AEV2 testeo de “Game of life”

2º DAM – Sostenibilidad aplicada al sistema productivo

Game of life

Photo by Kristin Morgan: https://www.pexels.com/photo/close-up-of-a-unique-snowflake-on-gradient-background-34194627/

Regla 30 (Stephen Wolfram, 1983): un autómata unidimensional que produce patrones caóticos.
Juego de la Vida (John Conway, 1970): un autómata bidimensional con reglas que generan patrones estables, osciladores y estructuras móviles.
Modelos de incendios, epidemias, tráfico, crecimiento de cristales → se modelan fácilmente con autómatas celulares.

Game of life

El Juego de la Vida fue creado en **1970** por el matemático británico **John Horton Conway**.

Se trata de un **autómata celular bidimensional**, un modelo matemático que simula cómo evoluciona un sistema de celdas según reglas sencillas.

Cada celda puede estar en dos estados: **viva** o **muerta**, y en cada iteración su estado depende de los vecinos.

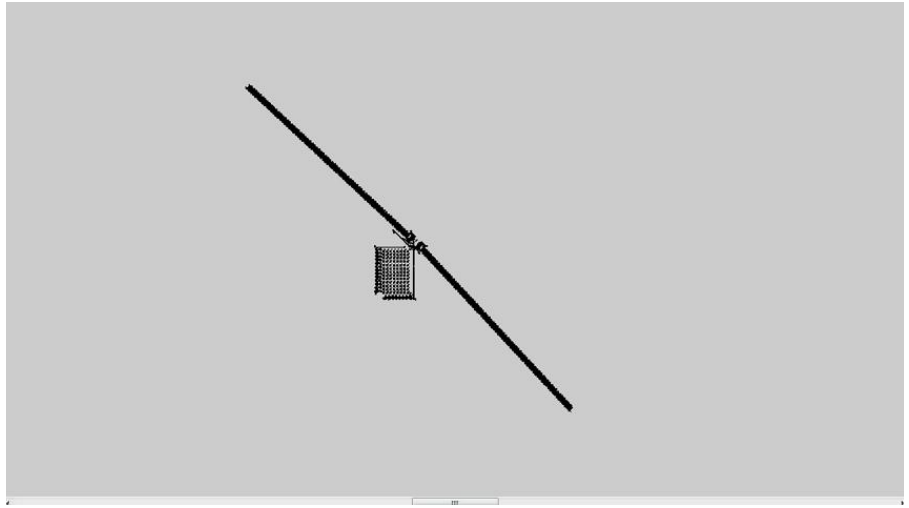
Conway diseñó las reglas con la idea de encontrar un sistema **capaz de generar comportamiento complejo a partir de condiciones simples**, y lo logró: del caos inicial surgen patrones estables, osciladores y estructuras que parecen "moverse" como las naves (*gliders*, *spaceships*).

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

El Juego de la Vida se hizo famoso cuando fue publicado en la revista **Scientific American** por Martin Gardner en octubre de 1970, en su columna "Mathematical Games".

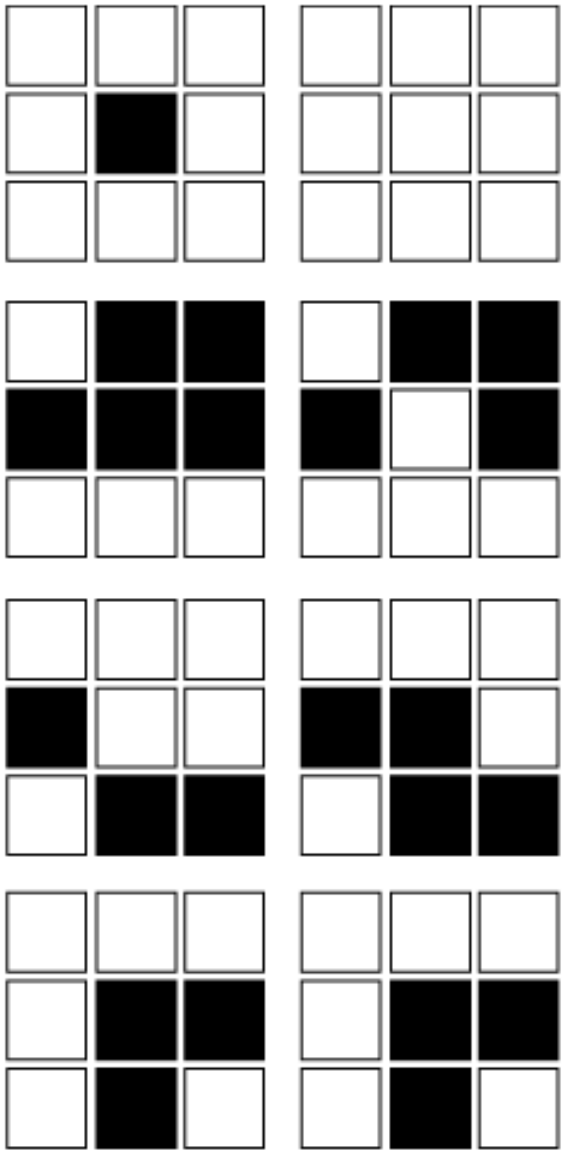
Desde entonces, se convirtió en un **clásico de la teoría de autómatas y la computación**, mostrando cómo la complejidad puede emerger de reglas muy básicas.

Hoy en día es estudiado en **matemáticas, informática, biología y filosofía de la ciencia**, y es considerado un ejemplo de **sistema complejo emergente** y **máquina de Turing universal** (puede realizar cualquier cálculo).



https://www.youtube.com/watch?v=My8AsV7bA94

Game of life



Loneliness
A cell with less than 2 adjoining cells dies.

Overcrowding
A cell with more than 3 adjoining cells dies.

Reproduction
An empty cell with more than 3 adjoining cells comes alive.

