

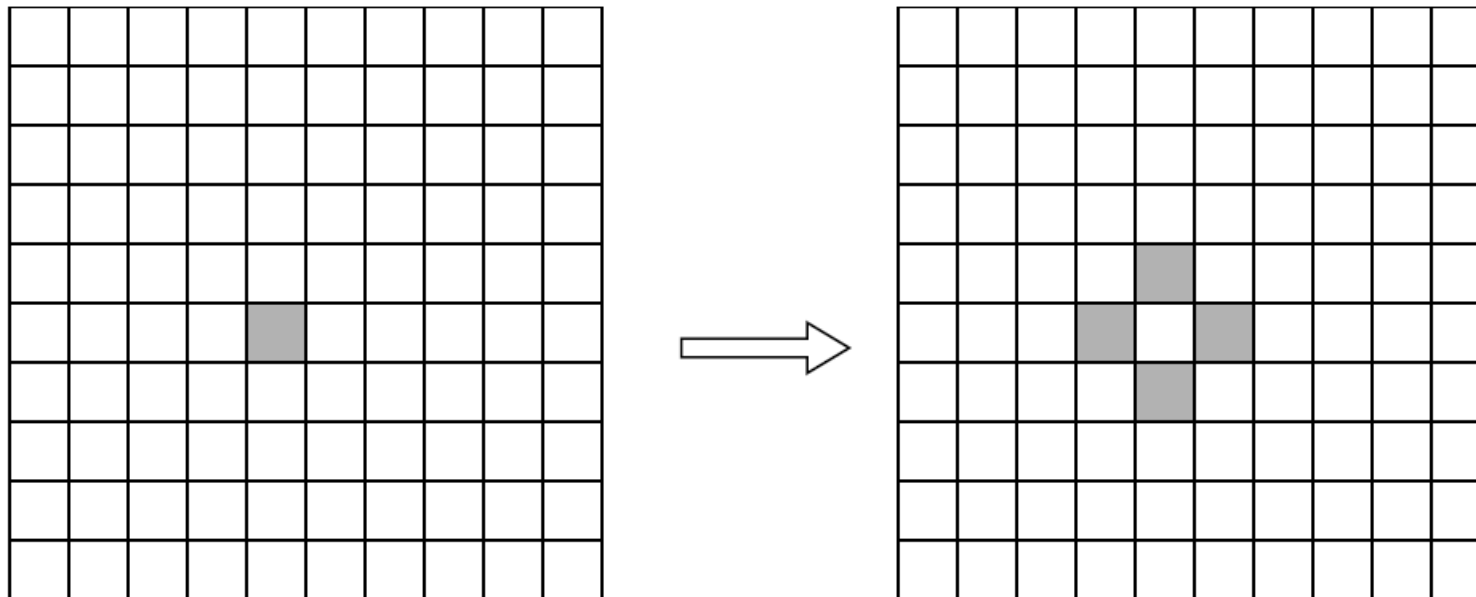


Cellular Automata

Complex Simplicity or Simple Complexity?

Example: Parity Rule

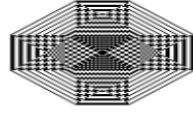
- Square Lattice
- Possible states $s_{ij} \in \{0,1\}$
- Rule:
 1. Each cell sums up the state of the 4 neighbors (south, west, east, north)
 2. If the sum is even, then $s_{ij} = 0$, else $s_{ij} = 1$



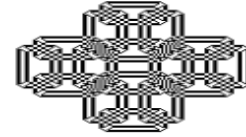
Evolution of the Parity rule



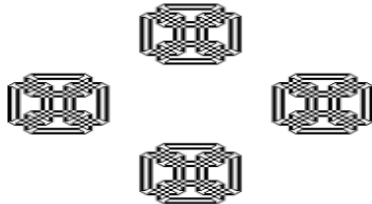
t=0



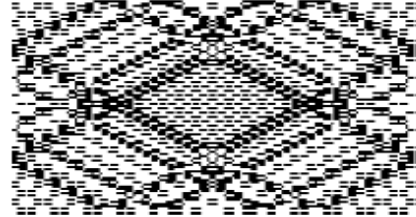
t=31



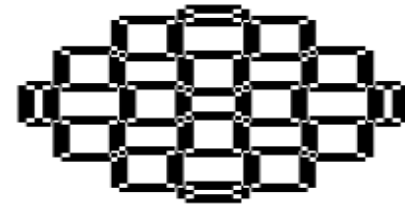
t=43



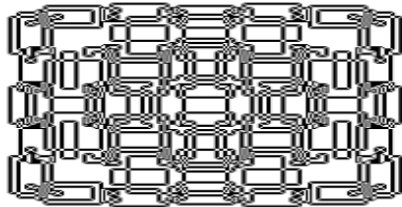
t=75



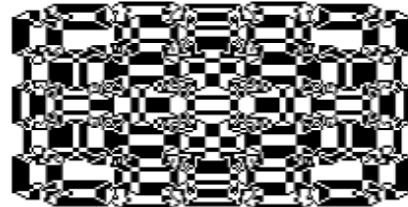
t=248



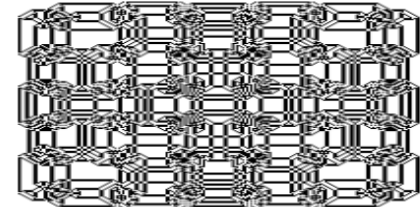
t=292



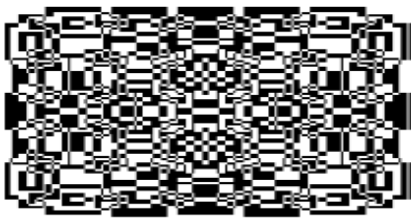
t=357



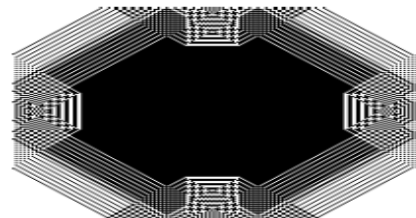
t=358



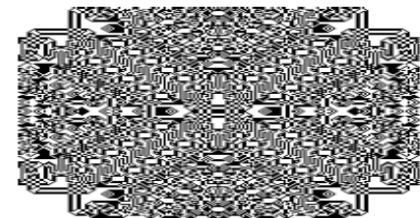
t=359



t=360



t=511

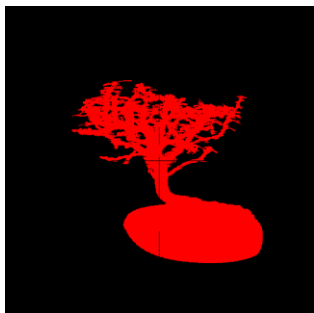
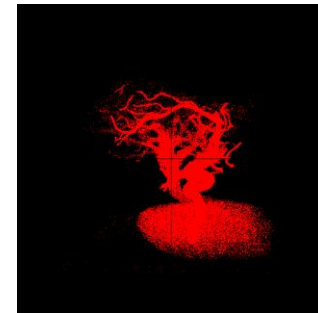
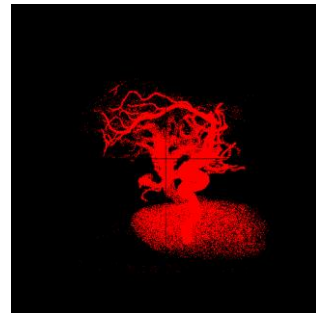
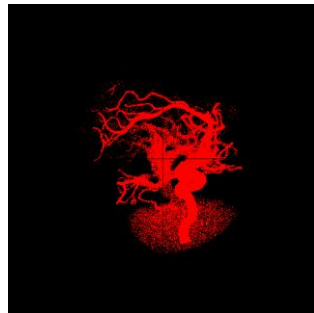
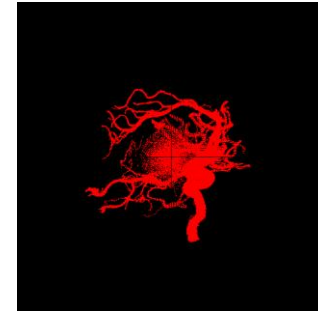
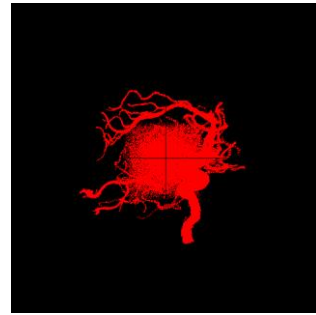
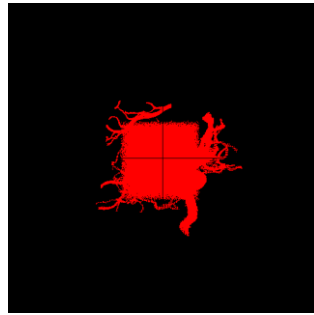
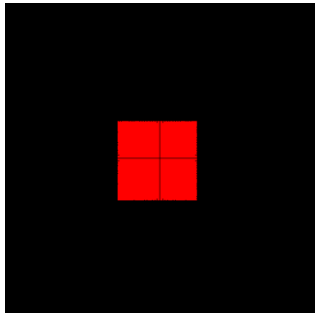


t=571

What are Cellular Automata?

- It is a **model** that can be used to show how the elements of a system **interact with each other**.
- Each element of the system is **assigned a cell**.
- The cells can be:
 - 2-dimensional squares (e.g., patches of land)
 - 3-dimensional blocks (e.g., voxels)
 - or another shape such as a hexagon (e.g., beehives)

Using Voxels: Morphing (Cube \rightarrow Bonsai)



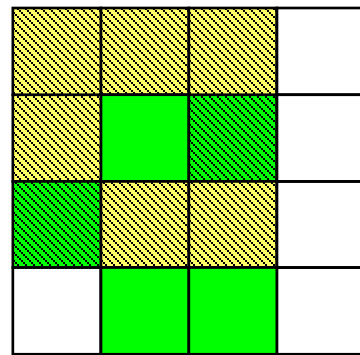
Cellular Automata: More Specifically

A CA is a spatial lattice of N cells, each of which is one of k states at time t .

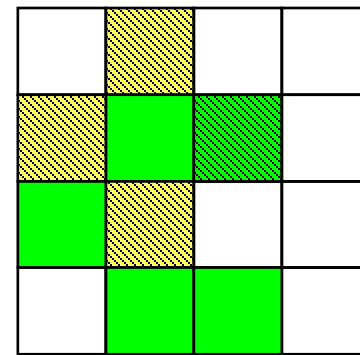
- Each cell follows the same simple rule for updating its state (**homogeneity**).
- The cell's state s at time $t + 1$ depends on its own state and the states of some number of neighbouring cells at t (**locality**).
- For 1-d CAs, the neighbourhood of a cell consists of the cell itself and r neighbours on either side. Hence, k and r are the parameters of the CA.
- CAs are often described as discrete dynamical systems with the capability to model various kinds of natural discrete or continuous dynamical systems

Cellular Automata: the specification

- A **neighborhood function** that specifies which of the cell's adjacent cells affect its state.
- A **transition function** that specifies mapping from state of neighbor cells to state of given cell



Moore
neighborhood



Neumann
neighborhood

...

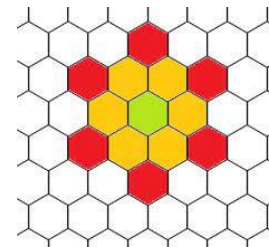
Computation Characteristics of Cellular Automata

- Synchronous computation
- Infinitely-large grid:
 - but finite occupancy,
 - grows when needed
- Various dimensions (1D, 2D, 3D, ...)
- If cells are distributed, they still need to communicate across boundaries; they communicate once per cycle.

To Recap

A Cellular Automaton is:

1. A Discrete Space A (regular lattice of cells/sites in d dimensions)
2. Discrete Time (Continuous Time as well but we are not interested 😊)
3. A set of states S for each cell
4. Local homogenous evolution rule Φ (defined for a neighborhood N)
5. Synchronous (parallel) updating of all cells
6. Tuple: $\langle A, S, \Phi, N \rangle$



What are the Applications of Cellular Automata?

- Universal computers (embedded Turing machines)
- Self-reproduction
- Diffusion equations
- Artificial Life
- Digital Physics
- ...

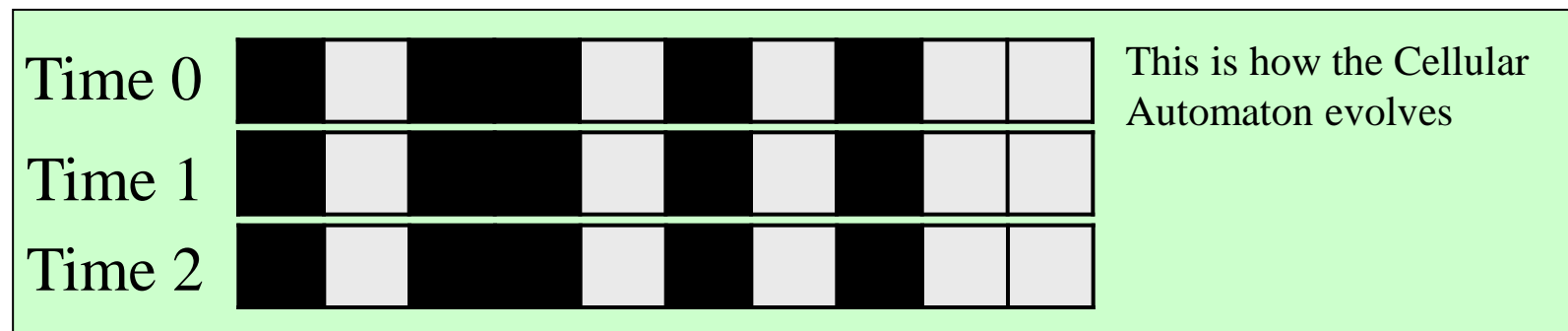
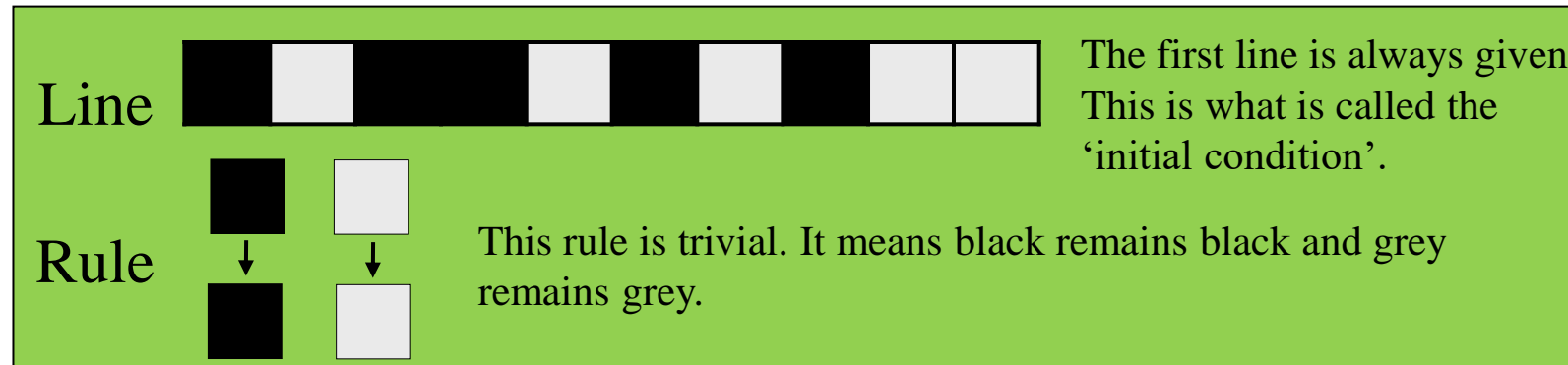
Some Examples of Application of CA: Simulation Models

- “Game of Life”
- Gas particles: Billiard-ball model
- Ising model: Ferro-magnetic spins
- Heat equation simulation
- Percolation models
- Wire models
- Lattice Gas models


1-d Cellular Automata

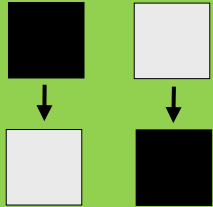
A Very Simple Rule


A (one-dimensional) cellular automaton consists of a line of 'cells' (boxes) each with a certain color like black or grey and a rule on how the colors of the cells change from one time step to the next.





Another Simple Rule

Line  The first line is always given. This is what is called the 'initial condition'.

Rule  Another simple rule. It means black turns into grey and grey turns into black.

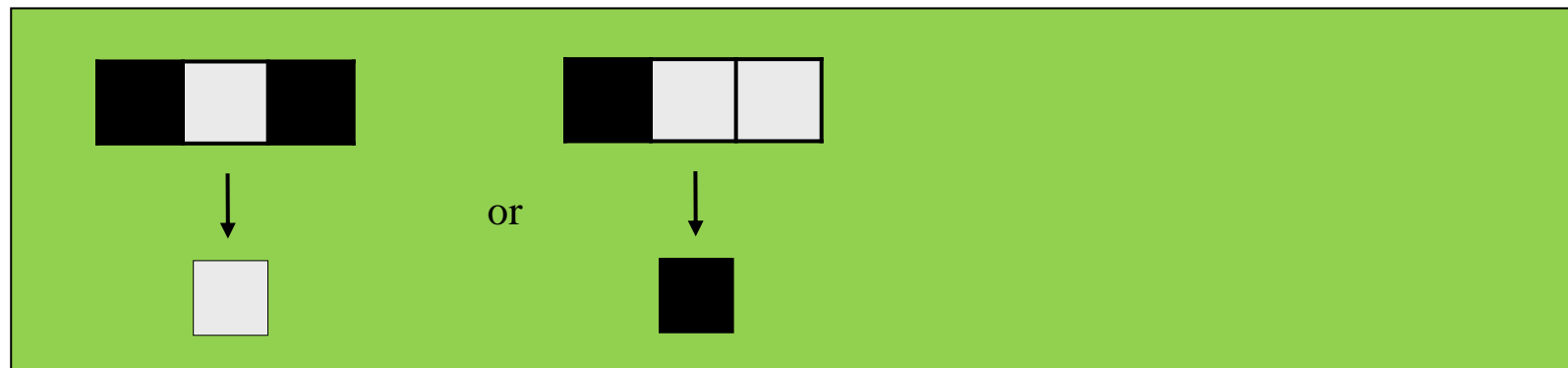
Time 0  This is how the Cellular Automaton evolves

Time 1 

Time 2 

Waking up the Neighbours^(B.A. 1991 ☺)

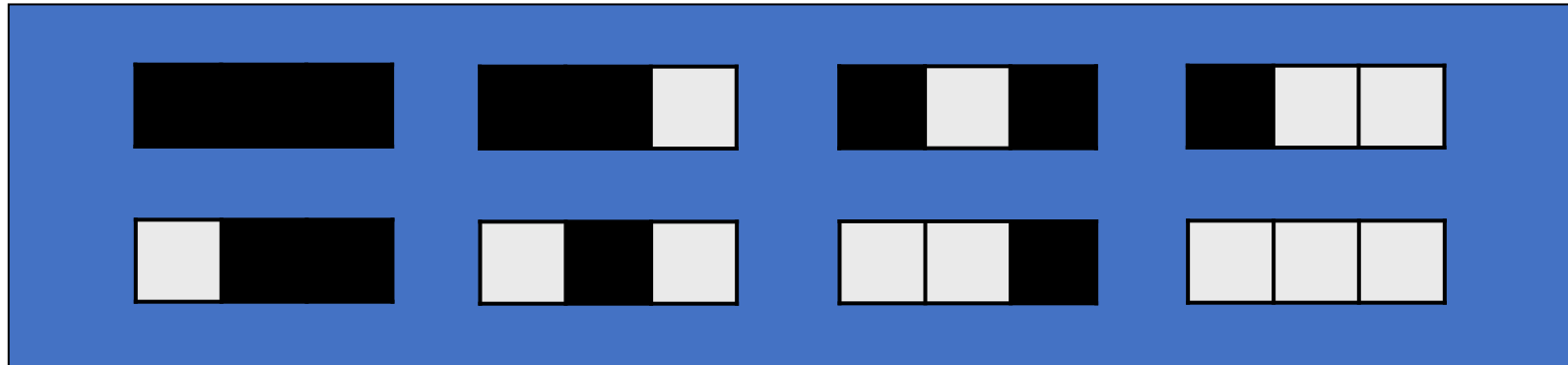
Like this, the rules are a bit boring of course because there is no spatial dependence. That is to say, neighboring cells have no influence. Therefore, let us look at rules that take nearest neighbors into account.



With 3 cells and 2 colors, there are 8 possible combinations.

3-Neighborhood

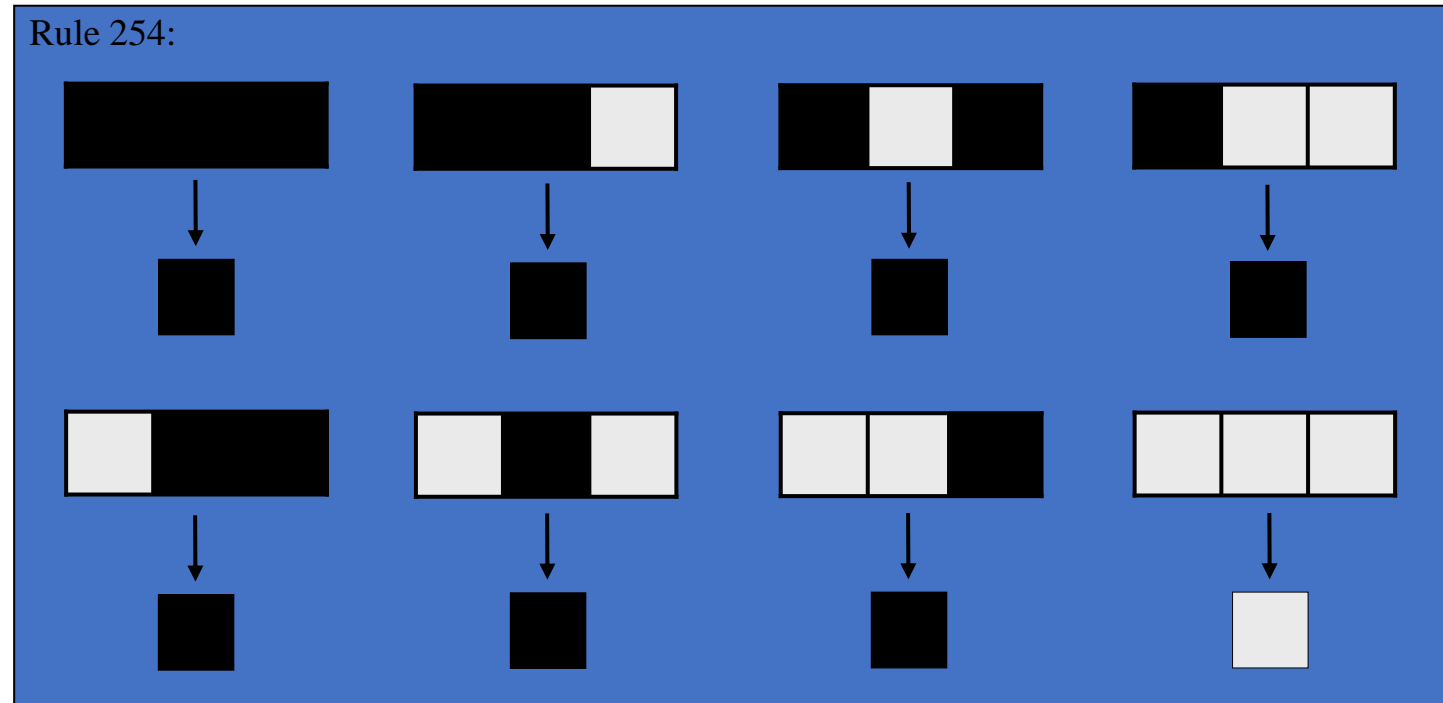
The 8 possible combinations:



Of course, for each possible combination we'll need to state to which color it will lead in the next time step.

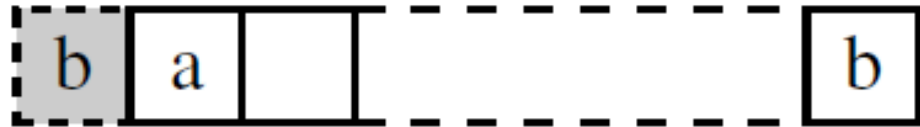
Let us look at rule 254 (we'll get back to why it has this name later).

Rule 254



We can of course apply this rule to the initial condition we had before but what to do at the boundary?

Boundary Conditions



periodic



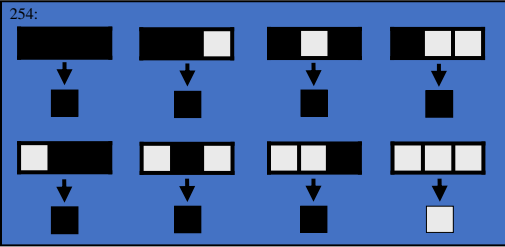
fixed



adiabatic



reflection

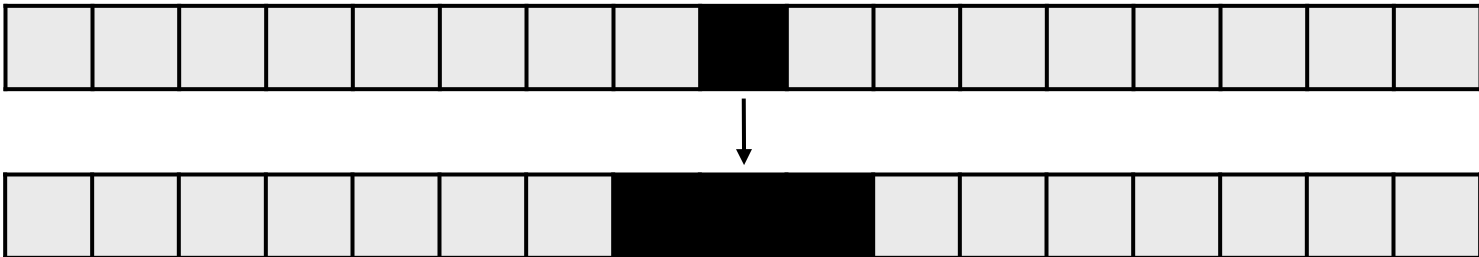


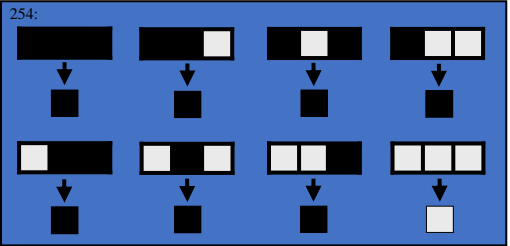
Executing Rule 254

Often one starts with a single black dot and takes all the neighbors on the right and left to be grey (ad infinitum).



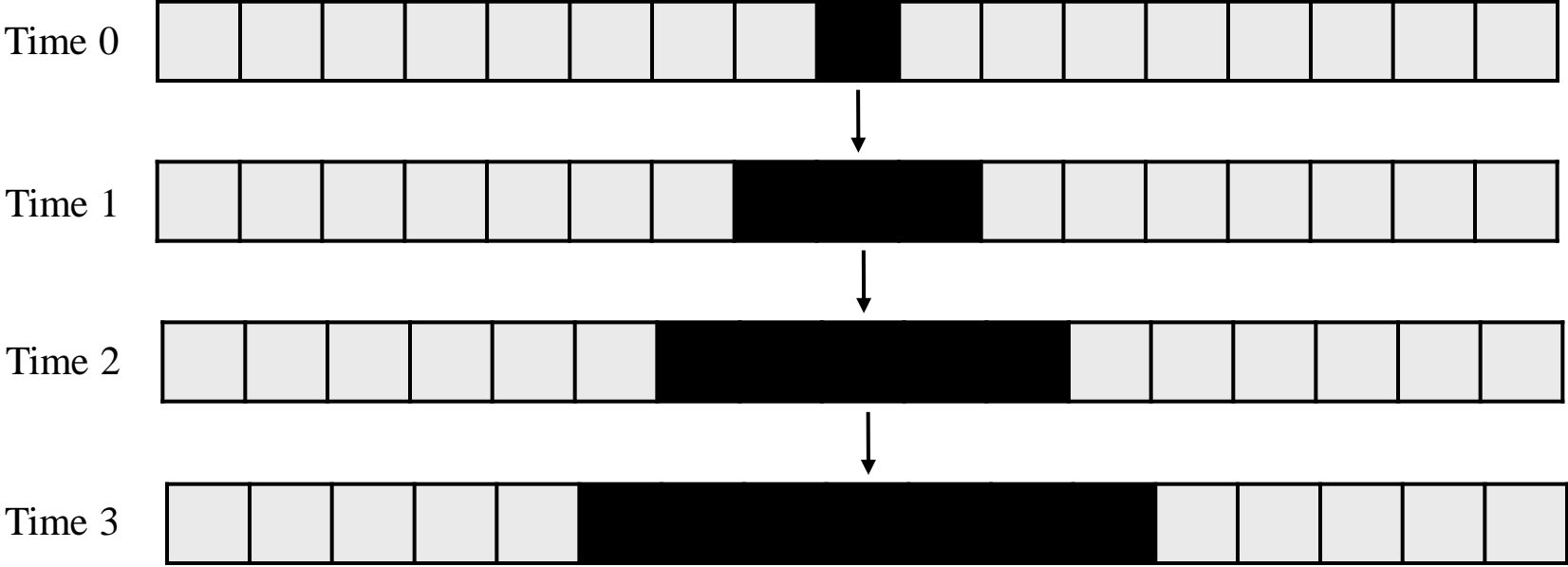
Now, let us apply rule 254. This is quite simple, everything, except for three neighboring grey cells will lead to a black cell.



