Wolfram Regla 30

Luis Andrés Contla Mota

Junio 2025

Índice

1 Introduction

1.1 ¿Qué es la Regla 30 de Wolfram?

La Regla 30 es un autómata celular elemental unidimensional propuesto por Stephen Wolfram en la década de 1980. Consiste en una fila de celdas que pueden estar en uno de dos estados posibles: activas (1) o inactivas (0). En cada generación, el estado de cada celda se actualiza simultáneamente según una regla fija que depende del estado actual de la celda y de sus dos vecinas inmediatas (izquierda y derecha). La Regla 30, en particular, se define por una tabla de transición específica que, a partir de patrones simples, genera comportamientos complejos y aparentemente aleatorios. Este autómata es famoso por mostrar cómo reglas simples pueden producir patrones caóticos y ha sido ampliamente estudiado en el campo de la computación y la teoría de sistemas complejos.

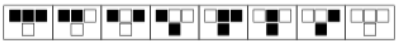


Figure 1: Regla 30

La evolución de cada celda en la siguiente generación depende de su estado actual y de los estados de sus dos vecinas inmediatas (izquierda y derecha), siguiendo una regla fija. Las condiciones de actualización para cada celda son las siguientes. Se observa el estado de la celda izquierda (L), la celda actual (C) y la celda derecha (R). Según la combinación de estos tres valores, la celda en la siguiente generación será 0 o 1, de acuerdo a la Figura 1. Si el patrón es 111, 110 o 101, la nueva celda será 0. Si el patrón es 100, 011, 010 o 001, la nueva celda será 1. Si el patrón es 000, la nueva celda será 0. Esta regla se aplica simultáneamente a todas las celdas en cada generación, produciendo patrones complejos a partir de condiciones iniciales simples.

1.2 Objetivo de la Práctica

El objetivo de esta práctica es desarrollar un programa capaz de simular el autómata celular elemental conocido como Regla 30, permitiendo visualizar su evolución a lo largo de múltiples generaciones. El programa debe calcular el mayor número de generaciones posibles sin que las células activas alcancen las fronteras del autómata, utilizando cualquier método que facilite este análisis. Además, se busca responder las tres preguntas fundamentales planteadas por Stephen Wolfram respecto a la Regla 30: la periodicidad de la columna central, la frecuencia relativa de ceros y unos, y la complejidad computacional para determinar el estado de una célula en una generación dada.

2 Desarrollo

2.1 ¿En que fue desarrollado?

Decidí desarrollar este proyecto con JavaScript y hacerlo una applicación web para que esté disponible en cualquier momento, por lo que además decidí usar React como framework principal. React es una librería de JavaScript desarrollada por Facebook para construir interfaces de usuario de manera rápida, eficiente y escalable. Se basa en el uso de componentes reutilizables, piezas pequeñas de código que controlan su propio estado y se combinan para formar interfaces web completas. Usando un sistema llamado Virtual DOM, React actualiza solo las partes necesarias de la página, mejorando el rendimiento. Además, su sintaxis basada en JSX permite escribir HTML dentro de JavaScript, haciendo el código más intuitivo y fácil de mantener.

Entre sus principales beneficios están la reutilización de componentes, el flujo de datos unidireccional que simplifica la depuración, un gran ecosistema de herramientas y la facilidad de integración en proyectos nuevos o existentes. Es ideal para construir Single Page Applications (SPAs) o aplicaciones web dinámicas, aunque para sitios muy simples o estáticos puede que no sea la mejor opción.

2.2 Mi experiencia

El desarrollo de este proyecto fue una experiencia enriquecedora tanto a nivel técnico como personal. Al comenzar, tenía un conocimiento general sobre los autómatas celulares, pero implementar la Regla 30 desde cero me permitió profundizar en su funcionamiento y en la importancia de las reglas locales para la generación de patrones complejos.

