El Juego de la Vida de Conway

Luis Andrés Contla Mota

Abril 2025

Índice

1 Introducción

1.1 ¿Qué es el juego de la Vida de Conway?

El Juego de la Vida de Conway es un autómata celular creado por el matemático John Conway en 1970. Es un "juego" de simulación matemática donde un tablero (una cuadrícula) evoluciona en el tiempo siguiendo reglas simples. Cada celda en la cuadrícula puede estar:

- Viva (1)
- Muerta (0)

La evolución se da por generaciones, y depende de los vecinos de cada celda (las 8 celdas alrededor). Las reglas son:

- Supervivencia: Una célula viva con 2 o 3 vecinos vivos sigue viva.
- Muerte por soledad: Una célula viva con menos de 2 vecinos vivos muere.
- Muerte por sobrepoblación: Una célula viva con más de 3 vecinos vivos muere.
- Nacimiento: Una célula muerta con exactamente 3 vecinos vivos nace.

Aunque las reglas son simples, el juego puede generar patrones muy complejos, como estructuras que se mueven (como los gliders), se reproducen o evolucionan de maneras impredecibles. No es un "juego" en el sentido tradicional (no hay jugadores), sino una simulación matemática de vida artificial.

1.2 Objetivo de la Práctica

El objetivo de esta tarea es desarrollar un programa que simule autómatas celulares bidimensionales del tipo Life, incorporando una representación gráfica interactiva del espacio celular. El sistema deberá ser capaz de manejar espacios de diferentes dimensiones, permitiendo la edición directa de las configuraciones, el ajuste del tamaño de las células, el cambio de colores de los estados y la posibilidad de guardar y cargar configuraciones desde archivos. Además, deberá incluir herramientas para controlar manual o automáticamente la evolución de las generaciones, graficar la densidad poblacional en escala lineal y logarítmica, y permitir la simulación bajo diferentes condiciones de frontera y reglas definidas mediante la notación B/S.

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

Mi experiencia desarrollando este programa fue un proceso de aprendizaje continuo y expansión de habilidades, con desafíos y logros en cada fase. El proyecto comenzó con la implementación del Juego de la Vida de Conway, un modelo celular que simula la evolución de células vivas en una cuadrícula. Mi

enfoque inicial fue crear una base funcional, donde las celdas se pudieran activar y desactivar manualmente, utilizando un lienzo (canvas) y una matriz de celdas.

La configuración del canvas fue el primer paso crucial, ya que me permitió visualizar el estado del juego en una cuadrícula de 50x100, donde las celdas podían tener diferentes estados (vivas o muertas). Posteriormente, me enfoqué en las funcionalidades de control, como iniciar, pausar y reiniciar el juego, además de permitir la generación aleatoria de patrones y la adición de nuevas celdas de manera aleatoria. Implementé un sistema de reglas B/S (nacimiento y supervivencia), que me permitió experimentar con diferentes configuraciones, además de la posibilidad de cambiar la velocidad de ejecución del juego.

Uno de los mayores retos fue implementar la funcionalidad toroidal. Este modo implicó que las celdas en los bordes se conectaran con las celdas opuestas, lo que resultó en un desafío para la lógica de actualización de la cuadrícula. Sin embargo, tras varios intentos y pruebas, logré integrar este modo de forma que fuera opcional para el usuario mediante un toggle, con una interfaz limpia y accesible.

A medida que el juego avanzaba, me di cuenta de la importancia de almacenar y mostrar estadísticas como el número de generaciones, el total de celdas vivas, la densidad poblacional, y el logaritmo de las celdas vivas. Estas estadísticas no solo eran útiles para el análisis del juego, sino también para hacer la experiencia más interactiva y educativa. Fue un reto calcular estas métricas en tiempo real, pero me permitió experimentar con conceptos como la varianza y la media de celdas vivas, lo cual me ayudó a mejorar mis habilidades en matemáticas aplicadas en programación.

Uno de los aspectos clave fue la importación y exportación de datos. Implementé funciones para permitir a los usuarios guardar y cargar el estado del juego en archivos JSON. Esto me permitió explorar el manejo de archivos y cómo almacenar de forma eficiente todos los datos necesarios para restaurar el estado del juego. Además, la funcionalidad de importar y exportar incluyó no solo la cuadrícula y las celdas vivas, sino también configuraciones como las reglas B/S, los colores, el estado del modo toroidal, y las estadísticas del juego.

Durante el proceso, también enfrenté desafíos relacionados con la interactividad del canvas, como permitir que el usuario dibujara sobre la cuadrícula con el mouse mientras el juego estaba en ejecución. Esto implicó pausar el juego cuando el usuario comenzaba a dibujar, lo cual fue un reto técnico en términos de cómo gestionaba el estado del juego. Sin embargo, logré hacer que el juego fuera lo suficientemente flexible como para adaptarse a diferentes interacciones sin perder la coherencia del estado.

Finalmente, un aspecto que me permitió aprender mucho fue la integración de event listeners y la manera de estructurar los controles para que fueran claros y fáciles de usar, mientras que el juego seguía siendo eficiente. A medida que desarrollaba cada funcionalidad, como la actualización de las estadísticas o el ajuste de la velocidad, fui comprendiendo mejor cómo manejar las interacciones en un entorno dinámico.

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 páginia Web.

3.1.1 Código de la página (HTML):

El código HTML es el siguiente:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" href="/src/assets/Conway.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Conway</title>
```

