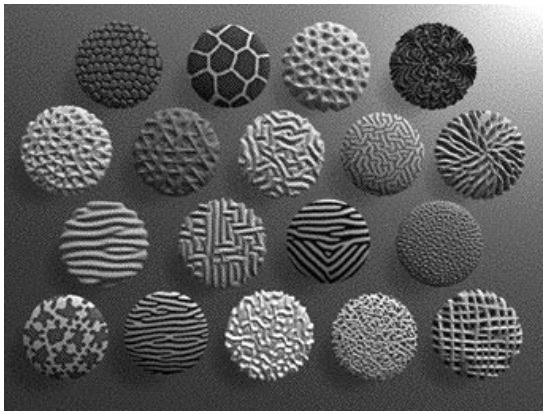
Lecture P4: Cellular Automata



Reaction diffusion textures.
Andy Witkin and Michael Kass

Array Review

Arrays allow manipulation of potentially huge amounts of data.

- All elements of the same type.
  - `double`, `int`
- N-element array has elements indexed 0 through N-1.
- Fast access to arbitrary element.
  - `a[i]`
- Waste of space if array is "sparse."

Cellular Automata

Cellular automata. (singular = cellular automaton)

- Computer simulations that try to emulate laws of nature.
- Simple rules can generate complex patterns.

John von Neumann. (Princeton IAS, 1950s)

- Wanted to create and simulate artificial life on a machine.
- Self-replication.
- "As simple as possible, but no simpler."

Applications of Cellular Automata

Modern applications.

- Simulations of biology, chemistry, physics.
  - ferromagnetism according to Ising mode
  - forest fire propagation
  - nonlinear chemical reaction-diffusion systems
  - turbulent flow
  - biological pigmentation patterns
  - breaking of materials
  - growth of crystals
  - growth of plants and animals
- Image processing.
- Computer graphics.
- Design of massively parallel hardware.
- Art.

How Did the Zebra Get Its Stripes?

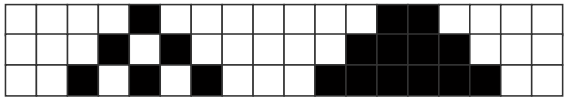
Synthetic zebra.
Greg Turk

One Dimensional Cellular Automata

1-D cellular automata.

- Sequence of cells.
- Each cell is either black (alive) or white (dead).
- In each time step, update status of each cell, depending on color of nearby cells from previous time step.

Example rule. Make cell black at time t if at least one of its proper neighbors was black at time t-1.



Cellular Automata: Designing the Code

How to store the row of cells.

- An array.

How to store the history of cells.

- A multidimensional array.
  - wastes a lot of space
- An array that stores only previous time step.
  - wastes a little time updating history array

How to output results.

- Turtle graphics.

Cellular Automata: The Code

cellular.c (part 1)

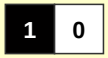
```
#include <stdio.h>
#define STEPS 128           // # of iterations to simulate
#define CELLS 256           // # of cells
#define PS 512.0            // size of canvas

int main(void) {
    int i, t;
    int cells[CELLS] = {0}; // cell contents at time t
    int old[CELLS];         // cell contents at time t-1

    cells[CELLS/2] = 1;     // init one cell to black

    // simulate cellular automata
    // INSERT CODE on next slide here

    return 0;
}
```



Cellular Automata: The Code

cellular.c (part 2)

