Estructura de Datos y Análisis de Algoritmos
Grupo B1
Proyecto de Grafos: LABERINTO

Jeferson Jair Acevedo Sarmiento
Brayan Yesid Quintero Santander

18 de julio de 2023

Índice

1. LABERINTO

1.1. Problema

El problema central de este proyecto es desarrollar un juego que se base en la generación de laberintos de tamaño nxn, con tres niveles de dificultad distintos: fácil, normal y difícil, todo eso mediante el uso de grafos. La idea principal es crear un juego desafiante y divertido que ponga a prueba las habilidades del jugador al enfrentarlo a la resolución de laberintos en un tablero de dimensiones variables.

En el nivel fácil, los laberintos serán más simples y de menor tamaño, lo que permitirá a los jugadores principiantes familiarizarse con los conceptos básicos del juego. A medida que se avanza al nivel normal, los laberintos se volverán más complejos y de tamaño mediano, lo que requerirá un mayor nivel de destreza y conocimiento por parte de los jugadores. Por último, en el nivel difícil, se plantearán desafíos especialmente diseñados para desafiar incluso a los jugadores más experimentados.

1.2. Estrategia de Solución

La estrategia de solución utilizada para realizar este codigo fue la siguiente:

1. La clase MazeGeneration extiende JPanel y representa el panel en el que se dibujará el laberinto generado.

2. Se definen constantes N, S, E y W para representar las direcciones de movimiento en el laberinto (norte, sur, este y oeste).

3. La clase tiene varios campos, incluyendo el ancho del laberinto (width), una matriz grid que representa las celdas del laberinto y sus conexiones, un generador de números aleatorios (random), matrices visited y solution para rastrear las celdas visitadas y la solución del laberinto, respectivamente, y una pila stack para realizar el retroceso recursivo.

4. El constructor MazeGeneration recibe el ancho del laberinto y realiza varias inicializaciones, incluyendo la creación de la matriz grid y las matrices visited y solution.

5. El método generateMaze implementa el algoritmo de generación del laberinto utilizando el enfoque de retroceso recursivo. El laberinto se genera mediante la creación de caminos entre las celdas utilizando las direcciones N, S, E y W. El algoritmo elige aleatoriamente una dirección y crea una conexión entre las celdas correspondientes. El proceso se repite hasta que todas las celdas han sido visitadas.

6. El método displayMaze se encarga de dibujar el laberinto en el panel utilizando la clase Graphics2D. Recorre la matriz grid y dibuja las paredes y conexiones entre las celdas, así como los marcadores de celdas visitadas y la solución del laberinto.

7. El método paintComponent se encarga de pintar el componente gráfico. Limpia el panel, llama a displayMaze para dibujar el laberinto y luego libera los recursos gráficos.

8. El método createAndShowGUI crea la interfaz gráfica para mostrar el laberinto. Crea un marco (JFrame) y un panel (MazeGeneration). Genera el laberinto llamando a generateMaze. Añade el panel al marco y configura los componentes adicionales como una etiqueta de dificultad y un botón para cambiar la dificultad. Finalmente, muestra el marco.

En resumen, la estrategia de solución de este código es utilizar el algoritmo de retroceso recursivo para generar un laberinto y luego mostrarlo en una interfaz gráfica utilizando componentes de Swing.

1.3. Informacion real utilizada

https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html

Utilizamos esta página como referencia y guía durante el proceso de representación del laberinto, selección de colores y otras configuraciones visuales. Esta página nos proporcionó información valiosa y ejemplos prácticos para garantizar una representación gráfica adecuada y estéticamente agradable del laberinto.

https://en.wikipedia.org/wiki/Maze$_g$*eneration*$_a$*lgorithm*
https : *//www.geeksforgeeks.org/rat* − *in* − *a* − *maze/*

Estas dos páginas fueron fundamentales en nuestra búsqueda de orientación durante la creación del laberinto. Nos proporcionaron valiosos tutoriales, algoritmos y ejemplos que nos guiaron a lo largo del proceso de generación del laberinto. A través de estas páginas, adquirimos conocimientos sobre diferentes enfoques de generación de laberintos y pudimos implementarlos en nuestro código.