Uno de los principales retos fue lograr una visualización eficiente y flexible, especialmente al trabajar con grandes cantidades de generaciones y columnas. Implementar el zoom interactivo y la navegación por el canvas, así como el manejo automático de los scrollbars, requirió investigar y experimentar con las capacidades del elemento canvas y la manipulación del DOM en JavaScript. Me resultó especialmente satisfactorio lograr que el zoom se centrara en el cursor y que el usuario pudiera desplazarse fácilmente por la simulación, ya fuera con scrollbars o mediante arrastre con Shift + clic.

Otra parte interesante fue la automatización de los análisis propuestos por Stephen Wolfram. Integrar la detección de periodicidad, el cálculo de frecuencias y la medición de la complejidad computacional me ayudó a comprender mejor la riqueza de comportamientos que puede surgir de reglas simples. Además, estructurar el código para que fuera modular y reutilizable, por ejemplo, centralizando los análisis en una sola función, facilitó mucho el mantenimiento y la extensión del programa.

En resumen, esta práctica no solo fortaleció mis habilidades de programación y resolución de problemas, sino que también me permitió apreciar la belleza y profundidad de los sistemas complejos. El resultado es una herramienta interactiva que facilita la exploración y el análisis de la Regla 30, y que me deja motivado para seguir aprendiendo sobre autómatas celulares y computación en general.

3 Código Fuente

3.1 Página principal

A continuación se muestra el código fuente de la aplicación. Al ser una aplicación desarrollada en React, se compone de varios módulos para su funcionamiento, sin embargo presentaré el código de una manera en la que se puedan omitir los módulos de React, es decir, como una página Web.

3.1.1 Código de la página (HTML):

```
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simulación regla 30</title>
</head>

<body>
    <div className="contenedor-principal">
        <h1>Simulación Regla 30</h1>
        <div className="zoom-control">
            <label>Zoom:</label>
            <button id="zoomInBtn" className="zoombtn">+</button>
            <button id="zoomOutBtn" className="zoombtn">-</button>
            <button id="resetZoomBtn">Reiniciar zoom</button>
            <div className="zoom-help" style=" background: #f5f5f5 , padding: 0.7rem ,
            ↪  borderRadius: 8px ,
                maxWidth: 600px">
                <strong>Controles de Zoom:</strong>
                <ul style= "textAlign: left , margin: 0.5rem 0 0 1.5rem ">
                    <li>
                        <b>Ctrl + Rueda del mouse</b>: Zoom enfocado en el punto del
                        ↪  cursor.
                    </li>
                    <li>
                        <b>Shift + clic y arrastrar</b>: Mover la vista del canvas.
                    </li>
```

```
                </ul>
            </div>
        </div>
        <div className="canvas-container">
            <canvas id="rule30Canvas"></canvas>
        </div>
        <div className="canvas-config">
            <label>Columnas:</label>
            <input id="colsInput" type="number" min="10" max="2000"
            ↪  defaultValue={1000} step={50} />
            <label>Filas:</label>
            <input id="rowsInput" type="number" min="10" max="1000" defaultValue={500}
            ↪  step={50} />
            <label>Celda (px):</label>
            <input id="cellSizeInput" type="number" min="1" max="15" defaultValue={1}
            ↪  step={1} />
            <button id="resizeCanvasBtn">Aplicar</button>
        </div>
        <div classNamе="botones-control">
            <button id="toggleBtn">Iniciar</button>
            <button id="resetBtn">Reiniciar</button>
            <select id="borderBehavior">
                <option value="stop">Detener al tocar borde</option>
                <option value="continue">Continuar normalmente</option>
                {/* <option value="invert">Invertir regla en borde</option> */}
            </select>
        </div>
        <div classNamе="analisis-control">
            <button id="runAllAnalysesBtn">Realizar todos los análisis</button>
            <button id="periodicityBtn">Analizar periodicidad</button>
            <button id="frequencyBtn">Analizar frecuencia</button>
            <button id="complexityBtn">Analizar complejidad</button>
            <input id="complexityInput" type="number" min="1" placeholder="Generacin
            ↪  n" defaultValue={1000} />
        </div>
        <div classNamе="contadores">
            <h3>Generaciones:<br /><span id="generationCounter">0</span></h3>
            <h3>Tiempo de generación:<br /><span id="generationTime">0 ms</span></h3>
            <h3>Tiempo total:<br /><span id="totalTime">0 ms</span></h3>
            <h3>Tiempo promedio:<br /><span id="averageTime">0 ms</span></h3>
        </div>
        <div classNamе="resultados-analisis">
            <div id="periodicityResult" classNamе="resultado-cuadro">Resultados de
            ↪  Periodicidad</div>
            <div id="frequencyResult" classNamе="resultado-cuadro">Resultados de
            ↪  Frecuencia</div>
            <div id="complexityResult" classNamе="resultado-cuadro">Resultados de
            ↪  Complejidad</div>
        </div>
    </div>
</body>

</html>
```