```
</head>
<body>
  <div class="contenedor-principal">
      <div class="titulo">
        <h1>Juego de la Vida de Conway</h1>
      </div>
      <div class="inputs-configuracion">
        <label html-for="inputRows">Filas:</label>
        <input type="number" id="inputRows" min={50} default-value={50} step={50} />
        <label html-for="inputCols">Columnas:</label>
        <input type="number" id="inputCols" min={50} default-value={100} step={50}
        ↪  />
        <label html-for="inputCellSize">Tamaño de las celdas:</label>
        <input type="number" id="inputCellSize" min={2} default-value={10} />
        <button id="updateSizeBtn">Actualizar Tamaño</button>
      </div>
      <br />
      <div class="juego-conway">
        <canvas id="gameCanvas" />
        <br /><br />
      </div>
      <div class="botones">
        <div class="botones-control">
          <button id="generateRandomBtn">Generar Aleatorio</button>
          <button id="addRandomBtn">Añadir Aleatorios</button>
          <button id="toggleGame">Iniciar</button>
          <button id="resetBtn">Reiniciar</button>
          <button id="previousGenerationBtn">Retroceder Generación</button>
          <button id="nextGenerationBtn">Avanzar Generación</button>
          <label html-for="toroidalToggle">Toroidal:</label>
          <label class="switch">
            <input type="checkbox" id="toroidalCheck" />
            <span class="slider round" />
          </label>
        </div>
        <div class="botones-velocidad">
          <label html-for="speedInput">Velocidad (ms): </label>
          <button id="minSpeed">min</button>
          <button id="decreaseSpeed"></button>
          <input type="number" id="speedInput" default-value={50} min={50} step={50}
          ↪  read-only />
          <button id="increaseSpeed">+</button>
          <button id="maxSpeed">máx</button>
        </div>
        <div class="botones-colores">
          <label html-for="celdaVivaColor">Celdas Vivas:</label>
          <input type="color" id="celdaVivaColor" name="celdaVivaColor"
          ↪  default-value="#000000" />
          <label html-for="celdaViva">Celdas Muertas:</label>
          <input type="color" id="celdaMuertaColor" name="celdaMuertaColor"
          ↪  default-value="#FFFFFF" />
        </div>
      </div>
      <div class="inputs-reglas">
        <label html-for="ruleB">B</label>
        <input class="rules" type="number" id="ruleB" default-value={3} />
        <label html-for="ruleS">/S</label>
        <input class="rules" type="number" id="ruleS" default-value={23} />
```

```
        </div>
        <div class="contadores">
          <h3>Generaciones:<br /><span id="generationCounter">0</span></h3>
          <h3>Celdas Vivas:<br /><span id="aliveCounter">0</span></h3>
          <h3>Densidad Poblacional:<br /><span id="populationDensity">0</span></h3>
          <h3>Logaritmo Base 10:<br /><span id="logBase10">0</span></h3>
          <h3> Media de Celdas Vivas:<br /><span id="meanAliveCells">0</span></h3>
          <h3>Varianza:<br /><span id="variance">0</span></h3>
          <h3>Total de Celdas Vivas:<br /><span id="totalAliveCells">0</span></h3>
        </div>
        <div class="export-import-controls">
          <button id="exportBtn">Exportar</button>
          <input type="file" id="importFile" style="display: none;" />
          <button id="importBtn">Importar</button>
        </div>
      </div>
    <script type="module" src="/src/main.jsx"></script>
  </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:

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

canvas {
  border: 2px solid black;
}

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

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

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

.botones-colores{
  gap: 1rem;
}

.contadores{
  display: flex;
```

```
  gap: 2rem;
}

.inputCanvas{

}

/* ---------- INPUTS TOGGLE ---------- */

/* The switch - the box around the slider */
.switch {
  position: relative;
  display: inline-block;
  width: 60px;
  height: 34px;
}

/* Hide default HTML checkbox */
.switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

/* The slider */
.slider {
  position: absolute;
  cursor: pointer;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: #ccc;
  -webkit-transition: .4s;
  transition: .4s;
}

.slider:before {
  position: absolute;
  content: "";
  height: 26px;
  width: 26px;
  left: 4px;
  bottom: 4px;
  background-color: white;
  -webkit-transition: .4s;
  transition: .4s;
}

input:checked + .slider {
  background-color: #2196F3;
}

input:focus + .slider {
  box-shadow: 0 0 1px #2196F3;
}

input:checked + .slider:before {
```

```
  -webkit-transform: translateX(26px);
  -ms-transform: translateX(26px);
  transform: translateX(26px);
}

/* Rounded sliders */
.slider.round {
  border-radius: 34px;
}

.slider.round:before {
  border-radius: 50%;
}

/* ---------- Ocultar flechas de inputs de número ---------- */

/* Chrome, Safari, Edge, Opera */
.rules::-webkit-outer-spin-button,
.rules::-webkit-inner-spin-button {
  -webkit-appearance: none;
  margin: 0;
}

/* Firefox */
.rules[type=number] {
  -moz-appearance: textfield;
}
```

3.2 Funcionalidades

El script principal de la página se divide en varias funciones. Presentaré una a una de ellas y las explicaré por separado.

3.2.1 Configurando el Canvas:

En esta sección del código, se configura el canvas donde se va a representar el Juego de la Vida. Se obtiene el elemento ¡canvas¿ del HTML utilizando document.getElementById(”gameCanvas”) y se establece el contexto de dibujo con canvas.getContext(”2d”). Esto permite interactuar con el canvas y dibujar sobre él. Se definen las dimensiones de la cuadrícula del juego con el número de filas (rows) y columnas (cols), que están fijados en 50 y 100, respectivamente. Además, se asigna un tamaño específico a cada celda de la cuadrícula mediante la variable cellSize, que se establece en 10 píxeles. Finalmente, se ajustan las dimensiones del canvas para que coincidan con el número total de celdas y su tamaño, calculando el ancho y el alto multiplicando las filas y columnas por el tamaño de la celda. Esto configura el área de trabajo para el Juego de la Vida.

```
// ---------- CONFIGURANDO EL CANVAS ----------

    const canvas = document.getElementById("gameCanvas");
    const ctx = canvas.getContext("2d");
    let rows = 500, cols = 500;
    let cellSize = 3;
    canvas.width = cols * cellSize;
    canvas.height = rows * cellSize;
```

3.2.2 Variables Principales:

En esta sección se definen las variables esenciales para controlar el estado y el comportamiento del Juego de la Vida. Se inicializa la velocidad del juego (speed), obteniendo el valor de un input en la interfaz. Se crea una matriz grid que almacena el estado del juego, donde cada celda tiene un valor inicial de

0 (muerta). La variable running indica si el juego está en ejecución, mientras que intervalId guarda la referencia al intervalo de actualización del juego.

Se utilizan las variables isDrawing y drawState para gestionar la interacción con el canvas y el estado de las celdas cuando el usuario dibuja sobre ellas. El contador de generaciones (generationCount) lleva el número de generaciones avanzadas, y aliveCount mantiene el total de celdas vivas en cada generación. La variable totalAliveCells acumula el número total de celdas vivas a lo largo de todas las generaciones. Además, se define un historial de generaciones (history), con su respectivo índice (historyIndex), que permite retroceder o avanzar entre generaciones pasadas.

Se incluye también la variable isToroidal, que determina si el juego sigue el comportamiento toroidal (conexión entre bordes de la cuadrícula), y las reglas de nacimiento (ruleB) y supervivencia (ruleS), con los valores predeterminados B3/S23.