Stasis
A cell with exactly 2 adjoining cells remains the same.

https://prostheticknowledge.tumblr.com/post/404756220/rules-of-conways-game-of-life

Game of life

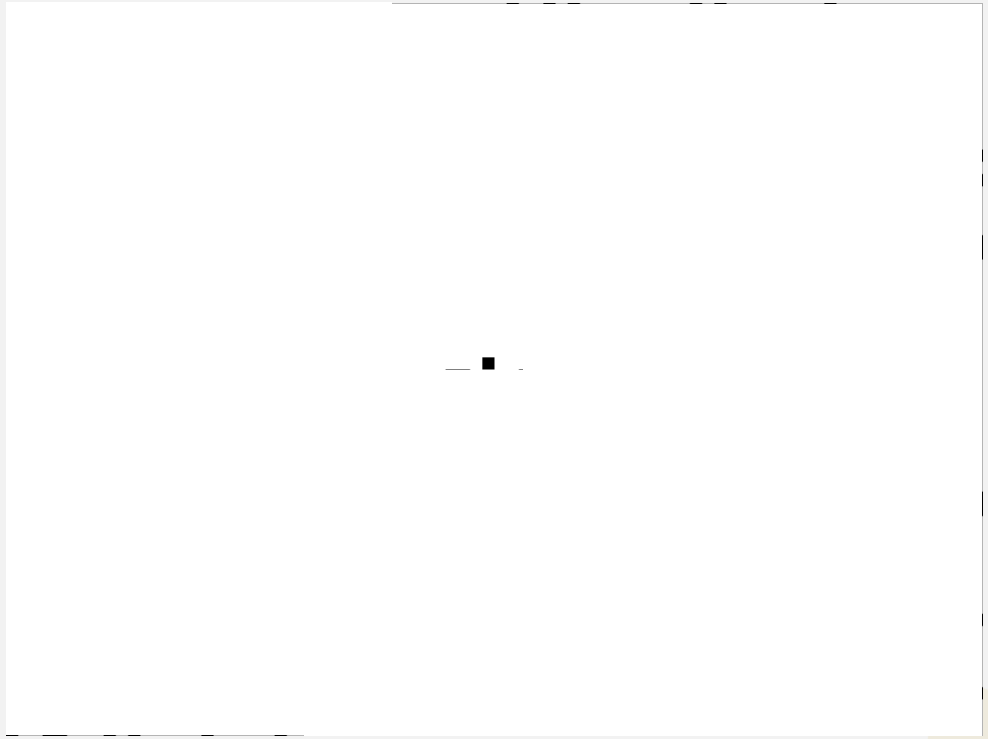
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Juego de la Vida</title>
    <style>
      canvas { border:1px solid; image-rendering: pixelated; }
      body { margin: 16px; font-family: system-ui, sans-serif; }
    </style>
  </head>
  <body>
    <h1>Juego de la Vida</h1>
    <canvas id="life" width="640" height="480"></canvas>
    <script src="js/main.js"></script>
  </body>
</html>
```


Canvas View

Setup en JavaScript

Accede al canvas y define constantes de tamaño. Prepara la cuadrícula vacía.

```
const canvas = document.getElementById('life');
const ctx = canvas.getContext('2d');

const CELL_SIZE = 8;
const COLS = Math.floor(canvas.width / CELL_SIZE);
const ROWS = Math.floor(canvas.height / CELL_SIZE);

// Crea una cuadrícula vacía (0 = muerta, 1 = viva)
function createGrid(rows, cols, fill=false) {
  const g = new Array(rows);
  for (let r = 0; r < rows; r++) {
    g[r] = new Array(cols).fill(fill ? 1 : 0);
  }
  return g;
}

let grid = createGrid(ROWS, COLS, false);
```

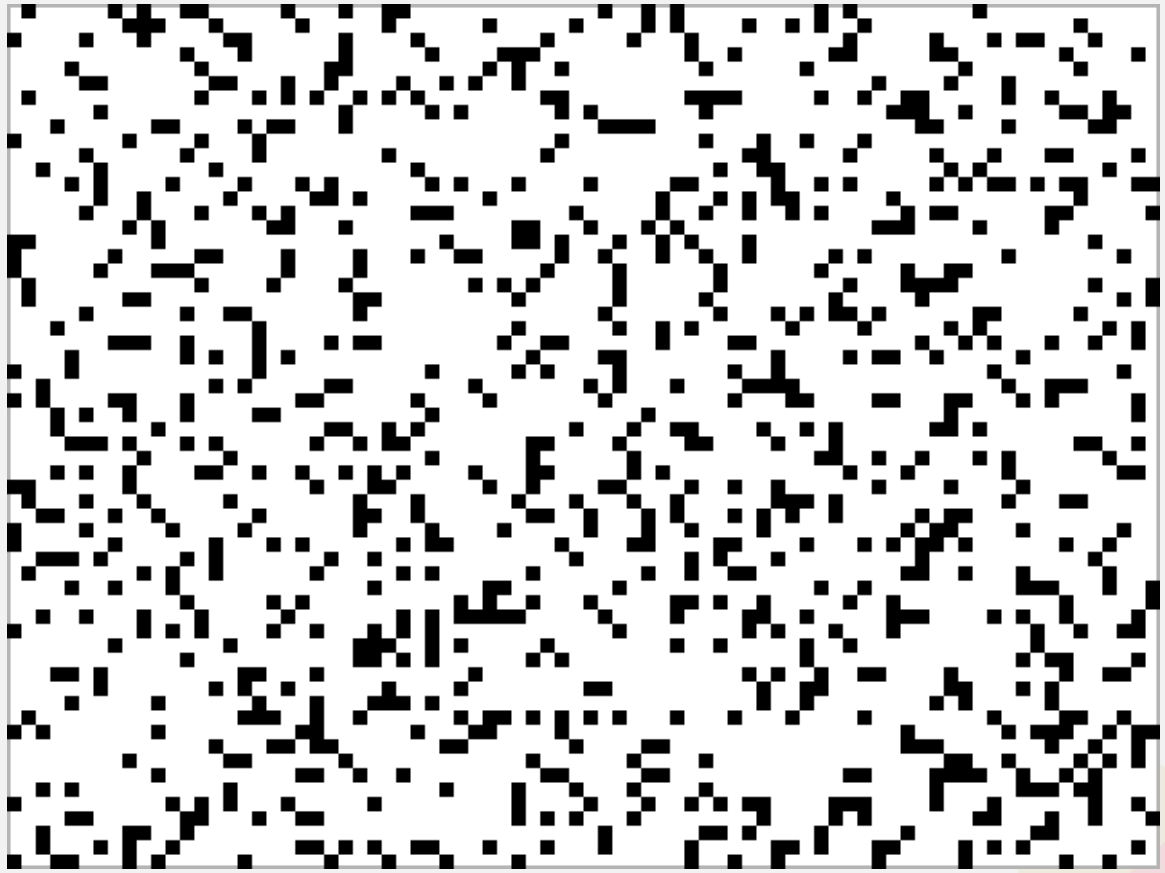


Canvas View

Estado inicial

Rellena la cuadrícula con celdas vivas al azar para ver algo en pantalla.