1.4. Solucion

Listing 1: Solucion

```
1
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.util.Random;
7  import java.util.Stack;
8
9  public class MazeGeneration extends JPanel {
10     public static String dificul="";
11     private final int width;
12     private final int[][] grid;
13     private final Random random;
14     private static final int N = 1; // North direction
15     private static final int S = 2; // South direction
16     private static final int E = 4; // East direction
17     private static final int W = 8; // West direction
18
```

```
19      private boolean[][] visited;
20      private boolean[][] solution;
21
22      private Stack<Point> stack;
23
24      public GrafoMatrizAdya matriz;
25
26      public MazeGeneration(int width) {
27          this.width = width;
28          matriz = new GrafoMatrizAdya(width);
29          this.grid = new int[matriz.getV()][matriz.getV()];
30          this.random = new Random();
31          this.visited = new boolean[matriz.getV()][matriz.getV()];
32          this.solution = new boolean[matriz.getV()][matriz.getV()];
33
34          setPreferredSize(new Dimension(matriz.getV() * 35, matriz.getV() * 30));
35      }
36
37      public static int getN() {
38          return N;
39      }
40
41      public static int getS() {
42          return S;
43      }
44
45      public static int getE() {
46          return E;
47      }
48
49      public static int getW() {
50          return W;
51      }
52
53      private void generateMaze() {
54          int cont = 0;
55          for (int y = 0; y < width; y++) {
56              int runStart = 0;
57              for (int x = 0; x < width; x++) {
58                  if (y > 0 && (x + 1 == width || random.nextInt(2) == 0)) {
59                      int cell = runStart + random.nextInt(x - runStart + 1);
60                      grid[y][cell] += N; // Set North direction
```

```
                    grid[y - 1][cell] += S; // Set South direction
                    runStart = x + 1;
                    cont++;
                } else if (x + 1 < width) {
                    grid[y][x] += E; // Set East direction
                    grid[y][x + 1] += W; // Set West direction
                    cont++;
                }
            }
        }
    }

private void displayMaze(Graphics2D g2d) {
    int cellSize = 30;
    int wallThickness = 3;
    int circleSize = cellSize / 3;

    g2d.setStroke(new BasicStroke(wallThickness));

    for (int y = 0; y < matriz.getV(); y++) {
        for (int x = 0; x < matriz.getV(); x++) {
            int cell = grid[y][x];
            int xPos = x * cellSize;
            int yPos = y * cellSize;

            // Dibujar borde superior
            if (y == 0) {

                g2d.setColor(Color.BLACK);
                g2d.drawLine(xPos, yPos, xPos + cellSize, yPos);
            }

            // Dibujar borde izquierdo
            if (x == 0) {

                g2d.setColor(Color.BLACK);
                g2d.drawLine(xPos, yPos, xPos, yPos + cellSize);
            }

            // Dibujar pared sur
            if ((cell & S) == 0) {
                g2d.setColor(Color.BLACK);
```

```
103                 g2d.drawLine(xPos, yPos + cellSize, xPos + cellSize, yPos +
                        cellSize);
104             }
105
106             // Dibujar pared este
107             if ((cell & E) == 0) {
108                 g2d.setColor(Color.BLACK);
109                 g2d.drawLine(xPos + cellSize, yPos, xPos + cellSize, yPos +
                        cellSize);
110             }
111
112             if (visited[y][x]) {
113                 g2d.setColor(Color.LIGHT_GRAY);
114                 g2d.fillRect(xPos + wallThickness, yPos + wallThickness, cellSize
                        - 2 * wallThickness, cellSize - 2 * wallThickness);
115             }
116