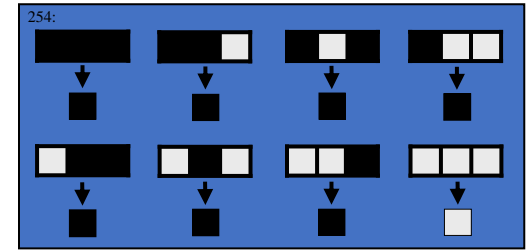


Executing Rule 254

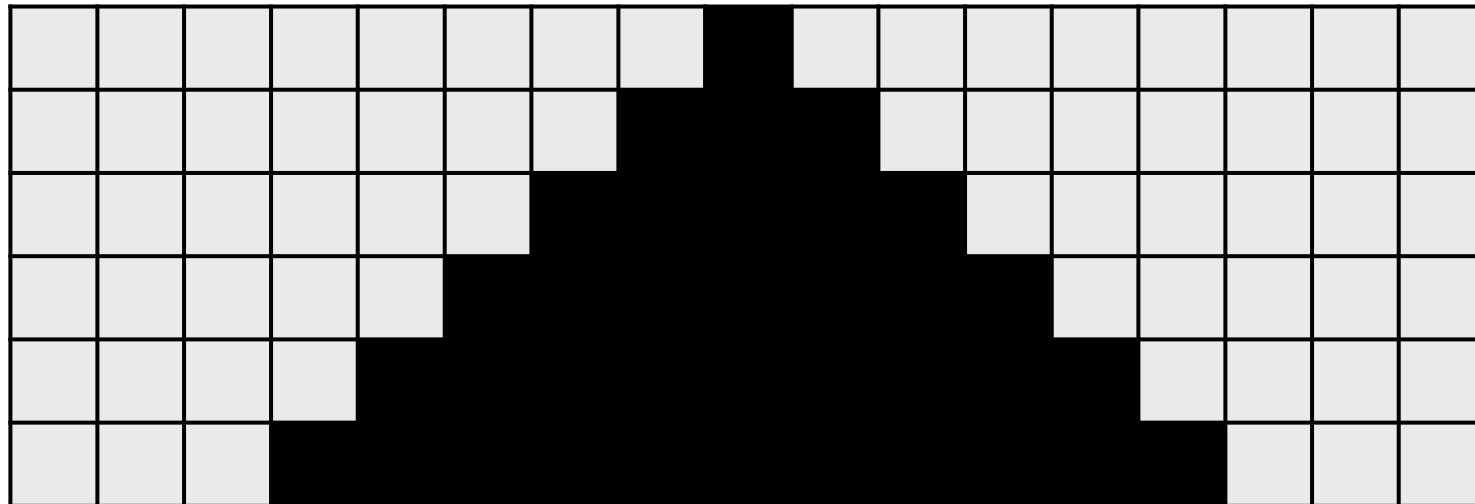
Continuing the procedure:



Executing Rule 254



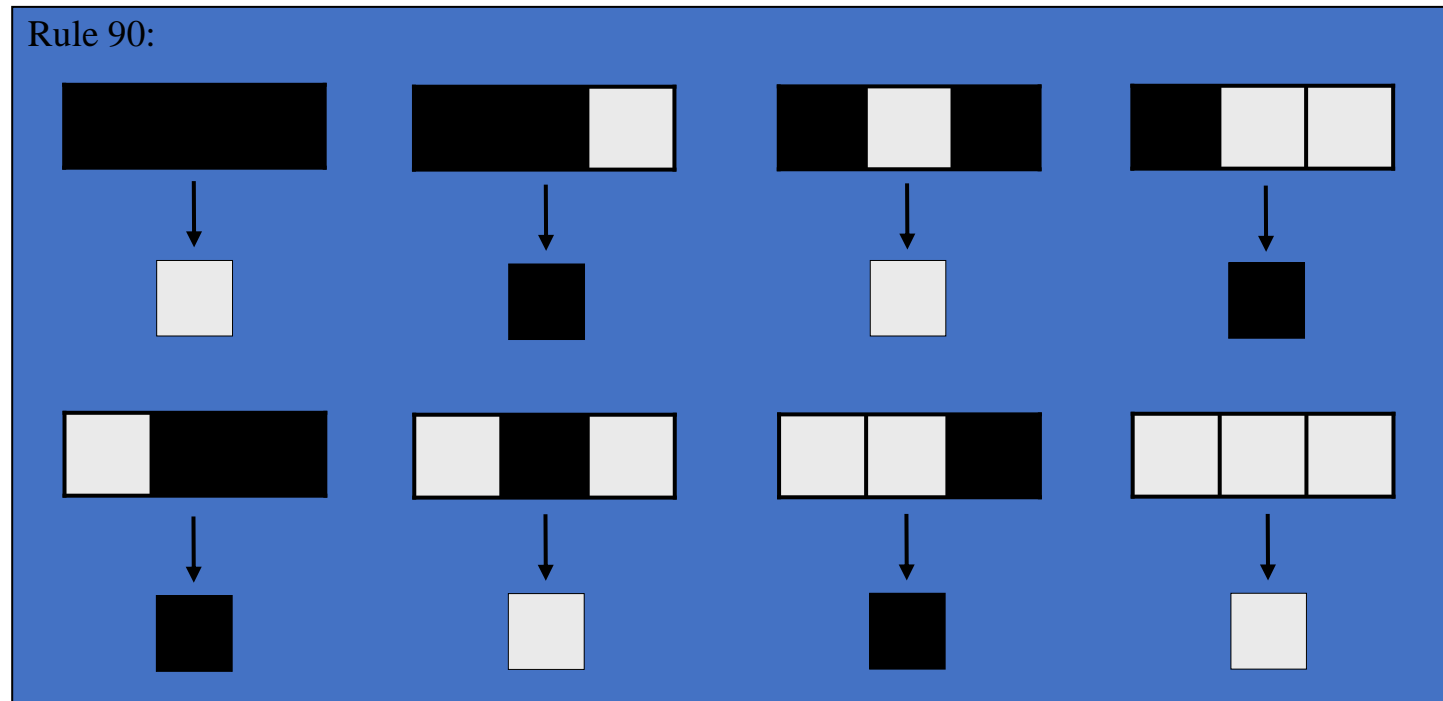
Of course, we don't really need those arrows and the time so we might just as well forget about them to obtain:



Nice, but well ... not very exciting.

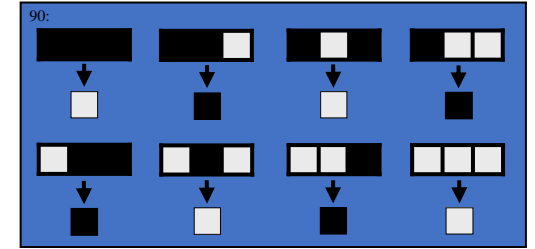
Rule 90

So let us look at another rule. This one is called rule 90.

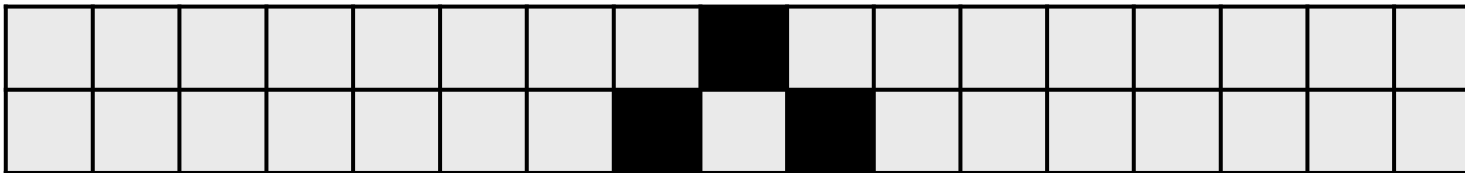


That doesn't look like it's very exciting either. What's the big deal?

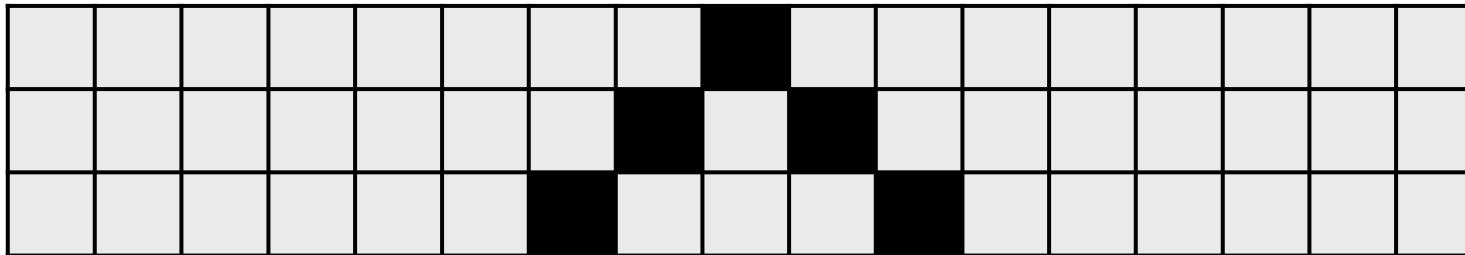
Applying Rule 90



After one time step:

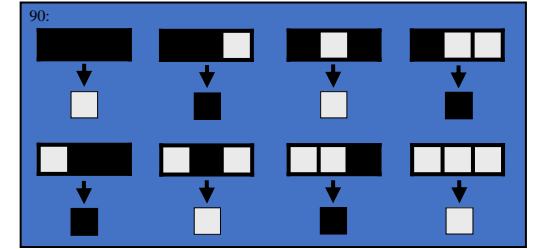


After two time steps:

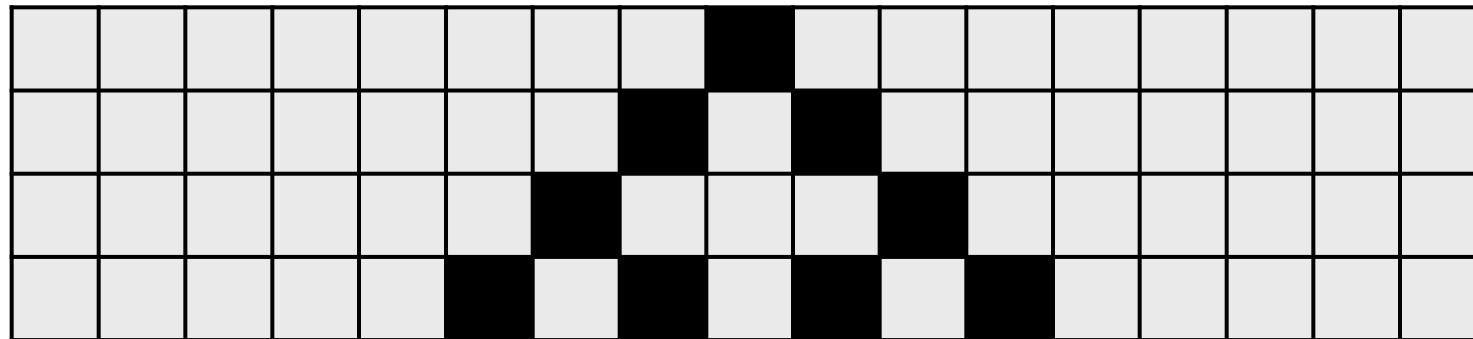


At least it seems to be a bit less boring than before....

Applying Rule 90

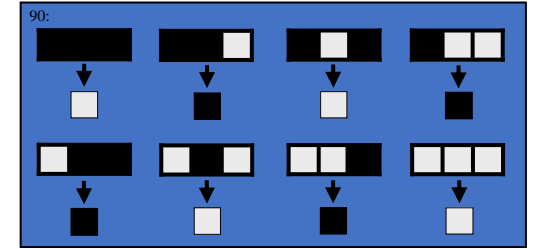


After three time steps:

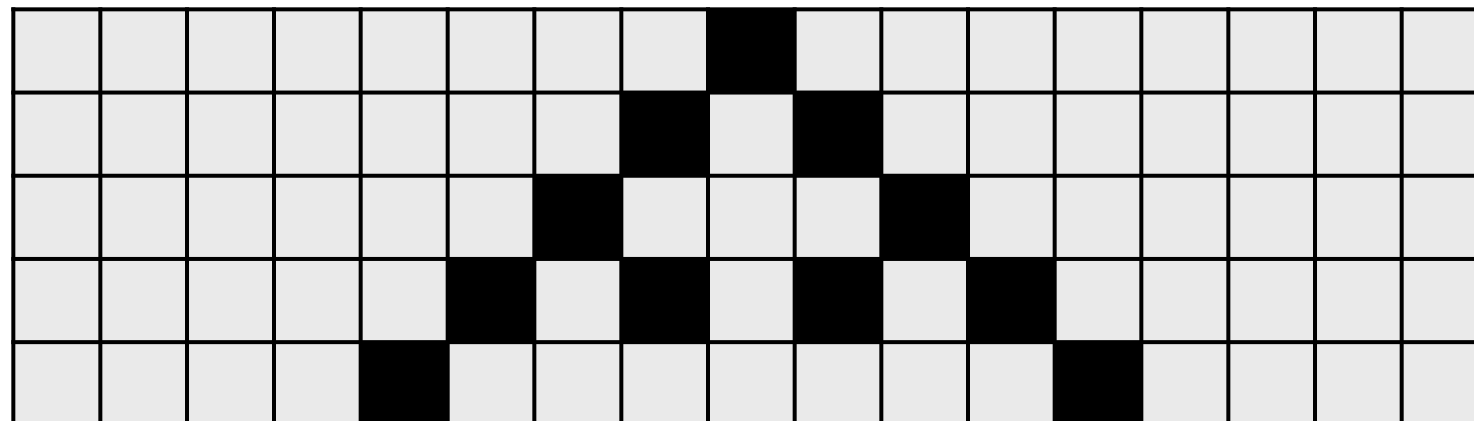


Hey! This is becoming more fun....

Applying Rule 90

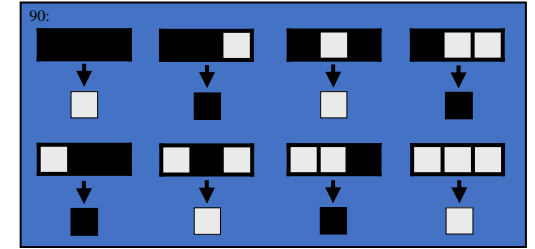


After four time steps:

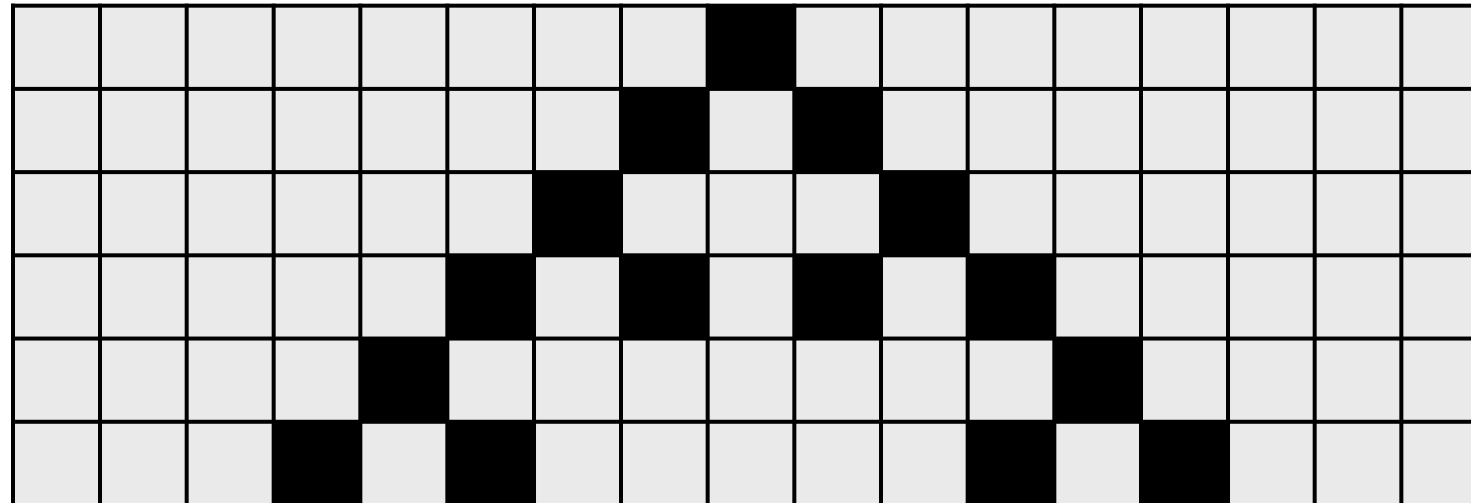


Hmmmm

Applying Rule 90

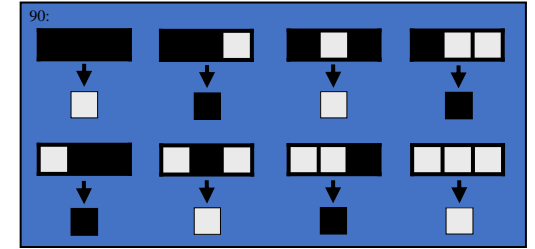


After five time steps:

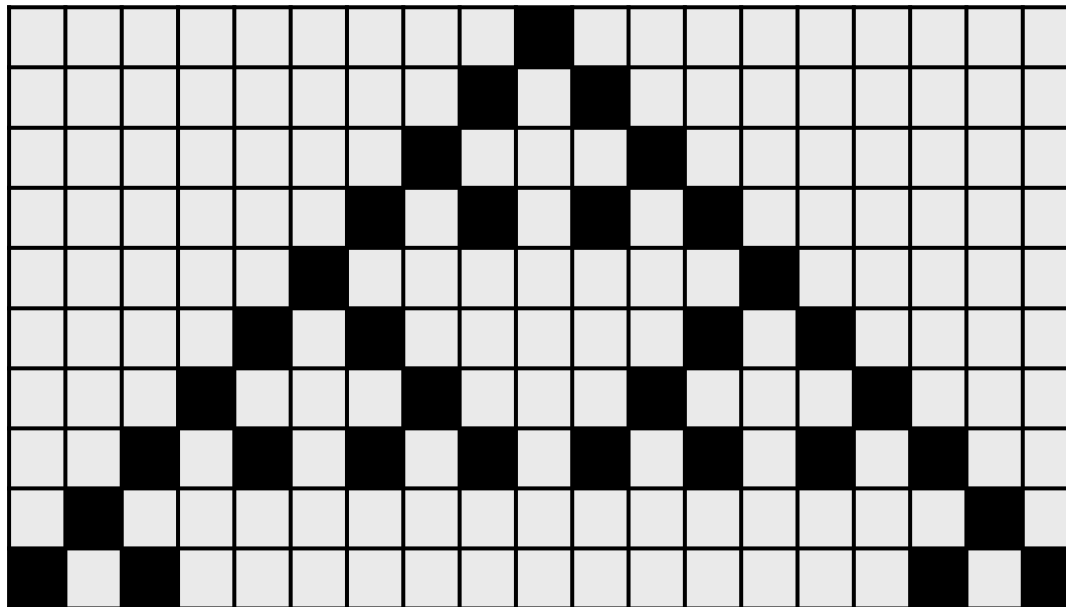


It's a Pac Man!

Applying Rule 90

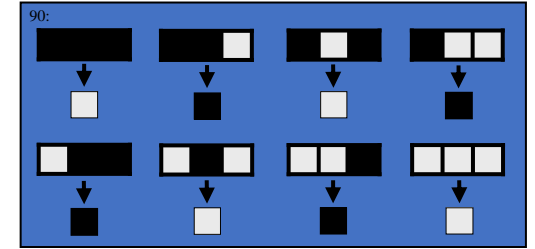


Well not really. It's a Sierpinsky gasket:

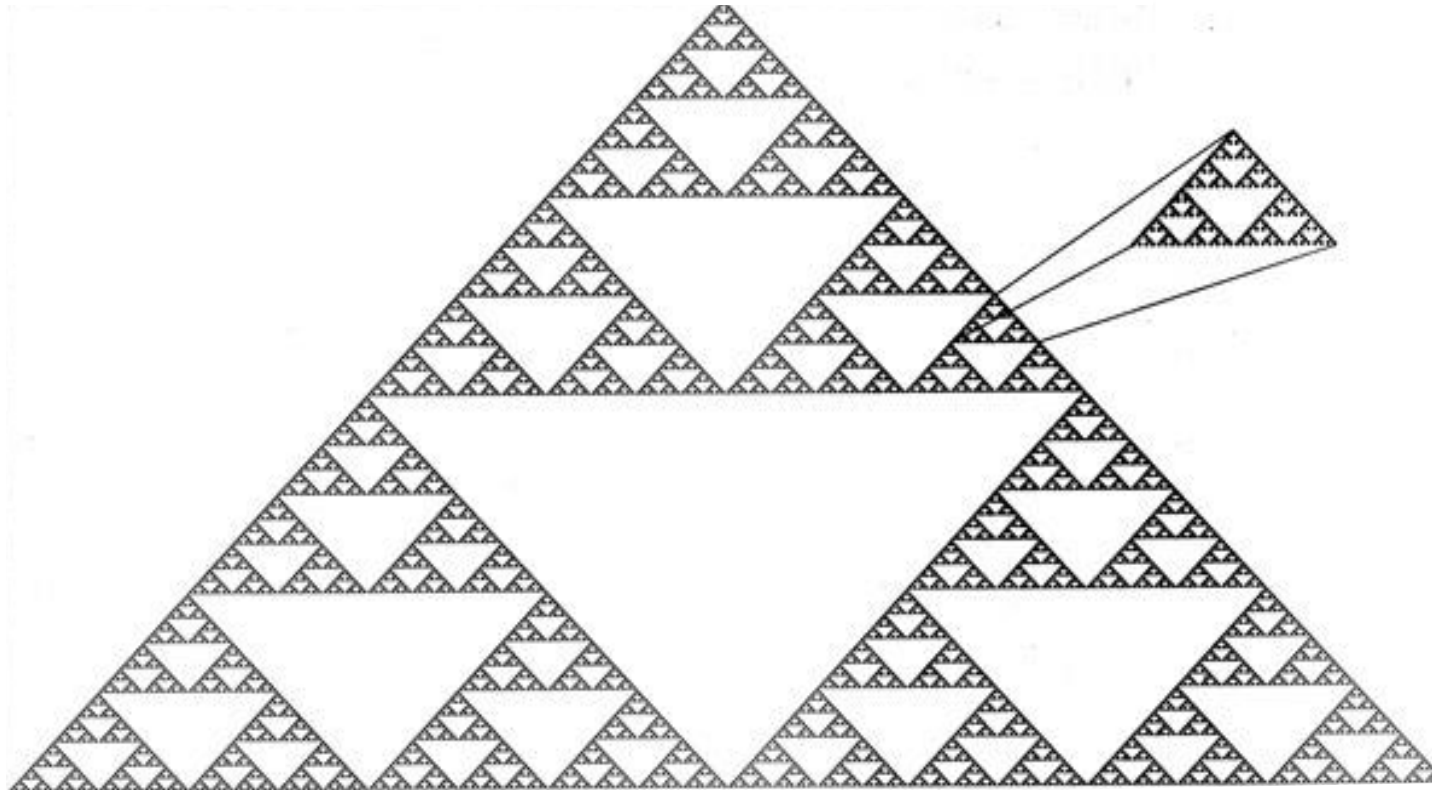


Which is a fractal!

Applying Rule 90



It's a Sierpinsky gasket:



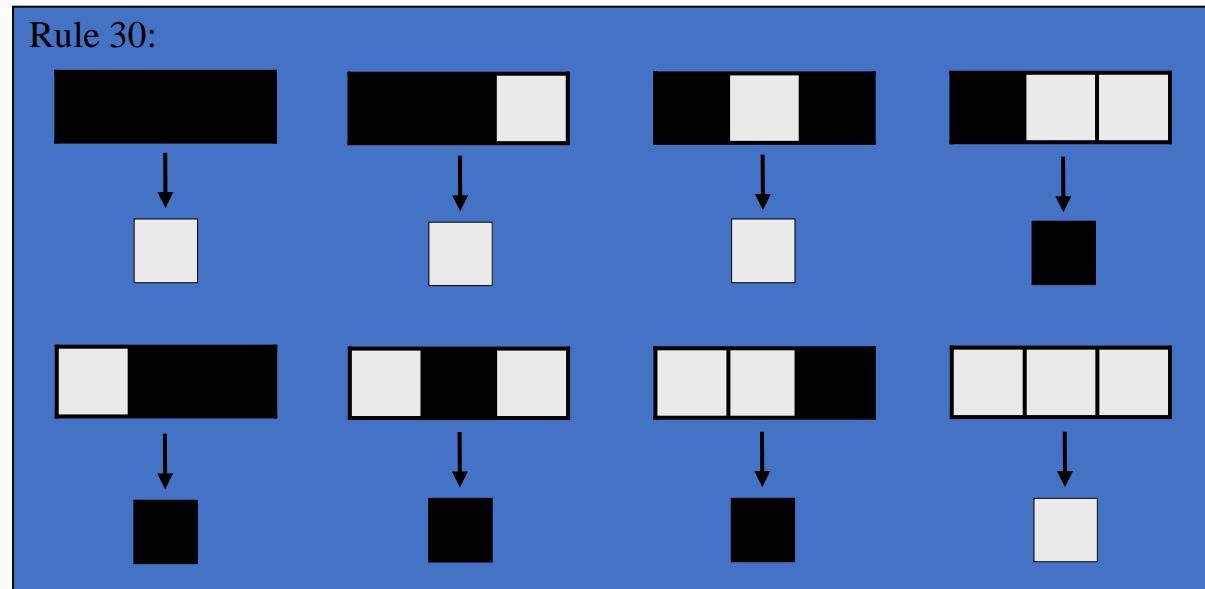
From S. Wolfram: *A new kind of Science*

Rule 30

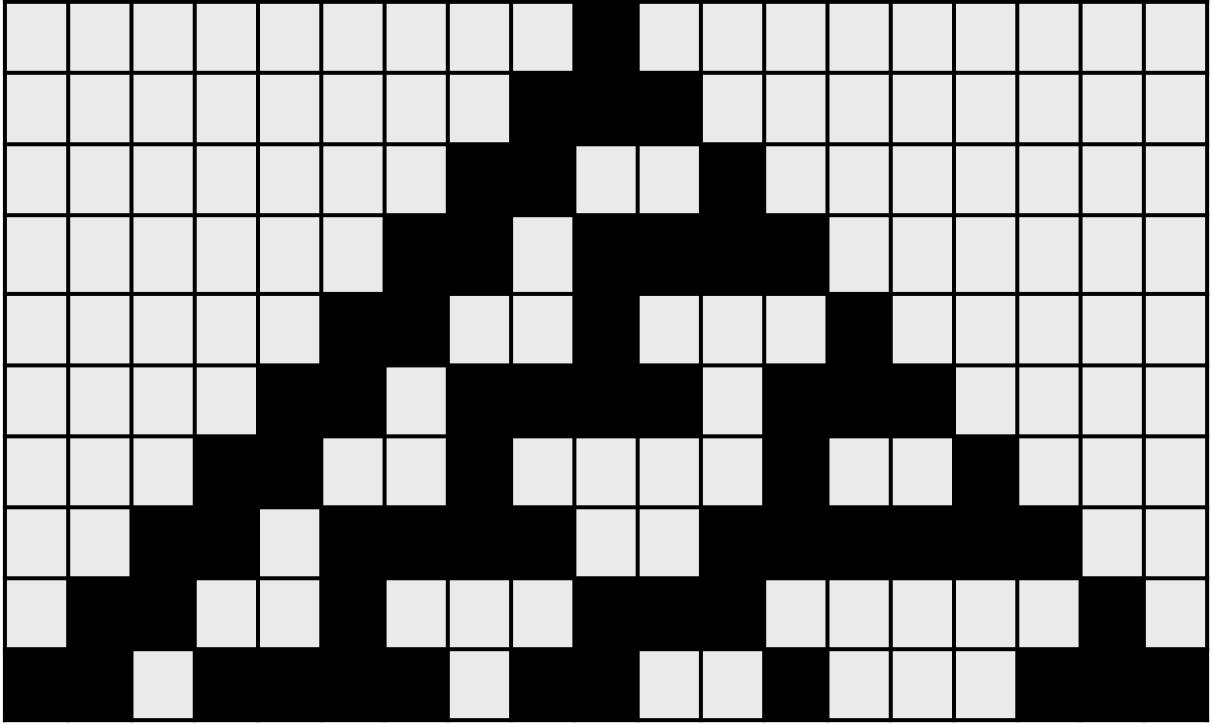
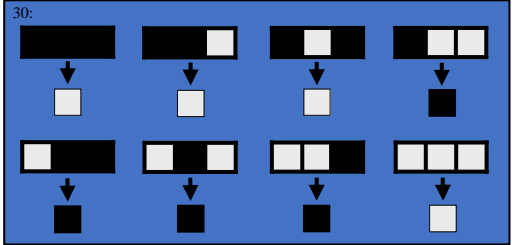
So, we have seen that simple cellular automata can display very simple and fractal behavior. Both these patterns are in a sense highly regular. One may wonder now whether ‘irregular’ patterns can also exist.

Surprisingly they do!

