Instituto Politécnico Nacional

Escuela Superior de Cómputo

Programa 1: Autómatas en dos dimensiones tipo Life

Vega Gloria Carlos Raymundo

Grupo: 3CV12

Asignatura: CST - Complex Systems

Docente: Juárez Martínez Genaro

Fecha de entrega: 24 de abril 2022

Índice general

Capítulo 1

Introducción

El Juego de la vida es un autómata celular diseñado por el matemático británico John Horton Conway en 1970. Es un juego de cero jugadores, en el que su evolución es determinada por un estado inicial, sin requerir intervención adicional. Se considera un sistema Turing completo que puede simular cualquier otra Máquina de Turing.

Desde su publicación, ha atraído mucho interés debido a la gran variabilidad de la evolución de los patrones. Se considera que el Juego de la vida es un buen ejemplo de emergencia y autoorganización. Es interesante para científicos, matemáticos, economistas y otros observar cómo patrones complejos pueden provenir de la implementación de reglas muy sencillas.

1.1. Checklist

El programa debe simular autómatas celulares en dos dimensiones tipo Life y debe de tener las siguientes características.

- ☑ Evaluar espacios de 300x300
- ☑ Evaluar espacios de 500x500
- ☒ Evaluar espacios de 10000x10000
- ☑ Animación en dos dimensiones.
- ☑ Poder cambiar los colores de los estados.
- ☑ Poder inicializar el espacio de evoluciones con diferentes densidades.
- ☑ Poder editar el espacio de evoluciones para dibujar configuraciones particulares.
- ☒ Poder salvar y levantar archivos con configuraciones en específico.
- ☑ Graficar el número de unos de cada generación (densidad).
- ☑ Graficar el número de unos de cada generación (densidad logaritmo base 10).
- ☒ Graficar la curva acerca de la entropía.
- ☑ Especificar las reglas de evolución con la siguiente notación: R(S_min,S_max,B_min,B_max).

Capítulo 2

Códigos

2.1. HTML

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>The Game Of Life</title>
    <link rel="stylesheet" href="style.css" />
    <script type="module" src="JS/gol-config.js"></script>
    <!-- Grafica -->
    <script src="https://cdn.anychart.com/releases/8.11.0/js/anychart-base.min.js"></script>
  </head>
  <body>
    <div id="container">
      <main id="canvas"></main>

      <section id="parameters-section">
        <fieldset id="parameters-values">
          <legend>Parámetros</legend>
          <div id="inputs">
            <div class="inputs">
              <label for="canvasSize">Tamaño del canvas:</label>
              <input
                id="canvasSize"
                name="canvasSize"
                type="number"
                min="400"
                max="1500"
              />
            </div>
            <div class="inputs">
              <label for="cellSize">Tamaño de célula:</label>
              <input
                id="cellSize"
                name="cellSize"
                type="number"
                min="0"
              max="9"
            />
          </div>
          <div class="inputs">
            <label> Delay (ms): </label>
            <input
              id="frame-rate"
              type="range"
              value="20"
              min="0"
              max="50"
              step="10"
            />
          </div>
          <div>
            <label> Presets: </label
            ><select id="presets"></select>
            <button id="load-preset-button">Cargar preset</button>
          </div>
          <div class="inputs">
            <label for="percent-life-reset">Probabilidad de vida:</label>
            <input
              id="percent-life-reset"
              name="percent-life-reset"
              type="number"
              value="0"
              min="0"
              max="1"
              step="0.01"
            />
          </div>
        </div>
        <div id="section-rules">
          <div class="config">
            <label for="S_min">Muerte por soledad (S_min):</label>
            <input id="S_min" name="S_min" type="number" min="0" max="9" />
          </div>
          <div class="config">
            <label for="S_max">Muerte por sobrepoblación (S_max):</label>
            <input id="S_max" name="S_max" type="number" min="0" max="9" />
          </div>
          <div class="config">
            <label for="B_min">Vecinos mínimos requeridos (B_min):</label>
            <input id="B_min" name="B_min" type="number" min="0" max="9" />
          </div>
          <div class="config">
            <label for="B_max">Vecinos máximos requeridos (B_max):</label>
            <input id="B_max" name="B_max" type="number" min="0" max="9" />
          </div>
        </div>
      </fieldset>
```

```html

        <fieldset>
          <legend>Datos</legend>
          <label class="labels">
            Generaciones:
            <div class="value" id="generations"></div>
          </label>
          <label class="labels">
            Células vivas:
            <div class="value" id="population"></div>
          </label>
        </fieldset>

        <fieldset class="controls">
          <legend>Controles</legend>
          <button id="update-rules-button">Update Rules</button>

          <button id="pause-play-button">Pause/Play</button>
          <button id="reset-life-button">Reset game</button>
          <button id="next-generation">Next Generation</button>

          <div id="colors">
            <div>
              <label for="deathStyle">Célula muerta</label>
              <input type="color" id="deathStyle" name="head" value="#ffffff" />
            </div>
            <div>
              <label for="lifeStyle">Célula viva</label>
              <input type="color" id="lifeStyle" name="head" value="#000000" />
            </div>
            <button id="btnColors">Cambiar colores</button>
          </div>
        </fieldset>

        <fieldset>
          <legend>Gráficas</legend>

          <!-- Chart -->
          <div id="chart">
            <div id="graph"></div>
          </div>
        </fieldset>
      </section>
    </div>
  </body>
</html>
```

2.2. JavaScript - Cell

```
export default class Cell {
  constructor(alive) {
    this.alive = alive;
    this.lifeColor = "#000000";
    this.deathColor = "#ffffff";
    this.S_min = 2;
    this.S_max = 3;
    this.B_min = 3;
    this.B_max = 3;

    this.neighbors = [];
    this.nextState = null;
    this.previousState = null;
    this.forceRepaint = true;

    if (this.B_max < this.B_min) {
      this.B_min = this.B_max;
    }
  } // fin del constructor

  prepareUpdate() {
    let sum = 0;
    let nextState = this.alive;

    /* Contamos los vecinos vivos excluyéndome */
    for (let n of this.neighbors) {
      if (n.alive && n !== this) sum++;
    }

    /* Reglas */
    if (nextState && (sum < this.S_min || sum > this.S_max)) {
      nextState = false;
    } else if (!nextState && sum >= this.B_min && sum <= this.B_max) {
      nextState = true;
    }

    this.nextState = nextState;
  }

  update() {
    this.previousState = this.alive;
    this.alive = this.nextState;
    this.nextState = null;
  }

  handleClick() {
    this.alive = !this.alive;
  }

  getLifeStyle() {
    return this.lifeColor;
  }

  setLifeStyle(color) {
    this.lifeColor = color;
  }

  getDeathStyle() {
    return this.deathColor;
  }

  setDeathStyle(color) {
    this.deathColor = color;
  }

  setPaintStyles(canvasCtx) {
    canvasCtx.fillStyle = this.alive ? this.lifeColor : this.deathColor;
  }
}
```

2.3. JavaScript - GOL-Interaction-DOM

```
import GOL from "./gol.js";
import { loadPresets } from "./presets.js";

let CURRENT_SIM = null;
let preset = null;

document.addEventListener("DOMContentLoaded", function () {
  const roundDelay = 20;
  const pixelSize = 4;
  const canvasSize = 800;
  const rules = [2, 3, 3, 3];

  resetSimulation(pixelSize, canvasSize, rules, roundDelay, 0.05);
  setupEventListeners(canvasSize, pixelSize, rules, 0.05, roundDelay);
  loadPresets();
});

function resetSimulation(
  pixelSize,
  canvasSize,
  rules,
  roundDelay,
  initialChanceOfLife = 0.05
) {
  const containerCanvas = document.getElementById("canvas");
  const previousCanvas = containerCanvas.querySelector("canvas");

  const chart = document.querySelector("#chart");
  const previousGraph = document.querySelector("#graph");

  if (previousCanvas) containerCanvas.removeChild(previousCanvas);

  if (previousGraph) {
    chart.removeChild(previousGraph);
    const newGraph = document.createElement("div");
    newGraph.setAttribute("id", "graph");
    newGraph.style.width = "100%";
    newGraph.style.height = "50%";
    chart.appendChild(newGraph);
  }

