INVESTIGATIONS INTO COMPLEXITY

A SURVEY OF POPULATION DYNAMICS IN CELLULAR AUTOMATA

Dissertation

Evolutionary and Adaptive Systems Msc

Department of Informatics - University of Sussex – Brighton - United Kingdom

Abstract

In the report *Looking for Life* (James 2005) the concept of Population Dynamics was introduced as a means of finding *interesting* and *complex* behaviours within Cellular Automata (CA); this paper aims to redefine these measures so as to represent a formal methodology for all CA analysis. Based on these dynamics, a number of general observations shall be highlighted, with particular attention being given to the contrasts between ordered, chaotic and complex behaviours. These contrasts shall then be further examined by a new method of probabilistic analysis wherein a more solid mathematical notion of ordered and chaotic behaviour shall be proffered and a clearer understanding of complex behaviour is proposed. Finally, it is hoped that these clearer definitions of complexity within cellular automata may assist in grounding the more general term used throughout the interdisciplinary Sciences of Complexity.

1 ACKNOWLEDGEMENTS 3

2 INTRODUCTION 4

2.1 Cellular Automata and the Science of Complexity 5
2.2 Previous work in Looking for Life 7

3 DEFINING POPULATION DYNAMICS 8

3.1 Life and Death - A Natural Bisection 8
3.2 Birth and Death – Adding A Measure of Change 9
3.3 Population Dynamic Demographics 10
3.4 Introducing the CATS toolset 10

4 OBSERVING POPULATION DYNAMICS 11

4.1 Introducing CA Examples 11
4.2 Population Dynamic Graphs for CA Examples 13
4.3 The Attraction of Birth and Death 14
4.4 Settling Down to Behavioural Stability 14
4.5 Contrasts Between Order and Chaos 15
4.6 Complexity and the Lack of Behavioural Stability 16
4.7 Population Dynamics and Entropy Variance 17

5 FOURIER ANALYSIS OF POPULATION DYNAMICS 18

5.1 Frequency Measures and Timewindows 18
5.2 Power/Frequency graphs for CA examples 19
5.3 Classifying Behaviour through Power Distribution 20
5.4 The Ubiquity of 1/f^B noise 20
5.5 Understanding 1/f^B noise 22
5.6 Complexity and Gradients 24

6 PROBABILISTIC ANALYSIS OF POPULATION DYNAMICS 25

6.1 The Probability of Life 25
6.2 The Probability of Babies and Corpses 27
6.3 How to Plot The Dynamics 28
6.4 Probable Dynamics Graphs for CA Examples 28
6.5 Population Attractors of Order and Chaos 30
6.6 Settling Down 31
6.7 Birth and Death 32
6.8 Complexity within Probability Dynamics 33
6.9 Classification Through Probability Dynamics 35

7 CONCLUSIONS 36

7.1 Refined Definitions of Complexity 36

8 PROPOSITIONS 38

8.1 A Measure of Improbability 38
8.2 Applications for the mathematical methods 38
8.3 Life Far From Equilibrium 39

9 REFERENCES 40

APPENDIX A: CA RULETABLES 42

APPENDIX B: CATS SCREEN SHOTS 44

APPENDIX C: SOURCE CODE 46

1 ACKNOWLEDGEMENTS

My sincere thanks go to Dr. Inman Harvey for his supervision, support and welcomed advice during the writing of this paper. I would also like to thank Dr. Andy Wuensche for his valuable encouragement, expertise and input into many of my findings and to Dr. Ron Chrisley for his probability knowledge at my most crucial hour. Finally thanks must go to my illustrious colleague James Carter (Douglas), and to my dearest friend and fiancée Sheryl.

2 INTRODUCTION

From the molecules to the stars, the natural world is awash with complex systems. They are our cells, our bodies, our societies and our minds. They are the whole, which is so often so much greater than the sum of its parts.

For thousands of years, man has grappled to understand how it is that we are both part and whole; how it is that our cells, together and alone, create our bodies and our souls.

For the most part, science has been dismantling the world with a reductionistic axe; redefining the whole to be *nothing more than* the atomic interplay of its constituent parts. But this paradigm is, fortunately, beginning to shift (Capra 1996, Davies 2003). New systems sciences (Van Bertalanffy 1969) are finally starting to study global properties and behaviours (or *wholes*) as classifiable phenomena and are trying and work out which of these behaviours arise under which systemic conditions (Lorenz 1963, Prigogine 1967, Mandelbrot 1977, Feigenbaum 1978).

Complexity Science, in particular, is actively studying the forces that give rise to adaptive, computational or just plain interesting behaviour found within complex dynamical networks. Its aim is to bridge the gap between the macroscopic behaviours of the whole and the microscopic dynamics of the parts (Kauffman 1993, Langton 1989, Wuensche 1996); a question that is pertinent to *all* complex systems whether biological, psychological, economical or social.

This paper aims to join this endeavour, and shall do so from the perspective of one of the core modelling techniques used within the science of complexity; the wonderful world of cellular automata.

In an earlier report, *Looking for Life* (James 2005), the concept of *Population Dynamics* was introduced as a means of finding complex behaviours in cellular automata. In chapter 3 of this paper, we aim to redefine these measures so as to represent a formal methodology for wider CA analysis.

From an analysis of these dynamics, Chapter 4 shall introduce the notion of *settling down to behavioural stability*, and use this to compare and contrast the three standard classes of global behaviour within CA (order, chaotic and complex). Chapter 5 shall then extend these observations through frequency analysis and proffer an explaination for the ubiquitious $1/f^b$ noise in order to bring together the notions of *settling down* and *complexity*.

A new method of probabilistic analysis shall be introduced in chapter 6, to further study and more clearly understand some of the observations of behavioural stability. A more solid mathematical definition of ordered and chaotic behaviour shall be offered and a refined definition of complexity within cellular automata shall be proposed.

Finally, we introduce a possible future measure of complexity within cellular automata and conclude with an examination of complexity's position within the wider, intedisciplanary sciences of systems.

First, however, it is useful to follow in some of the footsteps made by the early pioneers of this research program, and to understand some of the history between cellular automata and the science of complexity.

2.1 CELLULAR AUTOMATA AND THE SCIENCE OF COMPLEXITY

In the 1940's John von Neumann grappled with the problem of self-replication (Von Neumann 1966). Like so many of the early cybernaticians and game theorists, he wanted to use his understanding of the new mathematics of non-linear systems to brake down some of the vitalistic barriers buried within the study of biological systems. He aimed to devise a mathematical model for an imaginary factory that was capable of building other factories just like itself. In looking for a framework for his idea, he came across the work of his colleague, Stanislaw Ulam, whose crystal growth analysis was modelled within an abstract mathematical world called a lattice network. Using such a network, Von Neumann devised a 29-state automaton which, when placed in each cell of the lattice, was able to perform the kind of self replication he had envisaged (Wolfram 2002).

This marvellous model was the first example of a Cellular Automata, but it didn't attract any real attention until 25 years later, when John Conway discovered a rather majestic set of automaton rules known as the *Game of Life*. Conway, inspired by his recent mathematical success with symmetry groups, took on the daunting task of simplifying Von Neumann's self-replication model. *"After the rejection of many patterns… and of many other laws of birth and death, including the introduction of two and even three sexes"* he finally succeeded and found what he was looking for, *"a viable balance between life and death"* (Guy & Conway 1985).

The Game of Life gained immediate acclaim due to the remarkable array of evocative patterns it produced. A menagerie of lifelike gliders and guns could be seen interacting with each other inside a virtual mathematical world. Many began to question how, and why these patterns came to be, and why they didn't exist in any of the other rulesets.

In 1984, Stephen Wolfram published a report that looked at answering some of these questions (Wolfram 1984). By that time, cellular automata (CA) had become a well defined and widely studied branch of mathematics. A CA was defined to be a ***D***-dimensional lattice with a finite state automaton placed at each site in the lattice. Each automaton could have ***Q*** distinct states and at any given time (***t***) an automaton would be said to be in the specific state. Each automaton would calculate its next state from a lookup table of rules based on the configuration of its local neighbourhood. The automaton rule table would simply consist of a rule for each possible configuration within the *N* neighbourhood template. Numerically speaking, therefore, it was found that a particular class of CA would have $|Q|^{(|Q|^{|N|})}$ possible rulesets (Ganguly 2003).

Given that the Game of Life has 2 states and a 9 cell neighbourhood (known as the Binary Moore configuration) it was soon realised that this ruleset was one of 10^{154} possibilities. So Wolfram narrowed down the playing field by examining the smaller world of 1-dimensional CA. From the careful analysis of this CA landscape, he declared four classifications to categorise the various global behaviours.

Class I:	Evolution leads to a homogeneous state
Class II:	Evolution leads to a set of separated simple stable or periodic structures
Class III:	Evolution leads to a chaotic pattern
Class IV:	Evolution leads to complex localized structures, sometimes long-lived

By classifying the *behavioural landscape* in this way, the question immediately became "how do we determine which rulesets will produce which of these behaviours?"

The fledgling field of Complexity Science, nestling inside the Santa Fe Institute, was first to propose an answer. Chris Langton revised Wolfram's classes to fit with the growing language of dynamical systems. He regrouped class I and class II as a single class of *Ordered* behaviors (on the basis that a homogeneous state is a cyclical state of period one) and he simply renamed Class III behaviors to be *Chaotic.* The last, interesting set of behaviors (Class IV), he called *Complex (Langton 1990)*.

These three classes of behavior were suggested to be analogous to the three states of matter. Ordered behavior was analogous to the solidity of ice; chaotic behavior acted like the gaseous randomness of steam; whilst Complexity lay somewhere in between the two, a fluid place of liquidity and motion (Waldrop 1992, Coveney & Highfield 1995, Lewin 1993). This place became known as the Edge of Chaos (Packard 1988).

Chris Langton devised a parameter, the lambda (λ) parameter, which he hoped would be able to separate the three classes of behaviour (Langton 1990). He found that most rulesets with a λ above 0.2734 displayed chaotic behavior whilst most rulesets with a λ below 0.2734 displayed ordered behavior. The Game of Life, however, lay precisely upon the the edge of the two, with a λ of precisely 0.2734.

Complexity Theorists began to propose that dynamical systems tuned to this *Edge of Chaos* would possess a property known as Emergence (the process by which new forms of order could be generated within levels higher than the level of the constituent parts of a system) (Kauffman 1993, Holland 1998, Bickhard & Campbell 2000). Through this emergence complex patterns would persist in nature and, through the forces of self-organisation and natural selection, would be pruned for their adaptivity (Cariani 1990, Morowitz 2002).

However, complexity was not going to be so readily simplified. James Crutchfield and Melanie Mitchell have subsequently shown that the λ parameter is inadequate as a classification scheme (Mitchell, Hraber and Crutchfield 1993). It has been shown that evolution does NOT favour this mystical arena and that there are, in fact, a vast array of ordered CAs with a λ greater that 0.2734 just as there are a vast array of chaotic CAs with a λ of less that it. The edge of chaos, although a wonderfully evocative metaphor, is unfortunately just that.

Regretably, this has left Complexity Science a little bereft of its foundations and for the principles of emergence to continue a more solid concept of complexity is required. Although the reclassification of cellular automata into *ordered*, *chaotic* and *complex* seems justifiable, the landmarks and explanations for this classification are missing. The original questions posed by the existence of the Game of Life remain unanswered and a suitable understanding of emergent phenomena remains a mystery.

A dedicated few, however, have kept hold of the tail of this mystery. Andy Wuensche has proposed Entropy Variance as a suitable classification scheme, distinguishing chaotic and ordered systems by means of their respective high and low entropies (Wuensche 1996). So too, Shigera Ninagawa has expressed $1/f^b$ noise as a unique tell-tale signature of complexity in CAs and has gone on to find other rulesets showing complex behaviour using this signature (Ninagawa 1998, Ninagawa 2005). More recently however, within a short report entitled *Looking for Life* (James 2005), an alternative route to the problem has been opened up; and it is this route that shall be followed within this report.

2.2 PREVIOUS WORK IN LOOKING FOR LIFE

Looking for Life (James 2005) tentatively introduced a new series of population measures which were able to distinguish between chaotic and ordered systems and, importantly, could highlight the complex behaviour of the Game of Life.

Although poorly understood at the time, these observations were used to create a basic Genetic Algorithm which was able to adequately search the Binary Moore landscape looking for other rulesets that fit this profile of complexity. The results were very promising.

A series of rulesets were discovered which displayed complex emergent phenomena similar to, and in some cases beyond, those found in the Game of Life. Self-replicators and gliders were found in abundance, and in many shapes and sizes, throughout the massive rule-space.

The remainder of this paper aims to expand upon the work started within this report, with the hope of providing a greater understanding of complexity within CAs; but before proceeding, it is necessary to adapt and redefine the measuring techniques outlined to introduce a more formal set of measures called, the *Population Dynamics of CA*.

3 Defining Population Dynamics

The following chapter aims to redefine the population measures outlined in *Looking for Life* in order to establish a more formal framework for CA measurement. This framework has been called the *Population Dynamics of CA*.

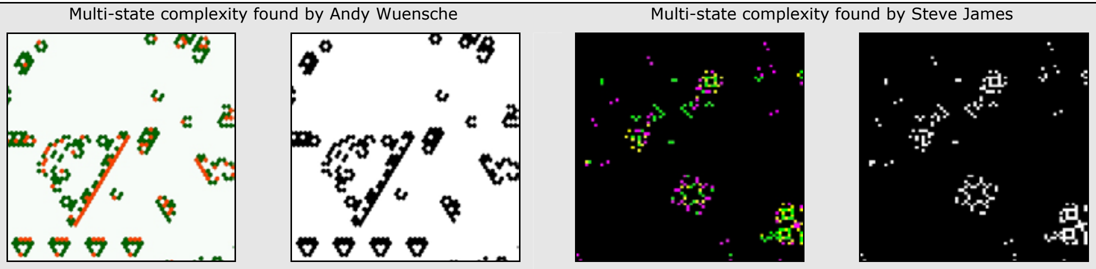
3.1 Life and Death - A Natural Bisection

First, it is important to make explicit a subtle shift in perspective in these new measures. Rather than looking at the standard world of "*cell states*", we shall instead view CA behaviours from the binary domain of "*cell existence*".

Chris Langton made a similar shift during his formal definitions of the Lambda parameter wherest he defines the bisection between "quiescent" and "non-quiescent" cell states. A "quiescent" state is described as a kind of background state *upon which* the "non-quiescent" states perform their complex behaviour. In the Game of Life, for example, one colour (state zero) acts as a background upon which a second colour (state one) forms gliders, guns and blinkers.

Indeed, it appears that this natural bisection is quite common, although often non-explicit, within CA research, especially when considering multi-state CAs.

As an illustration, figure 3.1 shows two example multi-state CAs. Both contain a distinct background colour *upon which* all complex behaviour can be observed. In fact, it can also be seen that the bodies of gliders, guns and other phenomena can contain many different colours (states) but that the identity of these emergent agents is observed purely by their relation to the background state (observed by examining the *facsimile* image in figure 3.1).


Multi-state complexity found by Andy Wuensche
Multi-state complexity found by Steve James

(Figure 3.1) The natural bisection found in 3-state and 4-state CAs

It is this observation which takes us away from *cell state* space and into the subtly different domain of *cell existence*. More formally, we declare that each cell in any CA (of any number of states) has a binary classification of either "*alive*" (non-quiescent) or "*dead*" (quiescent).

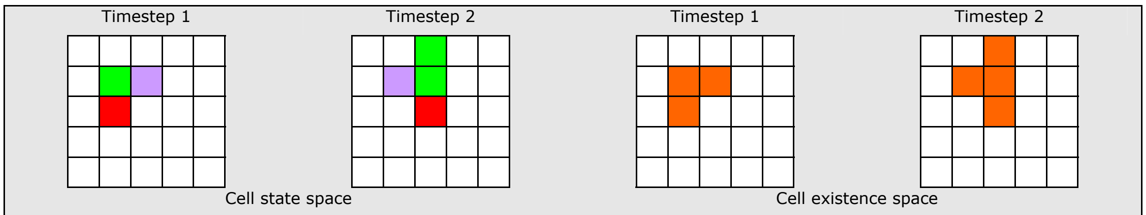
Total cells (X) - the total number of cells within the CA lattice
Alive cells (A_t) - the number of "*alive*" (non quiescent) cells at time (t)
Dead cells (D_t) - the number of "*dead*" (quiescent) cells at time (t)

$$X = A_t + D_t \tag{3.1}$$

From this binary perspective we can begin to include the further notions of change and time.

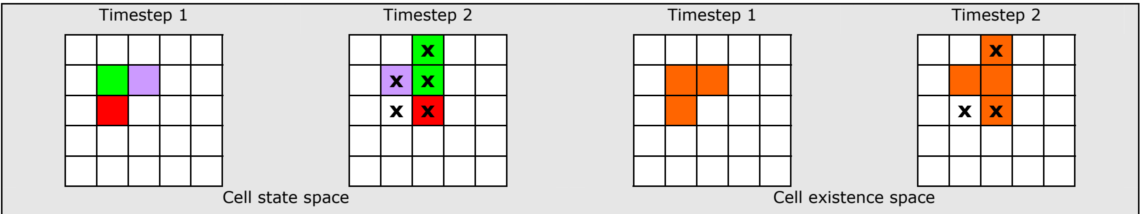
3.2 Birth and Death – Adding a Measure of Change

The data available for measurement in a CA has traditionally been considered in discrete snapshots. For example, the simple CA in figure 3.2 (at timestep 2) can be measured from the "*cell state"* perspective as containing 21 *white cells*, 2 *green cells*, 1 *blue cell* and 1 *red cell* (so providing 4 measurements of data). Similarly, if considered from the "*cell existence*" perspective, the CA can be said to contain 21 *dead cells* and 4 *alive cells* (so providing 2 measurements).



(**Figure 3.2**) Snapshots of information within a CA

However, there is another measurement hidden *between* the consecutive timesteps that has, up until now, gone unnoticed; the notion of change. (Figure 3.3 highlights this information by placing a cross inside the cells which have changed between the two timesteps)



(**Figure 3.3**) Adding the notion of change to informational snapshots within a CA

From the domain of "*cell existence*", these crosses are extremely interesting as they introduce the concepts of "*becoming alive*" and "*becoming dead*" to our arsenal of measurements. For example, in figure 3.3, we can now say that 2 cells have "*been born*" and 1 cell has "*died*".

However, in the domain of cell state space, these crosses are far less meaningful. Consider the similar concept of "*becomes green*"; because this may refer to a white, blue or red cell becoming green the concept is very broad and of little general use. Potentially, you may want to create a series of distinct concepts such as "*white becomes green*" but this would require Q^2-Q such concepts for a Q-state CA and these would only be of use for that particular class of CA.

One of the key advantages of the domain of "*cell existence*" is that these notions of change are widely comparable across all classes of CA and may even prove a useful measure for comparison amongst the wider community of dynamic systems in general.

To conclude, we can more formally define the notions of change as:-

Babies (B_t) - the number "alive" cells at time (t) which were "dead" at time (t-1)
Corpses (C_t) - the number "dead" cells at time (t) which were "alive" at time (t-1)
Stay Alives (SA_t) - the number "alive" cells at time (t) which were also "alive" at time (t-1)
Stay Deads (SD_t) - the number "dead" cells at time (t) which were also "dead" at time (t-1)

3.3 Population Dynamic Demographics

As shall be seen later in this paper, normalising these measures to make them proportional with respect to the overall size of the CA is essential for the sensible cross referencing of statistical data. As such we outline the four main statistical measures that shall be used below:-

Population Rate

$\mathbf{a_t = A_t / X}$	proportion of alive cells at time (t)
$\mathbf{d_t = D_t / X}$	proportion of dead cells at time (t)

Birth Rate

$\mathbf{b_t = B_t / X}$	proportion of cells which were "dead" at time (t) but "alive" at time (t-1)

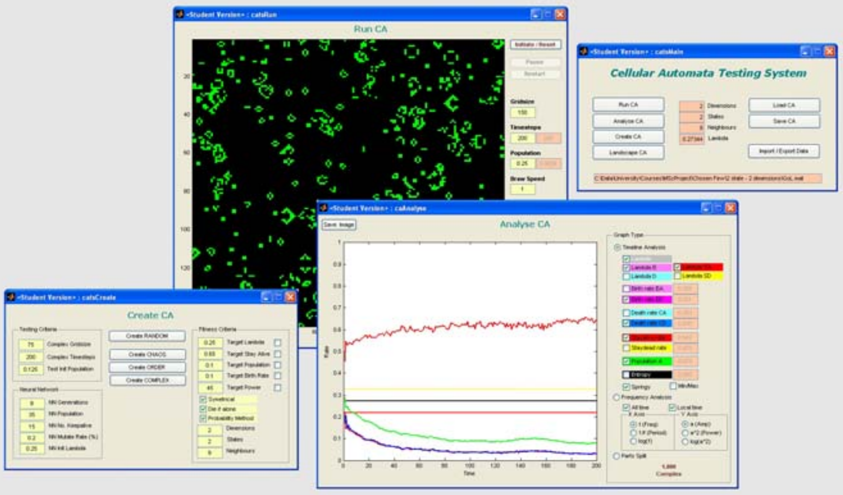
Death Rate

$\mathbf{c_t = C_t / X}$	proportion of cells which were "alive" at time (t) but "dead" at time (t-1)

3.4 Introducing the CATS toolset

As already stated, the primary concern of this paper lies with the observations and conclusions that are drawn from the measurement and analysis of Population Dynamics within CA. In order to perform this study a series of tools was created which would allow for the creation and running of a number of CAs and for the measurement and analysis of their respective dynamics from the perspective of time, frequency and probability. These tools have become collectively known as the *Cellular Automata Testing System* (CATS).

In is not the intention of this paper to go into any detail regarding the specific design decisions and coding techniques of the CATS systems; however a full list of the MATLab functions implemented is provided within Appendix C. Screenshots and results from the software will be used throughout the paper, but figure 3.4 below shows the main screens created and used (Appendix B also contains further details for each screen).



(**Figure 3.4**) An example of some of the **CATS** screens

4 OBSERVING POPULATION DYNAMICS

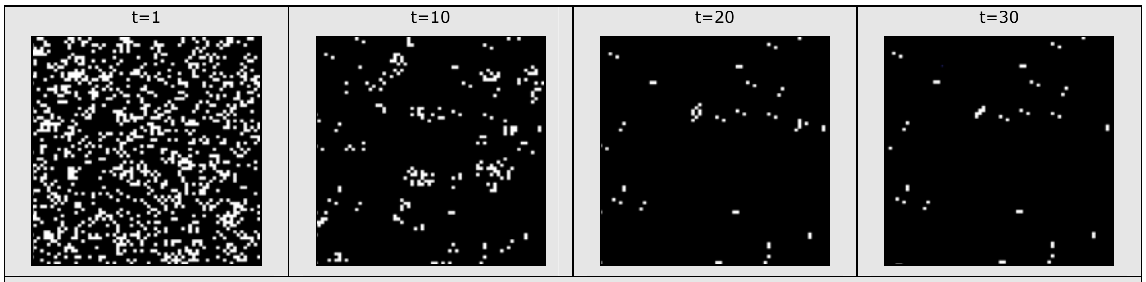
Having made a clear set of definitions for Population Dynamics in cellular automata, this chapter aims to outline how these dynamics are played out within an example set of cellular automata.

4.1 INTRODUCING CA EXAMPLES

We begin with a quick introduction of these example CA rulesets; the true nature of which can only really be appreciated by watching the dynamic evolution of their behaviour. In reducing this behaviour to two dimensional "snap-shots" most of the important features are lost and so a brief paragraph describing how each one evolves should help to enhance the readers understanding of some of the later observations and discussions.

All of our examples shall be 2-dimensional Moore neighbourhood CAs and the "snap shots" provided show dead cells as BLACK and alive cells as WHITE. Appendix A contains the full ruletable for each example ruleset.

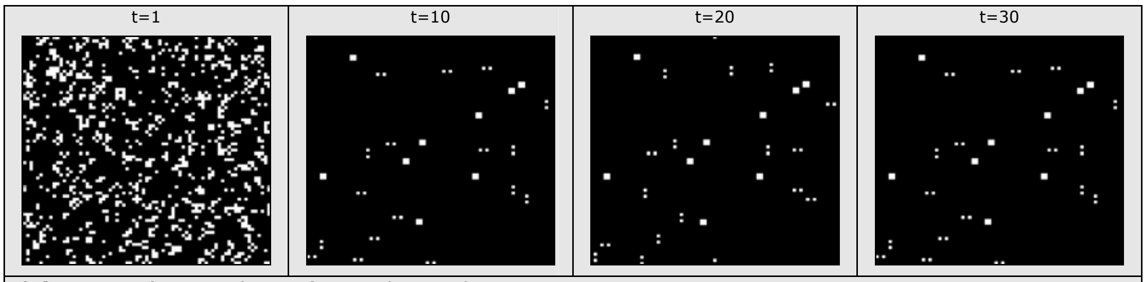
RULE 1: ORDERED: WITH A SLOW DECAY



(**Figure 4.1**) Snap shots of CA Rule 1, taken every 10 timesteps

The randomly distributed initial population slowly dissolves, into smaller and smaller pockets of activity. After about 20 timesteps all such activity has completely stopped leaving only a handful of simple blinkers and fixed blocks.

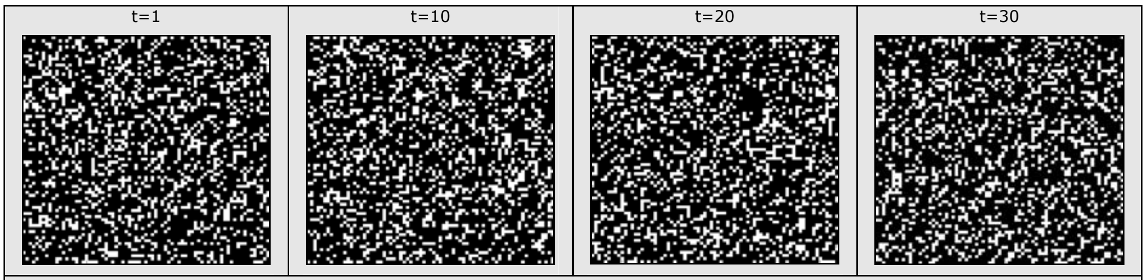
RULE 2: ORDERED: WITH A QUICK DECAY



(**Figure 4.2**) Snap shots of CA Rule 1, taken every 10 timesteps

The initial population quickly dissolves (within 5 timesteps) resulting in a handful of simple blinkers and fixed blocks.

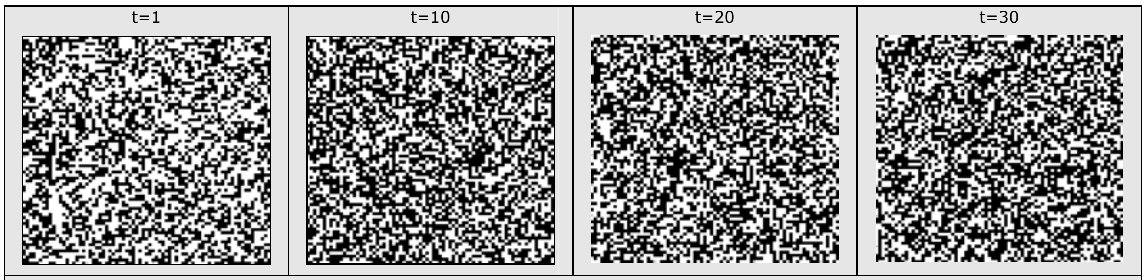
<u>Rule 3: CHAOTIC: with a low Λ</u>



(**Figure 4.3**) Snap shots of CA Rule 1, taken every 10 timesteps

A roughly constant density (approximately equal to λ) of rapidly changing noise is displayed.