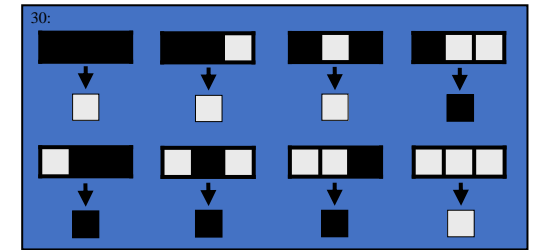
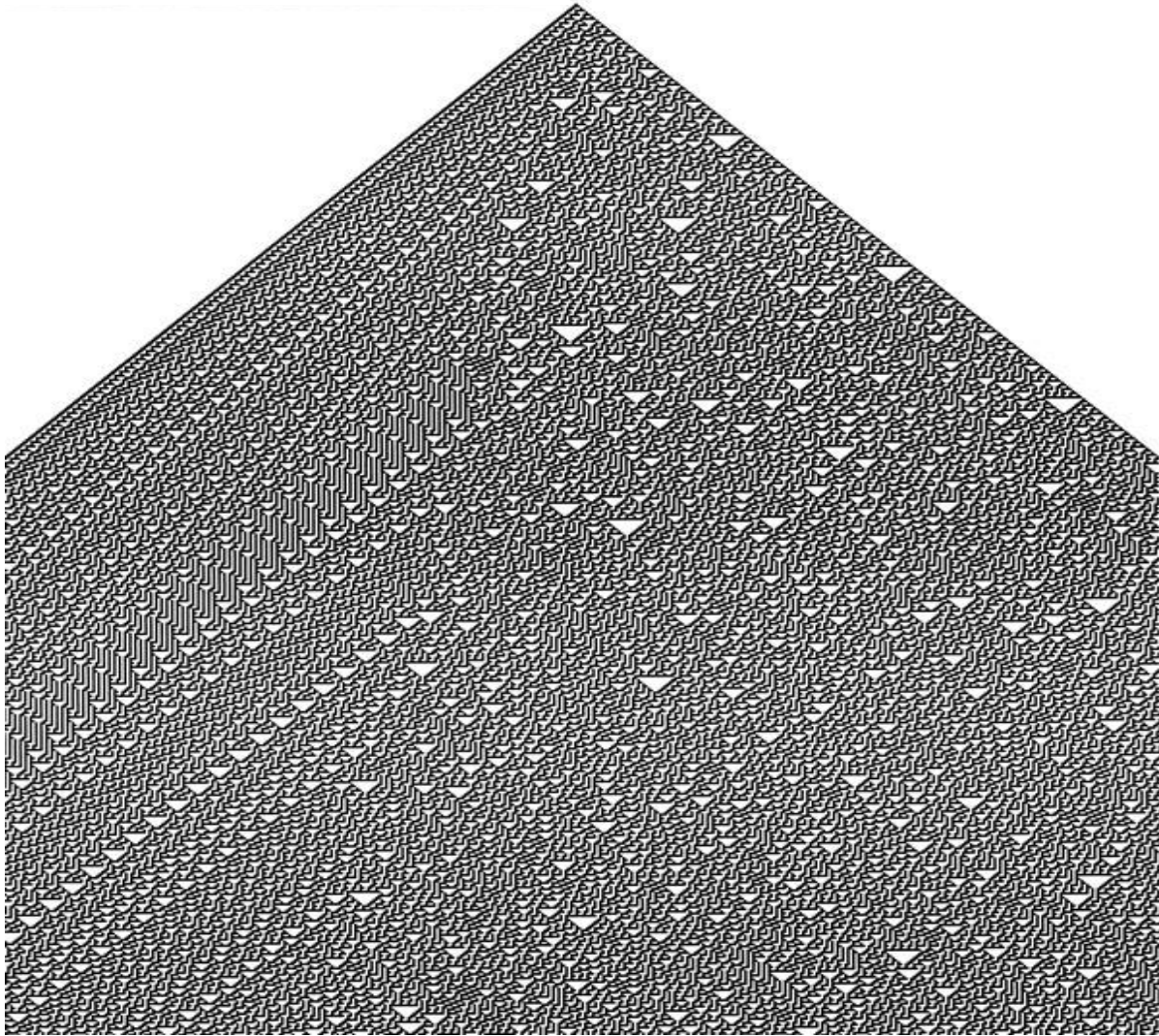
Note that only the color of two boxes has been changed compared to rule 90.



Applying Rule 30



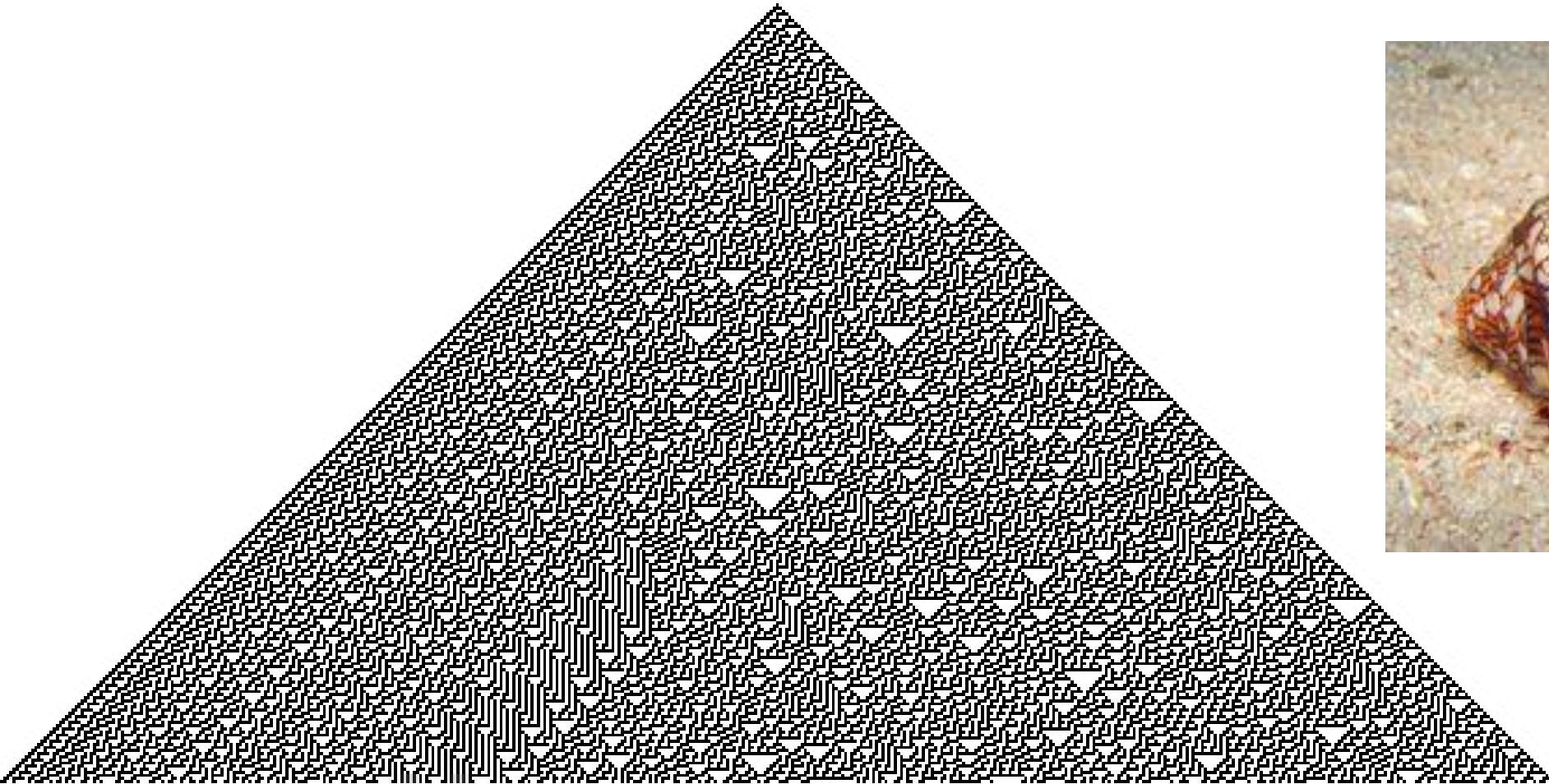
Applying Rule 30



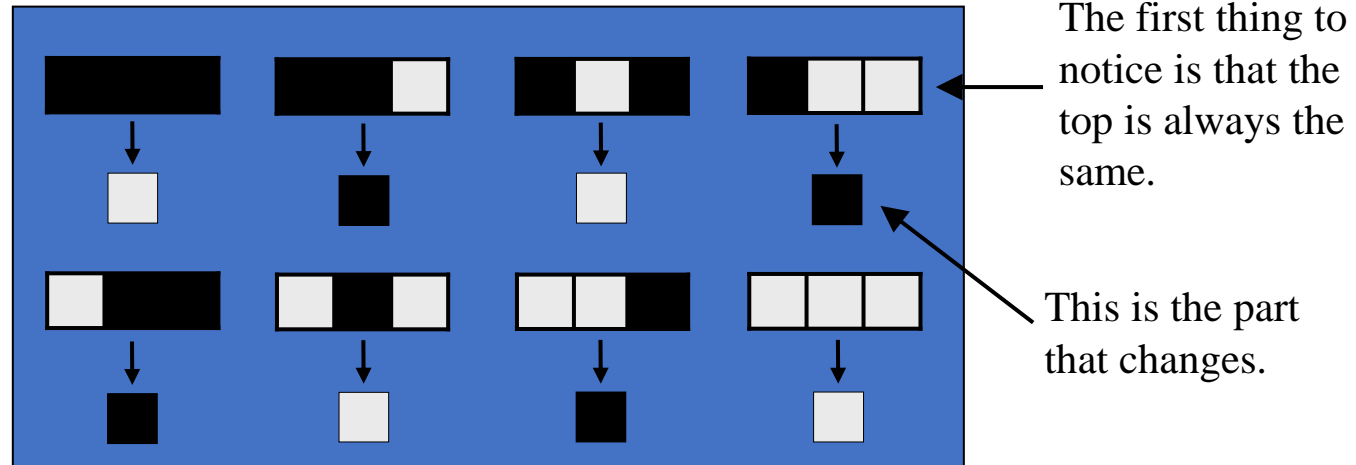
While one side has repetitive patterns, the other side appears random.

From S. Wolfram: *A new kind of Science*

Rule 30 in Nature



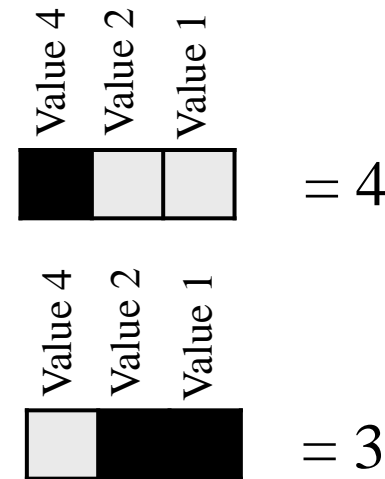
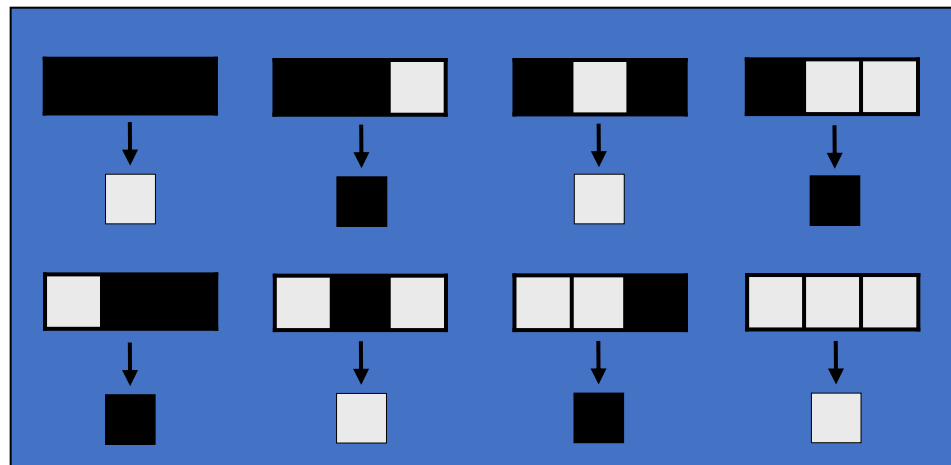
Numbering Scheme



Now if we examine the top more closely, we find that it just is the same pattern sequence that we obtain in binary counting.

Numbering Scheme

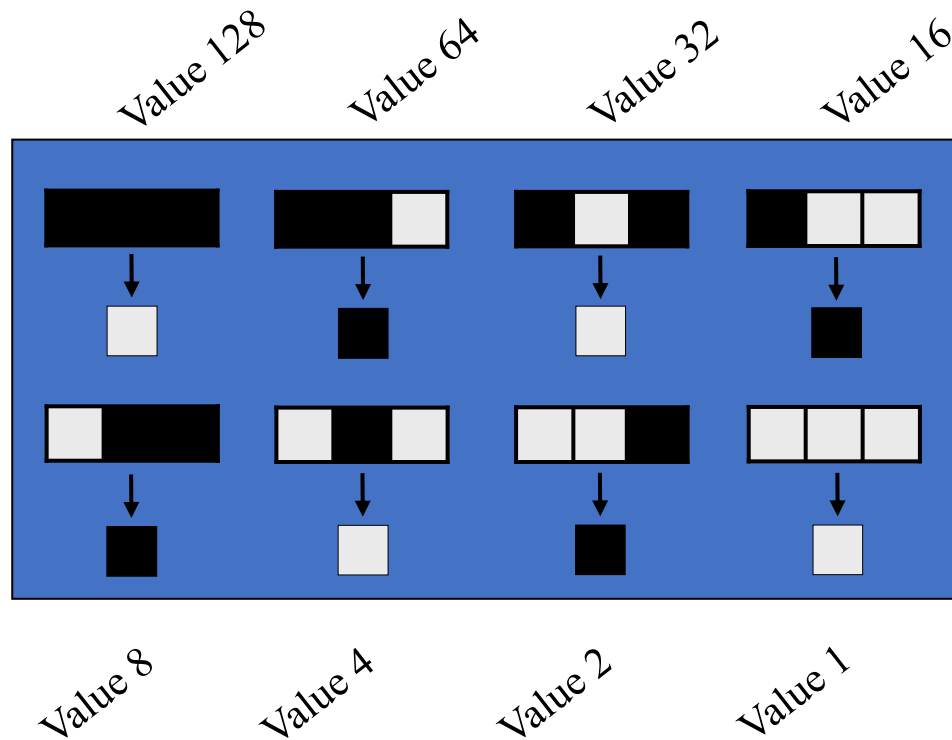
If we say that black is one and grey is zero, then we can see that the top is just counting from 7 to 0.



Good. Now we know how to get the sequence on the top.

Numbering Scheme

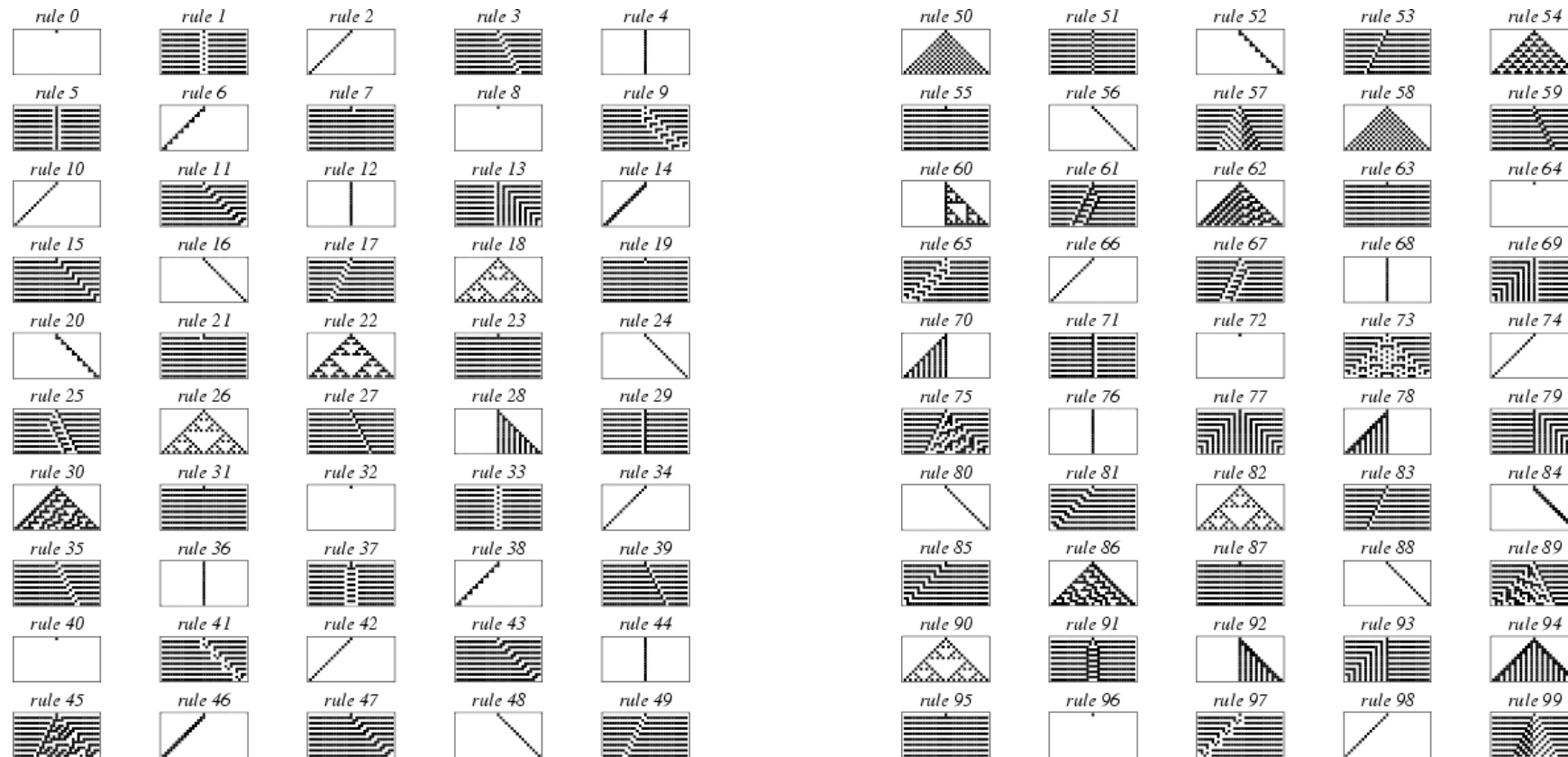
How about the bottom? We can do exactly the same thing but since we have 8 boxes on the bottom it's counting from 0 to 255.



$$= 2+8+16+64 = 90$$

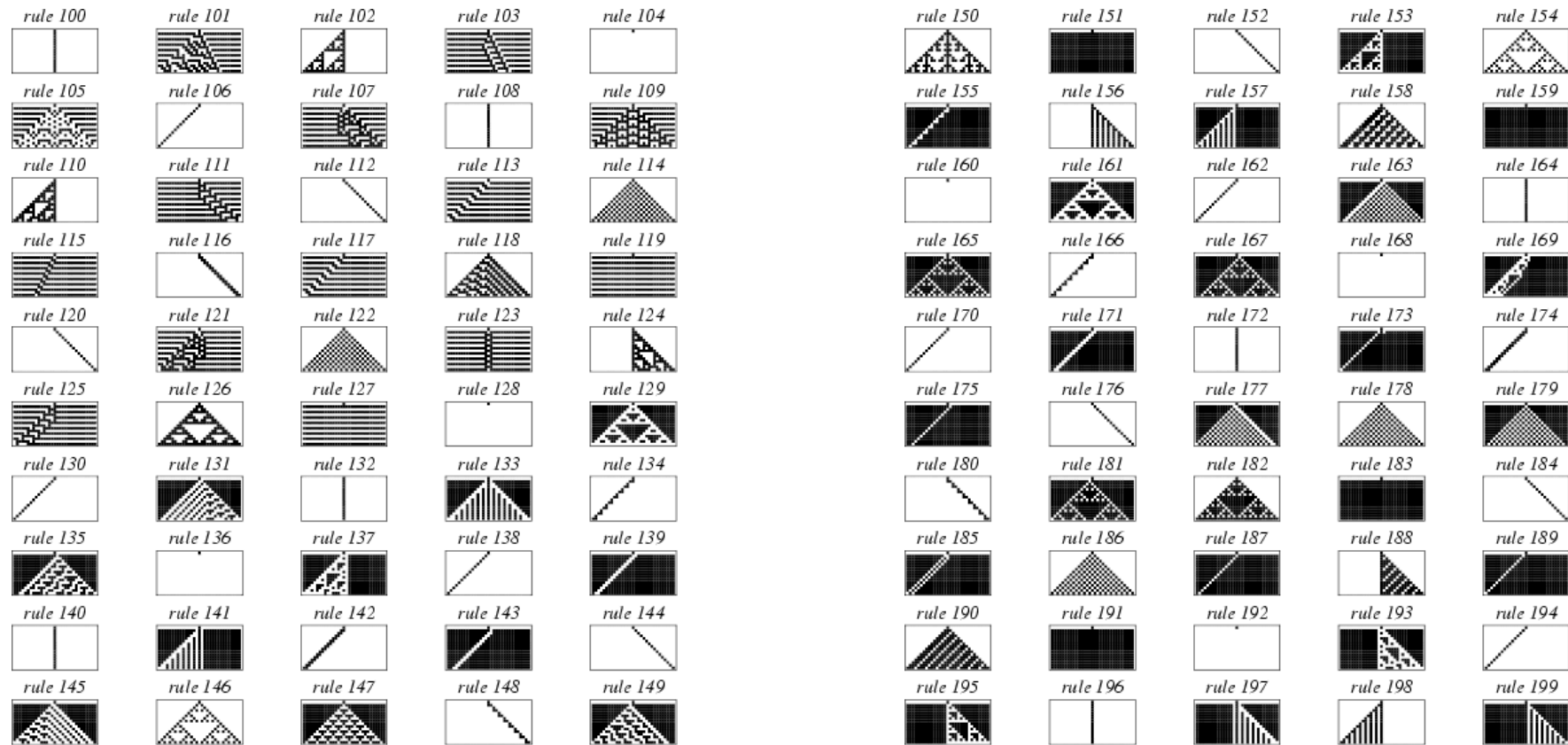
1-d Cellular Automata

Like this we can number all the possible 256 rules for this type of cellular automaton.



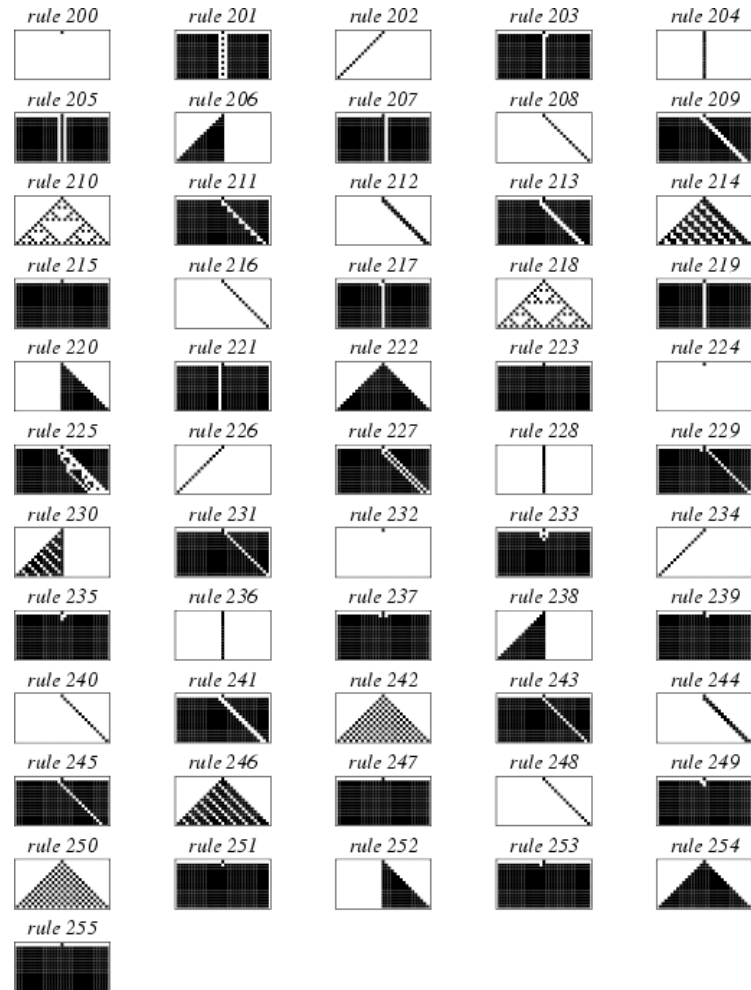
1-d Cellular Automata

Like this we can number all the possible 256 rules for this type of cellular automaton.



1-d Cellular Automata

Like this we can number all the possible 256 rules for this type of cellular automaton.



And of course,

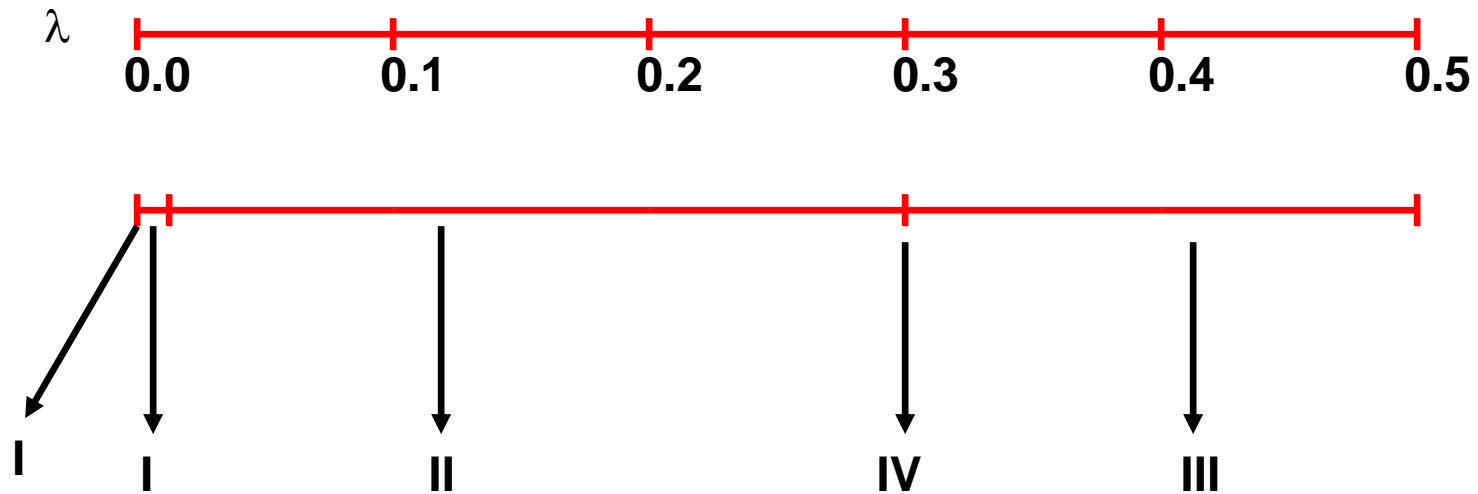
one does not need to restrict oneself to two colors and two neighbors ...

Classifying 1-d CA

- I. Always reaches a state in which all cells are dead or alive
- II. Periodic behavior
- III. Everything occurs “randomly”
- IV. Unstructured but complex behavior

Critical Probabilities?

