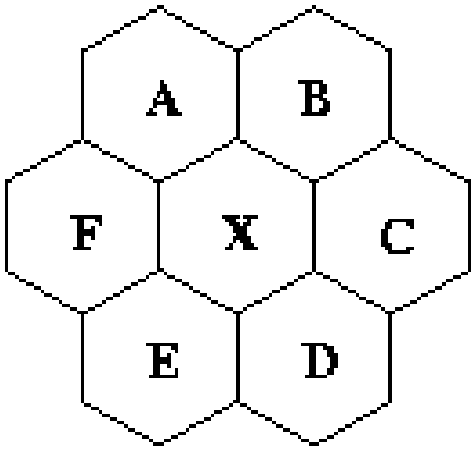
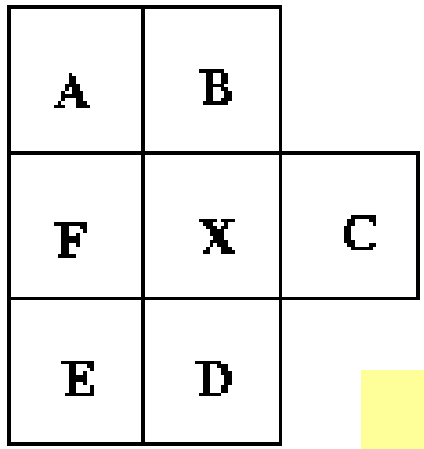
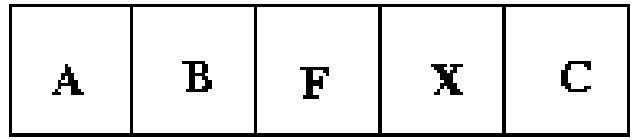
Cellular Automata

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, 3-dimensional blocks or another shape such as a hexagon.







Different models.

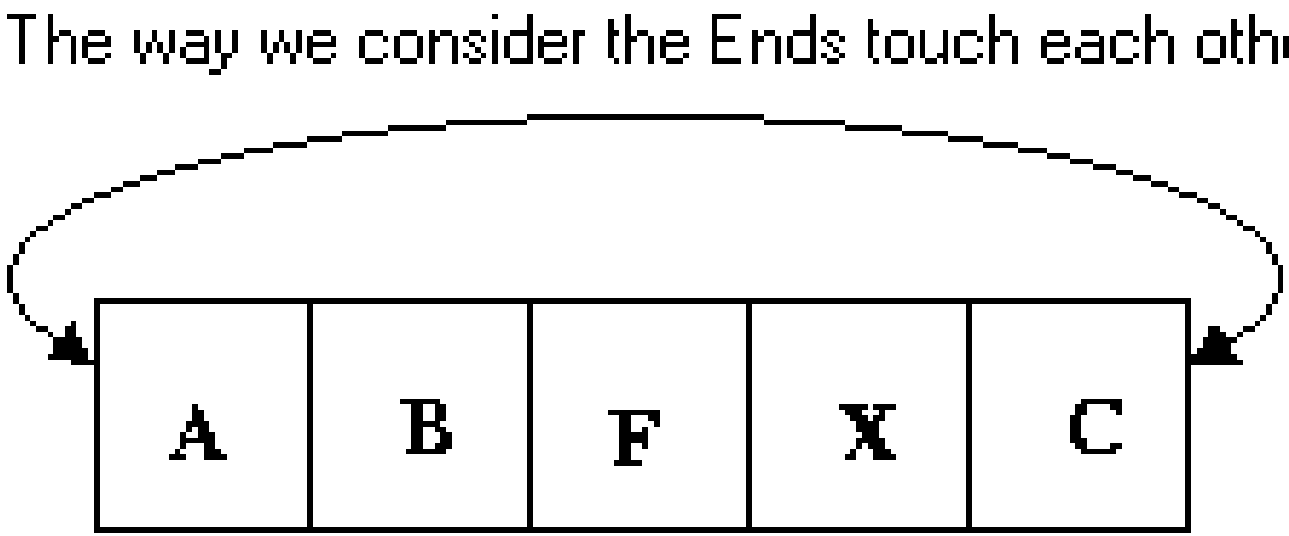
(A,B,C,D,E,F,X) are cells

Each cell has a defined neighborhood.

For example, in a one dimension cellular automaton, a neighborhood of radius one for a given cell would include the cell to the immediate right and the cell to the immediate left.

The cell itself may or may not be included in the neighborhood.