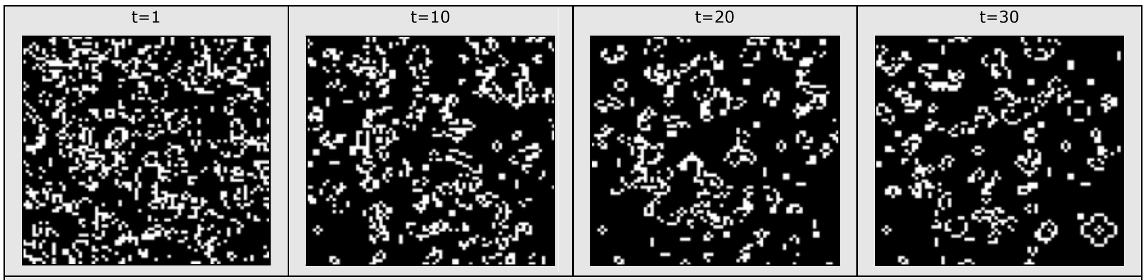
<u>Rule 4: CHAOTIC: with a high Λ</u>



(**Figure 4.4**) Snap shots of CA Rule 1, taken every 10 timesteps

As rule 3, but with a slightly higher density of *alive* cells maintained.

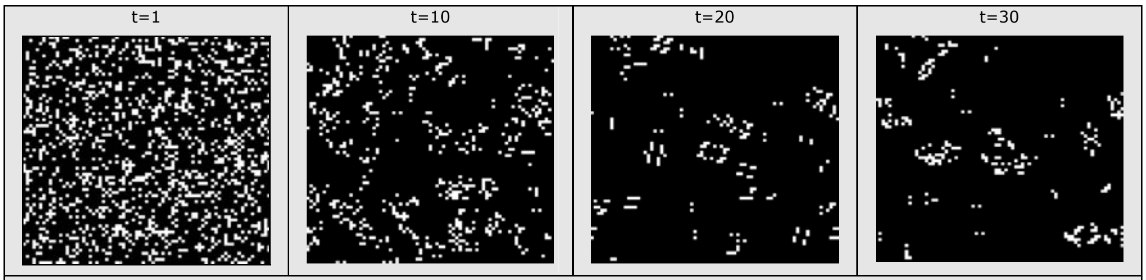
<u>Rule 5: COMPLEX: Game of Life</u>



(**Figure 4.5**) Snap shots of CA Rule 1, taken every 10 timesteps

John Conway's famous Game of Life. A world in which gliders are surrounded by a mixture of ordered blinkers and stable blocks as well as bubbling pockets of noisy activity.

<u>Rule 6: COMPLEX: Replicating Glider Guns</u>

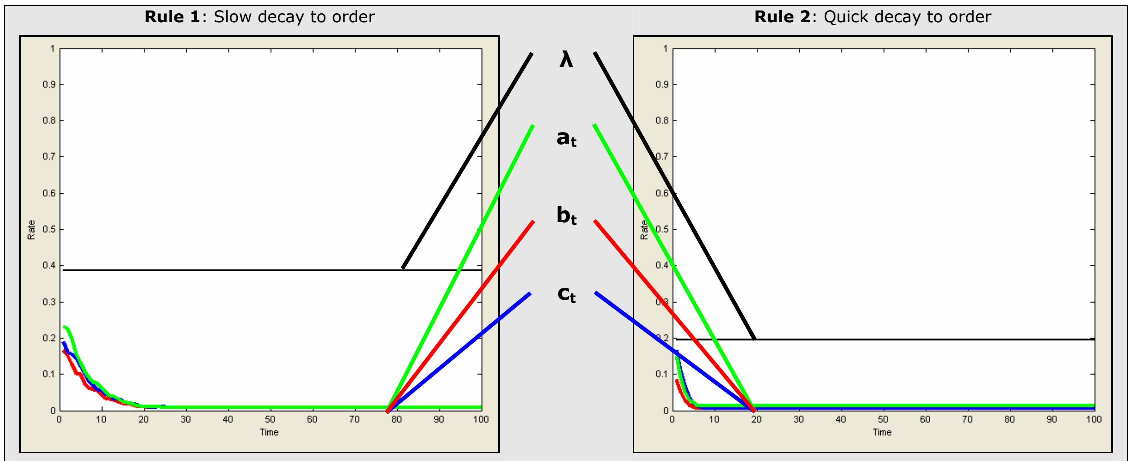


(**Figure 4.6**) Snap shots of CA Rule 1, taken every 10 timesteps

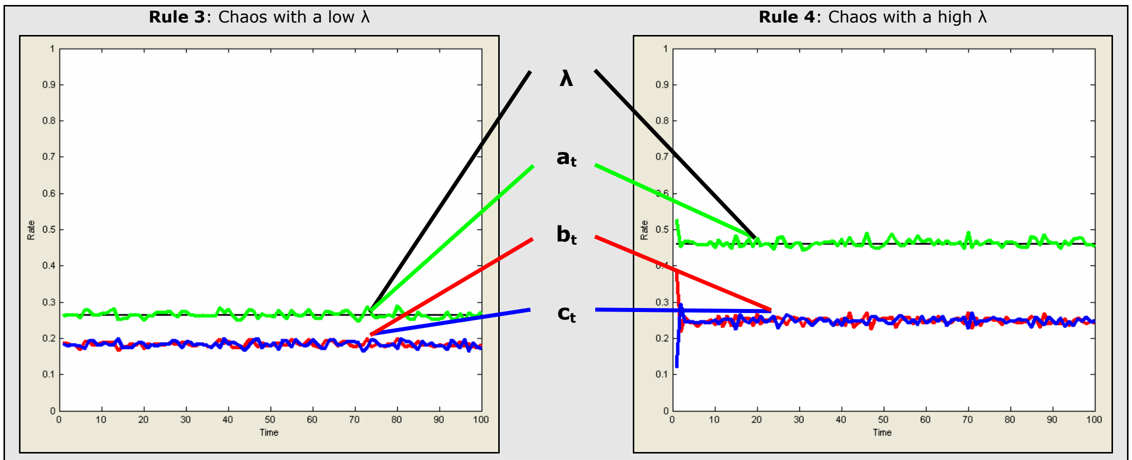
Discovered within "*Looking for Life*" this ruleset quickly dissolves to a near barren landscape in which growing patterns bubble and collide occasionally spewing forth gliders in all directions.

4.2 Population Dynamic Graphs for CA Examples

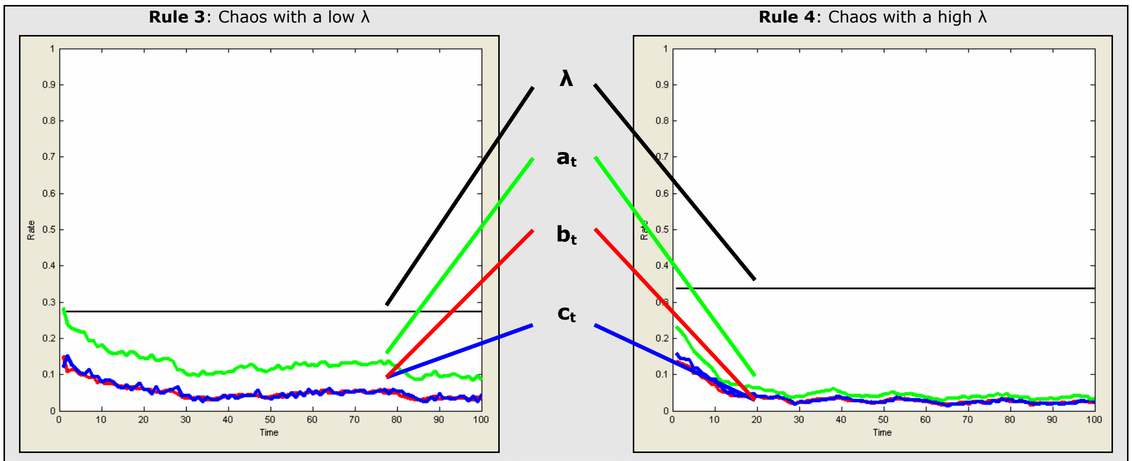
The following graphs show how the measures of Population Dynamics change over time for each of our example CAs. All tests were performed within a lattice of 50x50 cells, and were run over 100 timesteps. The CAs were all started with a random initial density (a_0) of 0.25.



(**Figure 4.7**) Rate of change graphs for cellular automata displaying **ORDERED** behaviour



(**Figure 4.8**) Rate of change graphs for cellular automata displaying **CHAOTIC** behaviour



(**Figure 4.9**) Rate of change graphs for cellular automata displaying **COMPLEX** behaviour

4.3 THE ATTRACTION OF BIRTH AND DEATH

Our first quick, and simple, observation relates to all of the examples provided (and in fact has been observed within all CA ever measured using these methods). In looking at figures 4.7, 4.8 and 4.9 it is immediately apparent that the birth and death rates follow each other extremely closely throughout the lifecycle of the cellular automata. In other words, the number of births and the number of deaths within any cellular automata at any given moment appears to be roughly equal.

$$B_t \approx C_t \tag{4.1}$$

This fact is examined in a little more depth later in this paper, but a further analysis (beyond our scope) may turn out to be of great benefit.

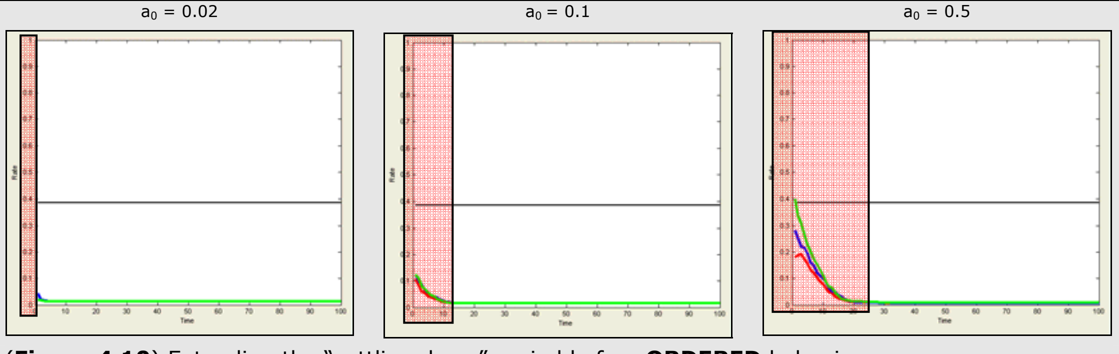
4.4 SETTLING DOWN TO BEHAVIOURAL STABILITY

In "*A New Kind of Science*" (Wolfram 2002), Stephen Wolfram describes a class of CA which "settles down" to either a stable or periodic pattern of behaviour (later known to be the class of ordered CA). This process of settling down, he claims, can be described as a kind of self-organisation for complex systems. In contrast to this, he describes an alternative class of CAs which never "settles down" in this way, but which instead keeps evolving in a random (chaotic) or sometimes computationally interesting (complex) way.

This view is the generally held view within most CA research, and clearly supposes that this "settling down" period is unique to ordered systems; with chaotic and complex systems being an opposing kind of behaviour that "never settles down". This view is challenged below.

Rule 1 of our examples, is described as having a "slow decay" (or settling down period) of around 20 timesteps. During this period of decay, one can describe the evolution of the CA as "looking like water draining from a bath riddled with holes". A world of chaos quickly becomes puddles of chaos which eventually drain away to nothingness. Once the bath has emptied, once the settling down period has ended, the CA is left with nothing but a handful of fixed points and blinkers.

These two distinct periods of activity (*settling down* and *order*) are clearly visible in the population dynamic graphs given in figure 4.7; an initial period where the population measures gradually decay towards zero (corresponding with the settling down period) followed by a period of perfect linear stability (corresponding with the ongoing period of ordered behaviour). This is further illustrated, and extended, in figure 4.10 which shows how the length of this settling down period can be changed by altering the initial density of the system.



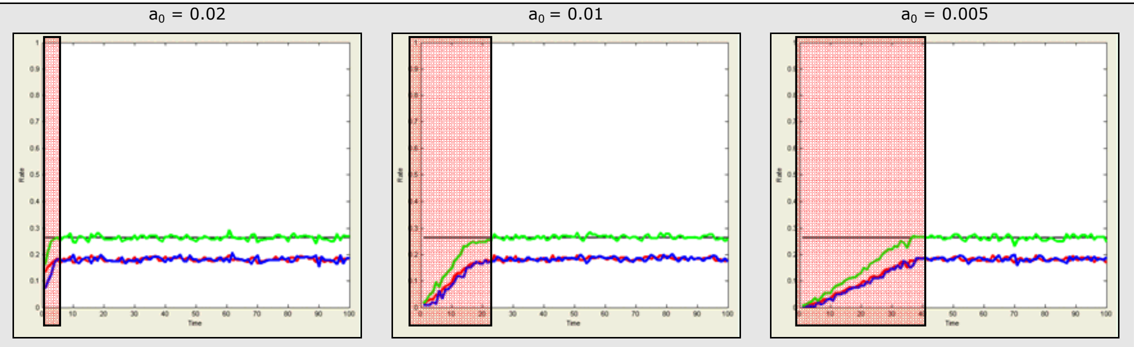
(**Figure 4.10**) Extending the "settling down" period before **ORDERED** behaviour

None of this is yet out of tune with Wolfram's observations or comments; ordered systems show a distinctive settling down period; but what about chaotic ones?

An initial glance at the population dynamic graphs of our chaotic examples (figure 4.8) would again seem to concur with Wolfram. There is no indicative slope in either of these graphs; in fact, the population measures simply seem to remain in a roughly straight, but rather noisy, line.

Interestingly the initial density for both graphs was around 0.25. The first system maintains this initial density throughout, but for some reason (discussed later) the second system seems to "bump this up a bit" to maintain a straight line of noise at around 0.45. Besides that oddity, however, neither of them seems to show any signs of settling down; totally in tune with Wolfram.

However, if, as before, the initial density is altered a little bit, we see a slightly different picture. Figure 4.11 below illustrates this a little more clearly.



(**Figure 4.11**) Extending the "settling down" period before **CHAOTIC** behaviour

At very low initial densities, the tell-tale signs of a "settling down" period seem to reappear; but this time in reverse. The population rate seems to slowly climb upwards to a more stable level.

When watching such a CA evolve, one can best describe it as looking like the exact opposite to the "settling down" period of an ordered system. It can be likened to watching the imaginary bath draining in reverse; with puddles of chaotic behaviour slowly growing until the CA is completely full.

So it would seem that we can again split the evolution of these CA into two periods of behaviour; a kind of "settling up" period followed by a maintained period of "chaotic" behaviour. For the majority of the time, this chaotic behaviour is reached within a matter of 1 or 2 timesteps (seen as a quick "bumping up" seen earlier in Rule 4) and so it has traditionally been overlooked.

In conclusion, rather than describing ordered systems as "settling down" and chaotic systems as "never settling down" (as Wolfram does), it seems that we would be better served by describing both systems as settling down to a level of behavioural stability; an important shift of perspective which shall be examined in much more depth throughout the remainder this paper.

4.5 Contrasts Between Order and Chaos

Looking more closely at these levels of behavioural or population stability, one can immediately see a stark contrast between the two classes of order and chaos. Based on these contrasts (and through the extended study of a wider collection of rulesets not outlined within this paper) we can state the following observational statements for all ordered and chaotic behaviour found within cellular automata.

ORDERED BEHAVIOUR

1. After the initial "*settling down*" period, all rates reach a distinctive mean (a point of behavioural stability) wherefrom they either remain static or show a flat periodic rate of change.
2. The mean of the population rate is always of a very low value, usually extremely close to zero.
3. The birth and death rates share the same mean which also tends to be extremely close to zero.

CHAOTIC BEHAVIOUR

1. After a (usually short) "*settling down*" period, all rates reach a distinctive mean (point of behavioural stability) wherefrom they show a roughly flat, but extremely noisy (non-periodic) rate of change.
2. The mean of the population rate is always a high value which tends to be reasonably close to λ.
3. The birth and death rates share the same mean which is also of a reasonably high value.

In short, therefore, it is observed that both types of system exhibit a distinctive point of behavioural stability and that a distinct difference can be found in the value (height) of this point, and in the type of wave-pattern displayed once it has been reached.

Ordered systems tend to have a low and flat (or periodic) point of behavioural stability whereas chaotic systems tend to have a high and noisy point of behavioural stability.

4.6 COMPLEXITY AND THE LACK OF BEHAVIOURAL STABILITY

Given such a clear and distinctive set of population dynamics found within ordered and chaotic systems, we can now go on to examine our examples of complex behaviour.

From figure 4.9 it seems immediately evident that defining a *distinctive mean* or *point of stability* becomes rather difficult. Although it may be said that the population rates are "generally low" it is much more difficult to pinpoint an accurate average value as they seem to show a reasonably high degree of long term variance (a behaviour that one might wish to call a random walk).

Unlike ordered and chaotic systems where a formal set of statements regarding the behaviour of population rates can be made, it seems that with complex behaviour we can only talk in general terms.

1. A distinctive settling down period is hard to observe, but there certainly appears to be a period of time during which a large part of the system decays to leave pockets of complex behaviour. At this point the random walk fluctuations of the population rates become less dramatic and they can be said to have reached a kind of *very rough* (but difficult to measure) mean.
2. The population rate's rough mean always lies at some low value between λ/2 and zero.
3. The birth and death rates share the same low rough mean somewhere below a_t.

Clearly, these statements are far less defined that those made for order and chaos; but importantly it seems that the systems contain certain elements of both. It is almost as though complex behaviour is a kind of difficult to pinpoint combination of ordered and chaotic characteristics.

The generally low population rate is a distinct characteristic of order, whilst the noisy fluctuations are more akin to chaos.

4.7 Population Dynamics and Entropy Variance

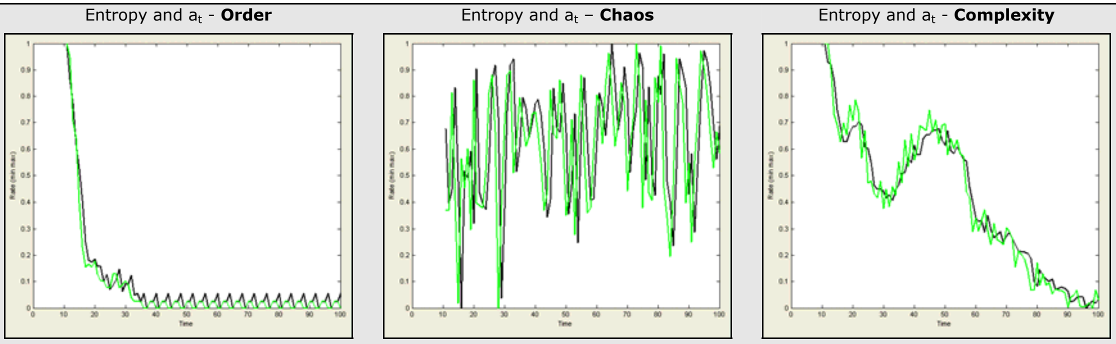
In 1996 Andy Wuensche highlighted a strikingly similar set of observations regarding the dynamics of ordered, chaotic and complex behaviours, but from the entirely separate standpoint of Entropy Variance. By recording the frequency distribution of rule lookups (or state configurations), he was able to measure the effective entropy of the system using Shannon's entropy equation (below).

$$S^{(t)} = -\sum_{i=1}^{2^k}\left(\left(Q_i^{(t)}\big/ n\right)\times\log\left(Q_i^{(t)}\big/ n\right)\right) \qquad (4.2)$$

He found that chaotic behaviour showed a distinctively high level of entropy whilst ordered behaviour showed a distinctively low level of entropy. For complex behaviour the entropy was seen to be much less *stable* and would fluctuate greatly between high and low values in a kind of random walk.

This obviously sounds rather reminiscent of the observations made previously for population dynamics; and in fact it seems that this relationship is stronger than mere descriptive analogy.

By plotting the min/max of the population rate alongside the min/max of the entropy variance, one immediately becomes aware that the two measurements follow a similar path (see figure 4.12 below – the entropy is shown in BLACK and the population rate in GREEN).



(**Figure 4.12**) Min/Max graphs showing the relationship between **entropy** and $\mathbf{a_t}$

As the population rate grows, the entropy grows proportionately with it. As the population rate decreases, so too does the entropy. The dynamics of the two measures are never quite the same but they are most remarkably similar. If they were the same, one might reasonably assume that they were measuring the same basic property of the system; but as they're not, one might summise that their closeness is due to them both being driven by the same underlying cause.

On discussing this observation with Andy himself, he proposed a possible (and seemingly likely) explanation for what this underlying cause might be.

As the population rate increases more cells becomes available to make a broader set of state configurations. The higher the population rate, the greater this distribution of configurations will become. Or to put it another way, *it is the change in population rate which drives the increase and decrease in entropy*.

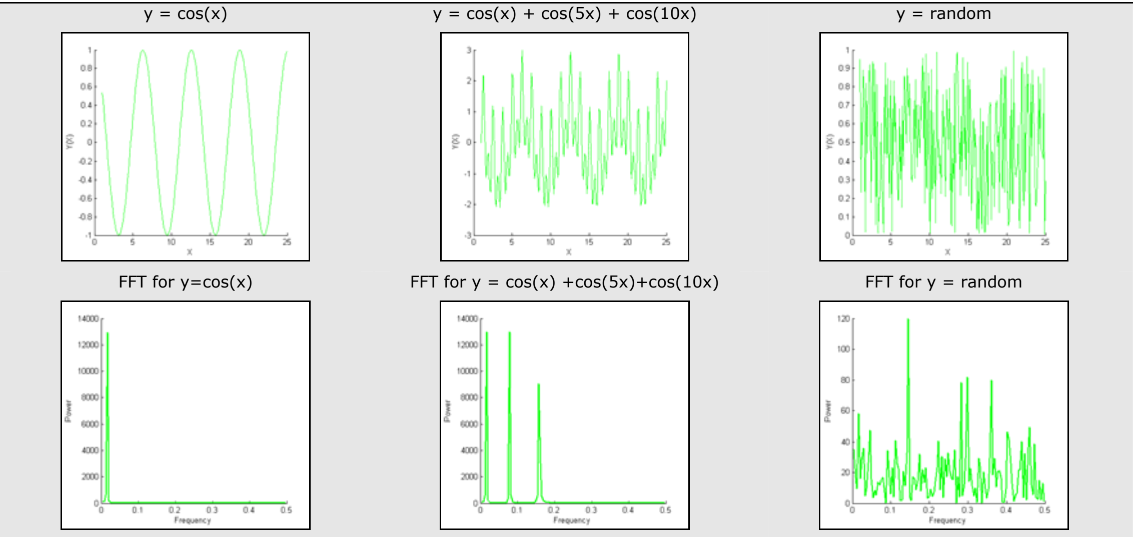
This is discussed a little further in the concluding chapter of this paper, but may also benefit from further study with an eye to crossing the academic bridge between CA and thermodynamic principles.

5 Fourier Analysis of Population Dynamics

Given our observations and measurements of Population Dynamics within the time domain, we now move on to examine this data from the alternative perspective of the frequency domain.

5.1 Frequency Measures and Timewindows

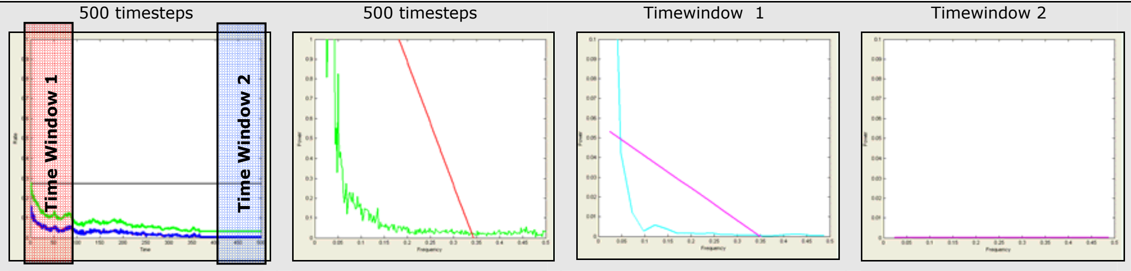
Methods for measuring frequency are well known and well documented. By use of a Fourier transformation it is possible to turn a time-line analysis into a simple Power/ Frequency graph showing the power of the underlying Sin and Cos fluctuations which make up the original waveform. Figure 5.1 shows an example of a FFT (Fast Fourier Transform) based on a set of simple equations:-



(**Figure 5.1**) Power /Frequency graphs for various simple equations

By taking the *wave-patterns* of the fluctuating population rates observed in the previous chapter we can perform a similar FFT transformation to establish their underlying frequencies.

Figure 5.2 shows three such a FFT graphs based on the population rate in the Game of Life as it is runs over 500 timesteps. The first shows the FFT of the population rate over the entire 500 timesteps. The second shows the FFT of the population rate over a subset of this time (the first 100 timesteps) and the third shows it over an opposing subset of time (the last 100 timesteps).

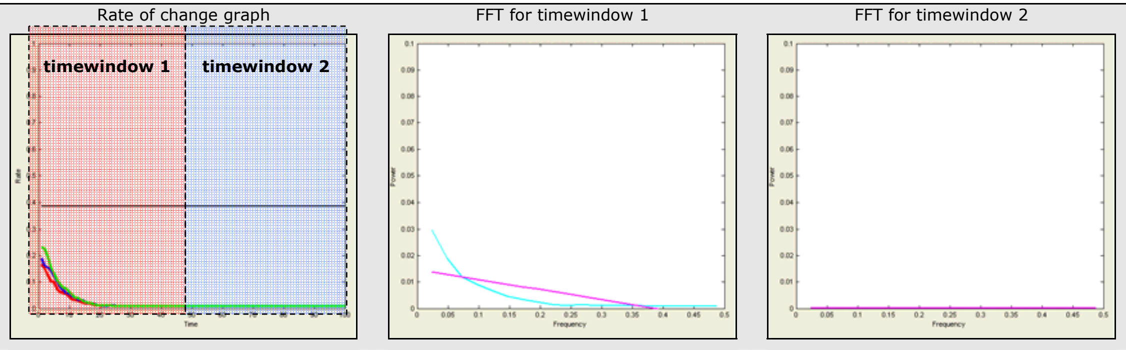


(**Figure 5.2**) Power/frequency graphs for the Game of Life

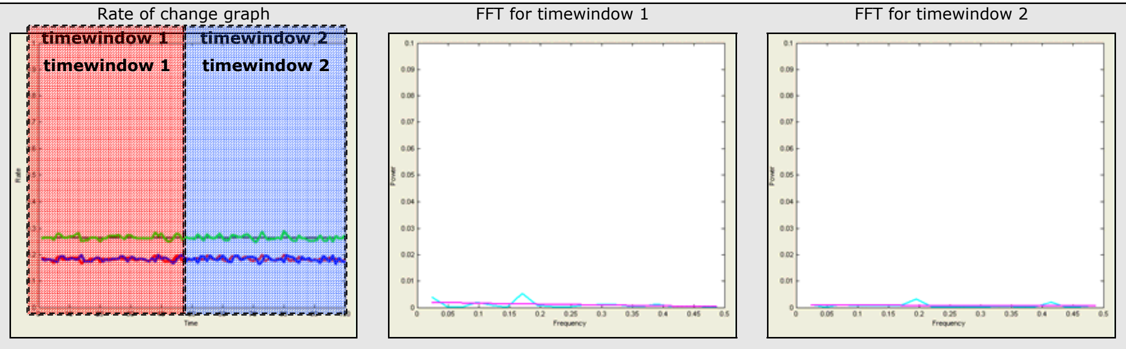
As can be seen, these subsets of time (timewindows) show distinctly different power/frequency distributions and this is of extreme importance for a number of the observations and conclusions that shall be drawn up later within this chapter.