λ = chance that a cell is alive in the next state



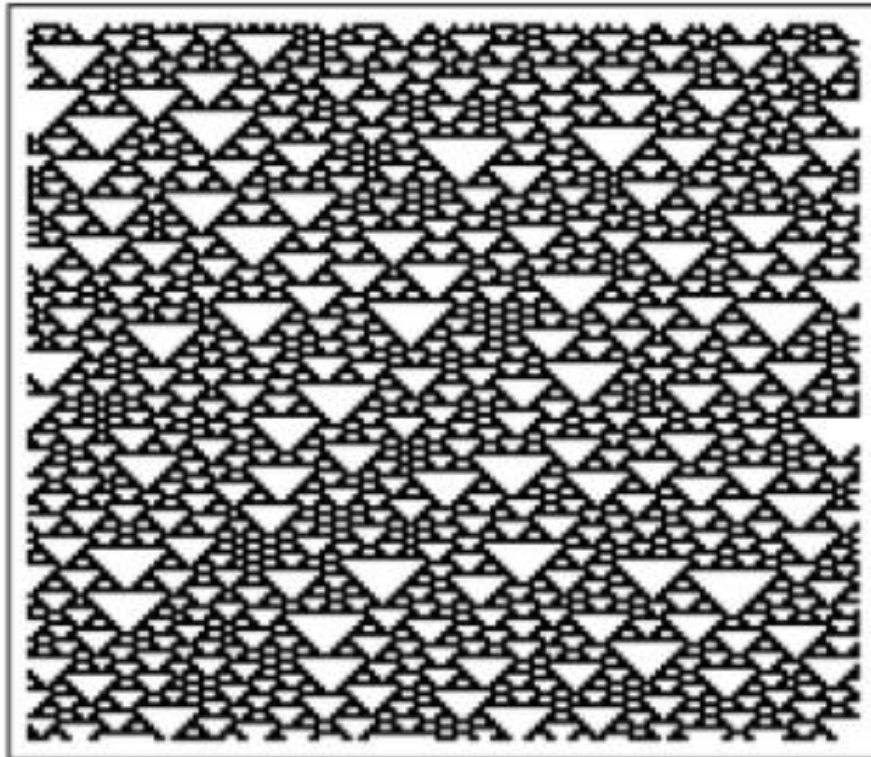
Class I: Very Dull

All Configurations map to a homogeneous state
(e.g., Rules 0, 160)



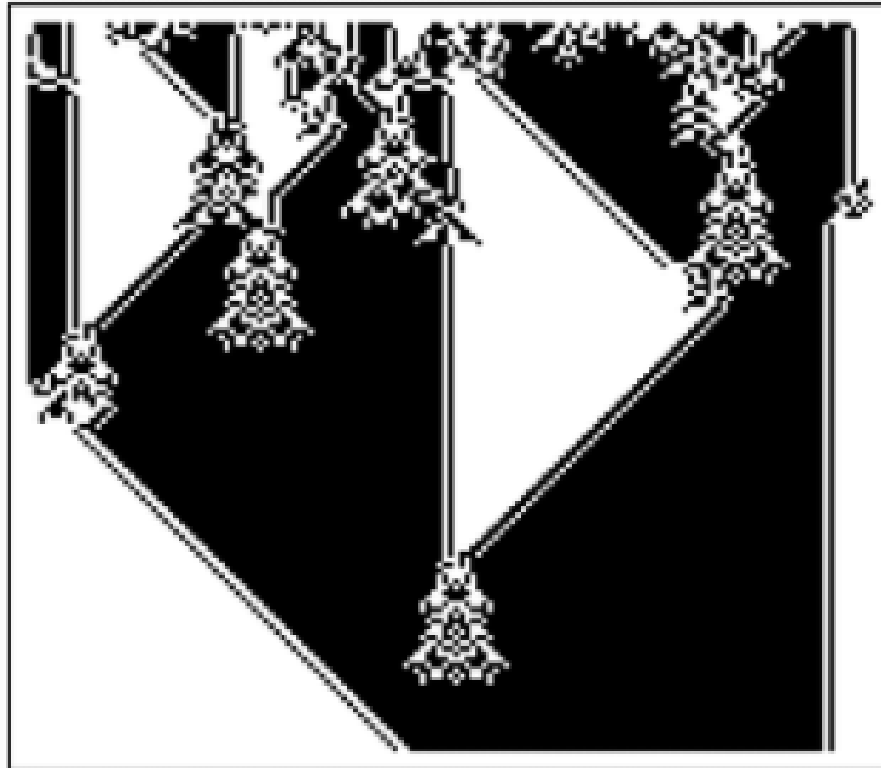
Class III: Interesting

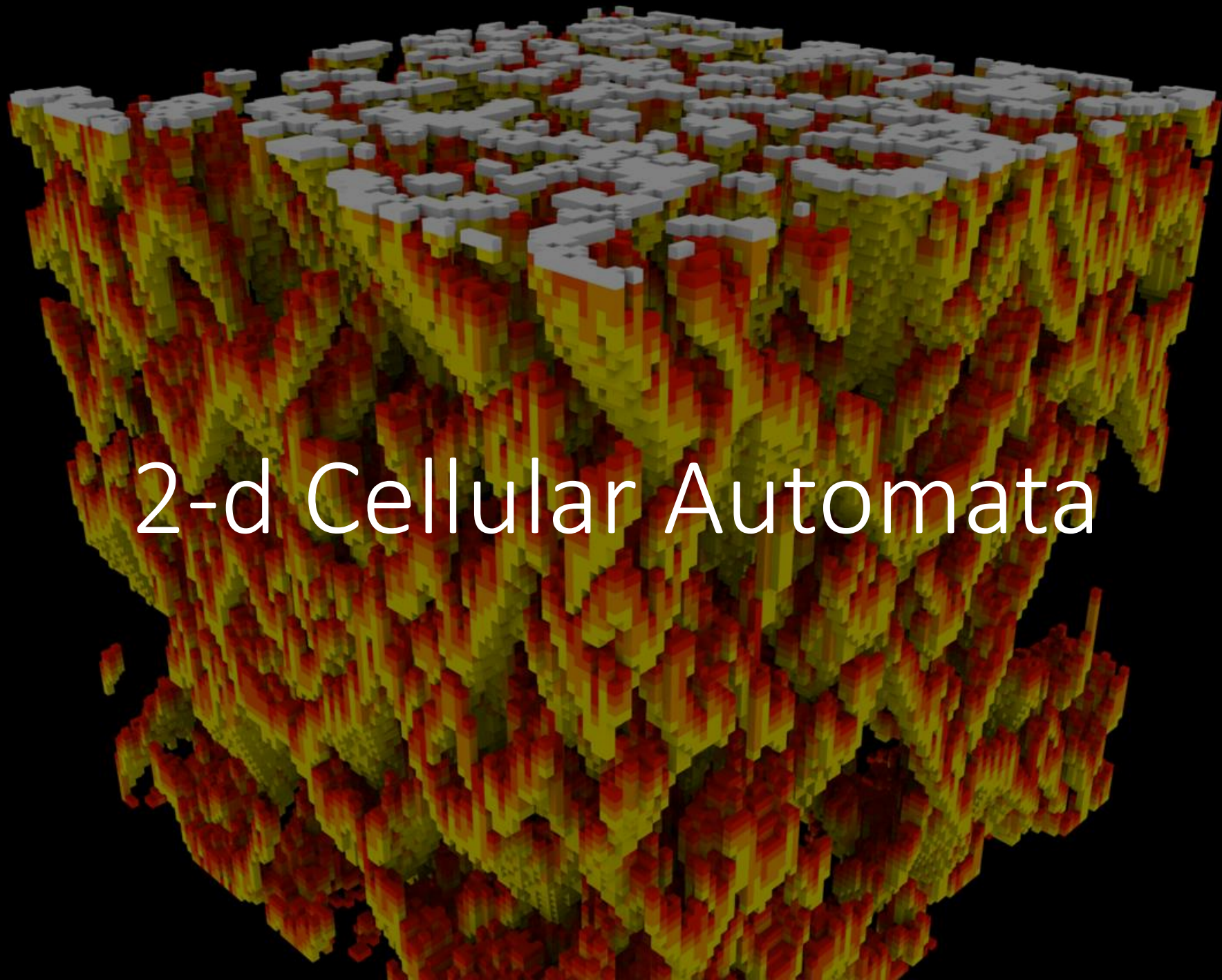
Produces chaotic patterns (impossible to predict long time behavior) (e.g., Rules 30, 90)



Class IV: Very Interesting

Produces propagating structures, may be used in computations (Rule 110)





2-d Cellular Automata

Example of a Cellular Automaton: VOTE

- Vote is an example of the simplest possible kind of eight-neighbor CA.
- Vote is so simple because:
 - (1) Vote is a "one-bit rule" and,
 - (2) Vote is "totalistic."

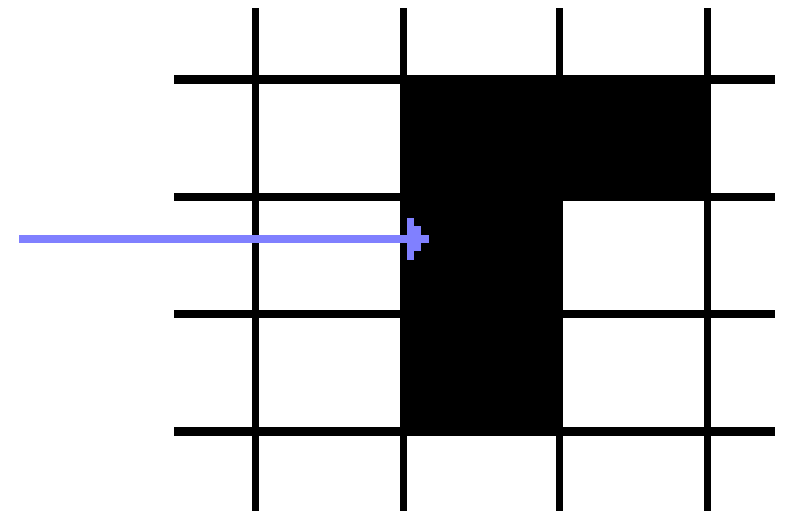
Sums

- **NineSums:** The NineSum for a cell (C) is the sum of 1's in all the surrounding cells (**neighbors including** cell (C)).
- **EightSum:** EightSum for a cell (C) is the sum of 1's in all the surrounding cells (**neighbors excluding** cell (C)).

Example

- In this example, each cell can be in either 0 or 1 state.
- Cell C has 8 neighbors, 3 of them are in state 1,
 - Then the EightSum for cell C is 3, NineSum is 4.

Cell (C)



One-bit Rule

Vote is a one-bit rule.

- The cells of Vote have only two possible states: **on** or **off**, zero or one.
- Choosing between two options requires one bit of information, and this is why we call Vote a **one-bit rule**.

Totalistic

Vote is **totalistic**.

- A totalistic rule updates a cell C by forming the EightSum of the eight neighbors, adding in the value of C itself to get the full NineSum, and then determining the cell's new value strictly on the basis of where the NineSum lies in the range of ten possibilities 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
- Under a **totalistic rule**, a cell's next state depends only on the total number of bits in its nine-cell neighborhood.

Vote

The idea behind Vote's rule is that if most cells in your neighborhood are 1, then you go to 1, and if most cells in your neighborhood are 0, then you go to 0.

What do we mean by "most cells in your neighborhood?"

Since there are nine cells in your neighborhood, the most obvious interpretation is to assume that "most" means "five or more".

Majority (copy what neighbors do): 992										
NineSum	0	1	2	3	4	5	6	7	8	9
NewState	0	0	0	0	0	1	1	1	1	1

Biased Majority (almost copy what neighbors do): 976										
NineSum	0	1	2	3	4	5	6	7	8	9
NewState	0	0	0	0	1	0	1	1	1	1

Implementation Issues

On the fly:

$$s_{ij}(t + 1) = s_{i-1,j}(t) \odot s_{i+1,j}(t) \odot s_{i,j-1}(t) \odot s_{i,j+1}(t)$$

Using a look-up table:

$$\begin{aligned} index &= 2^0 s_{i-1,j}(t) + 2^1 s_{i+1,j}(t) + 2^2 s_{i,j-1}(t) + 2^3 s_{i,j+1}(t) \\ s_{ij}(t + 1) &= \text{Rule}[index] \end{aligned}$$

A Small Diversion

History of CAs

Origins of CA

- 1940s: J. von Neuman and S. Ulam
- Design a better computer with self-repair and self-correction mechanisms
- Simpler Problem: Finding a mechanism for self-reproduction
 - (before the discovery of DNA) Devise an algorithmic scheme
 - Formalize in a discrete space
 - Automaton with 29 states, arrangements of thousand of cells that self-reproduce
 - Universal Computer

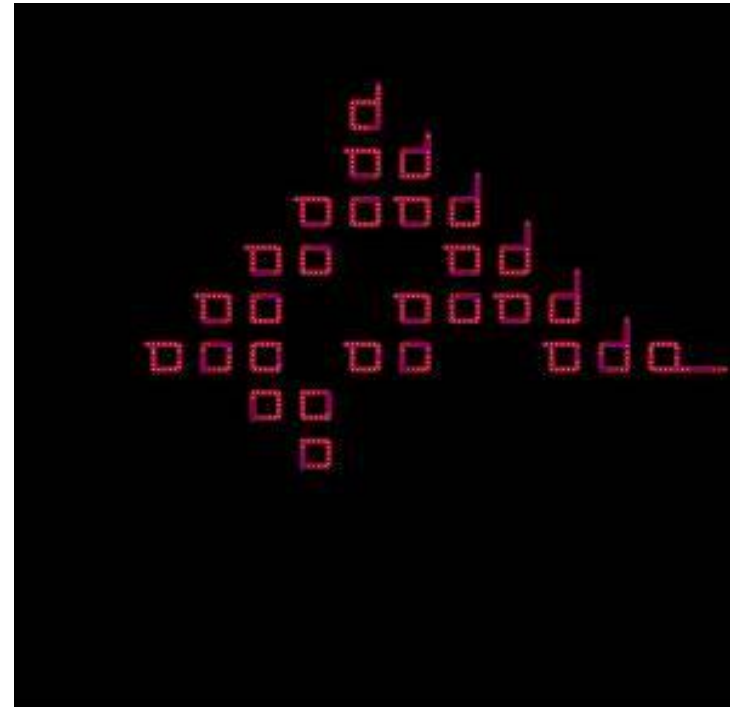
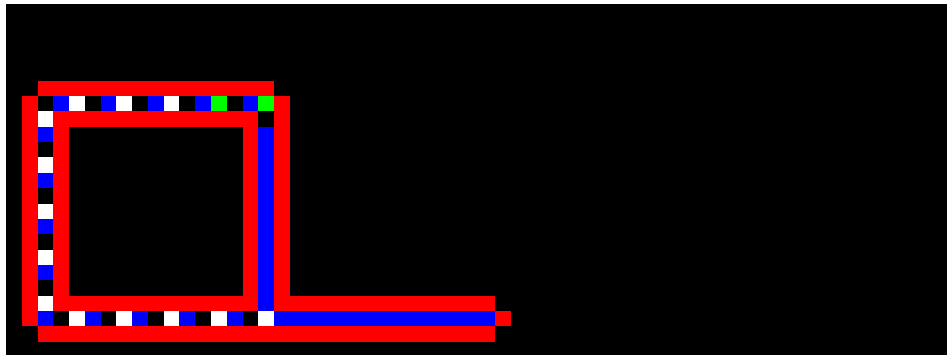
A Simpler CA that Self-Reproduces: Langton CA

- Simplified version: 8 states
- Not a universal computer
- Structures that contain their own fabrication recipe

- This is not a biological model, but an algorithmic abstraction
- Reproduction from a mechanistic point of view (energy and matter is needed)
- No need for hierarchical structure that the more complicated builds the less complicated
- Evolving hardware

Langton's Loops

Chris Langton formulated a much simpler form of self-rep structure - Langton's loops - with only a few different states, and only small starting structures.





Forest Fire

A Small Touch on Percolation Theory

Rules – 2d Square Lattice (4-neighborhood)

In each tick:

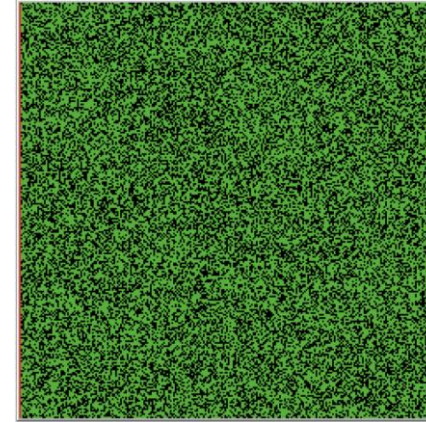
1. A burning tree becomes an empty site.
2. A green tree becomes a burning tree if at least one of its nearest neighbors is burning.
3. At an empty site, a tree grows with probability q .
4. A tree without a burning nearest neighbor becomes a burning tree during one time step with probability f (lightning).

Let $q = 0$ and $f = 0$

(no grow – no lightning – small time scale)

Initialization:

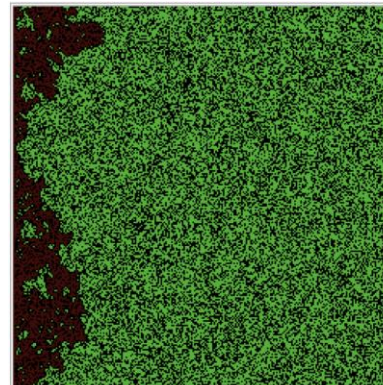
- Each cell with probability p will be filled with a tree, otherwise empty.
- First row of trees is ignited.



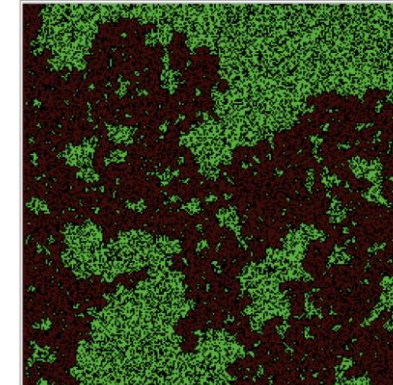
Questions:

Will the fire reach the last row? How many trees will it burn?

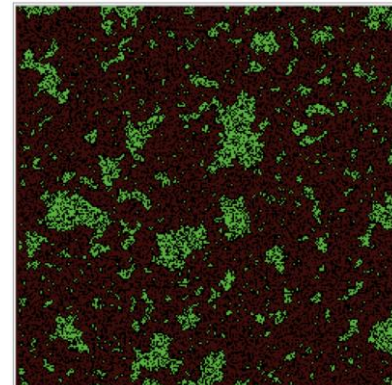
$p = 0,58$



$p = 0,59$



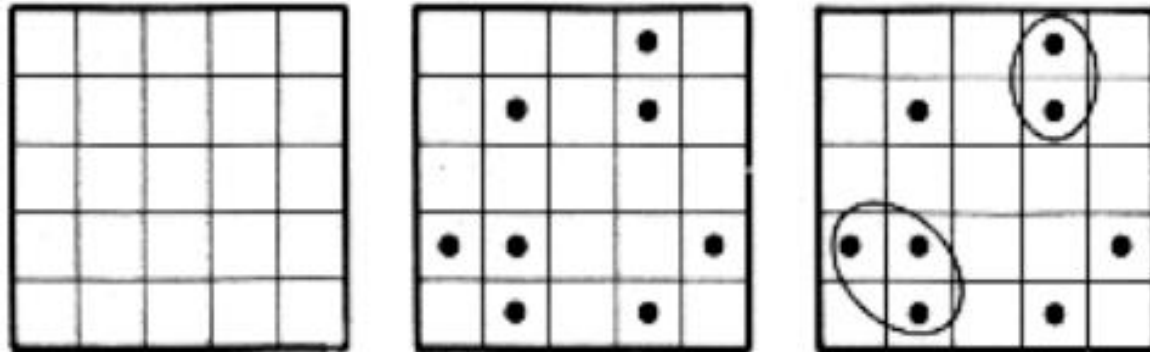
$p = 0,6$



What is Percolation Theory?

Start with an empty lattice - then occupy sites at random

- Connected occupied sites form **clusters**
- Percolation is about the properties of these clusters -- size, connectivity, etc.



Connectivity on a Square Lattice

- Connectivity depends on concentration of occupied sites p
- Connectivity changes at p_c (≈ 0.59 for site percolation on a square lattice)

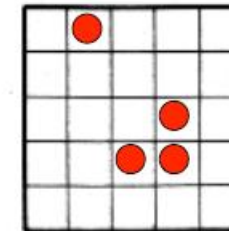
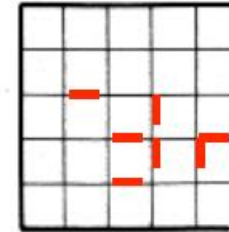
p_c is the “critical” concentration for percolation

- A “connectivity” phase transition occurs at $p_c \sim 0.59$
- A **spanning cluster** first appears at p_c
- Many properties are singular at p_c



p_c Depends on Lattice Type

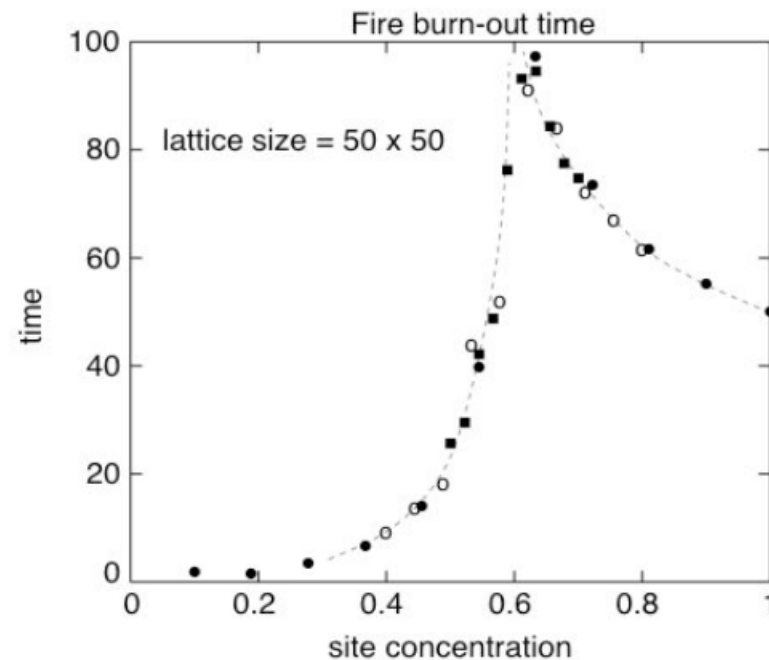
Lattice	Site	Bond
Honeycomb	0.6962	0.65271
Square	0.592746	0.50000
Triangular	0.500000	0.34729
Diamond	0.43	0.388
Simple cubic	0.3116	0.2488
BCC	0.246	0.1803
FCC	0.198	0.119
$d = 4$ hypercubic	0.197	0.1601
$d = 5$ hypercubic	0.141	0.1182
$d = 6$ hypercubic	0.107	0.0942
$d = 7$ hypercubic	0.089	0.0787



Let's Get Back to Forest Fire

The burn-out time diverges at p_c !

- An example of singular behavior at the percolation transition
- Singularity is due to the connectivity of the infinite cluster at p_c



Infinite????

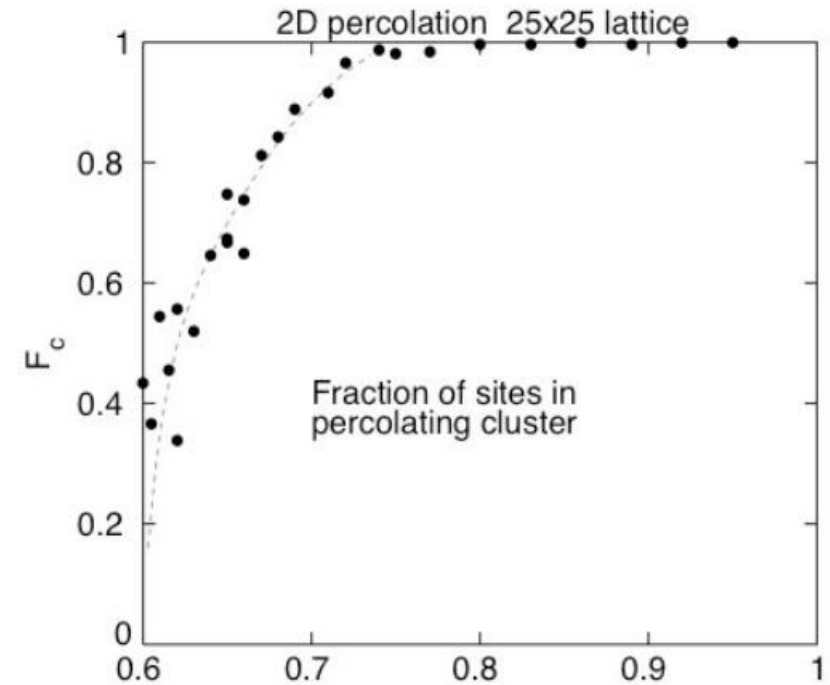
In theory we assume an infinite space to reason for CA. Of course, experimentation is conducted on finite spaces.

Strange Properties at p_c

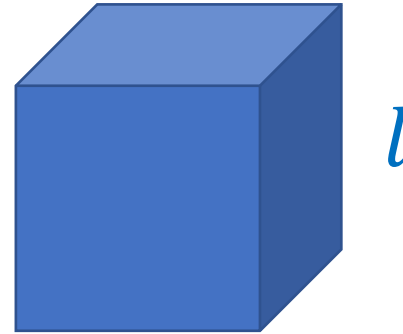
- The spanning cluster is **infinite** (since it spans the system) but contains a **vanishing fraction** of the occupied sites!
- Forms a **fractal**

Focus on just the spanning (critical) cluster at p_c

- Remove all sites that are not part of the infinite cluster
- The spanning cluster contains large holes
- Need a way to describe the geometry of this cluster



Finding the Dimensions of 1-d CA



Dimension for geometric objects: scaling behavior

- Square of edge length l : $Area = l^2$,
Dimension = exponent w.r.t. length = 2
- Cube of edge length l : $Volume = l^3$,
Dimension = exponent w.r.t. length = 3

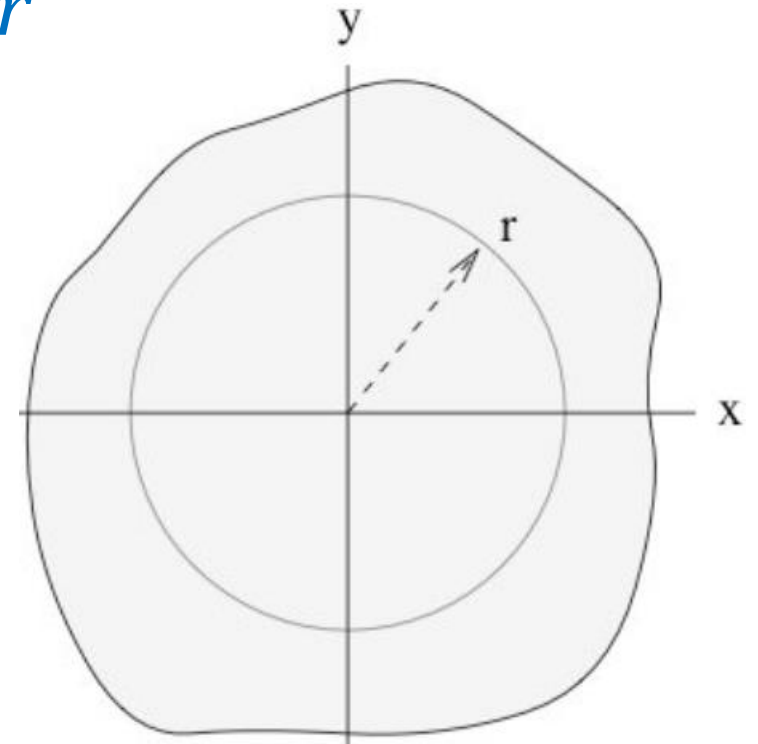
Fractal Dimensionality of a Cluster

Consider how the mass m varies with radius r

- m varies as a power law

$$m(r) \sim r^{d_f}$$

- Exponent $d_f \sim r^2$ for a “regular” 2-d cluster
- $d_f < 2$ for the spanning cluster at p_c
- *fractal cluster*



Fractal Dimension at Percolation Threshold

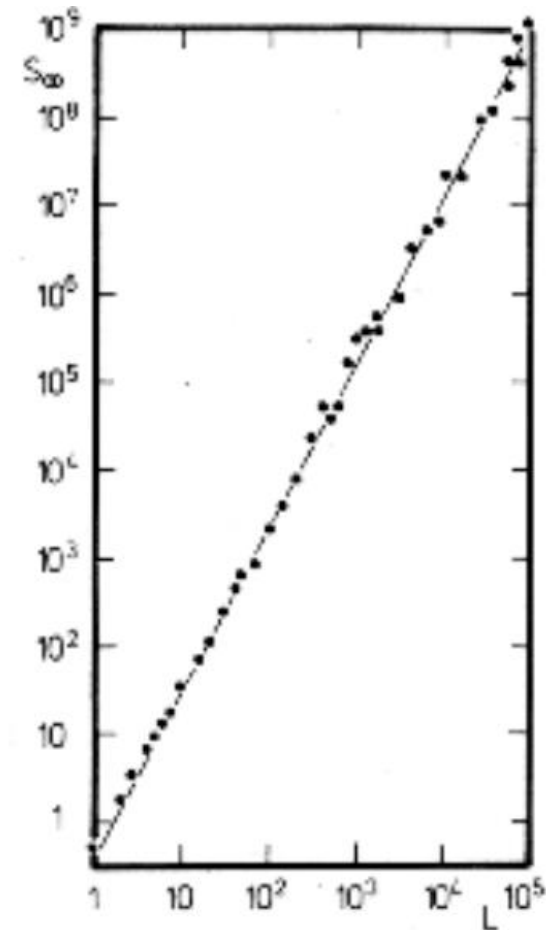
mass (m) of largest cluster as a function of lattice size (L)

$$m \sim r^{d_f}$$

$$d_f = \frac{91}{48} \approx 1.90$$

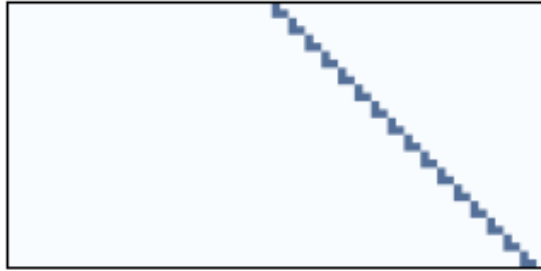
What makes a fractal cluster different?