  //const canvasSize = 800;
  const cols = parseInt(canvasSize / pixelSize);
  const rows = parseInt(canvasSize / pixelSize);

  CURRENT_SIM = new GOL(rows, cols, pixelSize, roundDelay, initialChanceOfLife);

  CURRENT_SIM.setRules(...rules);
  CURRENT_SIM.canvas.style.height = canvasSize + "px";
  CURRENT_SIM.canvas.style.width = canvasSize + "px";
  containerCanvas.append(CURRENT_SIM.canvas);
  CURRENT_SIM.repaint();
  CURRENT_SIM.start();

  window.CURRENT_SIM = CURRENT_SIM;
}

function setupEventListeners(
  initialCanvasSize,
  initialCellSize,
  initialRules,
  initialChanceOfLife,
  initialRoundDelay
) {
  const rulesForm = document.querySelector("#parameters-section");

  rulesForm.querySelector("#canvasSize").value = initialCanvasSize;
  rulesForm.querySelector("#cellSize").value = initialCellSize;
  rulesForm.querySelector("#S_min").value = initialRules[0];
  rulesForm.querySelector("#S_max").value = initialRules[1];
  rulesForm.querySelector("#B_min").value = initialRules[2];
  rulesForm.querySelector("#B_max").value = initialRules[3];
  rulesForm.querySelector("#percent-life-reset").value = initialChanceOfLife;
  rulesForm.querySelector("#frame-rate").value = initialRoundDelay;

  const pause = () => {
    if (CURRENT_SIM.paused) {
      CURRENT_SIM.start();
    } else {
      CURRENT_SIM.stop();
    }

    CURRENT_SIM.paused = !CURRENT_SIM.paused;
  };

  window.addEventListener("keydown", (e) => {
    if (e.which === 90) {
      pause();
    }
  });

  document.querySelector("#presets").addEventListener("change", (e) => {
    preset = e.target.value;
  });

  document
    .querySelector("#update-rules-button")
    .addEventListener("click", (e) => {
      let rules = [
        parseInt(rulesForm.querySelector("#S_min").value, 10),
        parseInt(rulesForm.querySelector("#S_max").value, 10),
        parseInt(rulesForm.querySelector("#B_min").value, 10),
        parseInt(rulesForm.querySelector("#B_max").value, 10),
      ];

      CURRENT_SIM.setRules(...rules);
    });

  document
    .querySelector("#load-preset-button")
    .addEventListener("click", (e) => {
      CURRENT_SIM.resetLife(preset, 0.0);
    });

  document
    .querySelector("#pause-play-button")
    .addEventListener("click", (e) => {
      pause();
    });

  document.querySelector("#next-generation").addEventListener("click", (e) => {
    CURRENT_SIM.advanceRound();
    CURRENT_SIM.repaint(true);
  });

  document
    .querySelector("#reset-life-button")
    .addEventListener("click", (e) => {
      const canvasSize = rulesForm.querySelector("#canvasSize").value;
      const cellSize = rulesForm.querySelector("#cellSize").value;
      const rules = [];
      rules.push(rulesForm.querySelector("#S_min").value);
      rules.push(rulesForm.querySelector("#S_max").value);
      rules.push(rulesForm.querySelector("#B_min").value);
      rules.push(rulesForm.querySelector("#B_max").value);
      const chanceOfLife = rulesForm.querySelector("#percent-life-reset").value;
      const roundDelay = rulesForm.querySelector("#frame-rate").value;

      resetSimulation(cellSize, canvasSize, rules, roundDelay, chanceOfLife);
    });

  document.querySelector("#frame-rate").addEventListener("change", (e) => {
    CURRENT_SIM.stop();
    CURRENT_SIM.interRoundDelay = Math.floor(Math.pow(e.target.value, 1.3));
    CURRENT_SIM.start();
  });

  document.querySelector("#btnColors").addEventListener("click", (e) => {
    const newLifeColor = document.querySelector("#lifeStyle").value;
    const newDeathColor = document.querySelector("#deathStyle").value;

    CURRENT_SIM.stop();
    CURRENT_SIM.setPixelColors(newLifeColor, newDeathColor);
    CURRENT_SIM.start();
  });
}
```

2.4. JavaScript - GOL-Logic

```
import Cell from "./Cell.js";
import { presets } from "./presets.js";
import { Chart } from "./chartConfig.js";
import { bindMultipleEventListener } from "./utilities.js";

export default class GOL {
  constructor(rows, cols, pixelSize, interRoundDelay, initialChanceOfLife) {
    this.rows = rows;
    this.cols = cols;
    this.pixelSize = pixelSize;
    this.interRoundDelay = interRoundDelay;
    this.mouseIsDown = false;
    this.paused = false;
    this.intervalId = 1;
    this.generations = 0;
    this.population = 0;
    /**
```

```
     * * Coordenadas de los vecinos respecto de cada célula
     */
    this.adjacentCells = [
      [-1, -1],
      [0, -1],
      [1, -1],
      [-1, 0],
      //[0, 0],
      [1, 0],
      [-1, 1],
      [0, 1],
      [1, 1],
    ];

    this.grid = [];
    for (let i = 0; i < rows; i++) {
      this.grid.push([]);
      for (let j = 0; j < cols; j++) {
        let alive = Math.random() < initialChanceOfLife;
        this.grid[i].push(new Cell(alive));
      }
    }

    /**
     * * Le asignamos los vecinos a cada celda para optimizar los calculos
     */
    for (let i = 0; i < this.rows; i++) {
      for (let j = 0; j < this.cols; j++) {
        this.grid[i][j].neighbors = this.getNeighbors(i, j);
      }
    }

    // Configuración del canvas
    let width = this.pixelSize * this.cols;
    let height = this.pixelSize * this.rows;
    this.canvas = document.createElement("canvas");
    this.canvas.width = width;
    this.canvas.height = height;
    this.canvasCtx = this.canvas.getContext("2d", { alpha: false });

    this.registerMouseListeners();

    // Para la gráfica
    this.chart = new Chart("graph", "Gráfica de densidades");
    this.chart2 = new Chart("graph", "Gráfica de densidades (log10)");
  } // fin del constructor

  start() {
    if (this.intervalId) {
      return;
    }

    this.intervalId = setInterval(() => {
      this.advanceRound();
      this.repaint();
    }, this.interRoundDelay);
  }

  stop() {
    if (this.intervalId) {
      clearInterval(this.intervalId);
      this.intervalId = null;
    }
  }

  getNeighbors(row, col) {
    let neighbors = [];

    /**
     * * Mundo con bordes muertos
     */
    /* for (let i = row - 1; i <= row + 1; i++) {
      for (let j = col - 1; j <= col + 1; j++) {
        if (i === row && j === col) continue;
        if (this.grid[i] && this.grid[i][j]) {
          neighbors.push(this.grid[i][j]);
        }
      }
    } */