117             if (solution[y][x]) {
118                 g2d.setColor(Color.GREEN);
119                 g2d.fillOval(xPos + (cellSize - circleSize) / 2, yPos + (cellSize
                        - circleSize) / 2, circleSize, circleSize);
120             }
121         }
122     }
123 }
124
125
126     @Override
127     protected void paintComponent(Graphics g) {
128         super.paintComponent(g);
129         Graphics2D g2d = (Graphics2D) g.create();
130         g2d.setBackground(Color.WHITE);
131         g2d.clearRect(0, 0, getWidth(), getHeight());
132
133         displayMaze(g2d);
134
135         g2d.dispose();
136     }
137
138
139     public void createAndShowGUI() {
140
```

```
141         JFrame frame = new JFrame("Maze Generation");
142         int ancho = 800;
143         int alto = 600;
144         frame.setSize(ancho, alto);
145         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
146         MazeGeneration maze = new MazeGeneration(width);
147         maze.generateMaze();
148         frame.getContentPane().add(maze, BorderLayout.CENTER);
149
150
151         JLabel Dificultad =new JLabel("Dificultad:"+dificul+"
                ",SwingConstants.LEADING);
152         JButton Cambiar=new JButton("Cambiar \ndificultad");
153         Cambiar.setBackground(Color.CYAN);
154         Font font = new Font("Script MT Bold", Font.PLAIN, 30);
155         Cambiar.setFont(font);
156         Dimension dimensiones = new Dimension(270, 40);
157         Cambiar.setPreferredSize(dimensiones);
158         Font fuente = new Font("Arial", Font.BOLD, 20);
159         Dificultad.setFont(fuente);
160         Dificultad.setBackground(Color.WHITE);
161         Dificultad.setOpaque(true);
162
163
164
165         JButton P=new JButton("                    ");
166         P.setBackground(Color.WHITE);
167         P.setEnabled(false);
168         frame.add(P, BorderLayout.SOUTH);
169       JPanel panelDificultad = new JPanel();
170       P.setBorderPainted(false);
171         P.setFocusPainted(false);
172
173     panelDificultad.setLayout(new GridBagLayout());
174     panelDificultad.setBackground(Color.WHITE);
175
176     // Crear las restricciones de diseo
177     GridBagConstraints constraints = new GridBagConstraints();
178     constraints.anchor = GridBagConstraints.EAST; // Alinear a la derecha
179     constraints.weightx = 1.0; // Expandir horizontalmente
180     constraints.insets = new Insets(20, 1, 0, 40); // Margen interno
181
```

```
182     panelDificultad.add(Dificultad, constraints);
183 
184     constraints.gridy = 1; // Fila 1
185     panelDificultad.add(Cambiar, constraints);
186 
187 
188     frame.add(panelDificultad, BorderLayout.EAST);
189 
190 
191 
192         frame.setVisible(true);
193 
194         Cambiar.addActionListener(new ActionListener() {
195     @Override
196     public void actionPerformed(ActionEvent e) {
197         frame.dispose();
198 
199         java.awt.EventQueue.invokeLater(new Runnable() {
200             public void run() {
201                 Interfaz interfaz = new Interfaz();
202                 interfaz.setVisible(true);
203 
204                 // Establecer la visibilidad de los paneles
205                 Interfaz.jPanel1.setVisible(false);
206                 Interfaz.jPanel2.setVisible(true);
207 
208                 // Habilitar los botones del jPanel2
209                 Component[] components = Interfaz.jPanel2.getComponents();
210                 for (Component component : components) {
211                     if (component instanceof JButton) {
212                         JButton button = (JButton) component;
213                         button.setEnabled(true);
214                     }
215                 }
216             }
217         });
218     }
219 });
220 
221     frame.pack();
222 
223     }
```

```
224     }
```