- Just having holes and cracks is not enough
- Presence of “holes” and “cracks” on **all** length scales

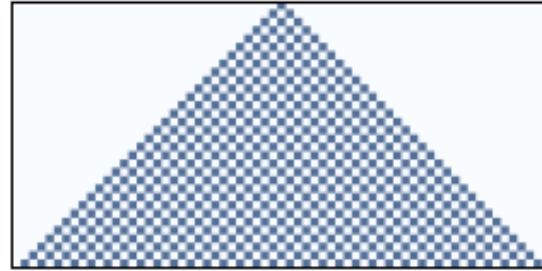


Finding the Dimensions of 1-d CA

Rule 20



Rule 50

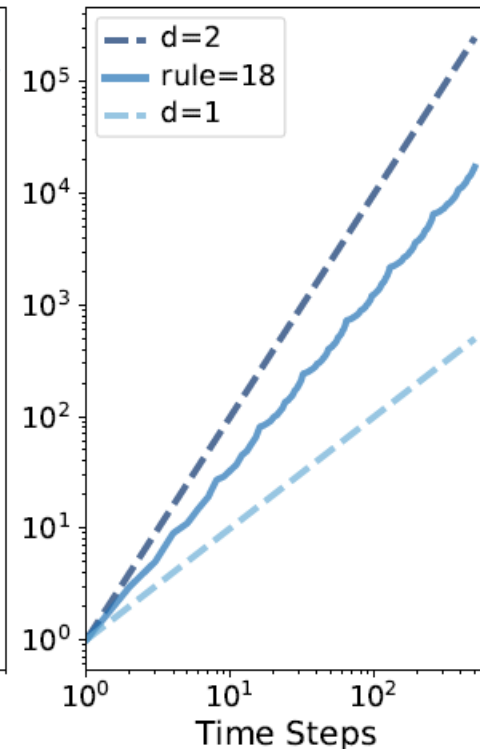
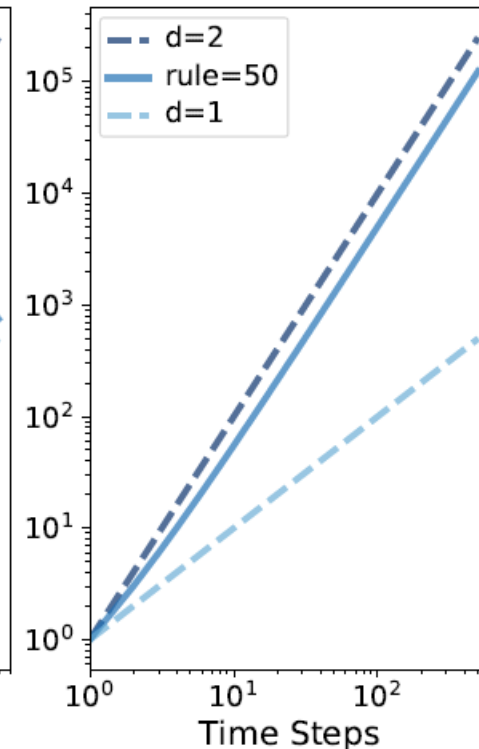
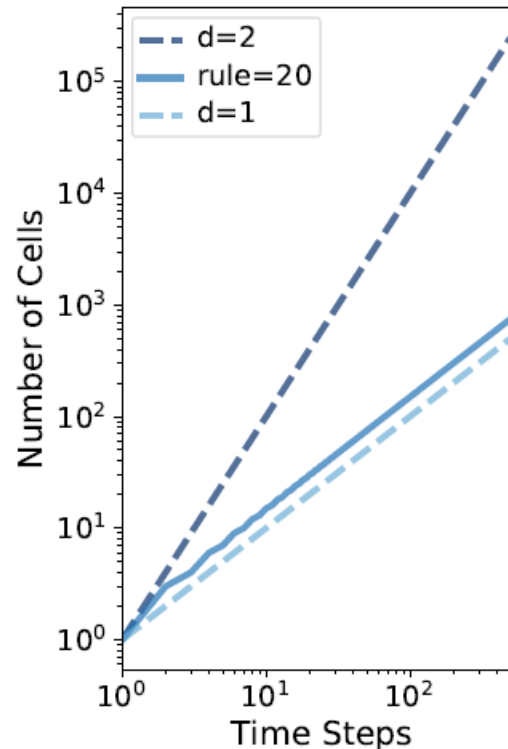


Rule 18



Experimental discovery of dimensions of a CA:
Count the number of cells in each step.

For rule 18, the slope is 1.57, meaning that the pattern generated has a fractional dimension, that is, it is a fractal.





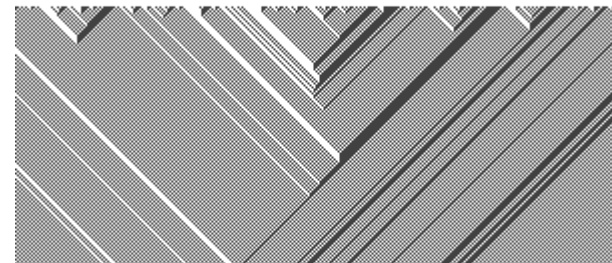
Cellular Automata Models for Traffic

Rule 184

current pattern	111	110	101	100	011	010	001	000
new state	1	0	1	1	1	0	0	0

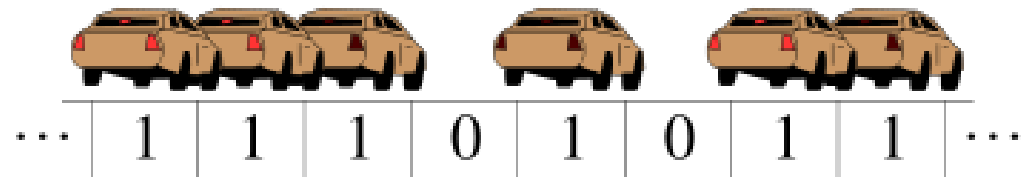
If a cell has state 0, its new state is taken from the cell to its left. Otherwise, its new state is taken from the cell to its right.

- Rule 184, run for 128 steps from random configurations with each of three different starting densities: top 25%, middle 50%, bottom 75%.
- The view shown is a 300-pixel crop from a wider simulation.



Traffic flow

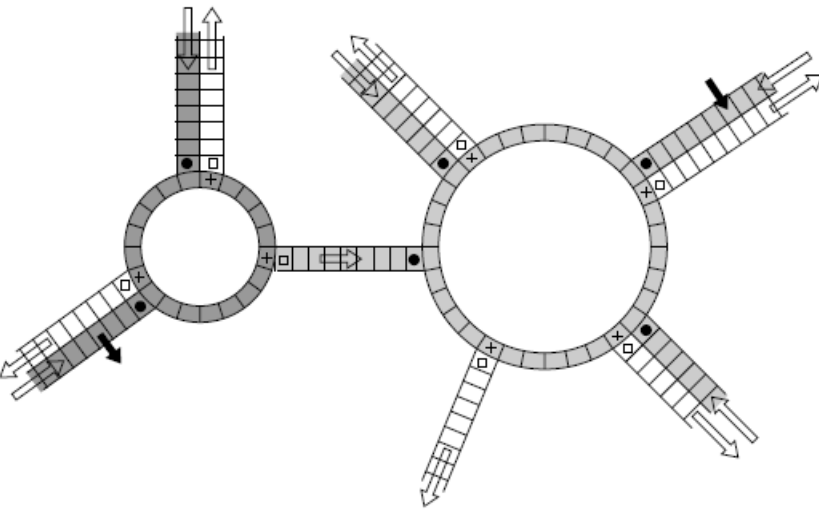
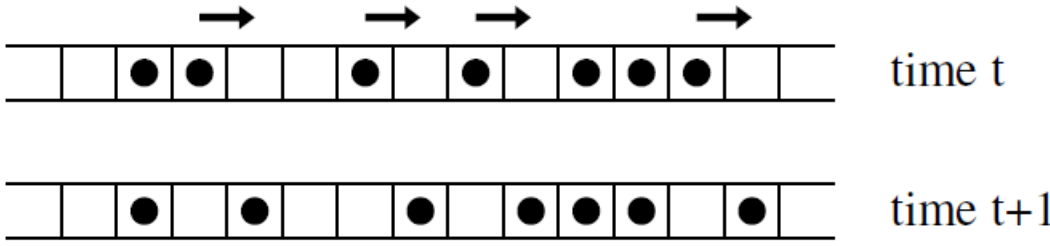
Rule 184 interpreted as a simulation of traffic flow. Each 1 cell corresponds to a vehicle, and each vehicle moves forwards only if it has open space in front of it.



Although very primitive, the Rule 184 model of traffic flow already predicts some of the familiar emergent features of real traffic: clusters of freely moving cars separated by stretches of open road when traffic is light, and waves of stop-and-go traffic when it is heavy.

Traffic Model

A vehicle can move only when downstream cell is free.



Motion Equations (for a single line)

$$n_i(t + 1) = n_i^{in}(t)(1 - n_i(t)) + n_i(t)n_i^{out}(t)$$

$n_i(t)$: 0 if i is free, 1 otherwise

$n_i^{in}(t)$: the state of the cell from which a car can come to i

$n_i^{out}(t)$: the state of the cell to which a car can go from i

Flow Diagram

The **car density** at time t on a road segment L of length $|L|$ is defined as

$$\rho(t) = \frac{N_L(t)}{|L|}$$

where $N_L(t)$ is the number of cars along segment L at time t

The **average velocity** $\langle v \rangle$ at time t on this segment is defined as

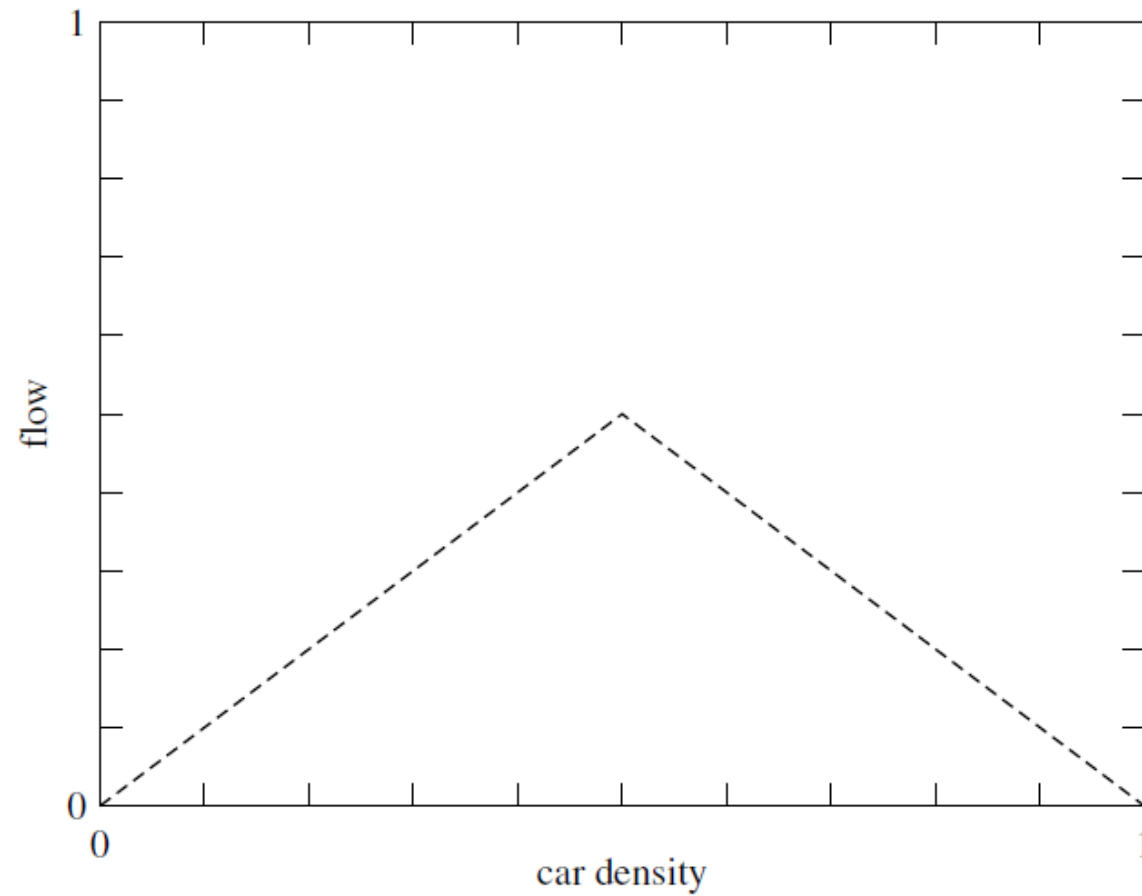
$$\langle v \rangle = \frac{M_L(t)}{N_L(t)}$$

where $M_L(t)$ is the number of cars moving at time t on segment L

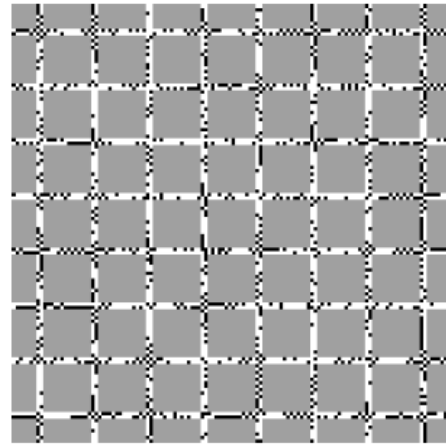
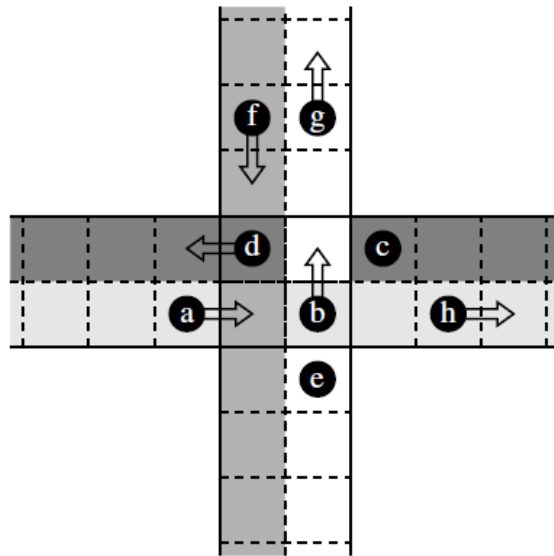
The **traffic flow** j is defined as

$$j = \rho(t)\langle v \rangle = \frac{M(t)}{|L|}$$

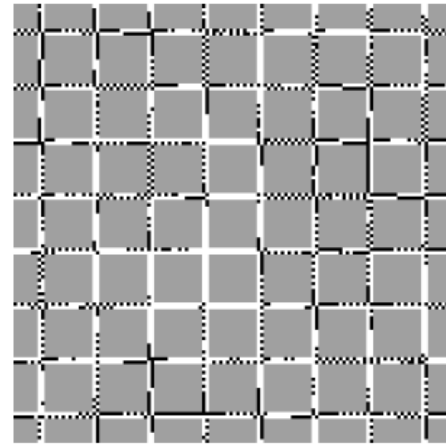
Flow Diagram of Rule 184



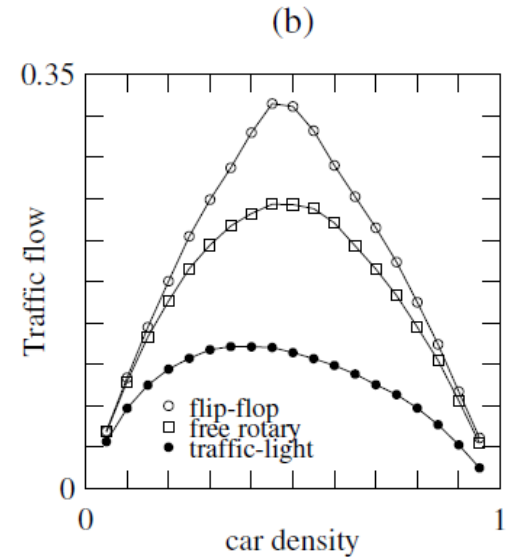
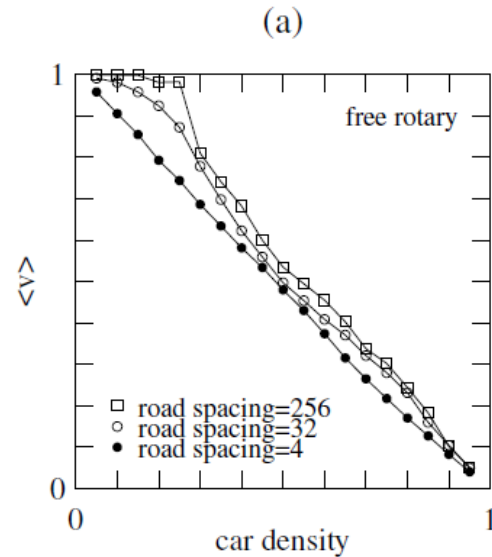
Traffic in a Manhattan-like City



(a)



(b)



Complex Systems



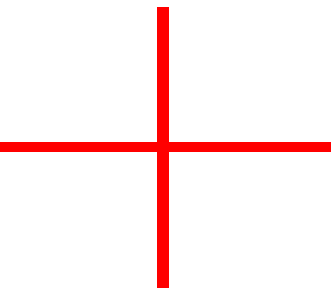
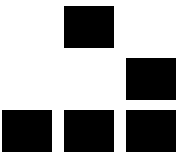
The Game of Life

3-d Game of life in Minecraft

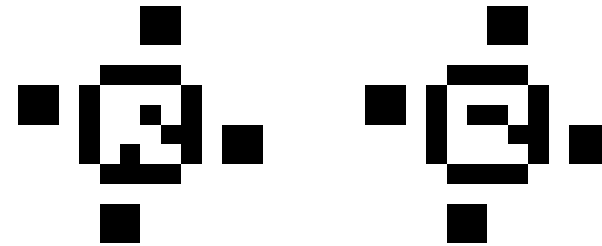
Simple rules, executed at each time step:

- A live cell with 2 or 3 live neighbors survives to the next round.
- A live cell with 4 or more neighbors dies of overpopulation.
- A live cell with 1 or 0 neighbors dies of isolation.
- An empty cell with exactly 3 neighbors becomes a live cell in the next round.

Glider



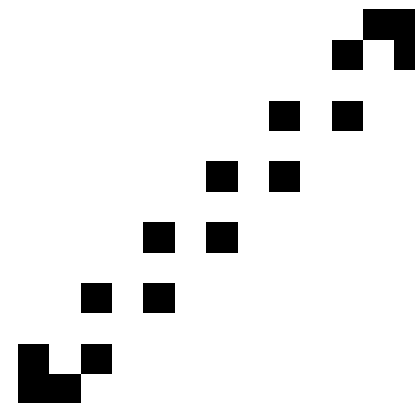
More



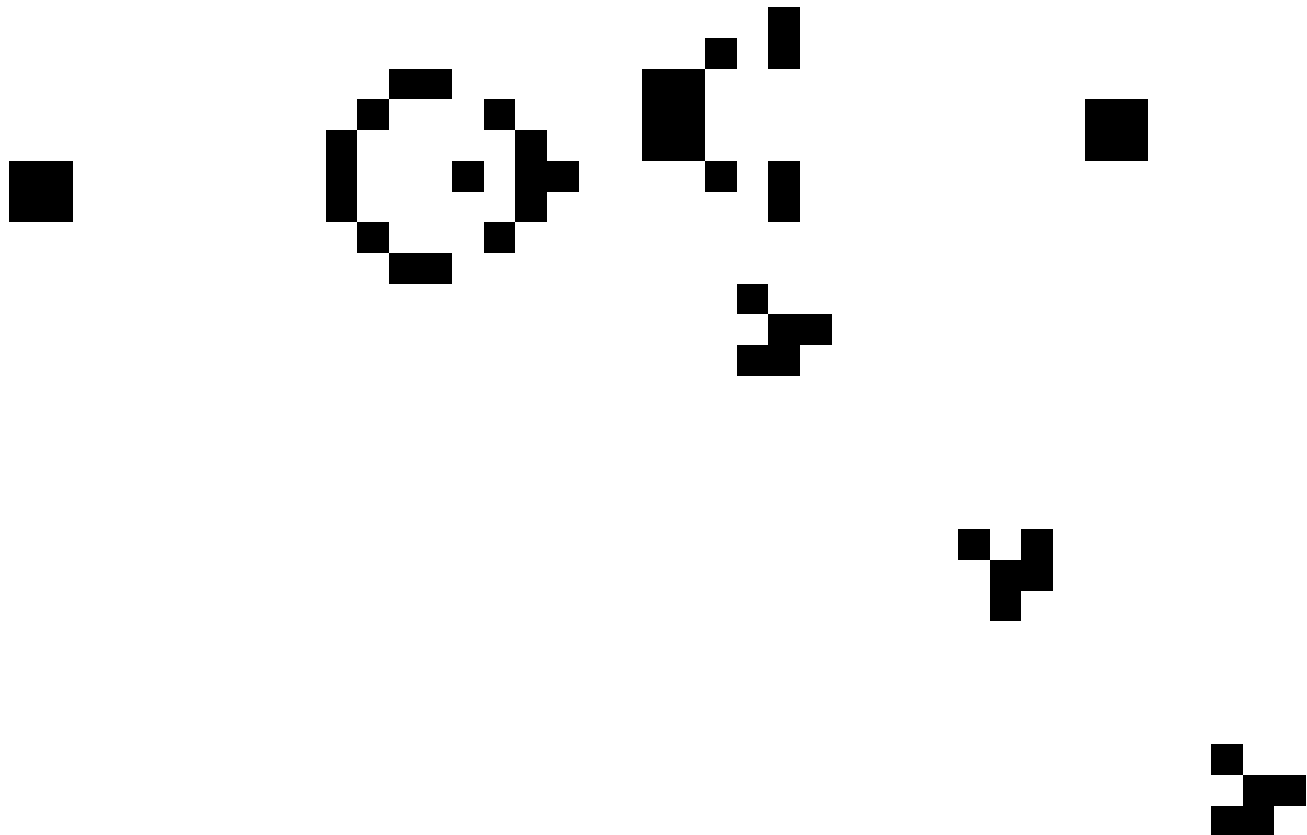
Sequence leading to
Blinkers

Clock

Barber's pole



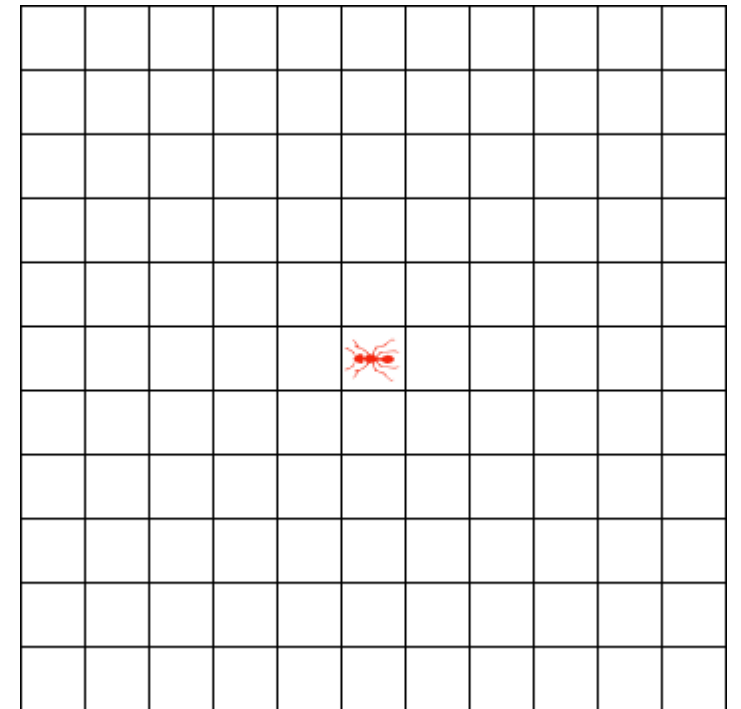
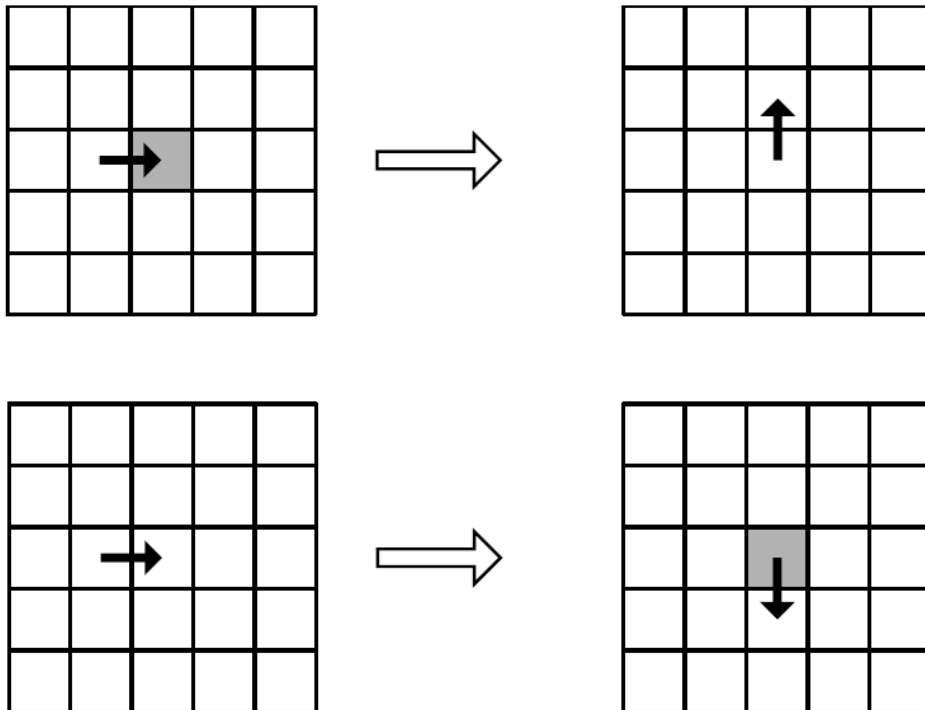
A Glider Gun



The Langton Ant

Ant's Rule

This is a hypothetical animal moving on a 2D lattice, according to a simple rule. This rule depends on the "color" of the cell on which the ant is.



Long-term Behavior

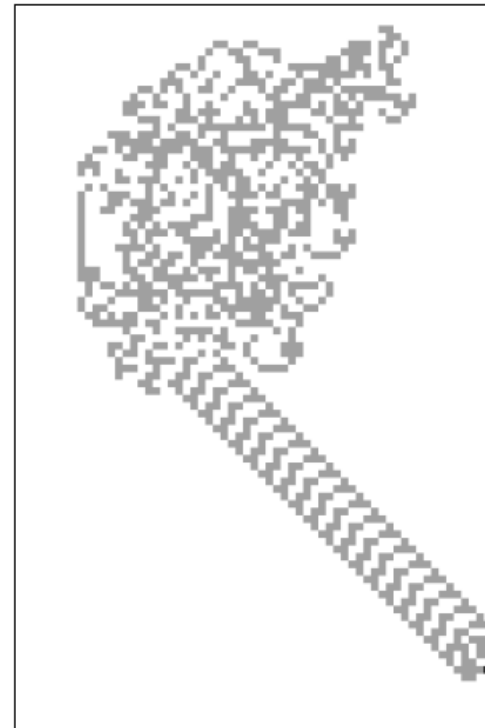
- The ant diverges to infinity (creates a highway)



t=6900



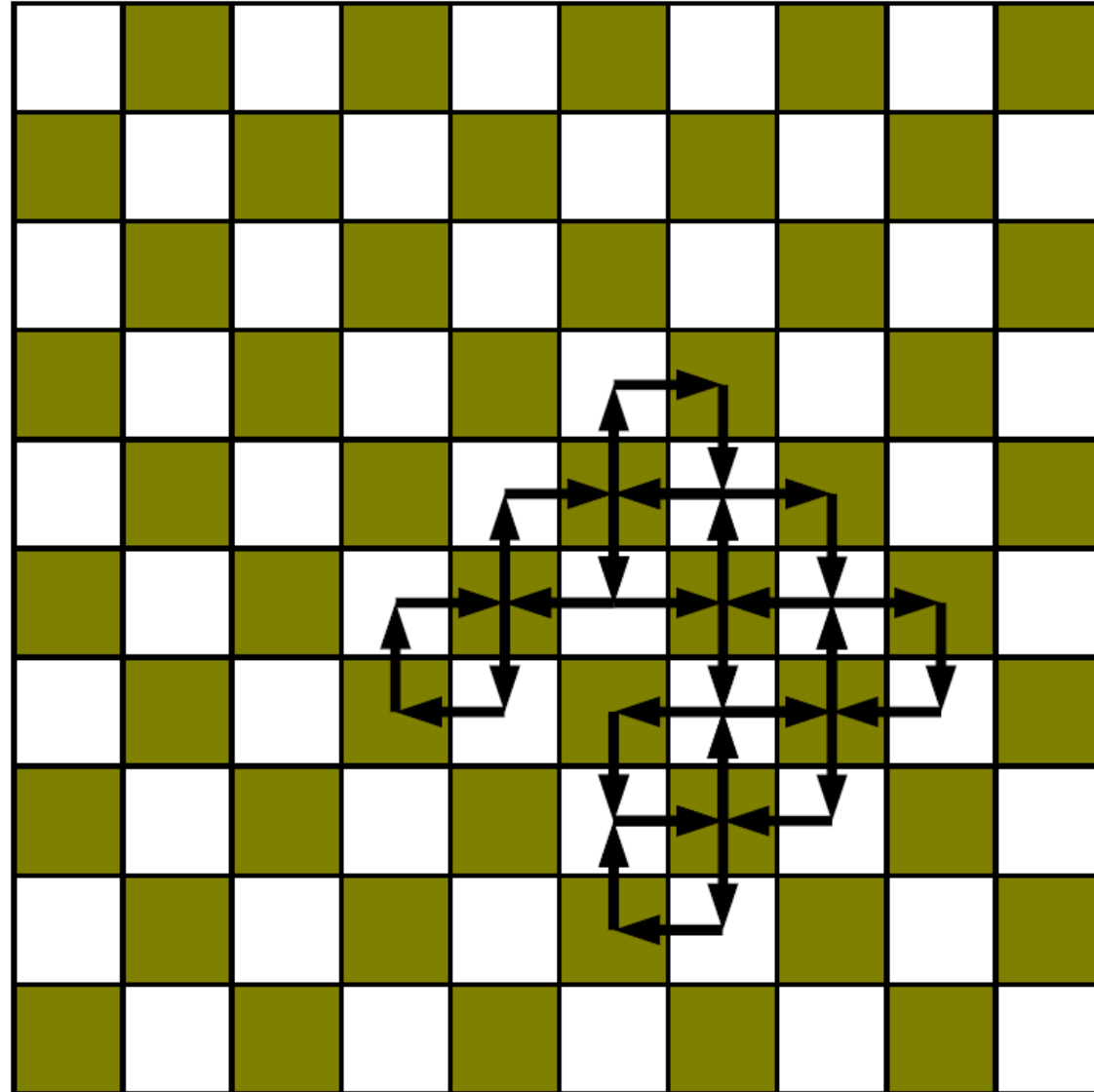
t=10431



t=12000

The Ant Always Escape to Infinity

(for any initial distribution of colors)



Impact on the Scientific Methodology

- We know perfectly well the fundamental law governing the system
 - ...because we define it ourselves
- However, we cannot predict the detailed motion of the ant (e.g. at what time does the highway appears)
- The microscopic description is not always able to predict the macroscopic behavior
- The only solution (up to now): observe the system
- The only information we get on the trajectory is global and reflects the symmetry of the rule.