    /**
     * * Mundo toroidal
     */
    for (const pair of this.adjacentCells) {
      const xCoord = (row + pair[0] + this.rows) % this.rows;
      const yCoord = (col + pair[1] + this.rows) % this.rows;

      neighbors.push(this.grid[xCoord][yCoord]);
    }

    return neighbors;
  }

  advanceRound() {
    if (this.mouseIsDown) return;

    for (let i = 0; i < this.rows; i++) {
      for (let j = 0; j < this.cols; j++) {
        this.grid[i][j].prepareUpdate();
      }
    }

    for (let i = 0; i < this.rows; i++) {
      for (let j = 0; j < this.cols; j++) {
        this.grid[i][j].update();
      }
    }

    this.generations++;
    this.population = this.grid
      .flat()
      .filter((cell) => cell.alive === true).length;
    /* Actualizamos las gráficas en cada ronda/generación */
    this.chart.updateChart(this.generations, this.population);
    this.chart2.updateChart(this.generations, Math.log10(this.population));

    document.querySelector("#generations").innerHTML = this.generations;
    document.querySelector("#population").innerHTML = this.population;
  }

  repaint(force = false) {
    if (this.mouseIsDown && !force) return;

    let byColor = {};
    for (let i = 0; i < this.rows; i++) {
      for (let j = 0; j < this.cols; j++) {
        let pixel = this.grid[i][j];

        if (
          !force &&
          !pixel.forceRepaint &&
          pixel.alive === pixel.previousState
        ) {
          continue; // No se repinta si no cambió su estado
        }

        let color = pixel.alive ? pixel.getLifeStyle() : pixel.getDeathStyle();
        if (byColor[color] === undefined) {
          byColor[color] = [];
        }

        byColor[color].push([i, j]);
        pixel.forceRepaint = false;
      }
    }

    for (let color in byColor) {
      this.canvasCtx.fillStyle = color;

      for (let [row, col] of byColor[color]) {
        this.canvasCtx.fillRect(
          col * this.pixelSize,
          row * this.pixelSize,
          this.pixelSize,
          this.pixelSize
        );
      }
    }
  }

  paintPixel(row, col) {
    this.grid[row][col].setPaintStyles(this.canvasCtx);
    this.canvasCtx.fillRect(
      col * this.pixelSize,
      row * this.pixelSize,
      this.pixelSize,
      this.pixelSize
    );
  }

  /**
   * * Change the rules to each cell in grid
   */
  setRules(S_min, S_max, B_min, B_max) {
    this.grid.forEach((row) => {
      row.forEach((cell) => {
        cell.S_min = S_min;
        cell.S_max = S_max;
        cell.B_min = B_min;
        cell.B_max = B_max;
      });
    });
  }

  resetLife(preset = "empty", chanceOfLife = 0.005) {
    this.generations = 0;

    console.log(preset);

    this.grid.forEach((row) => {
      row.forEach((pixel) => {
        pixel.previousState = pixel.alive;
        pixel.alive = Math.random() < chanceOfLife;
      });
    });

    if (preset !== 0 || preset !== null || preset !== "empty") {
      console.log("cargando preset");
      const centerX = Math.floor(this.cols / 2);
      const centerY = Math.floor(this.rows / 2);
      presets[preset].forEach((pair) => {
        this.grid[pair[1] + centerY][pair[0] + centerX].alive = true;
      });
    }

    this.repaint();
  }

  setPixelColors(lifeStyle, deathStyle) {
    this.grid.forEach((row) => {
      row.forEach((pixel) => {
        pixel.setLifeStyle(lifeStyle);
        pixel.setDeathStyle(deathStyle);
        pixel.forceRepaint = true;
      });
    });
  }

  registerMouseListeners() {
    bindMultipleEventListener(this.canvas, ["mousemove", "touchmove"], (e) => {
      e.preventDefault();

      /**
       * * getBoundingClientRect -> devuelve el tamaño del elemento y
       * * su posición relativa respecto al viewport
       */
      if (this.mouseIsDown) {
        let x, y;
        if (e.touches) {
          let rect = e.target.getBoundingClientRect();
          x = Math.floor((e.touches[0].pageX - rect.left) / this.pixelSize);
          y = Math.floor((e.touches[0].pageY - rect.top) / this.pixelSize);
        } else {
          x = Math.floor(e.offsetX / this.pixelSize);
          y = Math.floor(e.offsetY / this.pixelSize);
        }

        this.grid[y][x].handleClick();
        this.paintPixel(y, x);
      }
    });

    bindMultipleEventListener(this.canvas, ["mousedown", "touchstart"], (e) => {
      e.preventDefault();

      let x, y;
      if (e.touches) {
        let rect = e.target.getBoundingClientRect();
        x = Math.floor((e.touches[0].pageX - rect.left) / this.pixelSize);
        y = Math.floor((e.touches[0].pageY - rect.top) / this.pixelSize);
      } else {
        x = Math.floor(e.offsetX / this.pixelSize);
        y = Math.floor(e.offsetY / this.pixelSize);
      }

      this.grid[y][x].handleClick();
      this.paintPixel(y, x);
      this.mouseIsDown = true;
    });

    bindMultipleEventListener(this.canvas, ["mouseup", "touchend"], (e) => {
      e.preventDefault();
      this.mouseIsDown = false;
    });
  }
}
```

2.5. JavaScript - GOL-Presets

```
export const presets = {
  empty: [],
  block: [
    [0, 0],
    [1, 0],
    [0, 1],
    [1, 1],
  ],
  "bee-hive": [
    [0, -1],
    [1, -1],
    [-1, 0],
    [2, 0],
    [0, 1],
    [1, 1],
  ],
  loaf: [
    [0, -1],
    [1, -1],
    [-1, 0],
    [2, 0],
    [0, 1],
    [2, 1],
    [1, 2],
  ],
  boat: [
    [0, 0],
    [1, 0],
    [0, 1],
    [2, 1],
    [1, 2],
  ],
  tub: [
    [0, -1],
    [1, 0],
    [-1, 0],
    [0, 1],
  ],
  blinker: [
    [-1, 0],
    [0, 0],
    [1, 0],
  ],
  toad: [
    [0, 0],
    [1, 0],
    [2, 0],
    [-1, 1],
    [0, 1],
    [1, 1],
  ],
  beacon: [
    [-1, -1],
    [0, -1],
    [-1, 0],
    [2, 1],
    [1, 2],
    [2, 2],
  ],
  pulsar: [
    [-4, -6],
    [-3, -6],
    [-2, -6],
    [4, -6],
    [3, -6],
    [2, -6],
    [-6, -2],
    [-6, -3],
    [-6, -4],
    [-1, -2],
    [-1, -3],
    [-1, -4],
    [1, -2],
    [1, -3],
    [1, -4],
    [6, -2],
    [6, -3],
    [6, -4],
    [-4, -1],
    [-3, -1],
    [-2, -1],
    [4, -1],
    [3, -1],
    [2, -1],
    [-4, 1],
    [-3, 1],
    [-2, 1],
    [4, 1],
    [3, 1],
    [2, 1],
    [-4, 6],
    [-3, 6],
    [-2, 6],
    [4, 6],
    [3, 6],
    [2, 6],
    [1, 2],
    [1, 3],
    [1, 4],
    [6, 2],
    [6, 3],
    [6, 4],
    [-1, 2],
    [-1, 3],
    [-1, 4],
    [-6, 2],
    [-6, 3],
    [-6, 4],
  ],
  "penta-decathalon": [
    [0, -4],
    [0, -3],
    [0, -1],
    [0, 0],
    [0, 1],
    [0, 2],
    [0, 4],
    [0, 5],
    [-1, -2],
    [1, -2],
    [-1, 3],
    [1, 3],
  ],
  glider: [
    [0, -1],
    [1, 0],
    [-1, 1],
    [0, 1],
    [1, 1],
  ],
  "light-weight spaceship": [
    [-1, -1],
    [0, -1],
    [1, -1],
    [2, -1],
    [-2, 0],
    [2, 0],
    [2, 1],
    [-2, 2],
    [1, 2],
  ],
  "middle-weight spaceship": [
    [-1, -2],
    [0, -2],
    [1, -2],
    [2, -2],
    [3, -2],
    [-2, -1],
    [3, -1],
    [3, 0],
    [-2, 1],
    [2, 1],
    [0, 2],
  ],
  "heavy-weight spaceship": [
    [-2, -2],
    [-1, -2],
    [-0, -2],
    [1, -2],
    [2, -2],
    [3, -2],
    [-3, -1],
    [3, -1],
    [3, 0],
    [-3, 1],
    [2, 1],
    [-1, 2],
    [0, 2],
  ],
  "r-pentomino": [
    [0, -1],
    [1, -1],
    [-1, 0],
    [0, 0],
    [0, 1],
  ],
  diehard: [
    [-3, 0],
    [-2, 0],
    [3, -1],
    [-2, 1],
    [2, 1],
    [3, 1],
    [4, 1],
  ],
  acorn: [
    [-2, -1],
    [0, 0],
    [-3, 1],
    [-2, 1],
    [1, 1],
    [2, 1],
    [3, 1],
  ],
  "block-laying switch engine #1": [
    [3, -2],
    [1, -1],
    [3, -1],
    [4, -1],
    [1, 0],
    [3, 0],
    [1, 1],
    [-1, 2],
    [-1, 3],
    [-3, 3],
  ],
  "block-laying switch engine #2": [
    [-2, -2],
    [-1, -2],
    [0, -2],
    [2, -2],
    [-2, -1],
    [1, 0],
    [2, 0],
    [-1, 1],
    [0, 1],
    [2, 1],
    [-2, 2],
    [0, 2],
    [2, 2],
  ],
};