5.2 POWER/FREQUENCY GRAPHS FOR CA EXAMPLES

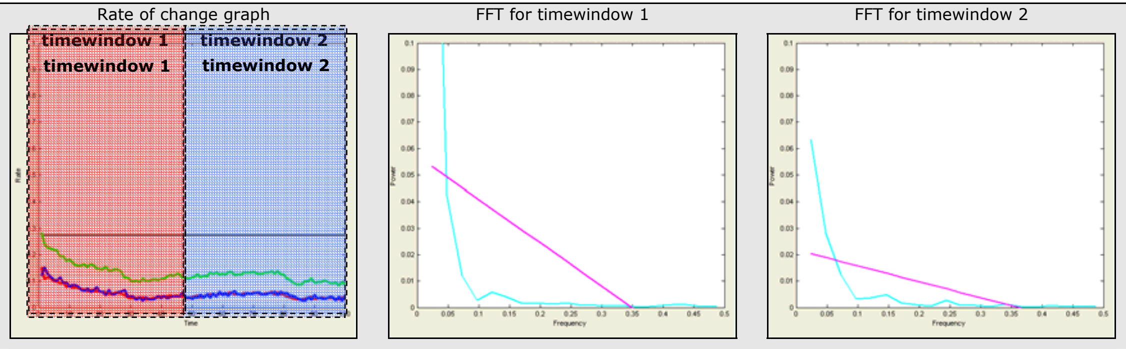
Figures 5.3, 5.4 and 5.5 below show the timewindowed FFT graphs for three of our example CA.



(**Figure 5.3**) FFT graphs for **ORDERED** example (rule 1)



(**Figure 5.4**) FFT graphs for **CHAOTIC** example (rule 3)



(**Figure 5.5**) FFT graphs for **COMPLEX** example (Game of Life)

Looking at the ordered example we see that the first timewindow (which corresponds with the settling down period) contains a certain amount of power at only the low frequencies. In the second timewindow (once the CA has settled down to its ordered behaviour), we find no power anywhere in the system; this is intuitively understood as at this stage the population rate remains constant.

In the chaotic system we find that both timewindows are very similar; they both show an extremely small amount of power distributed throughout the entire range of frequencies with no major or significant peaks.

Finally, in the complex system, both timewindows show significant amounts of power in low frequencies and this is a pattern that is maintained throughout the life time of the complex behaviour.

5.3 CLASSIFYING BEHAVIOUR THROUGH POWER DISTRIBUTION

These power/frequency graphs are rather telling when considered from the perspective of CA behavioural classification.

First, it seems that chaotic behaviour produces a distinctive type of population wave; a wave which contains a well distributed (but low) amount of power across the entire frequency band.

For our ordered systems (putting aside our settling down period for a moment) the population wave is also rather distinctive. It is generally flat and with little or no power.

Performing an FFT on the population wave of either of these behaviours, therefore, would allow one to readily determine the class of behaviour.

For complex behaviour the population wave also contains a rather distinctive pattern, this time having a significant amount of power in the lower frequencies. BUT, this pattern is also found within our settling down period for ordered CAs (albeit with a slightly lower amount of power).

This similarity prompts us to take another look at the "settling down" period. It has already been described as a kind of *complex* "bath emptying"; where chaos drains away to leave order.

In fact, if one examines certain CA with excessively long settling down periods (and stretches this period through the adjustment of the initial density) then this complex draining seems to get ever more complex.

Maybe, the settling down period can be considered to be a kind of complexity. The Game of Life, after all, ends up in an ordered state after about 300 timesteps; could these 300 timesteps not just be a prolonged kind of settling down?

The pattern of the population wave during the settling down period is certainly more indicative of complex behaviour than either chaotic or ordered behaviour. Could this not be because the two are both examples of the same systemic property? Is complex behaviour just a kind of prolonged settling down?

In an attempt to try and clarify, and perhaps answer, this suggestion, let us now make a slight, but important, detour through the often castigated world of $1/f^b$ noise.

5.4 THE UBIQUITY OF $1/F^B$ NOISE

$1/f^b$ noise (often called "flicker noise" or "pink noise") is cited within many disciplines but generally remains a rather poorly understood phenomenon (Milotti 2002). It is has been described by some as a "*measure of complexity*" and has been found to exist in a number of natural and evolutionary systems (West & Shlesinger 1990).

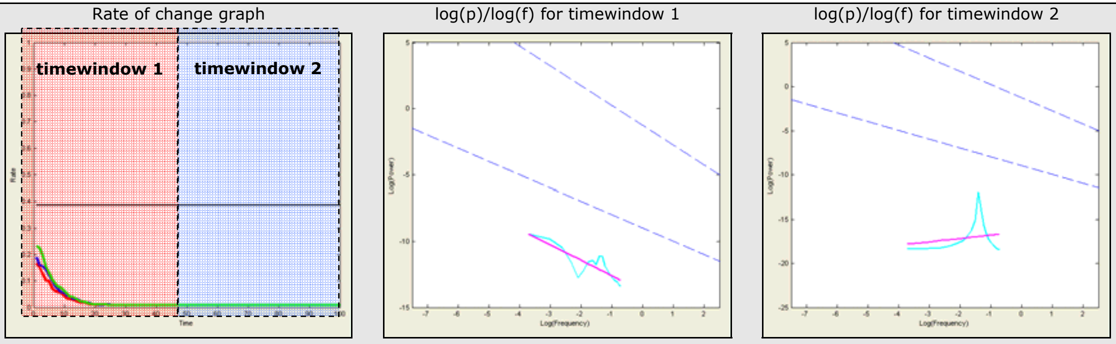
It is a measure of power versus frequency and can generally be stated as a phenomenon where the power decreases exponentially as the frequency increases. It is more formally defined as:-

$$\log(\text{power}) \approx \log(f)^b \qquad \text{where } 1 \le b \le 2$$

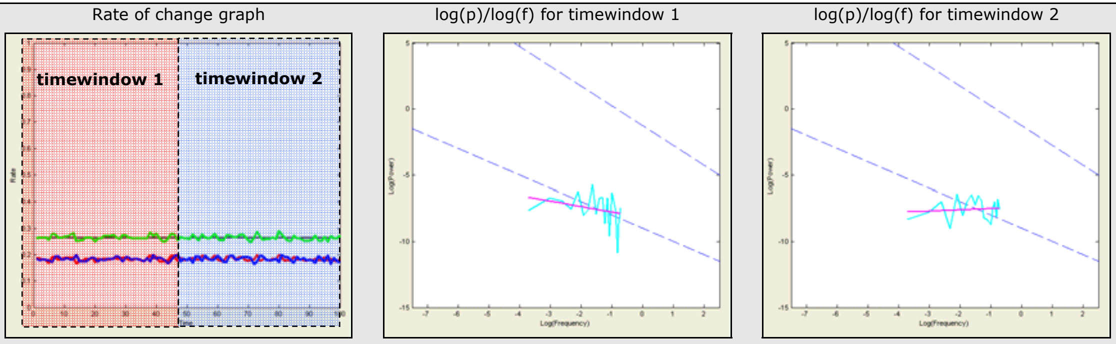
Graphically, this is shown as a straight line relationship between log(power) and log(frequency), where the gradient of the line corresponds with the variable **b**. For $1/f^b$ noise to be considered interesting, this gradient (b) must be of a significant value (i.e. between 1 and 2).

For some researchers, $1/f^b$ noise is a fabled power-law which describes a kind of temporal scale-invariance within dynamic systems (in other words, a variance which shows fractal characteristics in time rather than in space) and is a ubiquitous phenomenon unique to complex natural systems (Gisiger 2001); but for many, such a view is controversial and as such it is widely disputed.

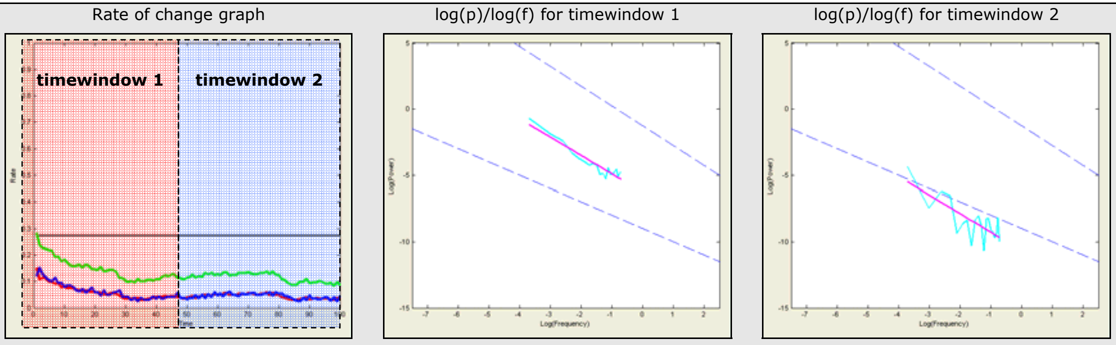
Given the previous power and frequency measures of our CA population dynamics, how do our own observations fit within this argument? Figures 5.6, 5.7 and 5.8 show the corresponding log/log plots for our earlier example timewindows (note, the dashed lines in each of the graphs demonstrate the critical gradients of **b=1** and **b=2**).



(**Figure 5.6**) Log(power)/Log(frequency) graphs for **ORDERED** example (rule 1)



(**Figure 5.7**) Log(power)/Log(frequency)graphs for **CHAOTIC** example (rule 3)



(**Figure 5.8**) Log(power)/Log(frequency) graphs for **COMPLEX** example (Game of Life)

Ordered behaviour shows a generally dull and flat log/log plot (figure 5.6) whilst chaotic behaviour is flat but with a few random peaks (figure 5.7). For both behaviours, the gradient **b** is distinctly low, and from the perspective of $1/f^b$ noise, they are uninteresting.

Complex behaviours, on the other hand, with their significant power in the lower frequency range show a log/log plot with a distinct and definite slope. In fact crucially, this slope (b) is found to waver constantly between 1 and 2. From the perspective of $1/f^b$ noise, this indicates that extremely interesting (or complex) behaviour is being displayed.

To reword this a little, it appears that $1/f^b$ noise is indeed highlighting a kind of complexity within cellular automata (if we include our settling down period), and in this light, it can indeed be called a "*measure of complexity*".

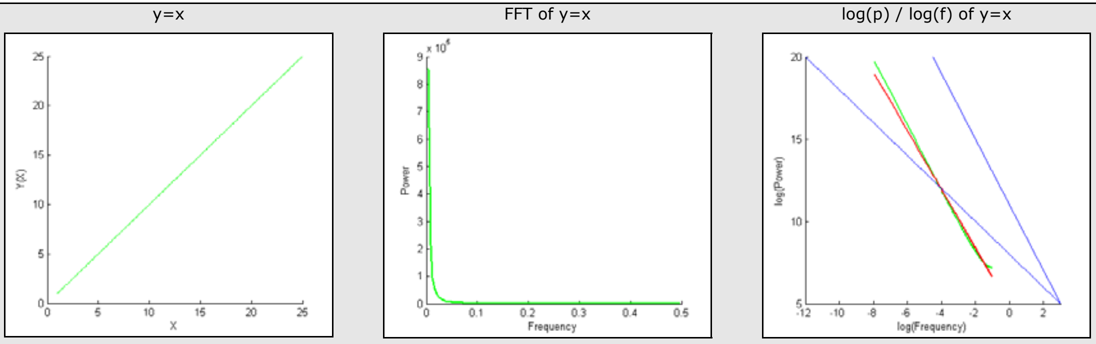
However, it is with *extreme caution* that we make such a statement; and, as shall be seen, it is not really a view that should be held onto too strongly. To generalise this statement to the wider domain of "*biological complexity"* would, at the very least, be misleading and, at the very worst, be flatly inaccurate.

As we shall see below, $1/f^b$ noise may highlight a factor which is significant in distinguishing complexity within cellular automata, but this factor may be so trivial that its ubiquity is meaningless.

5.5 UNDERSTANDING $1/F^B$ NOISE

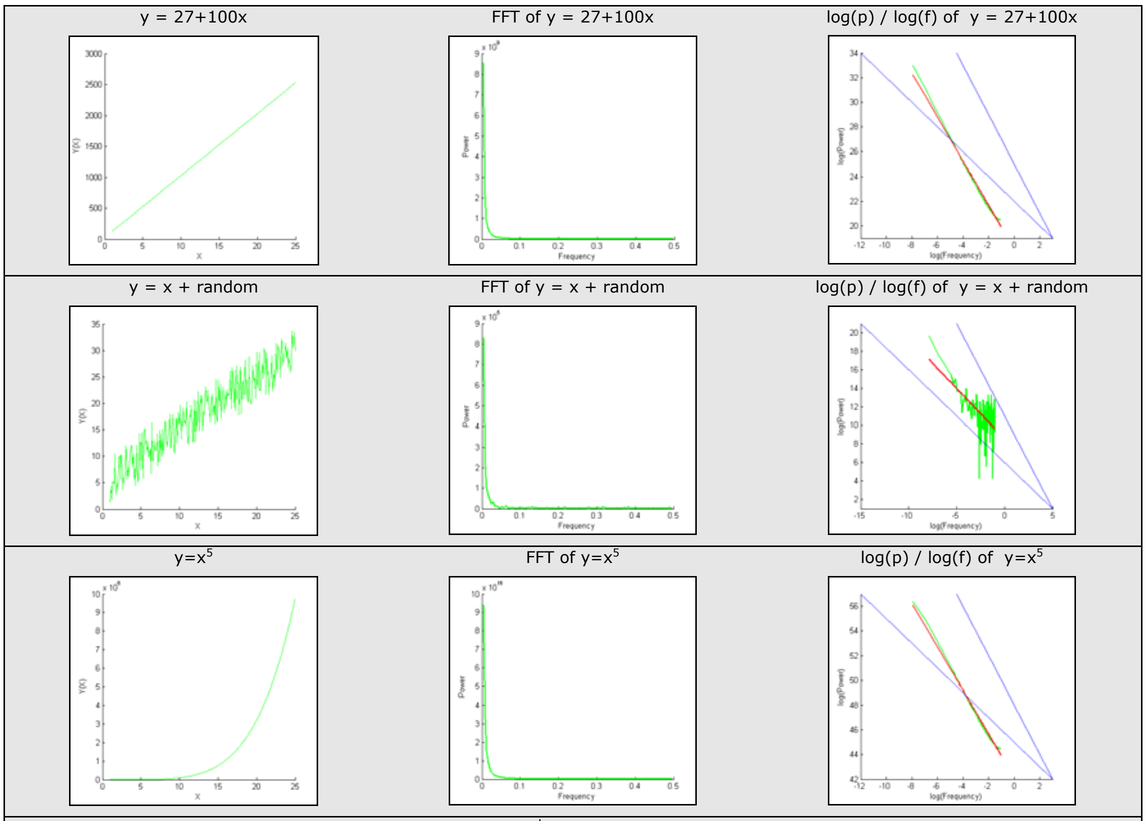
In order to propose a possible, and perhaps simple, explanation for the ubiquity of $1/f^b$ noise, let us take a quick look at the simplest of all wave functions; y=x.

Figure 5.9 shows this simple "*wave*" and, importantly, the results of a performing a FFT transformation upon it. Immediately we can see that it exhibits a high level of power in low frequencies dropping off exponentially as the frequency increases. Translated to a log/log plot, this forms a perfect display of $1/f^b$ noise (with $b \approx 2$). At this point one might, justifiably, ask: if $1/f^b$ noise is meant to be a measure for complexity, then how can it so easily be found in such a simple equation?

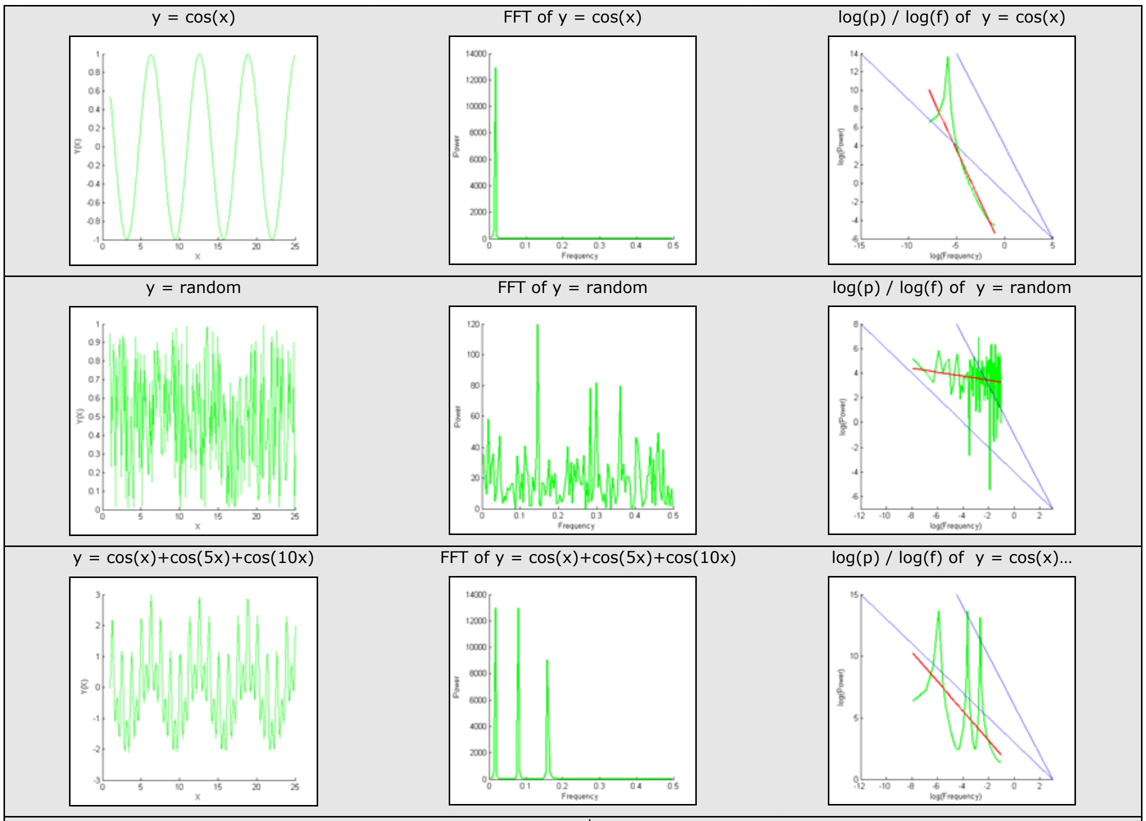


(**Figure 5.9**) Showing the FFT of **y=x**

Let us now examine this a little further, but studying the FFT of some other simple equations, to see whether there is a commonality amongst those that display $1/f^b$ noise.



(**Figure 5.10**) Examples of equations **WITH** $1/f^b$ noise



(**Figure 5.11**) Examples of equations **WITHOUT** $1/f^b$ noise

In fact, from these simple examples (figures 5.10, 5.11) we can *indeed* make a tentative proposal regarding the underlying reasons for the ubiquity of $1/f^b$ noise. *All of the examples where $1/f^b$ noise is found contain a proportional relationship between Y and X.*

This observation is particularly stark when contrasting the equations **y=random** with **y=x+random**. Simply by adding a relationship between y and x, the log(p)/log(f) plot suddenly shows the signature of $1/f^b$ noise. Could it simply be that $1/f^b$ noise is highlighting this relationship?

A thorough analysis of this is not provided within this paper, but from our intuitions this may make sense. A proportional relationship between Y and X basically means that Y is on some kind of long term gradient (or slope) with respect to X. This gradient would possibly be picked up in a Fourier Transform as a wave with an exceptionally long wavelength. Such a wave would effectively be translated as a large power in a very low frequency (exactly the shape of $1/f^b$ noise).

Let us now go back to our population graphs to see whether the $1/f^b$ "*measure of complexity*" corresponds with a gradient of some kind.

5.6 Complexity and Gradients

Consider the times where $1/f^b$ noise was found to be present, and contrast them with the times where it wasn't.

We have already stated that $1/f^b$ noise was NOT found during ordered or chaotic behaviours. At such times, the population rate has been observed to be flat; fluctuating in either an ordered or random fashion around a highly distinctive mean. In such behaviours, therefore, there is NO proportional relationship between X and Y. No gradient in Y exists.

Conversely, when $1/f^b$ noise IS present, we see that the population rate is either settling down or performing a kind of random walk. The settling down slope obviously fits with our proposed explanation of $1/f^b$ noise, but what of the random walk.

One thing that is clear about random walks is that finding a distinctive mean between any two points is extremely difficult. In fact, during a random walk one might describe a wave as being on a kind of *perpetual* and *constantly changing* slope. This paper unfortunately doesn't have the luxury to express this in any greater clarity, but if this idea of a perpetual gradient within a random walk is so, then it too fits with our proposed ubiquity of $1/f^b$ noise.

And so, perhaps, we are able to conclude that it is the long term gradient of the population rate that is the true signature of complexity (with $1/f^b$ noise simply being a method of highlighting this gradient).

If this IS the case, then our view of long term complex behaviours being *prolonged settling down* periods becomes much clearer. It could be said that it is upon the gradient towards behavioural stability that we find complex behaviours and it is through the self-organised extension of this period that complexity is able to stave off the dominance of order or chaos.

This view shall be discussed a little further in the concluding chapters of this paper; but first we shall get some more evidence for it from the mathematical world of probability.

6 PROBABILISTIC ANALYSIS OF POPULATION DYNAMICS

In chapter 4 we highlighted a number of observations regarding population dynamics within cellular automata. Within this chapter we aim to further understand these observations through an analysis of the "*Probable Dynamics"* of the same systems.

6.1 THE PROBABILITY OF LIFE

Imagine, if you will, an infinitely large CA with a random initial configuration where 25% of the cells are alive.

If we were to randomly choose a cell within this CA, we can say that there is a probability of 0.25 that that cell would be alive – this is the probability of life and it equates to the population rate at time t.

Now consider the Moore neighbourhood of 9 cells (figure 6.1). In our imaginary CA, we can say that each cell has a probability of life of 0.25; and hence can easily deduce that the chance of being in a configuration where all nine cells are *alive* is 0.25^9.

N5	N1	N6
N4	Self	N2
N8	N3	N7

(**Figure 6.1**) The Moore neighbourhood

Conversely, we know that the chance of *not being alive* (or being *dead*) is 0.75; and so we can state that the chance of being in a configuration where all nine cells are *dead* is 0.75^9.

The chance of at least 3 cells being *alive*, is 0.25^3, and the chance of at least 6 cells being *dead* is 0.75^6; and so we can state that the chance of being in a configuration where at least 3 cells are *alive* and at least 6 cells are *dead* (i.e. where exactly 3 cells are *alive*) is $0.25^3 \mathrm{x} 0.75^6$ (for each possible 3 cell configuration).

More formally, the chance of being in a configuration with **n** alive neighbours is $0.25^n \mathrm{x} 0.75^{9-n}$.

Finally, we can say that, for an **N** neighbourhood CA with a proportion of $\mathbf{a_t}$ alive cells, the probability for any cell being in a particular rule sate **r** (where r has $\mathbf{n_r}$ alive cells in its configuration) is calculated as:-

$$P(r_t) = a_t^{n_r} \cdot (1 - a_t)^{N - n_r} \qquad (6.1)$$

In other words (given a_t) *we can calculate the probability for being in each of the states in a CA ruletable*.

To clarify this a little, let us return to our imaginary CA with a population of 25% alive cells. It needs to have an underlying ruletable for us to calculate the probabilities, so for simplicity's sake, let us assume that it is a 1-dimensional, CA with a 3 cell neighbourhood and let us choose an arbitrary ruletable for it (shown in figure 6.2).

As we know that a_t=0.25, we can use the formula in 6.1 to calculate a probability, P(r), for any randomly chosen cell being in a particular configuration (figure 6.2).

Rule r	N1	Self	N2	Result	P(r_t)
1					0.421875
2	▨			ALIVE	0.140625
3		▨			0.140625
4	▨	▨		ALIVE	0.046875
5			▨	ALIVE	0.140625
6	▨		▨		0.046875
7		▨	▨		0.046875
8	▨	▨	▨		0.015625

(**Figure 6.2**) Calculating the probabilities for rule states (when a_t = 0.25)

Take rule state 1 for example; 0 of the 3 cells are alive for this rule (n_r=0, N=3). So:-

$$P(r_t) = 0.25^0 \cdot (1-0.25)^{3-0} = 0.421875$$

And so, the chance of our random cell being in this state is calculated to be roughly 0.42; and so too would it be 0.42 for some other randomly chosen cell, and so too for some other etc.

In fact, if we were to randomly choose all of the cells, we could predict that 42% of them will be in state 1. Similarly, roughly 14% of then will be in state 2, 14% in state 3 etc.

We know from looking at the ruletable that only those cells found to be in state 2, 4 or 5 will be alive at the next timestep; and so we can safely predict that, probabilistically speaking, roughly 33% of our imaginary CA is likely to be alive in the next timestep.

So in conclusion, from knowing the underlying ruletable, and the initial proportion of alive cells (a_t) we have been able to calculate a probable proportion of alive cells for the next timestep (a_{t+1}).

More formally, with **R** as the set of <u>*rules that result in an alive cell*</u>, we can state:-

$$a_{t+1} = \sum_{r \in R} P(r_t) \qquad (6.2)$$

And so in conjunction with (6.1):-

$$a_{t+1} = \sum_{r \in R} a_t^{n_r} \cdot (1-a_t)^{N-n_r} \qquad (6.3)$$

Because N, and all values of n_r in R are all constants derived from the CA ruletable, we can state that equation 6.3 above expresses a simple, one-dimensional, function (f) of the form:-

$$a_{t+1} = f(a_t) \qquad (6.4)$$

Thus, <u>we have devised a logistic map for the probable dynamics of the population rate</u>.

6.2 THE PROBABILITY OF BABIES AND CORPSES

In a similar vein, we can also derive a probability function for the number of babies and corpses that will occur for any given population rate. To recap, a baby simply describes the situation where a dead cell becomes alive, and a corpse describes the situation where an alive cell becomes dead.

So, studying the ruletable of our imaginary CA once again (figure 6.3 below) we can highlight the results where a baby occurs or when a corpse occurs (simply be examining the rules where the *self* cell and the *result* are different).

Rule r	N1	**Self**	N2	**Result**	P(r)
1					0.421875
2	▨			▨ **BABY**	0.140625
3		▨		**CORPSE**	0.140625
4	▨	▨		▨	0.046875
5			▨	▨ **BABY**	0.140625
6	▨		▨		0.046875
7		▨	▨	**CORPSE**	0.046875
8	▨	▨	▨	**CORPSE**	0.015625

(**Figure 6.3**) Calculating the probabilities for babies and corpses (with a_t=0.25)

In a similar way to before, we can state that the probability of falling into a configuration that results in a baby is $P(r_2)+P(r_5)$ (in this instance approximately 0.28), leading us to claim that approximately 28% of our CA will be babies in timestep (t+1).