```
function randomize(p = 0.20) {
  for (let r = 0; r < ROWS; r++) {
    for (let c = 0; c < COLS; c++) {
      grid[r][c] = Math.random() < p ? 1 : 0;
    }
  }
}
randomize(0.20); // 20% vivas
```

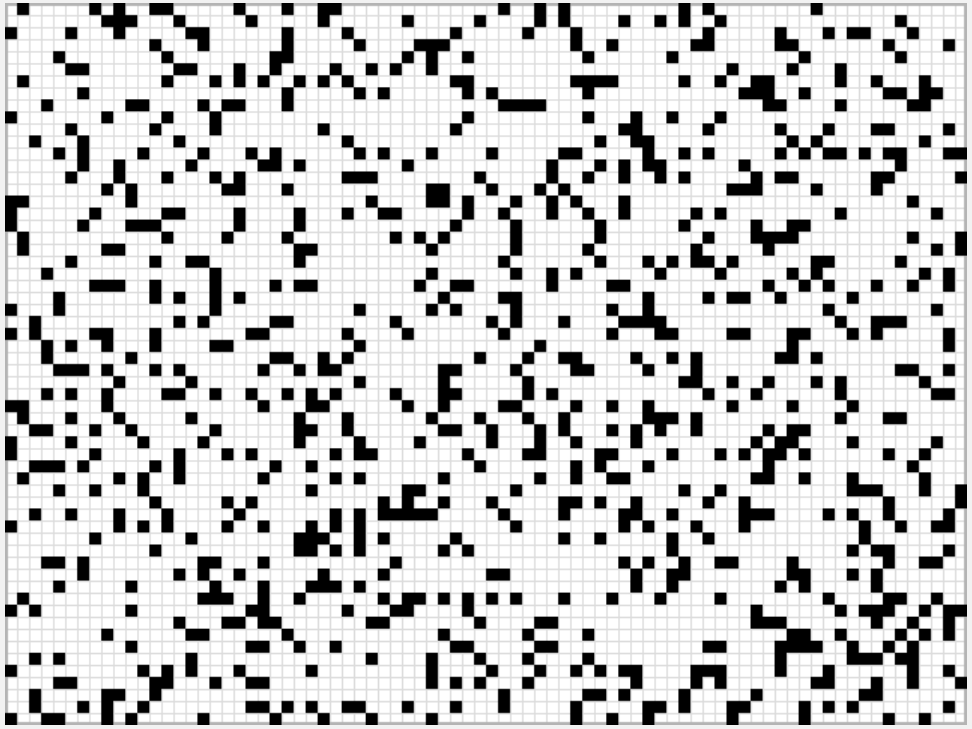


Canvas View

Dibujo

Dibuja cada celda viva como un rectángulo en el canvas. Opcional: líneas de rejilla.

```
function draw(showGrid = true) {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  // Celdas vivas
  for (let r = 0; r < ROWS; r++) {
    for (let c = 0; c < COLS; c++) {
      if (grid[r][c]) {
        ctx.fillRect(c * CELL_SIZE, r * CELL_SIZE, CELL_SIZE, CELL_SIZE);
      }
    }
  }
  // Rejilla opcional
  if (showGrid) {
    ctx.beginPath();
    for (let x = 0; x <= COLS; x++) {
      ctx.moveTo(x * CELL_SIZE, 0);
      ctx.lineTo(x * CELL_SIZE, ROWS * CELL_SIZE);
    }
    for (let y = 0; y <= ROWS; y++) {
      ctx.moveTo(0, y * CELL_SIZE);
      ctx.lineTo(COLS * CELL_SIZE, y * CELL_SIZE);
    }
    ctx.stroke();
  }
}
draw();
```

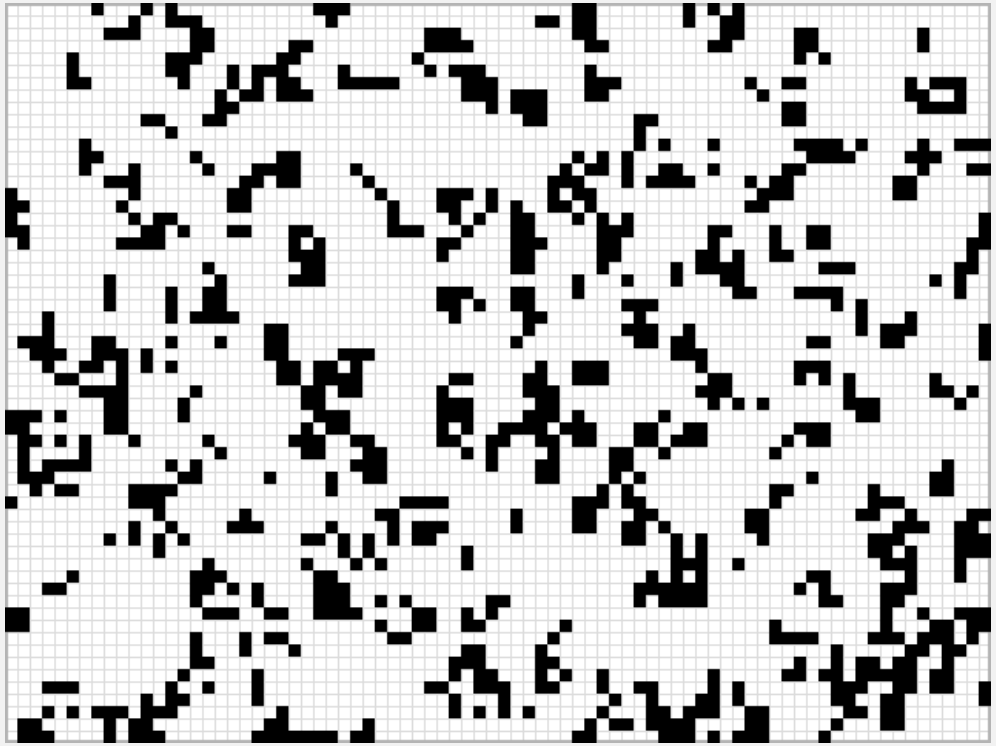


Canvas View

Reglas de Conway

Implementa el conteo de vecinos y la transición a la siguiente generación.

