Multi-coloured Cellular Automata

Saskia Hiltemann

August 25, 2008

Contents

1 Introduction

Cellular automata were initially developed as a model for crystal growth, and since then many more parallels with other natural phenomena have been uncovered. For example the patterns on seashells and other organisms, the shapes of leaves, and the arrangement of stems and branches in plants. Studying the properties of cellular automata may help us understand these processes better. Aside from the natural aspect, CAs can also be potentially useful in the following ways:

- Cryptography [8]
  Cellular automata can be potentially useful in the field of cryptography. In general, CAs are not reversible, that is, given a global state of the CA, it is difficult or even impossible to find the previous state. However, the designer

of the rule can create it in such a way as to be able to easily invert it. This would seem to make it a trapdoor function, which makes it suitable for use in public key cryptography.

- Computer Graphics
  Currently the possibility of using cellular automata as a tool for generating computer graphics is being investigated. Already they are used to easily and quickly create snowflake-like crystals[1], or to simulate peeling or other surface textures for 3D modelling[6].

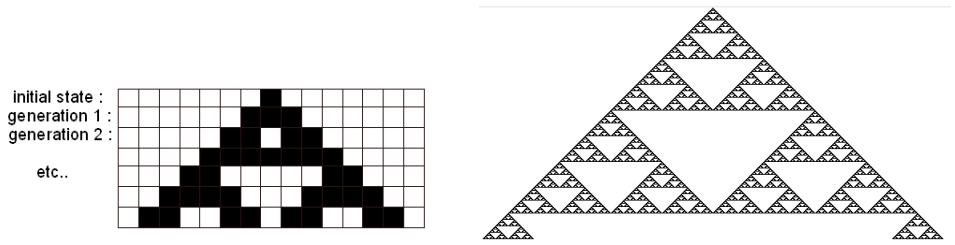
- Random number generators [9] [7]
  Due to the often chaotic nature of cellular automata, they can serve as processes for generating random numbers.

- Electronic Art [3]
  Cellular automata often yield pleasing pictures, so the question arises, could we do the same with sound as the output? Several schemes have been proposed to generate music using cellular automata, for example one could assign a note or chord to every cell or to every state of a cell.

2 What are Cellular Automata?

A cellular automaton, or CA for short, is a collection of interacting cells, spatially organized on a grid, each of which can assume a finite number of different states. At each discrete timestep t, the cells are updated according to some transition rule. This transition rule looks at the states of neighbouring cells, and depending on their values, and the state of the cell itself, determines the state the cell will assume at time $t+1$. The transition rule is applied in parallel to all cells, and each application of the rule to the entire grid leads to a new *generation* of the CA. This relatively simple model can lead to quite complex and diverse behaviour.

The simplest type of cellular automaton is the one-dimensional CA. Here the cells are arranged on a single line, and each cell looks to its left and right neighbours to decide its next state. How far left and right it looks, is determined by the *radius*. By graphically placing the one dimensional cell arrays of consecutive generations underneath each other, complex patterns can be observed. An example of this can be seen in Figure 1.



(a) First 7 generations (b) First 300 generations

Figure 1: Graphical representation of a one-dimensional CA

This principle can easily be extended to two or more dimensions. Though for the two dimensional case only the current state can be visualized on a 2D plane, not the entire history.

2.1 Transition rules

Stephen Wolfram introduced a way to represent all possible transition rules simply as numbers [10]. To illustrate this method, consider the one dimensional two-state cellular automaton:

Transition rule Θ can be represented by $R = \{b_1, b_2, .., b_{2^{2r+1}}\}$ where r is the radius, and b_i is the output of the transition rule for the mapping of the input to $2^{2r+1} - i$.
For example $R = \{0, 1, 1, 1, 1, 1, 0, 0\}$:

Neighbourhood state	111	110	101	100	011	010	001	000
Output	0	1	1	1	1	1	0	0

Now we can use the decimal notation of our bitstring R to represent this rule in Wolfram notation, so here the rule would be $01111100_2 = 124$.