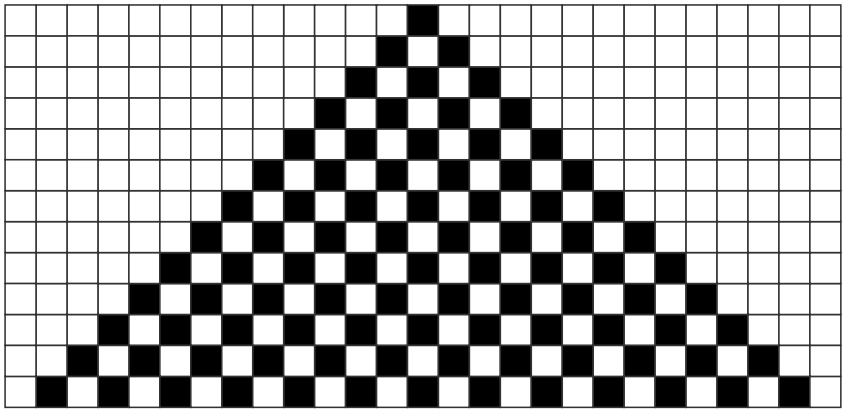
```
// simulate cellular automata
for (t = 1; t < STEPS; t++) {
    // output current row in turtle graphics
    for (i = 0; i < CELLS; i++) {
        if(cells[i] == 1) {
            printf("F %f %f\n", PS*i/N, PS - PS*j/CELLS);
            printf("S %f\n", PS / CELLS);
        }
    }

    // copy old values
    for (i = 0; i < CELLS; i++) old[i] = cells[i];

    // update new cells according to rule 250
    for (i = 1; i < CELLS - 1; i++) {
        if (old[i-1] + old[i+1] > 0) cells[i] = 1;
        else cells[i] = 0;
    }
}
```


Cellular Automata Rules

Rule 250 (repetition).



Cellular Automata: Change the Rules

What happens if we change the update rule?

New rule. Make cell black at time t if exactly one proper neighbor was black at time t-1.

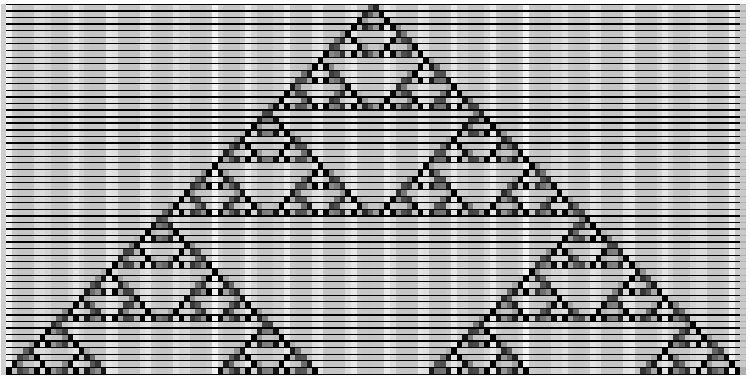
Replace update rule in cellular.c

```
// update new cells according to rule 90
for (i = 1; i < CELLS - 1; i++) {
    if (old[i-1] + old[i+1] == 1) cells[i] = 1;
    else cells[i] = 0;
}
```