3.1.2 Hoja de Estilos (CSS):

La Hoja de estilos no afecta de manera funcional la applicación. La hoja de estilos es la siguiente:

```css
* {
  font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

canvas {
  border: 2px solid black;
  display: block;
  max-width: none;
  margin-left: auto;
  margin-right: auto;
}

button {
  padding: 0.5rem;
  margin: 0rem 0.25rem 0rem 0.25rem;
}

select,
input[type="number"] {
  height: 2.2rem;
  padding: 0 0.5rem;
  font-size: 1rem;
  border: 1px solid lightblue;
  border-radius: 4px;
  background: #ffffff;
  margin: 0 0.25rem;
  box-sizing: border-box;
  display: flex;
  align-items: center;
}

select:focus,
input[type="number"]:focus {
  outline: 2px solid blue;
  border-color: #000000;
}

.contenedor-principal {
  display: flex;
  flex-direction: column;
  align-items: center;
  text-align: center;
}

.canvas-container {
  width: 95vw;
  max-height: 70vh;
  overflow: auto;
  margin: 0 auto;
  position: relative;
  margin: 1rem 0rem 1rem 0rem;
}

.canvas-config{
  display: flex;
```

```
  align-items: center;
  gap: 0.5rem;
}

.botones-control,
.analisis-control,
select {
  display: flex;
  justify-content: center;
  align-items: center;
  gap: 0.25rem;
  margin: 0.5rem 0rem 0.5rem 0rem;
  vertical-align: middle;
}

.contadores {
  display: flex;
  gap: 2rem;
}

.resultados-analisis {
  display: flex;
  gap: 16px;
  justify-content: space-evenly;
  width: 90%;
}

.resultado-cuadro {
  background: #ffffff;
  border: 1px solid #000000;
  padding: 12px;
  width: 30%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  text-align: center;
  min-height: 80px;
  min-width: 120px;
  box-sizing: border-box;
}

.zoom-control {
  display: flex;
  justify-content: center;
  align-items: center;
  margin: 0.5rem 0;
  gap: 0.5rem;
}

.zoombtn{
  font-weight: bold;
}
```

