

SpirallingCells - A Cellular Automaton

Dr. Peter U.

Version 1.0

March 2017

Contents

1	Introduction	2
2	The Cellular Automaton	3
3	The Simulation	4
3.1	Prerequisites	4
3.2	Controls	5
3.3	File naming conventions	7
3.4	Global constants	7
4	Simulation Results	8
4.1	Stage 1: Random State	8
4.2	Stage 2: First structure	9
4.3	Stage 3: Spirals taking over	10
4.4	Stage 4: Domain of the Spirals	11
5	Interpretation	12

1 Introduction

This paper covers a particular kind of cellular automaton (CA) which was mentioned in the *Computer Recreations* section of the August 1988 issue of *Scientific American* by A.K. Dewdney. Subsequently a german version of Dewdney's article was published in *Spektrum der Wissenschaft* (SdW), the German edition of *Scientific American*, in November 1989. As a subscriber of SdW, that was the first time I stumbled across this automaton or CAs in general.

The particular cellular automaton described here was invented by David Griffeath of the University of Wisconsin at Madison. It was mentioned by Heike Schuster and Martin Gerhardt of the University of Bielefeld in Germany in their book *Das digitale Universum - Zellulaere Automaten als Modelle der Natur* (ISBN: 978-3528066772, in German language).

The automaton can be understood as a primitive model for a certain kind of chemical reaction which results in geometric patterns which propagate through a medium. These are known as *Belousov-Zhabotinsky reactions* (BZ reactions). The discovery of this type of reactions was a major conceptual breakthrough because at the time it was believed all chemical reactions would end up in an equilibrium state while the BZ reactions result in a state far from equilibrium.

Accompanying this paper is an implementation of the CA in Python which allows modification of some parameters and a visualization of the automaton's evolution.

2 The Cellular Automaton

The automaton consists of a rectangular grid of cells (dimension **width** times **height**), each of which can possess one of **N** different states encoded as numbers $0, 1, \dots, N - 1$. The states are cyclic, i.e. state $N - 1$ is followed by state 0.

The grid is also cyclic, i.e. "wraps around" both horizontally and vertically. This means the neighbours of the cells in the bottom row include the cells in the top row and similarly for the leftmost/rightmost columns. So, topologically the grid is equivalent to a torus (think of a floating tire).

The CA evolves to the tick of an imaginary clock. In each generation a cell's state s either stays the same or increases by one (modulo N). It increases by one if and only if at least one of its neighbours is in the state $s + 1 \bmod N$. All cells evolve simultaneously. Therefore a cell's state s_{i+1} in generation $i + 1$ is given by:

$$s_{i+1} = \begin{cases} (s_i + 1) \bmod N & \text{if } \text{state } (s_i + 1) \bmod N \text{ in neighbourhood} \\ s_i & \text{if } \text{else} \end{cases}$$

"Neighbourhood" typically refers to the *von Neumann neighbourhood*, i.e. the four cells connected horizontally or vertically to the cell in question. The simulation (see section "The Simulation") also allows to choose the *Moore neighbourhood* which additionally includes the 4 diagonally connected neighbours, i.e. each cell then has 8 neighbours.

At the start of the simulation each cell is randomly assigned a state in the interval $[0, N - 1]$. The automaton then evolves according to the rule specified above.

Optionally, for each generation a parameter "Min Dist" is determined which is the minimum absolute distance ($\bmod N$) for each cell to its neighbours (according to the neighbourhood, von Neuman or Moore)

3 The Simulation

The implementation of the CA is done in a standalone Python 2.7 program.