For $k > 2$ possible states for each cell, this technique can also be applied, but now we do not interpret the rule as a binary number, but as a base-k number.

The number of possible rules grows explosively with increased neighbourhood size and number of possible states of each cell. Say there are n cells in the neighbourhood, and each cell can assume k different states, then the total number of different possible transition rules is: k^{k^n}. This number grows incredibly quickly with increasing values of k and n. The amount of memory required to store a rule is k^n bits.

2.2 Totalistic CAs

Introduced in 1983 by Stephen Wolfram, totalistic CAs are a special class of CA in which the transition rules depend only the total (or equivalently, the average) value of the neighbouring cells. This approach drastically reduces the total amount of different possible rules. If there are n cells which can each be in a state numbered between 0 and $k-1$ then the highest total value of the cells is $n(k-1)$, and the lowest is of course 0. Every value in between is also possible, making the total number of different cases $n(k-1)+1$ and thus the total number of possible rules becomes $k^{n(k-1)+1}$. The best known example of a totalistic CA is *Conway's Game of Life* [4] in two dimensions.

2.3 Neighbourhood Types

There are different neighbourhood types one can use for a two dimensional CA. The most commonly used are the *Moore* and *von Neumann* neighbourhoods (Figure 2). But of course, one can define any set of cells as the neighbourhood. Here we define two additional neighbourhood types, the *cross* and *plus* neighbourhoods (Figure 3).

Note that the *von Neumann* and *plus* neighbourhoods are identical for a radius of 1.

The *Moore* and *von Neumann* neighbourhoods are widely used because they are good abstractions of biological and chemical processes, as nearby cells have greater influence than cells further away.

Figure 2: The standard neighbourhoods (radius 1 and 2).

Figure 3: The custom neighbourhoods (radius 1 and 2).

Aside from choosing a neighbourhood type, one must also decide on a *radius*, which is a parameter that determines how far the neighbourhood extends.

2.4 The Cyclic CA

In the cyclic cellular automaton, first introduced by David Griffeath [2], each of the k different possible states, are assigned a number from 0 to $k-1$. At each discrete timestep, a cell of state i will change into state $(i+1)$ `mod` k, if there are at least a threshold of τ neighbouring cells that are of state $i+1$. This means that every cell will change to the next state if it reaches the necessary threshold, and a cell of state $k-1$ can change into state 0, hence the cyclic nature of these automata.

3 Research Questions

The research questions asked in order to investigate the behaviour of the two-dimensional cyclic CA are:

1. Which different classes of behaviour can be identified?
2. Which parameter settings lead to which kind of behaviour?

3. Does a specific set of parameters always yield the same class of behaviour, or can differences in the initial universe lead to different behaviour for identical parameters?

4. If a cyclic pattern has emerged, can we say anything about the (global) cycle length, and how this relates on the parameters and/or canvas size?

To answer these question, both theoretical and experimental analysis will be used.

4 Results

To answer the research questions, we can run the program for different parameter settings and observe the behaviour. The program is also able to determine the (global) cycle length of the CA when cyclic behaviour occurs, and if there is a convergence to a stable state, it will output the number of generations passed before convergence.

4.1 Theoretical results

For the *cross* neighbourhood, a theoretical analysis of the neighbourhood shape led to an interesting observation and also explains the somewhat unusual behaviour of this CA.

4.1.1 The *Cross* neighbourhood

Recall from Figure 3 that for the *cross* neighbourhood, the neighbourhood consists of the cells found diagonally in any direction. This means that there essentially are two independent interlaced cellular automata. Consider a chess board, with pieces which can only move diagonally. Pieces that start on a white square, will always remain on a white square, and will never encounter pieces that started on a black square. Similarly, for the *cross* neighbourhood, white cells will only be influenced by other white cells, and the same holds true for the black cells. Because of this, the collection of all white cells form one CA, and collection of all black cells form another, completely independent CA. In Figure 4, the two interlaced CAs can clearly be distinguished.