export function loadPresets() {
  const select = document.querySelector("#presets");
  let option;

  Object.keys(presets).forEach(function (preset) {
    option = document.createElement("option");

    option.value = option.textContent = preset;

    select.appendChild(option);
  });
}
```

2.6. JavaScript - Utilidades

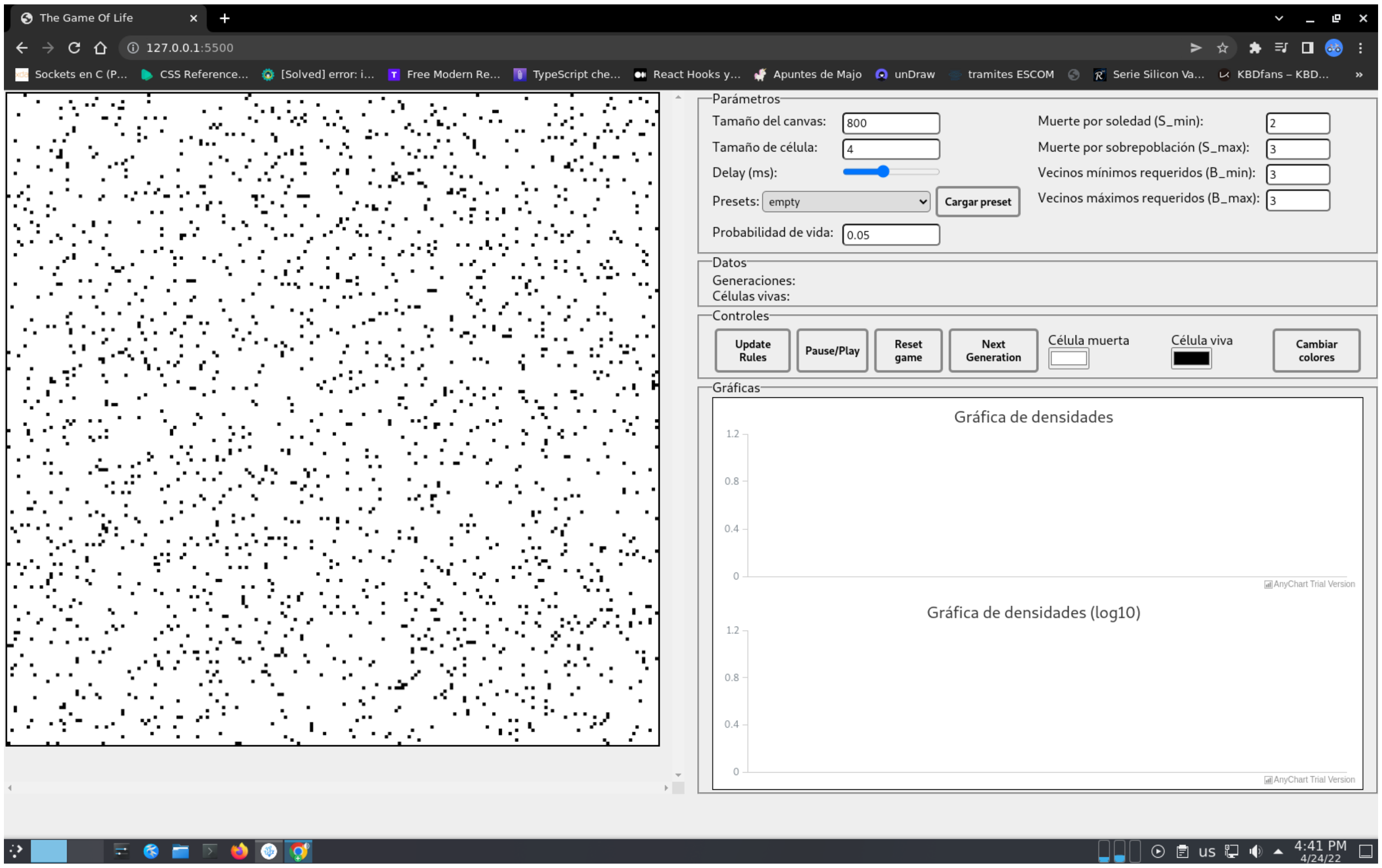
```
export function bindMultipleEventListener(element, eventNames, callback) {
  eventNames.forEach((eventName) => {
    element.addEventListener(eventName, callback);
  });
}
```

2.7. JavaScript - Configuración de gráfica

```
export class Chart {
  constructor(container, title) {
    this.dataset = anychart.data.set([]);

    // set chart type
    var chart = anychart.line();

    chart.title({
      text: title,
      fontColor: "#333",
      fontSize: 20,
    });

    // set data
    chart.spline(this.dataset).markers(null);

    // disable stagger mode. Only one line for x axis labels
    chart.xAxis().staggerMode(false);