```
function neighbors(r, c) {
  let n = 0;
  for (let dr = -1; dr <= 1; dr++) {
    for (let dc = -1; dc <= 1; dc++) {
      if (dr === 0 && dc === 0) continue;
      const rr = (r + dr + ROWS) % ROWS;
      const cc = (c + dc + COLS) % COLS;
      n += grid[rr][cc];
    }
  }
  return n;
}

function step() {
  const next = createGrid(ROWS, COLS, false);
  for (let r = 0; r < ROWS; r++) {
    for (let c = 0; c < COLS; c++) {
      const alive = grid[r][c] === 1;
      const n = neighbors(r, c);
      next[r][c] = (alive && (n === 2 || n === 3)) || (!alive && n === 3)
? 1 : 0;
    }
  }
  grid = next;
  draw();
}

step(); // prueba una generación
```

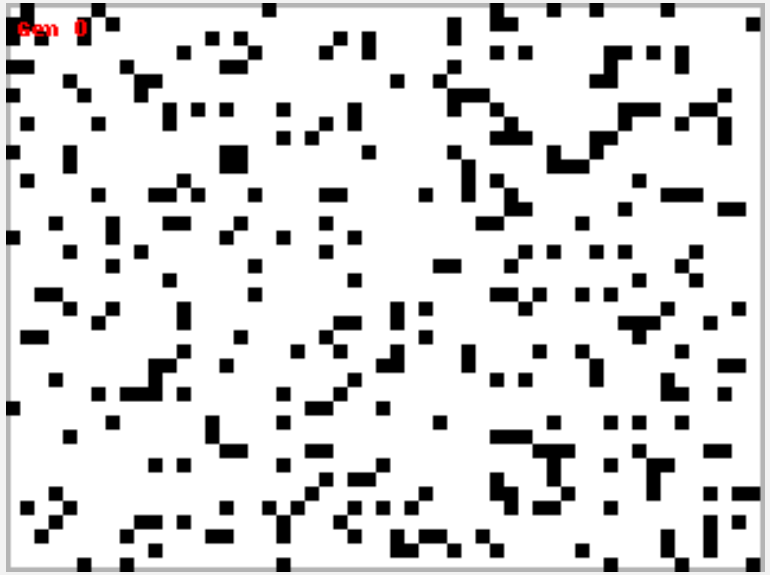


Canvas View

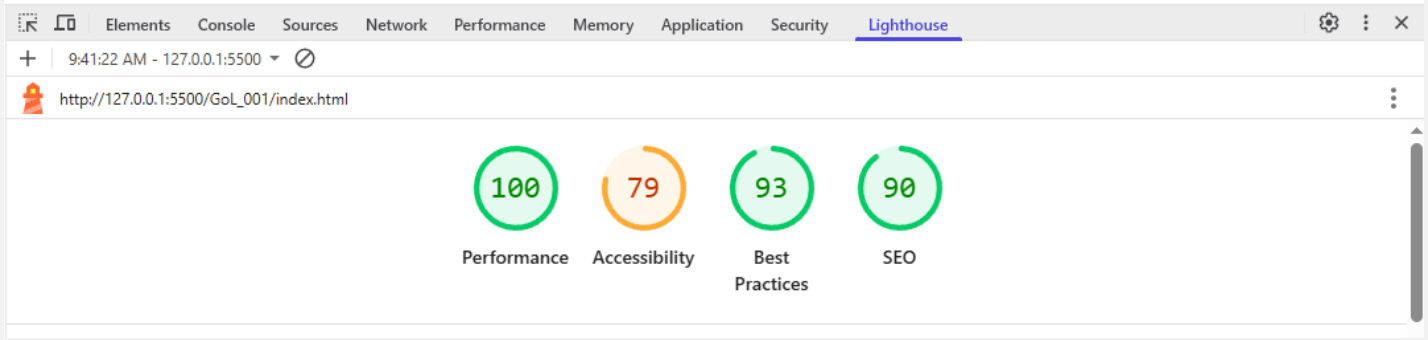
Bucle de animación

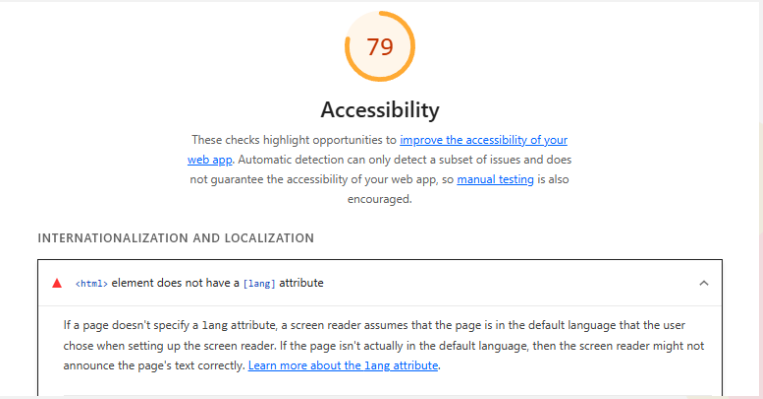
Anima el juego avanzando automáticamente fotograma a fotograma.

```
let running = true;

function loop() {
  if (running) {
    step();
  }
  requestAnimationFrame(loop);
}
loop();