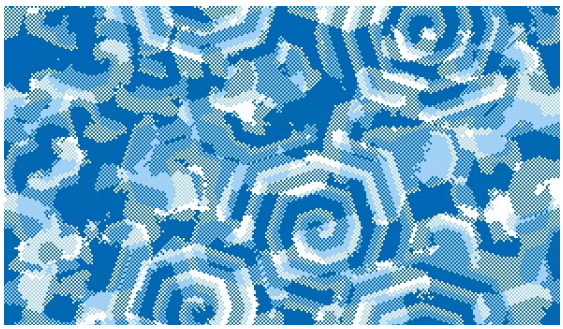
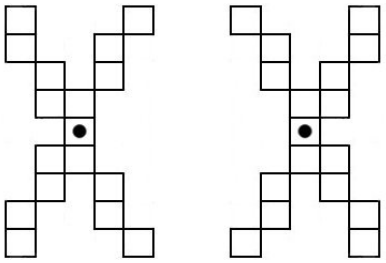


Figure 4: Two interlaced cellular automata

We can untangle the two CAs and arrange the cells back into a grid to see what the neighbourhood would look like. We do this as follows: imagine the chess board again, start by removing all black squares and squeezing all the white ones together to fill the gaps. More precisely, the ith cell of each row of the resulting CA will be the mapping of the ith *white* cell of the original. The same can of course be done for the *black* squares.



(a) odd rows (b) even rows

Figure 5: Mapping of the neighbourhood for the interlaced Cross CAs

These embedded neighbourhoods are axis symmetric but not point symmetric. Also, the neighbourhood used in these embedded CAs are different for odd and even rows. This means the CA does not have one single neighbourhood for all cells, but rather two different, vertically mirrored, neighbourhood types, one for even rows, the other for odd rows. The other embedded CA (the one consisting of all the "black" squares), has the same two neighbourhood types, but those of the even and odd rows are now interchanged.

4.2 Empirical Results

Most of the following results were obtained by simply running the program for different setting and (qualitatively) observing what happens.

4.2.1 Which types of behaviour can be identified?

Four different classes of behaviour can be identified for the two-dimensional cyclic CA:

1. **Convergence to a stable state**
   There are no more changes to the universe after a certain point.

2. **Organized behaviour**
   Emergent behaviour appears, in the form of one or more spirals. This category can be further divided as follows:

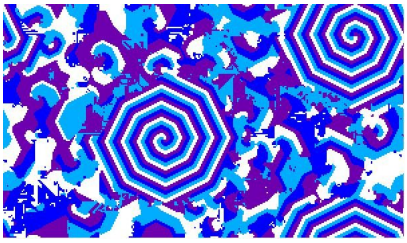
   - Diamond spirals
   - Square spirals
   - Octagonal spirals

- Circular spirals
- Non-spiral cyclic behaviour
  Here cyclic behaviour is occurs, but not in the form of spirals, but for instance bands of colours chasing each other from one side of the screen to the other.

NOTE: Perfectly circular spirals are of course not possible in a grid of square cells, but any polygonal shape with more than 8 sides, is classified as a circular spiral.

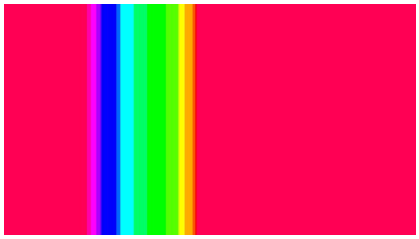
(a) diamond spirals

(b) square spirals



(c) octagonal spirals

(d) circular spirals



(e) non-spiral

Figure 6: Types of organized behaviour

3. **Fluctuation**
   In this class of behaviour, which we will call *fluctuation*, at every timestep, *every* cell is changed to the next state. This means the cycle length is always the same as the number of possible states. In this category, there is some fixed pattern, which cycles through all the states. This pattern can either be the initial state, which just *"flickers"* through all the colours, or this behaviour can occur after there has been some degree of clustering.