The cells on the end may (or may not) be treated as "touching" each other as if the line of cells were circular.
If we consider them as they touch each other, then the cell (A) is a neighbor of cell (C)

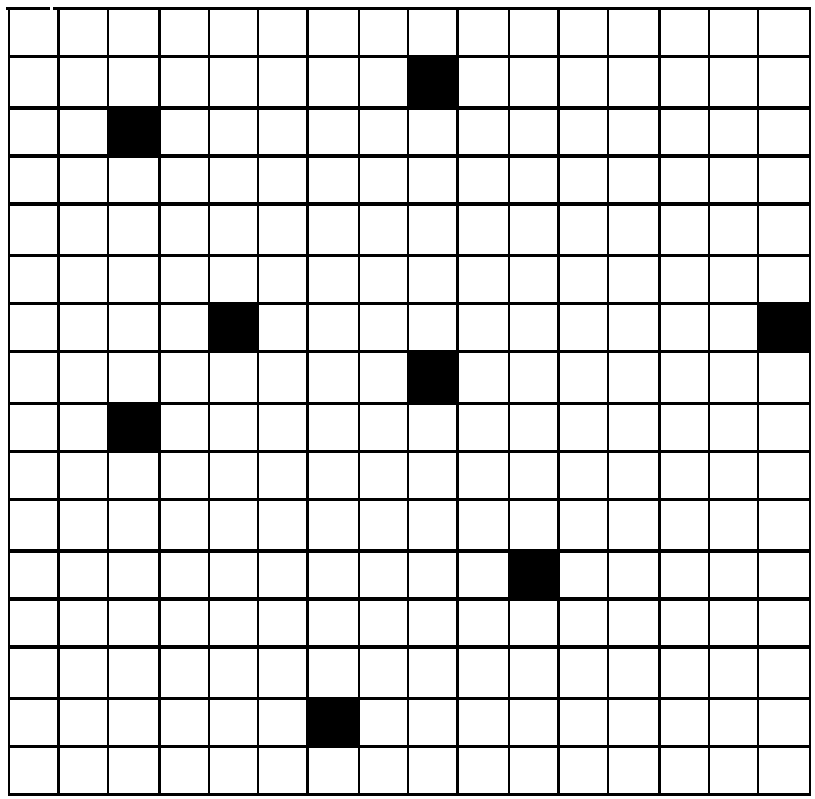


Let us take an example to make things clear.

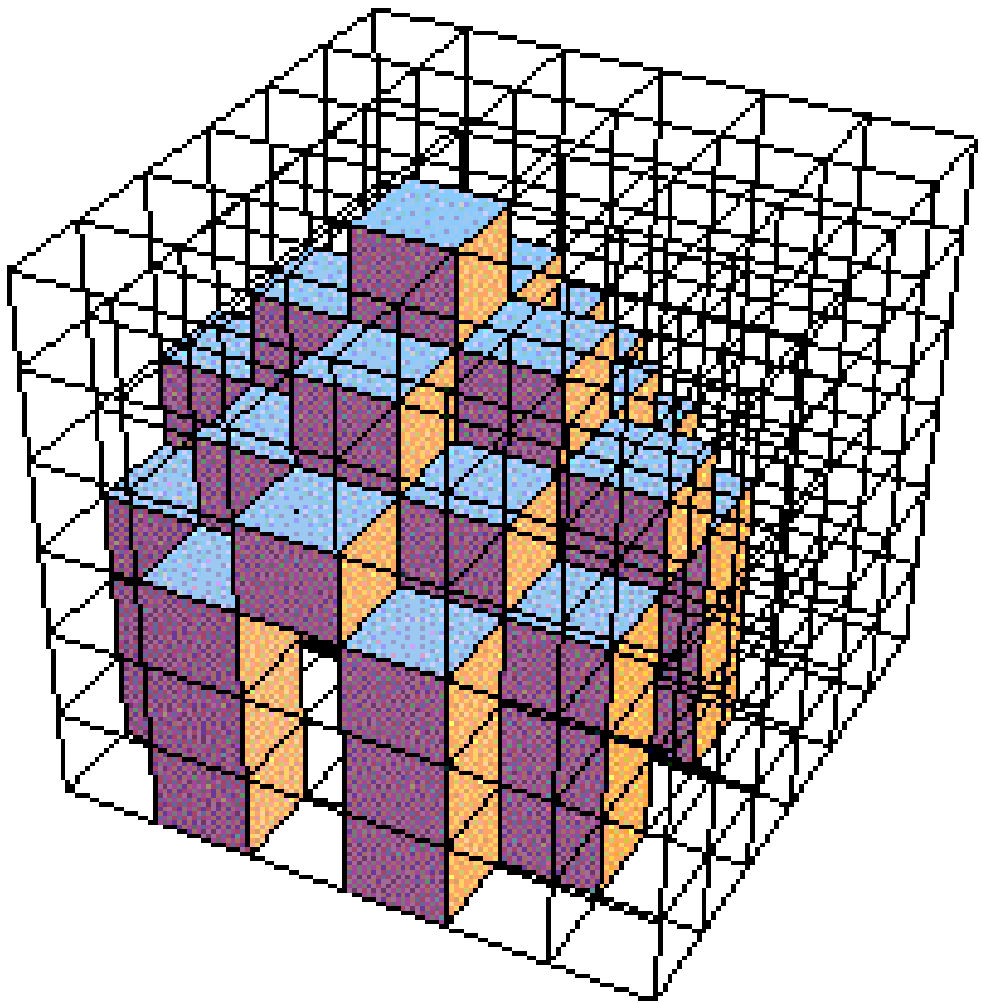
Here is a 2-d model, with 256 cells, each cell in this example can be in either (0 or 1) state,

State 1 is encoded with color black, 0 with white.

Each cell has eight neighbors (excluding itself).