1.5. Diagramas

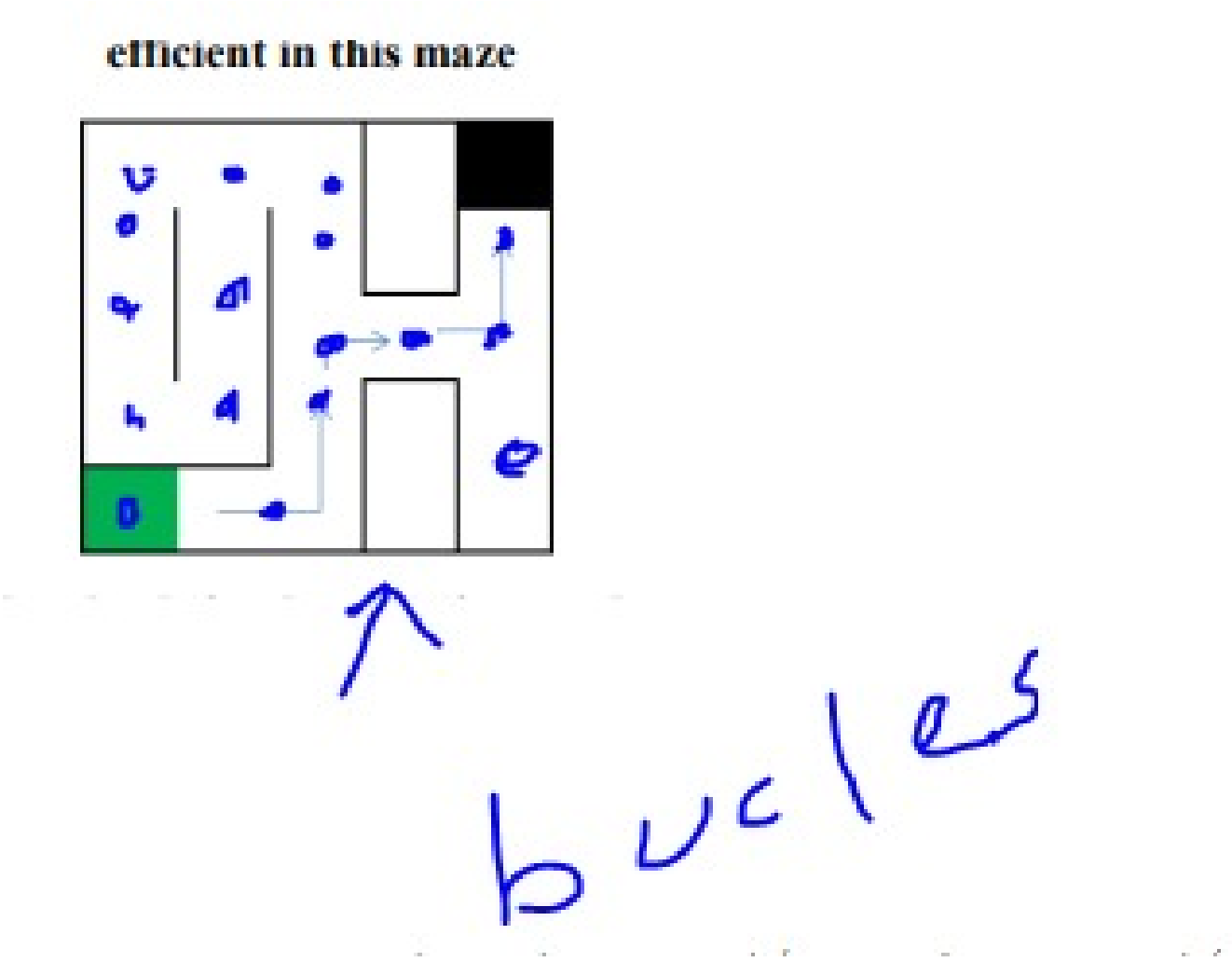

Figura 1: Laberinto