The probability of leading to a corpse is the sum of $P(r_3)$, $P(r_7)$ and $P(r_8)$; and so we can say that our CA will contain approximately 20% of corpses in timestep (t+1).

More formally, by defining two further subsets of our ruletable (with **β** as the set of rules that leads to a baby, and **δ** as the set of rules that lead to a corpse) we can state:-

$$b_{t+1} = \sum_{r\in\beta} P(r_t) \quad =$$

$$b_{t+1} = \sum_{r\in\beta} a_t^{n_r} \cdot (1-a_t)^{N-n_r} \qquad (6.6)$$

And:-

$$c_{t+1} = \sum_{r\in\delta} P(r_t) \quad =$$

$$c_{t+1} = \sum_{r\in\delta} a_t^{n_t} \cdot (1-a_t)^{N-n_r} \qquad (6.7)$$

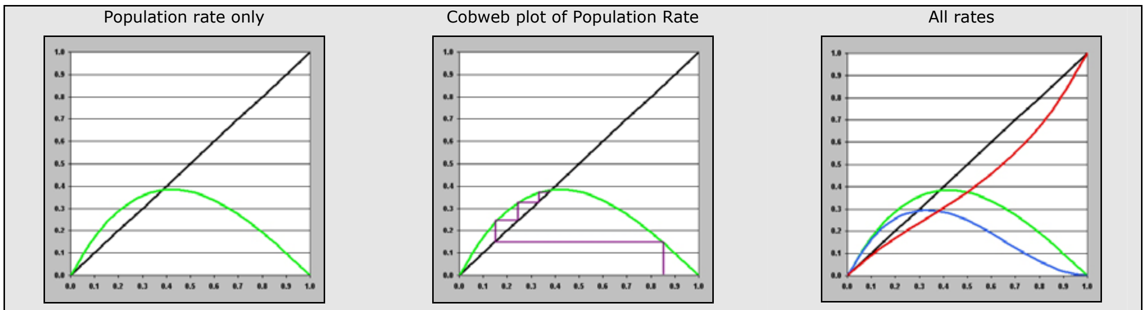
These equations are not one-dimensional logistic maps as they require an outside variable (a_t) in order to be solved; however, as we shall see, they do provide a useful tool for analysing the probable dynamics of cellular automata.

6.3 HOW TO PLOT THE DYNAMICS

The mathematics of dynamical systems has a marvellous tool for analysing one-dimensional logistic maps, known as the Cobweb plot.

By plotting the function y=f(x) upon the same set of axis as y=x one can immediately begin to visualise the dynamics of a one-dimensional system. For example, where the two paths cross, the system can be said to contain a fixed point. This is a point at which the value of the next timestep will be the same as the current; or, to put it another way, a point where the value will remain the same forever. By then adding a Cobweb plot, one can visualise the dynamic path that will be taken from any particular starting value; crucially allowing one to establish whether a fixed point is stable or unstable.

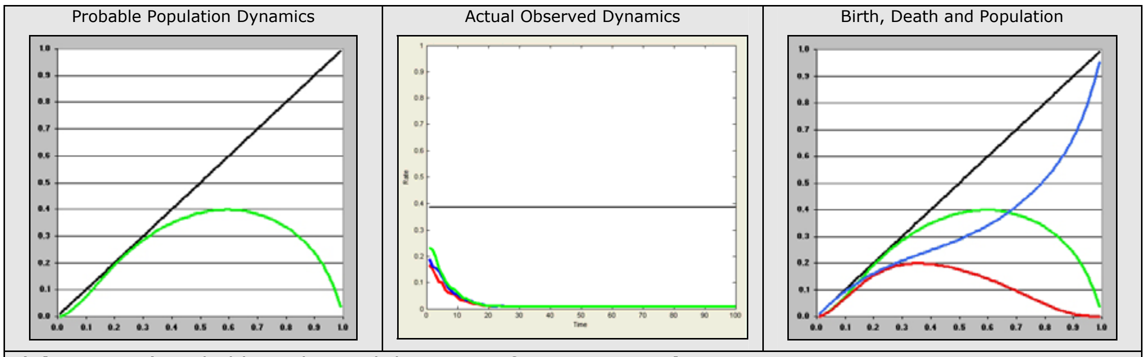
Figure 6.4 shows these plots for the previously defined functions for probable population, birth rate and death rates (using the ruleset already provided for our imaginary CA). The x axis is the value a_t, and the y axis is the value of either a_{t+1}, b_{t+1} or c_{t+1} respectively. Keeping the convention used in our rate-of-change graphs we use a GREEN line for the population rate (a_{t+1}), a RED line for our birth rate (b_{t+1}), and a BLUE line for our death rate (c_{t+1}). As the birth rate and death rate are NOT logistic maps, the concepts of fixed points and cobweb plotting are not relevant for them and so care must be taken when considering them on the same axis.



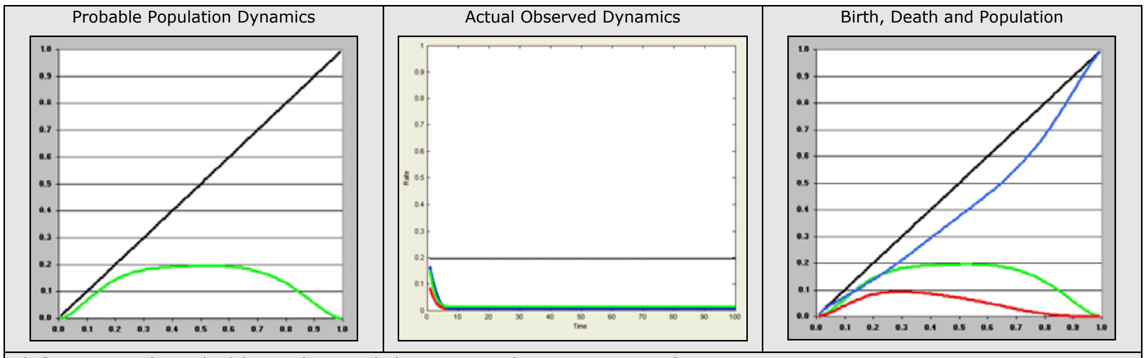
(**Figure 6.4**) Building a Cobweb plot for the probable dynamics of a CA

6.4 PROBABLE DYNAMICS GRAPHS FOR CA EXAMPLES

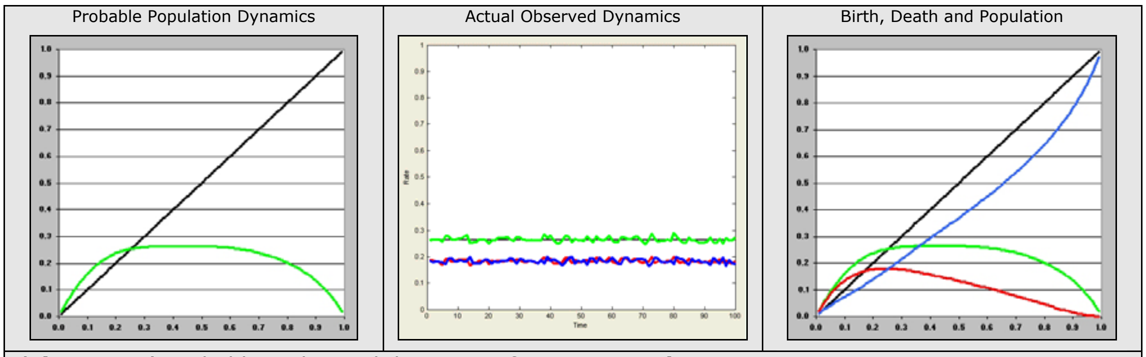
Armed with these plotting techniques, we can now return to our example CAs. Using the previously defined probability functions and the specific ruleset for each CA, we can draw up a series of graphs that display the probable dynamics of each. These are provided in the following figures, alongside the actual observed dynamics already highlighted in chapter 4.



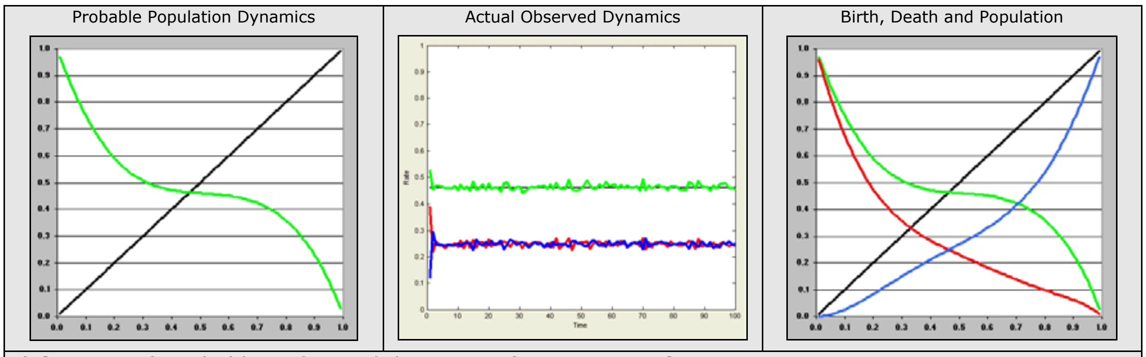
(**Figure 6.5**) Probable and actual dynamics of **ORDERED rule 1**



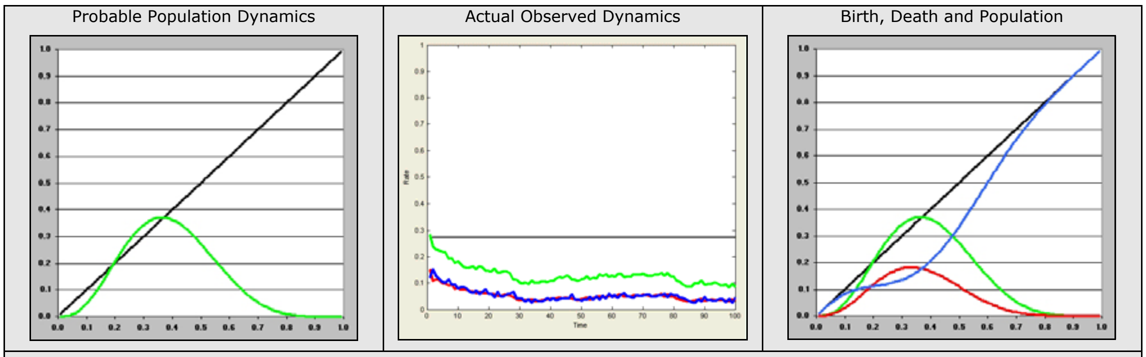
(**Figure 6.6**) Probable and actual dynamics of **ORDERED rule 2**



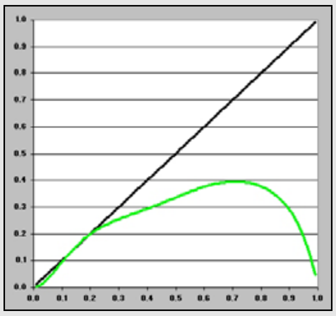
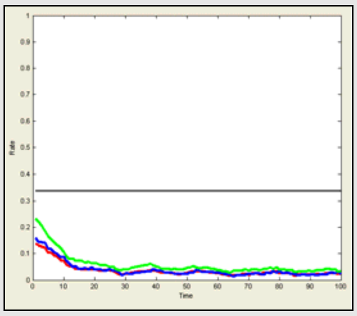
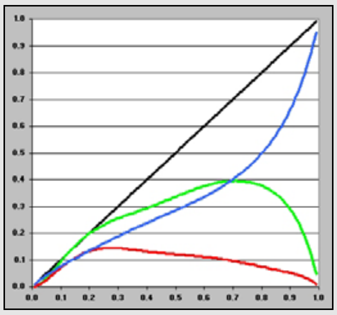
(**Figure 6.7**) Probable and actual dynamics of **CHAOTIC rule 3**



(**Figure 6.8**) Probable and actual dynamics of **CHAOTIC rule 4**



(**Figure 6.9**) Probable and actual dynamics of **COMPLEX rule 5 (Game of Life)**

Probable Population Dynamics	Actual Observed Dynamics	Birth, Death and Population
		

(**Figure 6.10**) Probable and actual dynamics of **COMPLEX rule 6**

These graphs highlight a number of important factors relating to the underlying dynamics of CA behaviour, and the remainder of this chapter aims to highlight and examine some of the most striking.

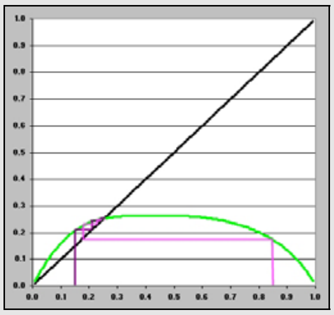
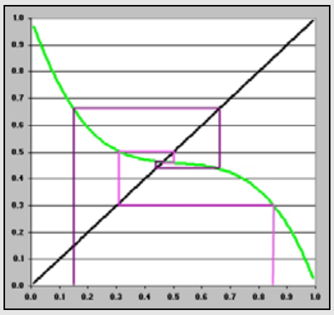
6.5 Population Attractors of Order and Chaos

The first, and potentially most important, observation relates to the fixed points found within the probable population dynamics graphs for our ordered and chaotic systems (an examination of the fixed points within complex systems is left for a later section of this chapter).

First, we note that all systems have a fixed point at ZERO (let this be called the Z fixed point). This may initially appear somewhat trivial as in the language of CAs it simply means "*if no cells are alive (a_t = 0), then no cells will come alive (a_{t+1} = 0)*". However, the "attraction" of this fixed point varies crucially amongst the different classes of CA.

For both of our ordered CAs the y=f(x) curve lies entirely below the y=x line and so no further fixed points exist. Consequently, this Z fixed point can be shown to be a perfectly *stable attractor*. In the language of population dynamics, this means that the population rate is destined to decay towards zero and, crucially, this corresponds **precisely** with what was previously observed in the real-time dynamics for ordered systems.

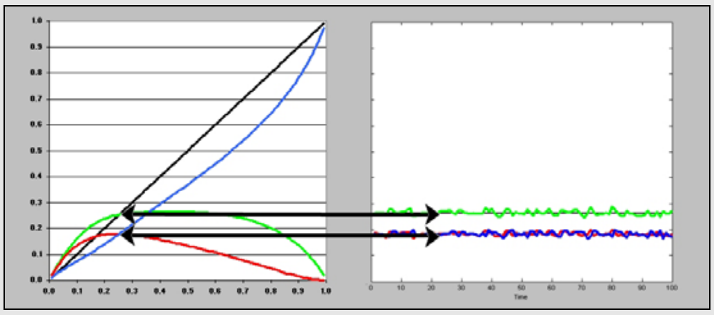
Conversely, in the probable dynamics plots for our chaotic CA, we find a second fixed point at some value higher than zero (let this be the C fixed point). The overall shape of the y=f(x) curve is such that this C fixed point can be shown to be a perfectly *stable attractor* (figure 6.11). Crucially, this once again matches what was observed in the real-time dynamics for chaotic systems. Whatever the initial conditions, the population rate always converges toward some value higher than zero.

Rule 3	Rule 4
	

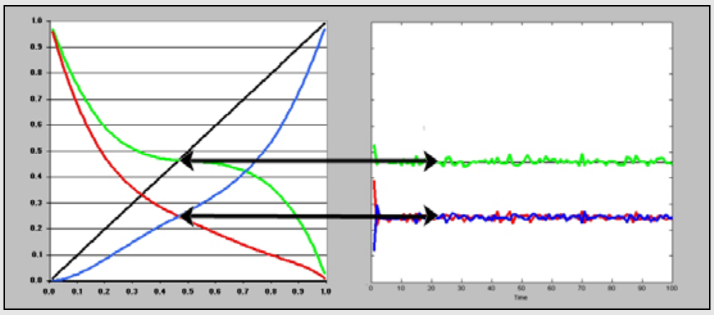
(**Figure 6.11**) Cobweb plots showing the stability of the **C fixed point**

In fact, the C fixed point seems to coincide **exactly** with the values observed for the *points of behavioural stability* (see figure 6.12). The C point (or chaotic) attractor drags the population rate up to a point of stability, from where the it simply fluctuates in a random manor.

Rule 3



Rule 4

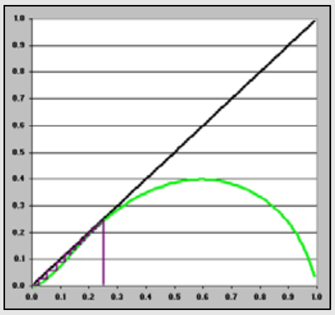


(**Figure 6.12**) Comparing probable with actual dynamics

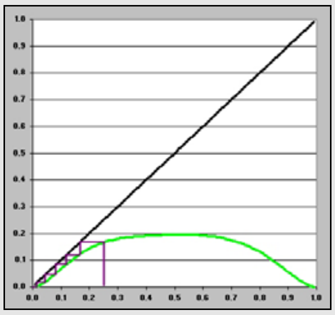
6.6 Settling Down

Let us now examine another phenomenon of CA behaviour that the probability graphs are able to accurately predict (at least within ordered or chaotic systems); the settling down period prior to behavioural stability. We have already observed that the two ordered examples have slightly different settling down periods; the first (rule 1) settles down after about 20 timesteps, whilst the second (rule 2) settles down after just 5. By drawing up a cobweb plot with similar initial conditions (a_t=0.25) we can see that this settling down period corresponds **exactly** with the expected number of steps needed to reach the Z attractor.

Rule 1: Settling down takes 20 timesteps



Rule 2: Settling down takes 5 timesteps



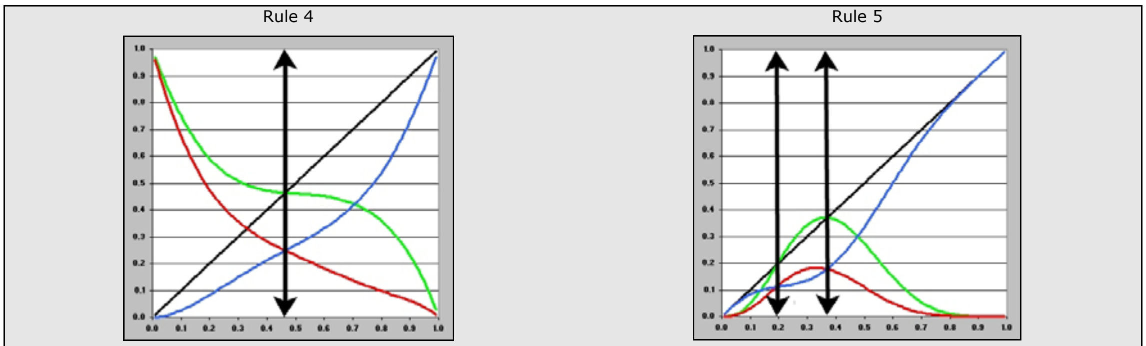
(**Figure 6.13**) Cobweb plots showing the **settling down** period starting at a_0 = 0.25

Within the graph for rule 1, y=f(x) comes very close to y=x, and this can be seen as a kind of "slow rut", during which the rate of change slows right down. Conversely in the graph for rule 2, the gap between y=f(x) and y=x is much wider, meaning that the rate of change will be a lot quicker.

6.7 Birth and Death

Before moving on to look at the probable dynamics of our complex system, let us first take a brief look at the probable dynamics of birth and death across all systems (please note once again that the birth and death graphs aren't setup to provide us with any of the traditional dynamical systems analysis of fixed points and attractors; they are merely cross-referenced with the population rate).

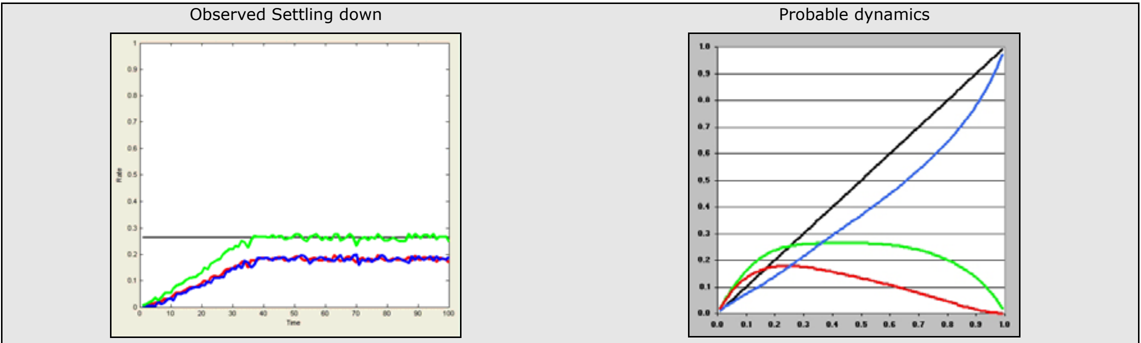
The first, intuitive, observation is that the point at which the two rates cross corresponds with the point at which the population rate of the system remains stable. In other words, a system will maintain a level population rate only when the number of births and deaths are the same (figure 6.14).



(**Figure 6.14**) The correspondence of Birth/Death crossover and Population Fixed Points

Secondly, from our earlier comparison of probable dynamics for chaotic systems with real-time observations (figure 6.12) we see that this birth/death crossover point also corresponds **exactly** with the distinctive mean that was observed for the two rates once behavioural stability had been reached.

Finally, however, it is important for us to observe a point where the probable dynamics and realtime observations don't quite marry up. Consider once again, the settling down period for our chaotic rule 3. We observed that the population rates remain in close proximity to one another on the climb towards stability, but in looking at the probable dynamic graphs this is at odds with the probabilistic behaviour (figure 6.15).



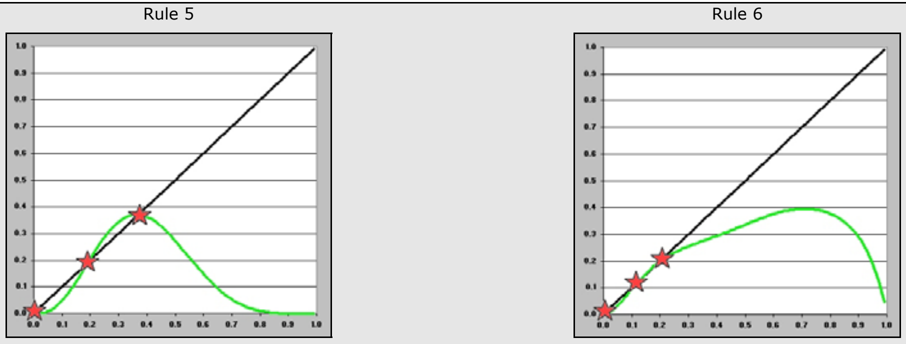
(**Figure 6.15**) Improbably birth and death convergence in chaotic settling down

According to probability the birth rate should be significantly higher than the death rate during this period; but, for some reason, reality maintains the birth/death status quo. During this complex settling down period, the birth and death rate appears to be acting in a somewhat improbable fashion; an observation that is unfortunately beyond the scope of this paper but would benefit greatly from further investigation.

6.8 COMPLEXITY WITHIN PROBABILITY DYNAMICS

Finally, we look at complexity from the perspective of our probability dynamic graphs.

In looking at figures 6.9 and 6.10 one immediate and potentially stunning observation arises. It would appear that both of our examples contain BOTH a semi-stable Z fixed point AND a semi-stable C fixed point with an unstable fixed point providing the bifurcation of stability somewhere in between (figure 6.16 highlights these fixed points a little more clearly).



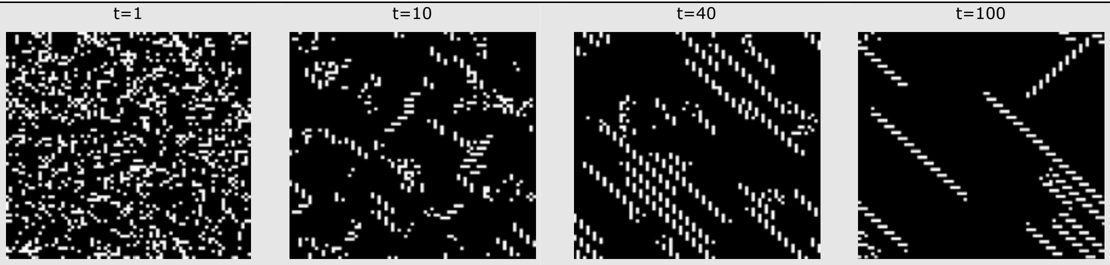
(**Figure 6.16**) Highlighting the Fixed points in examples of complex rulesets

Immediately one might be led into thinking that this is some kind of unique signature of complexity; a mix of attractions from the depths of order and the heights of chaos.

One might begin to visualise a system which is kept in continual motion from the forces of ordered decay and chaotic growth; a population rate forced into a perpetual gradient in a constantly changing landscape. One might further be reminded of the early Complexity Theoretic ideas of phase transitions but where complexity lies not on the *edge*, but on the *overlap* of chaos and order.

But alas, the results are a little misleading. Not all complex rulesets contain the two attractors of C and Z; in fact, a reasonable proportion of them contain no C fixed point at all (making them look probabilistically ordered). As an example, consider the following ruleset for a complex CA (full ruletable found in Appendix A).

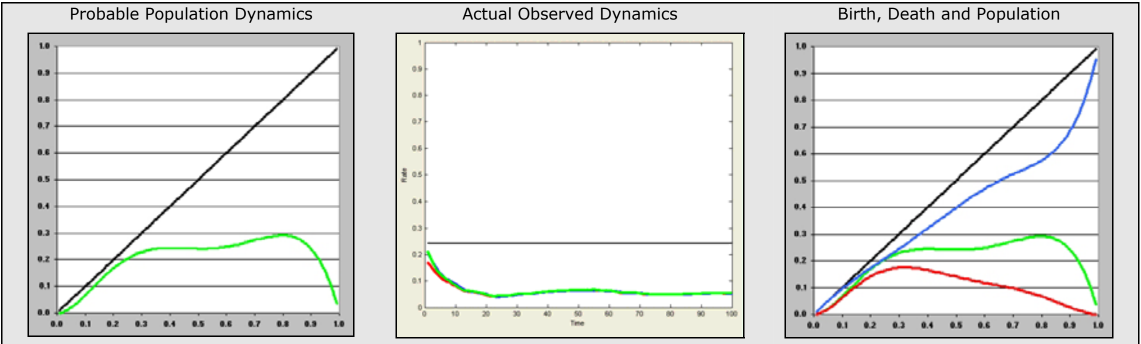
RULE 7: COMPLEX : MOVING DIAGONAL LINES



(**Figure 6.17**) Snap shots of CA Rule 1, taken every 10 timesteps

Chaos dissolves after around 10 timesteps to leave (predominantly) a series of growing diagonal lines. Collisions occur where the lines meet leading to a burst of chaotic activity until on line dominates. After a very long period of time, the system reaches an ordered state (where all diagonal lines have battled it out and no more collisions will take place).

The population and probability dynamic graphs for this new CA are shown below (figure 6.18).



(**Figure 6.18**) Probable and actual dynamics of **CHAOTIC rule 4**

The observed behaviour of the system is most definitely complex, but the probable dynamics graphs seem far more in tune with an ordered system (there is a distinct lack of a C fixed point). In fact, accordingly to the probability graphs, the population rate should just dissolve away to nothing (albeit reasonably slowly).

