label>
   <br />

   <table>
     <thead>
       <tr>
         <th>Variable Name</th>
         <th>Variable Type</th>
         <th>Variable Value</th>
       </tr>
     </thead>
     <%for(int i = 0; i < vars.size(); i++){
       VariableBean vb = (VariableBean)vars.get(i);%>
       <tr>
         <td>
            <%=vb.getVarName()%>
            <html:hidden property="varName" value="<%=vb.getVarName()%>"/>
            <html:hidden property="varType" value="<%=vb.getVarType()%>"/>
         </td>
         <td>
           <%=vb.getVarType()%>
         </td>
         <td>
           <%if(vb.getVarType().equals("Color")){%>
               <html:select property="varVal">
                  <html:option value="Color.black">Black</html:option>
                  <html:option value="Color.blue">Blue</html:option>
                  <html:option value="Color.cyan">Cyan</html:option>
                  <html:option value="Color.darkGray">Dark Gray</html:option>
                  <html:option value="Color.gray">Gray</html:option>
                  <html:option value="Color.green">Green</html:option>
                  <html:option value="Color.lightGray">Light Gray</html:option>
                  <html:option value="Color.magenta">Magenta</html:option>
                  <html:option value="Color.orange">Orange</html:option>
                  <html:option value="Color.pink">Pink</html:option>
                  <html:option value="Color.red">Red</html:option>
                  <html:option value="Color.white">White</html:option>
                  <html:option value="Color.yellow">Yellow</html:option>
              </html:select>
              <%}
              else{%>
                <html:text property="varVal" value="" />
              <%}%>
         </td>
       </tr>
       <%}%>
   </table>
   <html:submit value="Populate" property="Submit" />
</html:form>

<br />
<html:form action="/afterPopulateSystem" method="post">
  <html:submit property="Submit" value="<< Previous" />
```

```
  <html:submit property="Submit" value="Next     >>" />
</html:form>

<br />
<hr />

<%
  for(int i = 0; i < list.size(); i++){
    PopulateSystemBean psb = (PopulateSystemBean)list.get(i);
    String[] names = psb.getVarName();
    String[] vals = psb.getVarVal();
  %>
<table>
   <tr>
     <td>
        <table style="border:solid; border-width:1px; margin-bottom:20px; margin-left:15; border-collapse:collapse">
          <tr style="border:solid; border-width:1px">
            <td style="border:solid; border-width:1px"> X value  </td>
            <td style="border:solid; border-width:1px"> <%=psb.getXval()%>  </td>
          </tr>
          <tr style="border:solid; border-width:1px">
            <td style="border:solid; border-width:1px"> Y value  </td>
            <td style="border:solid; border-width:1px"> <%=psb.getYval()%>  </td>
          </tr>
          <%for(int q = 0; q < names.length; q++){%>
            <tr style="border:solid; border-width:1px">
              <td style="border:solid; border-width:1px"> <%=names[q]%>  </td>
              <td style="border:solid; border-width:1px"> <%=vals[q]%>  </td>
            </tr>
          <%}%>
        </table>
    </td>
    <td>
      <html:form action="/removePopulatedCellAction" method="post">
        <html:hidden property="index" value="<%=Integer.toString(i)%>" />
        <html:submit>Remove</html:submit>
      </html:form>
    </td>
  </tr>
</table>
 <%}%>

</div>

</body>
</html>
```