Generalizations of CA

- Stochastic CA
- Asynchronous Update
- Non-uniform CA
- Structurally Dynamic CA
- ...



Predator-Prey Model

In Cellular Automata

HD

PREDATOR

Main idea

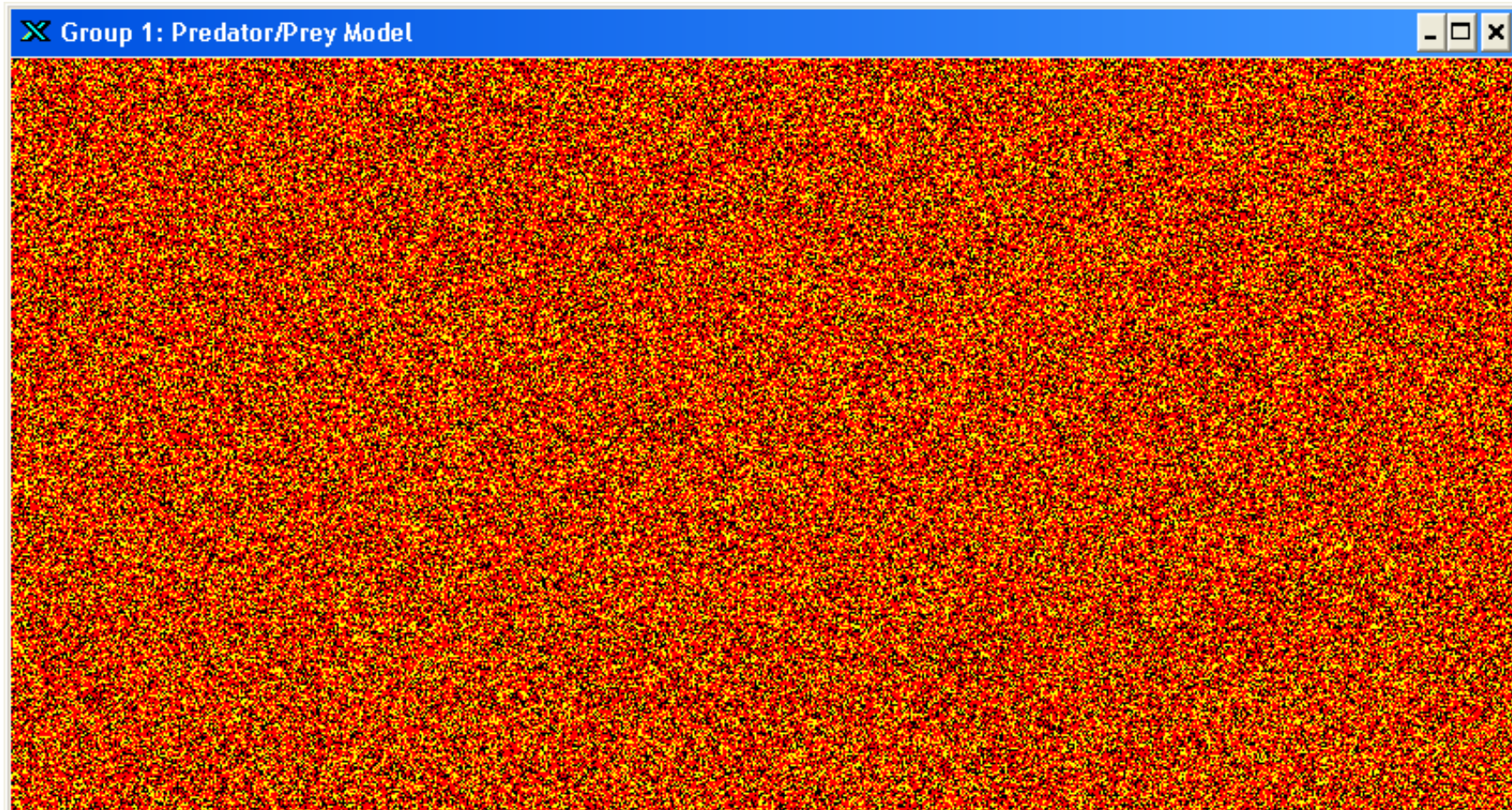
- Model predator(shark)/prey(fish) relationship by CA
- Define set of rules
- Begins with a randomly distributed population of fish, sharks, and empty cells in a 1000x2000 cell grid (2 million cells)
- Initially,
 - 50% of the cells are occupied by fish
 - 25% are occupied by sharks
 - 25% are empty

Based on the work of Bill Madden, Nancy Ricca and Jonathan Rizzo

*Adapted from: Wilkinson,B and M. Allen (1999): *Parallel Programming 2nd Edition*, NJ, Pearson Prentice Hall, p189

Here's the number 2 million

- Fish: red; sharks: yellow; empty: black



Rules

A dozen or so rules describe life in each cell:

- birth, longevity and death of a fish or shark
- breeding of fish and sharks
- over- and under-population
- fish/shark interaction
- Important: what happens in each cell is determined only by rules that apply locally, yet which often yield long-term large-scale patterns.

Do a LOT of computation!

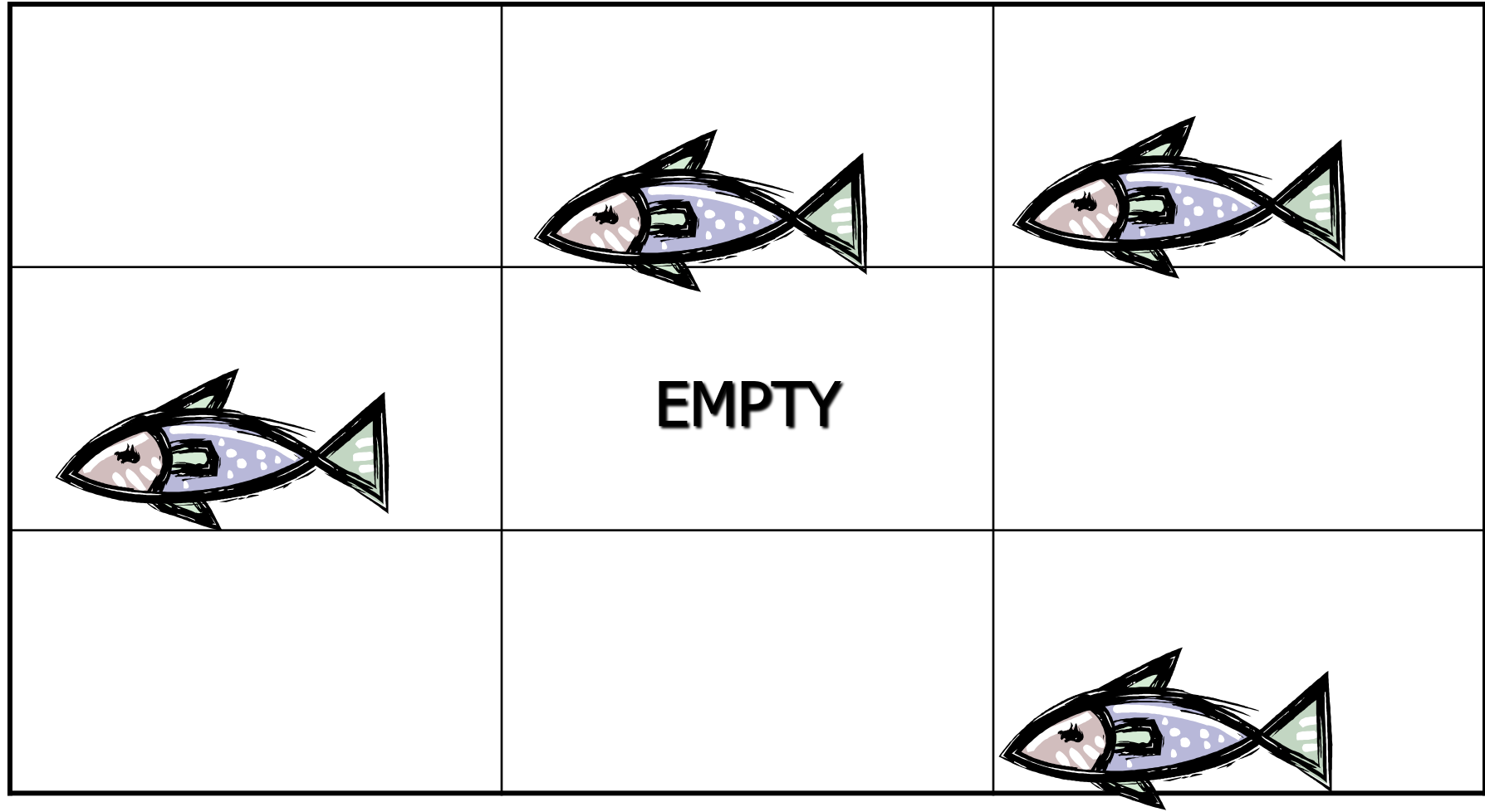
- Apply a dozen rules to each cell
- Do this for 2 million cells in the grid
- Do this for 20,000 generations
- Well over a trillion calculations per run!
- Do this as quickly as you can

Rules in detail: Breeding Rule

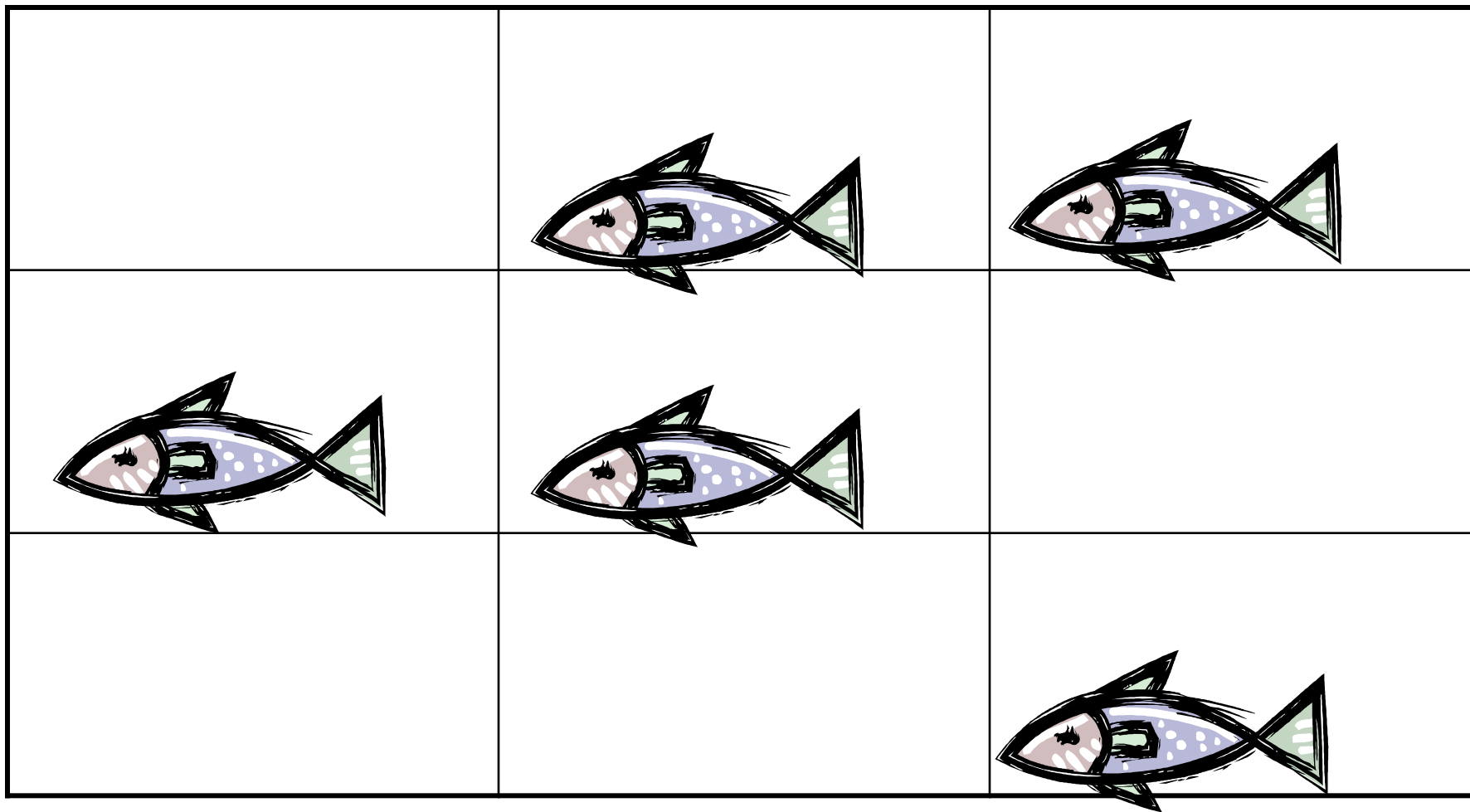
Breeding rule: if the current cell is empty

- If there are ≥ 4 neighbors of one species, and ≥ 3 of them are of breeding age,
 - Fish breeding age ≥ 2 ,
 - Shark breeding age ≥ 3 ,and there are < 4 of the other species:
 - then create a species of that type
 - $+1$ = baby fish (age = 1 at birth)
 - -1 = baby shark (age = $|-1|$ at birth)

Breeding Rule: Before



Breeding Rule: After



Rules in Detail: Fish Rules

If the current cell contains a fish:

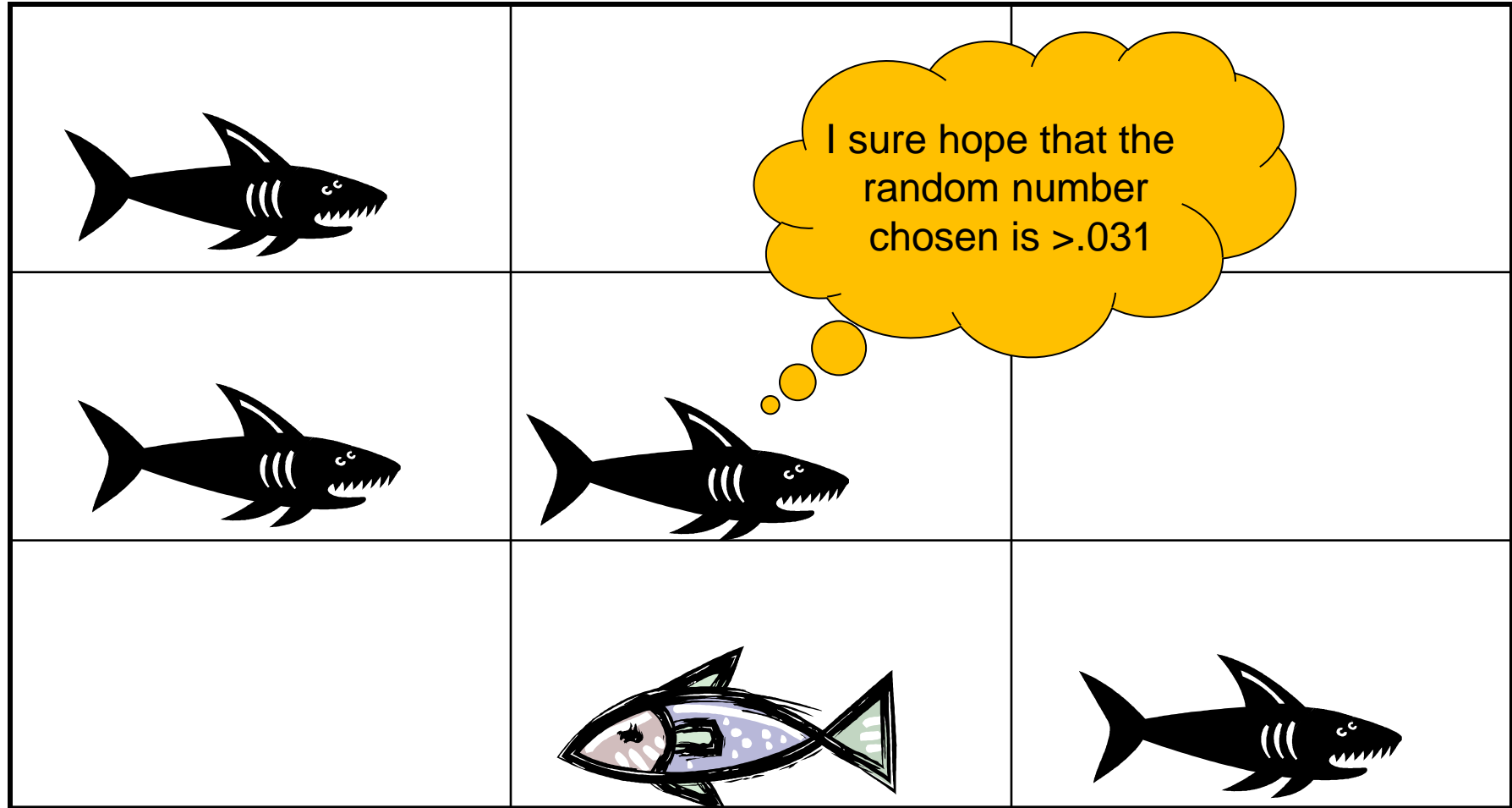
- Fish live for 10 generations
- If ≥ 5 neighbors are sharks, fish dies (shark food)
- If all 8 neighbors are fish, fish dies (overpopulation)
- If a fish does not die, increment age

Rules in Detail: Shark Rules

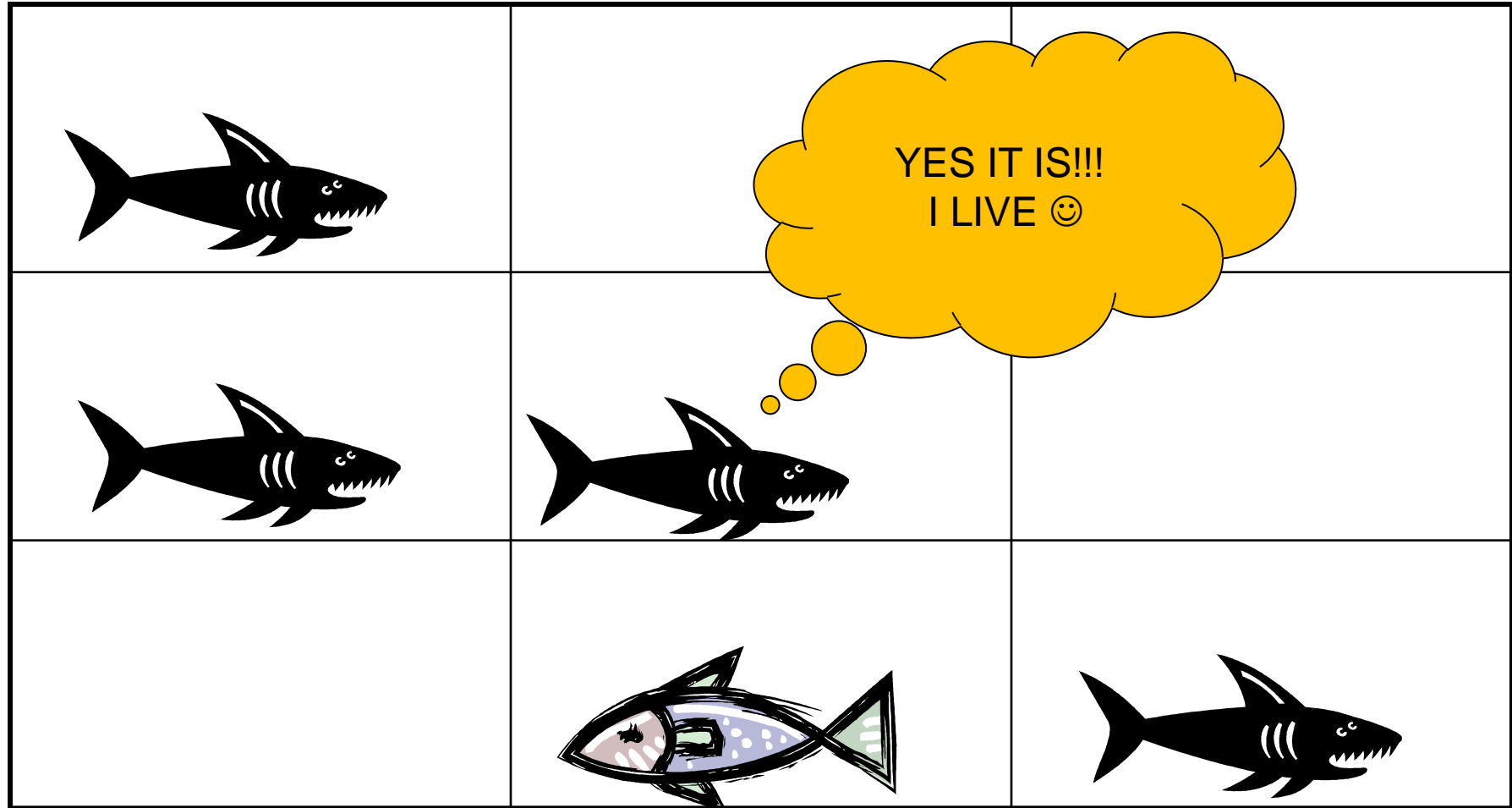
If the current cell contains a shark:

- Sharks live for 20 generations
- If ≥ 6 neighbors are sharks and fish neighbors = 0, the shark dies (starvation)
- A shark has a $1/32$ (.031) chance of dying due to random causes
- If a shark does not die, increment age

Shark Random Death: Before



Shark Random Death: After



Programming Logic

```
// If the current cell is empty, check to
// see if a shark or a fish is born
if (*current == 0)
{
    if ((fish >= 4) && (fish > sharks) && (fishbreed > 2))
        (*uu)(i,j) = 1;
    else
    {
        if ((sharks >= 4) && (sharks > fish) && (sharkbreed > 2))
            (*uu)(i,j) = -1;
        else
            (*uu)(i,j) = (*current);
    }
    continue;
}
```

- Use 2-dimensional array to represent grid
- At any one (x, y) position, value is:
 - Positive integer (fish present)
 - Negative integer (shark present)
 - Zero (empty cell)
 - Absolute value of cell is age

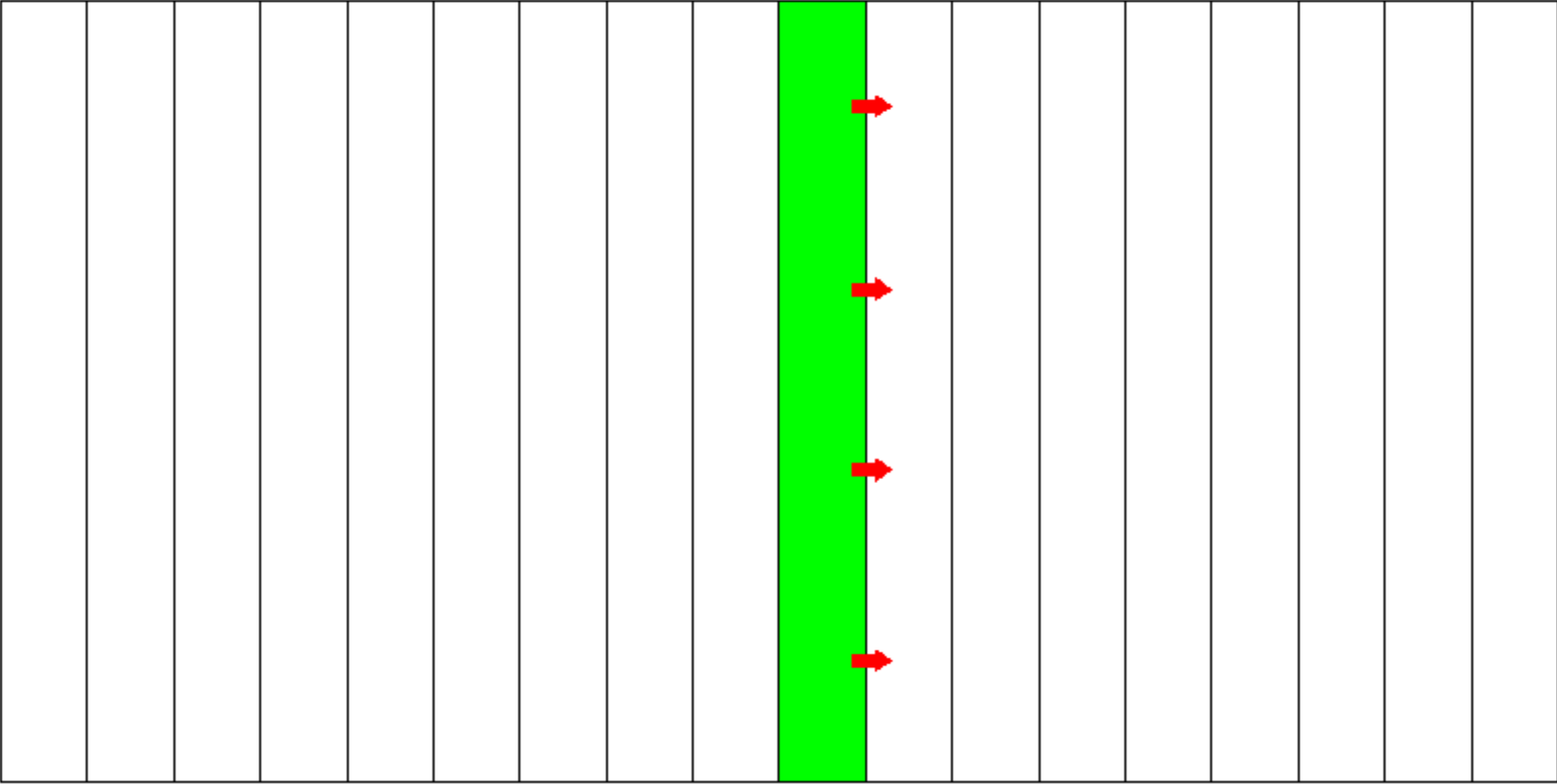
Parallelism

- A single CPU has to do it all:
 - Applies rules to first cell in array
 - Repeats rules for each successive cell in array
 - After 2 millionth cell is processed, array is updated
 - One generation has passed
 - Repeat this process for many generations
 - Every 100 generations or so, convert array to red, yellow and black pixels and send results to screen

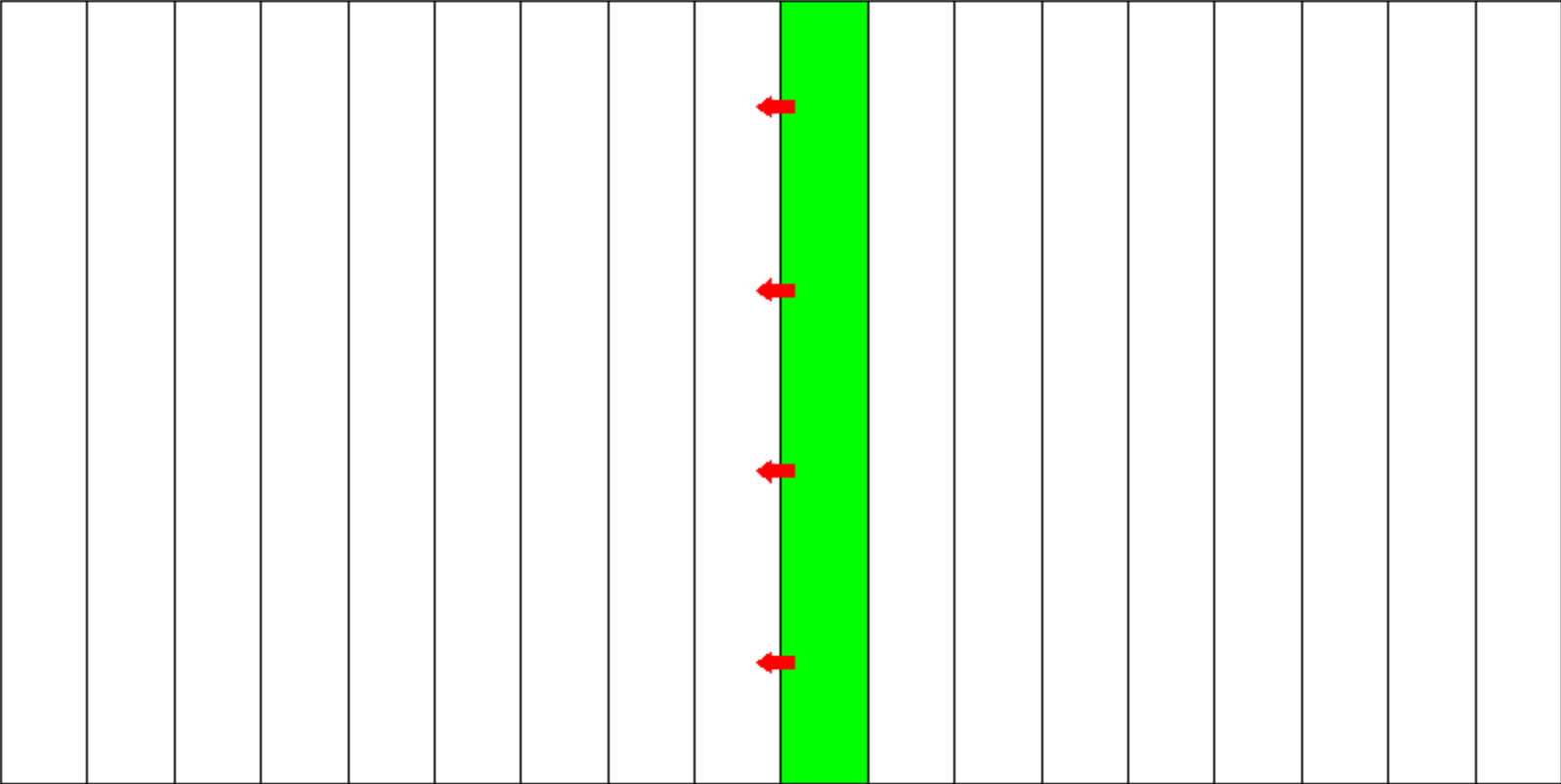
Parallelism

- How to split the work among 20 CPUs
 - 1 CPU acts as Master (has copy of whole array)
 - 18 CPUs act as Slaves (handle parts of the array)
 - 1 CPU takes care of screen updates
- Problem: communication issue concerning cells along array boundaries among slaves