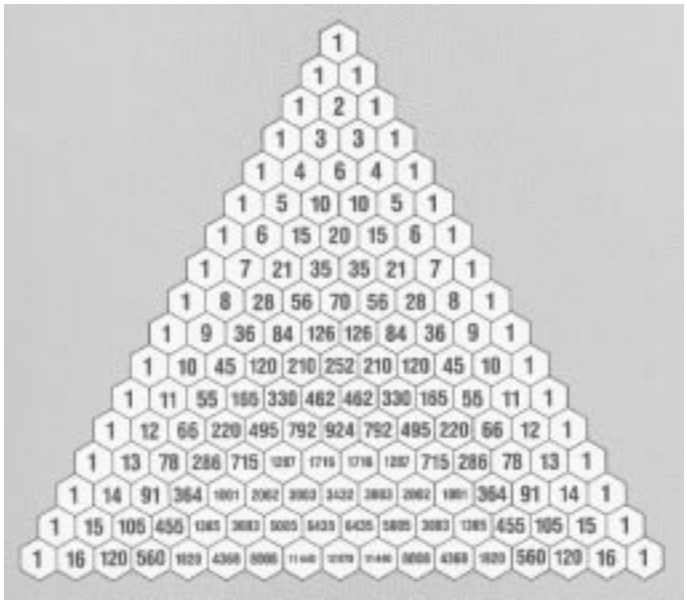
Original rule: > 1

Cellular Automata: Change the Rules

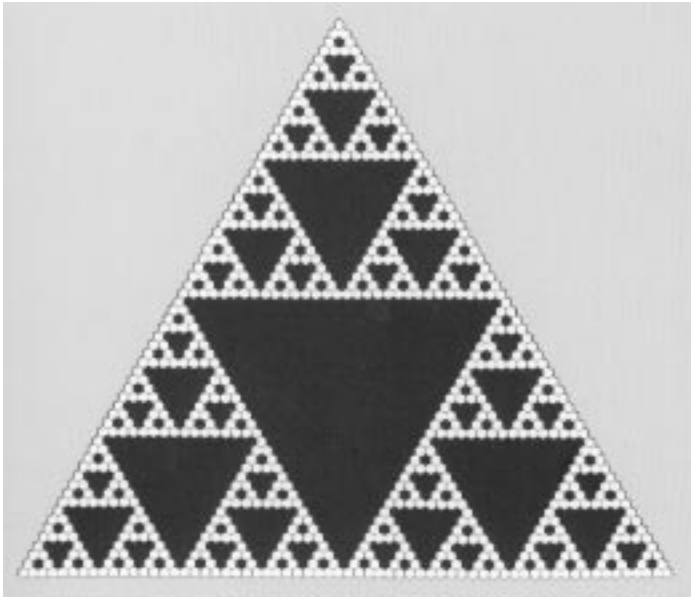
Rule 90 (nesting).



Pascal's Triangle



Pascal's Triangle



Binary Numbers

Binary and decimal representation of integers.

- Binary is base 2.
- Decimal is base 10.

Dec	Bin	Dec	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

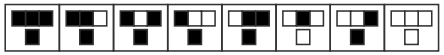
Example.

- $250_{10} = 11111010_2$.

7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	0
2^7	$+2^6$	$+2^5$	$+2^4$	$+2^3$	0	$+2^1$	0
$250_{10} =$ 128	+64	+32	+16	+8	0	+2	0

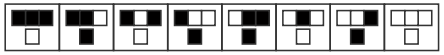
Cellular Automata Rules

Rule 250 (repetition).



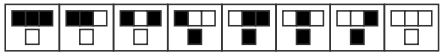
$11111010_2 = 250_{10}$

Rule 90 (nesting).



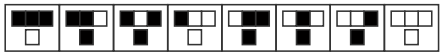
$01011010_2 = 90_{10}$

Rule 30 (randomness).



$00011110_2 = 30_{10}$

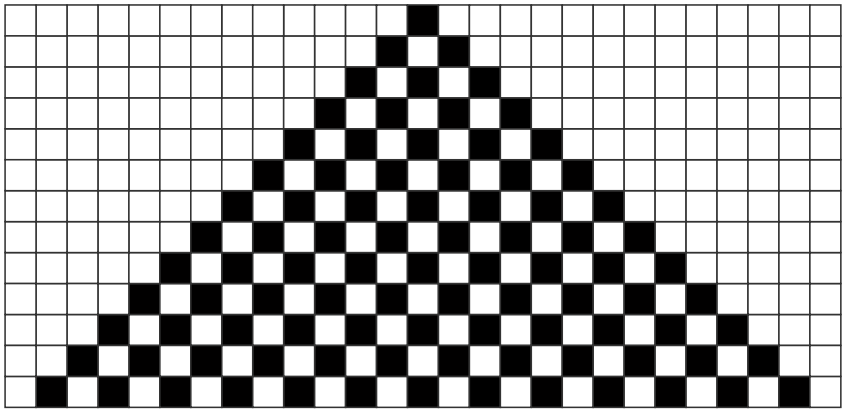
Rule 110 (localized structure).