3.2 Funcionalidades

3.2.1 Elementos y variables principales

En esta sección se inicializan los elementos principales de la interfaz y las variables globales que controlan el estado de la simulación. Se obtienen referencias al canvas, su contexto, los botones de control y los parámetros de la simulación, como el número de filas, columnas y el tamaño de cada celda. Además, se definen variables para el historial de la columna central, el control del tiempo y la navegación por zoom y arrastre.

```
/// [Configuración principal/Elementos y variables principales]
const canvas = document.getElementById("rule30Canvas");
const ctx = canvas.getContext("2d");
const canvasContainer = document.querySelector(".canvas-container");
const toggleBtn = document.getElementById("toggleBtn");
const borderBehavior = document.getElementById("borderBehavior");

/// [Configuración principal/Parámetros de la simulación]
let rows = 50, cols = 100, cellSize = 10;
let grid = Array(rows).fill(null).map(() => Array(cols).fill(0));
let running = false, animationId, currentRow = 0;
let centerColumnHistory = []; // Guarda la historia de la columna central
let totalExecutionTime = 0, generationTimes = [], simulationStartTime = null;
let generationCount = 0;

/// [Configuración principal/Parámetros de zoom y navegación]
let zoom = 1;
const minZoom = 0.5, maxZoom = 8, zoomStep = 0.25;
let isDragging = false, dragStartX = 0, dragStartY = 0, scrollStartX = 0,
↪  scrollStartY = 0;

// Inicializa el grid con una sola celda activa en el centro
grid[0][Math.floor(cols / 2)] = 1;
```

3.2.2 Dibujo de Canvas

Esta sección contiene las funciones responsables de renderizar el autómata en el canvas y de ajustar su tamaño cuando el usuario modifica los parámetros. La función principal, drawGrid, se encarga de dibujar cada celda según su estado, aplicando el nivel de zoom seleccionado. También se incluye la función para redimensionar el canvas y reiniciar la simulación.

```
/// [Dibujo de Canvas/Función para dibujar el autómata en el canvas]
function drawGrid() {
  // Ajusta el tamaño real del canvas según el zoom
  canvas.width = cols * cellSize * zoom;
  canvas.height = rows * cellSize * zoom;
  ctx.setTransform(1, 0, 0, 1, 0, 0); // Reset transform
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.scale(zoom, zoom);
  for (let y = 0; y <= currentRow; y++) {
    for (let x = 0; x < cols; x++) {
      ctx.fillStyle = grid[y][x] === 1 ? "black" : "white";
      ctx.fillRect(x * cellSize, y * cellSize, cellSize, cellSize);
    }
  }
  ctx.setTransform(1, 0, 0, 1, 0, 0); // Reset transform after drawing
}

/// [Dibujo de Canvas/Redimensiona el canvas y reinicia la simulación]
function resizeCanvas(newCols, newRows, newCellSize) {
```

```
  cols = newCols;
  rows = newRows;
  cellSize = newCellSize;
  canvas.width = cols * cellSize;
  canvas.height = rows * cellSize;
  resetSimulation();
}
```