    // set container and draw chart
    chart.container(container).draw();
  }

  updateChart(generations, population) {
    this.dataset.append({
      x: generations,
      value: population,
    });
  }
}
```

Capítulo 3

Capturas



Parámetros

Tamaño del canvas:	800	Muerte por soledad (S_min):	2
Tamaño de célula:	4	Muerte por sobrepoblación (S_max):	3
Delay (ms):		Vecinos mínimos requeridos (B_min):	3
Presets:	empty	Vecinos máximos requeridos (B_max):	3
Probabilidad de vida:	0.05		

Datos

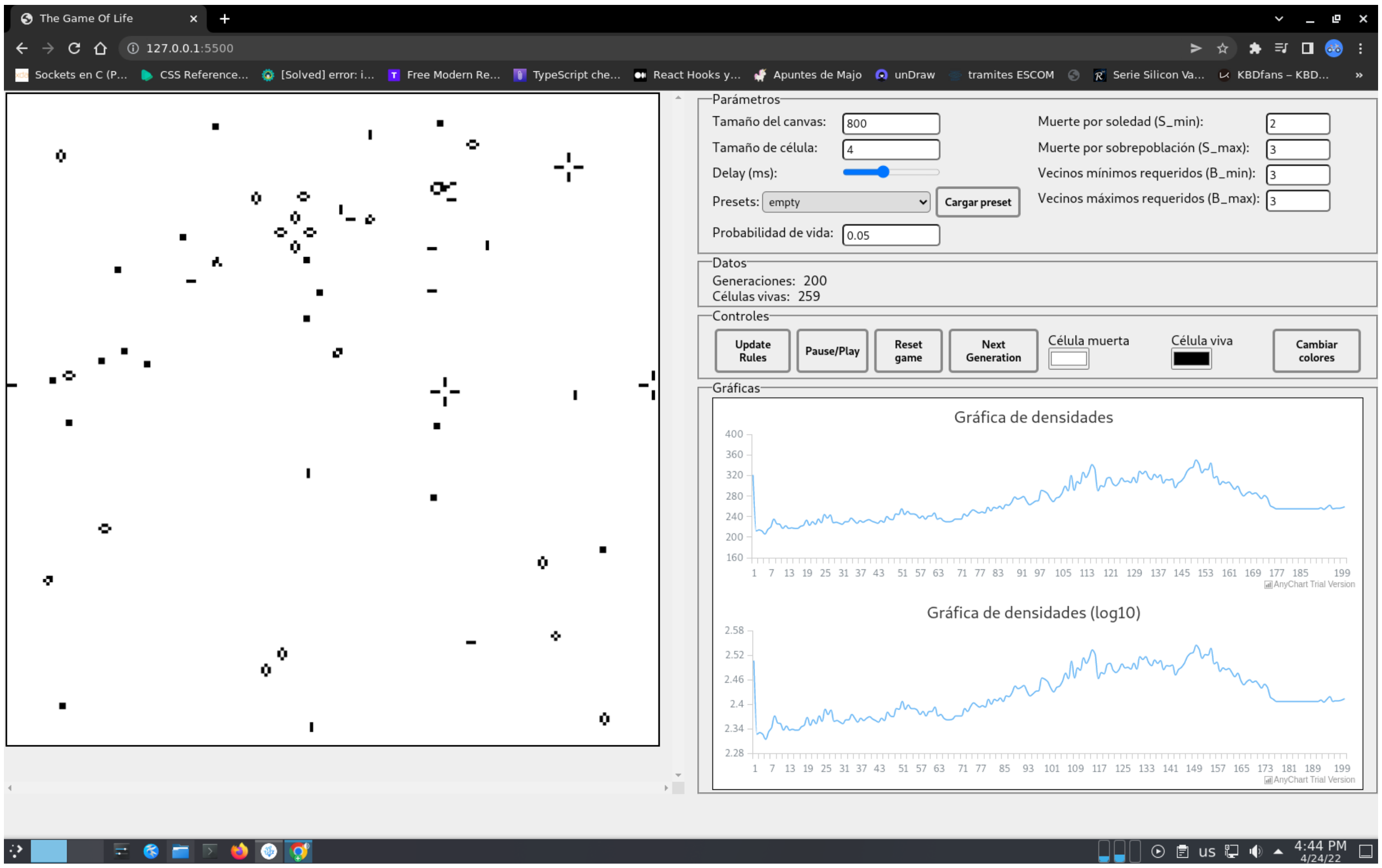
Generaciones:

Células vivas:

Controles

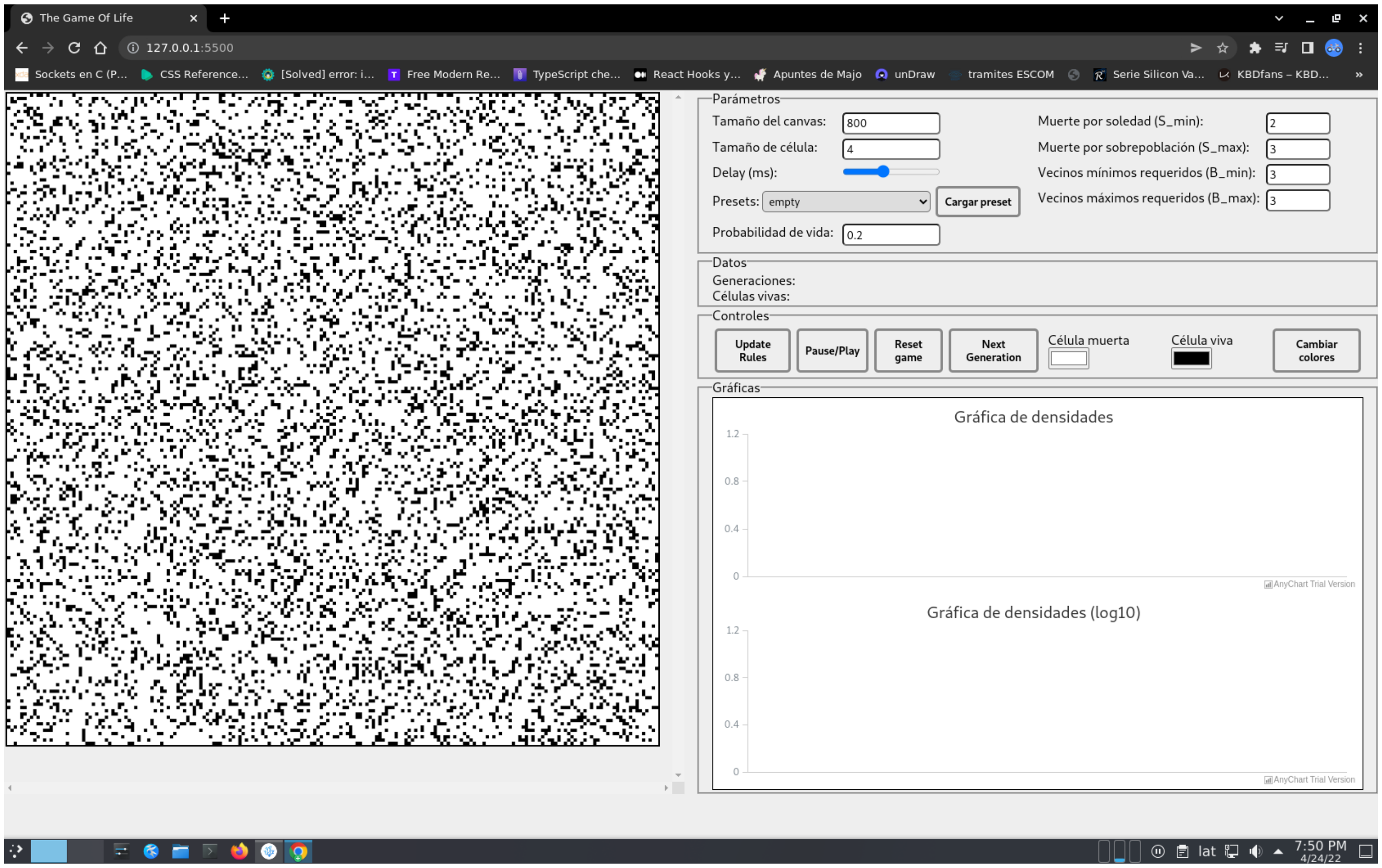
Gráficas

Figura 3.1: Configuración inicial, densidad 0.05


Parámetros
Tamaño del canvas: 800
Tamaño de célula: 4
Delay (ms):
Presets: empty
Probabilidad de vida: 0.05
Muerte por soledad (S_min): 2
Muerte por sobrepoblación (S_max): 3
Vecinos mínimos requeridos (B_min): 3
Vecinos máximos requeridos (B_max): 3

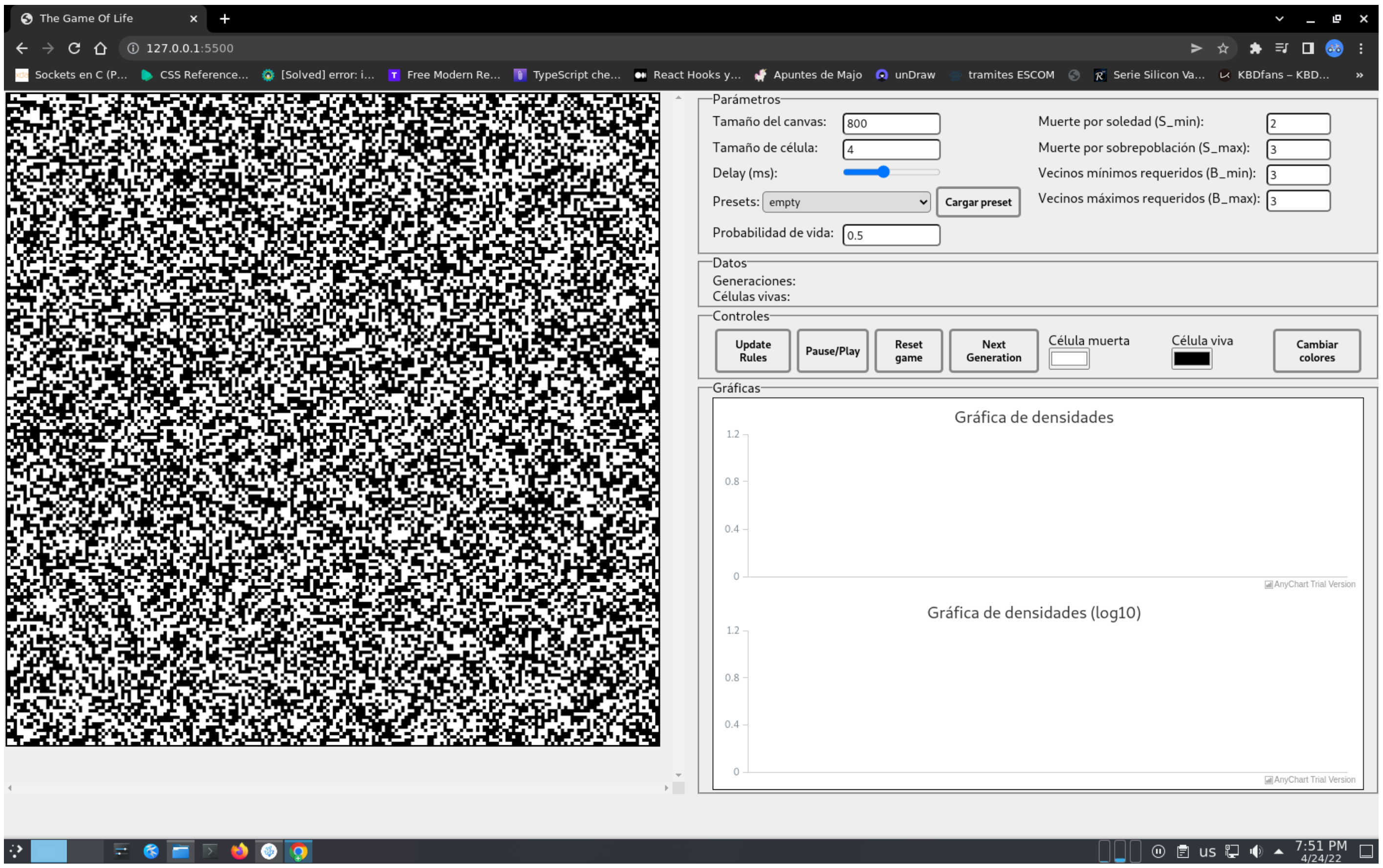
Datos
Generaciones: 200
Células vivas: 259

Figura 3.2: Funcionando con la configuración inicial


Parámetros
Tamaño del canvas: 800
Tamaño de célula: 4
Delay (ms):
Presets: empty
Probabilidad de vida: 0.2
Muerte por soledad (S_min): 2
Muerte por sobrepoblación (S_max): 3
Vecinos mínimos requeridos (B_min): 3
Vecinos máximos requeridos (B_max): 3

Datos
Generaciones:
Células vivas:

Figura 3.3: Densidad 0.2


The Game Of Life
127.0.0.1:5500
Sockets en C (P... CSS Reference... [Solved] error: i... Free Modern Re... TypeScript che... React Hooks y... Apuntes de Majo unDraw tramites ESCOM Serie Silicon Va... KBDfans – KBD...
Update Rules
Pause/Play
Reset game
Next Generation
Cambiar colores
Gráfica de densidades
Gráfica de densidades (log10)
AnyChart Trial Version
7:51 PM 4/24/22

Parámetros

Tamaño del canvas: 800

Tamaño de célula: 4

Delay (ms):

Presets: empty

Cargar preset

Probabilidad de vida: 0.5

Muerte por soledad (S_min): 2

Muerte por sobrepoblación (S_max): 3

Vecinos mínimos requeridos (B_min): 3

Vecinos máximos requeridos (B_max): 3

Datos

Generaciones:

Células vivas:

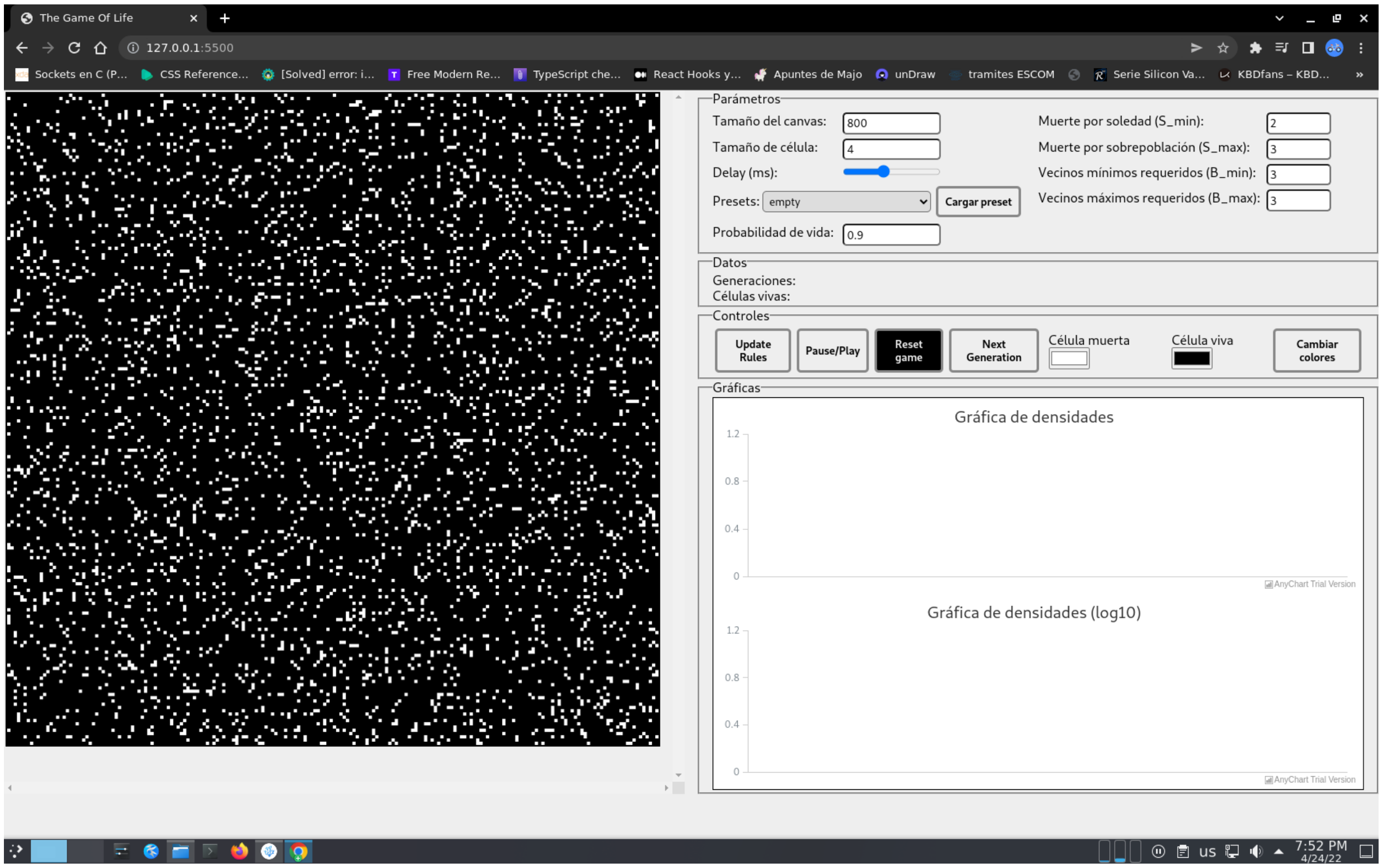
Controles

Célula muerta ⬜

Célula viva ⬛

Gráficas

Figura 3.4: Densidad 0.5


The Game Of Life
127.0.0.1:5500
Sockets en C (P... CSS Reference... [Solved] error: i... Free Modern Re... TypeScript che... React Hooks y... Apuntes de Majo unDraw tramites ESCOM Serie Silicon Va... KBDfans – KBD...
Update Rules
Pause/Play
Reset game
Next Generation
Cambiar colores
Gráfica de densidades
Gráfica de densidades (log10)
AnyChart Trial Version
7:52 PM 4/24/22

Parámetros

Tamaño del canvas: 800

Tamaño de célula: 4

Delay (ms):

Presets: empty

Cargar preset

Probabilidad de vida: 0.9

Muerte por soledad (S_min): 2

Muerte por sobrepoblación (S_max): 3

Vecinos mínimos requeridos (B_min): 3

Vecinos máximos requeridos (B_max): 3

Datos

Generaciones:

Células vivas:

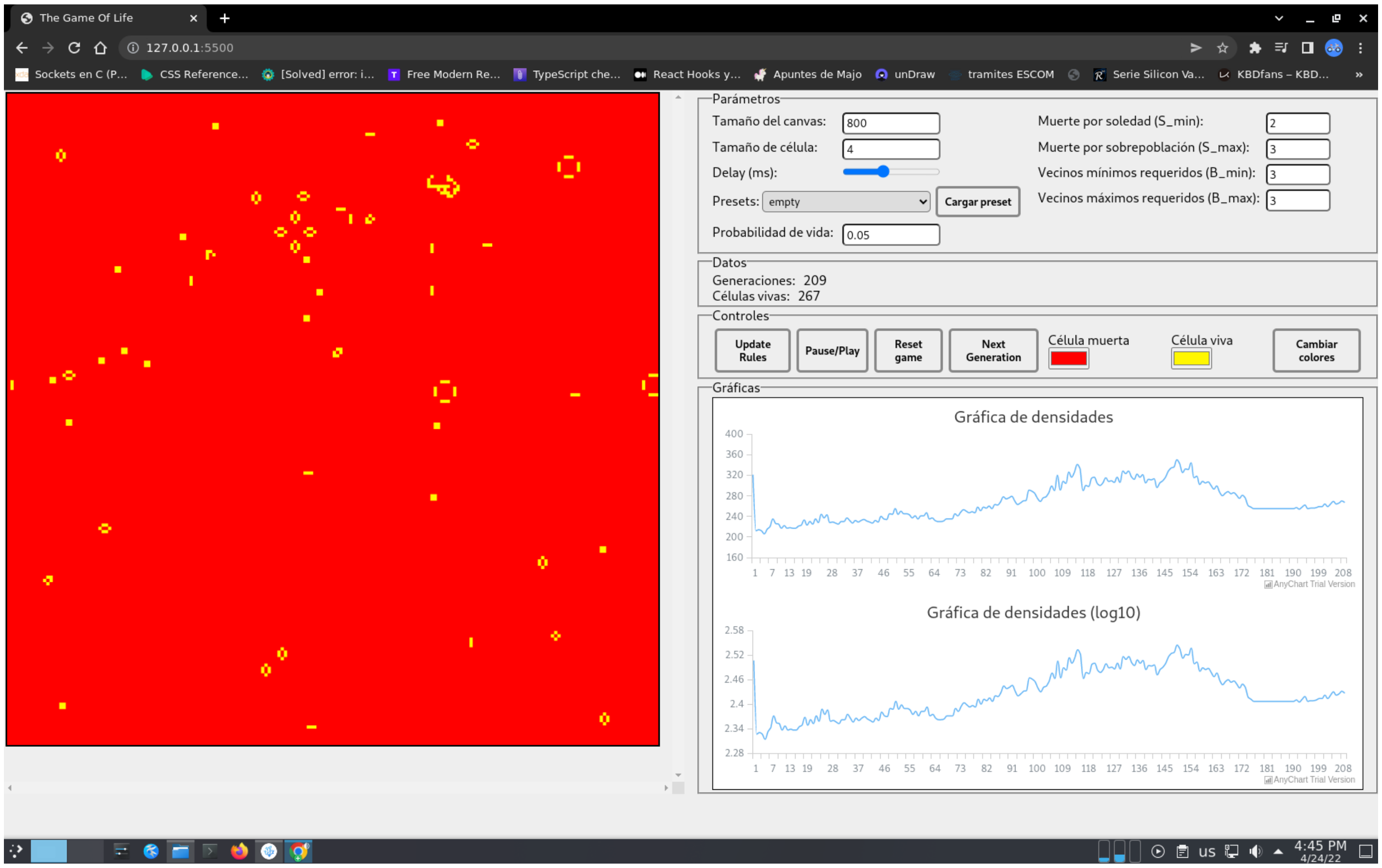
Controles

Célula muerta ⬜

Célula viva ⬛

Gráficas

Figura 3.5: Densidad 0.9



Parámetros

Tamaño del canvas: 800

Tamaño de célula: 4

Delay (ms):

Presets: empty

Cargar preset

Probabilidad de vida: 0.05

Muerte por soledad (S_min): 2

Muerte por sobrepoblación (S_max): 3

Vecinos mínimos requeridos (B_min): 3

Vecinos máximos requeridos (B_max): 3

Datos

Generaciones: 209