```
// ---------- VARIABLES PRINCIPALES ----------

    let speed = parseInt(document.getElementById("speedInput").value); // Velocidad
    ↪ inicial desde el input
    let grid = Array.from({ length: rows }, () => Array(cols).fill(0)); // Matriz para
    ↪ almacenar el estado del juego
    let running = false; // Indica si el juego está corriendo
    let intervalId = null; // Referencia del intervalo de actualización
    let isDrawing = false; // Indica si el usuario está dibujando en el canvas
    let drawState = null; // Estado de la celda al hacer clic (1 o 0)
    let wasRunning = false; // Indica si el juego estaba corriendo antes de dibujar
    let generationCount = 0; // Contador de generaciones
    let aliveCount = 0; //Contador de celdas vivas
    let totalAliveCells = 0; // Variable para almacenar la suma total de celdas vivas
    let maxGenerations = 10; // Definir el mximo de generaciones a guardar en el
    ↪ historia
    let history = []; // Historial de generaciones
    let historyIndex = -1; // Índice de la generación actual en el historial
    let isToroidal = false; // Variable para verificar si el modo toroidal est
    ↪ activado
    let ruleB = [3]; // Reglas de nacimiento, por defecto B3
    let ruleS = [2, 3]; // Reglas de supervivencia, por defecto S23

```

3.2.3 Referencias a botones e Inputs:

En esta sección se obtienen las referencias a los elementos HTML que permiten la interacción con el juego. Se usa document.getElementById() para vincular los botones e inputs del HTML con variables en el código JavaScript. Estas variables permiten manipular y controlar las acciones del usuario.

Se definen botones como el toggleGameButton para alternar entre iniciar y pausar el juego, el generateRandomBtn para generar una cuadrícula aleatoria, y los botones exportBtn y importBtn para exportar e importar archivos de configuración. Además, se configuran los inputs de color celdaVivaColorInput y celdaMuertaColorInput para permitir al usuario elegir el color de las celdas vivas y muertas, respectivamente. Finalmente, se incluyen los inputs ruleBInput y ruleSInput, que permiten modificar las reglas de nacimiento y supervivencia (B/S) en el juego. Entre otras. Estas referencias se utilizan más adelante para interactuar con el estado del juego, habilitar/deshabilitar acciones o actualizar la interfaz.

```
// ---------- REFERENCIAS A BOTONES E INPUTS ----------

    const toggleGameButton = document.getElementById("toggleGame");
    const generateRandomBtn = document.getElementById("generateRandomBtn");
    const celdaVivaColorInput = document.getElementById("celdaVivaColor");
    const celdaMuertaColorInput = document.getElementById("celdaMuertaColor");
    const ruleBInput = document.getElementById("ruleB");
    const ruleSInput = document.getElementById("ruleS");
```

```
    const exportBtn = document.getElementById("exportBtn");
    const importBtn = document.getElementById("importBtn");
    const inputRows = document.getElementById("inputRows");
    const inputCols = document.getElementById("inputCols");
    const inputCellSize = document.getElementById("inputCellSize");
    const updateSizeBtn = document.getElementById("updateSizeBtn");
```

3.2.4 Dibujo del Canvas:

La función drawGrid() es responsable de dibujar el estado actual del juego en el canvas. Primero, obtiene los valores de color seleccionados por el usuario para las celdas vivas y muertas mediante los inputs de color. Luego, limpia el canvas usando ctx.clearRect() para borrar cualquier dibujo anterior.

A continuación, la función recorre cada celda de la cuadrícula (grid) y determina si la celda está viva o muerta (representada por los valores 1 o 0 en la matriz). Según el estado de la celda, se asigna el color correspondiente (aliveColor o deadColor). Después, se dibujan las celdas como rectángulos en el canvas utilizando ctx.fillRect() y se les dibuja un borde gris con ctx.strokeRect().

Esta función se llama continuamente para actualizar la visualización de la cuadrícula cada vez que se realizan cambios en el estado del juego.

La función updateCanvasSize() permite actualizar dinámicamente el tamaño del canvas según los valores proporcionados por los inputs para filas, columnas y tamaño de celdas. Después de ajustar las dimensiones del canvas, se reinicia la cuadrícula y se redibuja con las nuevas configuraciones. Esto permite cambiar la resolución del juego sin tener que recargar la página.

```
// ---------- DIBUJO DEL CANVAS ----------

    function drawGrid() {
        // Leemos el valor actual de los colores
        const aliveColor = document.getElementById("celdaVivaColor").value;
        const deadColor = document.getElementById("celdaMuertaColor").value;

        ctx.clearRect(0, 0, canvas.width, canvas.height);

        for (let y = 0; y < rows; y++) {
            for (let x = 0; x < cols; x++) {
                ctx.fillStyle = grid[y][x] === 1 ? aliveColor : deadColor; // Celda
                ↪  viva (negra) o muerta (blanca)
                ctx.fillRect(x * cellSize, y * cellSize, cellSize, cellSize);
                ctx.strokeStyle = "gray";
                ctx.strokeRect(x * cellSize, y * cellSize, cellSize, cellSize);
            }
        }
    }

    // Función para actualizar el tamaño del canvas
    function updateCanvasSize() {
        // Obtener las nuevas dimensiones del canvas desde los inputs
        rows = parseInt(inputRows.value);
        cols = parseInt(inputCols.value);
        cellSize = parseInt(inputCellSize.value);

        // Actualizar las dimensiones del canvas
        canvas.width = cols * cellSize;
        canvas.height = rows * cellSize;

        // Re-inicializar la cuadrícula
        grid = Array.from({ length: rows }, () => Array(cols).fill(0));
```

```
33
34          // Redibujar el canvas
35          drawGrid();
36      }
```

3.2.5 Funcionalidades de Control:

En esta sección se gestionan las funcionalidades de control del juego, como iniciar, pausar y reiniciar el juego, así como la manipulación de los botones y la interfaz de usuario.

La función toggleGame() alterna el estado entre "Iniciar" y "Pausar". Si el juego no está corriendo, se establece el intervalo para actualizar el juego y se deshabilitan los controles de reglas y generación aleatoria, bloqueando las modificaciones mientras el juego está en curso. Si el juego está corriendo, el intervalo se detiene y los controles se vuelven habilitados nuevamente, permitiendo cambiar las reglas y realizar otras modificaciones.

La función stopGame() detiene el juego, limpiando el intervalo de actualización y cambiando el texto del botón a "Iniciar".

La función resetGame() reinicia completamente el juego. Detiene el juego, borra la cuadrícula, restablece los contadores de generaciones y celdas vivas, y limpia el historial. También restablece las reglas predeterminadas de nacimiento y supervivencia (B3/S23), habilita los botones para cambiar configuraciones y restablece todos los valores visuales, incluidos los colores de las celdas y las estadísticas de la interfaz.