One difficulty with three-dimensional cellular automata is the graphical representation (on two-dimensional paper or screen)

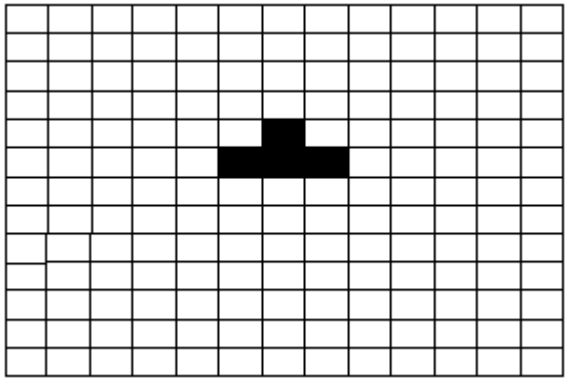


In the initial configuration of the cellular automata, each cell is assigned a "starting" value from the range of possible values.

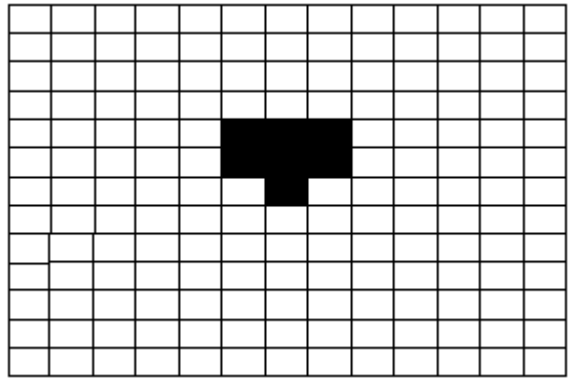
For example, if the range of possible values is 0 or 1, then each cell would be assigned a 0 or a 1 in the initial configuration.

Each value represents a color to the computer.

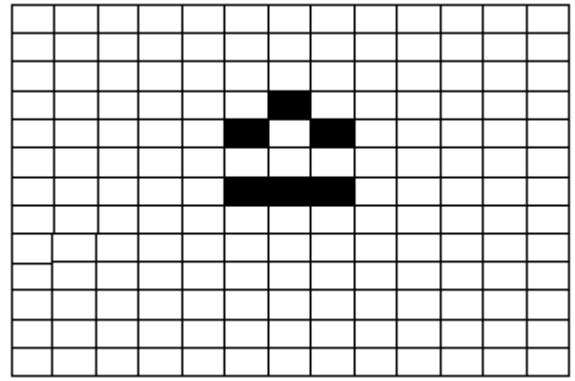
Each cell is associated a transition rule.