3.2.3 Regla 30: Lógica y avance de la simulación

Aquí se implementa la lógica de la Regla 30, incluyendo la función que aplica la regla a cada fila (applyRule30) y la función que avanza la simulación una generación (step). También se incluye la función para reiniciar la simulación y limpiar los contadores. Estas funciones permiten la evolución del autómata y el registro de la historia de la columna central.

```
/// [Dibujo de Canvas/Verifica si la fila actual toca los bordes]
function isAtBorder(row) {
  return row[0] === 1 || row[cols - 1] === 1;
}

/// [Regla 30/Aplica la Regla 30 (o invertida) para generar la siguiente fila]
function applyRule30(prevRow, invert = false) {
  const newRow = Array(cols).fill(0);
  for (let i = 0; i < cols; i++) {
    const left = prevRow[i - 1] || 0;
    const center = prevRow[i];
    const right = prevRow[i + 1] || 0;
    const pattern = (left << 2) | (center << 1) | right;
    if (!invert) {
      newRow[i] = [0, 1, 1, 1, 1, 0, 0, 0][pattern];
    } else {
      newRow[i] = [1, 0, 0, 0, 0, 1, 1, 1][pattern]; // Invertida
    }
  }
  return newRow;
}

/// [Regla 30/Avanza una generación en la simulación]
function step() {
  if (currentRow < rows - 1) {
    let invert = false;
    if (isAtBorder(grid[currentRow])) {
      if (borderBehavior.value === "stop") {
        running = false;
        toggleBtn.innerText = "Iniciar";
        return;
      }
      if (borderBehavior.value === "invert") {
        invert = true;
      }
    }

    // Medición de tiempo de generación
    const genStart = performance.now();

    grid[currentRow + 1] = applyRule30(grid[currentRow], invert);
    currentRow++;
    const centerValue = grid[currentRow][Math.floor(cols / 2)];
    centerColumnHistory.push(centerValue);
```

```
45      drawGrid();
46      generationCount++;
47
48      // Actualización de tiempos y contadores
49      const genEnd = performance.now();
50      const genTime = genEnd - genStart;
51      generationTimes.push(genTime);
52
53      totalExecutionTime = performance.now() - simulationStartTime;
54      const avgTime = generationTimes.reduce((a, b) => a + b, 0) /
        ↪  generationTimes.length;
55
56      document.getElementById("generationCounter").innerText = generationCount;
57      document.getElementById("generationTime").innerText = formatTime(genTime);
58      document.getElementById("totalTime").innerText = formatTime(totalExecutionTime);
59      document.getElementById("averageTime").innerText = formatTime(avgTime);
60
61      animationId = requestAnimationFrame(step);
62    } else {
63      running = false;
64      toggleBtn.innerText = "Iniciar";
65      runAllAnalyses(); // Ejecuta los análisis al terminar
66    }
67  }
68
69  /// [Regla 30/Reinicia la simulación y limpia los contadores]
70  function resetSimulation() {
71    cancelAnimationFrame(animationId);
72    running = false;
73    toggleBtn.innerText = "Iniciar";
74    ctx.clearRect(0, 0, canvas.width, canvas.height);
75    grid = Array(rows).fill(null).map(() => Array(cols).fill(0));
76    grid[0][Math.floor(cols / 2)] = 1;
77    currentRow = 0;
78    generationCount = 0;
79    centerColumnHistory = [];
80    generationTimes = [];
81    totalExecutionTime = 0;
82    simulationStartTime = null;
83    document.getElementById("generationCounter").innerText = generationCount;
84    document.getElementById("generationTime").innerText = "0 ms";
85    document.getElementById("totalTime").innerText = "0 ms";
86    document.getElementById("averageTime").innerText = "0 ms";
87    drawGrid();
88  }
```