```
1   // ---------- FUNCIONALIDADES DE CONTROL ----------
2
3       // Función para alternar entre "Iniciar" y "Pausar"
4       function toggleGame() {
5           if (!running) {
6               running = true;
7               intervalId = setInterval(update, speed);
8               ruleBInput.disabled = true;
9               ruleSInput.disabled = true;
10              generateRandomBtn.disabled = true;
11              exportBtn.disabled = true;
12              importBtn.disabled = true;
13              toggleGameButton.innerText = "Pausar";
14          } else {
15              running = false;
16              clearInterval(intervalId);
17              ruleBInput.disabled = false;
18              ruleSInput.disabled = false;
19              generateRandomBtn.disabled = false;
20              exportBtn.disabled = false;
21              importBtn.disabled = false;
22              toggleGameButton.innerText = "Iniciar";
23          }
24      }
25
26      // Pausa el juego
27      function stopGame() {
28          running = false;
29          clearInterval(intervalId);
30          toggleGameButton.innerText = "Iniciar";
31      }
32
33      // Reinicia el juego y el contador de generaciones
34      function resetGame() {
35          stopGame();
```

```
        grid = Array.from({ length: rows }, () => Array(cols).fill(0));
        generationCount = 0;
        aliveCount = 0;
        totalAliveCells = 0;
        history = []; // Limpiamos el historial
        historyIndex = -1; // Restablecemos el índice del historial
        ruleB = [3]; // Reglas de nacimiento, por defecto B3
        ruleS = [2, 3]; // Reglas de supervivencia, por defecto S23
        generateRandomBtn.disabled = false;
        ruleBInput.disabled = false;
        ruleSInput.disabled = false;
        exportBtn.disabled = false;
        importBtn.disabled = false;
        celdaMuertaColorInput.value = "#FFFFFF";
        celdaVivaColorInput.value = "#000000";
        document.getElementById("generationCounter").innerText = generationCount;
        document.getElementById("aliveCounter").innerText = aliveCount;
        document.getElementById("populationDensity").innerText = 0;
        document.getElementById("meanAliveCells").innerText = 0;
        document.getElementById("variance").innerText = 0;
        document.getElementById("logBase10").innerText = 0;
        document.getElementById("totalAliveCells").innerText = 0;
        document.getElementById("ruleB").value = ruleB;
        document.getElementById("ruleS").value = 23;
        drawGrid();
    }
```

3.2.6 Funcionalidades Principales:

En esta sección del código se manejan las principales funcionalidades del Juego de la Vida. Estas incluyen la generación de la cuadrícula aleatoria, la actualización de las generaciones, el cambio de las reglas del juego y el manejo del modo toroidal.

La función generateRandomGrid() genera una cuadrícula completamente aleatoria, reinicia el contador de generaciones y pausa el juego. La cuadrícula se llena con un 30 porciento de celdas vivas, y el contador de celdas vivas se actualiza y muestra en la interfaz.

La función addRandomCells() agrega nuevas células vivas a los espacios vacíos sin afectar las celdas ya existentes. Al igual que la anterior, pausa el juego y actualiza las estadísticas, como el número de celdas vivas.

La función getNextGeneration() calcula la siguiente generación siguiendo las reglas del Juego de la Vida. Recorre cada celda y evalúa sus vecinos para determinar su estado en la siguiente iteración. Si el modo toroidal está activado, las celdas que están en los bordes se conectan a las celdas opuestas en el borde opuesto de la cuadrícula. Luego, actualiza el contador de generaciones y celdas vivas, guarda el estado en el historial y actualiza las estadísticas.

La función update() es la que se ejecuta repetidamente para actualizar el estado del juego. Llama a getNextGeneration() para calcular la siguiente generación y actualiza la interfaz con los valores de las estadísticas en tiempo real.

La función updateRules() actualiza las reglas de nacimiento (B) y supervivencia (S) dinámicamente, tomando los valores de los inputs de la interfaz y actualizándolos en el juego solo cuando este está en pausa.

Finalmente, el evento del toggle del modo toroidal activa o desactiva el comportamiento toroidal, es decir, cuando se activa, las celdas en los bordes de la cuadrícula se conectan a las celdas en el lado opuesto. Además, al cambiar el estado del modo toroidal, se ajusta el borde del canvas para reflejar el estado actual (1px cuando está activado, 2px cuando está desactivado).

```
// ---------- FUNCIONALIDADES PRINCIPALES ----------

    // Genera una cuadrcula completamente aleatoria, reinicia el contador y pausa el juego
    function generateRandomGrid() {
        stopGame();
        grid = Array.from({ length: rows }, () => Array(cols).fill(0));

        for (let y = 0; y < rows; y++) {
            for (let x = 0; x < cols; x++) {
                grid[y][x] = Math.random() > 0.7 ? 1 : 0; // 30% de probabilidad de clula viva
            }
        }

        generationCount = 0; // Reiniciar el contador de generaciones
        totalAliveCells = 0;
        document.getElementById("generationCounter").innerText = generationCount;
        drawGrid();

        // Actualizamos el contador de celdas vivas
        const aliveCount = countAliveCells();
        document.getElementById("aliveCounter").innerText = aliveCount; // Muestra el contador de celdas vivas
        updateStatistics(aliveCount);
    }

    // Añade nuevas células vivas sin afectar las ya existentes, y pausa el juego
    function addRandomCells() {
        for (let y = 0; y < rows; y++) {
            for (let x = 0; x < cols; x++) {
                if (grid[y][x] === 0 && Math.random() > 0.9) {
                    grid[y][x] = 1; // Solo añade células en los espacios vacíos
                }
            }
        }
        drawGrid();

        // Actualizamos el contador de celdas vivas
        const aliveCount = countAliveCells();
        document.getElementById("aliveCounter").innerText = aliveCount; // Muestra el contador de celdas vivas
        updateStatistics(aliveCount);
    }

    // Calcula la siguiente generación basada en las reglas del Juego de la Vida
    function getNextGeneration() {
        let newGrid = grid.map(arr => [...arr]); // Copia la matriz actual

        for (let y = 0; y < rows; y++) {
            for (let x = 0; x < cols; x++) {
                let neighbors = 0;