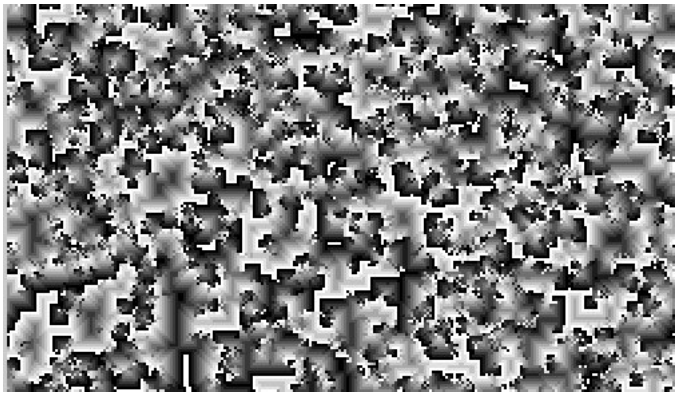


Figure 7: Fluctuation (after some clustering)

4. **Deterministic Chaos**
   Before the organized behaviour emerges, the automaton goes through a so-called transient phase. Sometimes this phase can persist for a very long time. Just because no organized behaviour is observed in these cases, does not mean it will not in the future.

   Because there are only a finite number of cells, at some point, the automaton MUST return to a universe it has already been in once before. For the automaton's behaviour to be classified as *deterministic chaos*, it must visit ALL possible (global) states, before repeating any previously visited states.

   Verifying that a behaviour is indeed *deterministic chaos* is however, not an easy matter. The only way to know that *deterministic chaos* has occurred, is to observe all possible states of the CA without any repeats. But for an NxM matrix, there are a total of k^{NM} different universes. So a 4-coloured 200x300 CA (like the one in the program) has $2,29 \times 10^{3612}$ possible universes. So even if it were possible to calculate a billion generations per second, the time it would take to generate all possible universes would still far exceed the time elapsed since the big bang.

   If for some setting no emergent behaviour has arisen within 100.000 generations, we will classify its behaviour as *Deterministic Chaos*. However, this does not guarantee that this is strictly correct, and that at no point in the future one of the other classes of behaviour will emerge, but merely that we do not have the time to verify this.

4.2.2 Do the same parameters always lead to the same class of behaviour?

In short, it can be said that the same parameters do not always lead to the same class of behaviour for different random initializations. In the vast majority of cases, the same type of behaviour will emerge every time for the same settings regardless of the particular random initialization. However, occasionally a difference in the

random initialization of the CA can lead a different class of behaviour. An example of this occurs for the 4-coloured case, with a Moore neighbourhood, a radius of 1 and a threshold of 2. In approximately 90 % of the cases (tested over 30 runs of the program), organized behaviour in the form of diamond spirals will evolve and settle into a cyclic pattern. (see Figure 8)

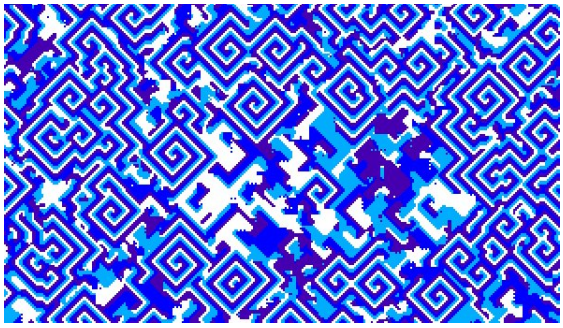
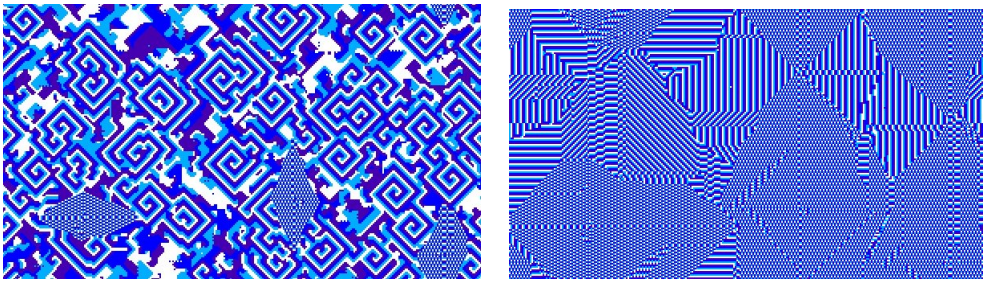