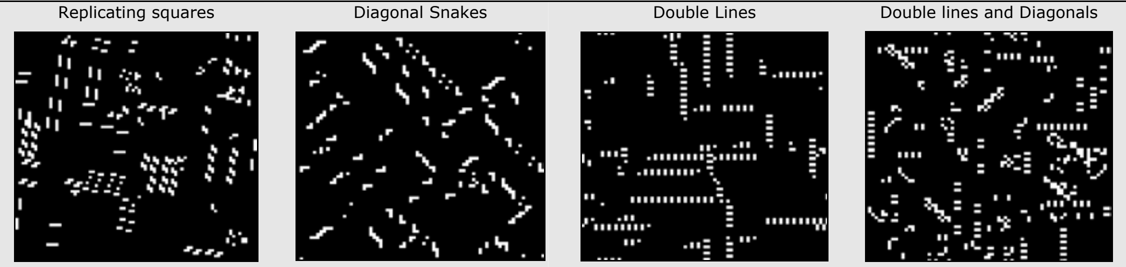
From the wider study of all the complex rulesets found within the *Looking for Life* report we seem to find two categories of complex CA ruleset. Some with the two points of attraction, and others with just a Z point attractor.

Without giving an evolutionary description of each it is difficult to fully detail the difference between the two, but in watching their multifarious evolutions in real-time, they do display subtly distinctive characteristics.

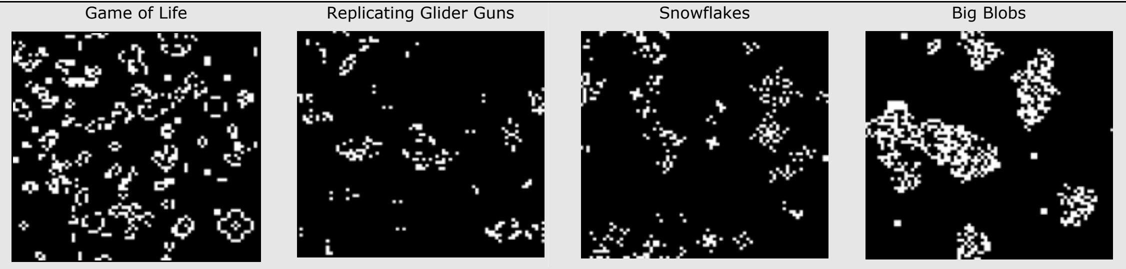
For the multi-attractor complex rulesets, the complex behaviour is, for want of a better word, more complex. It often contains large, long cycle replicators and pockets of chaos which sporadically fire off gliders with a generally long and convoluted cycles of states.

For the more ordered looking complex rulesets, the complex behaviour usually consists of a far cleaner environment containing simple 2-state replication and minimal gliders made up of just a couple of blocks and with just a couple of periods in their lifecycle.

Figures 6.19 and 6.20 give a handful of screenshots to try and help illustrate this difference.



(**Figure 6.19**) Example screenshots of *ordered* complexity



(**Figure 6.20**) Example screenshots of *multi-attractor* (chaotic) complexity

To take this is little further, it might be said that the complex behaviour found within CAs varies all the way from simple (2 or 3 state cyclical patterns) to much more unusual (high-state) cycles surrounded with pockets of chaos. In other words, there is a kind of scale, a kind of "strength of complexity", which corresponds (to some degree) with the observations from the probability dynamics graphs. The *louder* the chaotic attractor, the more *complex* the behaviour found within the system.

This admittedly loose and tentative observation introduces the concept of a *"strength of complexity"* and is looked at a little further in the concluding chapter of this report. For the most part, however, this idea really needs much further investigation if it is to gain credibility and, most regrettably, this cannot be achieved within the bounds of this paper.

6.9 CLASSIFICATION THROUGH PROBABILITY DYNAMICS

Through the wider study of a great many ordered, chaotic and complex rulesets, it seems that the following statements can be made regarding the classification of CA behaviour through the study of probability dynamics.

Firstly, all chaotic CA contain a *perfectly stable* C-point attractor toward which the population rate will converge.

Secondly, all ordered CA lack this C-point attractor and, instead, contain a perfectly stable Z point attractor; leading the population rate to dissolve quickly away (what is not captured by the probable dynamics is the extent to which simple blinkers and blocks may be seen – this is discussed within the concluding chapter of this paper).

Finally, about two thirds of complex rulesets contain two, *semi stable,* attractors with the rest containing just a single, *stable,* Z point. Those that have two attractors show a far "stronger" kind of complexity than those that only contain one.

From these observations we can conclude that a CA with a *perfectly stable* C-point attractor can be predicted to be chaotic; and that, similarly, a system with a *semi-stable* C-point attractor can be predicted to be strongly complex.

Unfortunately, the lack of a C-point attractor may indicate either an ordered system or a weakly complex one and so we do not yet have a complete prediction scheme from probability dynamics. It is noted, however, that being able to predict chaotic and strongly complex behaviours is a faculty that is of importance enough to make probability dynamics a study worth further examination.

7 Conclusions

7.1 Refined Definitions of Complexity

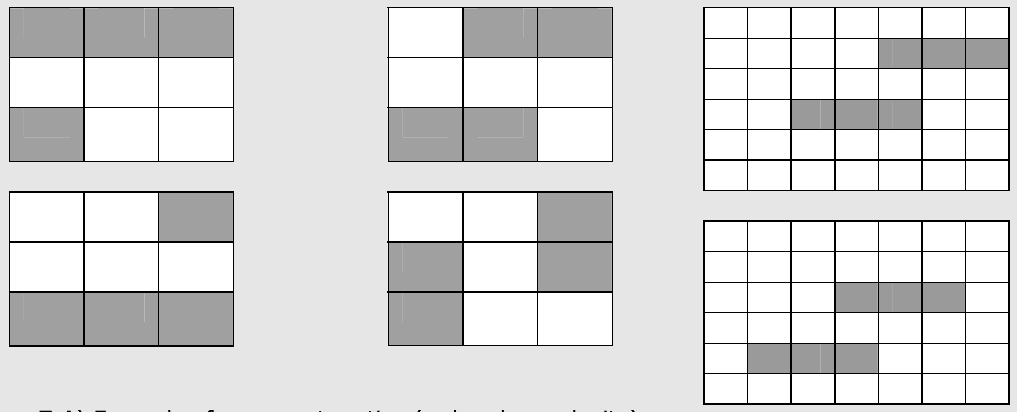
To conclude, we shall now attempt to combine some of these observations in order to refine the current definitions of order chaos and complexity within cellular automata; but first let us remind ourselves of the link between entropy and population rate highlighted in chapter 4. It was stated that as the population rate rises, it drives up the entropy of the system (through the generation of a wider repertoire of rule-states).

Consider now a dynamic system with a stable C point attractor. Probability alone dictates that the population rate will naturally adjust to reach this point of stability. During this "settling down" the entropy of the system fluctuates in accordance with the population rate's own fluctuation; if the population rate makes a gentle incline, so too will the entropy. This happens all the way towards the point of stability when both population and entropy will remain at a reasonably high value. In short, the existence of the C-point attractor pulls the system to a point of high and stable entropy – a point, in thermodynamics, known as chaos.

Next consider a perfectly ordered dynamical system where all cells eventually die (Wolfram Class I). Such a system contains a perfectly stable Z point attractor and the population rate swiftly drops off to zero. This drop pulls the entropy of the system down with it leaving the system, eventually, in a state of zero entropy – a point of thermodynamic order.

Some dynamical systems don't quite die away to complete quiescence, they leave behind exceedingly simple blocks or blinkers (Wolfram Class II). Although *close* to perfect order (and most definitely a kind of behavioural stability) one may like to reconsider this class as the first step in an increase in behavioural complexity. The population rate in such a system drops away extremely quickly towards zero (as predicted by the probable dynamics) but is kept slightly away from zero by this slight rise of complexity. In such a system, the entropy, dragged by the low population rate, is extremely low, and extremely stable, but it is not quite in a state of perfect order.

As we increase the complexity these blocks and blinkers become a little more complex and start to contain a kind of *basic motion* or *ordered complexity*. An example of this motion is given in figure 7.1 below (taken directly from our rule 7).


(**Figure 7.1**) Example of *emergent motion* (ordered complexity)

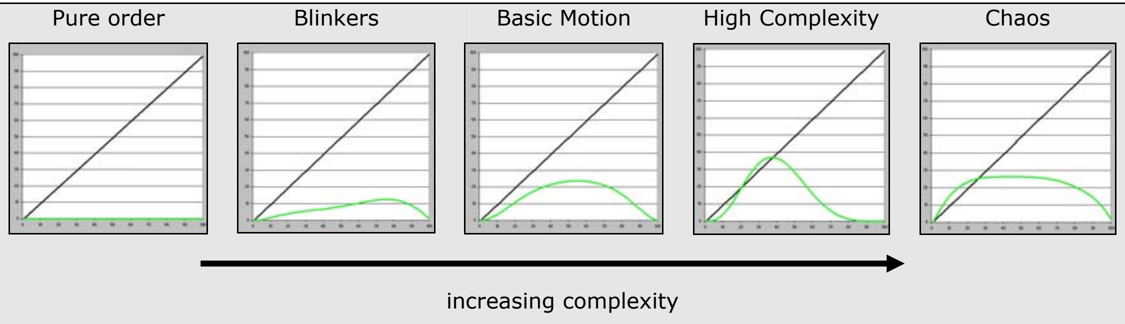
In such a system, the population rate drops again towards zero, but collisions of emergent moving particles add a certain amount of random-walk noise into the proceedings; this is mirrored within the system's entropy which shares these occasional fluctuations but remains at a generally low and stable value.

As the complexity is increased further, the C-point attractor begins to gain strength and the system starts to become partly driven by chaos. At this point (whilst neither the C and Z point attractor is fully stable) we see much more advanced kinds of behaviour, including complex replicators and multi-state movement (gliders). The population rate fluctuations become far louder and so too do the corresponding fluctuations of entropy.

As the C-point attractor gets stronger and stronger, the complexity of the system is eventually dragged to a point where chaos (the most complex of all systemic behaviours) finally takes hold.

This hypothesised rise in complexity within dynamic systems is only tentatively proposed and requires a great deal more investigation in order to gain acceptance as a general and measurable property of cellular automata (and perhaps complex dynamic systems in general). It bears great semblance with the early ideas proposed by Chris Langton and the Santa Fe Institute (Langton 1990); but has a crucial shift in including fixed and periodic (Class II) behaviours as basic examples of complex emergent phenomenon.

The rise is partly observed within the probability diagrams outlined in chapter 6, but unfortunately it isn't yet stark enough to be considered a measurable quantity. For a general illustration, however, the purposely evocative series of diagrams in figure 8.1 aims to show an initial flavour of how this rise in complexity may look once observed.



(**Figure 8.1**) The rise of complexity

8 PROPOSITIONS

8.1 A MEASURE OF IMPROBABILITY

Let us now take a look at a possible piece of future work which aims to create a firmer measure for this hypothesised rise in complexity.

Chaotic systems, it seems, work in a probable fashion (in other words, they are in accordance with the probable dynamic of the system).

Purely ordered systems (ones which end up in a purely quiescent state) also work in a probable fashion (one that is in accordance with the probable dynamics of the system).

The rise of blinkers, and the further rise of motion, seems to equate to a general *decrease* in accordance with the probable dynamics of the system (or in other words, a rise of improbability). The more convoluted the emergent phenomena, the more *improbable* it becomes (the less in accordance it is with the probable dynamics predicted for the system). Perhaps, therefore, one could equate a rise in complexity with a rise in improbability.

If one were to be able to determine a mathematical model of the actual dynamics (say from a test set of initial conditions) and combine this with the models of probability one could determine a measure for improbability. As such, one would have a potential measure for complexity within cellular automata.

8.2 APPLICATIONS FOR THE MATHEMATICAL METHODS

The measurements of Population Dynamics outlined in chapter 3 are not exactly new to the field of cellular automata research; however, until now they have not been precisely defined so as to represent a formally methodology for CA analysis. These methods, free from the binds of cell-states, are measurable for all CA of any size or shape and as such they represent a power tool for cross referencing amongst all CA research. It is also suggested that these methods will prove to be general enough to move beyond the basic examination of CAs to the wider fields of discrete dynamic systems (with particular focus on the notions of complexity within such systems). For example, one might extend this study to observe the global behaviours of Random Boolean Networks and contrast the results with this research to gain a wider understanding of complexity within both systems.

The introduction of probability theory and the shift from rule-space to probability space in chapter 6 has proven to be an exceptionally useful method of moving from the world of discrete dynamics to the more attractive world of continuous dynamical systems. By defining a logistic map for a macroscopic behaviour (the population rate) based on the microscopic discrete dynamical rules of the system, it was possible to use the standard visualisation techniques adopted for continuous dynamics systems to view the probable attractors of this behaviour. Such a technique may prove to have a much wider scope of application within a number of different fields ranging from economics to the burgeoning field of mathematical biology. Any complex system based on a simple set of discrete rules may benefit from the introduction of a probabilistic analysis of likely macroscopic behaviours.

8.3 Life Far From Equilibrium

Let us close this paper with a look at the complexity's place within the general sciences of systems (particularly within the resurgent field of thermodynamics). An initial look at the timeline graphs in chapter 4 showed a "settling down" period for both ordered and chaotic behaviour, after which a period of behavioural stability (or behavioural equilibrium) was observed to exist. For complex behaviour, no such point of behavioural stability was observed.

Through the Fourier transforms of chapter 5, it was found that these behavioural equilibria (ordered or chaotic) both fluctuate with generally insignifant amounts of power distributed throughout the frequency band. Complex and settling down behaviours, on the other hand, both possess a disproportionate amount of power in the low frequencies. From the perspective of $1/f^b$ noise, these power distributions seemed indicative of the presence of a "*gradient*" within the waveform. As both *settling down* and *complex* behaviours possess such a power distribution, it was proposed that <u>both</u> represent a *slope* towards behavioural stability; the only difference being how long this slope is able to be maintained.

From Chapter 6's examinations from the world of probability it was found that the settling down period could be redefined as the process of reaching a systemic attractor (a point of systemic equilibrium); or, in other words, that *settling down* was a process of equilibrialisation.

If complex behaviour is just a prolonged settling down period, then it makes sense that it is the interactions of emergent particles which are the <u>means</u> for this proloinging to take place. In other words, complex emergent phenemomena are the means by which a system keeps itself far from equilibrium (see also Prigogine 1967).

It is interesting to contemplate that (for CA at least) the behaviours that maintain this "*far from equilibrium"* position for the longest are those that contain gliders (emergent particles able to move) and replicators (emerging particles able to reproduce).

Movement and reproduction, the prolongers of systemic disequilibrialium and the stavers of order and chaos, also happen to be two of the fundamental properties of Life.

In modern thermodynamics, gradient reduction and the principle of maximum entropy production have already begun to make a similar crossover between Life and its position within our systemic Universe. Schneider and Sagan, authors of a wonderful book on the subject, capture these ideas magnificently when they declare that "*nature abhors a gradient*" (Schneider & Sagan 2005).

From the perspective of this paper, however, if nature abhors a gradient, maybe complexity, and Life, adores one.

9 REFERENCES

Baas N.A. (1994) "*Emergence, hierarchies and hyperstructures*", Artificial Life III, SFI Studies in the Sciences of Complexity (17) pp. 515-537

Baas N.A., Emmeche C. (1997) "*On E*mergence and Explanation", Intellectica (25) pp. 67-83

Berlekamp E. R., Conway J. H., Guy R. K. (1982) "*Winning Ways for Your Mathematical Plays, Vol. 2: Games in Particular*", Ch 25, London: Academic Press

Bickhard M.H., Campbell D.T. (2000) "*Emergence*", in Downward Causation, Aarhus University Press.

Campbell R.J., Bickhard M.H. (2002) "*Physicalism, Emergence and Downward Causation*", unpublished www.lehigh.edu/~mhb0/physicalemergence.pCf

Capra F. (1996) "*The Web of Life: A new synthesis of mind and matter*", Harper Collins

Capra F. (2002) "*The Hidden Connections*", Harper Collins

Cariani P. (1997) "*Emergence of new signal-primitives in neural systems*", Intellectica (25) pp. 95-143

Cariani P. (1990) "*Emergence and Artificial Life*", Artificial Life II, SFI Studies in the Sciences of Complexity (10) pp. 775-798

Crutchfield J.P. (1994) "*Is anything ever new? Considering emergence*", Complexity: Metaphors, Models and Reality, Santa Fe Institute Studies in the Sciences of Complexity (19) pp. 479-497

Crutchfield J.P., Mitchell M. (1995) "*The Evolution of Emergent Computation*", Proceedings of the National Academy of Sciences, USA 92:23 10742-10746.

Coveney P., Highfield R.. (1995) "*Frontiers of Complexity: Search for Order in a Chaotic World*", Faber and Faber.

Davies P. (2003) "*Towards an Emergentist Worldview*", in From Complexity to Life, Oxford University Press

Feigenbaum M. (1978) "*Quantitative Universality for a class of nonlinear transformations*", Journal of Statistical Physics (19) pp 25-52

Forrest S. (1990) "*Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks.*", Physica D (42: 1-3) pp. 1-11

Ganguly N., et al. (2003) "*A Survey on Cellular Automata*", Technical Report Centre for High Performance Computing, Dresden University of Technology

Gershenson C. (2004) "*Introduction to Random Boolean Networks*", in Workshop and Tutorial Proceedings 9th Int Conf. on the Simulation and Synthesis of Living Systems (ALife IX) pp. 160-173

Gisiger T. (2001) "*Scale invariance in biology: coincidence or footprint of a universal mechanism?*" Biol Rev Camb Philos Soc 76, pp. 161-209

Guy R.K., Conway J.H. (1985) "*Mathematical People: Profiles and Interviews*", Cambridge MA, pp. 43-50.

Hanson J.E., Cruthfield J.P. (1994) "*The attractor-basin portrait of a cellular automaton*"

Holland J.H. (1998) "*Emergence: From Chaos to Order*", Oxford University Press

James S. (2005) "*Looking for Life: Finding Complexity in the CA Landscape*", Unpublished, University of Sussex

Johnson S. (2001) "*Emergence*", Penguin Press

Kauffman S. (1993) "*The Origins of Order*", Oxford University Press

Kubik A. (2003) "*Towards* a Formalization of Emergence", Artificial Life (9) pp. 41-65

Langton C.G. (1989), "*Artificial Life*", In Artificial Life. Addison-Wesley. pp. 1-47

Langton C.G. (1990) "*Computation at the edge of chaos: Phase transitions and emergent computation*", Physica D vol. 42, pp. 12-37.

Lorenz E. (1963) "*Deterministic nonperiodic flow*", Journal of the Atmospheric Sciences (20) pp 130-141

Lovelock J. (1979) "*Gaia: A new look at life on Earth*", Oxford University Press

Lewin R. (1993) "*Complexity: Life at the Edge of Chaos*", Phoenix Press

Mandelbrot B. (1977) "*The Fractal Geometry of Nature*", New York: Freeman

Milotti E. (2002), "*1/f noise: a pedagogical review*", arxiv preprint, physics/0204033

Mitchell M., Crutchfield J.P., Das R. (1997) "*Evolving Cellular Automata to Perform Computations*", working paper, SFI

Mitchell M., Hraber P.T., Crutchfield J.P. (1993) "*Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations*", Complex Systems 7, pp. 89-130.

Mitchell M. (1998) "*A Complex-Systems Perspective on the "Computation vs. Dynamics" Debate in Cognitive Science",* Twentieth Annual Conference of the Cognitive Science Society

Morowitz H. (2002) "T*he Emergence of Everything: how the world became complex*", Oxford University Press

Ninagawa S. (1998) "*1/f fluctuation in the Game of Life*", Physica D vol. 118, pp. 49-52

Ninagawa S. (2005) "*Evolving Cellular Automata by 1/f Noise*", proceedings of ECAL 2005

Packard N.H. (1988) "*Adaptation toward the edge of chaos*", Dynamic Patterns in Complex Systems, pp. 293-301

Prigogine I. (1967) "*Thermodynamics of irreversible process*", New York: John Wiley and Sons

Prigogine I., Stengers I. (1981) "*Order out of chaos: Man's new dialogue with nature*", New York: Bantam

Schneider E.D., Sagan D. (2005) "*Into the cool: Energy flow, thermodynamics and life*", University of Chicago Press

Sipper M (1996) "Co-evolving Non-uniform Cellular Automata to perform Computations", Physica D vol. 92, pp. 193-208

Van Bertalanffy L. (1969) "*General System Theory: Foundations, Development, Applications*", New York: George Braziller

Von Neumann J. (1966) "*The theory of self reproducing automata*", Univ. of Illinois Press

Van Gulick R. (2001) "*Reduction, Emergence and Other Recent Options on the Mind/Body Problem: A Philosophic Overview*", Journal of Consciousness Studies (8: 9-10) pp. 1-34

Waldrop M. (1992) "Complexity: The emerging science at the edge of order and chaos", Simon & Schuster

West B.J., Shlesinger M.F. (1990), "*The noise in Natural Phenomena*", American Scientist, 78:40-45

Wolfram S. (1984) "*Universality and complexity in cellular automata*", Physica D vol. 10

Wolfram S. (2002) "*A New Kind of Science*", Wolfram Media

Wuensche A (1996) "*Attractor Basins of Discrete Networks*", unpublished D.Phil thesis, University of Sussex

APPENDIX A: CA RULETABLES

The CA ruletables are defined using the following binary numbering scheme.