                // Recorre los 8 vecinos de cada celda
                for (let i = -1; i <= 1; i++) {
                    for (let j = -1; j <= 1; j++) {
                        if (i === 0 && j === 0) continue; // Ignora la propia celda

```

```
55                          let nx = x + i;
56                          let ny = y + j;
57
58                          // Si el modo toroidal est activado, usamos el mdulo para
                            ↪  envolver las coordenadas
59                          if (isToroidal) {
60                              nx = (nx + cols) % cols;
61                              ny = (ny + rows) % rows;
62                          }
63
64                          // Verificamos que las nuevas coordenadas estn dentro del
                            ↪  tablero
65                          if (nx >= 0 && nx < cols && ny >= 0 && ny < rows) {
66                              neighbors += grid[ny][nx]; // Contamos el vecino si est
                                ↪  vivo
67                          }
68                      }
69                  }
70
71                  // Aplicamos las reglas de B/S dinámicamente
72                  if (grid[y][x] === 1 && !ruleS.includes(neighbors)) {
73                      newGrid[y][x] = 0; // Muerte por no estar en la regla de
                        ↪  supervivencia
74                  }
75                  if (grid[y][x] === 0 && ruleB.includes(neighbors)) {
76                      newGrid[y][x] = 1; // Nace una nueva clula segn la regla de
                        ↪  nacimiento
77                  }
78              }
79          }
80
81          grid = newGrid;
82          generationCount++; // Incrementa el contador de generaciones
83
84          aliveCount = countAliveCells(); // Calcula el número de celdas vivas
85          totalAliveCells += aliveCount; // Acumula el total de celdas vivas
86
87          document.getElementById("generationCounter").innerText = generationCount; //
            ↪  Actualiza la interfaz
88          document.getElementById("aliveCounter").innerText = aliveCount; // Actualiza
            ↪  el contador de celdas vivas
89          drawGrid(); // Dibuja la cuadrícula actualizada
90
91          // Guardamos el estado de la generación en el historial
92          saveToHistory(); // Guardamos el estado del juego en el historial
93          updateStatistics(); // Actualiza las estadísticas
94      }
95
96      // Evento para el toggle del modo toroidal
97      document.querySelector("input[type='checkbox']").addEventListener("change",
        ↪  function () {
98          // Pausamos el juego cuando se cambia el estado del toggle
99          // if (running) {
100         //     stopGame();
101         // }
102
103         const aliveColor = celdaVivaColorInput.value;  // Color de las celdas vivas
104
```

```
105         // Activamos o desactivamos el modo toroidal
106         isToroidal = this.checked;
107 
108         // Actualizamos el borde del canvas dependiendo del estado del modo toroidal
109         const canvas = document.getElementById("gameCanvas");
110         if (isToroidal) {
111             canvas.style.border = "1px solid " + aliveColor; // Borde de 1px cuando el
                ↪  modo toroidal est activado
112         } else {
113             canvas.style.border = "2px solid " + aliveColor; // Borde de 3px cuando el
                ↪  modo toroidal est desactivado
114         }
115 
116         // Mantenemos las celdas como estaban antes de cambiar el estado
117         drawGrid(); // Dibuja la cuadrícula con las celdas tal como están
118     });
119 
120     // Ejecuta la actualización del juego en cada iteración
121     function update() {
122         getNextGeneration(); // Calcula la siguiente generación
123         saveToHistory(); // Guardamos el estado de la generación en el historial
124         drawGrid(); // Dibuja la cuadrícula
125 
126 
127         // Actualiza el contador de celdas vivas
128         const aliveCount = countAliveCells(); // Llama a la funcin que cuenta las
            ↪  celdas vivas
129         document.getElementById("aliveCounter").innerText = aliveCount; // Muestra el
            ↪  contador de celdas vivas en la interfaz
130         updateStatistics(aliveCount);
131     }
132 
133     // Actualizamos las reglas B/S desde los inputs
134     function updateRules() {
135         const ruleBValue = document.getElementById("ruleB").value;
136         const ruleSValue = document.getElementById("ruleS").value;
137 
138         // Convertir los valores de entrada a arrays de números
139         ruleB = ruleBValue.split("").map(Number); // Convierte B (Ejemplo: 3 -> [3])
140         ruleS = ruleSValue.split("").map(Number); // Convierte S (Ejemplo: 23 -> [2,
            ↪  3])
141 
142         console.log("B" + ruleB + "/S" + ruleS);
143     }
144 
145     // Evento para cuando se cambian las reglas
146     document.getElementById("ruleB").addEventListener("input", function () {
147         if (!running) updateRules(); // Solo actualizamos si el juego está pausado
148     });
149 
150     document.getElementById("ruleS").addEventListener("input", function () {
151         if (!running) updateRules(); // Solo actualizamos si el juego está pausado
152     });
```

3.2.7 Funcionalidades de Generaciones:

En esta sección se gestionan las funcionalidades para retroceder y avanzar entre generaciones, así como la gestión del historial del juego.

La función previousGeneration() permite retroceder una generación del historial. Si hay generaciones anteriores en el historial, se actualiza el índice del historial y se restaura el estado de la cuadrícula, el contador de generaciones y el total de celdas vivas. Posteriormente, se recalculan las celdas vivas y se actualiza la interfaz con la nueva generación. Finalmente, se pausa el juego al retroceder.

La función nextGeneration() permite avanzar una generación en el historial. Si ya existen generaciones futuras en el historial, se avanza al siguiente estado del juego, se restauran los valores correspondientes y se redibuja la cuadrícula. Si no hay generaciones futuras en el historial, la función calcula una nueva generación y la guarda en el historial, recalculando también las celdas vivas y actualizando las estadísticas. Al igual que en la función anterior, el juego se pausa al avanzar.

La función saveToHistory() guarda el estado actual del juego en el historial, creando una copia profunda de la cuadrícula junto con el contador de generaciones y el total de celdas vivas. El historial se limita a un máximo de 10 generaciones, eliminando las generaciones más antiguas cuando se supera este límite. Esto permite gestionar el retroceso y avance de generaciones de manera eficiente, manteniendo un historial controlado y actualizado.