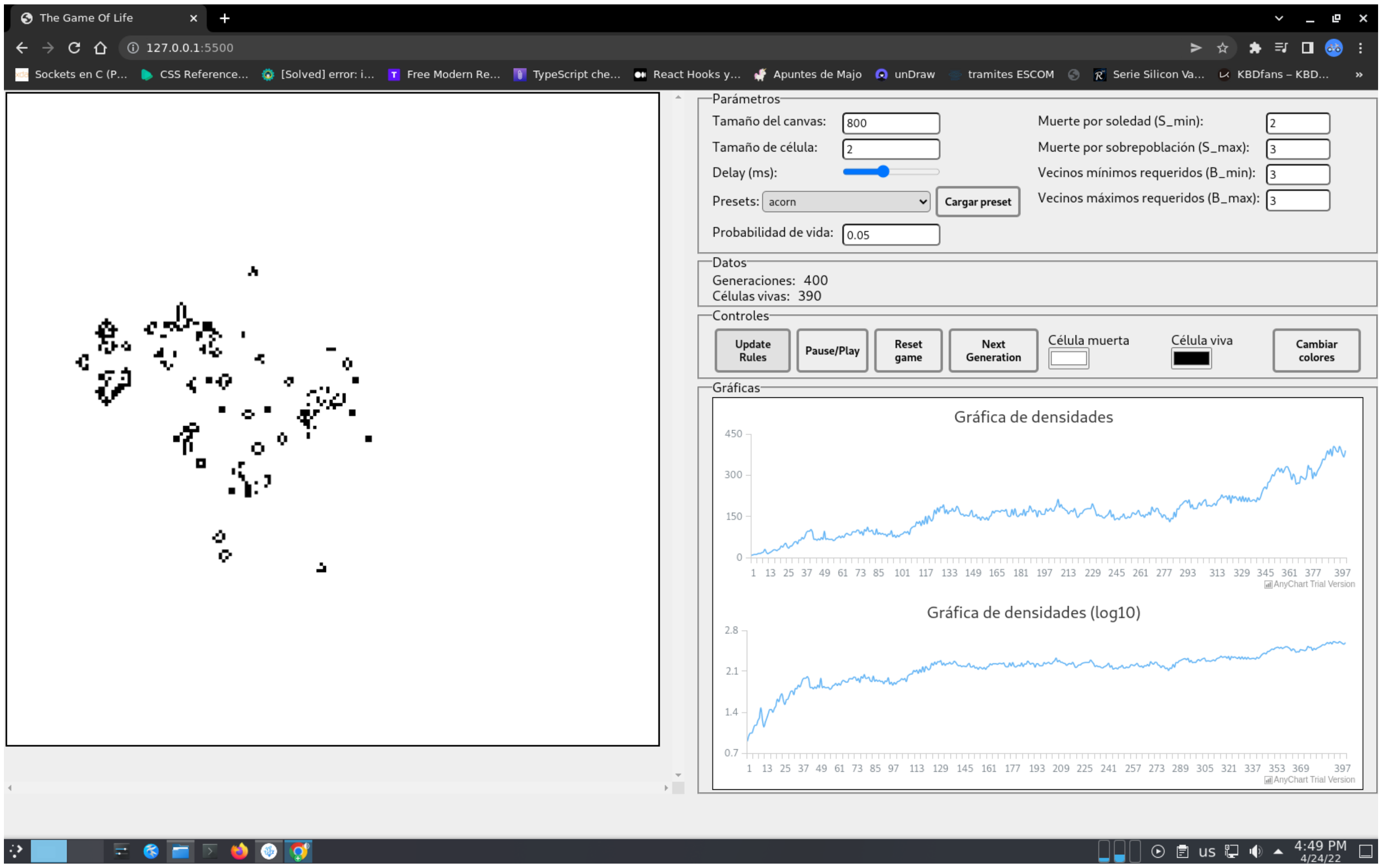
Células vivas: 267

Controles

Update Rules | Pause/Play | Reset game | Next Generation | Célula muerta 🟥 | Célula viva 🟨 | Cambiar colores

Gráficas

Figura 3.6: Colores cambiados



Parámetros

Tamaño del canvas: 800

Tamaño de célula: 2

Delay (ms):

Presets: acorn

Cargar preset

Probabilidad de vida: 0.05

Muerte por soledad (S_min): 2

Muerte por sobrepoblación (S_max): 3

Vecinos mínimos requeridos (B_min): 3

Vecinos máximos requeridos (B_max): 3

Datos

Generaciones: 400

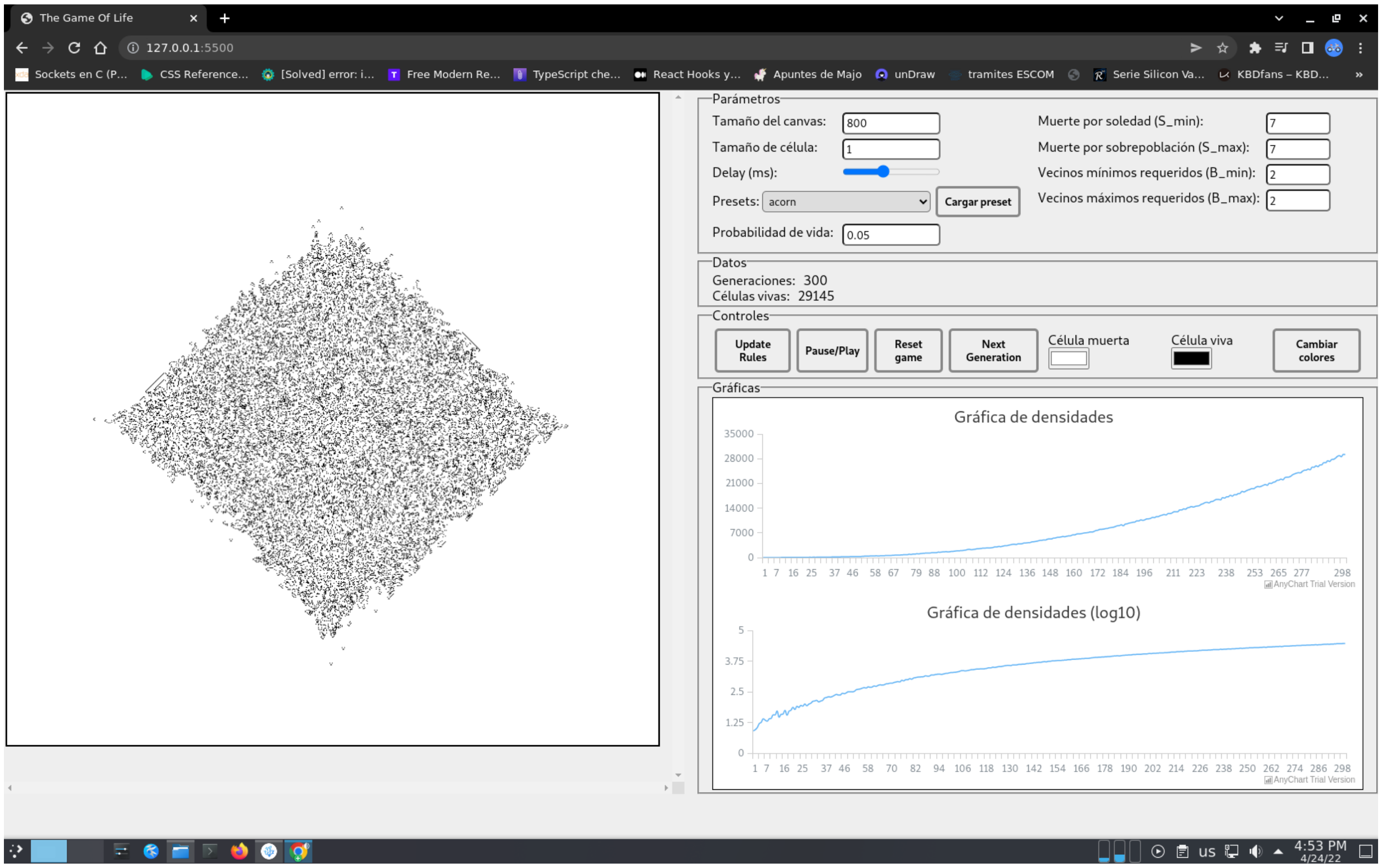
Células vivas: 390

Controles

Update Rules | Pause/Play | Reset game | Next Generation | Célula muerta ⬜ | Célula viva ⬛ | Cambiar colores

Gráficas

Figura 3.7: Acorn con reglas R(2, 3, 3, 3)



Tamaño del canvas: 800

Tamaño de célula: 1

Delay (ms):

Presets: acorn

Probabilidad de vida: 0.05

Muerte por soledad (S_min): 7

Muerte por sobrepoblación (S_max): 7

Vecinos mínimos requeridos (B_min): 2

Vecinos máximos requeridos (B_max): 2

Generaciones: 300

Células vivas: 29145

Figura 3.8: Acorn con reglas R(7, 7, 2, 2)