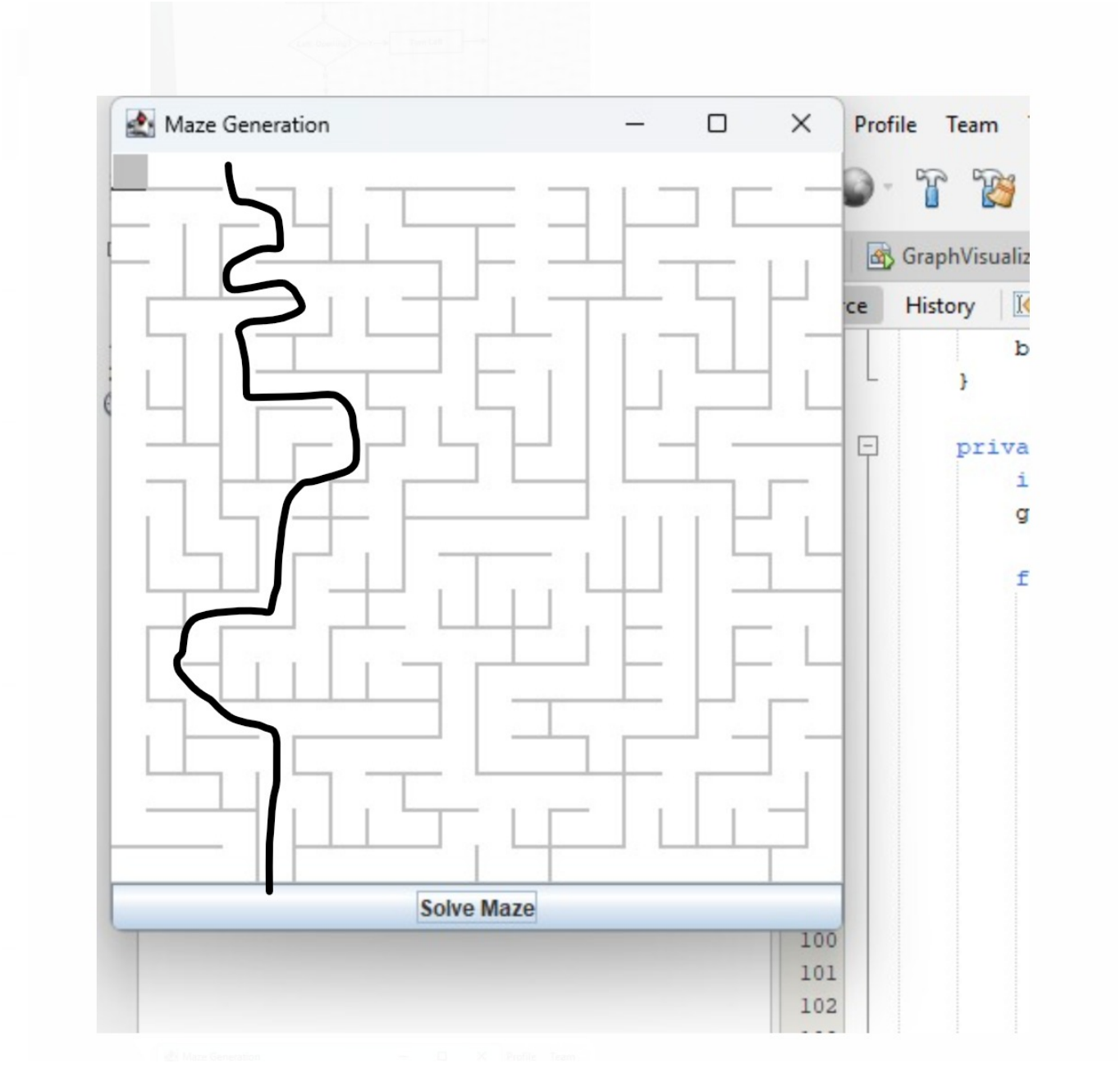

Figura 2: Laberinto

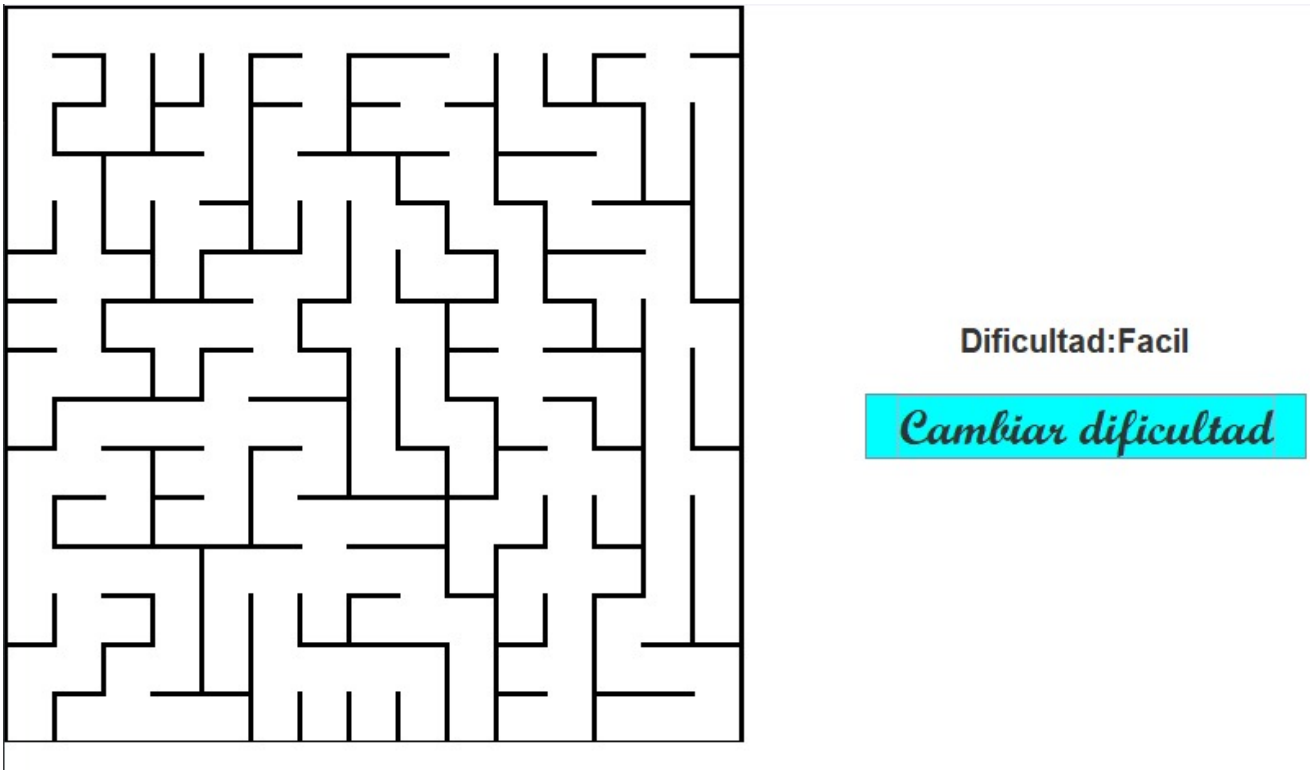