```
// ---------- FUNCIONALIDADES DE GENERACIONES ----------

    // Retroceder una generación
    function previousGeneration() {
        if (historyIndex > 0) {
            historyIndex--; // Retrocedemos en el historial
            const previousState = history[historyIndex]; // Obtenemos el estado
            ↪  anterior

            // Restauramos la cuadrícula, las generaciones y el totalAliveCells
            grid = JSON.parse(JSON.stringify(previousState.grid));
            generationCount = previousState.generationCount;
            totalAliveCells = previousState.totalAliveCells; // Restauramos el total
            ↪  de celdas vivas

            document.getElementById("generationCounter").innerText = generationCount;
            document.getElementById("aliveCounter").innerText = countAliveCells(); //
            ↪  Calculamos de nuevo las celdas vivas
            drawGrid(); // Redibujamos el canvas

            // Actualizamos las estadísticas
            updateStatistics();
            stopGame(); // Pausa el juego al retroceder
        }
    }

    // Avanzar una generación
    function nextGeneration() {
        if (historyIndex < history.length - 1) {
            historyIndex++; // Avanzamos en el historial
            const nextState = history[historyIndex]; // Obtenemos el siguiente estado

            // Restauramos la cuadrícula, las generaciones y el totalAliveCells
            grid = JSON.parse(JSON.stringify(nextState.grid));
            generationCount = nextState.generationCount;
            totalAliveCells = nextState.totalAliveCells; // Restauramos el total de
            ↪  celdas vivas

            document.getElementById("generationCounter").innerText = generationCount;
            document.getElementById("aliveCounter").innerText = countAliveCells(); //
            ↪  Calculamos de nuevo las celdas vivas
            drawGrid(); // Redibujamos el canvas
```

```
38
39              // Actualizamos las estadísticas
40              updateStatistics();
41              stopGame(); // Pausa el juego al avanzar
42          } else {
43              // Si no hay generaciones futuras, calculamos una nueva
44              getNextGeneration();
45              saveToHistory();
46              drawGrid();
47
48              // Actualizamos el contador de celdas vivas
49              const aliveCount = countAliveCells();
50              document.getElementById("aliveCounter").innerText = aliveCount;
51              updateStatistics();
52              stopGame(); // Pausa el juego al avanzar
53          }
54      }
55
56      // Función para guardar el estado de la generación en el historial
57      function saveToHistory() {
58          if (historyIndex < history.length - 1) {
59              // Si estamos en medio del historial, eliminamos las generaciones
                ↪  "futuras"
60              history = history.slice(0, historyIndex + 1);
61          }
62
63          // Guardamos el estado completo (cuadrícula, generaciones y totalAliveCells)
64          history.push({
65              grid: JSON.parse(JSON.stringify(grid)),
66              generationCount: generationCount,
67              totalAliveCells: totalAliveCells
68          });
69
70          historyIndex++; // Incrementamos el índice del historial
71
72          // Limitar el historial a las últimas 10 generaciones
73          if (history.length > maxGenerations) {
74              history.shift(); // Elimina la generacin ms antigua (el primer elemento
                ↪  del array)
75              historyIndex--; // Reducimos el índice para reflejar el cambio
76          }
77      }
```

3.2.8 Control de Velocidades:

La función changeSpeed() permite ajustar la velocidad de actualización del juego, asegurando que se mantenga dentro de un rango válido entre 50 y 500 milisegundos. Si se intenta establecer un valor fuera de este rango, la función lo ajusta automáticamente para que se quede dentro de los límites definidos. Cuando el juego está en ejecución, la función detiene el intervalo actual usando clearInterval() y lo reinicia con el nuevo valor de velocidad, garantizando que la simulación se actualice con la nueva velocidad seleccionada por el usuario. Además, se actualiza el valor del input de velocidad en la interfaz para reflejar el cambio.

```
1   // ---------- CONTROL DE VELOCIDADES ----------
2
3       // Cambia la velocidad asegurando que esté entre 50 y 500 ms
4       function changeSpeed(newSpeed) {
5           speed = Math.max(50, Math.min(500, newSpeed)); // Limita la velocidad
6           document.getElementById("speedInput").value = speed;
```

```
7
8           if (running) {
9               clearInterval(intervalId);
10              intervalId = setInterval(update, speed);
11          }
12      }
```

Estadísticas:
En esta sección se gestionan las funciones que calculan y actualizan las estadísticas relacionadas con el estado del juego, específicamente el número de celdas vivas y diversas métricas derivadas de ese valor.

La función countAliveCells() recorre la cuadrícula para contar cuántas celdas están vivas, incrementando el contador cada vez que se encuentra una celda activa (con valor 1). Devuelve el número total de celdas vivas.

La función calculatePopulationDensity() calcula la densidad poblacional dividiendo el número de celdas vivas por el total de celdas disponibles en la cuadrícula. Esto da una indicación de cuán "poblada" está la cuadrícula en un momento determinado.

La función calculateLogBase10() calcula el logaritmo en base 10 del número de celdas vivas. Si no hay celdas vivas (es decir, si el conteo es 0), devuelve 0, evitando errores al calcular logaritmos de 0.

La función calculateMeanAliveCells() calcula la media de celdas vivas por generación. Esto se obtiene dividiendo el total de celdas vivas acumuladas entre el número total de generaciones.

La función calculateVariance() calcula la varianza de las celdas vivas a lo largo de las generaciones. Compara el número de celdas vivas en cada generación con la media y calcula la diferencia al cuadrado, luego promedia esas diferencias para obtener la varianza.

Finalmente, la función updateStatistics() actualiza todas las métricas en la interfaz de usuario. Llama a las funciones anteriores para obtener el número actual de celdas vivas, la densidad poblacional, el logaritmo base 10, el total acumulado de celdas vivas, la media y la varianza, y muestra estos valores en la interfaz.