3.2.4 Análisis de las preguntas de Wolfram

Esta sección agrupa las funciones que responden a las tres preguntas planteadas por Stephen Wolfram sobre la Regla 30: periodicidad, frecuencia y complejidad. Cada función realiza el análisis correspondiente sobre la columna central del autómata y muestra los resultados en la interfaz.

```
1   /// [Análisis Preguntas/Análisis de periodicidad]
2   function analyzePeriodicity() {
3     const arr = centerColumnHistory;
4     let result = "No se detectó periodicidad en la columna central.";
5     for (let period = 1; period <= arr.length / 2; period++) {
6       let periodic = true;
7       for (let i = 0; i < arr.length - period; i++) {
```

```
8         if (arr[i] !== arr[i + period]) {
9           periodic = false;
10          break;
11        }
12      }
13      if (periodic) {
14        result = `¡Secuencia periódica detectada con periodo ${period}!`;
15        break;
16      }
17    }
18    document.getElementById("periodicityResult").innerText = result;
19  }
20
21  /// [Análisis Preguntas/Análisis de frecuencia de ceros y unos]
22  function analyzeFrequency() {
23    const arr = centerColumnHistory;
24    const zeros = arr.filter(x => x === 0).length;
25    const ones = arr.filter(x => x === 1).length;
26    const result = `Frecuencia:\nCeros: ${zeros} (${((zeros / arr.length) *
      ↪  100).toFixed(2)}%)\nUnos: ${ones} (${((ones / arr.length) * 100).toFixed(2)}%)`;
27    document.getElementById("frequencyResult").innerText = result;
28  }
29
30  /// [Análisis Preguntas/Análisis de complejidad (tiempo de cómputo para n)]
31  function analyzeComplexity() {
32    const input = document.getElementById("complexityInput");
33    let n = parseInt(input.value);
34    if (isNaN(n) || n < 1) {
35      document.getElementById("complexityResult").innerText = "Por favor ingresa un
        ↪  nmero de generacin vlido.";
36      return;
37    }
38    const increments = [0, 1000, 10000, 100000];
39    let times = [];
40    let results = "";
41
42    increments.forEach(inc => {
43      const gen = n + inc;
44      let prevGrid = Array(gen).fill(0);
45      prevGrid[Math.floor(prevGrid.length / 2)] = 1;
46      let row = prevGrid;
47      const t0 = performance.now();
48      for (let i = 1; i < gen; i++) {
49        row = applyRule30(row);
50      }
51      const t1 = performance.now();
52      const elapsed = t1 - t0;
53      times.push(elapsed);
54      results += `Generación ${gen}: ${elapsed.toFixed(2)} ms\n`;
55    });
56
57    let growth = "";
58    for (let i = 1; i < times.length; i++) {
59      const factor = times[i] / times[i - 1];
60      growth += `Crecimiento de ${n + increments[i - 1]} a ${n + increments[i]}:
        ↪  x${factor.toFixed(2)}\n`;
61    }
62
```

```
  document.getElementById("complexityResult").innerText =
    `Tiempos para calcular la celda central:\n${results}\nFactores de
    ↪  crecimiento:\n${growth}`;
}

/// [Análisis Preguntas/Ejecuta todos los análisis]
function runAllAnalyses() {
  analyzePeriodicity();
  analyzeFrequency();
  analyzeComplexity();
}
```

3.2.5 Eventos de control y navegación

En esta sección se encuentran los controladores de eventos para los botones de la interfaz, el zoom, el desplazamiento y la navegación por arrastre. Permiten al usuario interactuar con la simulación, modificar parámetros, controlar el avance, hacer zoom y moverse por el canvas de manera intuitiva.