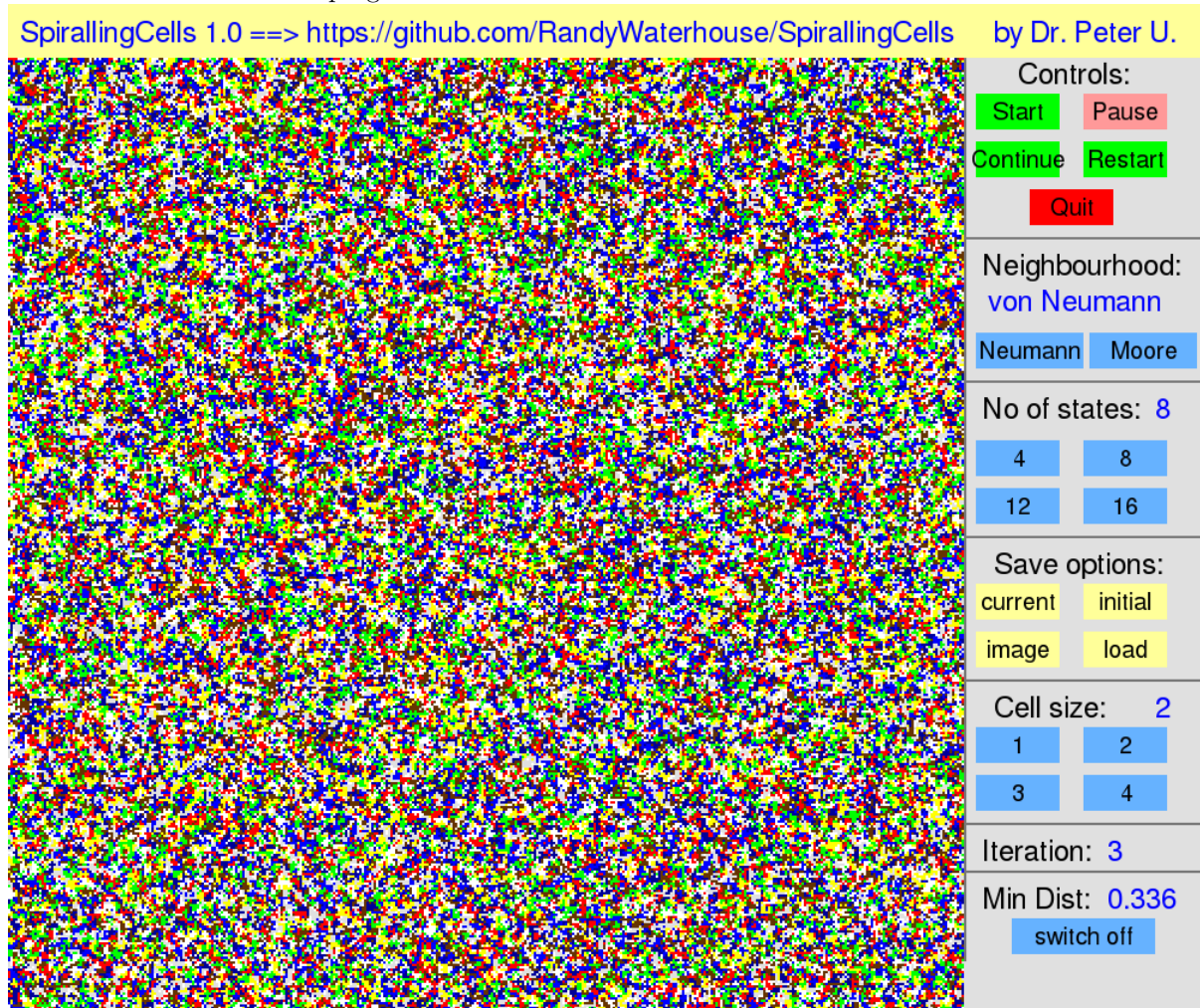
3.1 Prerequisites

Besides a number of standard packages it uses the following packages which therefore must be installed:

- **pygame:** Graphics output, i.e. visualization of the CA's state as well as the control section of the program
- **pickle:** Saving and loading of simulation states
- **Tkinter:** Used for "open file" dialog, only needed for loading of formerly saved simulation states
- **cProfile:** Profiling of the program's performance, can be commented out if not needed

3.2 Controls

A screenshot of the program looks like this:



The user control section on the right-hand side of the program window has the following options:

- **Program flow:**

- **Start:** Starts the simulation (upon starting the program as such, the simulation is already running); if "Start" is clicked during a running simulation, a new simulation (with new randomly chosen initial

state) is started

- **Pause:** Pauses the simulation, i.e. evolution halts and the program waits for further user actions
- **Continue:** Continue the simulation, typically after pausing it previously
- **Restart:** Restarts the simulation with the same initial state as for the currently running simulation
- **Quit:** Quit the program; the current state is not saved; also keys **ESC** or **Q** may be used to quit