Figure 8: Most common behaviour

However, in approximately one in ten runs of these settings, certain disturbances form and can eventually take over the entire grid (fig 9). Once this has happened, each cell updates at every step, and thus this case falls under the category *fluctuation*.



(a) Disturbances are formed

(b) Disturbances take over the grid, leading to fluctuation

Figure 9: less common behaviour

Another example is that for certain setting, usually having thresholds on the lower end of the range, for some initializations of the grid, it will converge to a stable state, because not enough cells reach the threshold, while for certain other initializations just enough activity arises to bring the automaton over some kind of *critical momentum*, leading to cyclic behaviour.

4.2.3 Which parameter settings result in which type of behaviour?

Depending on the initialization method, the program can be run once or many times for each setting in order to investigate the behaviour of the CA.

Sparse initial universes

For the two-state cyclic CA, it might be interesting to look at the behaviour for sparse initial universes, that is, only one cell of state 1, while the rest is of state 0.

To get any interesting behaviour, threshold had to be 1, otherwise everything would become of state 0 after one generation, and stay that way.

- **Moore**
  For this neighbourhood, a square-shaped pattern evolved, consisting of bands alternated in colour. The width of these bands is always equal to the radius. An overview of the evolution in this case can be seen in figure 10.

Figure 10: Moore, evolution from t=0 to t=8, radius 2

- **von Neumann**
  The behaviour using the *von Neumann* neighbourhood is similar to that of the *Moore* neighbourhood, except here the basic shape was not a square, but a diamond. (see Figure 11)

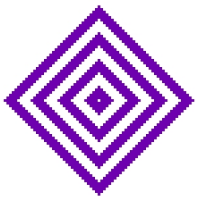
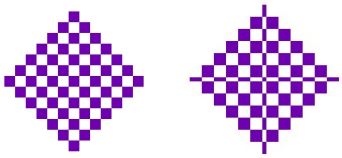


Figure 11: von Neumann, radius 5, generation 7

- **Plus**
  For this neighbourhood, the behaviour was a bit more complicated. Figure 12(a) shows the behaviour for a radius of 1. Here at every timestep, another *"layer"* is added around the boundary, so the diameter at its longest point is always $2 * t + 1$ cells.



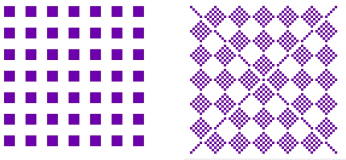
(a) radius 1 (b) radius ≥ 2

Figure 12: Plus neighbourhood after 6 generations for different radii

But changing the radius to 2 or higher, does not simply lead to thicker squares in the pattern. In figure 12(b) the pattern for higher radii is shown. A radius

of 2 is the same as higher radii, except again for the *"thickness"* of the squares (and length of the lines).

- **Cross**
  The *Cross* neighbourhood is similar to the *Plus* neighbourhood, except the pattern is rotated 90 degrees, to resemble a square, and for radii ≥ 2, the areas that were solid squares under the *Plus* neighbourhood are now checkered squares (due to the disjoint neighbourhoods explained in section 4.1.1).