// (Espacio) para pausar/reanudar
document.addEventListener('keydown', (e) => {
  if (e.key === ' ') { running = !running; e.preventDefault(); }
});
```

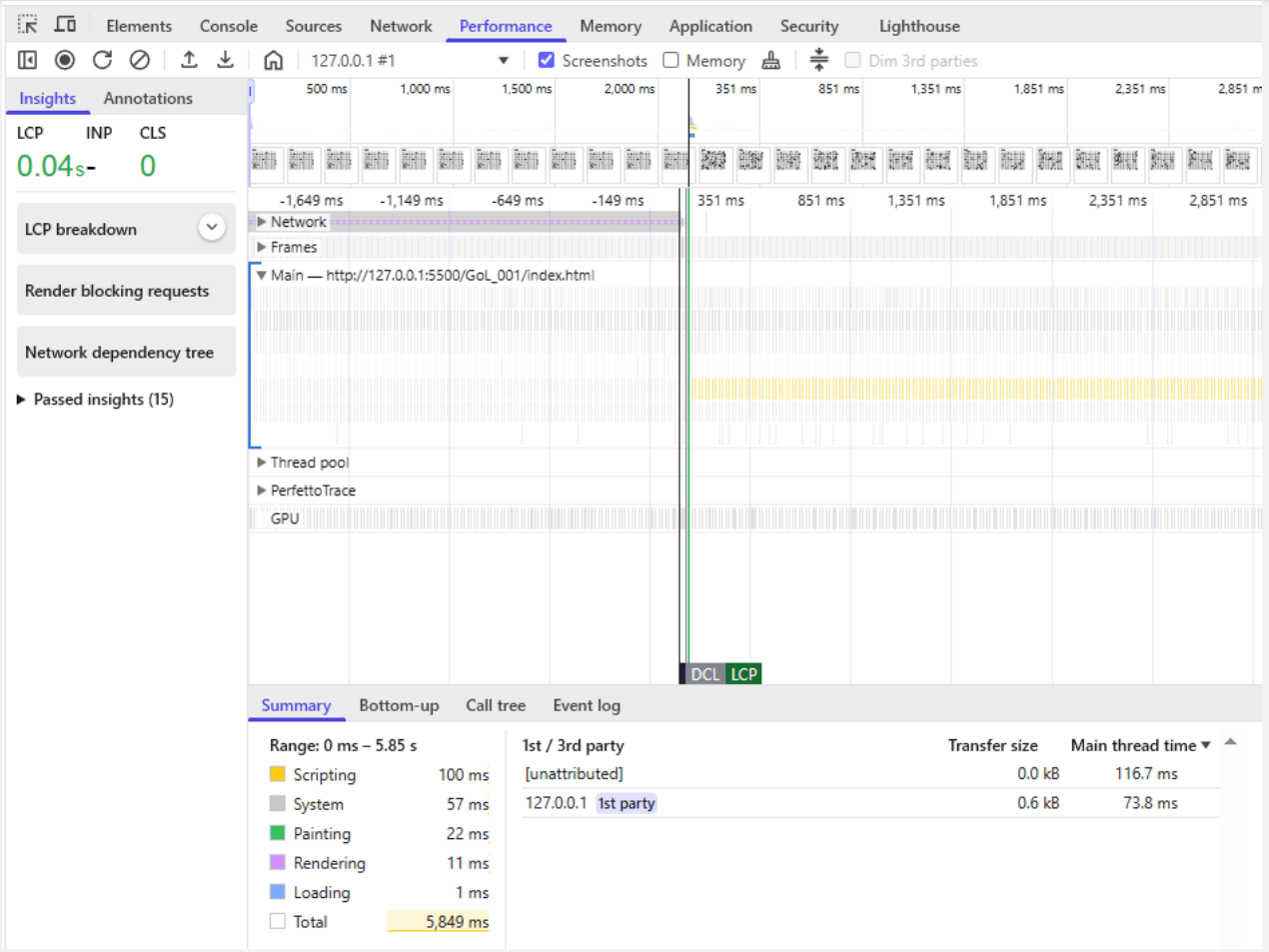