```
/// [Eventos de control de la simulación]
toggleBtn.onclick = () => {
  if (!running) {
    running = true;
    toggleBtn.innerText = "Pausar";
    if (!simulationStartTime) simulationStartTime = performance.now();
    animationId = requestAnimationFrame(step);
  } else {
    running = false;
    toggleBtn.innerText = "Iniciar";
    cancelAnimationFrame(animationId);
  }
};

document.getElementById("periodicityBtn").onclick = analyzePeriodicity;
document.getElementById("frequencyBtn").onclick = analyzeFrequency;
document.getElementById("complexityBtn").onclick = analyzeComplexity;
document.getElementById("resetBtn").onclick = resetSimulation;
document.getElementById("runAllAnalysesBtn").onclick = runAllAnalyses;

document.getElementById("resizeCanvasBtn").onclick = () => {
  const newCols = parseInt(document.getElementById("colsInput").value);
  const newRows = parseInt(document.getElementById("rowsInput").value);
  const newCellSize = parseInt(document.getElementById("cellSizeInput").value);
  resizeCanvas(newCols, newRows, newCellSize);
};

/// [Zoom/Navegación y zoom con scroll]
canvas.addEventListener("wheel", function (e) {
  if (!e.ctrlKey) return;
  e.preventDefault();

  const rect = canvas.getBoundingClientRect();
  const mouseX = e.clientX - rect.left + canvasContainer.scrollLeft;
  const mouseY = e.clientY - rect.top + canvasContainer.scrollTop;
  const logicalX = mouseX / zoom;
  const logicalY = mouseY / zoom;

  let newZoom = zoom + (e.deltaY < 0 ? zoomStep : -zoomStep);
  newZoom = Math.max(minZoom, Math.min(maxZoom, newZoom));
  if (newZoom === zoom) return;
```

```
42
43    zoom = newZoom;
44    drawGrid();
45
46    canvasContainer.scrollLeft = (logicalX * zoom) - (e.clientX - rect.left);
47    canvasContainer.scrollTop = (logicalY * zoom) - (e.clientY - rect.top);
48  }, { passive: false });
49
50  /// [Zoom/Controles de zoom]
51  document.getElementById("zoomInBtn").onclick = () => {
52    if (zoom < maxZoom) {
53      zoom += zoomStep;
54      drawGrid();
55    }
56  };
57  document.getElementById("zoomOutBtn").onclick = () => {
58    if (zoom > minZoom) {
59      zoom -= zoomStep;
60      drawGrid();
61    }
62  };
63  document.getElementById("resetZoomBtn").onclick = () => {
64    zoom = 1;
65    drawGrid();
66  };
67
68  /// [Dibujo interactivo/Navegación por arrastre]
69  canvasContainer.addEventListener("mousedown", function (e) {
70    if (e.shiftKey) {
71      isDragging = true;
72      dragStartX = e.clientX;
73      dragStartY = e.clientY;
74      scrollStartX = canvasContainer.scrollLeft;
75      scrollStartY = canvasContainer.scrollTop;
76      canvasContainer.style.cursor = "grab";
77      e.preventDefault();
78    }
79  });
80
81  window.addEventListener("mousemove", function (e) {
82    if (isDragging) {
83      const dx = e.clientX - dragStartX;
84      const dy = e.clientY - dragStartY;
85      canvasContainer.scrollLeft = scrollStartX - dx;
86      canvasContainer.scrollTop = scrollStartY - dy;
87    }
88  });
89
90  window.addEventListener("mouseup", function () {
91    if (isDragging) {
92      isDragging = false;
93      canvasContainer.style.cursor = "";
94    }
95  });
```

