Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Lenguajes Formales y de Programación
Sección A-
Inga. Vivian Damaris Campos González
Tutor académico: Luisa María Ortíz Romero

Manual Técnico - Generador Visual de Mapas Narrativos

José Alexander López López
Carné: 202100305
Fecha de Elaboración: 30/04/2025

Índice

1. Introducción

1.1. Propósito

Este manual proporciona una guía detallada para la instalación, configuración y mantenimiento del programa **Generador Visual de Mapas Narrativos**, desarrollado en Java como parte del curso de Lenguajes Formales y de Programación.

1.2. Alcance

El sistema permite generar visualizaciones gráficas de mapas narrativos a partir de descripciones textuales en un lenguaje estructurado específico, aplicando conceptos de análisis léxico, sintáctico y programación orientada a objetos.

1.3. Público Objetivo

Dirigido a desarrolladores y personal técnico que requieran comprender la estructura y funcionamiento del sistema, así como a docentes y estudiantes del área de Ciencias de la Computación interesados en la generación visual de mapas narrativos.

2. Descripción General del Sistema

2.1. Arquitectura

El sistema está basado en una arquitectura de interfaz gráfica en Java Swing, con componentes de análisis léxico y sintáctico para procesar los archivos de entrada, y utiliza Graphviz para generar las representaciones visuales de los mapas.

2.2. Tecnologías Utilizadas

- Java SE 8+
- Java Swing para la interfaz gráfica
- Manejo de archivos con `File` y `FileReader`
- Estructuras de datos dinámicas (`HashMap`, `ArrayList`)
- Graphviz para la visualización de mapas
- Análisis léxico y sintáctico manual (sin uso de herramientas como JFlex o Cup)

2.3. Requisitos del Sistema

- Sistema Operativo: Windows, Linux o MacOS
- Java Development Kit (JDK) 8 o superior
- IDE recomendado: NetBeans, Eclipse o IntelliJ IDEA
- Graphviz instalado en el sistema

- Memoria RAM: 2GB mínimo
- Espacio en disco: 100MB mínimo

3. Instalación y Configuración

3.1. Instalación de Java

Descargar e instalar JDK desde `https://www.oracle.com/java/technologies/javase-downloads
html`.

3.2. Instalación de Graphviz

Para generar las imágenes de los mapas narrativos, es necesario instalar Graphviz:

- Windows: Descargar desde `https://graphviz.org/download/` e instalar.
- Linux: `sudo apt-get install graphviz` (Ubuntu/Debian) o `sudo yum install graphviz` (Fedora/CentOS).
- MacOS: `brew install graphviz` (usando Homebrew).

3.3. Clonación del Repositorio

Ejecutar en terminal:

```
git clone https://github.com/JoseArt777/-LFP-202100305-.git
cd -LFP-202100305-/Proyecto2
```

3.4. Ejecución del Proyecto

1. Abrir el proyecto en el IDE de su preferencia.
2. Compilar y ejecutar la clase principal `Main.java`.

4. Estructura del Código

El código se encuentra estructurado en los siguientes paquetes:

- **com.mycompany.main:** Contiene todas las clases del proyecto.

4.1. Clases Principales

- `Main`: Contiene el método principal `main()` para iniciar la aplicación.
- `MainWindow`: Implementa la interfaz gráfica principal.
- `Lexer`: Implementa el análisis léxico de los archivos de entrada.
- `Parser`: Implementa el análisis sintáctico basado en los tokens generados por el Lexer.

- `Token`: Representa un token identificado durante el análisis léxico.
- `TokenType`: Enumeración de los tipos de tokens reconocidos.
- `CharacterToken`: Representa un token a nivel de carácter para el reporte detallado.
- `World`: Representa un mundo o escenario en el mapa narrativo.
- `Place`: Representa un lugar en el mapa.
- `Connection`: Representa una conexión entre dos lugares.
- `MapObject`: Representa un objeto en el mapa.
- `DotGenerator`: Genera archivos DOT para Graphviz.
- `ReportGenerator`: Genera reportes HTML de tokens y errores.