Figura 3: Laberinto

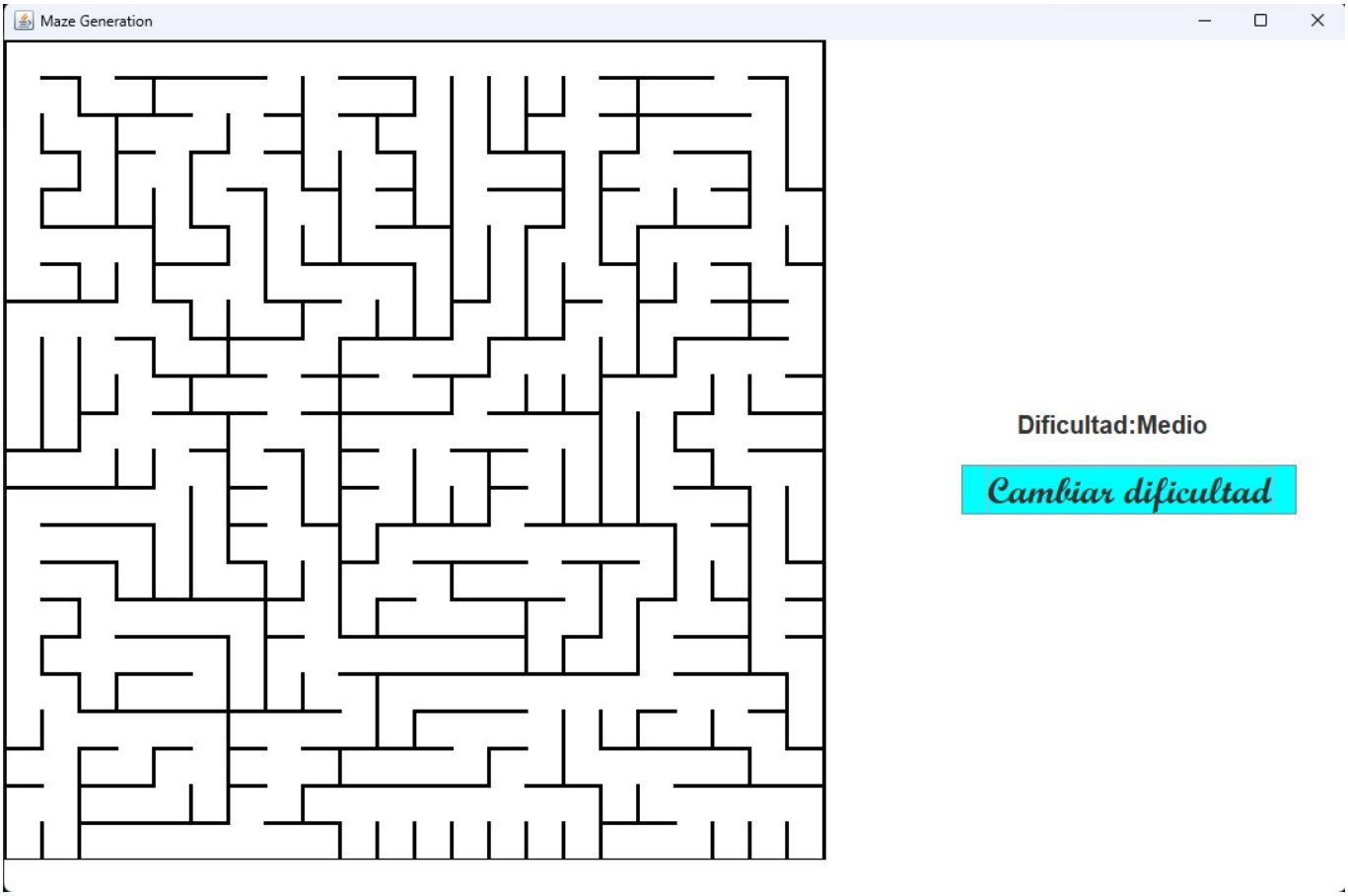