(a) radius 1 (b) radius ≥ 2

Figure 13: Cross neighbourhood after 6 generations for different radii

Uniformly distributed initial universes

In the case of uniformly distributed random initial universes, there is a lot of randomness in the initial configuration, and the behaviour may vary between separate runs using the same parameters. Therefore, the program should be run several times to find the most common class of behaviour. This also allows us to answer the research question: *Will the same settings always lead to the same class of behaviour?* Also, to be able to give an accurate estimate of the convergence time and cycle length, we will have to average over several runs of the program. These were the results for the four-state and the sixteen-state CA:

4 Colours

Neighbourhood	Radius	Threshold	Type of behaviour	Average convergence time	Cycle length
Moore	1	1	fluctuation		4
		2	organized, diamond spirals occasionally fluctuation		8
		3	organized, octagonal spirals		72
		4	convergence to stable state	26	
		5		7	
		6		4	
		7		3	
		8		2	
	2	1	fluctuation		4
		2			
		3			
		4			
		5	organized, octagonal spirals		12
		6	organized, circular spirals		
		7	organized, diamond spirals		
		8	deterministic chaos		
		9			
		10	convergence	2500	
		11		20	
		12		10	
		13		5	
		14		5	
		15		3	
		16-24		1	
Von Neumann	1	1	fluctuation		4
		2	convergence	40	
		3		5	
		4		2	
	2	1	fluctuation		4
		2			
		3	organized, octagonal spirals		72
		4	deterministic chaos		
		5	convergence	200	
		6		20	
		7		6	
		8		4	
		9-12		1	
Plus	2	1	fluctuation		4
		2	organized, octagonal spirals		32
		3	convergence	104	
		4		17	
		5		7	
		6		3	
		7		2	
		8		1	

Table 1: The observed behaviour, cycle length, and convergence time for the 4-state CA with different radii and neighbourhood types.

16 colours

Neighbourhood	Radius	Threshold	Type of behaviour	Average convergence time	Cycle length
Moore	1	1	fluctuation		16
		2	convergence	21	
		3		5	
		4		4	
		5		2	
		6-8		1	
	2	1	fluctuation		16
		2			
		3	convergence	610	
		4		20	
		5		8	
		6		6	
		7		3	
		8-24		1	
von Neumann	1	1	organized, diamond spirals		32
		2	convergence	4	
		3		2	
		4		2	
	2	1	fluctuation	16	
		2	organized, non-spiral occasionally convergence	2000	300
		3	convergence	9	
		4		6	
		5-12		1	
Plus	2	1	fluctuation		16
		2	convergence	19	
		3		5	
		4		3	
		5		2	
		6-8		1	

Table 2: The observed behaviour, cycle length, and convergence time for the 16-state CA with different radii and neighbourhood types

NOTE: for the *Plus* neighbourhood, radius 1 is not shown in the tables because it is identical to the *von Neumann* neighbourhood with radius 1.

Conclusions

From the results we can see that in all cases, when changing threshold from low to high, and keeping all other parameters constant, the behaviour goes from *fluctuation* for the low thresholds, through *organized behaviour/deterministic chaos*, to *convergence* for high thresholds.

This behaviour is not surprising, because for lower thresholds, every cell will easily have enough neighbours of the next colour, and will always change to the next colour, which leads to *fluctuation*. And for high thresholds, not many cells have enough neighbours of the next colour, and so not a lot of cells will change value, which causes convergence to a stable state. In between lies the area of complexity, where organized cyclic behaviour appears.

In fact, changing any one parameter from low to high, whether it be the number

of colours, the radius or the threshold, and keeping the others constant, the observed behaviour goes through these same pattern. Fluctuation for low values, convergence to a stable state for high values, and complexity or deterministic chaos in between.

Our fourth and final research question was: *If a cyclic pattern has emerged, can we say anything about the (global) cycle length, and how this relates on the parameters and/or canvas size?* When we do observe complex behaviour, the cycle length can vary between separate runs with the same settings, but it must always be a multiple of the number of possible states per cell. Usually when complex behaviour is observed, a number of spirals appear, and grow bigger and bigger until they fill most of the screen. How the space in between these spirals is filled, determines the exact cycle length, so the same settings can lead to the same type of behaviour, but with a different cycle length. In the case of non-spiral cyclic behaviour, cycle length can also differ, but in the case of bands of colours moving from one side to the other, the cycle length will always be the width of the CA (or height if the stripes move from top to bottom).