Initial



Step1



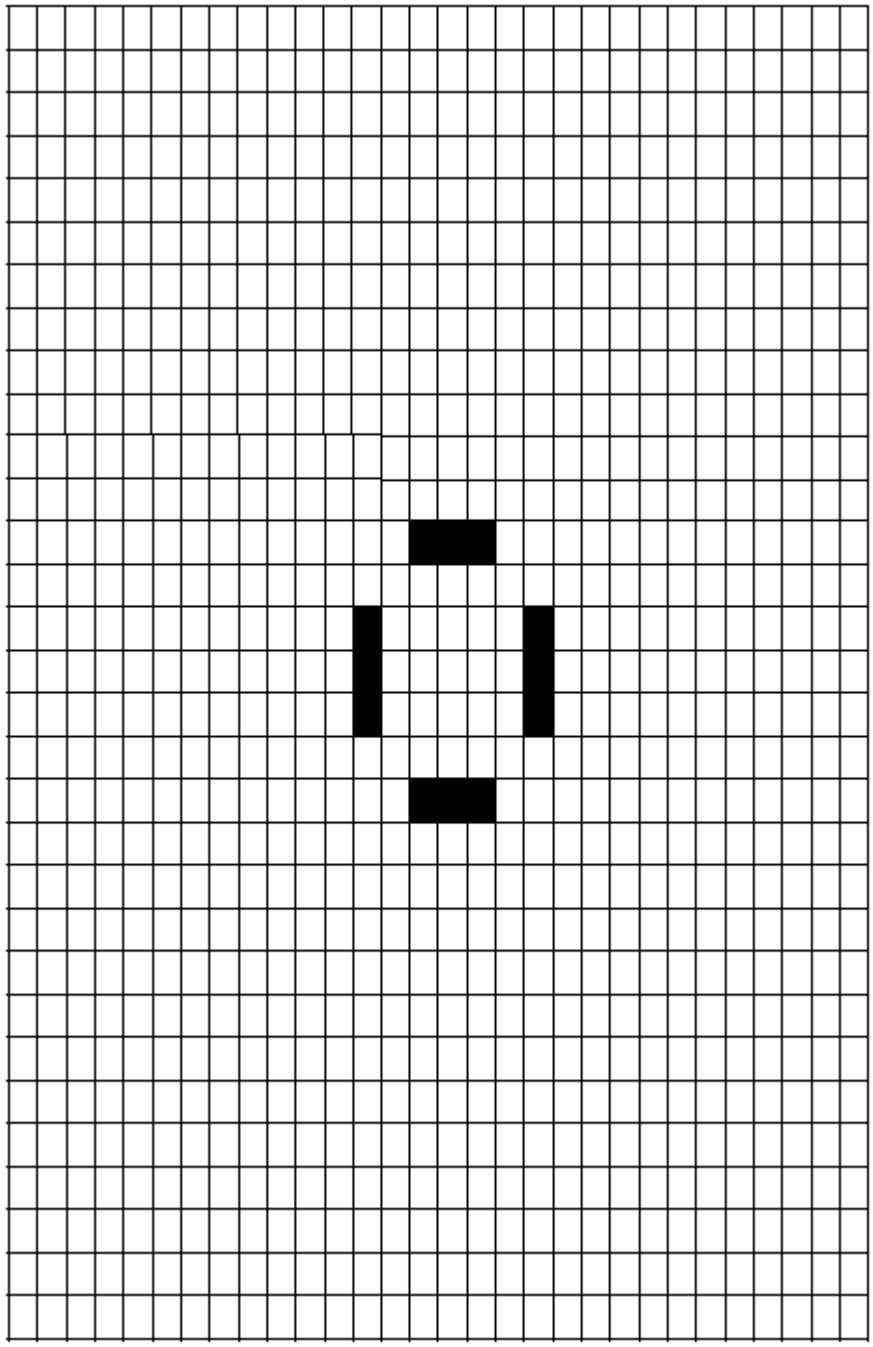
Step2

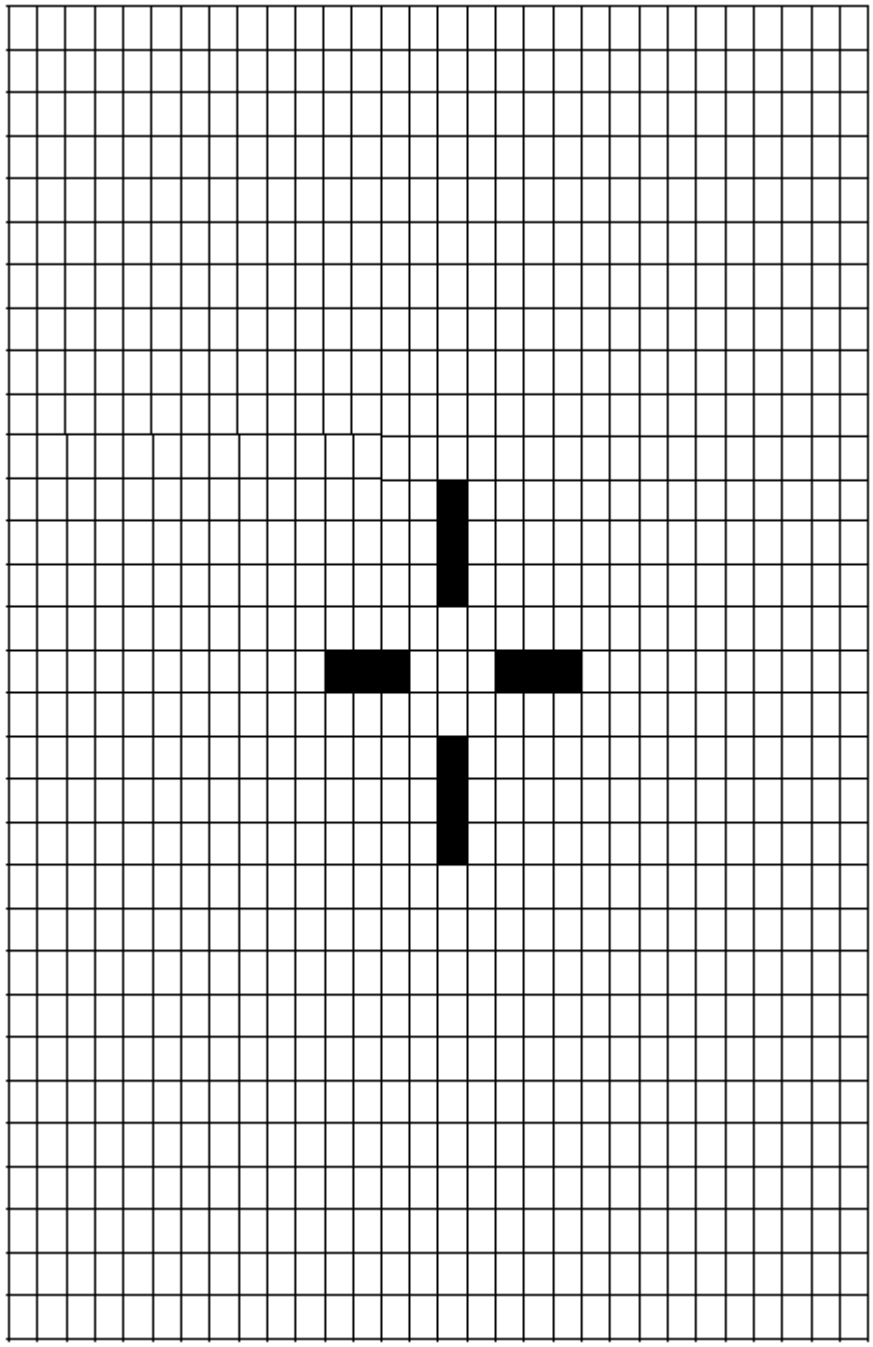
Here is an example

In this example, the initial configuration for all the cells is state 0, except for 4 cells in state 1.