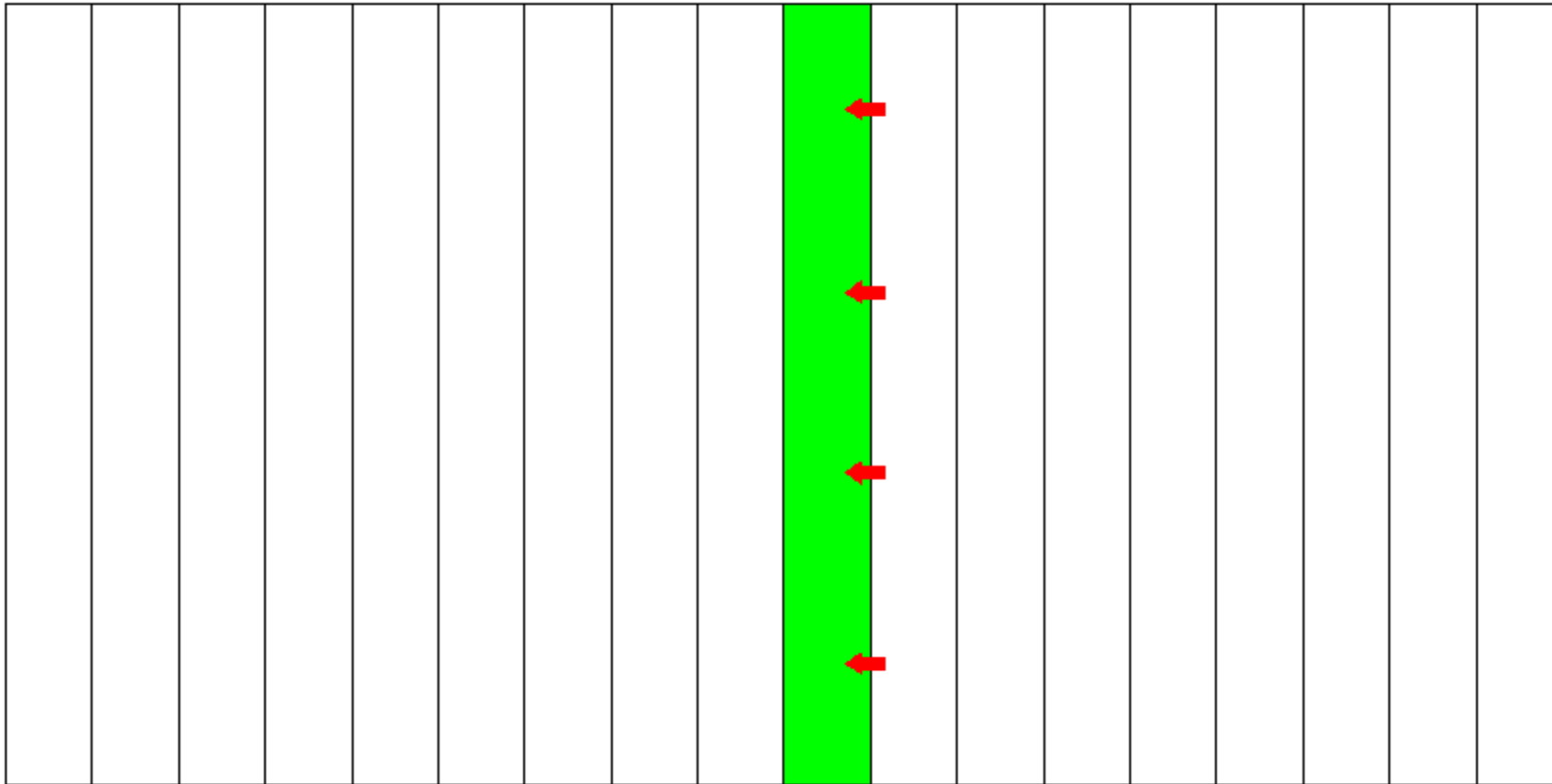
Send Right Boundary Values



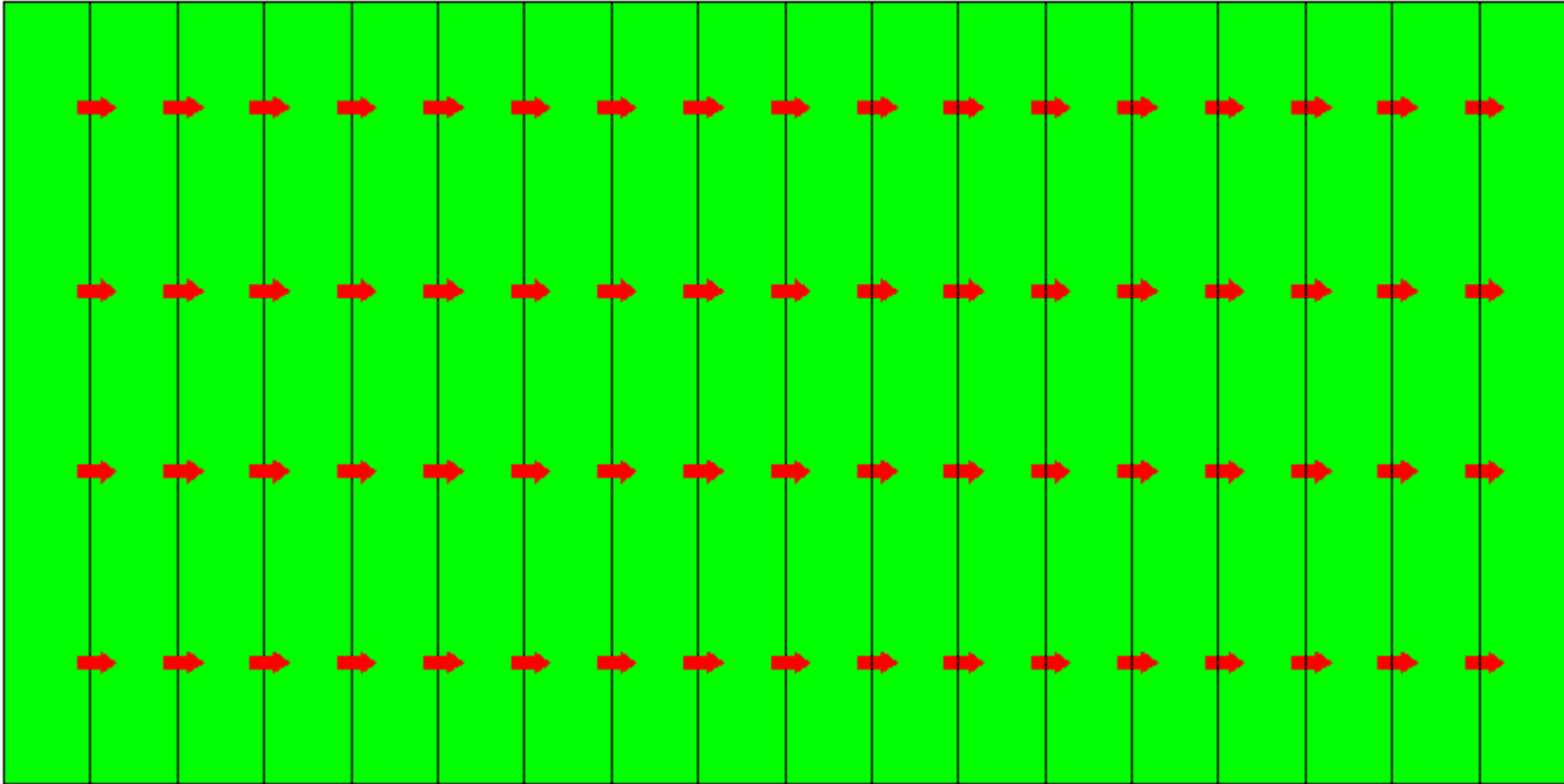
Send Left Boundary Values



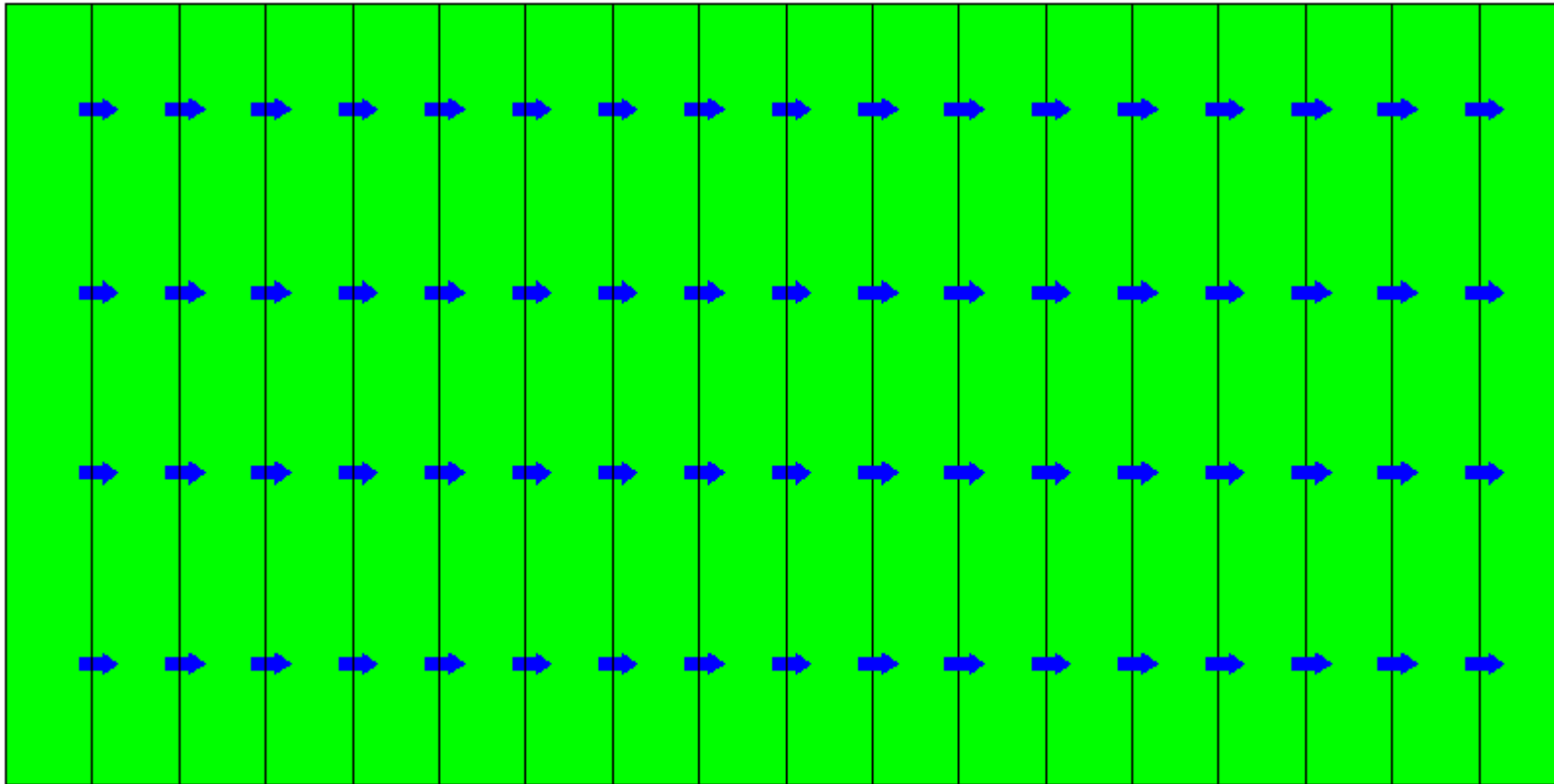
Receive Right Boundary Values



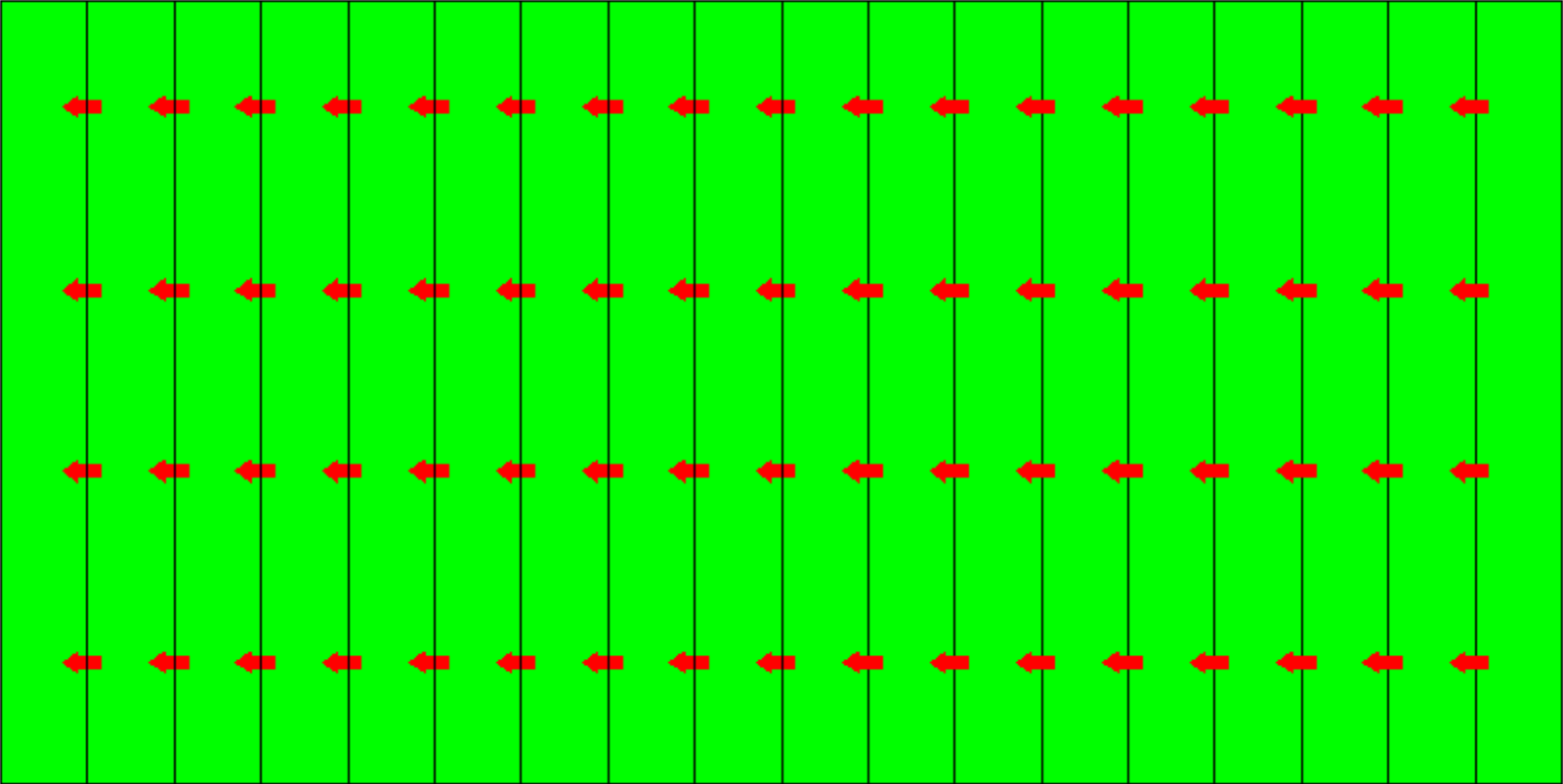
Send Right Boundary Values



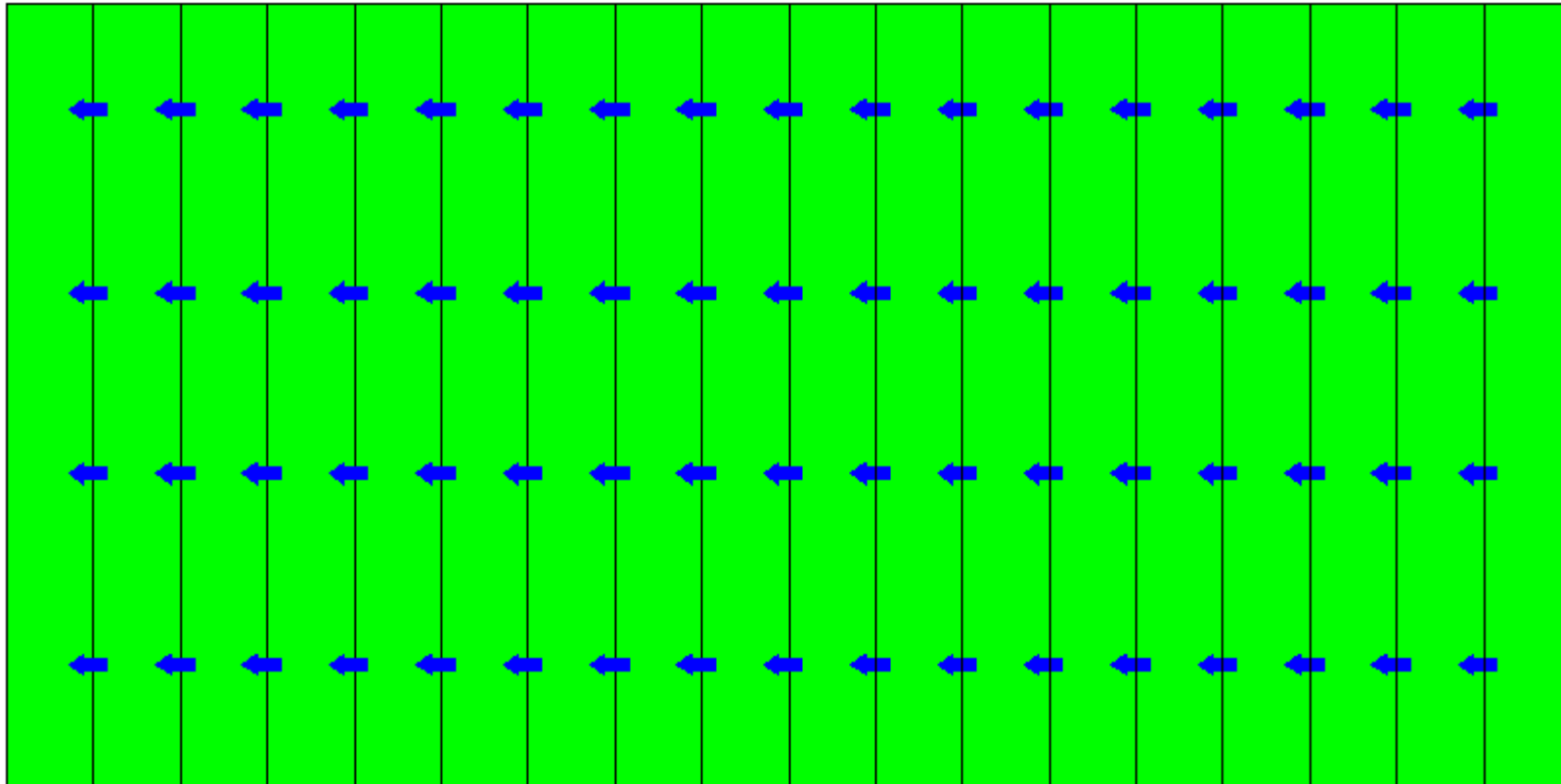
Receive Left Boundary Values



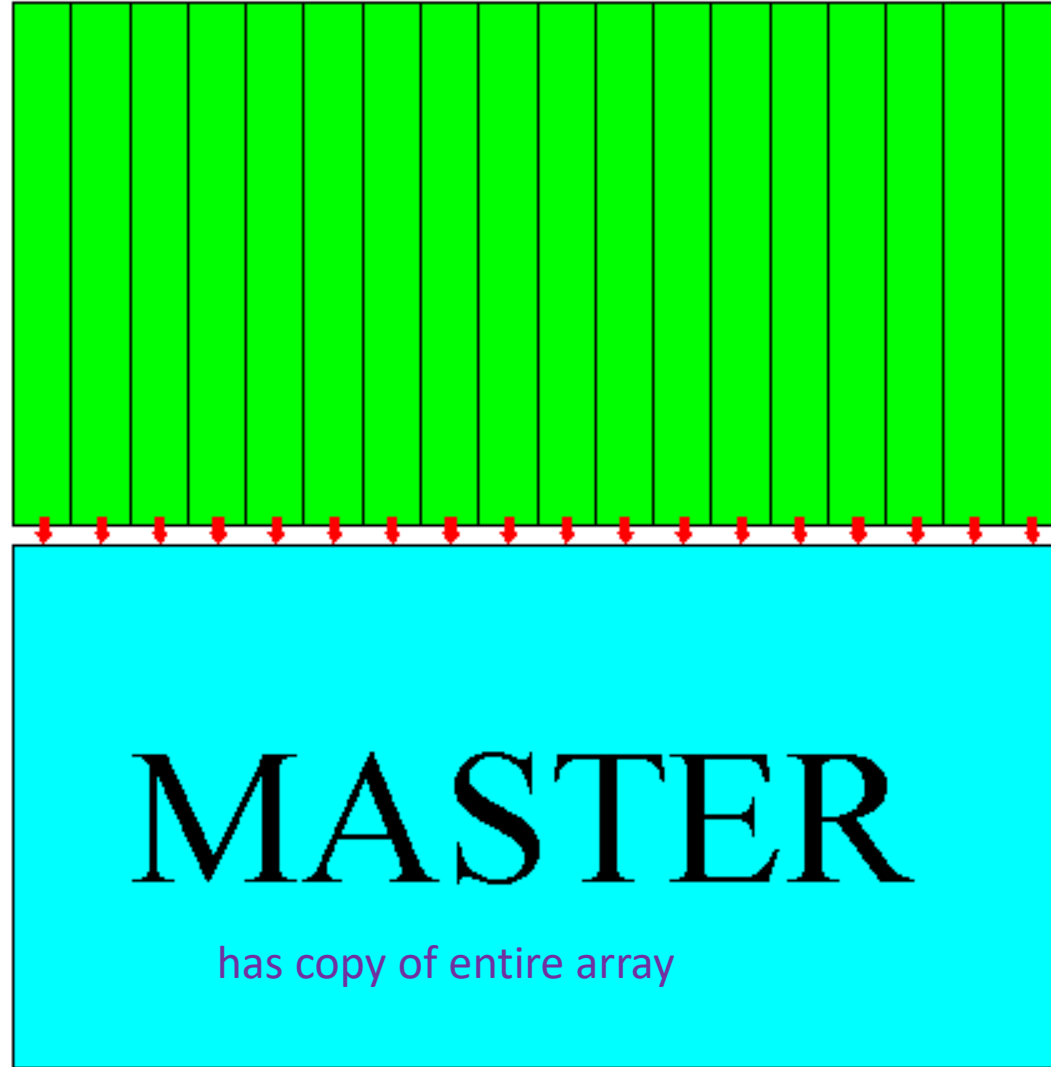
Send Left Boundary Values



Receive Right Boundary Values



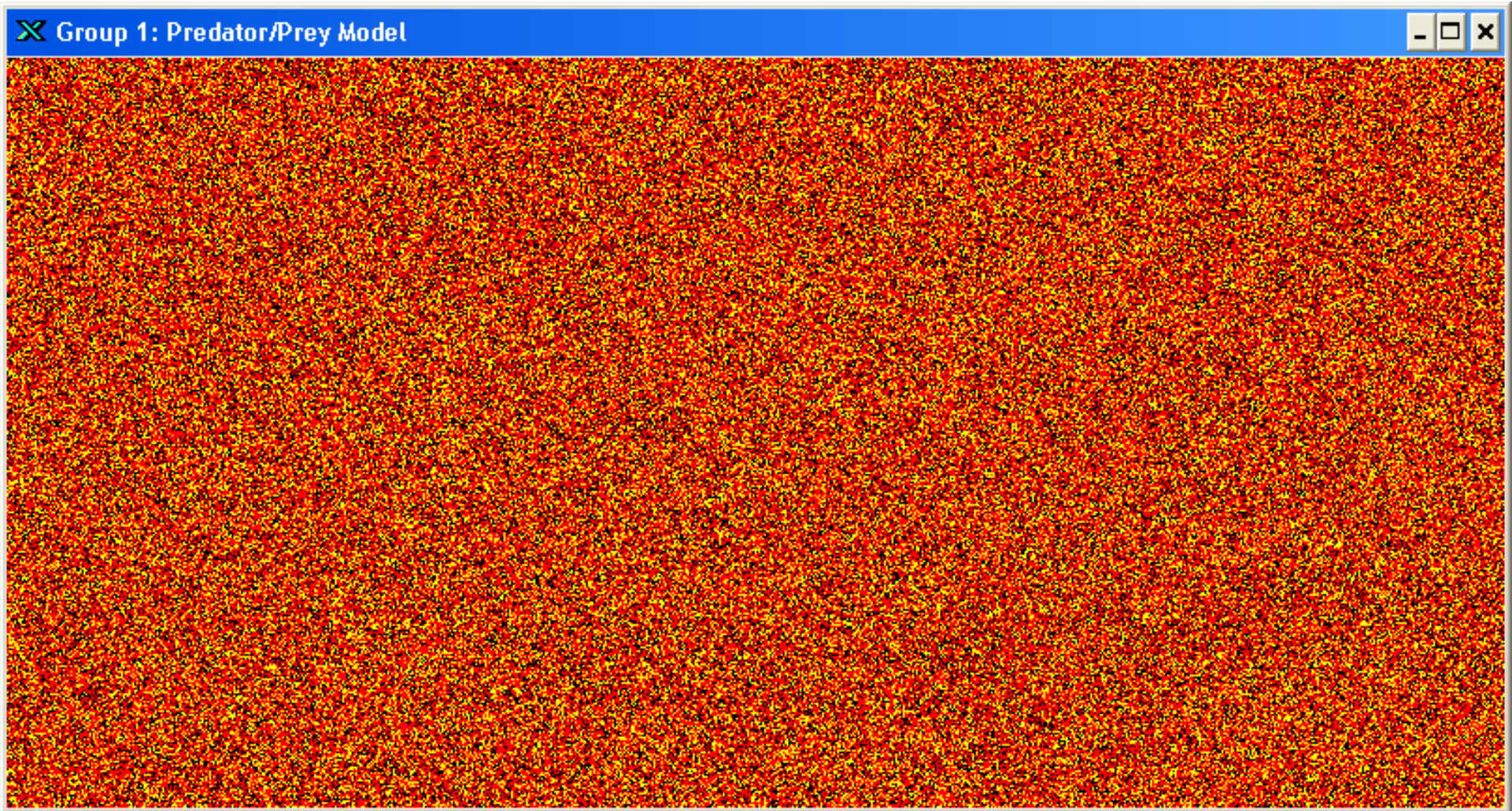
At intervals, update the master CPU



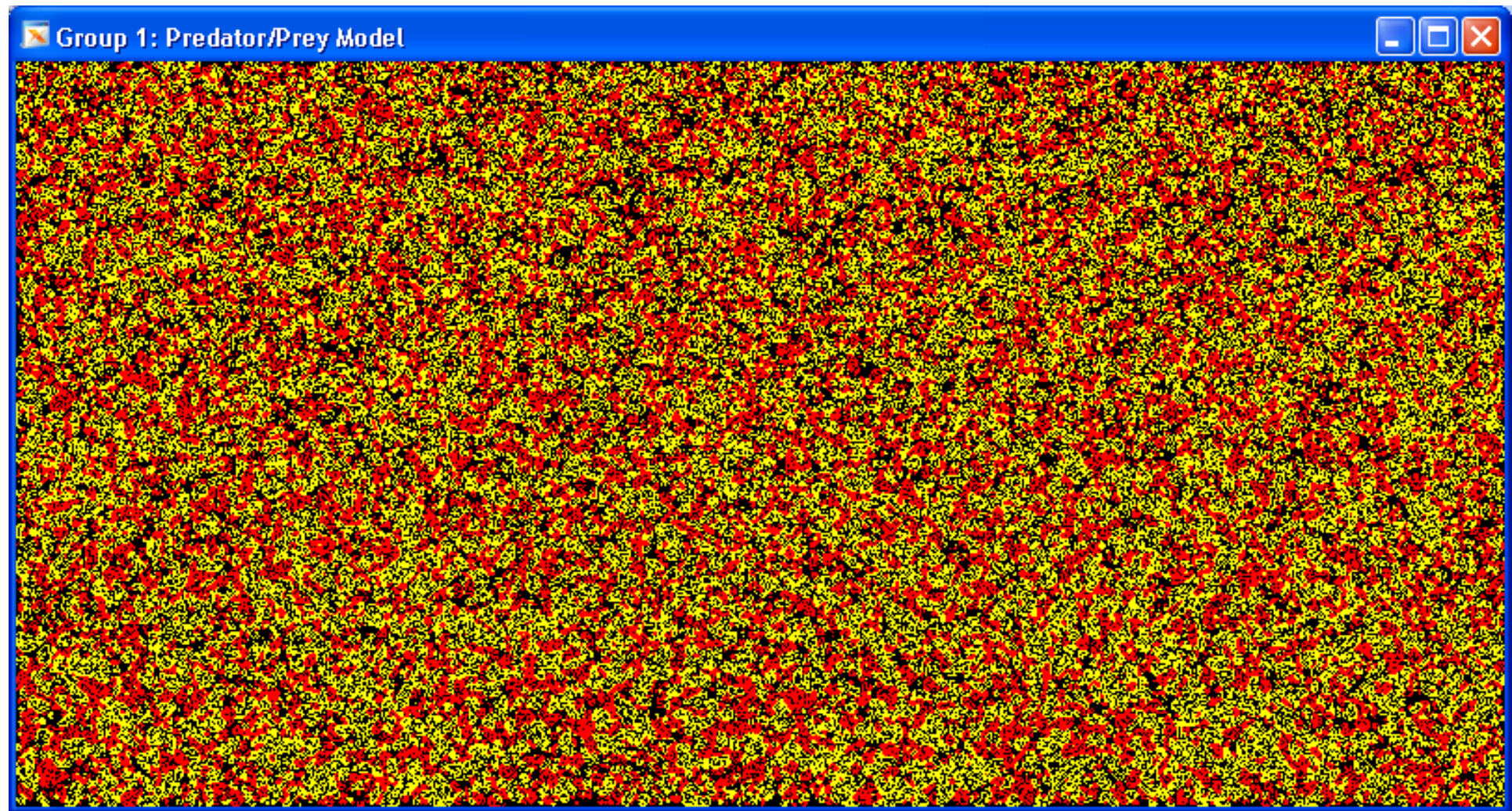
Illustration

- Next several screens show behavior over a span of 10,000+ generations (about 25 minutes on a cluster of 20 processors)