Moore neighbourhood numbering scheme

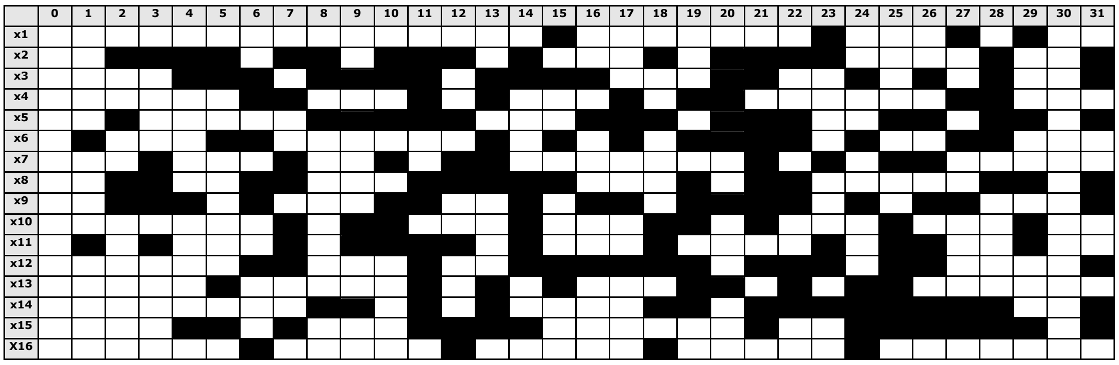
32	2	64
16	1	4
256	8	128

Examine rule = 354

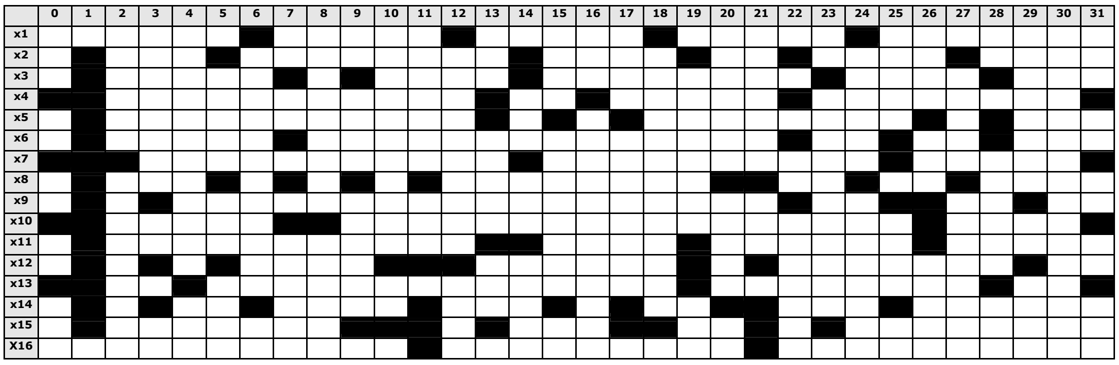
32	2	64
256		

(**Figure A.1**) Binary numbering scheme for CA ruletables

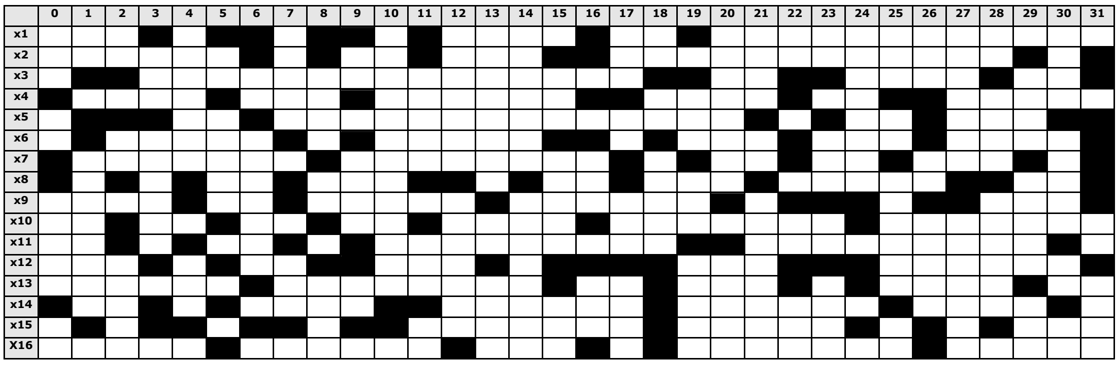
RULE 1: ORDERED: WITH A SLOW DECAY



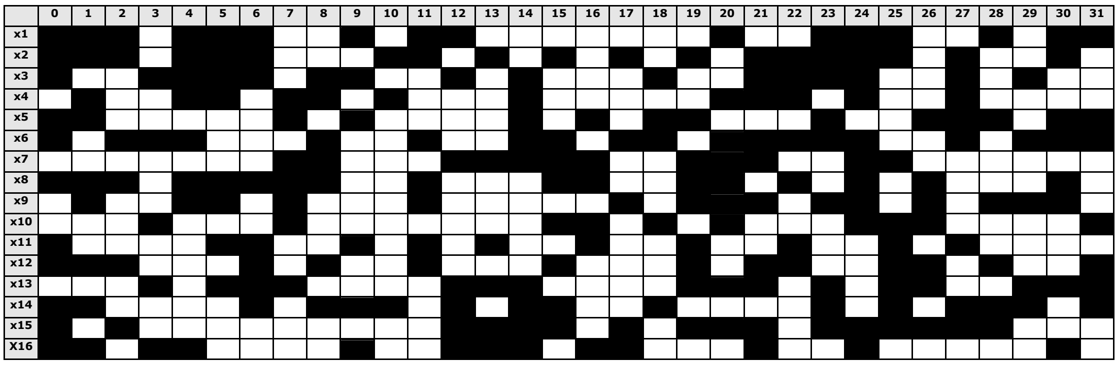
RULE 2: ORDERED: WITH A QUICK DECAY



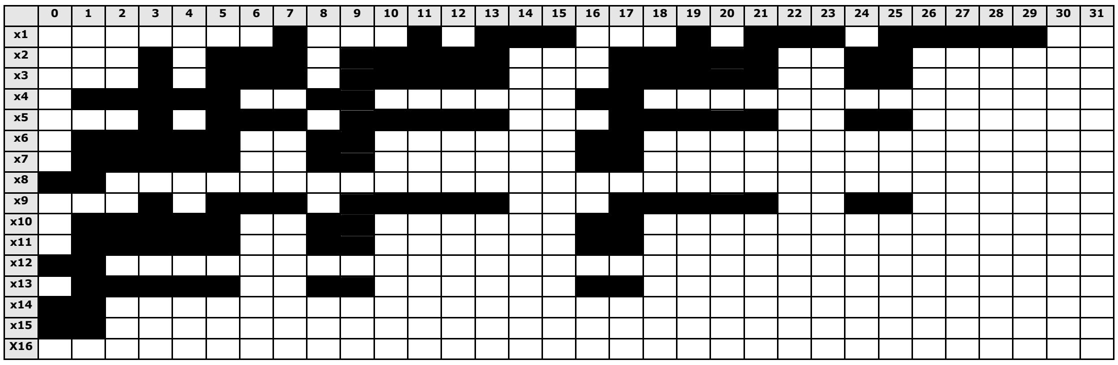
RULE 3: CHAOTIC: WITH A LOW Λ



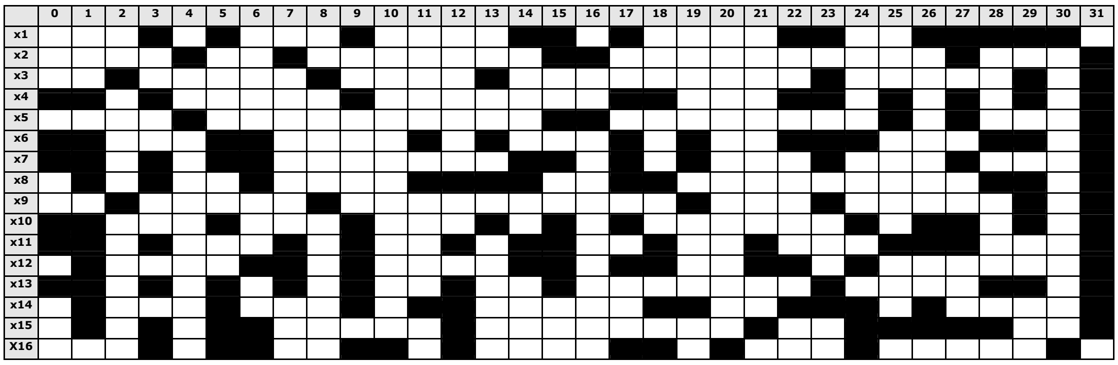
<u>Rule 4: CHAOTIC: with a high Λ</u>



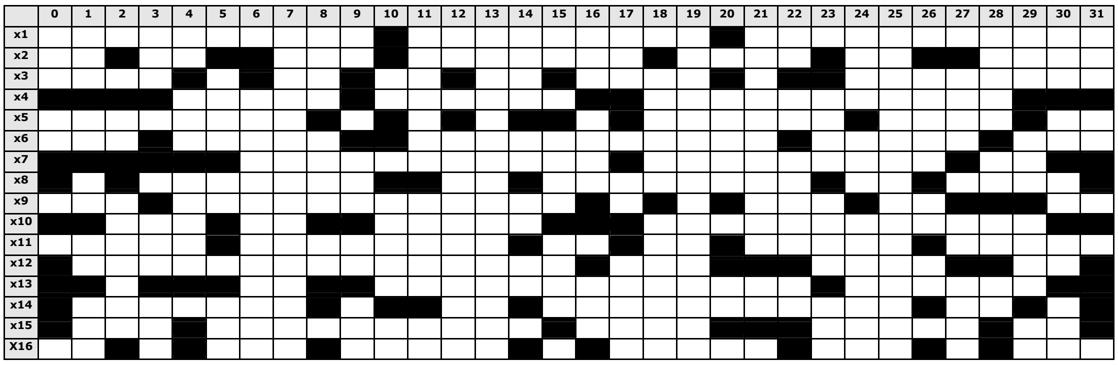
<u>Rule 5: COMPLEX: Game of Life</u>



<u>Rule 6: COMPLEX: Replicating Glider Guns</u>

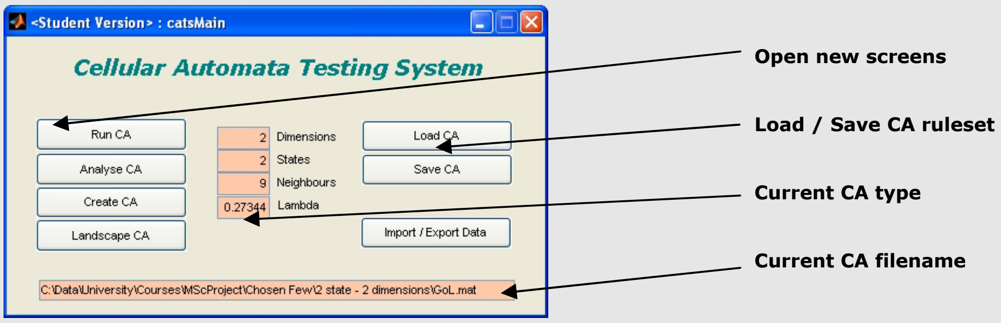


<u>Rule 7: COMPLEX: Moving Diagonal Lines</u>



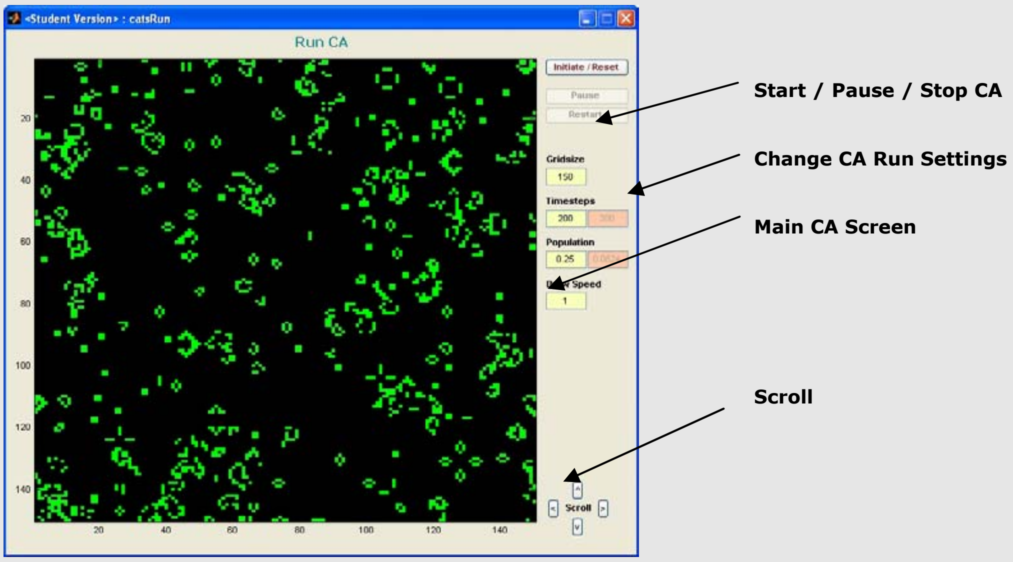
APPENDIX B: CATS SCREEN SHOTS

CATSMAIN



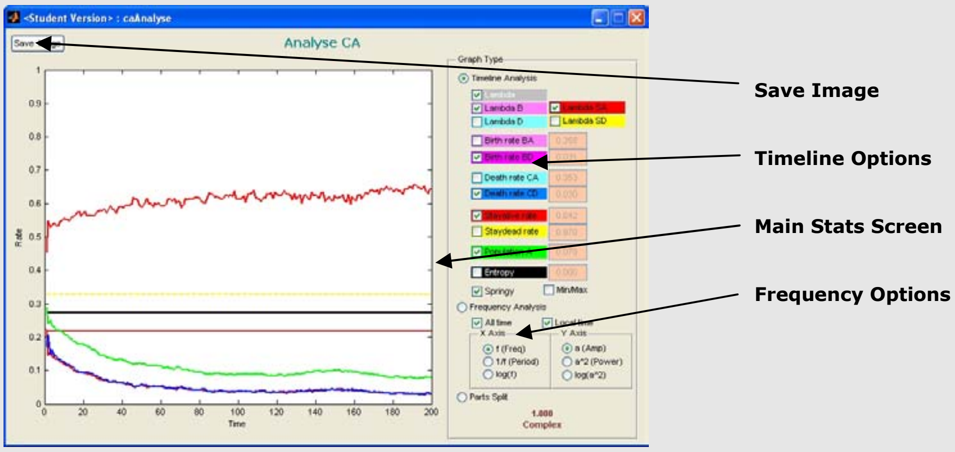
(**Figure B.1**) catsMAIN screen

CATSRUN



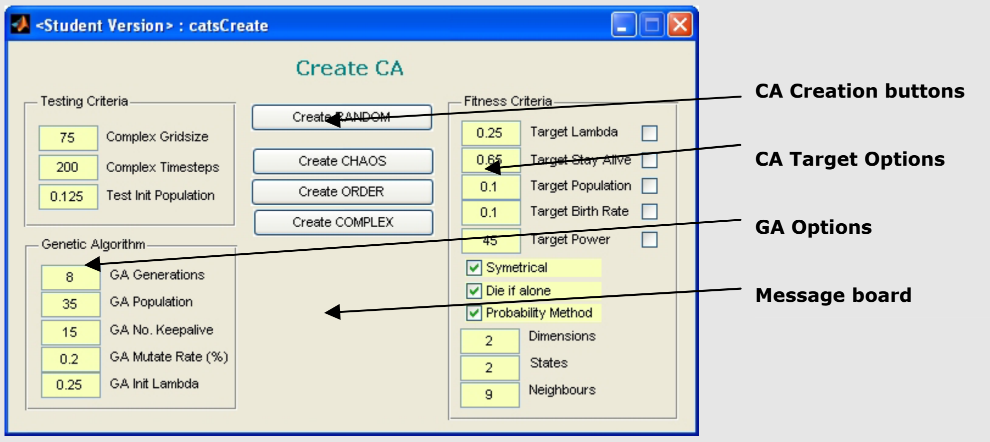
(**Figure B.2**) catsMAIN screen

CATSANALYSE



(**Figure B.3**) catsMAIN screen

<u>CATSCREATE</u>



Testing Criteria

75	Complex Gridsize
200	Complex Timesteps
0.125	Test Init Population

Genetic Algorithm

8	GA Generations
35	GA Population
15	GA No. Keepalive
0.2	GA Mutate Rate (%)
0.25	GA Init Lambda

Fitness Criteria

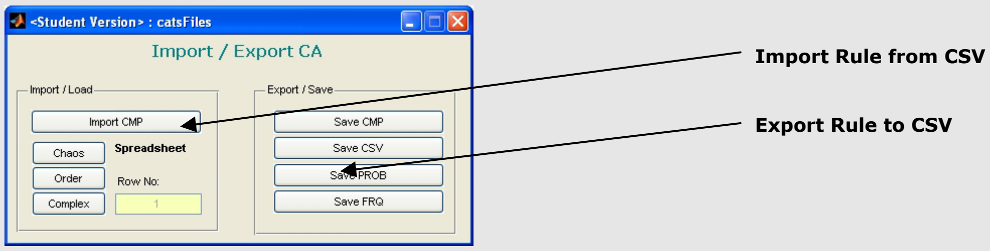
0.25	Target Lambda	☐
0.65	Target Stay Alive	☐
0.1	Target Population	☐
0.1	Target Birth Rate	☐
45	Target Power	☐

✓ Symetrical
✓ Die if alone
✓ Probability Method

2	Dimensions
2	States
9	Neighbours

(**Figure B.4**) catsMAIN screen

<u>CATSFILE</u>



Row No: 1

(**Figure B.5**) catsMAIN screen

APPENDIX C: SOURCE CODE

The CATS Project is split into four main elements.

First, the main function sets up two core global classes. One for the CA and one for its STATS. This function also opens the catsMAIN screen.

Next, there is the source code which underlies all of the screens (shown in Appendix B). This code captures all user interaction requests and changes the properties within the global classes accordingly.

Finally there are the two main classes and their methods.

classCA – contains all properties and methods relevant for the setup and running of cellular automata.

classStats – contains all of the properties and methods required for the setup and running of Population Dynamic statistical information

MAIN FUNCTION

```
function[] = cats()
   clear;

   global fld;
   global theCA; global pauseca; global stats;
   global figMain; global figRun; global figCreate; global figAnalyse; global figFiles; global figLandscape;

   fld = 'C:\Data\University\Courses\MScProject\';
   initfile = 'Chosen Few\2 state - 2 dimensions\GoL.mat';

   theCA = classCA;
   stats = classStats;
   theCA = classCALoad(theCA, cat(2,fld,initfile));

   figMain     = openfig('catsMain.fig');
   catsMain('setupscreen');
end
```

SCREEN – CATSMAIN

```
function varargout = catsMain(varargin)
   gui_Singleton = 1;
   gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @catsMain_OpeningFcn, ...
                  'gui_OutputFcn',  @catsMain_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
   if nargin && ischar(varargin{1})
      if (strcmp(varargin{1},'setupscreen') == 1)
         setupscreen();
      else
         gui_State.gui_Callback = str2func(varargin{1});
      end
   end

   if nargout
      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
   else
      gui_mainfcn(gui_State, varargin{:});
   end
end

function catsMain_OpeningFcn(hObject, eventdata, handles, varargin)
   handles.output = hObject;  guidata(hObject, handles);
end

function varargout = catsMain_OutputFcn(hObject, eventdata, handles)
   varargout{1} = handles.output;
end

function cmdrunca_Callback(hObject, eventdata, handles)
   global figRun; figRun = openfig('catsRun.fig'); catsRun('setupscreen');
end

function cmdimportca_Callback(hObject, eventdata, handles)
   global figFiles; figFiles = openfig('catsFiles.fig');
end

function cmdanalyseca_Callback(hObject, eventdata, handles)
   global figAnalyse; figAnalyse = openfig('catsAnalyse.fig'); catsAnalyse('setupscreen');
end

function cmdcreateca_Callback(hObject, eventdata, handles)
   global figCreate; figCreate = openfig('catsCreate.fig');
end

function cmdlandscapeca_Callback(hObject, eventdata, handles)
   global figLandscape; figLandscape = openfig('catsLandscape.fig');
end

function cmdloadca_Callback(hObject, eventdata, handles)
   global theCA; global fld;
   global figMain; handles = guihandles(figMain);

   a = actxcontrol('MSComDlg.CommonDialog.1');
   a.InitDir = fld; a.Filename = ''; a.Filter = 'mat'; a.ShowOpen;
   theCA = classCALoad(theCA, a.Filename);
   setupscreen(); release(a);
end

function cmdsaveca_Callback(hObject, eventdata, handles)
   global theCA; global fld; global figMain; handles = guihandles(figMain);

   a = actxcontrol('MSComDlg.CommonDialog.1');
   a.InitDir = fld; a.Filename = ''; a.Filter = 'mat'; a.ShowSave;
   theCA = classCASave(theCA, cat(2,regexprep(a.Filename,'.mat',''),'.mat'));
   setupscreen(); release(a);
end

function setupscreen()
   global theCA;  global figMain; handles = guihandles(figMain);

   set(handles.filename,'String',theCA.filename);
   set(handles.nodim,'String',num2str(theCA.nodim));
   set(handles.nostates,'String',num2str(theCA.nostates));
   set(handles.noneighbours,'String',num2str(theCA.noneighbours));
   set(handles.lambda,'String',num2str(theCA.lambda));
end
```

SCREEN - CATSRUN

```
function varargout = catsRun(varargin)
   gui_Singleton = 1;
   gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @catsRun_OpeningFcn, ...
                  'gui_OutputFcn',  @catsRun_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
   if nargin && ischar(varargin{1})
      if (strcmp(varargin{1},'setupscreen') == 1)
         setupscreen();
      else
         gui_State.gui_Callback = str2func(varargin{1});
      end
   end
   if nargout
      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
   else
      gui_mainfcn(gui_State, varargin{:});
   end
end

function catsRun_OpeningFcn(hObject, eventdata, handles, varargin)
   handles.output = hObject;
   guidata(hObject, handles);
end

function varargout = catsRun_OutputFcn(hObject, eventdata, handles)
   varargout{1} = handles.output;
end

function cmdrestartca_Callback(hObject, eventdata, handles)
   global theCA; global pauseca;
   global figRun; handles = guihandles(figRun);

   set(handles.cmdrestartca,'Enable','off');
   set(handles.gridsize,'Enable','off');
   set(handles.initpopulation,'Enable','off');
   set(handles.cmdstopca,'Enable','on');
   set(handles.cmdresetca,'Enable','off');

   pauseca = 0;
   theCA = classCARun(theCA,0,0);

   if (theCA.currenttimestep >= theCA.maxt)
      set(handles.cmdresetca,'Enable','on');
      set(handles.gridsize,'Enable','on');
      set(handles.initpopulation,'Enable','on');
      set(handles.cmdstopca,'Enable','off');
      set(handles.cmdrestartca,'Enable','off');
   end
end

function cmdstopca_Callback(hObject, eventdata, handles)
   global theCA; global pauseca;
   global figRun; handles = guihandles(figRun);

   set(handles.cmdrestartca,'Enable','on');
   set(handles.cmdstopca,'Enable','off');
   set(handles.cmdresetca,'Enable','on');
   set(handles.gridsize,'Enable','on');
   set(handles.initpopulation,'Enable','on');

   pauseca = 1;
end

function cmdresetca_Callback(hObject, eventdata, handles)
   global theCA; global pauseca; global stats;
   global figRun; handles = guihandles(figRun);

   set(handles.cmdrestartca,'Enable','off');
   set(handles.gridsize,'Enable','off');
   set(handles.initpopulation,'Enable','off');
   set(handles.cmdstopca,'Enable','on');
   set(handles.cmdresetca,'Enable','off');

   theCA.gridsize         = str2num(get(handles.gridsize,'String'));
   theCA.maxt             = str2num(get(handles.timesteps,'String'));
   theCA.drawspeed        = str2num(get(handles.drawspeed,'String'));
   theCA.initpopulation   = str2num(get(handles.initpopulation,'String'));
```

```
    pauseca = 0;
    stats = classStatsReset(stats);
    theCA = classCASetDim(theCA);
    theCA = classCASetRun(theCA);
    theCA = classCARun(theCA,0,0);

    if (theCA.currenttimestep >= theCA.maxt)
        set(handles.cmdresetca,'Enable','on');
        set(handles.gridsize,'Enable','on');
        set(handles.initpopulation,'Enable','on');
        set(handles.cmdstopca,'Enable','off');
        set(handles.cmdrestartca,'Enable','off');
    end
end

function timesteps_Callback(hObject, eventdata, handles)
    global theCA;
    global figRun; handles = guihandles(figRun);

    theCA.maxt = str2num(get(handles.timesteps,'String'));
end

function initpopulation_Callback(hObject, eventdata, handles)
    global theCA;
    global figRun; handles = guihandles(figRun);

    theCA.initpopulation = str2num(get(handles.initpopulation,'String'));
end

function drawspeed_Callback(hObject, eventdata, handles)
    global theCA;
    global figRun; handles = guihandles(figRun);

    theCA.drawspeed = str2num(get(handles.drawspeed,'String'));
end

function setupscreen()
    global theCA;
    global figRun; handles = guihandles(figRun);

    set(handles.gridsize,'String',theCA.gridsize);
    set(handles.timesteps,'String',theCA.maxt);
    set(handles.currenttimestep,'String',theCA.currenttimestep);
    set(handles.initpopulation,'String',theCA.initpopulation);
    set(handles.currentpopulation,'String',theCA.currentalives);
    set(handles.drawspeed,'String',theCA.drawspeed);
end

function scrollright_Callback(hObject, eventdata, handles)
    global theCA; global pauseca; global stats; global figRun;

    if (pauseca == 0) cmdstopca_Callback(hObject, eventdata, handles); end
    for x = theCA.gridsize:-1:2
        tmpCells(:,x) = theCA.cells(:,x-1);
    end
    tmpCells(:,1) = theCA.cells(:,theCA.gridsize);

    theCA.cells = tmpCells;
    imgr        = reshape(stats.colmap(theCA.cells+1,1),theCA.griddim);
    imgg        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
    imgb        = reshape(stats.colmap(theCA.cells+1,3),theCA.griddim);

    drawhandles      =      guihandles(figRun);      axes(get(figRun,'CurrentAxes'));      IMH      =
image(cat(3,imgr,imgg,imgb));
end

function scrollleft_Callback(hObject, eventdata, handles)
    global theCA; global pauseca; global stats; global figRun;

    if (pauseca == 0) cmdstopca_Callback(hObject, eventdata, handles); end
    for x = 1:1:theCA.gridsize-1
        tmpCells(:,x) = theCA.cells(:,x+1);
    end
    tmpCells(:,theCA.gridsize) = theCA.cells(:,1);

    theCA.cells = tmpCells;
    imgr        = reshape(stats.colmap(theCA.cells+1,1),theCA.griddim);
    imgg        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
    imgb        = reshape(stats.colmap(theCA.cells+1,3),theCA.griddim);

    drawhandles      =      guihandles(figRun);      axes(get(figRun,'CurrentAxes'));      IMH      =
image(cat(3,imgr,imgg,imgb));
end

function scrolldown_Callback(hObject, eventdata, handles)
    global theCA; global pauseca; global stats; global figRun;

    if (pauseca == 0) cmdstopca_Callback(hObject, eventdata, handles); end
    for y = theCA.gridsize:-1:2
        tmpCells(y,:) = theCA.cells(y-1,:);
    end
    tmpCells(1,:) = theCA.cells(theCA.gridsize,:);

    theCA.cells = tmpCells;
    imgr        = reshape(stats.colmap(theCA.cells+1,1),theCA.griddim);
    imgg        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
    imgb        = reshape(stats.colmap(theCA.cells+1,3),theCA.griddim);

    drawhandles      =      guihandles(figRun);      axes(get(figRun,'CurrentAxes'));      IMH      =
image(cat(3,imgr,imgg,imgb));
end

function scrollup_Callback(hObject, eventdata, handles)
    global theCA; global pauseca; global stats; global figRun;

    if (pauseca == 0) cmdstopca_Callback(hObject, eventdata, handles); end
    for y = 1:1:theCA.gridsize-1
        tmpCells(y,:) = theCA.cells(y+1,:);
    end
    tmpCells(theCA.gridsize,:) = theCA.cells(1,:);

    theCA.cells = tmpCells;
    imgr        = reshape(stats.colmap(theCA.cells+1,1),theCA.griddim);
    imgg        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
    imgb        = reshape(stats.colmap(theCA.cells+1,3),theCA.griddim);

    drawhandles      =      guihandles(figRun);      axes(get(figRun,'CurrentAxes'));      IMH      =
image(cat(3,imgr,imgg,imgb));
end

function gridsize_Callback(hObject, eventdata, handles)
    global theCA; global pauseca; global stats;
    global figRun; handles = guihandles(figRun);

    if (pauseca == 0) cmdstopca_Callback(hObject, eventdata, handles); end

    tmpCA = theCA;
    tmpCA.gridsize = str2num(get(handles.gridsize,'String'));
    tmpCA = classCASetDim(tmpCA);

    tmpCA.cells = zeros(tmpCA.griddim);
    if (tmpCA.nodim == 2)
        if (tmpCA.gridsize > theCA.gridsize)
            gridblock = ceil((tmpCA.gridsize-theCA.gridsize)/2);
            tmpCA.cells(gridblock:1:gridblock+theCA.gridsize-1,gridblock:1:gridblock+theCA.gridsize-1)      =
theCA.cells;
        else
            gridblock = ceil((theCA.gridsize-tmpCA.gridsize)/2);
            tmpCA.cells          =          theCA.cells(gridblock:1:gridblock+tmpCA.gridsize-
1,gridblock:1:gridblock+tmpCA.gridsize-1);
        end
    end

    theCA = tmpCA;
    if (theCA.nodim == 2)
        imgr        = reshape(stats.colmap(theCA.cells+1,1),theCA.griddim);
        imgg        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
        imgb        = reshape(stats.colmap(theCA.cells+1,3),theCA.griddim);
        drawhandles      =      guihandles(figRun);      axes(get(figRun,'CurrentAxes'));      IMH      =
image(cat(3,imgr,imgg,imgb));
    end
end
```

Screen – catsAnalyse

```
function varargout = catsAnalyse(varargin)
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @caAnalyse_OpeningFcn, ...
                       'gui_OutputFcn',  @caAnalyse_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        if (strcmp(varargin{1},'setupscreen') == 1)
            setupscreen();
        else
            gui_State.gui_Callback = str2func(varargin{1});
        end
    end

    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end
end

function caAnalyse_OpeningFcn(hObject, eventdata, handles, varargin)
    handles.output = hObject;
    guidata(hObject, handles);
end

function varargout = caAnalyse_OutputFcn(hObject, eventdata, handles)
    varargout{1} = handles.output;
end

function showMinMax_Callback(hObject, eventdata, handles)
    global figAnalyse; handles = guihandles(figAnalyse);
    global stats; stats.showMinMax = get(handles.showMinMax,'Value');
end

function showRates_Callback(hObject, eventdata, handles)
    global figAnalyse; handles = guihandles(figAnalyse);
    global stats;

    stats.showRates = 1; stats.showParts = 0; stats.showFreq = 0;
    set(handles.showRates,'Value',1);

    stats.showL = get(handles.showL,'Value');
    stats.showLB = get(handles.showLB,'Value');
    stats.showLD = get(handles.showLD,'Value');
    stats.showLSA = get(handles.showLSA,'Value');
    stats.showLSD = get(handles.showLSD,'Value');

    stats.showP = get(handles.showP,'Value');
    stats.showBA = get(handles.showBA,'Value');
    stats.showBD = get(handles.showBD,'Value');
    stats.showCA = get(handles.showCA,'Value');
    stats.showCD = get(handles.showCD,'Value');
    stats.showSA = get(handles.showSA,'Value');
    stats.showSD = get(handles.showSD,'Value');
    stats.showE = get(handles.showE,'Value');;

    stats.showSpringy = get(handles.showSpringy,'Value');
end

function showParts_Callback(hObject, eventdata, handles)
    global figAnalyse; handles = guihandles(figAnalyse);
    global stats; stats.showRates = 0; stats.showParts = 1; stats.showFreq = 0;
    set(handles.showParts,'Value',1);
end

function showFreq_Callback(hObject, eventdata, handles)
    global figAnalyse; handles = guihandles(figAnalyse);
    global stats; stats.showRates = 0; stats.showParts = 0; stats.showFreq = 1;
    set(handles.showFreq,'Value',1);
    stats.showFull = get(handles.showFull,'Value');
    stats.showRecent = get(handles.showRecent,'Value');
    stats.showXFreq = get(handles.showXFreq,'Value');
    stats.showXLogFreq = get(handles.showXLogFreq,'Value');
    stats.showXPeriod = get(handles.showXPeriod,'Value');
    stats.showYAmp = get(handles.showYAmp,'Value');
    stats.showYPower = get(handles.showYPower,'Value');
    stats.showYLogPower = get(handles.showYLogPower,'Value');
end

function setupscreen()
    global stats;
    global figAnalyse; handles = guihandles(figAnalyse);

    set(handles.showL,'Value',stats.showL);
    set(handles.showLB,'Value',stats.showLB);
    set(handles.showLD,'Value',stats.showLD);
    set(handles.showLSA,'Value',stats.showLSA);
    set(handles.showLSD,'Value',stats.showLSD);

    set(handles.showP,'Value',stats.showP);
    set(handles.showBA,'Value',stats.showBA);
    set(handles.showBD,'Value',stats.showBD);
    set(handles.showCA,'Value',stats.showCA);
    set(handles.showCD,'Value',stats.showCD);
```

```
   set(handles.showSA,'Value',stats.showSA);
   set(handles.showSD,'Value',stats.showSD);
   set(handles.showE,'Value',stats.showE);

   set(handles.showMinMax,'Value',stats.showMinMax);
   set(handles.showSpringy,'Value',stats.showSpringy);
end

function saveimage_Callback(hObject, eventdata, handles)
   global fld; global figAnalyse; handles = guihandles(figAnalyse);

   a = actxcontrol('MSComDlg.CommonDialog.1',[0,600,5,5]);
   a.InitDir = fld;
   a.Filename = '';
   a.Filter = 'bmp';
   a.ShowSave;
   thefilename = cat(2,regexprep(a.Filename,'.bmp',''),'.bmp');
   release(a);

   calcaxes = get(figAnalyse,'CurrentAxes');
   F = getframe(figAnalyse,[5,5,555,485]);
   [X, Map] = frame2im(F);
   imwrite(X, thefilename, 'bmp');
end
```