3.2.6 Dibujar Cuadrícula:

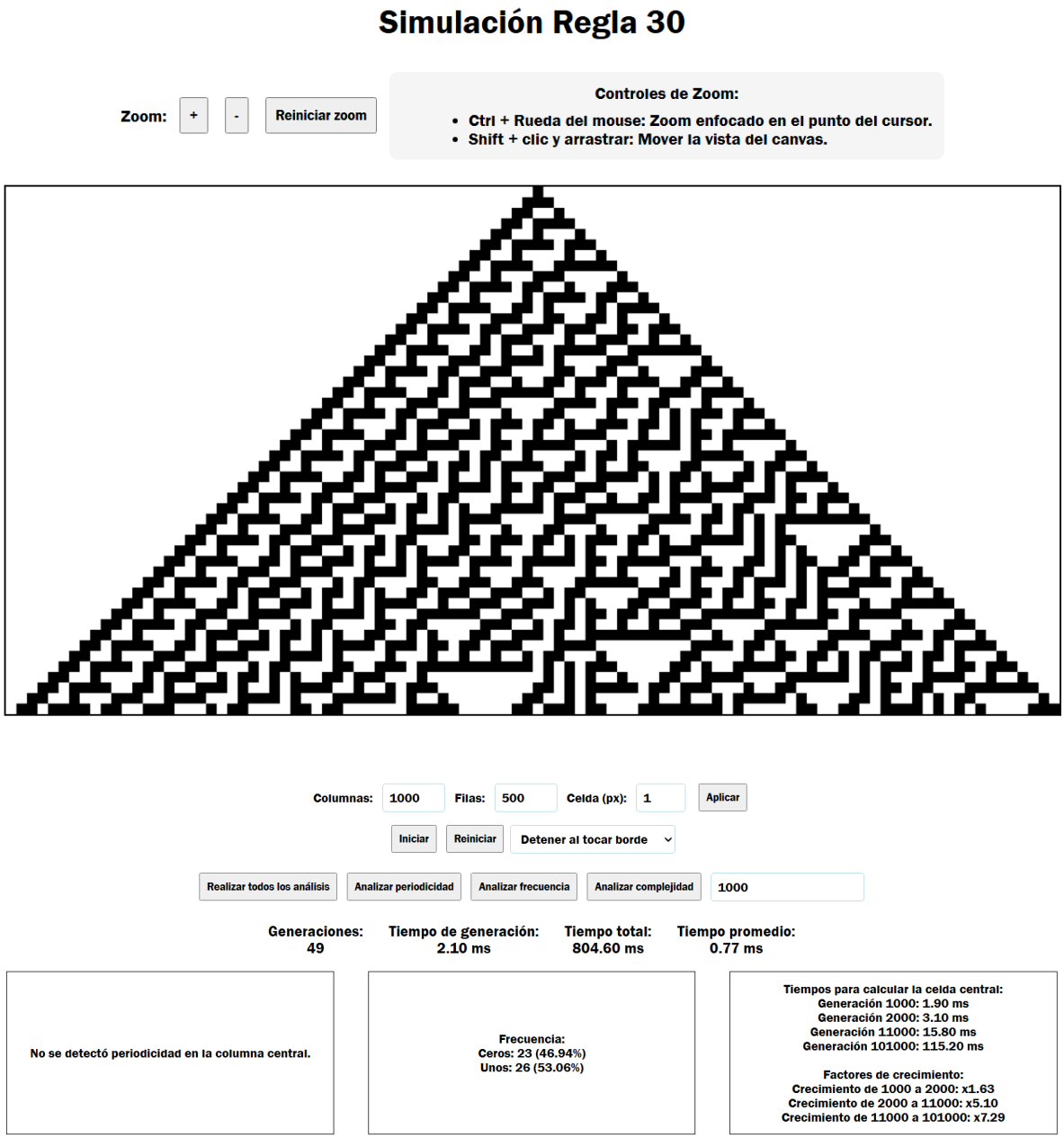
Finalmente, debo mencionar que todo el código anterior se encuentra dentro de una función que se carga cuando la pestaña del navegador carga. Al final de todo esté código anterior tengo la llamada a la función drawGrid() para el dibujo del Canvas.

```
window.onload = function () {
    // ----- Código anterior en el orden presentado -----

    // ---------- DUBUJAR CUADRÍCULA ----------
    drawGrid();
}
```

3.3 Vista Final

A continuación muestro como se ve la página una vez cargada.



4 ¿Dónde encontrar el código?

El código se encuentra disponible de manera pública en el siguiente repositorio de GitHub:

- https://github.com/LuisContla/WolframRegla30

Además, si no se desea descargar el código completo se puede encontrar una demo en el siguiente enlace:

- https://wolfram-regla30.vercel.app/

5 Conclusión

El desarrollo de este proyecto me permitió comprender a profundidad el funcionamiento de los autómatas celulares, en particular la Regla 30 de Wolfram, y apreciar cómo reglas simples pueden generar comportamientos complejos y patrones impredecibles. A lo largo de la práctica, aprendí a diseñar simulaciones visuales interactivas, implementar análisis automáticos y optimizar la experiencia del usuario mediante herramientas como el zoom, el desplazamiento y la configuración dinámica del autómata.

Uno de los principales retos fue lograr que la simulación fuera eficiente y navegable incluso con grandes cantidades de generaciones y columnas, lo que requirió investigar técnicas para manipular el canvas, gestionar el zoom centrado en el cursor y permitir la navegación fluida mediante scrollbars y arrastre. También resultó desafiante automatizar los análisis propuestos por Wolfram, especialmente la detección de periodicidad y la medición de la complejidad computacional, lo que me llevó a estructurar el código de manera modular y reutilizable.

En resumen, esta práctica fortaleció mis habilidades de programación, resolución de problemas y visualización computacional, además de motivarme a seguir explorando el fascinante mundo de los sistemas complejos y los autómatas celulares.