Figura 4: Laberinto

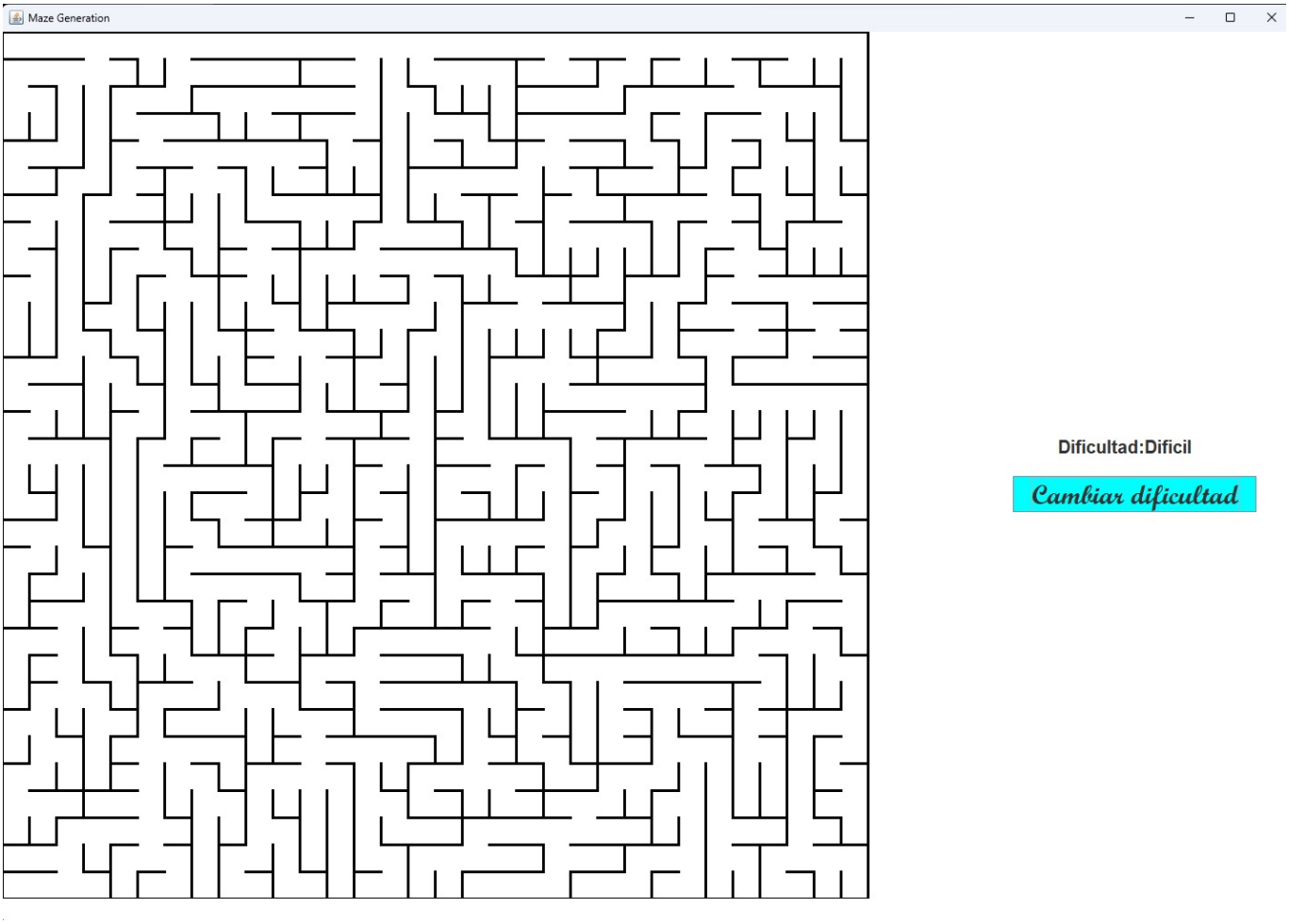

Figura 5: Laberinto