- The transition rule for this example, is :
  - a cell stays in state 1 (black), if it has two or three black neighbors.
  - a cell changes to black, if it has exactly three black neighbors.

The next slide shows animation for this example





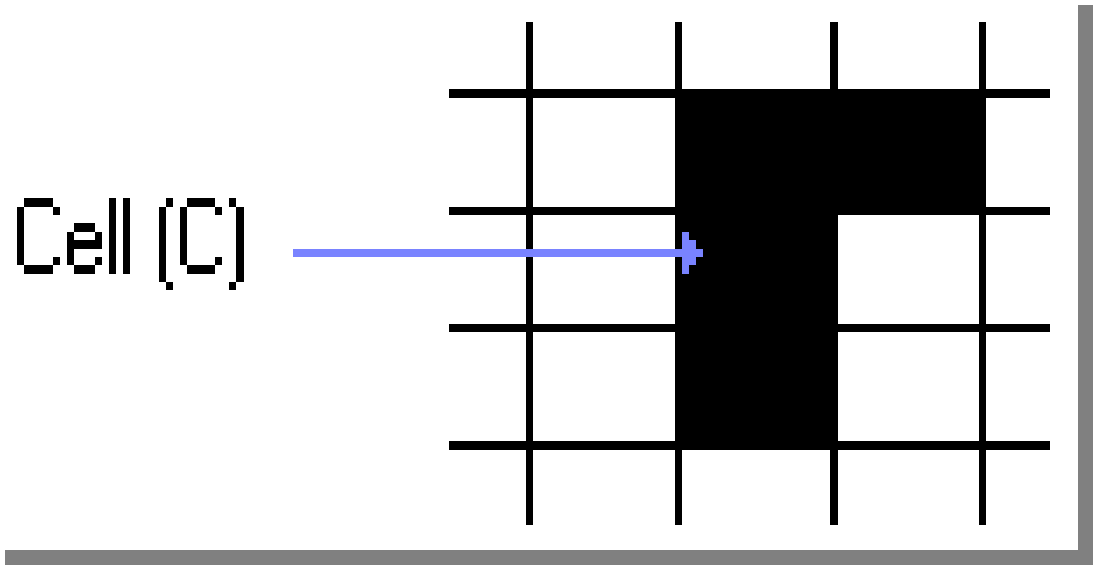
In this example:

- **Each cell has 8 neighbors,**
- **Each cell can be in one possible value at any given time,**
  - **the transition takes place in discrete times (imagining a clock feeds all the cells will help),**
- **Each State is encoded with a unique color,**
  - **The transition rule takes as input the present states (i.e., the present values) of all of the cells in a given cell's neighborhood and generates the next state (i.e., the next value) of the given cell.**
  - **When applied to all of the cells individually in a cellular automata, the next state of the whole cellular automata is generated from the present state.**
  - **Then the next state of the cellular automata is copied to the (new) present state and the process is repeated for as many clock cycles as desired.**

Let us now find some real-life examples!!!

- **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."**
- **What do these expressions mean?**

- **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)).



- Let’s consider the above example to explain what NineSume & EightSum are.
- 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, NineSume is 4.

- **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.**

- **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.**
- **Let us discuss this real-life example.**

How many different eight-neighbor 1-bit totalistic rules are there?

A rule like this is completely specified by a ten-entry lookup table which gives the new cell value for each of the ten possible neighborhood NineSums.

Each of the entries has to be 0 or 1, so filling in such a lookup table involves making ten consecutive binary decisions, which can be done in 2^{10} different ways.

Each time a cell is updated, the NineSum of the cell and its eight neighbors is formed.

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".

Here is the lookup table for this simple majority rule.

Majority: Totalistic Code 1111100000b = 992d										
NineSum	0	1	2	3	4	5	6	7	8	9
NewState	0	0	0	0	0	1	1	1	1	1

At this time we have an idea about what the Cellular Automata is.

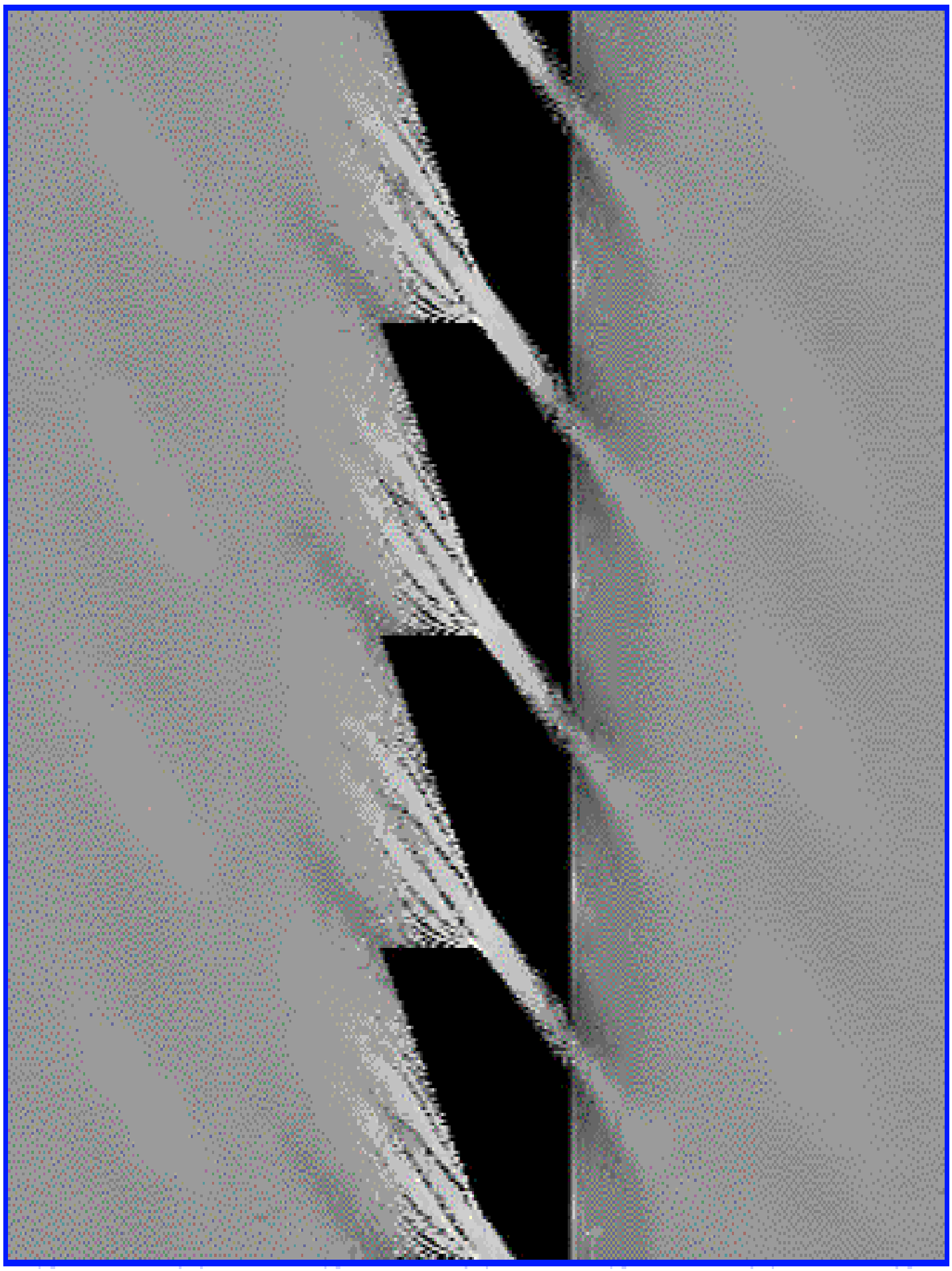
Let us now try to get closer to the basic digital logic aspects

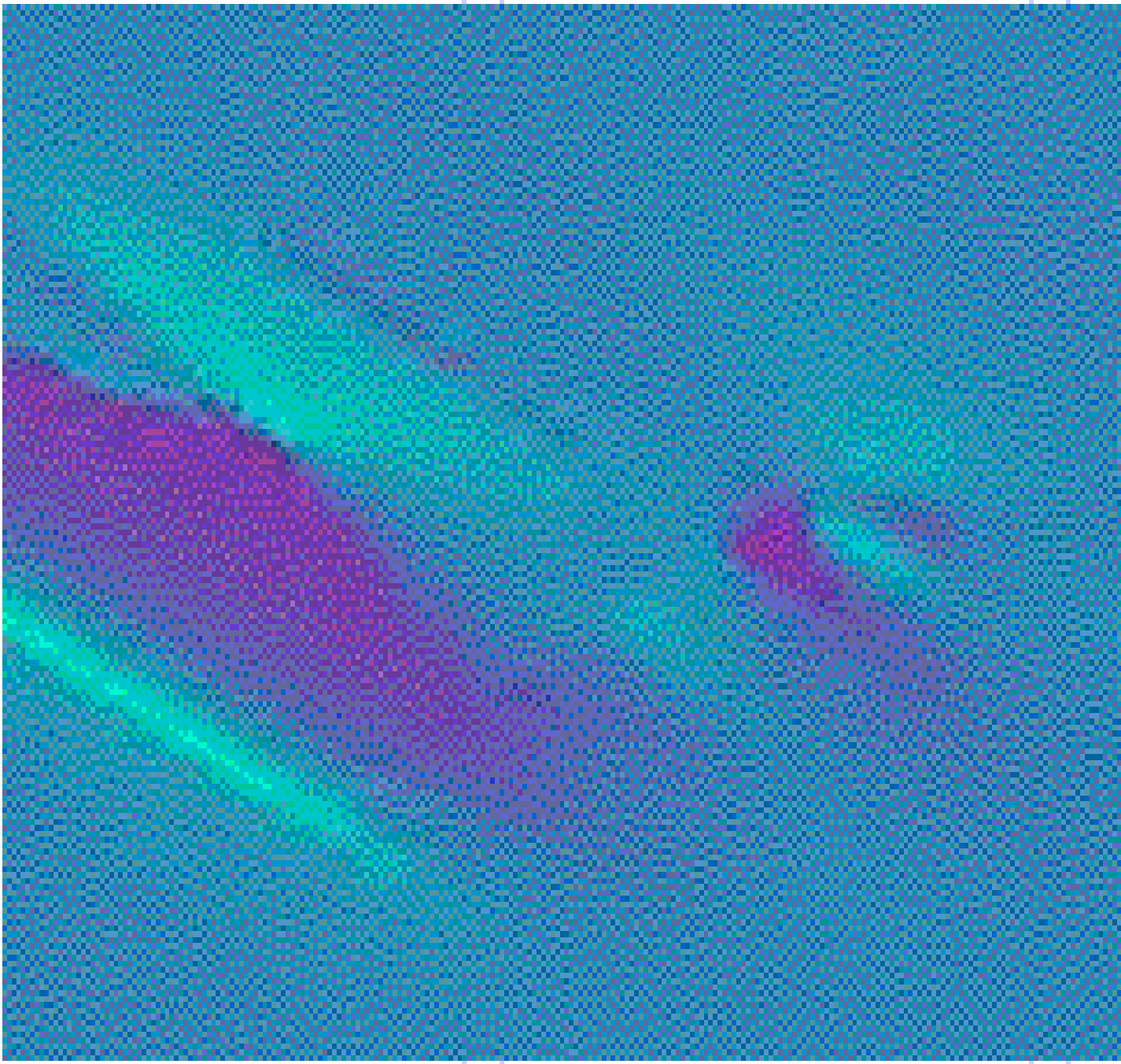
and find a different definition for Cellular Automata.

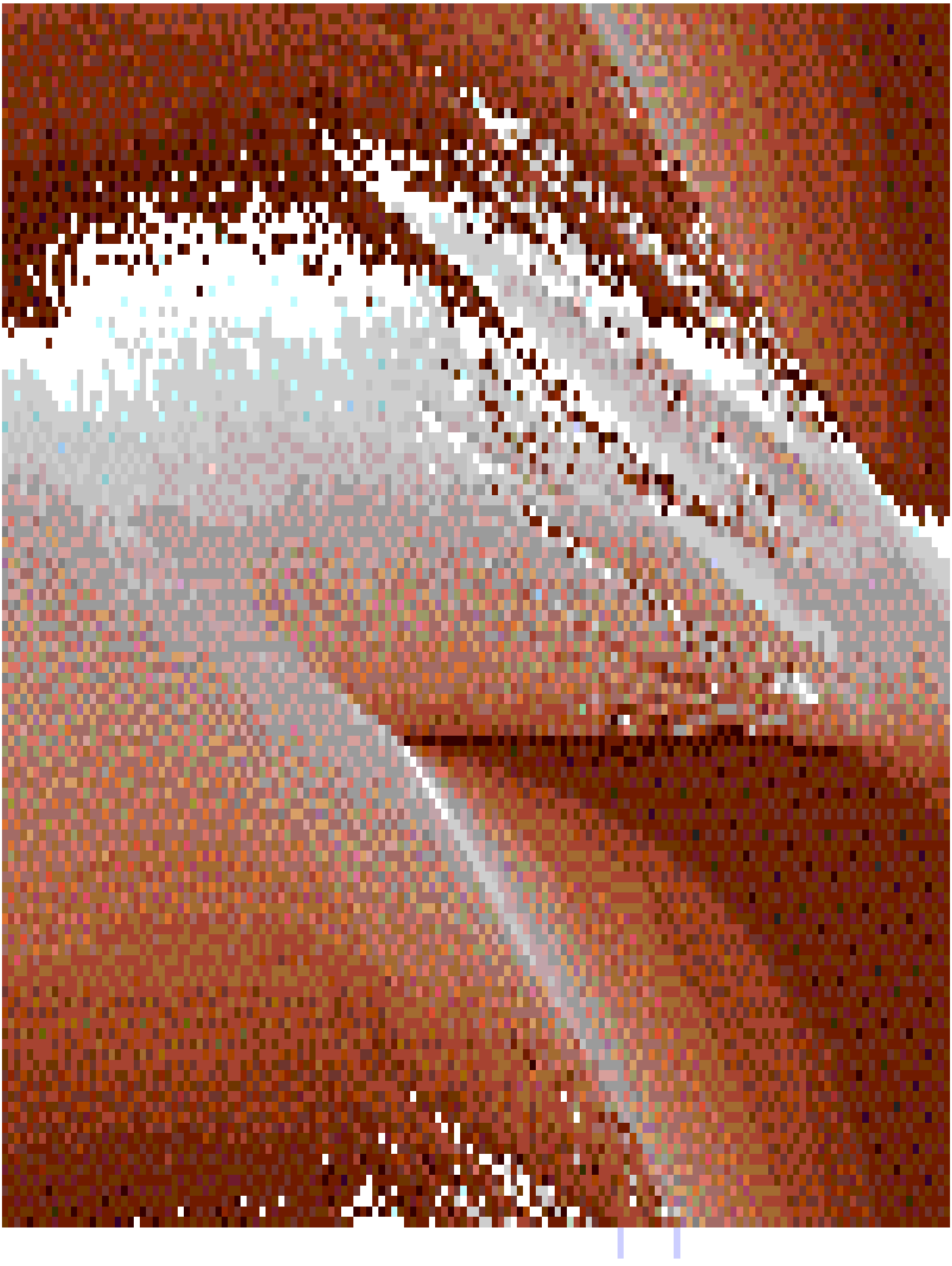
- **Let us first try to define the cellular automata as follows:**
  - **it is a Finite State Machine, with one transition function for all the cells,**
  - **this transition function changes the current state of a cell depending on the previous state for that cell and its neighbors.**
- **All we can do is to design the transition function and set the initial state.**
- **Cells here use Flip-Flops.**
- **In general, registers.**

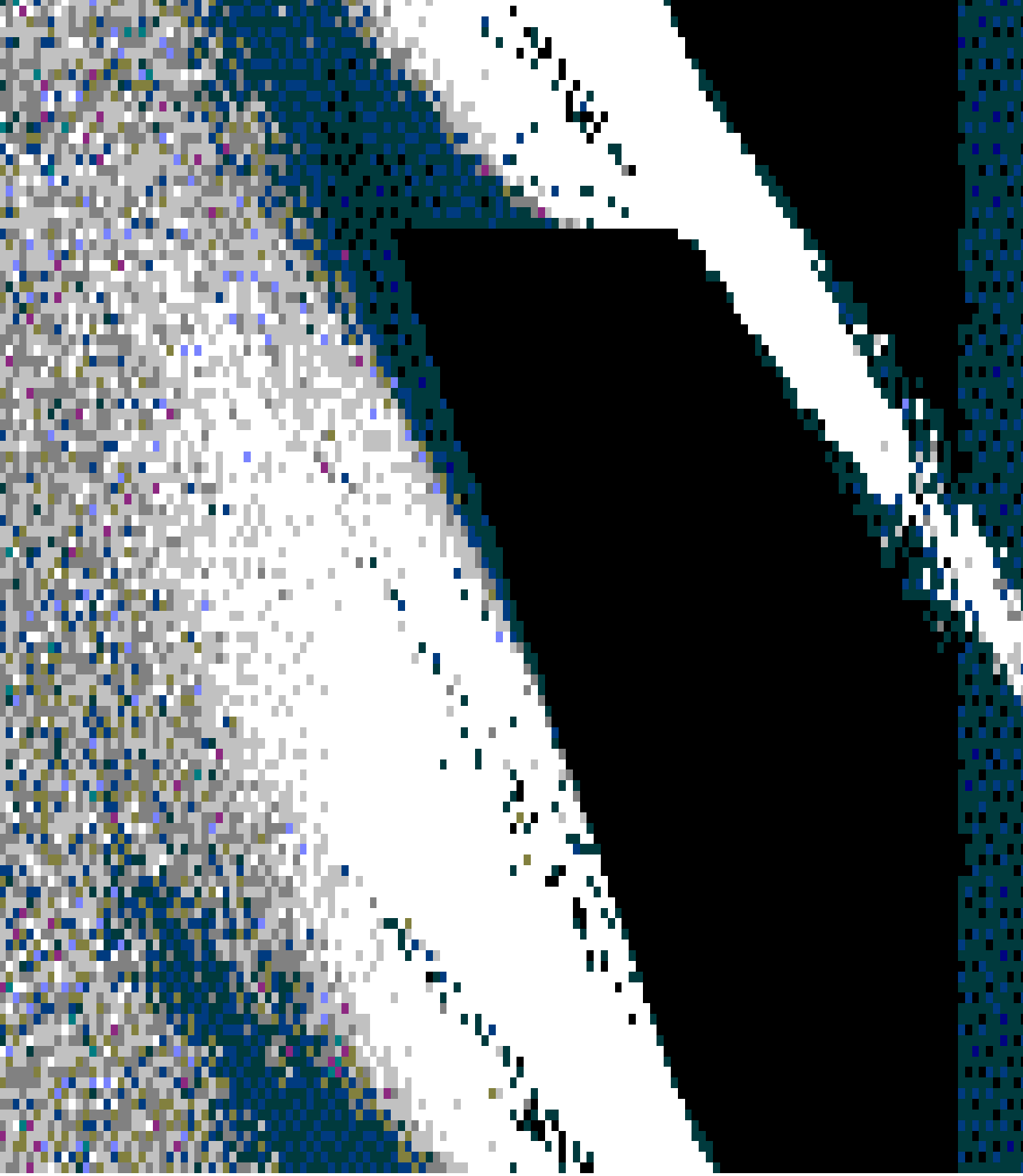
- **The advantage of the CA over the FSMs is:**
  - **that each cell uses as many data as number of neighbor cells to calculate its next state which represents an information that will be available for all neighbors including itself,**
  - **and the goal is to design the transition function for these cells**
  - **there is no State Assignment problem because all cells are the same**
  - **transition function is the same for all cells.**
- **But the complexity is to design the Transition function that fits our application!!!**

- **Images that represent different cellular automata:**
  - **each cell can be in state (0 or 1or 2 …..or n),**
  - **each state is encoded with a unique color,**
  - **each cellular automata, has its own transition rule.**









Sources

Nouraddin Alhagi, class 572