SCREEN – CATSCREATE

```
function varargout = catsCreate(varargin)
   gui_Singleton = 1;
   gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @catsCreate_OpeningFcn, ...
                  'gui_OutputFcn',  @catsCreate_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
   if nargin && ischar(varargin{1})
      gui_State.gui_Callback = str2func(varargin{1});
   end

   if nargout
      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
   else
      gui_mainfcn(gui_State, varargin{:});
   end
end

function catsCreate_OpeningFcn(hObject, eventdata, handles, varargin)
   handles.output = hObject;
   guidata(hObject, handles);
end

function varargout = catsCreate_OutputFcn(hObject, eventdata, handles)
   varargout{1} = handles.output;
end

function cmdcreaterandom_Callback(hObject, eventdata, handles)
   global figCreate; handles = guihandles(figCreate);
   createCAByGA(handles,'',0)
end

function cmdcreatecomplex_Callback(hObject, eventdata, handles)
   global figCreate; handles = guihandles(figCreate);

   probmethod = get(handles.probmethod,'Value');
   if probmethod == 1
      createCAByGA(handles,'Complex',1)
   else
      createCAByGA(handles,'Complex',0)
   end
end

function cmdcreatechaos_Callback(hObject, eventdata, handles)
   global figCreate; handles = guihandles(figCreate);

   probmethod = get(handles.probmethod,'Value');
   if probmethod == 1
      createCAByGA(handles,'Chaotic',1)
   else
      createCAByGA(handles,'Chaotic',0)
   end
end
```

```
function cmdcreateorder_Callback(hObject, eventdata, handles)
   global figCreate; handles = guihandles(figCreate);

   probmethod = get(handles.probmethod,'Value');
   if probmethod == 1
      createCAByGA(handles,'Ordered',1)
   else
      createCAByGA(handles,'Ordered',0)
   end
end

%-------------------------
% CREATE CA
%-------------------------
function createCAByGA(handles,whichSort,probmethod)
   % Define Constants
   %-----------------
   global fld; global theCA; global pauseca; global stats;
   global figMain; global figAnalyse;

   createdim            = str2num(get(handles.nodims,'String'));
   createsta            = str2num(get(handles.nostates,'String'));
   createnei            = str2num(get(handles.noneighbours,'String'));
   nngenerations        = str2num(get(handles.nngenerations,'String'));
   nngenerations        = str2num(get(handles.nngenerations,'String'));
   nninitlambda         = str2num(get(handles.nninitlambda,'String'));
   nnpopulation         = str2num(get(handles.nnpopulation,'String'));
   nnkeepalive          = str2num(get(handles.nnkeepalive,'String'));
   nnmutate             = str2num(get(handles.nnmutate,'String'));
   targetonlambda       = (get(handles.targetonlambda,'Value'));
   targetonpower        = (get(handles.targetonpower,'Value'));
   targetonbirthrate    = (get(handles.targetonbirthrate,'Value'));
   targetonpopulation   = (get(handles.targetonpopulation,'Value'));
   targetonsa           = (get(handles.targetonsa,'Value'));
   targetlambda         = str2num(get(handles.targetlambda,'String'));
   targetpower          = str2num(get(handles.targetpower,'String'));
   targetbirthrate      = str2num(get(handles.targetbirthrate,'String'));
   targetpopulation     = str2num(get(handles.targetpopulation,'String'));
   targetsa             = str2num(get(handles.targetsa,'String'));
   testgridsize         = str2num(get(handles.testgridsize,'String'));
   testtimesteps        = str2num(get(handles.testtimesteps,'String'));
   testinitpopulation   = str2num(get(handles.testinitpopulation,'String'));

   dieifalone           = get(handles.dieifalone,'Value');
   symetrical           = get(handles.symetrical,'Value');

   for popcounter = 1:1:nnpopulation
      eval(cat(2,'popCA',num2str(popcounter),'                                        =
classCA(',num2str(createdim),',',num2str(createsta),',',num2str(createnei),');'));
   end

   % Create CA
   %-----------------
   isfound = 0; gencounter = 0; changecount = 0; classcount = 0;
   bestca = 0; bestscore = -100;

   % Initialise Rulesets
   %----------------------
   for popcounter = 1:1:nnpopulation
      eval(cat(2,'tmpCA = popCA',num2str(popcounter),';'));
      tmpCA.rules = zeros(1,tmpCA.norules);
      if (nninitlambda == 1)
         tmplambda = rand() * 0.5;
      else
         tmplambda = nninitlambda;
      end

      if (symetrical == 1)
         nosym = 4;
         if (tmpCA.nodim == 3) nosym = 7; end
         for j = 1:1:ceil(((tmpCA.norules*tmplambda)/nosym))
            tmprule = ceil(rand() * (tmpCA.norules-1));
            while(tmpCA.rules(1,tmprule) ~= 0) tmprule = ceil(rand() * (tmpCA.norules-1)); end
            tmpCA.rules(1,tmprule) = ceil(rand()*(theCA.nostates-1));
            tmpCA.rules(1,tmpCA.staterot(tmprule)) = tmpCA.rules(1,tmprule);
            tmpCA.rules(1,tmpCA.staterot(tmpCA.staterot(tmprule))) = tmpCA.rules(1,tmprule);
            tmpCA.rules(1,tmpCA.staterot(tmpCA.staterot(tmpCA.staterot(tmprule))))          =
tmpCA.rules(1,tmprule);
            if(tmpCA.nodim == 3)
               tmpCA.rules(1,tmpCA.staterotz(tmprule)) = tmpCA.rules(1,tmprule);
               tmpCA.rules(1,tmpCA.staterotz(tmpCA.staterotz(tmprule))) = tmpCA.rules(1,tmprule);
               tmpCA.rules(1,tmpCA.staterotz(tmpCA.staterotz(tmpCA.staterotz(tmprule))))    =
tmpCA.rules(1,tmprule);
            end
         end
```

```
      else
         for j = 1:1:ceil(tmpCA.norules*tmplambda)
            tmprule = ceil(rand() * (tmpCA.norules-1));
            while(tmpCA.rules(1,tmprule) ~= 0) tmprule = ceil(rand() * (tmpCA.norules-1)); end
            tmpCA.rules(1,tmprule) = ceil(rand()*(theCA.nostates-1));
         end
      end
      if (dieifalone == 1) tmpCA.rules(1,1:tmpCA.nostates) = 0; end
      tmpCA = classCASetValues(tmpCA);
      eval(cat(2,'popCA',num2str(popcounter),' = tmpCA;'));
   end

   if isempty(whichSort)
      isfound = 1;
      theCA = popCA1;
      if ishandle(figMain) catsMain('setupscreen'); end
   end

   % For Each GENERATION
   %--------------------
   while (isfound == 0) && (gencounter < nngenerations)
      gencounter = gencounter + 1;

      % Test Rulesets
      %----------------------
      popcounter = 0; classcount = 0;
      while (isfound == 0) && (popcounter < nnpopulation)
         set(handles.createmsg,'String',cat(2,'creating                                      -
',num2str(gencounter,'%3.0f'),':',num2str(nngenerations),'
',num2str(popcounter,'%3.0f'),':',num2str(nnpopulation))); drawnow;
         popcounter = popcounter + 1;
         eval(cat(2,'tmpCA = popCA',num2str(popcounter),';'));

         % Test CA by Running it
         %----------------------
         if probmethod == 0
            pauseca = 0;
            if (tmpCA.nodim == 2)
               tmpCA.gridsize        = 50;
               tmpCA.maxt            = 60;
            else
               tmpCA.gridsize        = 30;
               tmpCA.maxt            = 60;
            end
            tmpCA.initpopulation    = testinitpopulation;

            stats = classStatsReset(stats);
            tmpCA = classCAInitCells(tmpCA);
            tmpCA = classCASetRun(tmpCA);
            tmpCA = classCARun(tmpCA,1,1);
            output = [tmpCA.class]
            tmpCA.score = tmpCA.cscore^2;

            if (tmpCA.class == whichSort)
               isfound = 1; classcount = classcount + 1; tmpCA.score = 0;
               if (strcmp(whichSort,'Complex')==1)
                  tmpCA.gridsize        = testgridsize;
                  tmpCA.maxt            = testtimesteps;
                  tmpCA.initpopulation    = testinitpopulation;

                  pauseca = 0;
                  stats = classStatsReset(stats);
                  tmpCA = classCASetDim(tmpCA);
                  tmpCA = classCASetRun(tmpCA);
                  tmpCA = classCARun(tmpCA,1,1);
                  tmpCA.score = tmpCA.cscore + 1;
                  if (strcmp(tmpCA.class,'Complex') == 0) isfound = 0;  end
               end
            end
         end

         if ((targetonlambda == 1) && (abs(tmpCA.lambda-targetlambda)/targetlambda > 0.1)) isfound = 0;
end
         if    ((targetonsa    ==    1)    &&    (abs(mean(stats.SA(ceil(tmpCA.maxt/2):1:tmpCA.maxt))-
targetsa)/targetsa > 0.1)) isfound = 0; end
         if ((targetonbirthrate == 1) && (abs(tmpCA.lambdab-targetbirthrate)/targetbirthrate > 0.1)) isfound
= 0; end
         if     ((targetonpopulation     ==     1)     &&     (targetpopulation     <     1)     &&
(abs(mean(stats.P((ceil(tmpCA.maxt/4)*2):1:(ceil(tmpCA.maxt/4)*3)))-targetpopulation)/targetpopulation >
0.1)) isfound = 0; end
         if     ((targetonpopulation     ==     1)     &&     (targetpopulation     <     1)     &&
(abs(mean(stats.P((ceil(tmpCA.maxt/4)*3):1:(ceil(tmpCA.maxt/4)*4)))-targetpopulation)/targetpopulation >
0.1)) isfound = 0; end
```

```
      if ((targetonpopulation == 1) && (targetpopulation == 1) &&
(abs(mean(stats.P((ceil(tmpCA.maxt/4)*2):1:(ceil(tmpCA.maxt/4)*3)))-
mean(stats.P((ceil(tmpCA.maxt/4)*3):1:(ceil(tmpCA.maxt/4)*4)))) > 0.2)) isfound = 0; end
      if ((targetonpower == 1) && (abs(tmpCA.maxpower-targetpower)/targetpower > 0.1)) isfound = 0;
end

      if (targetonlambda == 1)          tmpCA.score = tmpCA.score - (abs(tmpCA.lambda-
targetlambda)/targetlambda); end
      if (targetonsa == 1)                  tmpCA.score = tmpCA.score -
(abs(mean(stats.SA(ceil(tmpCA.maxt/2):1:tmpCA.maxt))-targetsa)/targetsa); end
      if (targetonbirthrate == 1)        tmpCA.score = tmpCA.score - (abs(tmpCA.lambdab-
targetbirthrate)/targetonbirthrate); end
      if (targetonpopulation == 1) && (targetpopulation < 1)  tmpCA.score = tmpCA.score -
(abs(mean(stats.P((ceil(tmpCA.maxt/4)*2):1:(ceil(tmpCA.maxt/4)*3)))-targetpopulation)/targetpopulation);
end
      if (targetonpopulation == 1) && (targetpopulation < 1)  tmpCA.score = tmpCA.score -
(abs(mean(stats.P((ceil(tmpCA.maxt/4)*3):1:(ceil(tmpCA.maxt/4)*4)))-targetpopulation)/targetpopulation);
end
      if (targetonpopulation == 1) && (targetpopulation == 1) tmpCA.score = tmpCA.score -
(abs(mean(stats.P((ceil(tmpCA.maxt/4)*2):1:(ceil(tmpCA.maxt/4)*3)))-
mean(stats.P((ceil(tmpCA.maxt/4)*3):1:(ceil(tmpCA.maxt/4)*4))))); end
      if (targetonpower == 1)          tmpCA.score = tmpCA.score - (abs(tmpCA.maxpower -
targetpower)/targetpower); end

      if (tmpCA.score > bestscore) || (isfound == 1)
         bestscore = tmpCA.score; bestca = popcounter;
         set(handles.currentmsg,'String',cat(2,'Gen:',num2str(gencounter),',
Pop:',num2str(popcounter,'%1.0f\n'),...
               'Score: ',num2str(bestscore,'%1.2f\n'),...
               'CScore: ',num2str(tmpCA.cscore,'%1.2f\n'), ...
               'MaxPower: ',num2str(tmpCA.maxpower,'%1.2f\n'),...
               'Lamda: ',num2str(tmpCA.lambda,'%1.2f\n'),...
               'Lamda B: ',num2str(tmpCA.lambdab,'%1.2f\n'),...
               'Class: ',tmpCA.class)); drawnow;
         theCA = tmpCA;
         if ishandle(figMain) catsMain('setupscreen'); end
      end
      eval(cat(2,'popCA',num2str(popcounter),' = tmpCA;'));
   end
   %----------------------
   % Genetically Mutate Rulesets
   %---------------------------
   popcounter = 0; changecount = 0;
   while (isfound == 0) && (popcounter < nnpopulation)
      popcounter = popcounter + 1; mate = popcounter;
      while (mate == popcounter) mate = floor(rand()*nnpopulation)+1; end
      eval(cat(2,'tmpCA = popCA',num2str(popcounter),';'));
      eval(cat(2,'mateCA = popCA',num2str(mate),';'));

      if ((tmpCA.score < mateCA.score) && (popcounter ~= bestca))
         tmpCA.rules = mateCA.rules;
         changecount = changecount + 1;
         % --- SYMMETRICAL ---
         if (symetrical == 1)
            nosym = 4;
            if (tmpCA.nodim == 3) nosym = 7; end
            if (nnmutate >= 1)
               noofmutations = nnmutate;
            else
               muterate = rand * nnmutate;
               noofmutations = ceil((tmpCA.norules*muterate)/nosym);
            end
            for j = 1:1:noofmutations
               tmprule = ceil(rand() * (tmpCA.norules-1));
               if (targetonbirthrate == 1) && (abs(targetbirthrate - tmpCA.lambdab) > 0.01)
                  if (targetbirthrate > tmpCA.lambdab)
                     while(tmpCA.rules(1,tmprule) == 1) || (rand() < 0.25) tmprule = ceil(rand() *
(tmpCA.norules-1)); end
                  else
                     while(tmpCA.rules(1,tmprule) == 0) || (rand() < 0.25) tmprule = ceil(rand() *
(tmpCA.norules-1)); end
                  end
               elseif (targetlambda == 1) && (abs(targetlambda - tmpCA.lambda) > 0.01)
                  if (targetlambda > tmpCA.lambda)
                     while(tmpCA.rules(1,tmprule) == 1) || (rand() < 0.25) tmprule = ceil(rand() *
(tmpCA.norules-1)); end
                  else
                     while(tmpCA.rules(1,tmprule) == 0) || (rand() < 0.25) tmprule = ceil(rand() *
(tmpCA.norules-1)); end
                  end
               end
               tmpCA.rules(1,tmprule) = abs(1-tmpCA.rules(1,tmprule));
               tmpCA.rules(1,tmpCA.staterot(tmprule)) = tmpCA.rules(tmprule);
               tmpCA.rules(1,tmpCA.staterot(tmpCA.staterot(tmprule))) = tmpCA.rules(1,tmprule);
               tmpCA.rules(1,tmpCA.staterot(tmpCA.staterot(tmpCA.staterot(tmprule))))          =
tmpCA.rules(1,tmprule);
               if(tmpCA.nodim == 3)
                  tmpCA.rules(1,tmpCA.staterotz(tmprule)) = tmpCA.rules(tmprule);
                  tmpCA.rules(1,tmpCA.staterotz(tmpCA.staterotz(tmprule))) = tmpCA.rules(tmprule);
                  tmpCA.rules(1,tmpCA.staterotz(tmpCA.staterotz(tmpCA.staterotz(tmprule))))     =
tmpCA.rules(tmprule);
               end
               tmpCA = classCASetLambda(tmpCA);
            end
         else
            % --- NORMAL ---
            for j=1:1:tmpCA.norules
               if(rand()<nnmutate) tmpCA.rules(1,j) = abs(1-tmpCA.rules(1,j)); end
            end
         end
         % --- DIE IF ALONE ---
         if (dieifalone == 1) tmpCA.rules(1,1:tmpCA.nostates) = 0; end
         tmpCA = classCASetLambda(tmpCA);
         eval(cat(2,'popCA',num2str(popcounter),' = tmpCA;'));
      end
   end
   %----------------------
end

if (isfound == 1)
   set(handles.createmsg,'String','Successful'); drawnow;
else
   set(handles.createmsg,'String','Failed'); drawnow;
end
end

function[y] = bitcount(x,size)
   y = bitand(x,1)/1 + ...
      bitand(x,2)/2 + ...
      bitand(x,4)/4 + ...
      bitand(x,8)/8 + ...
      bitand(x,16)/16;
   if (size == 9)
      y = y + ...
         bitand(x,32)/32 + ...
         bitand(x,64)/64 + ...
         bitand(x,128)/128 + ...
         bitand(x,256)/256 + ...
         bitand(x,512)/512;
   end
end
```

Screen – catsFile

```
function varargout = catsFiles(varargin)
   gui_Singleton = 1;
   gui_State = struct('gui_Name',       mfilename, ...
                 'gui_Singleton',  gui_Singleton, ...
                 'gui_OpeningFcn', @catsFiles_OpeningFcn, ...
                 'gui_OutputFcn',  @catsFiles_OutputFcn, ...
                 'gui_LayoutFcn',  [] , ...
                 'gui_Callback',   []);
   if nargin && ischar(varargin{1})
      gui_State.gui_Callback = str2func(varargin{1});
   end

   if nargout
      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
   else
      gui_mainfcn(gui_State, varargin{:});
   end
end

function catsFiles_OpeningFcn(hObject, eventdata, handles, varargin)
   handles.output = hObject;
   guidata(hObject, handles);
end

function varargout = catsFiles_OutputFcn(hObject, eventdata, handles)
   varargout{1} = handles.output;
end

function cmdimportcmp_Callback(hObject, eventdata, handles)
   global theCA; global fld; global figMain;

   a = actxcontrol('MSComDlg.CommonDialog.1');
   a.InitDir = fld;
   a.Filename = '';
   a.Filter = 'csv';
   a.ShowOpen;
   filename = cat(2,regexprep(a.Filename,'.csv',''),'.csv');
   release(a);

   fid = fopen(filename,'r');
   i = 1;
   while (feof(fid) == 0)
      line = fgetl(fid);
      [a rem] = strtok(line,',');
      [b rem] = strtok(rem,',');
      theCA.rules(1,i) =  str2num(a);
      theCA.rules(1,i+1) = str2num(b);
      i = i + 2;
   end
   fclose(fid);
   theCA = classCASetValues(theCA);
   theCA = classCASetRun(theCA);
   if ishandle(figMain) catsMain('setupscreen'); end
end

function loadrowno_CreateFcn(hObject, eventdata, handles)
   if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
      set(hObject,'BackgroundColor','white');
   end
end

function cmdsavecmp_Callback(hObject, eventdata, handles)
   global theCA; global fld;

   a = actxcontrol('MSComDlg.CommonDialog.1');
   a.InitDir = fld;
   a.Filename = '';
   a.Filter = 'csv';
   a.ShowSave;
   filename = cat(2,regexprep(a.Filename,'.csv',''),'.csv');
   release(a);

   fid = fopen(filename,'w');
   for i = 1:2:size(theCA.rules,2)
      fprintf(fid,'%1d,',theCA.rules(1,i));
      fprintf(fid,'%1d\n',theCA.rules(1,i+1));
   end
   fclose(fid);
end
```

Class CA

```
function[theCA] = classCA(nodim, nostates, noneighbours)
   if exist('nodim')
      theCA.nodim              = nodim;
      theCA.nostates           = nostates;
      theCA.noneighbours         = noneighbours;
   else
      theCA.nodim              = 2;
      theCA.nostates           = 2;
      theCA.noneighbours         = 9;
   end
   theCA.norules             = theCA.nostates ^ theCA.noneighbours;
   theCA.symettrical          = 1;
   theCA.dieifalone           = 1;
   theCA.rules              = 1;
   theCA.cells              = [];
   theCA.gridsize            = 50;
   theCA.griddim              = [];
   theCA.gridsplit           = 5;
   theCA.nocells             = 0;
   theCA.maxt                = 100;
   theCA.initpopulation        = 0.25;
   theCA.drawspeed            = 1;

   theCA.lambda               = 0;           % Lambda
   theCA.lambdab               = 0;           % Lambda Birth
   theCA.lambdad               = 0;           % Lambda Death

   theCA.name                 = 'Unnamed';
   theCA.filename              = '';
   theCA.class              = 'Unknown';
   theCA.score               = 0;
   theCA.cscore               = 0;
   theCA.ccount              = 0;
   theCA.slope              = 0;
   theCA.maxpower              = 0;
   theCA.staterot             = [];
   theCA.staterotz            = [];
   if (theCA.noneighbours == 9)
      theCA.symcells = [1 3 7 11  15  31  33  35  37  39  41  43  45  47  49  51  53  55  57  59  61  63  97
99  101 103 105 107 109 111 113 115 117 119 121 123 125 127 161 163 165 167 169 171 173 175 177
179 181 183 185 187 189 191 225 227 229 231 233 235 237 239 241 243 245 247 249 251 253 255 481
483 487 491 495 511];
   else
      theCA.symcells = [1 3 7 15 31];
   end

   theCA = classCASetDim(theCA);
   theCA = classCASetRot(theCA);
   theCA = classCASetRun(theCA);
end
```