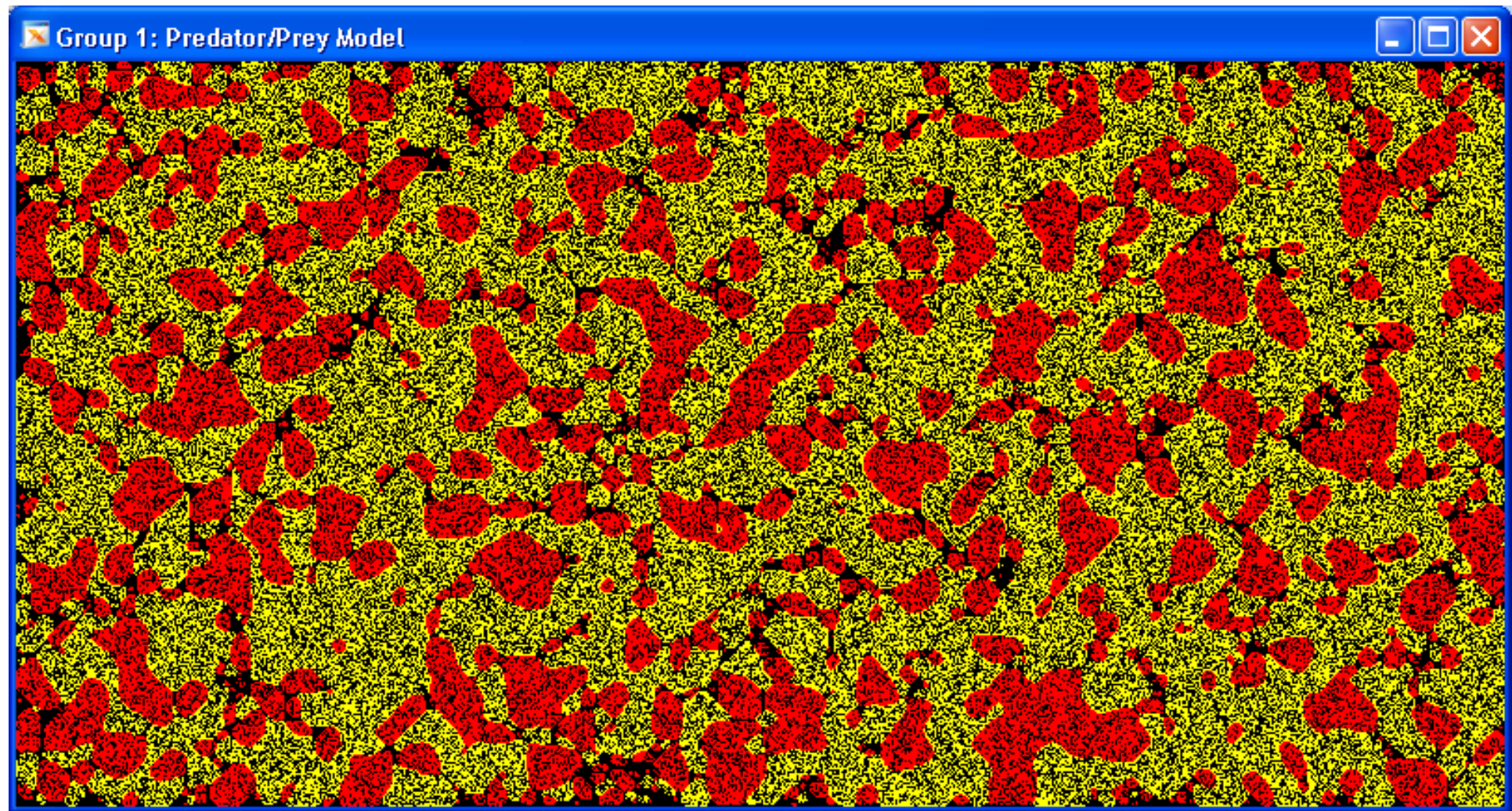
Generation: 0



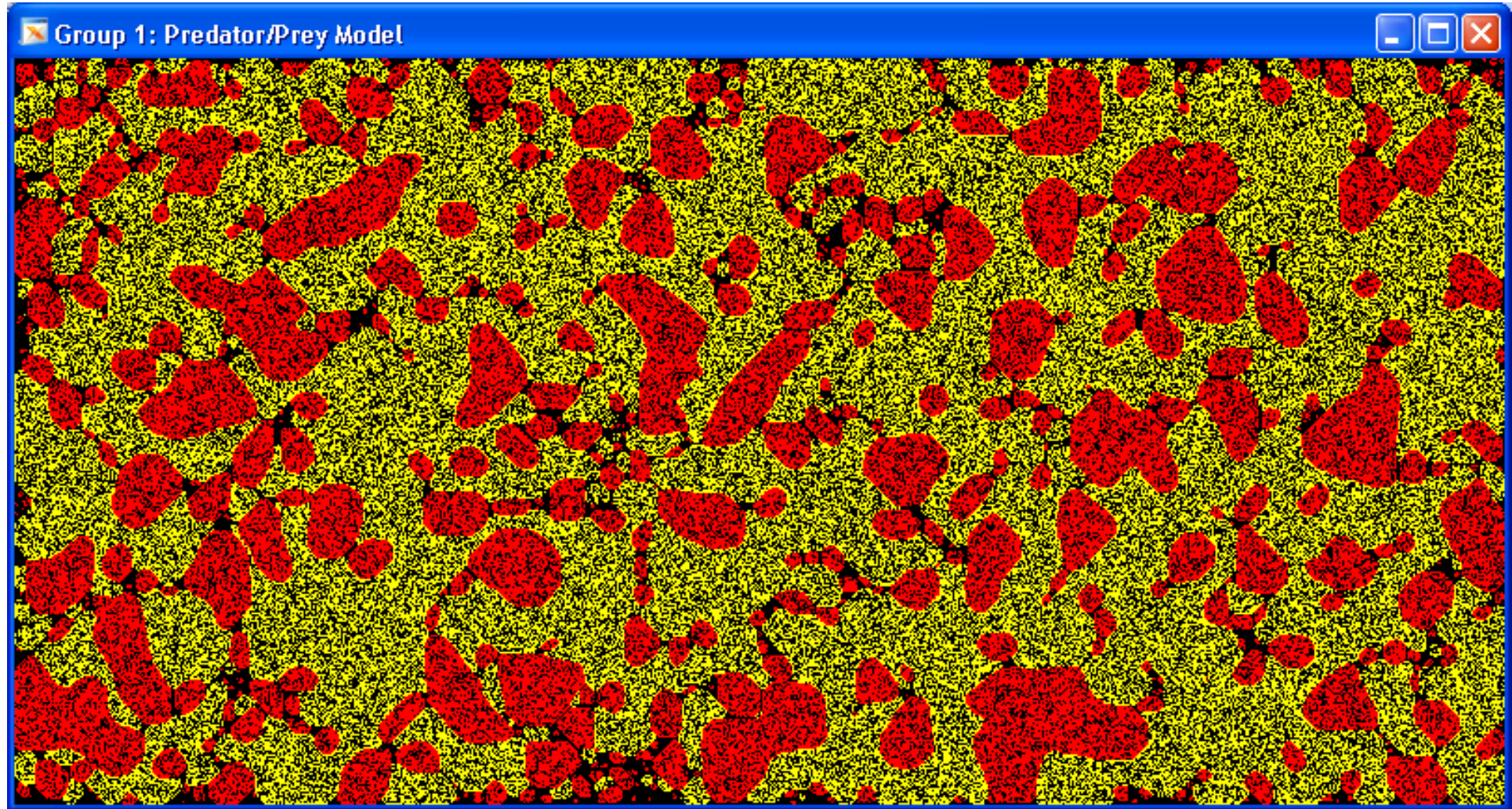
Generation: 100



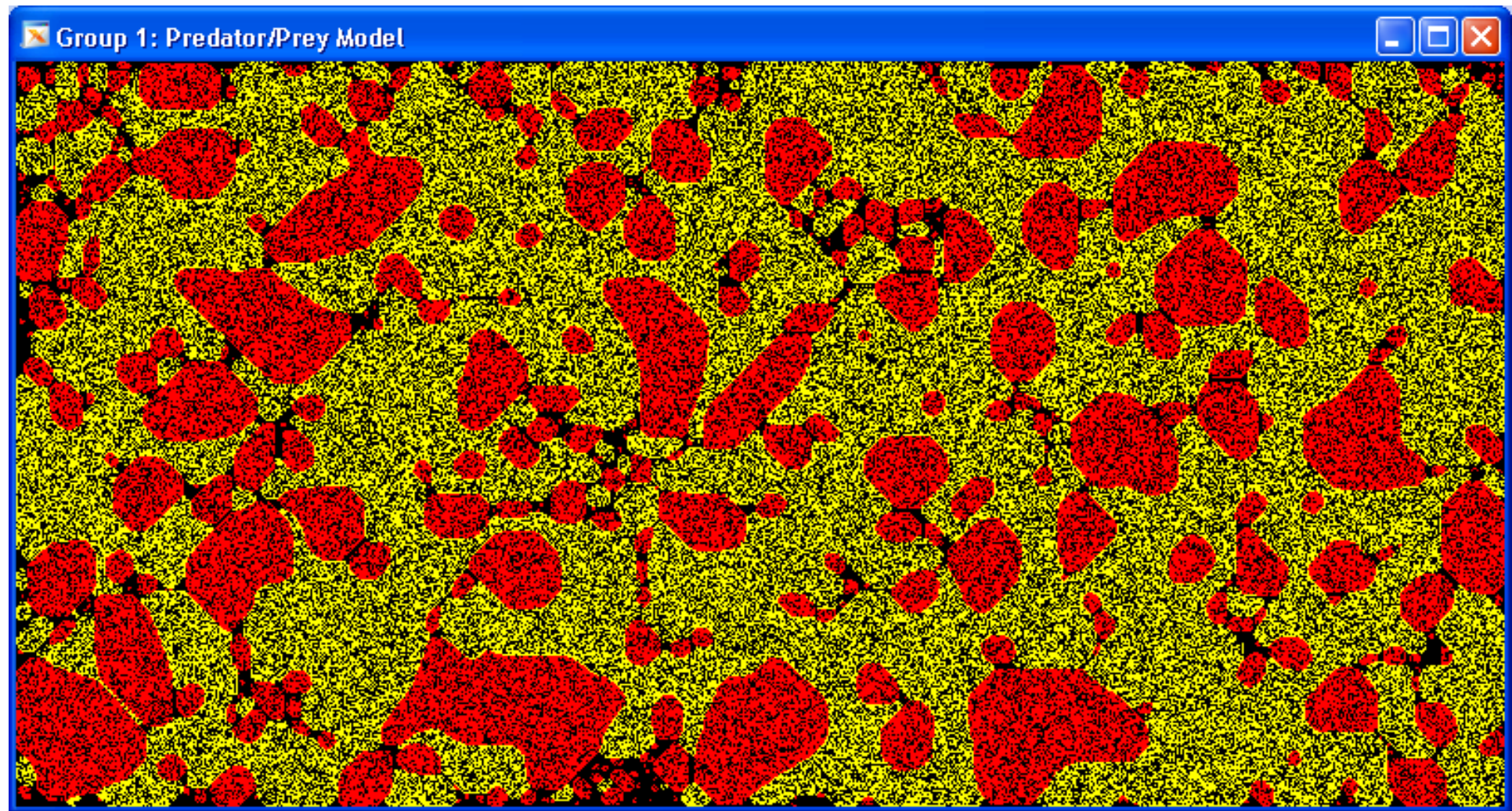
Generation: 500



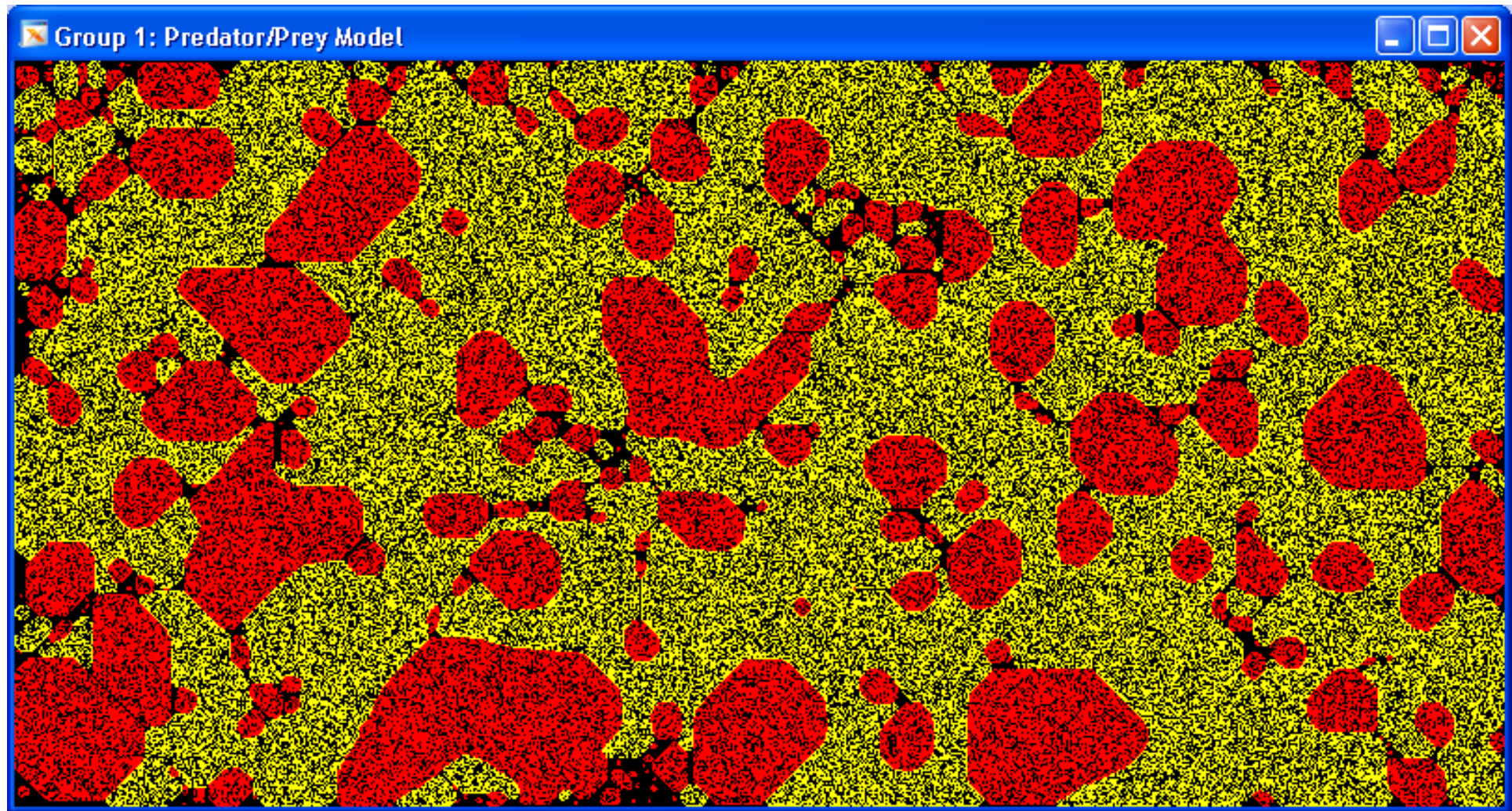
Generation: 1,000



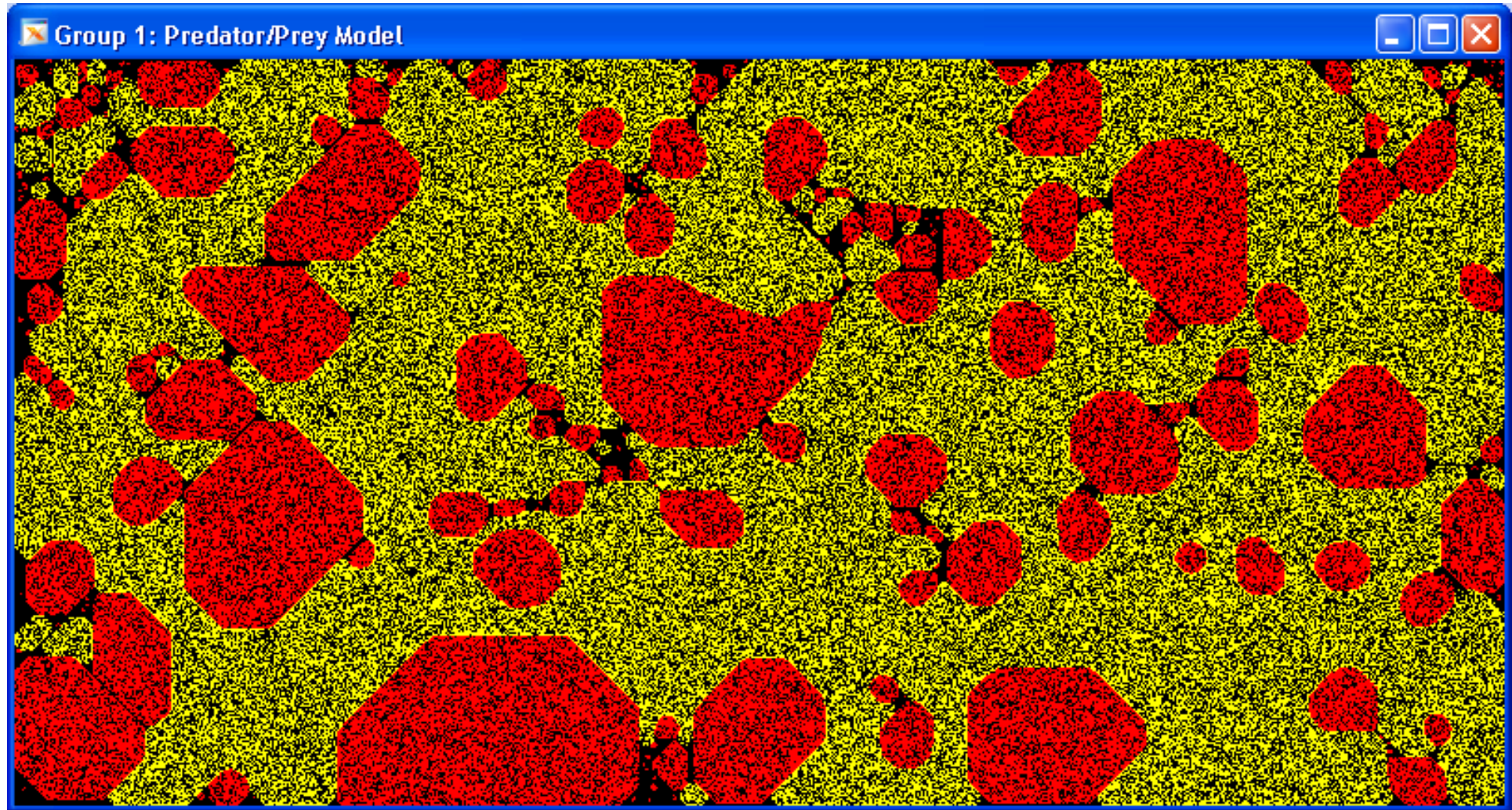
Generation: 2,000



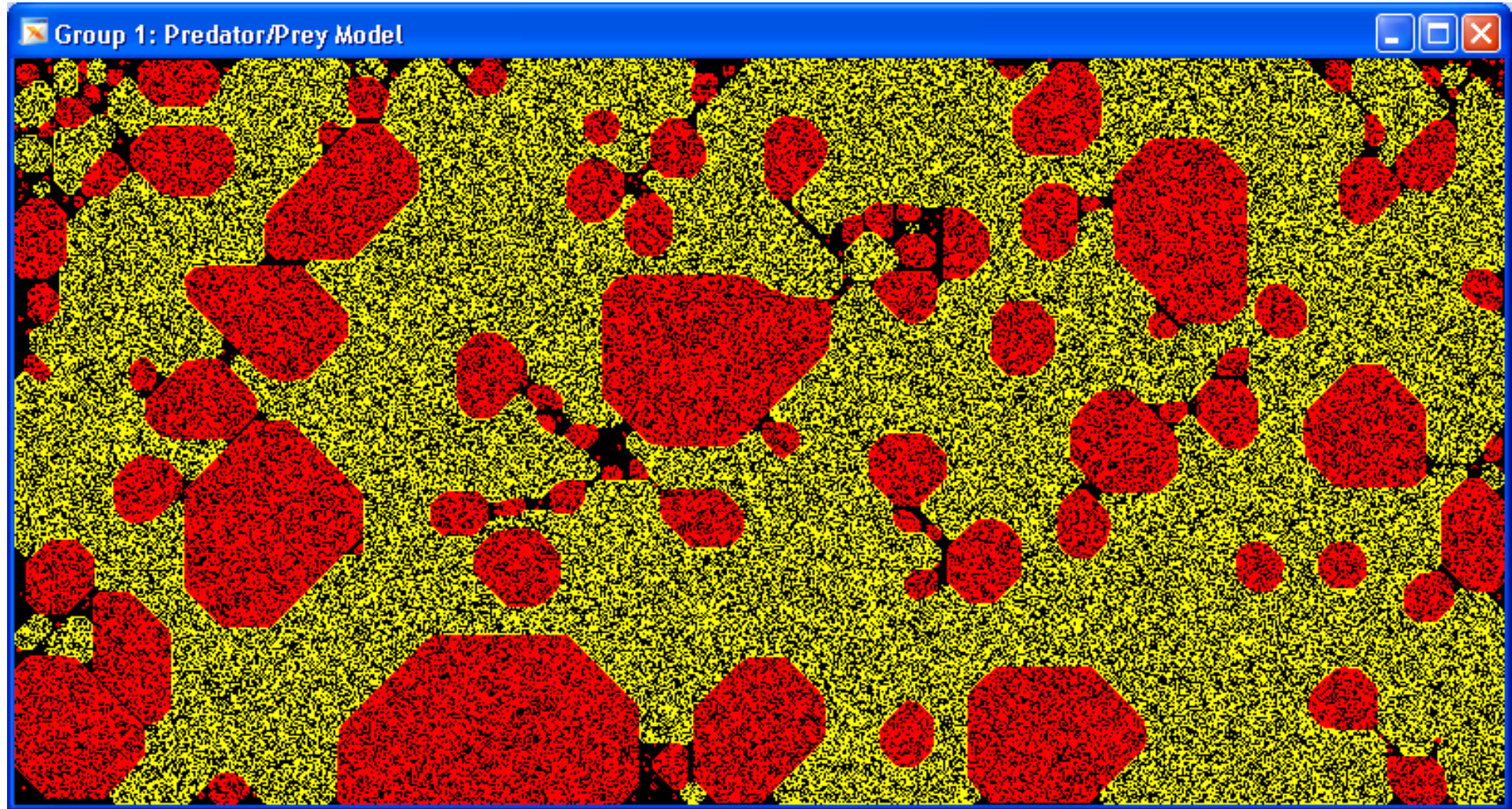
Generation: 4,000



Generation: 8,000



Generation: 10,500



Variations of Initial Conditions

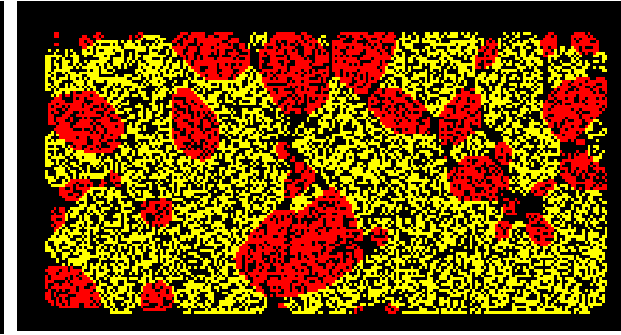
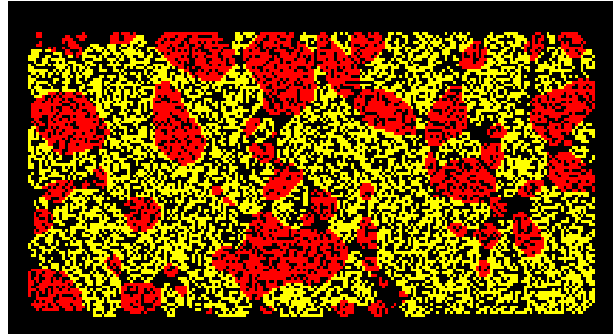
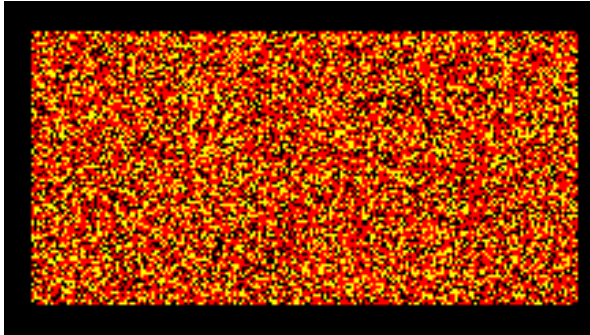
- Still using randomly distributed populations:
 - Medium-sized population. Fish/sharks occupy:
1/16th of total grid
Fish: 62,703; Sharks: 31,301
 - Very small population. Fish/sharks occupy:
1/800th of total grid
Initial population:
Fish: 1,298; Sharks: 609

Medium-sized population (1/16 of grid)

Generation 100

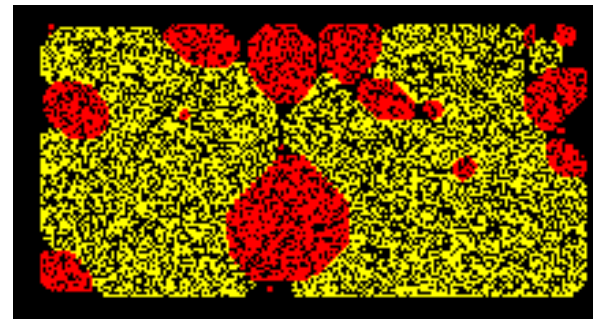
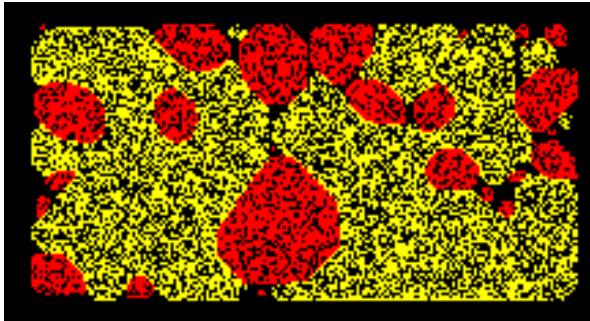
1000

2000



4000

8000



- Random placement of very small populations can favor one species over another
- Fish favored: sharks die out
- Sharks favored: sharks predominate, but fish survive in stable small numbers

Very Small Populations

- Random placement of very small populations can favor one species over another
- Fish favored: sharks die out
- Sharks favored: sharks predominate, but fish survive in stable small numbers

References

1. [Cellular Automata](#). Jarkko Kari, Lecture Notes, 2011.
2. [Think Complexity](#). A.B. Downey, Green Tea Press, 2016.
3. [An Introduction to Percolation](#). A. Yadin, Lecture Notes, 2020.