$01101110_2 = 110_{10}$

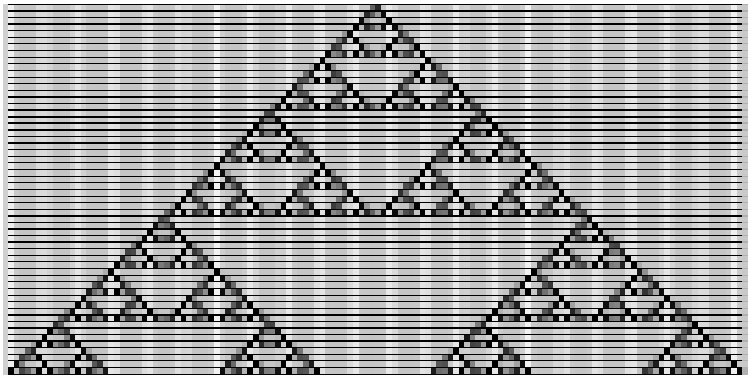
Cellular Automata Rules

Rule 250 (repetition).



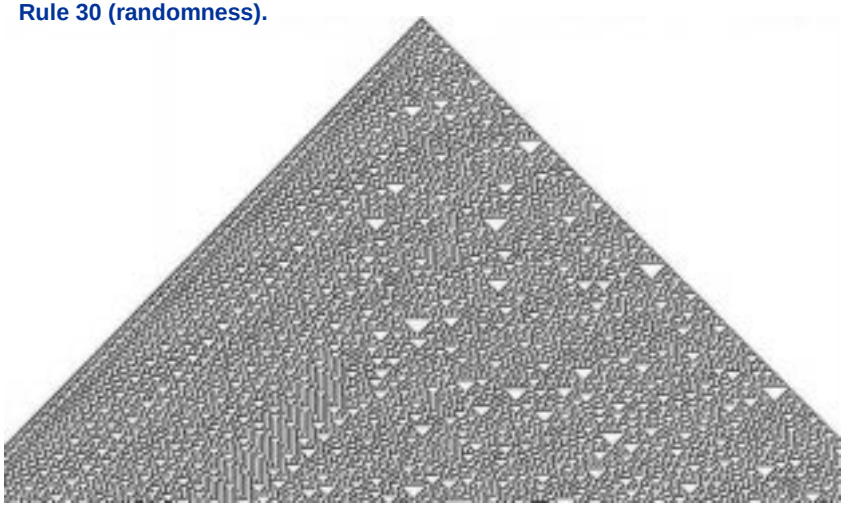
Cellular Automata Rules

Rule 90 (nesting).



Cellular Automata Rules

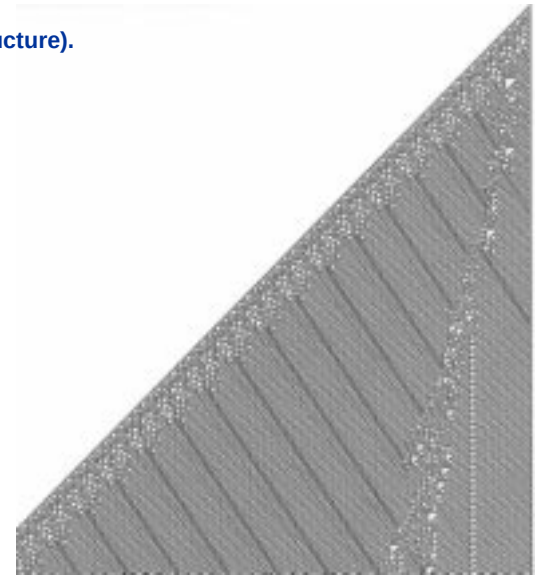
Rule 30 (randomness).



Source: *A New Kind of Science* by Steve Wolfram.

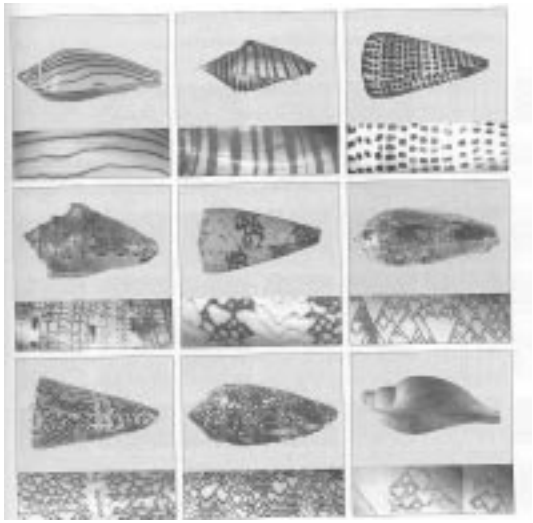
Cellular Automata Rules

Rule 110 (localized structure).



Source: *A New Kind of Science* by Steve Wolfram.

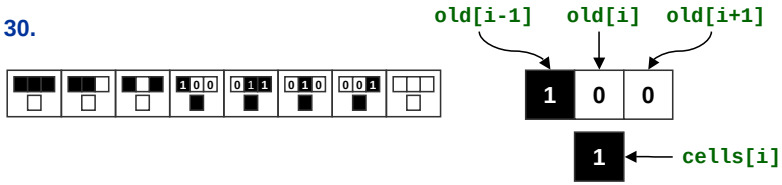
Pigmentation Patterns on Mollusk Shells



Source: *A New Kind of Science* by Steve Wolfram.

Cellular Automata: Designing the Code

Rule 30.



possible code for rule 30

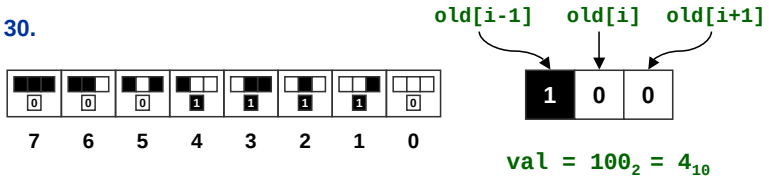
```
// update new cells according to rule 30
for (i = 1; i < CELLS - 1; i++) {
    if ((old[i-1] == 1 && old[i] == 0 && old[i+1] == 0) ||
        (old[i-1] == 0 && old[i] == 1 && old[i+1] == 1) ||
        (old[i-1] == 0 && old[i] == 1 && old[i+1] == 0) ||
        (old[i-1] == 0 && old[i] == 0 && old[i+1] == 1))

        cells[i] = 1;

    else cells[i] = 0;
}
```


Cellular Automata: Designing the Code

Rule 30.



$val = 100_2 = 4_{10}$

possible code for rule 30

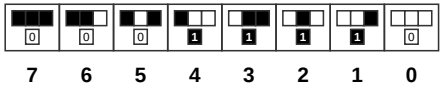
```
// rule 30
int rule[8] = {0, 1, 1, 1, 1, 0, 0, 0};

. . .

// update new cells according to rule 30
for (i = 1; i < CELLS - 1; i++) {
    val = 4*old[i-1] + 2* old[i] + old[i+1];
    cells[i] = rule[val];
}
```


Cellular Automata: Designing the Code

Rule 30.



Ex. $rule = 30_{10} = 00011110_2$

$val = 2_{10} = 010_2$

$00011110_2 >> 2_{10} = 000111_2$

$$\begin{array}{r} 000111_2 \\ \& \ \underline{000001_2} \\ 000001_2 \end{array}$$

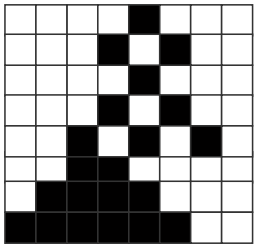
code for arbitrary rule

```
#define RULE 30
. . .
// update new cells according to arbitrary rule
for (i = 1; i < CELLS - 1; i++) {
    val = 4*old[i-1] + 2* old[i] + old[i+1];
    cells[i] = (RULE >> val) & 1;
}
```

Two Dimensional Cellular Automata

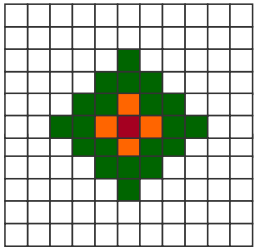
2-D cellular automata.

- N x N grid of cells.
- Each cell is either black (1) or white (0).
- Update status of each cell, depending on neighbors.
- Repeat.



Example.

- Update cell (i, j) by considering all cells within Manhattan distance 3 of (i, j).
- Color cell (i, j) black if following SUM is greater than 1.4:
  - add together twice the color values of each cell that is distance 0 or 1 away
  - subtract 120% of color values of cells that are distance 2 or 3 away



Two Dimensional Cellular Automata

2-d cellular automata

```
#include <stdio.h>
#define N 512
#define STEPS 10
#define GRIDSIZE 512.0
#define MAXDIST 3

int randomInteger(int n) { . . . }
int cells[N][N];   // two dimensional array

int main(void) {
    int i, j, k, ii, jj, dist, color;
    double sum, threshhold = 1.4;
    double weight[MAXDIST + 1] = {2.0, 2.0, -1.2, -1.2};

    // initialize with random pattern
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            cells[i][j] = randomInteger(2);
```


Two Dimensional Cellular Automata

2-d cellular automata (brute force)

```
for (k = 0; k < STEPS; k++) {
   for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++)  {

         // update cell (i, j)
         sum = 0.0;

         // consider only cells within distance 3
         for (ii = 0; ii < N; ii++) {
            for (jj = 0; jj < N; jj++) {
               dist = abs(ii-i) + abs(jj-j);
               color = cells[ii][jj];
               if (dist <= MAXDIST) sum += weight[dist] * color;
            }
         }
         if (sum > threshhold) cells[i][j] = 1;
         else cells[i][j] = 0;
      }
   }
}
```

absolute value

Why Are You So Slow?

Why is the program so slow?

- 221 seconds for N = 128.
  - $10N^4 > 2.7$ billion
- 15.7 hours for N = 512.
  - $10N^4 > 687$ billion.

A quintuply nested loop

```
for (a = 0; a < 10; a++) {
   for (b = 0; b < N; b++) {
      for (c = 0; c < N; c++)  {
         for (d = 0; d < N; d++) {
            for (e = 0; e < N; e++) {
               // code here in innermost loop is
               // executed 10N^4 times.
            }
         }
      }
   }
}
```

Two Dimensional Cellular Automata

2-d cellular automata

```
for (k = 0; k < STEPS; k++) {
   for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++)  {

         // update cell (i, j)
         sum = 0.0;

         // consider only cells within distance 3
         for (ii = i - MAXDIST; ii <= i + MAXDIST; ii++) {
            for (jj = j - MAXDIST; jj <= j + MAXDIST; jj++) {
               dist = abs(ii-i) + abs(jj-j);
               color = cells[ii % N][jj % N];
               if (dist <= MAXDIST) sum += weight[dist] * color;
            }
         }
         if (sum > 0) cells[i][j] = 1;
         else cells[i][j] = -1;
      }
   }
}
```


Algorithmic Speedup

Original program. (code in innermost loop executed $10 N^4$ times)

- 221 seconds for N = 128.
  - $10N^4 > 2.7$ billion
- 15.7 hours for N = 512.
  - $10N^4 > 687$ billion.

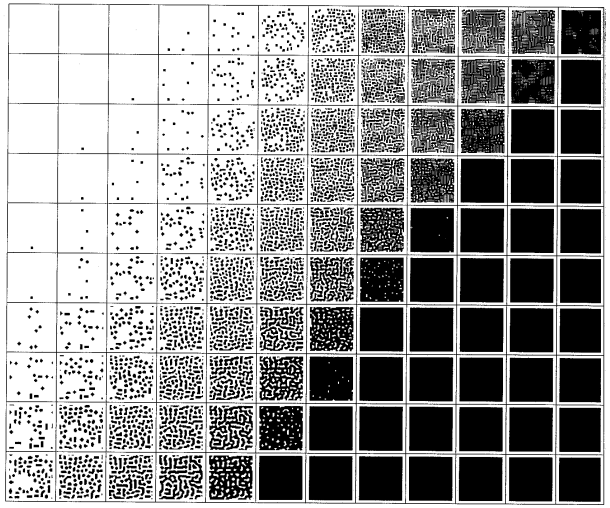
Improved program. (code in innermost loop executed $490 N^2$ times)

- 1.1 seconds for N = 128.
  - 200x speedup
- 17.5 seconds for N = 512.
  - 3000x speedup

Stay tuned: analysis of algorithms lecture.

Two Dimensional Cellular Automata

Tweaking the parameters.



Source: *A New Kind of Science* by Steve Wolfram.

Source: *A New Kind of Science* by Steve Wolfram.

Conway's Game of Life

Conway's Game of Life.

- Based on theories of von Neumann.
- 2-D cellular automaton to simulate synthetic universe.
- Can also simulate general purpose computer.

Critters live and die in depending on 8 neighboring cells:

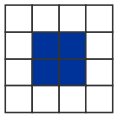
- too few? (0 or 1)
  - die of loneliness
- too many? (4-8)
  - die of overcrowding
- just right? (2 or 3)
  - survive to next generation
- exactly 3 parents?
  - critter *born* in empty square

1	2	3
8		4
7	6	5

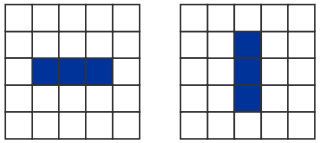
John Conway

Conway's Game of Life

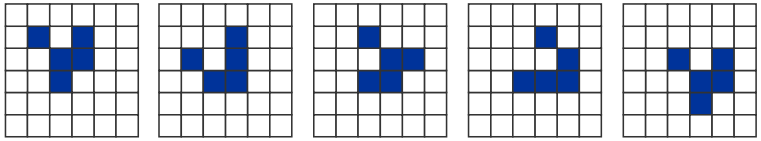
Stable.



Oscillator.



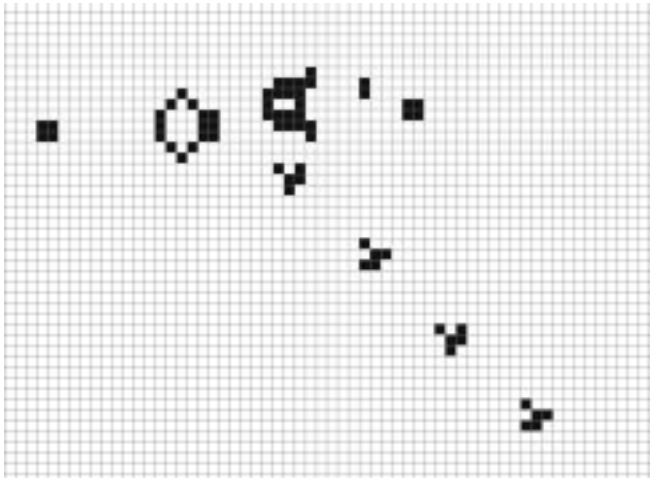
Glider.



Conway's Game of Life

Glider gun.

- Produces a new glider every 30 iterations.



Conway's Game of Life

Glider gun synthesizer.

- 8 gliders collide to form a glider gun.