5 Java Program

The Java application is intended as an educational tool and as a way to interest users in the subject by showcasing as many of the interesting and often wondrous properties of cellular automata.

There are 5 different subprograms, implemented as different tabs in the main program. The subprograms all have some standard features in common. There are buttons at the top of the control panel to start the evolution of the CA, to stop it, to take a single step forward, and to go a single step backwards. There is a restart button which resets the CA to its original state, and a *random* button which randomly initializes the CA. The five different subprograms are:

1. **2D Cyclic**

   This implements the two dimensional cyclic cellular automaton. Apart from the standard buttons, there is one more button labelled *snapshot*. When this button is pressed, the program remembers the current state of the CA and for each state in the future, checks if it is the same as this one. If it is identical, a cycle has occurred, and the program reports this along with the length of the cycle (how many generations passed before same state was observed again).

   Furthermore, a choice of neighbourhood can be made using a drop-down menu. Also, the radius and the number of states per cell can be chosen. The threshold can also be set using a scrollbar. Here, the maximum value is automatically adjusted to the neighbourhood type and radius, so that the threshold can never exceed the total number of neighbours.

   An added feature for this particular subprogram, is that it reports when convergence to a stable state has occurred, and after how many generations.

2. **1D Cyclic**
   This is the one dimensional version of the cyclic CA. Here the only parameters

are threshold, radius, and the number of states each cell can be in. Again a snapshot can be taken to detect cycles.

3. **2D Parity**
   This particular instance of a totalistic CA, first described by Edward Fredkin around 1960 [5], produces interesting and quite amusing results. Every cell can be in one of 2 states. The transition rule here is a simple *modulo-2* operation on the total value of all the neighbours. The effect that is observed here is that every 2^k, $k = 0, 1, ..$ generations, the configuration we started with is duplicated at exactly the positions of the neighbours themselves. At every next 2^k generations, the distance of those duplicates has doubled. Below is an example that uses a von Neumann neighbourhood of radius 2. Starting from the initial configuration, we see complicated patterns emerge until at generation 8 we recognize four copies of the initial figure, and again at generation 16, but now further apart.

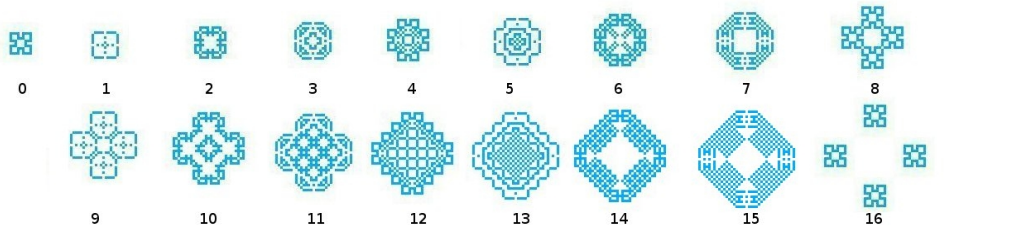


Figure 14: evolution of a 2-state Modulo CA

We see that here the copies of the original figure are positioned at the top, bottom, left and right of the original, exactly the positions in the *von Neumann* neighbourhood. If we use a radius of 2 instead of 1, the 16th generation looks like this:

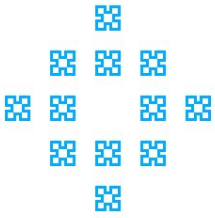


Figure 15: generation 16 for *von Neumann* neighbourhood, radius 2

And were we to use a *Moore* neighbourhood, the 16th generation would look like this:

So we see that the duplicates always end up exactly at the positions of the neighbours, and the distance doubles at every next power of two.