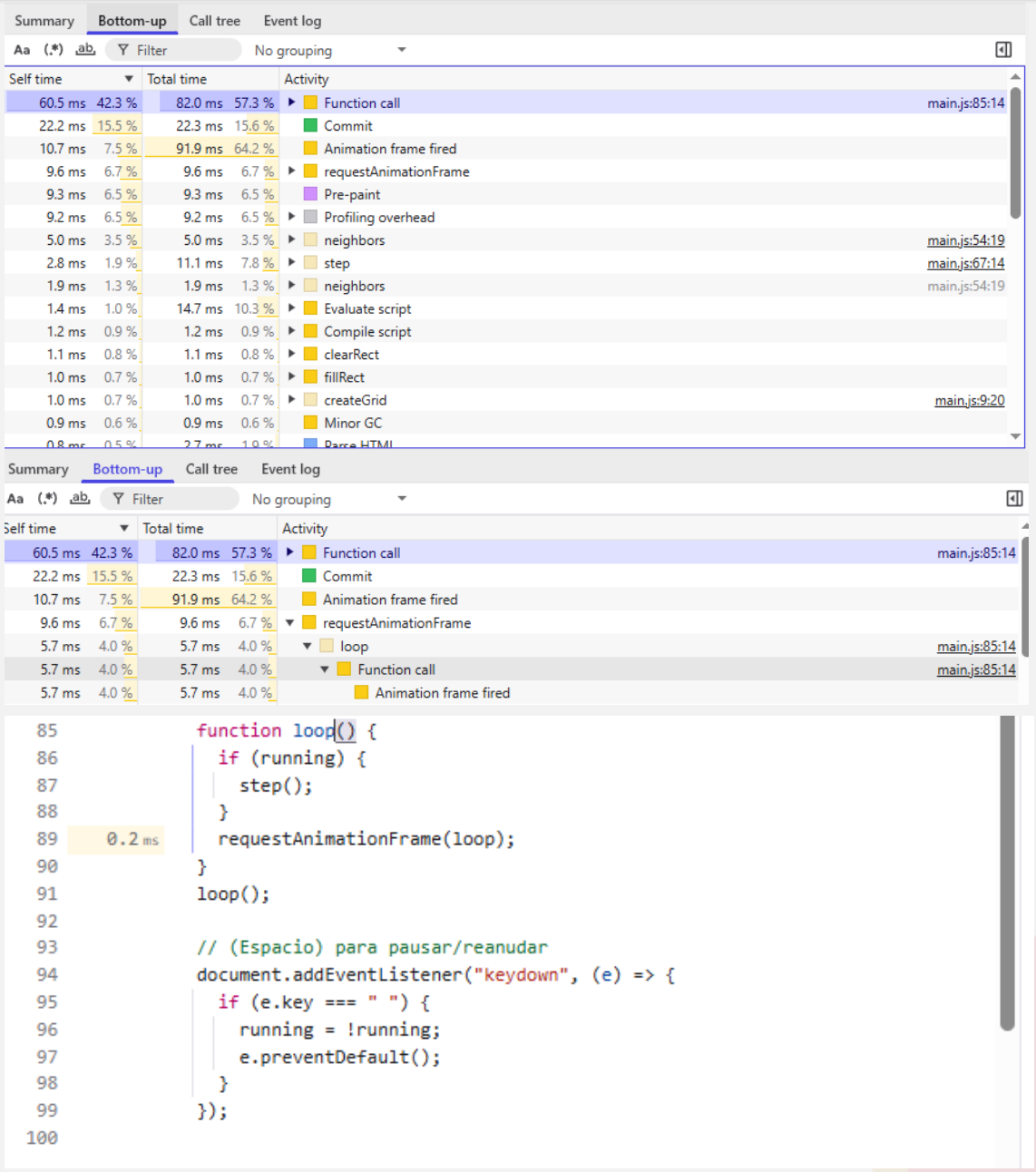
Informe del Lighthouse, proponiendo mejoras para visualización responsive





Game of life





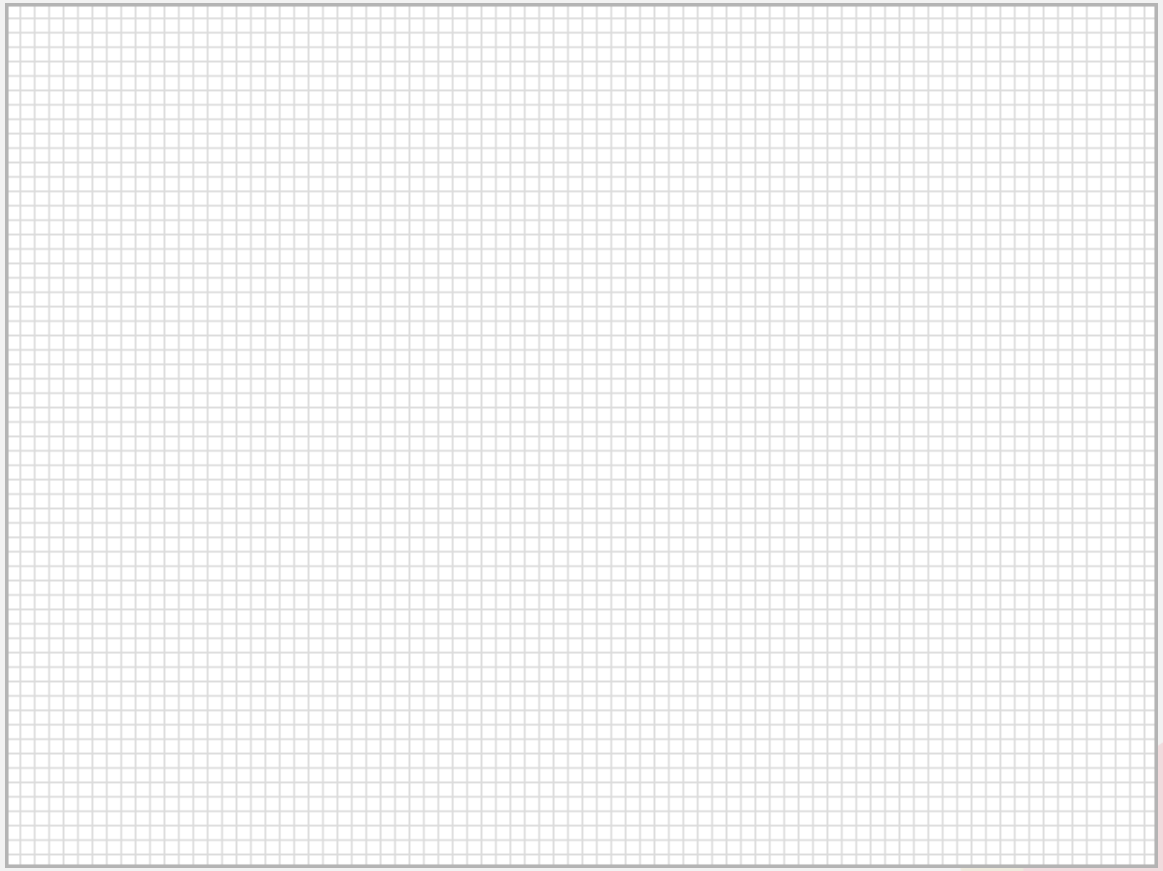