5. Análisis Léxico

5.1. Tokens Reconocidos

Token	**Patrón**	**Descripción**
WORLD	”world”	Palabra reservada para definir un mundo
PLACE	”place”	Palabra reservada para definir un lugar
CONNECT	çonnect”	Palabra reservada para definir una conexión
OBJECT	.ºbject”	Palabra reservada para definir un objeto
AT	.ªt”	Palabra reservada para indicar posición
TO	”to”	Palabra reservada para indicar destino
WITH	”with”	Palabra reservada para indicar tipo de conexión
LBRACE	{	Llave de apertura
RBRACE	}	Llave de cierre
LPAREN	(	Paréntesis de apertura
RPAREN	)	Paréntesis de cierre
COMMA	,	Coma
COLON	:	Dos puntos
IDENTIFIER	[a-zA-Z][a-zA-Z0-9]*	Identificadores
STRING	”.*”	Cadena entre comillas
NUMBER	[0-9]+	Número entero
EOF	fin de archivo	Marca el final del archivo
ERROR	cualquier otro	Carácter no reconocido

Cuadro 1: Tokens reconocidos por el analizador léxico.

5.2. Implementación del Analizador Léxico

```
1 public class Lexer {
2     private String input;
3     private int position;
```

```
4       private int line;
5       private int column;
6       private char currentChar;
7
8       private ArrayList<Token> tokens;
9       private ArrayList<Token> errors;
10      private ArrayList<CharacterToken> charTokens;
11
12      private static final Map<String, TokenType> KEYWORDS;
13
14      static {
15          KEYWORDS = new HashMap<>();
16          KEYWORDS.put("world", TokenType.WORLD);
17          KEYWORDS.put("place", TokenType.PLACE);
18          KEYWORDS.put("connect", TokenType.CONNECT);
19          KEYWORDS.put("object", TokenType.OBJECT);
20          KEYWORDS.put("at", TokenType.AT);
21          KEYWORDS.put("to", TokenType.TO);
22          KEYWORDS.put("with", TokenType.WITH);
23      }
24
25      public Lexer() {
26          this.tokens = new ArrayList<>();
27          this.errors = new ArrayList<>();
28          this.charTokens = new ArrayList<>();
29      }
30
31      public void analyze(String input) {
32          this.input = input;
33          this.position = 0;
34          this.line = 1;
35          this.column = 1;
36          this.tokens = new ArrayList<>();
37          this.errors = new ArrayList<>();
38          this.charTokens = new ArrayList<>();
39
40          if (!input.isEmpty()) {
41              currentChar = input.charAt(0);
42          } else {
43              currentChar = '\0';
44          }
45
46          Token token;
47          do {
48              token = getNextToken();
49              if (token.getType() != TokenType.ERROR) {
50                  tokens.add(token);
51              } else {
52                  errors.add(token);
53              }
54          } while (token.getType() != TokenType.EOF);
```

```
    }

    private Token getNextToken() {
        skipWhitespace();

        if (currentChar == '\0') {
            return new Token(TokenType.EOF, "", line, column);
        }

        if (isAlpha(currentChar)) {
            return scanIdentifier();
        }

        if (isDigit(currentChar)) {
            return scanNumber();
        }

        if (currentChar == '"') {
            return scanString();
        }

        switch (currentChar) {
            case '{': return consumeToken(TokenType.LBRACE);
            case '}': return consumeToken(TokenType.RBRACE);
            case '(': return consumeToken(TokenType.LPAREN);
            case ')': return consumeToken(TokenType.RPAREN);
            case ',': return consumeToken(TokenType.COMMA);
            case ':': return consumeToken(TokenType.COLON);
        }

        Token errorToken = Token.createError(String.valueOf(
           currentChar), line, column);
        advance();
        return errorToken;
    }

    // ... otros m todos del analizador l xico
}
```

Listing 1: Implementación del analizador léxico

6. Análisis Sintáctico

6.1. Gramática del Lenguaje

A continuación se presenta la gramática libre de contexto que define el lenguaje de entrada:

worlds → world { world_body }, { world_body } ...
world → “world” STRING “{” place_list connection_list object_list “}”
place_list → place_decl*
place_decl → “place” IDENTIFIER “:” IDENTIFIER “at” “(” NUMBER “,” NUMBER “)”
connection_list → connection_decl*
connection_decl → “connect” IDENTIFIER “to” IDENTIFIER “with” STRING
object_list → object_decl*
object_decl → “object” STRING “:” IDENTIFIER “at” IDENTIFIER
| “object” STRING “:” IDENTIFIER “at” “(” NUMBER “,” NUMBER “)”