```
1   // ---------- ESTADÍSTICAS ----------
2
3       // Función para contar las celdas vivas
4       function countAliveCells() {
5           let aliveCount = 0;
6           for (let y = 0; y < rows; y++) {
7               for (let x = 0; x < cols; x++) {
8                   if (grid[y][x] === 1) {
9                       aliveCount++; // Incrementa el contador si la celda está viva
10                  }
11              }
12          }
13
14          return aliveCount; // Devuelve el número total de celdas vivas
15      }
16
17      // Función para calcular la densidad poblacional
18      function calculatePopulationDensity(aliveCount) {
19          const totalCells = rows * cols;
20          return (aliveCount / totalCells);
21      }
22
23      // Función para calcular el logaritmo base 10 del número de celdas vivas
24      function calculateLogBase10(aliveCount) {
25          if (aliveCount === 0) return 0;
```

```
        return Math.log10(aliveCount); // Retorna 0 si aliveCount es 0
    }

    // Función para calcular la media de celdas vivas
    function calculateMeanAliveCells() {
        if (generationCount === 0) return 0;
        return totalAliveCells / generationCount;
    }

    // Función para calcular la varianza
    function calculateVariance(aliveCount) {
        const mean = calculateMeanAliveCells();
        const squaredDifferences = generationCount.map(count => Math.pow(count -
        ↪  mean, 2));
        const variance = squaredDifferences.reduce((acc, diff) => acc + diff, 0) /
        ↪  squaredDifferences.length;
        return variance;
    }

    // Función para mostrar las estadísticas actualizadas en la interfaz
    function updateStatistics() {
        const aliveCount = countAliveCells();
        totalAliveCells += aliveCount;
        document.getElementById("populationDensity").innerText =
        ↪  calculatePopulationDensity(aliveCount).toFixed(4);
        document.getElementById("logBase10").innerText =
        ↪  calculateLogBase10(aliveCount).toFixed(4);
        document.getElementById("totalAliveCells").innerText = totalAliveCells;
        document.getElementById("meanAliveCells").innerText =
        ↪  calculateMeanAliveCells().toFixed(4);
        document.getElementById("variance").innerText =
        ↪  calculateVariance().toFixed(4);
    }
```

3.2.9 Personalización de colores:

La función updateCellColors() permite personalizar los colores de las celdas vivas y muertas en el juego. Primero, obtiene los valores de color de los inputs correspondientes a las celdas vivas y muertas. Luego, actualiza el borde del canvas a 1px de grosor, utilizando el color de las celdas vivas, siempre que el modo toroidal esté activado. Finalmente, llama a la función drawGrid() para redibujar la cuadrícula con los nuevos colores establecidos, asegurando que los cambios se reflejen inmediatamente en la interfaz del usuario.

```
// ---------- PERSONALIZACIÓN DE COLORES ----------

    // Función para actualizar los colores de las celdas inmediatamente
    function updateCellColors() {
        const aliveColor = celdaVivaColorInput.value;  // Color de las celdas vivas
        const deadColor = celdaMuertaColorInput.value; // Color de las celdas muertas

        canvas.style.border = "1px solid " + aliveColor; // Borde de 1px cuando el
        ↪  modo toroidal est activado

        // Vuelve a dibujar la cuadrícula con los nuevos colores
        drawGrid(aliveColor, deadColor);
    }
```

3.2.10 Exportación e Importación de Archivos:

La función exportCanvas() permite al usuario exportar el estado del juego (incluyendo la cuadrícula, el contador de generaciones, el número de celdas vivas, las reglas de B/S, el modo toroidal, los colores y el tamaño de las celdas) a un archivo JSON. El nombre del archivo puede ser proporcionado por el usuario a través de un cuadro de texto emergente (con un valor predeterminado en caso de no ser ingresado). Luego, el archivo es generado como un objeto Blob y el usuario puede descargarlo.

La función importCanvas() permite importar un archivo JSON previamente guardado. Al seleccionar un archivo, los datos son leídos y convertidos a un objeto, lo que permite restaurar el estado del juego a partir de esos datos. Esto incluye la cuadrícula, las generaciones, el número de celdas vivas, las reglas de B/S, y el modo toroidal, y se actualiza la interfaz con la información recuperada. Además, se asegura que el borde del canvas se actualice de acuerdo con el estado toroidal (1px o 3px). Finalmente, el canvas se redibuja para reflejar el estado restaurado.

Se utiliza un input de archivo oculto que se dispara al hacer clic en un botón de "Importar", lo que permite seleccionar el archivo desde el explorador de archivos.

```
// ---------- EXPORTACIÓN E IMPORTACIÓN DE ARCHIVOS ----------

    // Exporta el estado del canvas a un archivo JSON
    function exportCanvas() {
        const data = {
            grid: grid, // El estado actual de la cuadrícula
            generationCount: generationCount, // Contador de generaciones
            aliveCount: countAliveCells(), // Número de celdas vivas
            totalAliveCells: totalAliveCells,
            isToroidal: isToroidal,
            ruleB: ruleB, // Reglas de nacimiento
            ruleS: ruleS, // Reglas de supervivencia
            rows: rows, // Número de filas
            cols: cols, // Número de columnas
            cellSize: cellSize, // Tamaño de las celdas
            aliveColor: celdaVivaColorInput.value, // Color de las celdas vivas
            deadColor: celdaMuertaColorInput.value, // Color de las celdas muerta
        };

        // Solicitar al usuario el nombre del archivo
        const fileName = prompt("Cmo quieres nombrar el archivo?",
        ↪  "conway_grid.json");

        // Si el usuario no introduce un nombre, usamos un valor por defecto
        const name = fileName ? fileName : "conway_grid.json";

        // Crear un objeto Blob con el contenido de los datos
        const blob = new Blob([JSON.stringify(data, null, 2)], { type:
        ↪  'application/json' });

        // Crear un enlace para descargar el archivo
        const link = document.createElement('a');
        link.href = URL.createObjectURL(blob);
        link.download = name; // Usar el nombre proporcionado por el usuario
        link.click(); // Descargar el archivo
    }

    // Función para importar el archivo de datos JSON
    function importCanvas(event) {
        const file = event.target.files[0]; // Obtener el archivo seleccionado

```

```
        if (!file) return;

        const reader = new FileReader();

        // Leer el contenido del archivo como texto
        reader.onload = function (e) {
            const data = JSON.parse(e.target.result); // Convertir el JSON a un objeto

            // Establecer el estado de la cuadrcula, el contador de generaciones y el
            ↪ nmero de celdas vivas
            grid = data.grid;
            generationCount = data.generationCount;
            const aliveCount = data.aliveCount; // Celdas vivas recuperadas del
            ↪ archivo
            totalAliveCells = data.totalAliveCells;
            ruleB = data.ruleB;
            ruleS = data.ruleS;

            // Actualizar la interfaz con los nuevos valores
            document.getElementById("generationCounter").innerText = generationCount;
            document.getElementById("aliveCounter").innerText = aliveCount;
            document.getElementById("totalAliveCells").innerText = totalAliveCells;
            document.getElementById("ruleB").innerText = ruleB;
            document.getElementById("ruleS").innerText = ruleS;
            celdaMuertaColorInput.value = data.deadColor;
            celdaVivaColorInput.value = data.aliveColor;