```
85          function loop() {
86            if (running) {
87              step();
88            }
89    0.2 ms    requestAnimationFrame(loop);
90          }
91          loop();
92
93          // (Espacio) para pausar/reanudar
94          document.addEventListener("keydown", (e) => {
95            if (e.key === " ") {
96              running = !running;
97              e.preventDefault();
98            }
99          });
100
```

La pestaña **Performance** en Google Chrome DevTools sirve para **analizar cómo se comporta tu página o aplicación web en tiempo real**: mide tiempos de carga, uso de CPU, memoria, renderizado, repaints, FPS, etc. Es la herramienta clave para encontrar cuellos de botella en rendimiento.

Controles extra (opcional)

Añade botones: limpiar, aleatorizar, variar la densidad o velocidad.

```
function clearAll() {
  grid = createGrid(ROWS, COLS, false);
  draw();
}

function randomFill(p = 0.20) {
  randomize(p);
  draw();
}

// Ejemplo HTML adicional:
// <button onclick="clearAll()">Limpiar</button>
// <button onclick="randomFill(0.20)">Aleatorio</button>
```

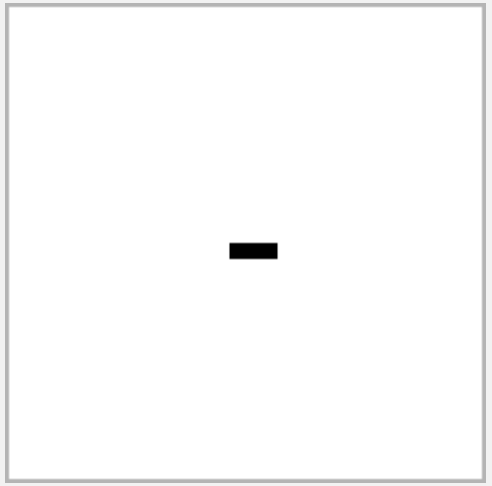


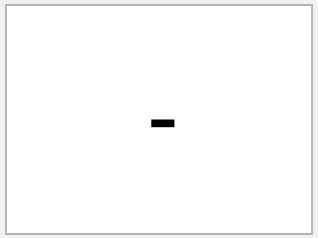
EJEMPLOS

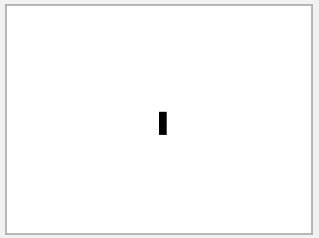
Ejemplo: Oscillator (Blinker)

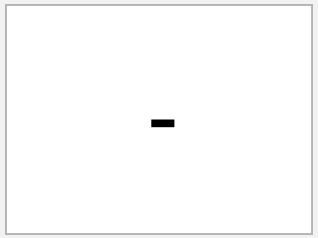
El 'blinker' es un oscilador: tres celdas que alternan entre horizontal y vertical cada generación.

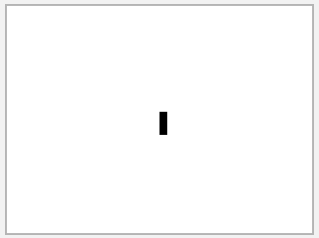
```
// Insertar un blinker (oscilador) en el centro
function setBlinker(r, c) {
  grid[r][c-1] = 1;
  grid[r][c]   = 1;
  grid[r][c+1] = 1;
}
setBlinker(Math.floor(ROWS/2), Math.floor(COLS/2));
draw();
```







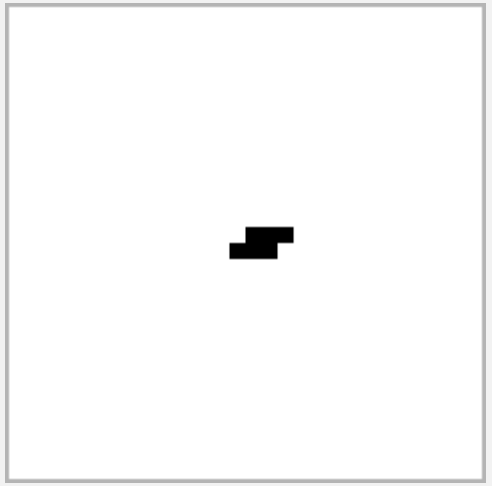


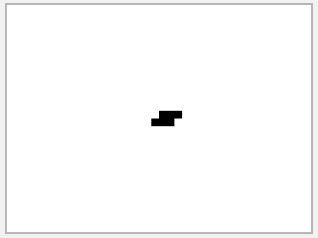


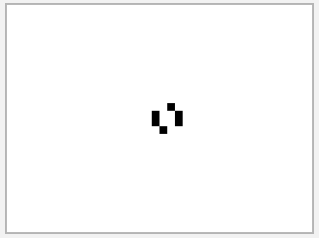
Ejemplo: Evolución del Oscillator: Toad

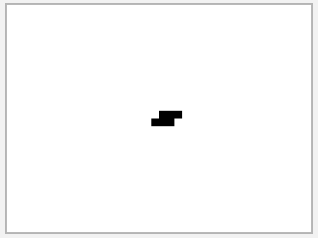
El 'toad' es un oscilador de periodo 2 formado por 6 celdas.

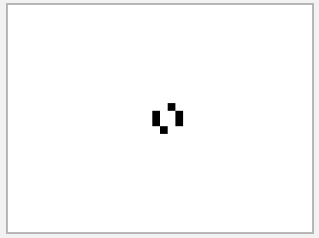
```
// Oscillator: Toad (6 celdas)
function setToad(r, c) {
  grid[r][c-1] = 1;
  grid[r][c]   = 1;
  grid[r][c+1] = 1;
  grid[r-1][c]   = 1;
  grid[r-1][c+1] = 1;
  grid[r-1][c+2] = 1;
}
```







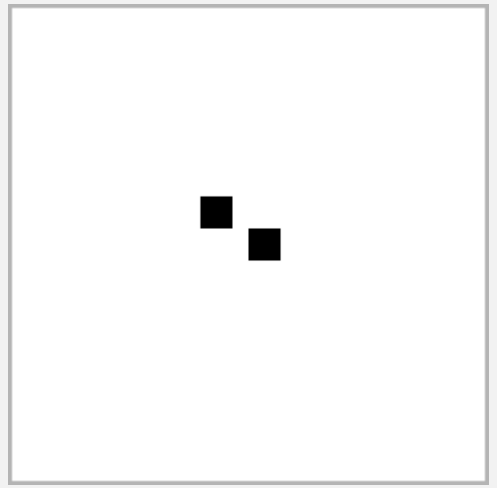


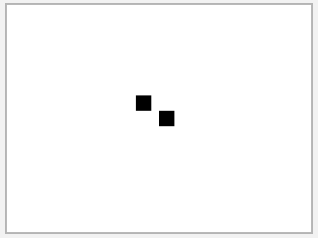


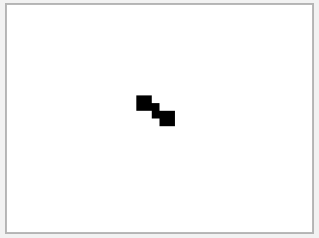
Ejemplo: Beacon

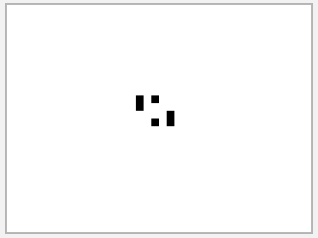
El 'beacon' es un oscilador de periodo 2, compuesto por dos bloques que alternan.

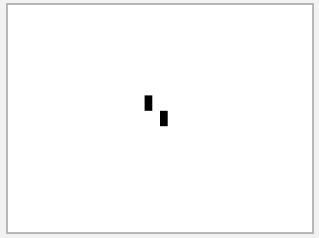
```
// Oscillator: Beacon (2 bloques que alternan)
function setBeacon(r, c) {
  // bloque superior izquierdo
  grid[r][c]     = 1;
  grid[r][c+1]   = 1;
  grid[r-1][c]   = 1;
  grid[r-1][c+1] = 1;