This effect will always appear, no matter what initial picture is used, though the bigger the initial picture, the longer it may take to see the copies, because at the lower powers of two, the copies will overlap, making them indistinguishable.

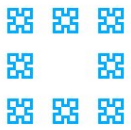


Figure 16: generation 16 for *Moore* neighbourhood (radius 1)

The principle can be extended to higher-state CAs as well, but here the effects are slightly different.

In the program, some pre-coded initial pictures can be selected, or alternately, custom initial states can be drawn onto the canvas, each mouse click updating the state of the cell to the next colour. (To get the picture of the example, pick the *clover* option from the menu.)

4. **1D Standard**
   This is a standard one-dimensional CA. A rule number can be entered and the number of states and radius can be selected. There is a box that can be checked to make it a totalistic CA. The maximum rule number is always displayed as well, so users know how high they can choose their rule number. For large neighbourhoods and/or many states, the number of possible rules explodes and can become more than the program can handle, therefore for those setting the maximum rule number is the highest that can be entered into the program, not the total number of theoretically possible rules.

5. **Game of Life**
   This is an implementation of the popular *Game of Life* first created by John Conway. This is an instance of a two dimensional, two-state, totalistic CA using a Moore neighbourhood with radius 1, in which the following transition rules apply:

   (a) Any *living* cell with fewer than two living neighbours dies, as if by loniness.

   (b) Any *living* cell with more than three living neighbours dies, as if by overcrowding.

   (c) Any *non-living* cell with exactly three living neighbours, comes to life.

   (d) In all other cases the state of the cell remains unchanged.

   Here different initial configurations can be selected from a dropdown menu. Alternately, a custom pattern can be drawn on the grid using the mouse. Clicking the right mouse-button clears the screen.

5.1 Recommended settings

Finally, let us list some of the parameter settings which nicely illustrate some of the different classes of behaviour:

- **Diamond Spirals**
  subprogram: 2D cyclic, colours: 16, colour scheme: rainbow, neighbourhood: von Neumann, radius: 1, threshold: 1.

- **Fluctuation**
  subprogram: 2D cyclic, colours: 10, colour scheme: gray, neighbourhood: Moore, radius: 1, threshold: 1.

References

[1] C. A. Reiter C. Ning. A cellular model for three-dimensional snow crystallization. *Computer Graphics*, 31(4):668–677, August 2007, doi:10.1016/j.cag.2007.02.015.

[2] J. Gravner D. Griffeath, R. Fisch. *Spatial Stochastic Processes: A Festschrift in Honor of Ted Harris on His Seventieth Birthday*, chapter Cyclic Cellular Automata in Two Dimensions, pages 171–185. K.S. Alexander and J.C. Wadkins eds. Birkhäuser Verlag, Boston, 1991.

[3] Dan Livingstone Eduardo Reck Miranda Dave Burraston, Ernest Edmonds. Cellular automata in midi based computer music. International Computer Music Association, Inc. (ICMA), San Francisco, CA, 2004.

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

[5] Martin Gardner. *Wheels, Life, and Other Mathematical Amusements*. W. H. Freeman and Company, New York, 1983.

[6] N. Chiba S. Gobron. Simulation of peeling using 3d-surface cellular automata. pages 338–347. IEEE Computer Society, Washington DC, 2001, doi:10.1109/PCCGA.2001.962890.

[7] Barry Shackleford, Motoo Tanaka, Richard J. Carter, and Greg Snider. High-performance cellular automata random number generators for embedded probabilistic computing systems. In *EH '02: Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02)*, page 191. IEEE Computer Society, Washington DC, 2002, doi:10.1109/EH.2002.1029885.

[8] Stephen Wolfram. Cryptography with cellular automata. pages 428–432. Springer-Verlag New York, Inc., June 1986.

[9] Stephen Wolfram. Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7(2):123–169, June 1986.

[10] Stephen Wolfram. *A New Kind of Science*. Wolfram Media Inc., May 2002.