- **Neighbourhood:**

- **Neumann, Moore:** Select the type of neighbourhood to use in the simulation: von Neumann (4 neighbours) or Moore (8 neighbours); this option may be changed during a running simulation

- **Save options:**

- **current:** Save current state of the simulation to a file, it can later be reloaded
- **initial:** Save initial state of the current simulation to a file, it can later be reloaded
- **image:** Save image of current program window to file
- **load:** Load a previously saved simulation state to further simulate it; an "open file" dialog pops up

- **Cell Size:**

- **Options 1, 2, 3, 4:** Each cell is drawn as a square of side length 1, 2, 3 or 4, respectively; larger squares means less cells in a given window size and therefore a faster simulation

- **Iteration:**

- Shows current iteration number

- **Min Dist:**

- Show current value of "Min Dist" (see above) if so selected; determining the MinDist value slows down the simulation and therefore can be switched off

– `switch on/switch off`: toggle calculation and display of MinDist

3.3 File naming conventions

Files (simulation states and images) saved by the program follow a naming convention, an example file name would be:

`SpirallingCells_current_20170320_0737_8_2_400_400.p`

- `current`, `initial`, `image`: indicates saves of current state, initial state or image
- `20170320`: Date in format `YYYYMMDD`
- `0737`: Time in format `HHMM`
- Remaining underscore-delimited numbers: number of states (N), cell size, width, height
- ending: `"p"`: pickle files, i.e. dump of program variables; `"png"`: image in PNG format

3.4 Global constants


A number of constants are defined in the Python program, they are commented in the code. Especially important for user purposes are `MAIN_WIDTH` and `HEIGHT` which define the width and height of the graphics window for the CA output (without the control section), respectively. The width and height of the CA are therefore `MAIN_WIDTH / cell_size` and `HEIGHT / cell_size`, respectively.

4 Simulation Results

Running the simulation lets one identify four different stages of the Cellular Automaton:

4.1 Stage 1: Random State

SpirallingCells 1.0 ==> <https://github.com/RandyWaterhouse/SpirallingCells> by Dr. Peter U.



Controls:

Start Pause

Continue Restart

Quit

Neighbourhood:
von Neumann

Neumann Moore

No of states: 8

4 8

12 16

Save options:

current initial

image load

Cell size: 2

1 2

3 4

Iteration: 2

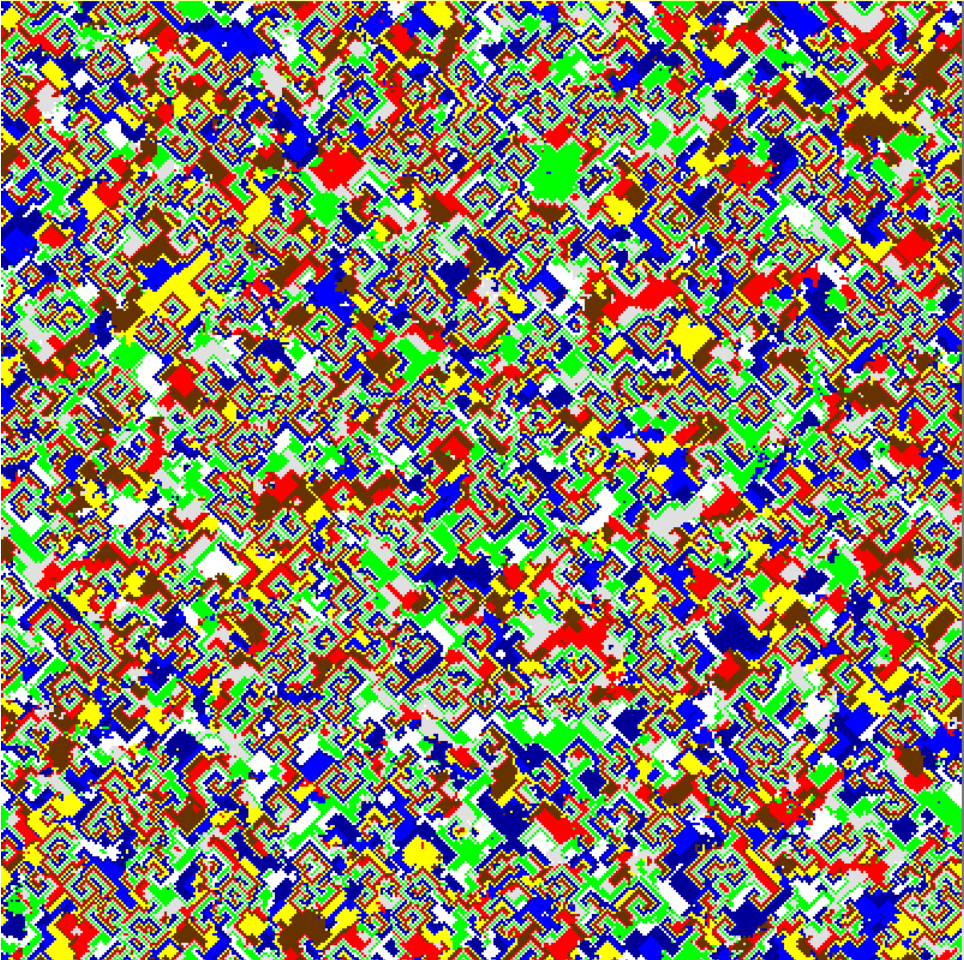
Min Dist: 0.385

switch off

Initial state, cells have random values in interval $[0, N - 1]$

4.2 Stage 2: First structure

SpirallingCells 1.0 ==> <https://github.com/RandyWaterhouse/SpirallingCells> by Dr. Peter U.



Controls:

Start Pause

Continue Restart

Quit

Neighbourhood:

von Neumann

Neumann Moore

No of states: 8

4 8

12 16

Save options:

current initial

image load

Cell size: 2

1 2

3 4

Iteration: 21

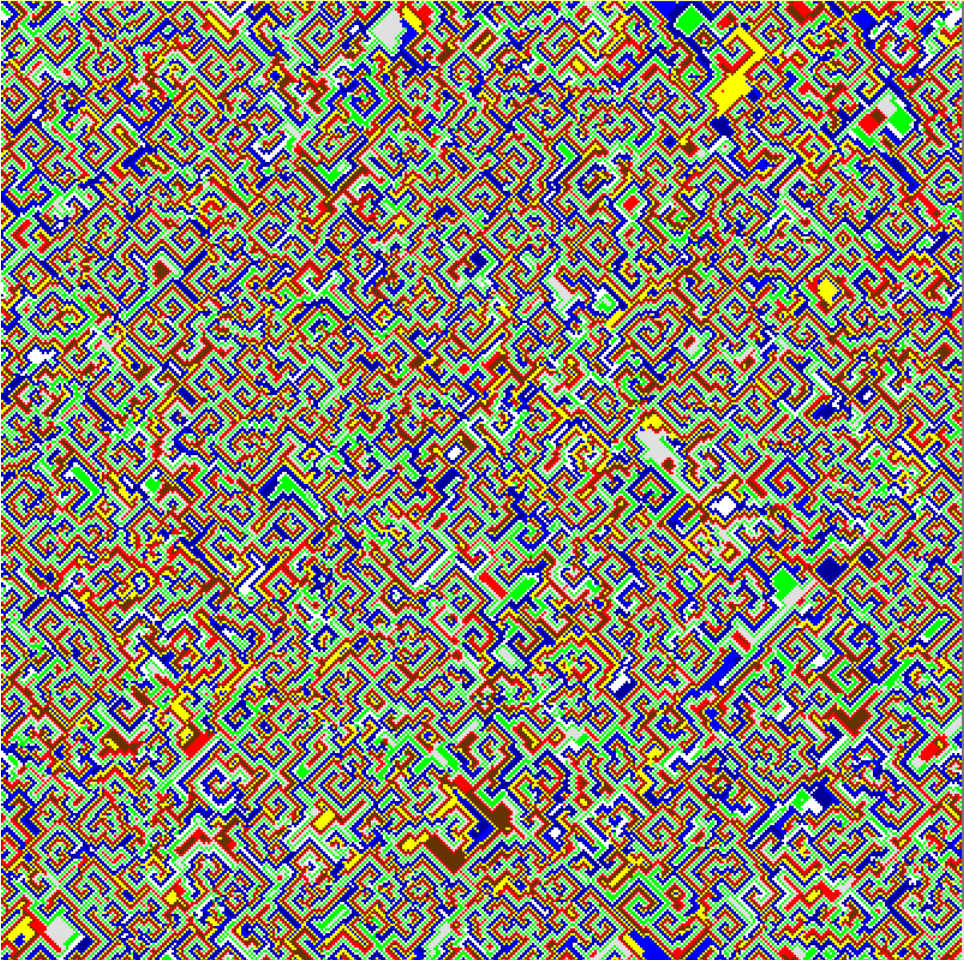
Min Dist: 0.387

switch off

First structures are visible: small patches of uniform state (i.e. same color) as well as some initial spirals (see stages 3 and 4)

4.3 Stage 3: Spirals taking over

SpirallingCells 1.0 ==> <https://github.com/RandyWaterhouse/SpirallingCells> by Dr. Peter U.



Controls:

Start Pause

Continue Restart

Quit

Neighbourhood:
von Neumann

Neumann Moore

No of states: 8

4 8

12 16

Save options:

current initial

image load

Cell size: 2

1 2

3 4

Iteration: 32

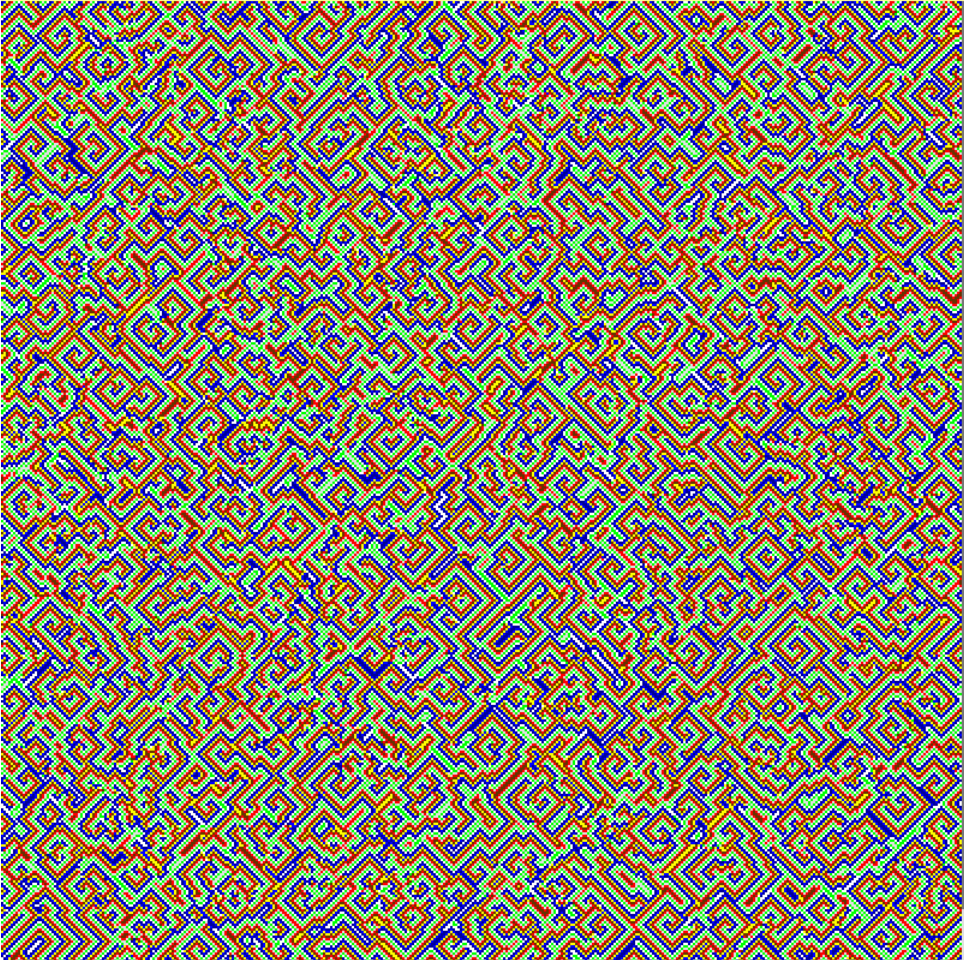
Min Dist: 0.701

switch off

The picture is now dominated by spirals with some isolated areas still showing uniform color.

4.4 Stage 4: Domain of the Spirals

SpirallingCells 1.0 ==> <https://github.com/RandyWaterhouse/SpirallingCells> by Dr. Peter U.



Controls:

Start Pause

Continue Restart

Quit

Neighbourhood:
von Neumann

Neumann Moore

No of states: 8

4 8

12 16

Save options:

current initial

image load

Cell size: 2

1 2

3 4

Iteration: 140

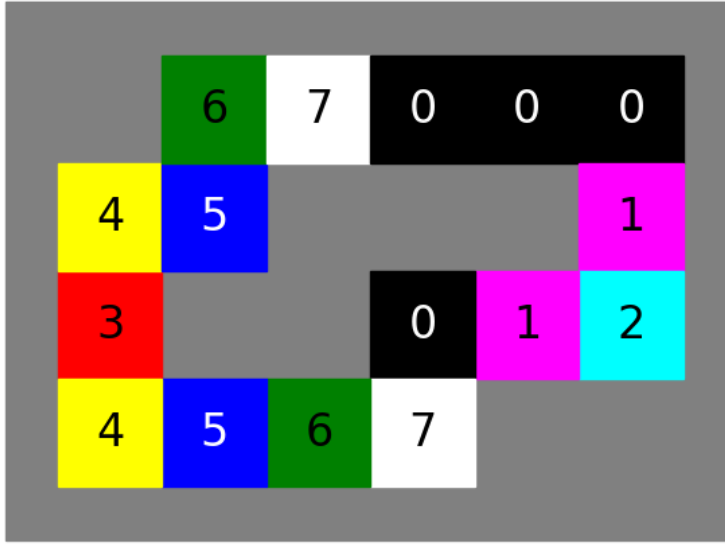
Min Dist: 0.91

switch off

The spirals now occupy all or almost all of the CA. The CA is in a meta-stable state, i.e. the structure of the spirals remains constant but the colors (states) are "cycling through". See next section for interpretation.

5 Interpretation

So what are the spirals all about? Let's look at a "chain" of cells which form, to use Griffeath's terminology, a "defect":