Class CA – Run

```
function[theCA] = catsRunCA(theCA,forcenodraw,forcecalc)
   global figRun; global figAnalyse; global pauseca; global stats; global notes;

   warning off all

   if (theCA.nodim == 3)
      splitsize = theCA.gridsize;
      x              = 1:splitsize;
      xplus          = [2:splitsize 1];
      xplus2          = [3:splitsize 1 2];
      xminus          = [splitsize 1:splitsize-1];
      xminus2          = [splitsize-1 splitsize 1:splitsize-2];
      y = x;  yplus = xplus;  yplus2 = xplus2; yminus = xminus; yminus2 = xminus2;
      z = x;  zplus = xplus;  zplus2 = xplus2; zminus = xminus; zminus2 = xminus2;
   else
      splitsize = theCA.gridsize;
      x              = 1:splitsize;
      xplus          = [2:splitsize 1];
      xplus2          = [3:splitsize 1 2];
      xminus          = [splitsize 1:splitsize-1];
      xminus2          = [splitsize-1 splitsize 1:splitsize-2];
      y = x;  yplus = xplus;  yplus2 = xplus2; yminus = xminus; yminus2 = xminus2;
      z = x;  zplus = xplus;  zplus2 = xplus2; zminus = xminus; zminus2 = xminus2;
   end
   cellages       = theCA.cells;
   ages          = 0;
   nolives        = 0;
   tmp3           = 0;
   localsize = 40;

   drawRun        = ishandle(figRun);
   drawAnalyse     = ishandle(figAnalyse);
   calcit          = drawAnalyse;
   if (forcenodraw > 0) drawRun = 0; drawAnalyse = 0; end
   if (forcecalc > 0) calcit = forcecalc; end

   if (drawRun == 1)
      drawhandles = guihandles(figRun); axes(get(figRun,'CurrentAxes'));
      if (theCA.nodim == 2)
         imgr           = reshape(stats.colmap(theCA.cells+1,1),theCA.griddim);
         imgg           = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
         imgb           = reshape(stats.colmap(theCA.cells+1,3),theCA.griddim);
         IMH = image(cat(3,imgr,imgg,imgb));
      elseif (theCA.nodim == 3)
         runaxes = get(figRun,'CurrentAxes'); cla(runaxes);
         axis(runaxes, [0 theCA.gridsize+1 0 theCA.gridsize+1 0 theCA.gridsize+1]);
         colour = summer(theCA.gridsize);
         FVb.vertices = []; FVb.faces = [];
         %cubeface = [1 2 3 4; 4 3 7 8; 1 5 8 4; 2 6 7 3; 1 2 6 5; 5 6 7 8];
         cubeface = [1 2 3 4; 4 3 7 8; 1 5 8 4; 2 6 7 3; 1 2 6 5; 5 6 7 8];
         for ypos = 1:theCA.gridsize
            for xpos = 1:theCA.gridsize
               xs=[xpos,xpos+1];
               ys=[ypos,ypos+1];
               FVb.vertices=[FVb.vertices; [xs([1 2 2 1 1 2 2 1]) ; ys([1 1 2 2 1 1 2 2]) ; [1 1 1 1 2 2 2 2]]'];
            end
         end

         pah = [];
         for zpos = 1:theCA.gridsize
            FV(zpos).vertices = FVb.vertices;
            FV(zpos).vertices(:,3) = FV(zpos).vertices(:,3) + (zpos-1);
            FV(zpos).facecolor = colour(zpos,:);
            pah(zpos) = patch(FV(zpos));
         end
      end
   end
   if (drawAnalyse == 1) calchandles = guihandles(figAnalyse); calcaxes = get(figAnalyse,'CurrentAxes'); end
   if (calcit == 1)
      theCA.class = 'Unknown';
   end

   %-------
   % RUN CA
   %-------
   i = theCA.currenttimestep;
   while i <= theCA.maxt
      if (ishandle(figAnalyse) == 0) drawAnalyse = 0; end
      if(pauseca == 1)
         theCA.currenttimestep = i;
         i = theCA.maxt+1;
      else
         if (calcit > 0) fullstates = ones(theCA.griddim); end

         % CALCULATE NEXT CELLS
         %---------------------
         oldcells = theCA.cells;
         if (theCA.noneighbours == 9)
            states(x,y)   = 1 + ...
                        ((theCA.nostates^0) * theCA.cells(x,y)) + ...
                        ((theCA.nostates^1) * theCA.cells(xplus,y)) + ...
                        ((theCA.nostates^2) * theCA.cells(x,yplus)) + ...
                        ((theCA.nostates^3) * theCA.cells(xminus,y)) + ...
                        ((theCA.nostates^4) * theCA.cells(x,yminus)) + ...
                        ((theCA.nostates^5) * theCA.cells(xplus,yminus)) + ...
                        ((theCA.nostates^6) * theCA.cells(xplus,yplus)) + ...
                        ((theCA.nostates^7) * theCA.cells(xminus,yplus)) + ...
                        ((theCA.nostates^8) * theCA.cells(xminus,yminus));

            theCA.cells(x,y)  = theCA.rules(states(x,y));
            if (calcit > 0) fullstates(x,y) = states(x,y); end
         elseif (theCA.noneighbours == 5)
            states(x,y)   = 1 + ...
                        ((theCA.nostates^0) * theCA.cells(x,y)) + ...
                        ((theCA.nostates^1) * theCA.cells(xplus,y)) + ...
                        ((theCA.nostates^2) * theCA.cells(x,yplus)) + ...
                        ((theCA.nostates^3) * theCA.cells(xminus,y)) + ...
                        ((theCA.nostates^4) * theCA.cells(x,yminus));

            theCA.cells(x,y)  = theCA.rules(states(x,y));
            if (calcit > 0) fullstates(x,y) = states(x,y); end
         elseif (theCA.noneighbours == 7)
            if (i > 1)
               states(x,y,z)   = 1 + ...
                           ((theCA.nostates^0) * theCA.cells(x,y,z)) + ...
                           ((theCA.nostates^1) * theCA.cells(xplus,y,z)) + ...
                           ((theCA.nostates^2) * theCA.cells(x,yplus,z)) + ...
                           ((theCA.nostates^3) * theCA.cells(xminus,y,z)) + ...
                           ((theCA.nostates^4) * theCA.cells(x,yminus,z)) + ...
                           ((theCA.nostates^5) * theCA.cells(x,y,zminus)) + ...
                           ((theCA.nostates^6) * theCA.cells(x,y,zplus));

               theCA.cells(x,y,z)  = theCA.rules(states(x,y,z));
               if (calcit > 0) fullstates(x,y,z) = states(x,y,z); end
            end
         end

         % CALCULATE STATS
         %----------------
         stats.A(i) = sum(sum(sum(theCA.cells~=0)));
         if (calcit > 0)
            stats.P(i) = stats.A(i) / theCA.nocells;
            if (calcit == 1)
               stats.D(i)  = theCA.nocells - stats.A(i);
               stats.B(i)  = sum(sum(sum(oldcells==0 & theCA.cells~=0)));
               stats.C(i)  = sum(sum(sum(oldcells~=0 & theCA.cells==0)));
               stats.BA(i) = stats.B(i) / stats.A(i);
               if (i > 1) stats.BD(i) = stats.B(i) / stats.D(i-1); else stats.BD(i) = 0; end
               if (i > 1) stats.CA(i) = stats.C(i) / stats.A(i-1); else stats.CA(i) = 0; end
               %stats.BD(i) = stats.B(i) / stats.D(i);
               %stats.CD(i) = stats.C(i) / stats.D(i);
               stats.BD(i) = stats.B(i) / theCA.nocells;
               stats.CD(i) = stats.C(i) / theCA.nocells;
               stats.SA(i) = sum(sum(sum(oldcells~=0 & theCA.cells~=0))) / stats.A(i);
               stats.SD(i) = sum(sum(sum(oldcells==0 & theCA.cells==0))) / stats.D(i);

               % CALCULATE ENTROPY
               S = 0;
               if (stats.showE == 1)
                  Qn = sum(hist(fullstates,theCA.norules)') / theCA.nocells;
                  tQn = Qn.*log(Qn);
                  tQn(isnan(tQn)) = 0;
                  S = 0 - sum(tQn);
               end
               stats.E(i)   = S;

               % CALCULATE FULL SLOPE
               Y1 = []; Y = []; N = 0; N1 = 0; slope = 0; ydata = 0; xdata = 0; y1data = 0; x1data = 0;
d1data = 0; theCA.slope = 0;
               if (i > 1)
                  Y = fft(stats.P(1:1:i),i); N = length(Y); Y(1) = []; nyquist = 1/2;
                  ampl = abs(Y(1:N/2));
                  power = ampl.^2;
                  freq = (1:N/2)/(N/2)*nyquist;
                  period = 1 ./ freq;
                  theCA.maxpower = max(power);
                  freq1 = freq; ampl1 = ampl; power1 = power;  period1 = period;
               end
            end

            % CALCULATE CLASS
            if (i > localsize)
               % CALCULATE LOCAL SLOPE
               Y1 = fft(stats.P((i-localsize):1:i)); N1 = length(Y1); Y1(1) = []; nyquist1 = 1/2;
               ampl1 = abs(Y1(1:N1/2));
               power1 = ampl1.^2;
               freq1 = (1:N1/2)/(N1/2)*nyquist1;
               period1 = 1 ./ freq1;
               p = polyfit(period1,power1,1); d1data = polyval(p,period1);
               theCA.slope = ((d1data(1) - d1data(2))/(period1(1) - period1(2)))*1000;

               if (theCA.nodim == 2) isOrder = (stats.A(i) / theCA.nocells) < (theCA.lambda / 2); end
               if (theCA.nodim == 3) isOrder = ((stats.A(i)) / theCA.nocells) < (theCA.lambda / 2); end
               isChaos = (sum(d1data) > 0.0001);
               if (isChaos == 1) theCA.class = 'Chaotic'; end
               if (isOrder == 1) theCA.class = 'Ordered'; end
               %if (isOrder == 1) && (isChaos == 1) theCA.class = 'Complex'; theCA.ccount = theCA.ccount
+ 1; end
               if (isOrder == 1) && (isChaos == 1) && (theCA.slope > 0.01) theCA.class = 'Complex';
theCA.cscore = i / theCA.maxt; end
               %theCA.cscore = theCA.ccount / (i-localsize);
            end
         end

         % DRAW STATS
         %-----------
         if (drawAnalyse > 0)
            if (theCA.drawspeed <= 1) || (mod(i,theCA.drawspeed) == 0)

               set(calchandles.txtclass,'String',cat(2,num2str(theCA.cscore,'%1.3f\n'),theCA.class));

               % DRAW FREQ GRAPHS
               %----------------

               if (stats.showFreq == 1)
                  cla(calcaxes); set(calcaxes,'NextPlot','add');
                  if (i > 5)
                     if (stats.showXFreq == 1) xlabel(calcaxes,'Frequency'); xdata = freq; x1data = freq1; end
                     if (stats.showXLogFreq == 1) xlabel(calcaxes,'Log(Frequency)'); xdata = log(freq);
x1data = log(freq1); end
                     if (stats.showXPeriod == 1) xlabel(calcaxes,'Period (1/Frequency)'); xdata = period;
x1data = period1; end
                     if (stats.showYAmp == 1) ylabel(calcaxes,'Amplitude'); ydata = ampl; y1data = ampl1;
end
                     if (stats.showYPower == 1) ylabel(calcaxes,'Power'); ydata = power; y1data = power1;
end
                     if (stats.showYLogPower == 1) ylabel(calcaxes,'Log(Power)'); ydata = log(power); y1data
= log(power1); end
                     p = polyfit(xdata,ydata,1); ddata = polyval(p,xdata);
                     p = polyfit(x1data,y1data,1); d1data = polyval(p,x1data);

                     minx = 0; maxx = 0; miny = 0; maxy = 0;
                     if (stats.showFull == 1) minx = min(xdata); maxx = max(xdata); miny = min(ydata);
maxy = max(ydata); end;
                     if (stats.showRecent == 1) && (min(x1data) < minx) minx = min(x1data); end
                     if (stats.showRecent == 1) &&(max(x1data) > maxx) maxx = max(x1data); end
                     if (stats.showRecent == 1) &&(min(y1data) < miny) miny = min(y1data); end
                     if (stats.showRecent == 1) &&(max(y1data) > maxy) maxy = max(y1data); end
                     if (stats.showXLogFreq == 1) && (stats.showYLogPower == 1) minx = -7.5; maxx = 2.5;
miny = -25; maxy = 5; end
                     if (stats.showXLogFreq == 1) && (stats.showYPower == 1) miny = 0; maxy = 0.1; end
                     if (stats.showXPeriod == 1) && (stats.showYAmp == 1) minx = 0; miny = 0; maxy =
ceil(i/50); end
                     if (stats.showXPeriod == 1) && (stats.showYAmp == 1) && (stats.showFull == 0) maxy =
0.1; end
                     if (stats.showXPeriod == 1) && (stats.showYPower == 1) minx = 0; miny = 0; maxy =
ceil(i/10); end
                     if (stats.showXPeriod == 1) && (stats.showYPower == 1) && (stats.showFull == 0) maxy
= 0.01; end
                     if (stats.showXFreq == 1) minx = 0; maxx = 0.5; miny = 0; maxy = 1; end
                     if (stats.showXFreq == 1) && (stats.showFull == 0) maxy = 0.1; end

                     if (stats.showYLogPower == 1) plot(calcaxes,[minx-5 maxx+1],[maxx+1 minx-5],'b--');
end;
                     if (stats.showYLogPower == 1) plot(calcaxes,[minx+1 maxx+1],[maxx+1+((maxx-
minx)/2) minx+1],'b--'); end;
                     if (stats.showFull == 1) plot(calcaxes,xdata,ydata,'g-','LineWidth',2);
plot(calcaxes,xdata,ddata,'r-','LineWidth',2); end
                     if (stats.showRecent == 1) plot(calcaxes,x1data,y1data,'c-','LineWidth',2);
plot(calcaxes,x1data,d1data,'m-','LineWidth',2); end
                     axis(calcaxes,'normal'); axis(calcaxes,[minx maxx miny maxy]);
                  end

               % DRAW PARTS TIME DIAGRAM
               elseif (stats.showParts == 1)
                  xlabel(calcaxes,'Part'); ylabel(calcaxes,'Time');
                  cla(calcaxes); set(calcaxes,'NextPlot','replace');
                  if (i == 1)
                     axis(calcaxes,'normal'); axis(calcaxes,[1 theCA.nocells 0 theCA.maxt]);
                     prtsmap     = ones(theCA.maxt,theCA.nocells);
                     axes(calcaxes);
                     PRTS = image(cat(3,prtsmap,prtsmap,prtsmap));
                     drawnow;
                  end
                  for j = 0:1:theCA.gridsize-1
                     for k = 1:1:theCA.gridsize
                        prtsmap(i,(j*theCA.gridsize)+k) = 1-theCA.cells(j+1,k);
                     end
                  end
                  set(PRTS,'CData',cat(3,prtsmap,prtsmap,prtsmap));

               % DRAW RATES STATISTICS
               elseif (stats.showRates == 1)
                  set(calchandles.txtba,'String',num2str(stats.BA(i),'%1.3f'));
                  set(calchandles.txtbd,'String',num2str(stats.BD(i),'%1.3f'));
                  set(calchandles.txtca,'String',num2str(stats.CA(i),'%1.3f'));
                  set(calchandles.txtcd,'String',num2str(stats.CD(i),'%1.3f'));
                  set(calchandles.txtsa,'String',num2str(stats.SA(i),'%1.3f'));
                  set(calchandles.txtsd,'String',num2str(stats.SD(i),'%1.3f'));
                  set(calchandles.txtp,'String',num2str(stats.P(i),'%1.3f'));
                  set(calchandles.txte,'String',num2str(stats.E(i),'%1.3f'));

                  cla(calcaxes); set(calcaxes,'NextPlot','add');
                  if (stats.showSpringy == 1) axis(calcaxes,'normal'); axis(calcaxes,[0 i 0 1]); end
                  if (stats.showSpringy == 0) axis(calcaxes,'normal'); axis(calcaxes,[0 theCA.maxt 0 1]); end
                  xlabel(calcaxes,'Time');
                  % PERCENTAGES
                  if (stats.showMinMax == 0)
                     ylabel(calcaxes,'Rate');
                     showbit = 1:1:i;
                     if (stats.showL == 1) plot(calcaxes,showbit,ones(size(showbit))*theCA.lambda,'k-
','LineWidth',2); end
                     if (stats.showLB == 1) plot(calcaxes,showbit,ones(size(showbit))*theCA.lambdab,'r-
','LineWidth',2); end
                     if (stats.showLD == 1) plot(calcaxes,showbit,ones(size(showbit))*theCA.lambdad,'b-
','LineWidth',2); end
                     if (stats.showLSA == 1) plot(calcaxes,showbit,ones(size(showbit))*(1-theCA.lambdad),'y-
-','LineWidth',2); end
                     if (stats.showLSD == 1) plot(calcaxes,showbit,ones(size(showbit))*(1-theCA.lambdab),'y-
-','LineWidth',2); end

                     if (stats.showBA == 1) plot(calcaxes,showbit,stats.BA(showbit),'m-','LineWidth',3); end
                     if (stats.showBD == 1) plot(calcaxes,showbit,stats.BD(showbit),'r-','LineWidth',3); end
                     if (stats.showCA == 1) plot(calcaxes,showbit,stats.CA(showbit),'c-','LineWidth',3); end
                     if (stats.showCD == 1) plot(calcaxes,showbit,stats.CD(showbit),'b-','LineWidth',3); end
                     if (stats.showSA == 1) plot(calcaxes,showbit,stats.SA(showbit),'r-','LineWidth',3); end
                     if (stats.showSD == 1) plot(calcaxes,showbit,stats.SD(showbit),'y-','LineWidth',3); end
                     if (stats.showE == 1) plot(calcaxes,showbit,stats.E(showbit),'k-','LineWidth',3); end
                     if (stats.showP == 1) plot(calcaxes,showbit,stats.P(showbit),'g-','LineWidth',3); end
                  % MIN MAX
                  else
                     ylabel(calcaxes,'Rate (min max)');
                     if (i>stats.statsstart)
                        showbit = stats.statsstart+1:1:i;
                        if (stats.showBA == 1) plot(calcaxes,showbit,minmax(stats.BA(showbit)),'m-
','LineWidth',2); end
                        if (stats.showBD == 1) plot(calcaxes,showbit,minmax(stats.BD(showbit)),'r-
','LineWidth',2); end
                        if (stats.showCA == 1) plot(calcaxes,showbit,minmax(stats.CA(showbit)),'c-
','LineWidth',2); end
                        if (stats.showCD == 1) plot(calcaxes,showbit,minmax(stats.CD(showbit)),'b-
','LineWidth',2); end
                        if (stats.showSA == 1) plot(calcaxes,showbit,minmax(stats.SA(showbit)),'r-
','LineWidth',2); end
                        if (stats.showSD == 1) plot(calcaxes,showbit,minmax(stats.SD(showbit)),'y-
','LineWidth',2); end
                        if (stats.showE == 1) plot(calcaxes,showbit,minmax(stats.E(showbit)),'k-
','LineWidth',2); end
                        if (stats.showP == 1) plot(calcaxes,showbit,minmax(stats.P(showbit)),'g-
','LineWidth',2); end
                     end
                  end
               end
            end
         end

         % DRAW CA
         %-----------
         if(drawRun > 0)
               if (theCA.drawspeed <= 1) || (mod(i,theCA.drawspeed) == 0)
                  if (theCA.nodim == 2)
                     imgr        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
                     imgg        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);
                     imgb        = reshape(stats.colmap(theCA.cells+1,2),theCA.griddim);

                     set(IMH,'CData',cat(3,imgr,imgg,imgb));
                  elseif(theCA.nodim == 3)
                     view(runaxes,[30+(i/4) 10+(i/2)]);
                     for zpos = 1:theCA.gridsize
                        showfaces = []; showcells = []; b = []; ix = [];
                        [b ix] = sort(reshape(theCA.cells(:,:,zpos),1,theCA.gridsize*theCA.gridsize));
                        showcells = (b.*ix);
                        showcells(showcells == 0) = [];
                        showfaces =
(repmat(sort(repmat(showcells,1,size(cubeface,1)),'descend')',1,size(cubeface,2))*8)-8 +
repmat(cubeface,size(showcells,2),1);
                        set(pah(zpos),'Faces',showfaces);
                     end
                  end
                  set(drawhandles.currenttimestep,'String',num2str(i));
                  set(drawhandles.currentpopulation,'String',num2str(stats.A(i) / theCA.nocells,'%2.4f'));
                  drawnow;
               end
               if (theCA.drawspeed <  1) pause(theCA.drawspeed); end
         end
         i = i + 1;
      end
   end
   if (pauseca == 0) theCA.currenttimestep = theCA.maxt; end
end

function[y] = minmax(x)
   y = (x - min(x)) .* 1/(max(x)-min(x));
end
```

Class CA – Init Cells

```
function[theCA] = classCAInitCells(theCA)
   theCA.cells = zeros(theCA.griddim);
   if (theCA.initpopulation < 1)
      theCA.cells       = ceil(rand(theCA.griddim) * (theCA.nostates-1));
      theCA.cells       = theCA.cells .* (rand(theCA.griddim)<theCA.initpopulation);
   else
      theCA.cells = zeros(theCA.griddim);
      mid = 0 + ceil(theCA.gridsize/2);
      x = 0 + ceil(theCA.gridsize/2);
      if (theCA.nodim == 3)
         theCA.cells(mid,mid,mid) = 1;
         theCA.cells(mid+1,mid,mid) = 1;
      else
         theCA.cells(mid,mid) = 1;
         theCA.cells(mid+1,mid) = 1;
      end
   end
end
```

Class CA – Set Values

```
function[theCA] = classCASetValues(theCA)
   theCA.norules = size(theCA.rules,2);

   if (theCA.norules == 2^9 ) theCA.nodim = 2; theCA.noneighbours = 9; theCA.nostates = 2; end
   if (theCA.norules == 3^9 ) theCA.nodim = 2; theCA.noneighbours = 9; theCA.nostates = 3; end
   if (theCA.norules == 4^9 ) theCA.nodim = 2; theCA.noneighbours = 9; theCA.nostates = 4; end
   if (theCA.norules == 5^9 ) theCA.nodim = 2; theCA.noneighbours = 9; theCA.nostates = 5; end
   if (theCA.norules == 2^7 ) theCA.nodim = 3; theCA.noneighbours = 7; theCA.nostates = 2; end
   if (theCA.norules == 2^27 ) theCA.nodim = 3; theCA.noneighbours = 27; theCA.nostates = 2; end
   if (theCA.norules == 2^16 ) theCA.nodim = 2; theCA.noneighbours = 16; theCA.nostates = 2; end

   theCA.dieifalone = 0; if (theCA.rules(1,1) == 0) theCA.dieifalone = 1; end

   theCA.staterot              = [];

   theCA = classCASetLambda(theCA);
   theCA = classCASetDim(theCA);
   theCA = classCASetRot(theCA);
end
```

Class CA – Set Dim

```
function[theCA] = classCASetDim(theCA)
   if (theCA.nodim == 1) theCA.griddim = [1]; end
   if (theCA.nodim == 2) theCA.griddim = [theCA.gridsize, theCA.gridsize]; end
   if (theCA.nodim == 3) theCA.griddim = [theCA.gridsize, theCA.gridsize, theCA.gridsize]; end

   theCA.nocells = prod(theCA.griddim);
end
```

Class CA – Set Lambda

```
function[theCA] = classCASetLambda(theCA)
   theCA.lambda          = sum(sum(theCA.rules>0))/theCA.norules;
   theCA.lambdab         = 0;
   theCA.lambdad         = 0;
   for i = 1:2:(theCA.norules-1)
      theCA.lambdab = theCA.lambdab + theCA.rules(1,i);
      theCA.lambdad = theCA.lambdad + (1-theCA.rules(1,i+1));
   end
   theCA.lambdab = theCA.lambdab / (theCA.norules/2);
   theCA.lambdad = theCA.lambdad / (theCA.norules/2);
end
```

Class CA – Set Run

```
function[theCA] = classCASetRun(theCA)
   theCA = classCAInitCells(theCA);

   theCA.currenttimestep       = 1;
   theCA.currentalives         = sum(sum(theCA.cells>0));
   theCA.currentsas            = 0;
   theCA.currentbirths         = 0;
   theCA.currentdeaths         = 0;
   theCA.currentdeathage       = 0;
   theCA.currententropy        = 0;
   theCA.score                 = 0;
   theCA.cscore                = 0;
   theCA.ccount                = 0;
   theCA.slope                 = 0;
   theCA.maxpower              = 0;
   theCA.class                 = 'Unknown';
end
```

Class CA – Set Rot

```
function[theCA] = classCASetRot(theCA)
   global fld;

   theCA.staterot        = [];
   theCA.staterotz       = [];
```

```
    for i = 1:1:ceil(theCA.norules/10000)
        minrot = (10000*(i-1))+1;
        if theCA.norules < (i*10000)
            maxrot = theCA.norules;
        else
            maxrot = i*10000;
        end
        if exist(cat(2,fld,'\System Files\','statrot-',num2str(theCA.nostates),'-',num2str(theCA.noneighbours),'-
',num2str(i),'.mat'))
            load(cat(2,fld,'\System Files\','statrot-',num2str(theCA.nostates),'-',num2str(theCA.noneighbours),'-
',num2str(i),'.mat'));
            theCA.staterot(1,minrot:1:maxrot) = staterottmp;
        else
            staterottmp = [];
            for j = minrot:1:maxrot
                k = staterotate(j, theCA.nostates, theCA.nodim, theCA.noneighbours);
                staterottmp(j-minrot+1) = k;
                output = j
            end
            save(cat(2,fld,'\System Files\','statrot-',num2str(theCA.nostates),'-',num2str(theCA.noneighbours),'-
',num2str(i),'.mat'),'staterottmp');
        end
    end

    for i = 1:1:ceil(theCA.norules/10000)
        minrot = (10000*(i-1))+1;
        if theCA.norules < (i*10000)
            maxrot = theCA.norules;
        else
            maxrot = i*10000;
        end
        if exist(cat(2,fld,'\System Files\','statrotz-',num2str(theCA.nostates),'-',num2str(theCA.noneighbours),'-
',num2str(i),'.mat'))
            load(cat(2,fld,'\System Files\','statrotz-',num2str(theCA.nostates),'-',num2str(theCA.noneighbours),'-
',num2str(i),'.mat'));
            theCA.staterotz(1,minrot:1:maxrot) = staterottmp;
        else
            staterottmp = [];
            for j = minrot:1:maxrot
                k = staterotatez(j, theCA.nostates, theCA.nodim, theCA.noneighbours);
                staterottmp(j-minrot+1) = k;
                output = j
            end
            save(cat(2,fld,'\System Files\','statrotz-',num2str(theCA.nostates),'-
',num2str(theCA.noneighbours),'-',num2str(i),'.mat'),'staterottmp');
        end
    end
end

function[y] = staterotate(x, nostates, nodim, noneighbours)

    states = zeros(1,noneighbours);
    x1     = x-1;
    for i = noneighbours-1:-1:0
        for j = nostates-1:-1:1
            if (x1 >= (nostates^i)*j) states(1,i+1) = j; x1 = x1 - ((nostates^i)*j); end
        end
    end

    if (noneighbours == 3)
        y = (states(1,1) * (nostates^0)) + ...
            (states(1,3) * (nostates^1)) + ...
            (states(1,2) * (nostates^2));
    elseif (noneighbours == 5)
        y = (states(1,1) * (nostates^0)) + ...
            (states(1,2) * (nostates^2)) + ...
            (states(1,3) * (nostates^3)) + ...
            (states(1,4) * (nostates^4)) + ...
            (states(1,5) * (nostates^1));
    elseif (noneighbours == 9)
        y = (states(1,1) * (nostates^0)) + ...
            (states(1,2) * (nostates^2)) + ...
            (states(1,3) * (nostates^3)) + ...
            (states(1,4) * (nostates^4)) + ...
            (states(1,5) * (nostates^1)) + ...
            (states(1,6) * (nostates^6)) + ...
            (states(1,7) * (nostates^7)) + ...
            (states(1,8) * (nostates^8)) + ...
            (states(1,9) * (nostates^5));
    elseif (noneighbours == 7)
        y = (states(1,1) * (nostates^0)) + ...
            (states(1,2) * (nostates^2)) + ...
            (states(1,3) * (nostates^3)) + ...
            (states(1,4) * (nostates^4)) + ...
            (states(1,5) * (nostates^1)) + ...
            (states(1,6) * (nostates^5)) + ...
            (states(1,7) * (nostates^6));
    end
    y = y + 1;
end

function[y] = staterotatez(x, nostates, nodim, noneighbours)

    states = zeros(1,noneighbours);
    x1     = x-1;
    for i = noneighbours-1:-1:0
        for j = nostates-1:-1:1
            if (x1 >= (nostates^i)*j) states(1,i+1) = j; x1 = x1 - ((nostates^i)*j); end
        end
    end
    y = x1;
    if (noneighbours == 7)
        y = (states(1,1) * (nostates^0)) + ...
            (states(1,2) * (nostates^6)) + ...
            (states(1,3) * (nostates^2)) + ...
            (states(1,4) * (nostates^5)) + ...
            (states(1,5) * (nostates^4)) + ...
            (states(1,6) * (nostates^1)) + ...
            (states(1,7) * (nostates^3));
    end
    y = y + 1;
end
```

CLASS CA - LOAD

```
function theCA = classCALoad(theCA, filename)
    theCA.filename  = filename;

    load(theCA.filename);
    theCA.rules     = bestrules;

    theCA          = classCASetValues(theCA);
    theCA          = classCASetRun(theCA);
end
```

CLASS CA - SAVE

```
function[theCA] = classCASave(theCA, filename)
    bestrules = theCA.rules;
    theCA.filename = cat(2,regexprep(filename,'.mat',''),'.mat');
    save(theCA.filename,'bestrules');
end
```

CLASS STATS - MAIN

```
function[stats] = classStats()
    stats.showRates       = 1;
    stats.showFreq        = 0;
    stats.showParts       = 0;

    stats.A              = [];
    stats.D              = [];
    stats.B              = [];
    stats.C              = [];

    stats.P              = [];
    stats.BA             = [];
    stats.BD             = [];
    stats.CA             = [];
    stats.CD             = [];
    stats.SA             = [];
    stats.SD             = [];
    stats.E              = [];
    stats.F              = [];

    stats.showL          = 1;
    stats.showLB         = 1;
    stats.showLD         = 0;
    stats.showLSA        = 1;
    stats.showLSD        = 0;

    stats.showP          = 1;
    stats.showBA         = 0;
    stats.showBD         = 1;
    stats.showCA         = 0;
    stats.showCD         = 1;
    stats.showSA         = 1;
    stats.showSD         = 0;
    stats.showE          = 0;

    stats.showMinMax      = 0;
    stats.showSpringy     = 1;

    stats.showFull        = 1;
    stats.showRecent      = 1;
    stats.showXFreq       = 1;
    stats.showXLogFreq    = 0;
    stats.showXPeriod     = 0;
    stats.showYAmp        = 1;
    stats.showYPower      = 0;
    stats.showYLogPower   = 0;

    stats.statsstart      = 10;

    stats.colmap          = [0.0 0.0 0.0;...
                    0.0 1.0 0.0;...
                    1.0 1.0 0.0;...
                    1.0 0.0 1.0;...
                    0.0 1.0 1.0;...
                    0.0 0.0 1.0;...
                    1.0 0.0 0.0;...
                    0.7 0.0 0.0;...
                    0.0 0.7 0.0;...
                    0.0 0.0 0.7;...
                    0.7 0.7 0.0;...
                    0.7 0.0 0.7;...
                    0.0 0.7 0.7];

    [stats.gauss stats.gaussix] = hist(randn(1, 10000),21);
End
```

CLASS STATS –RESET

```
function[stats] = classStatsReset(stats)
    stats.P             = [];          % Population Statistics
    stats.SA            = [];          % Stay Alive statistics
    stats.B             = [];          % Birth Statistics
    stats.D             = [];          % Death Statistics
    stats.E             = [];          % Entropy Statistics
    stats.A             = [];          % Age statistics
    stats.F             = [];          % 1/f Slope Statistics
end
```