  // bloque inferior derecho
  grid[r-2][c-2] = 1;
  grid[r-2][c-3] = 1;
  grid[r-3][c-2] = 1;
  grid[r-3][c-3] = 1;
}
```







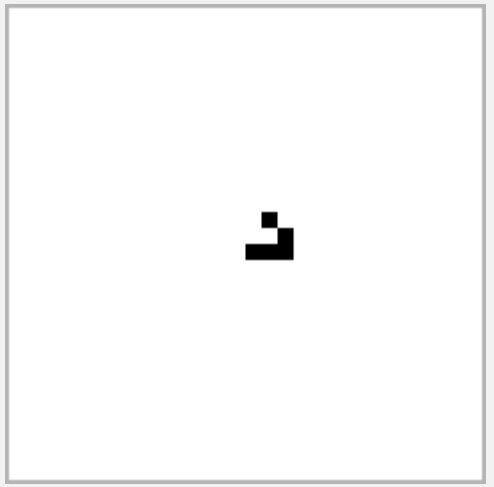


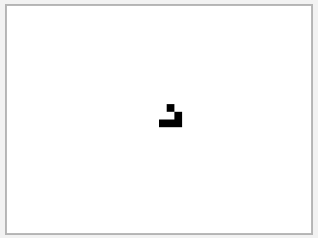


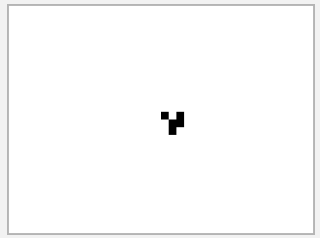
Ejemplo: Glider

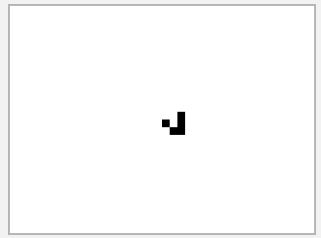
El glider se mueve en diagonal a través de la cuadrícula, repitiendo un ciclo de 4 pasos.

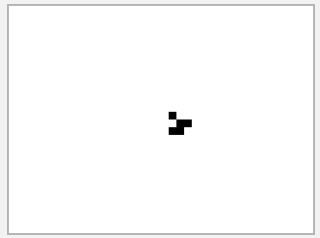
```
// Insertar un glider en el centro
function setGlider(r, c) {
  grid[r][c] = 1;
  grid[r][c+1] = 1;
  grid[r][c+2] = 1;
  grid[r-1][c+2] = 1;
  grid[r-2][c+1] = 1;
}
setGlider(Math.floor(ROWS/2), Math.floor(COLS/2));
draw();
```

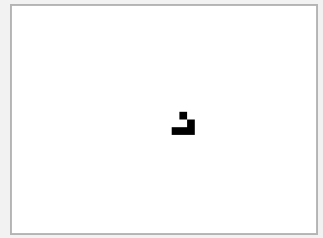




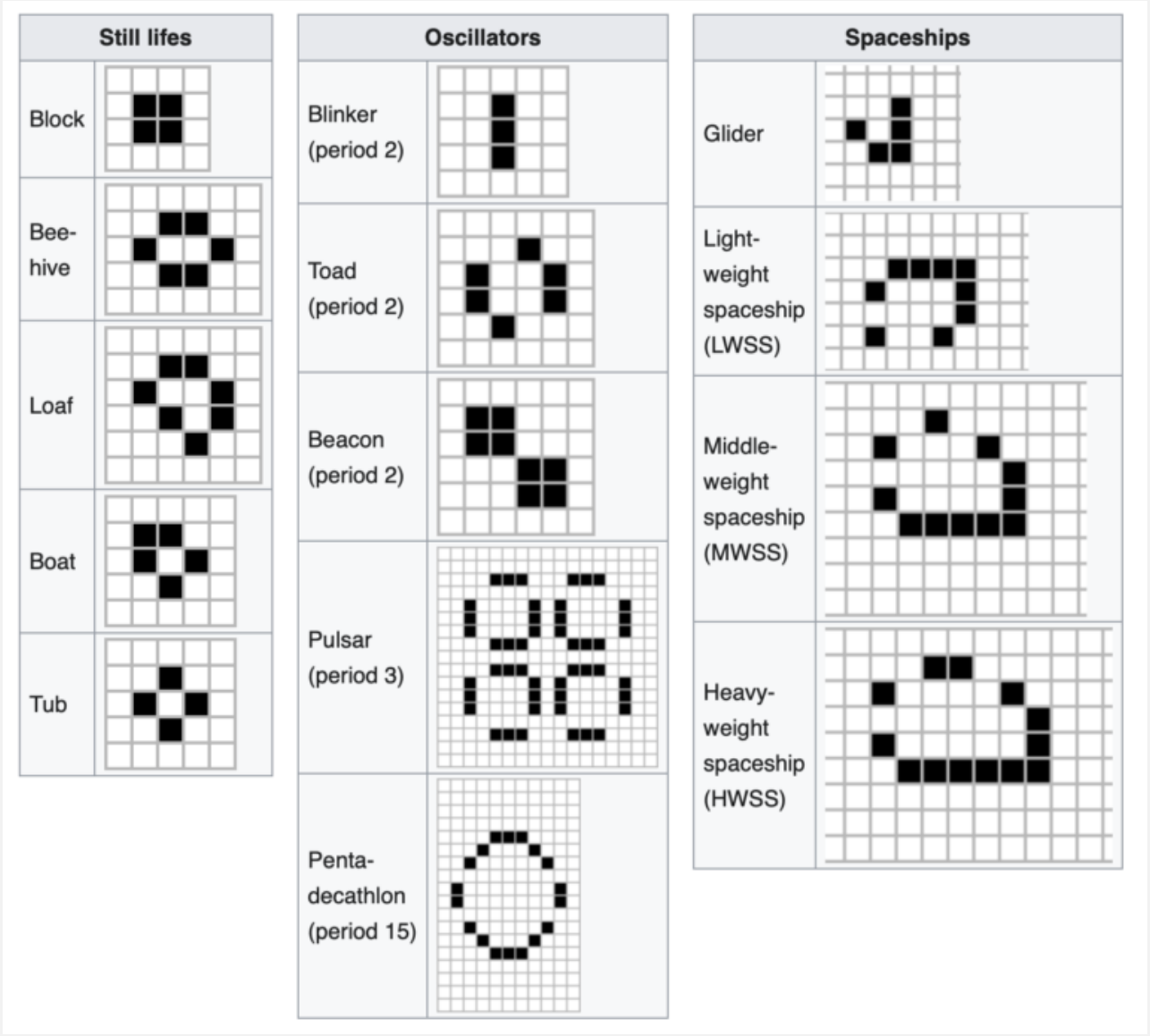








Ejemplos adicionales



https://blog.xojo.com/2022/05/11/conways-game-of-life/