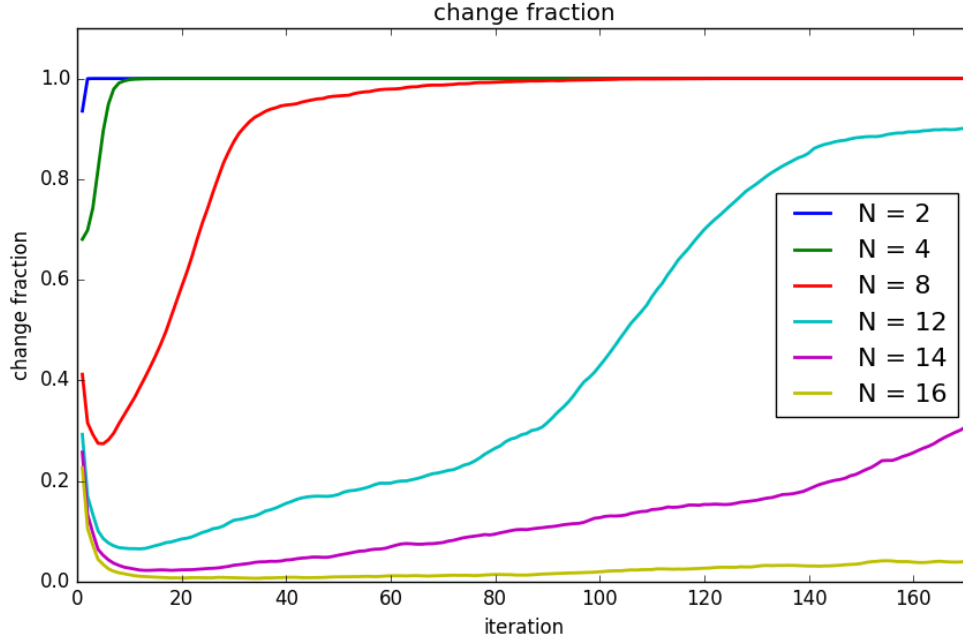


Such a defect consists of a sequence of cells where each cell's state is either equal to at least one of its neighbour's states or higher by one and where at one point the chain is "mirrored". The picture shows an example for $N = 8$ with the "mirror" being at the cell in state 2 (cyan-colored).

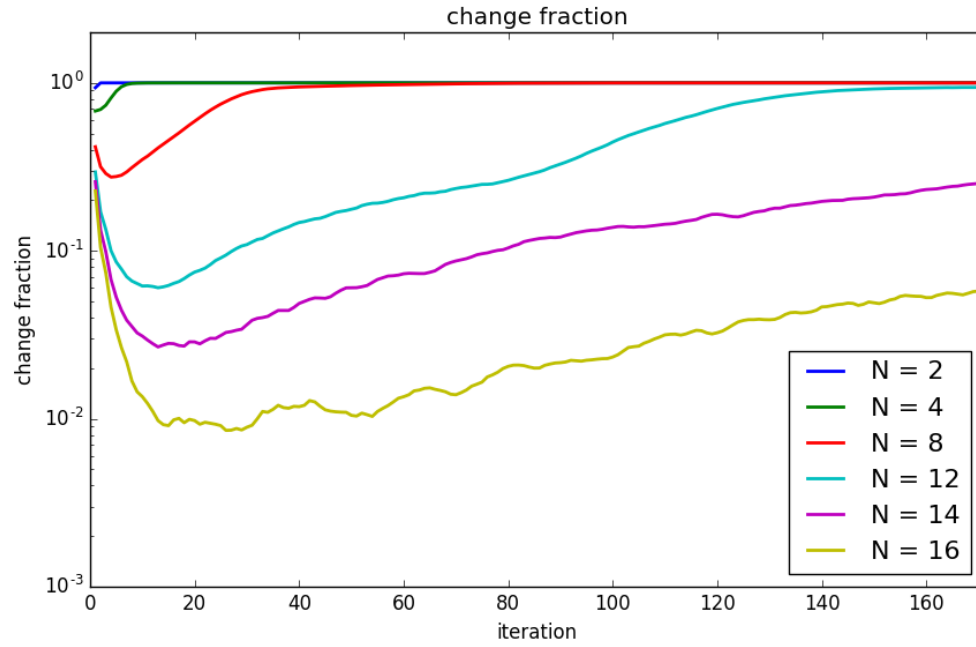
Such defects are the "core" or "seed" of the spirals. A chain cycles through the states from 0 to $N-1$ infinitely and "eats" neighbouring cells which are in the right state, thereby expanding until they are met by other spirals which expand too. At this point the structure is meta-stable.

In the meta-stable state the MinDist parameter is close to one because most cells have a neighbour which is in distance 1 but no neighbour which is in the same state.

To illustrate the analysis, the following plot shows the change fraction as a function of iteration number for a grid of 200 by 200 cells and a von Neumann-neighbourhood. The change fraction is defined as the fraction of all cells which changed their state from the last generation to the current one. Different number of states N are used and color-coded:



We note that for a small number of possible states ($N = 2$ or 4) the system quickly reaches a state where all cells change their respective states in each iteration (change fraction equal to 1). For a larger number of states this takes longer but is eventually achieved. The next plot shows the same data on a logarithmic y-scale:



The behaviour shown in the plots is to be expected: the more states there are the longer it takes for the spirals to evolve and take over the complete grid. Eventually this happens, though, as can be seen in the next plot which shows the long-term behaviour (1000 iterations) for $N = 16$:

