２. Cellular Automata

automata： golem → automatic machine, sequential machine

［John von Neuman］

Autonomously replicating automata： self-propagating program with local neighborhood rule, discrete system

⇩

Cellular automata

２．１ general items of cellular automata

（１）definition of cellular automata

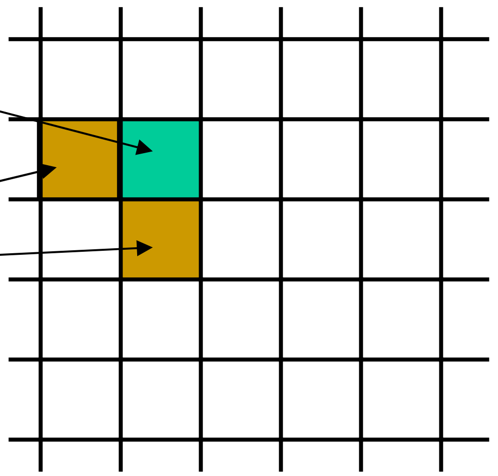
Software ‘objects' (short algorithms) running under the control of a special computer program (such as the ‘Game Of Life') and represented on the monitor screen as small squares, triangles, or other shapes called ‘cells'. Each cell is connected to its neighboring cells by a set of simple rules which also govern which state out of several a cell (such as color, movement, and replication) can have at any given moment. The program begins usually with one or a few cells which trace a simple and often predictable pattern. But as the program progresses (simulates changing environments and random mutations) the number of cells increases and the pattern becomes exceedingly intricate and completely unpredictable, sometimes mimicking the behavior of complex adaptive systems (such as live biological cells). In some patterns, for example, cells 'fight' over temitory and may even ‘kill' one another (stop the growth of the pattern). The usefulness of cellular automata (at present) lies in their ability to model certain biological, economic, physical, and sociological phenomenon, and in showing how their components or parts may interact if conditions change. The concept of cellular automata was proposed by the Hungarian-US mathematician John von Neumann (1903-1957) and developed and put into practice by the UK mathematician John Conway (born 1937) in his 'Game Of Life simulation.' Called also artificial life (A-life).

① discrete system
lattice cite having various initial value

② new value based on some values of local neighboring cites

⇒ developing by discrete time step

⮤ Assuming states in past several time steps



(2)CA lattice

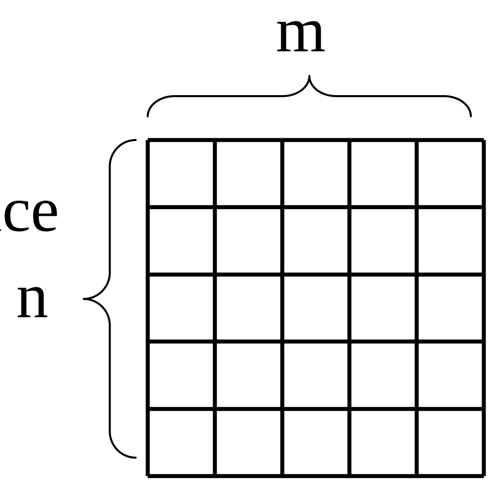
[one dimension]

Cellular automata＝linear list Table[expr, {i, 1, s}]

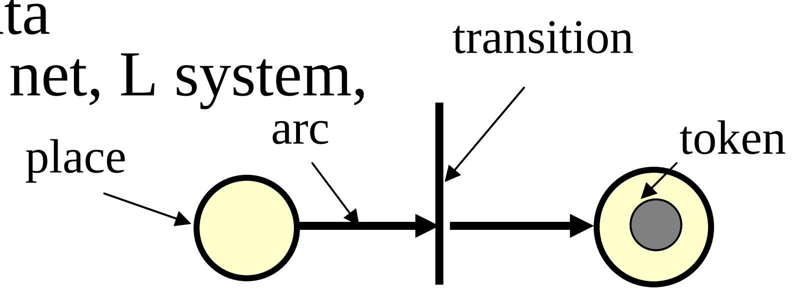
[two dimension]

rectangular lattice, triangle lattice, hexagonal lattice

Rectangular lattice n×m
Table[expr, {i, 1, n}, {j, 1, m}]



Various two dimensional cellular automata
game of life, lattice gas automata, Petri net, L system, multi agent system



（Petri net）

A **Petri net** (also known as a **place/transition net** or **P/T net**) is one of several mathematical modeling langauges for the description of discrete distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. discrete events that may occur), places (i.e. conditions), and directed arcs (that describe which places are pre- and/or postconditions for which transitions). Petri nets were invented in August 1939 by Carl Adam Petri – at the age of 13 – for the purpose of describing chemical processes.

Like industry standards such as UML activity diagrams, BPMN and EPCs, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis.

A Petri net consists of *places*, *transitions*, and *directed arcs*. Arcs run between places and transitions, never between places or between transitions. The places from which an arc runs to a transition are called the *input places* of the transition; the places to which arcs run from a transition are called the *output places* of the transition. Places may contain any non-negative number of tokens. A distribution of tokens over the places of a net is called a *marking*. A transition of a Petri net may *fire* whenever there is a token at the end of all input arcs; when it fires, it consumes these tokens, and places tokens at the end of all output arcs. A firing is atomic, i.e., a single non-interruptible step.

Execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire. If a transition is enabled, it may fire, but it doesn't have to.
Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurent behavior of distributed systems.

Application areas

Software design, Workflow management , Data analysis, Concurrent programming , Reliability engineering , Diagnosis , Discrete process control

Syntax

A **Petri net graph** (called *Petri net* by some, but see below) is a 3-tuple (S,T,W) , where

S is a finite set of *places*

T is a finite set of *transitions*

S and T are disjoint, i.e. no object can be both a place and a transition

$W:(S\times T)\cup(T\times S)\rightarrow \mathrm{N}$ is a multiset of arcs, i.e. it defines arcs and assigns to each arc a non-negative integer *arc multiplicity*; note that no arc may connect two places or two transitions.

The *flow relation* is the set of arcs: $F=\{(x,y)|W(x,y)>0\}$. In many textbooks, arcs can only have multiplicity 1, and they often define Petri nets using F instead of W.

A Petri net graph is a bipartite multigraph $(S\cup T,F)$ with node partitions S and T.

The *preset* of a transition t is the set of its *input places*: ${}^\bullet t=\{s\in S|W(s,t)>0\}$; its *postset* is the set of its *output places*: $t^\bullet=\{s\in S|W(t,s)>0\}$

A *marking* of a Petri net (graph) is a multiset of its places, i.e., a mapping M:$S\rightarrow$N. We say the marking assigns to each place a number of *tokens*.

A **Petri net** (called *marked Petri net* by some, see above) is a 4-tuple (S, T, W, M_0), where

(S,T,W) is a Petri net graph;

M_0 is the *initial marking*, a marking of the Petri net graph.

Execution semantics

The behavior of a Petri net is defined as a relation on its markings, as follows.

Note that markings can be added like any multiset:

The execution of a Petri net graph G=(S, T, W) can be defined as the *transition relation* $\rightarrow_G$ on its markings,

In words:

- firing a transition t in a marking M consumes $W(s,t)$ tokens from each of its input places s, and produces $W(t,s)$ tokens in each of its output places s
- a transition is *enabled* (it may *fire*) in M if there are enough tokens in its input places for the consumptions to be possible, i.e. iff $\forall s : M(s) \geq W(s,t)$.

$\rightarrow_G$

We are generally interested in what may happen when transitions may continually fire in arbitrary order.

We say that a marking M' *is reachable from* a marking M *in one step* if $M \rightarrow_G M'$; we say that it *is reachable from M* if $M \rightarrow_G^* M'$, where $\rightarrow_G^*$ is the transitive closure of $\rightarrow_G$; that is, if it is reachable in 0 or more steps.

For a (marked) Petri net , we are interested in the firings that can be performed starting with the initial marking M0. Its set of *reachable markings* is the set

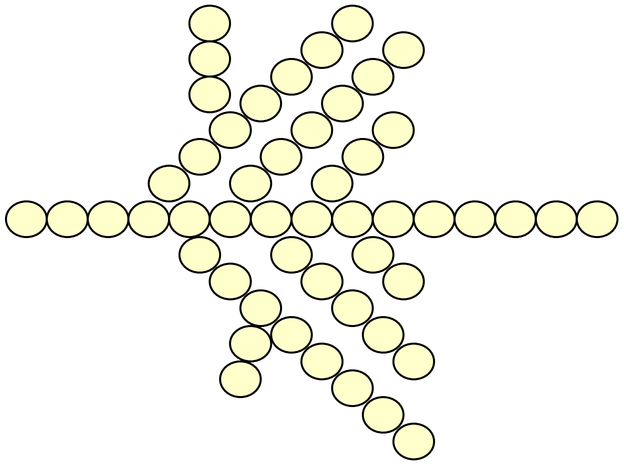
The *reachability graph* of N is the transition relation   restricted to its reachable markings $R(N)$. It is the state space of the net.

A *firing sequence* for a Petri net with graph G and initial marking M0 is a sequence of transitions   such that . The set of firing sequences is denoted as $L(N)$.

(L system)

L system (Lindenmayer system, Universiteit Utrecht) is a kind of formal grammar in information science, and an algorithm which represents various natural structure, such as growth of a plant. Besides it is used to generate iterated function system, such as self similar figure and fractal figure.

Structure of L system



Chomsky hierarchy, tuple

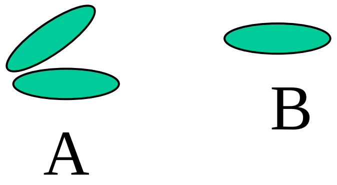
G = {V, S, ω, P},

V(letter): variable, S: constant,

ω: initial state of V, P: rule changing V

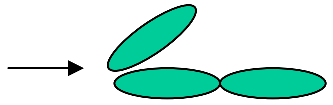
Examples of L system

(alga) V: A, B S: - , ω: A, P: (A→AB), (B→A)



n=0: A

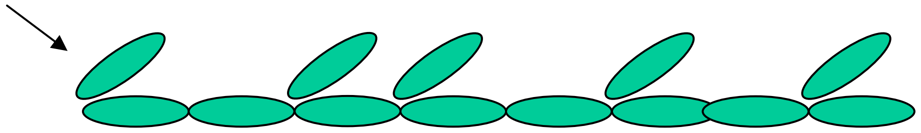
n=1: AB



n=2: ABA

n=3: ABAAB

n=4: ABAABABA



Fibonacci series (1 1 2 3 5 8 13 21 34 55 89 - - -)

V: A, B   *S*: - ,   *w*: A,   *P*: (A$\rightarrow$B), (B$\rightarrow$AB)

n = 0 : A

n = 1 : B

n = 2 : AB

n = 3 : BAB

n = 4 : ABBAB

n = 5 : BABABBAB

n = 6 : ABBABBABABBAB

n = 7 : BABABBABABBABBABABBAB

Cantor set

V: A, B   *S*: - ,   *w*: A,   *P*: (A$\rightarrow$ABA), (B$\rightarrow$BBB)

n = 0 : A

n = 1 : ABA

n = 2 : ABABBBABA

n = 3 : ABABBBABABBBBBBBBBABABBBABA

Koch curve

V: F   *S*: +, - ,   *w*: F,   *P*: (F$\rightarrow$F+F-F-F+F)

n=0: F

n=1: F+F-F-F+F

n=2: F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F

n=3: F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+ F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-
F+F+F+F-F-F+F- F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F- F+F-F-F+F+F+F-F-F+F-
F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+ F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F

（3） neighborhood of CA

von Neumann neighborhood

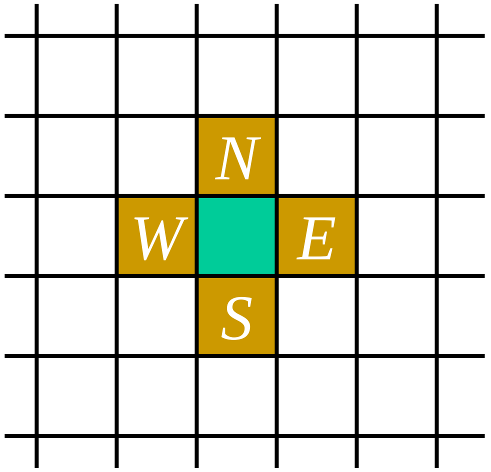
It is the smallest symmetric 2D aligned neighborhood usually described by directions on the compass $N = \{N,W,C,E,S\}$ sometimes the central cell is omitted.

Formal definition

Formally the von Neumann neighborhood is the set of neighbors

$$N = \{\{0,-1\},\{-1,0\},\{0,0\},\{+1,0\},\{0,+1\}\}$$

or a subset of the rectangular neighborhood size $k_x = k_y = 3$ with the output cell at the center $k_{0x} = k_{0y} = 1$.



Moore neighborhood

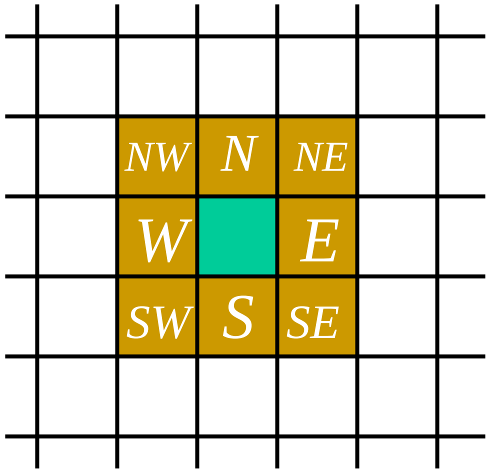
Is a simple square (usually 3 × 3 cells) with the output cell in the center. Usually cells in the neighborhood are described by directions on the compass $N = \{NW,N,NE,W,C,E,SW,S,SE\}$ sometimes the central cell is omitted.

Formal definition

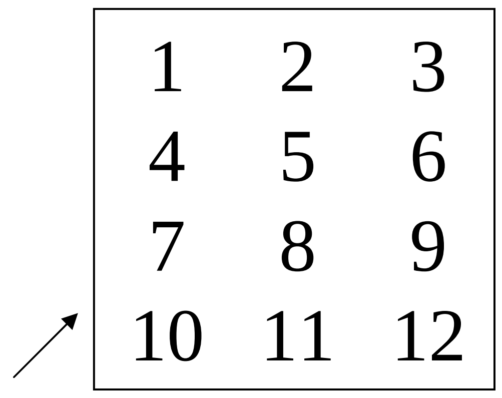
Formally the Moore neighborhood is the set of neighbors

$$N = \{\{-1,-1\},\{0,-1\},\{1,-1\},\{-1,0\},\{0,0\},\{+1,0\},\{-1,+1\},\{0,+1\},\{1,+1\}\}$$

or simply a square size $k_x = k_y = 3$ with the output cell at the center $k0x = k0y = 1$.

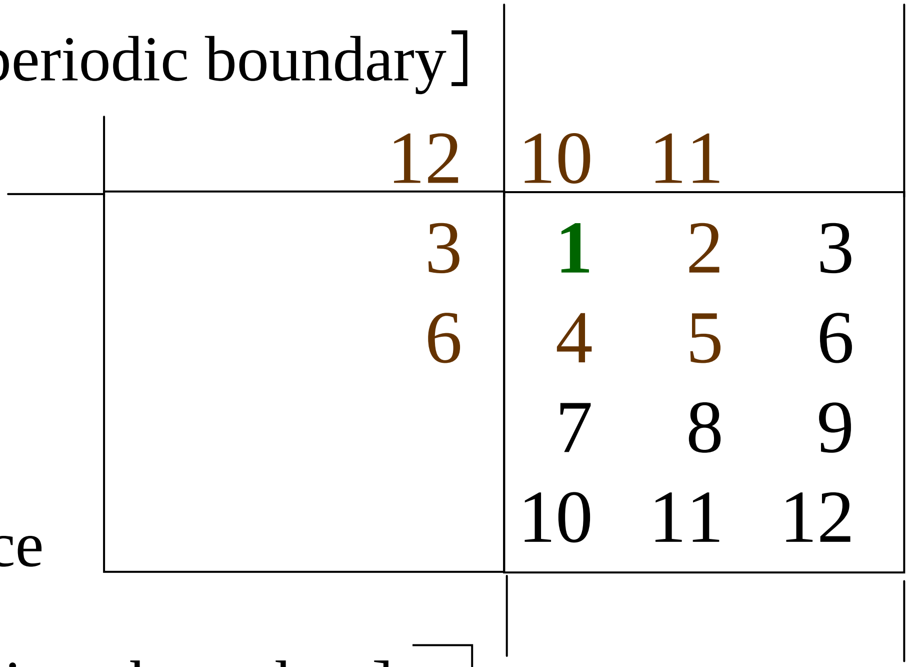


（4）boundary condition

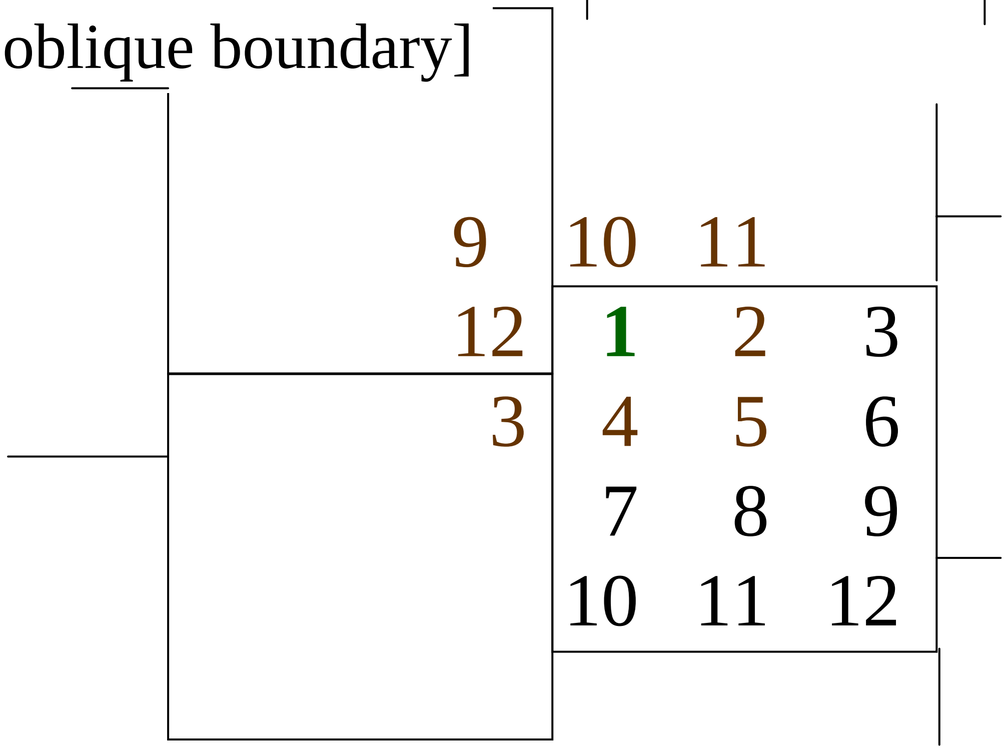


Moore neighborhood of simple lattice

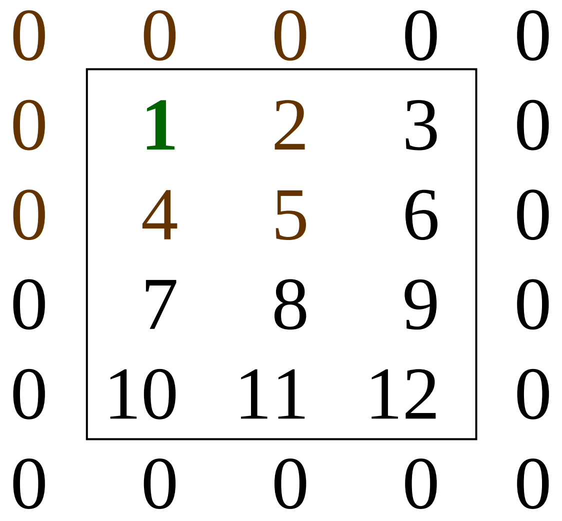
[periodic boundary]



[oblique boundary]

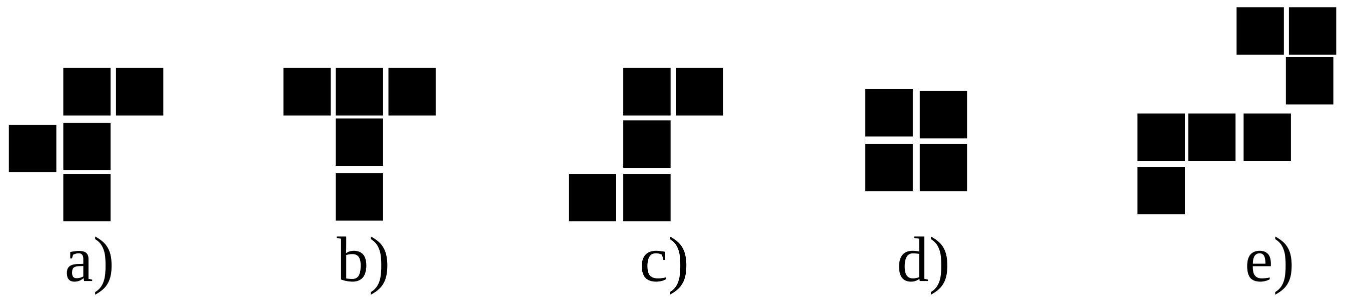


[absorption boundary]



2．2 *The Game of Life*

The *Game of Life* is a simple mathematical system invented by John Horton Conway (Cambridge University). It is an infinite two dimensional toy universe, which we call here simply C. Parts of this universe can be easily embedded on the computer and observed on the display screen. A pixel of the screen corresponds to its smallest element and can be in one of two colors. To see the universe evolve all you have to do is to specify the initial state. The pattern of b) can be an initial state. Except for a few black pixel/elements all other elements of the universe are white. After evolution starts you see a succession of new states. After several discrete time steps (generations) the pattern of b) evolves to a simple periodic pattern of period 2 known as *traffic lights*. The pattern of c) disappears after only three generations. The *r-pentomino* pattern of a) is growing.



・ pioneer of artificial life system
・ “intelligent agent” on a computer
・ program of world’s first parallel computer “connecting machine”

・ transition rule of cellular automata ⇒ coding in gene

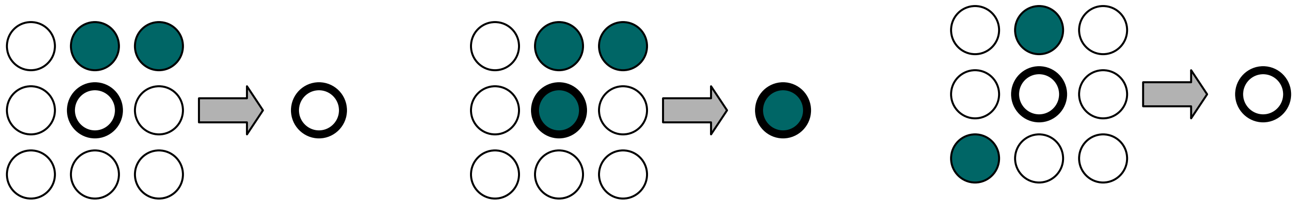
・ large scale cellular automata ⇒ neural network

・ human’s brain＝network constituted of several billion elements
⇒ autonomous constitution

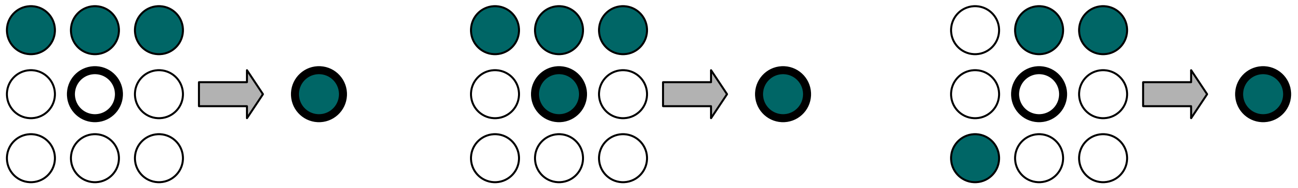
（rules）
・ periodic boundary condition, two dimensional square lattice
・ Moore neighborhood
・ It dies when there is no mate or there are excessive mates in neighborhood it lives when the number of mates is appropriate.

１(●)，０(○)

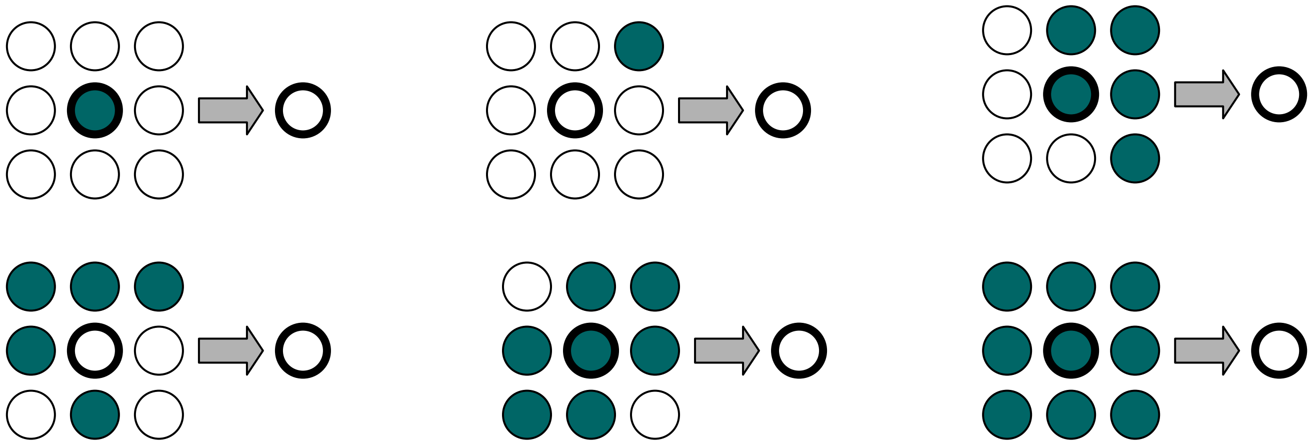
（１）If it has two 1 in neighborhood, it does not change.



（２）If it has three 1 in neighborhood, it becomes 1.

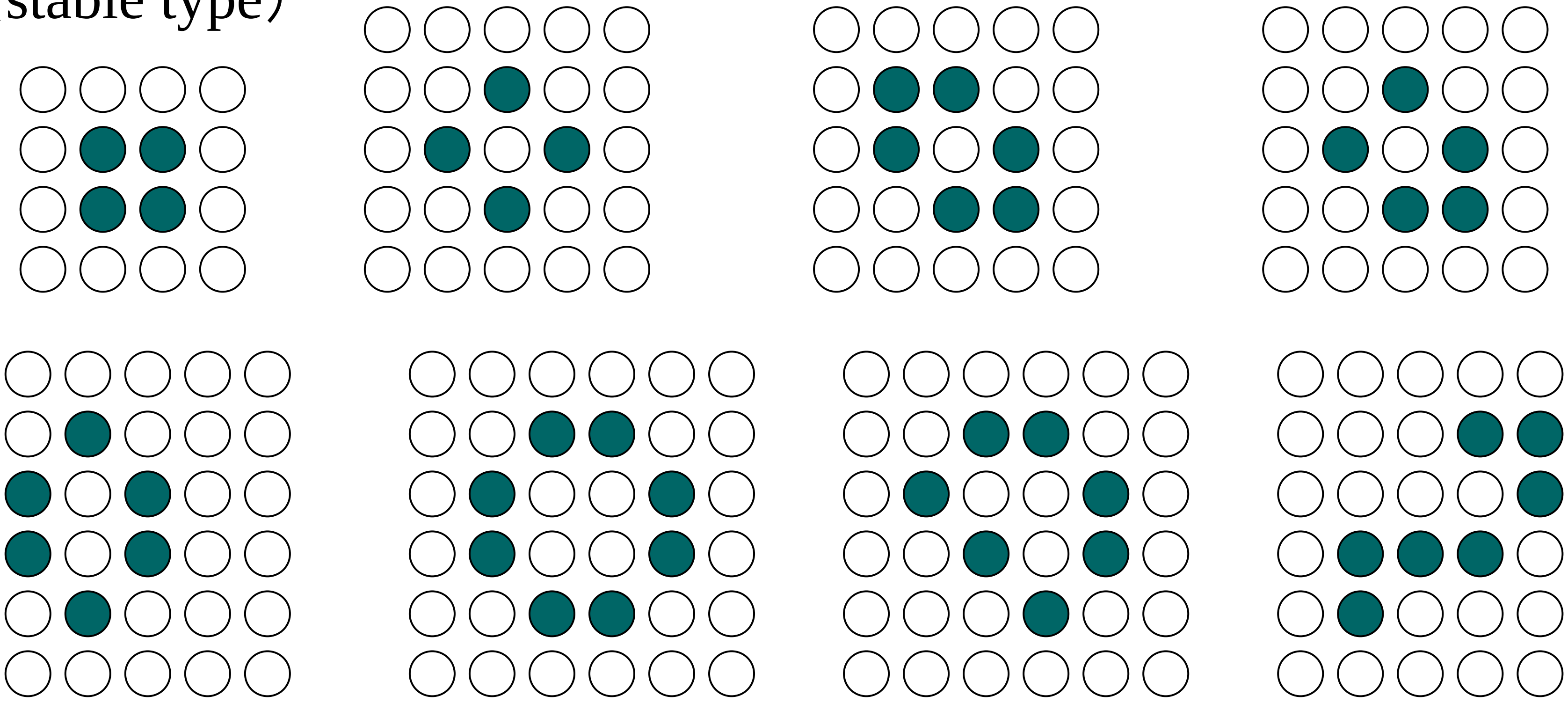


（３）In different cases from (1)(2) it becomes 0.

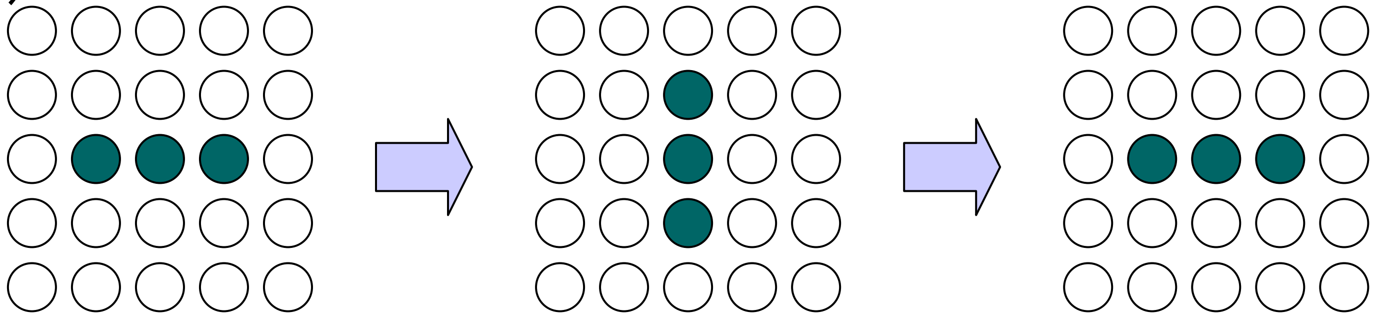


[various kinds of patterns]

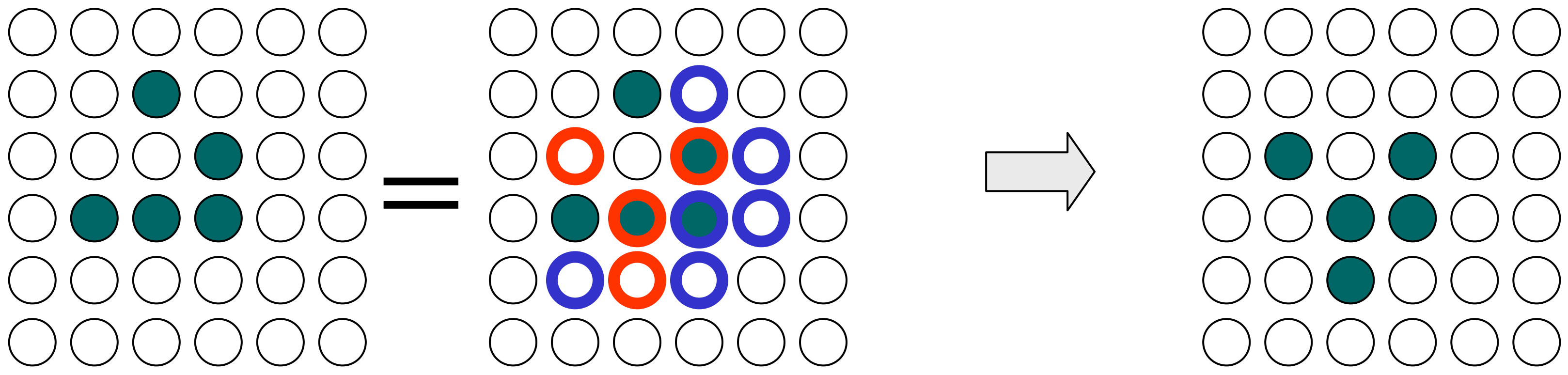
(stable type)



(oscillating type)

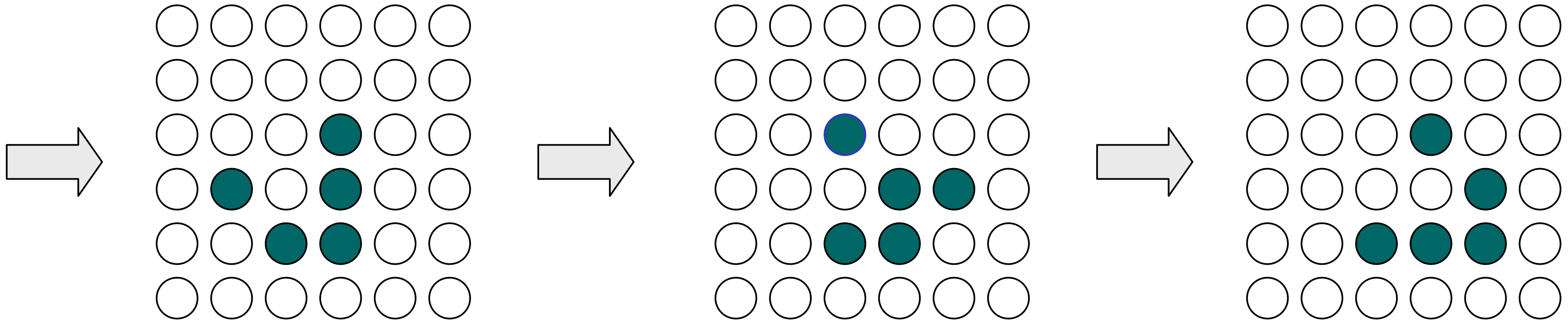


(moving type) glider



Number of 1 in neighborhood

○ 3 ○ 2



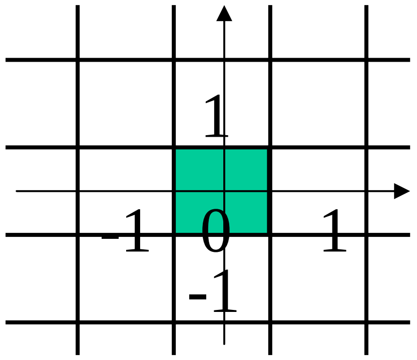
Algorithm (mathematica）

(1)Initial arrangement

```
initconfig = Table[Random[Integer], {s}, {s}]
```

(2) Assign the number of most neighboring living cites to matrices

```
In[1] := livingNghbrs[mat_] := Apply[Plus,
            Map[RotateRight[mat, #]&,
               {{-1, -1}, {-1, 0}, {-1, 1}, {0, -1},
               {0, 1}, {1, -1}, {1, 0}, {1, 1}}]]
```



(example)

```
In(2) := (board = Table[Random[Integer], {4}, {4}]) //MatrixForm
Out[2]//MatrixForm =  0   1   1   1
                      0   1   0   1
                      0   0   0   0
                      1   0   1   0
```

```
In[3] := (livingNghbrs[board]) //MatrixForm
 Out[3]//MatrixForm =  5   4   5   4
                       4   2   5   2
                       3   3   3   3
                       2   4   3   4

In[4] := bc = Join[{Last[#]}, #, {First[#]}]&;
              Partition[bc[Map[bc, board]],{3, 3}, {1, 1}]//MatrixForm
    Out[4]//MatrixForm =   0 1 0   1 0 1   0 1 0   1 0 1
                           1 0 1   0 1 1   1 1 1   1 1 0
                           1 0 1   0 1 0   1 0 1   0 1 0

                           1 0 1   0 1 1   1 1 1   1 1 0
                           1 0 1   0 1 0   1 0 1   0 1 0
                           0 0 0   0 0 0   0 0 0   0 0 0

                           1 0 1   0 1 0   1 0 1   0 1 0
                           0 0 0   0 0 0   0 0 0   0 0 0
                           0 1 0   1 0 1   0 1 0   1 0 1

                           0 0 0   0 0 0   0 0 0   0 0 0
                           0 1 0   1 0 1   0 1 0   1 0 1
                           1 0 1   0 1 1   1 1 1   1 1 0
```

(3) Rule of life and death

```
update[1, 2] := 1
update[_, 3] := 1
update[_, _] := 0
```

```
In[5] := Attributes[g] = Listable;
         g[board, livingNghbrs[board]] //MatrixForm

  Out[5]//MatrixForm =        g[0, 5]   g[1, 4]   g[1, 5]   g[1, 4]
                              g[0, 4]   g[1, 2]   g[0, 5]   g[1, 2]
                              g[0, 3]   g[0, 3]   g[0, 3]   g[0, 3]
                              g[1, 2]   g[0, 4]   g[1, 3]   g[0, 4]
```

```
In[6] := update[1, 2] := 1
         update[_, 3] := 1
         update[_, _] := 0
         Attributes[update] =Listable;
         update[board, livingNghbrs[board]] //MatrixForm

Out[6]//MatrixForm =       0  0  0  0
                           0  1  0  1
                           1  1  1  1
                           1  0  1  0
```

```
update[1, 2] := 1
update[_, 3] := 1
update[_, _] := 0
```

Program of life game

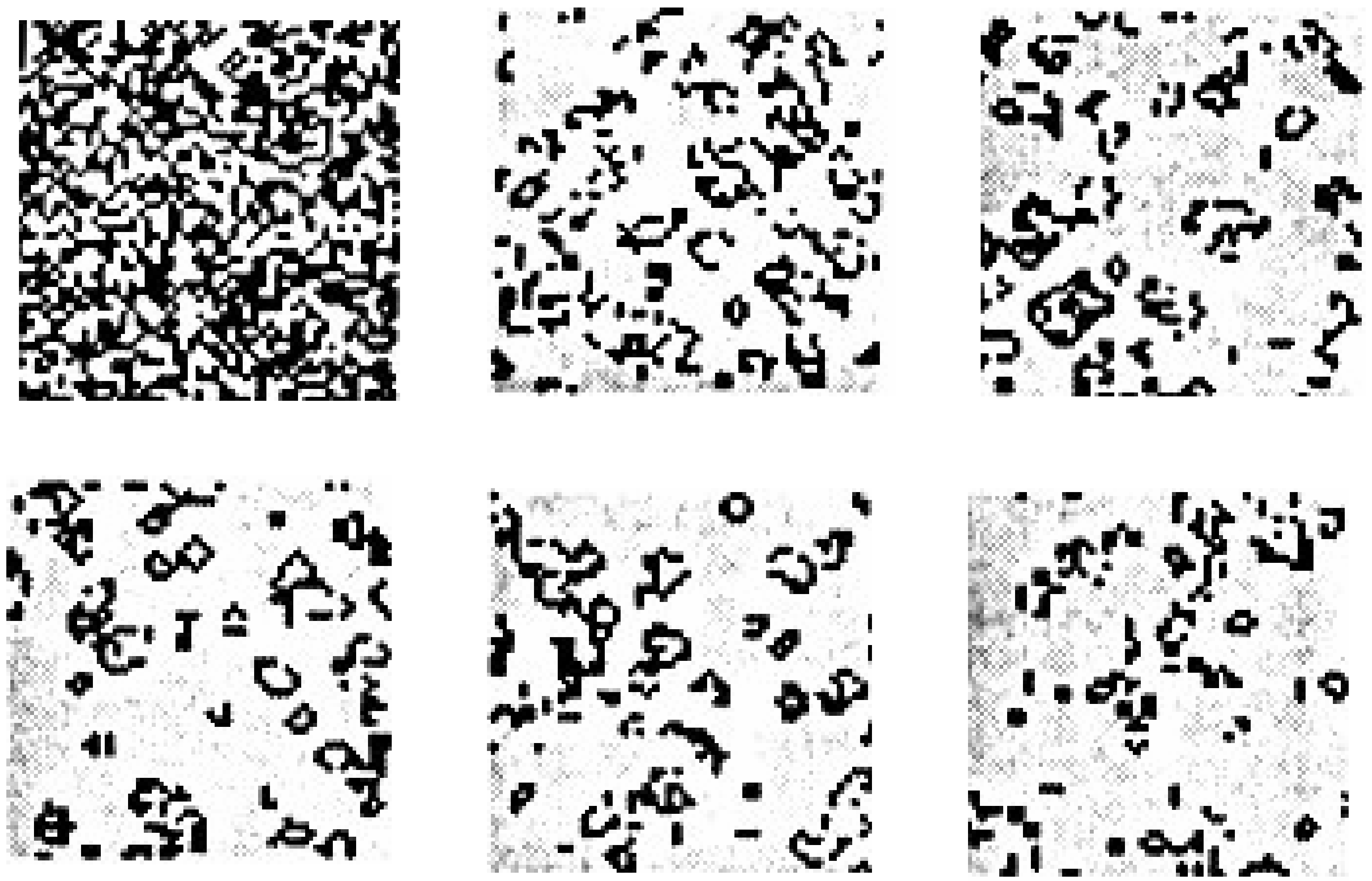
```
In[1] := LifeGame[s_, t_] :=
                Module[{initconfig, livingNghbrs, update},
                    initconfig = Table[Random[Integer], {s}, {s}];
                    livingNghbrs[mat_] :=
                      Apply[Plus, Map[RotateRight[mat, #]&,
                                       {{-1, -1}, {-1, 0}, {-1, 1}, {0, -1},
                                         {0, 1}, {1, -1}, {1, 0}, {1, 1}}]];
                      update[1, 2] := 1;
                      update[_, 3] := 1;
                      update[_, _] := 0;
                      Attributes[update] =Listable;
                      FixedPointList[update[#, livingNghbrs[#]]&,
                                      initconfig, t]
                    ]
```

（graphic representation）

```
In[3] := Showlife[list_, opts_ _ _Rule] :=
               Map[(Show[Graphics[RasterArray[
                         Reverse[list[[#]]/.
                                  {1 -> RGBColor[1, 0, 0],
                                    0 -> RGBColor[0, 0, 0]}]],
                         AspectRatio ->Automatic,
                         opts]])&,
                    Range[Length[list]]]
```

Animation cell

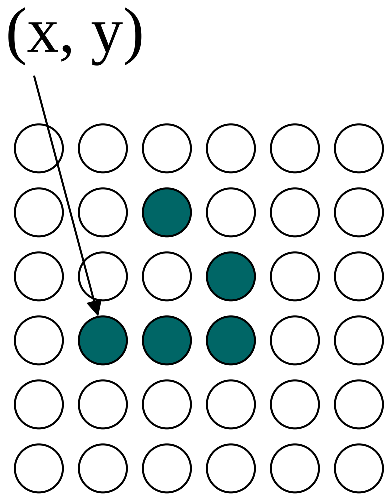
50×50 lattice



（life-forms）

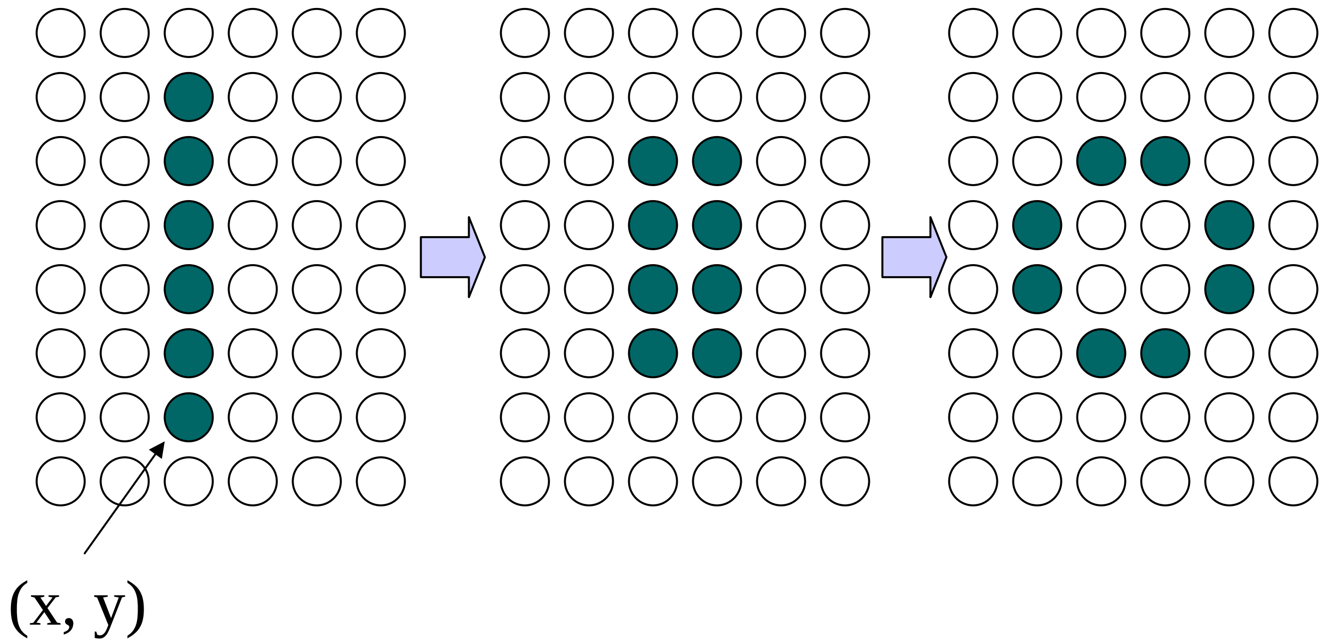
glider

glider[x_, y_] :={{x, y}, {x+1, y}, {x+2, y}, {x+2, y+1}, {x+1, y+2}}



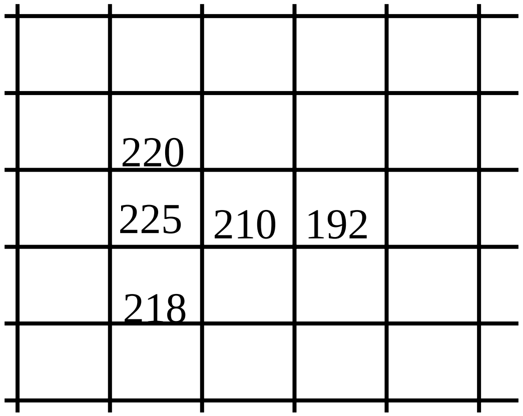
beehive

beehive[x_, y_] :={{x, y}, {x, y +1}, {x, y +2}, {x, y +3},{x, y+4}, {x, y+5}, {x, y+5}}



（diffusion）

Appearance in which molecules diffuse
Model of thermal conductance in substrate
r =0~255
r represents concentration and temperature.



```
Melt[r_, s_, t_] := Module[{init, ngbrsAve},
      init = Table[Random[Integer, {0, r}], {s}, {s}];
      ngbrsAve[mat_] := Floor[Apply[Plus,
                              Map[RotateRight[mat, #]&,
                                   {{-1, -1}, {-1, 0}, {-1, 1}, {0, -1},
                                     {0, 1}, {1, -1}, {1, 0}, {1, 1}}]]/8];
      NestList[ngbrsAve, init, t]]
```

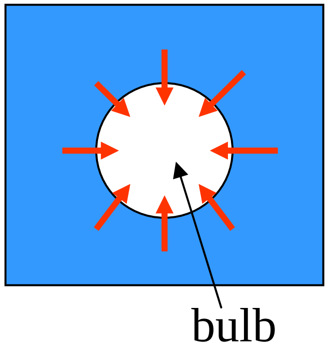
（boiling）

Modeling of phase transition from liquid to gas

- updated value of cite

$$\begin{cases} x < r \quad \Rightarrow \quad x \\ x \geq r \quad \Rightarrow \quad 0 \end{cases}$$

x = average value of 8 most neighborhood cites + 1

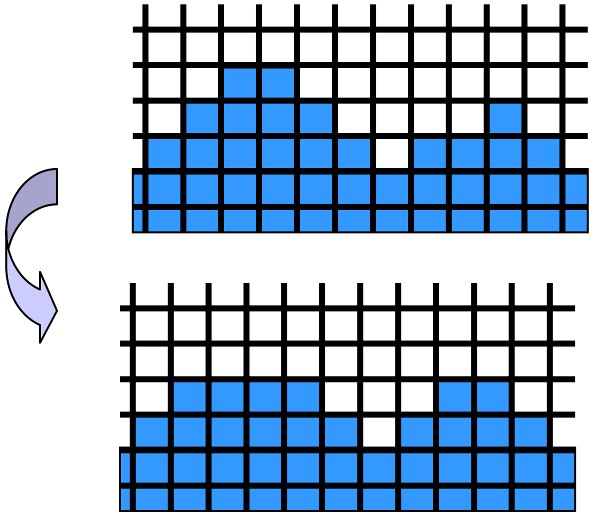


```
Rug[r_, s_, t_] := Module[{init, ngbrsAve},
      init = Table[Random[Integer, {0, r-1}], {s}, {s}];
      ngbrsAve[mat_] := Floor[Apply[Plus,
                              Map[RotateRight[mat, #]&,
                                   {{-1, -1}, {-1, 0}, {-1, 1}, {0, -1},
                                     {0, 1}, {1, -1}, {1, 0}, {1, 1}}]]/8];
      NestList[ngbrsAve[#] +1], r]&, init, t]]
```

（weathering）

- modeling of “smoothing” irregular end
- value of cite＝0 or 1

  updated value＝large number in 9 neighborhood cites



```
In[1]:= Print["                The CA Vote Rule Table"];
            TableForm[{Range[0, 9],
                {0, 0, 0, 0, 1, 0, 1, 1, 1, 1}},
               TableHeadings ->
                   {{"Some over neighborhood", "New cell value"},
                       None}]

                              CA Vote Rule Table
Sum over neighborhood   0  1  2  3  4  5  6  7  8  9
New cell value           0  0  0  0  1  0  1  1  1  1
```

```
VoteNearCallsToLosers[s_, t_] :=
   Module[{rule, init, ngbrhdTotal},
     init = Table[Random[Integer], {s}, {s}];
     ngbrhdTotal[mat_] := Apply[Plus, Map[RotateRight[mat, #]&,
                              {{-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1},
                                {0, 0}, {1, -1}, {1, 0}, {1, 1}}]];
     rule[5] := 0;
     rule[x_] := Floor[x/4];
     Attributes[rule] = Listable;
     NestList[rule[ngbrhdTotal[#]]&, init, t]]
```