            // Aseguramos que el borde se actualice al estado toroidal
            const canvas = document.getElementById("gameCanvas");
            if (data.isToroidal) {
                canvas.style.border = "1px solid " + data.aliveColor; // Borde de 1px
                ↪ cuando el modo toroidal est activado
            } else {
                canvas.style.border = "3px solid " + data.aliveColor; // Borde de 3px
                ↪ cuando el modo toroidal est desactivado
            }

            drawGrid(); // Redibujar el canvas con los nuevos datos
        };

        reader.readAsText(file); // Leer el archivo como texto
    }

    // Función para abrir el explorador de archivos al hacer clic en el botón Importar
    document.getElementById('importBtn').addEventListener('click', function () {
        document.getElementById('importFile').click(); // Disparar el clic en el input
        ↪ de archivo
    });
```

3.2.11 Event Listeners:

Los Event Listeners gestionan las interacciones del usuario con los controles del juego. Permiten iniciar o pausar el juego, reiniciar la cuadrícula, generar patrones aleatorios, añadir celdas vivas, retroceder o avanzar generaciones, así como exportar e importar el estado del juego. También permiten ajustar la velocidad del juego y cambiar los colores de las celdas vivas y muertas. Las funciones asociadas a cada evento ejecutan las acciones correspondientes para controlar y modificar el comportamiento del juego.

```
// ---------- EVENT LISTENERS ----------

```

```
    // Botones
    toggleGameButton.addEventListener("click", toggleGame);
    updateSizeBtn.addEventListener("click", updateCanvasSize);
    document.getElementById("resetBtn").addEventListener("click", resetGame);
    document.getElementById("generateRandomBtn").addEventListener("click",
    ↪ generateRandomGrid);
    document.getElementById("addRandomBtn").addEventListener("click", addRandomCells);
    document.getElementById("previousGenerationBtn").addEventListener("click",
    ↪ previousGeneration);
    document.getElementById("nextGenerationBtn").addEventListener("click",
    ↪ nextGeneration);
    document.getElementById('exportBtn').addEventListener('click', exportCanvas);
    document.getElementById('importFile').addEventListener('change', importCanvas);
    document.getElementById("increaseSpeed").addEventListener("click", () =>
    ↪ changeSpeed(speed + 50));
    document.getElementById("decreaseSpeed").addEventListener("click", () =>
    ↪ changeSpeed(speed - 50));
    document.getElementById("minSpeed").addEventListener("click", () =>
    ↪ changeSpeed(50));
    document.getElementById("maxSpeed").addEventListener("click", () =>
    ↪ changeSpeed(500));

    // Inputs de color
    celdaVivaColorInput.addEventListener("input", updateCellColors);
    celdaMuertaColorInput.addEventListener("input", updateCellColors);
```

3.2.12 Dibujo interactivo del Canvas:

La sección de Dibujo Interactivo del Canvas permite que el usuario interactúe con el lienzo (canvas) para modificar las celdas del juego. Al hacer clic en el canvas, se activa el modo de dibujo, pausando automáticamente el juego si está en ejecución. Al mover el mouse sobre el canvas mientras dibuja, las celdas seleccionadas se activan o desactivan según el estado de la celda en la cuadrícula. La cantidad de celdas vivas se actualiza constantemente y se muestra en la interfaz. Al soltar el mouse, el modo de dibujo se desactiva y, si el juego estaba pausado, se reanuda.

```
// ---------- DIBUJO INTERACTIVO DEL CANVAS ----------

    canvas.addEventListener("mousedown", function (event) {
        if (running) {
            wasRunning = true;
            toggleGame(); // Pausa el juego automáticamente
        } else {
            wasRunning = false;
        }

        isDrawing = true;
        const rect = canvas.getBoundingClientRect();
        const x = Math.floor((event.clientX - rect.left) / cellSize);
        const y = Math.floor((event.clientY - rect.top) / cellSize);

        drawState = grid[y][x] === 1 ? 0 : 1;
        grid[y][x] = drawState;
        drawGrid();

        // Actualizamos el contador de celdas vivas
        const aliveCount = countAliveCells();
        document.getElementById("aliveCounter").innerText = aliveCount; // Muestra el
        ↪ contador de celdas vivas
    });
```

```
24
25      canvas.addEventListener("mousemove", function (event) {
26          if (!isDrawing) return;
27          const rect = canvas.getBoundingClientRect();
28          const x = Math.floor((event.clientX - rect.left) / cellSize);
29          const y = Math.floor((event.clientY - rect.top) / cellSize);
30
31          grid[y][x] = drawState;
32          drawGrid();
33
34          // Actualizamos el contador de celdas vivas
35          const aliveCount = countAliveCells();
36          document.getElementById("aliveCounter").innerText = aliveCount; // Muestra el
            ↪  contador de celdas vivas
37      });
38
39      canvas.addEventListener("mouseup", function () {
40          isDrawing = false;
41          if (wasRunning) {
42              toggleGame();
43          }
44      });
```

3.2.13 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.

```
1  window.onload = function () {
2      // ----- Código anterior en el orden presentado -----
3
4      // ---------- DUBUJAR CUADRÍCULA ----------
5      drawGrid();
6  }
```

4 ¿Dónde encontrar el código?

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

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

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

- https://juego-de-la-vida-conway.vercel.app/

5 Conclusión

Este proyecto es una implementación del Juego de la Vida de Conway, un autómata celular que simula cómo una población de células evoluciona con base en reglas simples de nacimiento y supervivencia. Es útil para entender fenómenos complejos y emergentes, con aplicaciones en biología, computación y teoría de sistemas dinámicos.

Durante el desarrollo, aprendí a crear una simulación interactiva utilizando HTML, CSS y JavaScript, implementando funcionalidades como:

- Modificación de reglas de nacimiento y supervivencia (B/S).
- Generación aleatoria de patrones y adición de células aleatorias.
- Exportación e importación de estados para guardar y cargar configuraciones.
- Cálculo y visualización en tiempo real de estadísticas como densidad poblacional y celdas vivas.

Aunque no haya cumplido con todas las especificaciones, este proyecto fue una excelente oportunidad para aprender sobre la manipulación de canvas en JavaScript, gestión de estados complejos, y cómo hacer que un juego interactivo y visualmente atractivo sea funcional y fácil de usar. A través de este desarrollo, mejoré mis habilidades en JavaScript, diseño de interfaces, lógica de programación y manejo de datos. Fue una experiencia que me permitió aplicar y consolidar muchos conceptos, al tiempo que creaba algo que no solo era funcional, sino también visualmente interesante y educativo.