6.2. Implementación del Analizador Sintáctico

```
public class Parser {
    private ArrayList<Token> tokens;
    private int position;
    private Token currentToken;

    private ArrayList<World> worlds;
    private ArrayList<Token> errors;

    public Parser() {
        this.worlds = new ArrayList<>();
        this.errors = new ArrayList<>();
    }

    public void parse(ArrayList<Token> tokens) {
        this.tokens = tokens;
        this.position = 0;
        this.worlds = new ArrayList<>();
        this.errors = new ArrayList<>();

        if (!tokens.isEmpty()) {
            currentToken = tokens.get(0);
            parseWorlds();
        }
    }

    private void advance() {
        position++;
        if (position < tokens.size()) {
            currentToken = tokens.get(position);
        }
    }

    private boolean consume(TokenType expectedType, String
       errorMessage) {
```

```
34          if (currentToken.getType() == expectedType) {
35              advance();
36              return true;
37          } else {
38              errors.add(Token.createError(
39                  errorMessage + ", se encontr  " + currentToken.
                        getLexeme(),
40                  currentToken.getLine(),
41                  currentToken.getColumn(),
42                  Token.ErrorType.SYNTACTIC
43              ));
44              return false;
45          }
46      }
47
48      private void parseWorlds() {
49          while (currentToken.getType() != TokenType.EOF) {
50              if (currentToken.getType() == TokenType.WORLD) {
51                  parseWorld();
52              } else {
53                  if (currentToken.getType() != TokenType.EOF) {
54                      errors.add(Token.createError(
55                          "Se esperaba 'world', se encontr  " +
                                currentToken.getLexeme(),
56                          currentToken.getLine(),
57                          currentToken.getColumn()
58                      ));
59                      advance();
60                  }
61              }
62          }
63
64          if (!errors.isEmpty() && errors.get(errors.size() - 1).
                getLexeme().contains("Se esperaba 'world'")) {
65              errors.remove(errors.size() - 1);
66          }
67      }
68
69      private void parseWorld() {
70          // Consumir "world"
71          advance();
72
73          // Obtener el nombre del mundo
74          String worldName = "";
75          if (currentToken.getType() == TokenType.STRING) {
76              worldName = currentToken.getLexeme();
77              worldName = worldName.substring(1, worldName.length()
                        - 1);
78              advance();
79          } else {
80              errors.add(Token.createError(
```

```
                    "Se esperaba un nombre de mundo entre comillas",
                    currentToken.getLine(),
                    currentToken.getColumn()
                ));
            }

            World world = new World(worldName);

            // Consumir "{"
            if (consume(TokenType.LBRACE, "Se esperaba '{'")) {
                parseWorldBody(world);
                consume(TokenType.RBRACE, "Se esperaba '}'");
                worlds.add(world);

                if (currentToken.getType() == TokenType.COMMA) {
                    advance();
                }
            } else {
                while (currentToken.getType() != TokenType.WORLD &&
                       currentToken.getType() != TokenType.EOF) {
                    advance();
                }
            }
        }

        // ... otros m todos del analizador sint ctico
}
```

Listing 2: Fragmento de la implementación del analizador sintáctico

7. Generación de Gráficos con Graphviz

7.1. Estructura de los Archivos DOT

El sistema genera archivos DOT para Graphviz con la siguiente estructura:

```
digraph "Isla del Tesoro" {
  // Configuraci n general
  graph [fontname="Arial", rankdir=TB, overlap=false, splines=
     true];
  node [fontname="Arial", style=filled];
  edge [fontname="Arial"];

  // Lugares
  "Playa" [shape=ellipse, fillcolor="lightblue", label="Playa",
     pos="0,0!"];
  "Selva" [shape=parallelogram, fillcolor="forestgreen", label="
     Selva", pos="1,1!"];
  "Cima" [shape=triangle, fillcolor="sienna", label="Cima", pos="
     3,2!"];
```

```
    "Caverna" [shape=box, fillcolor="gray", label="Caverna", pos="2,1!"];
    "Embarcadero" [shape=house, fillcolor="burlywood", label="Embarcadero", pos="0,2!"];

    // Objetos en coordenadas espec ficas
    "obj_Llave_Dorada" [shape=pentagon, fillcolor="lightsteelblue", label="       Llave Dorada", pos="2,2!"];

    // Objetos en lugares
    "obj_Cofre_del_Tesoro" [shape=box3d, fillcolor="gold", label="           Cofre del Tesoro"];
    "obj_Cofre_del_Tesoro" -> "Caverna" [label="en", dir=none, style=dotted];
    "obj_Mapa_Antiguo" [shape=note, fillcolor="navajowhite", label="       Mapa Antiguo"];
    "obj_Mapa_Antiguo" -> "Embarcadero" [label="en", dir=none, style=dotted];
    "obj_Poci n_Curativa" [shape=cylinder, fillcolor="plum", label="       Poci n Curativa"];
    "obj_Poci n_Curativa" -> "Selva" [label="en", dir=none, style=dotted];

    // Conexiones entre lugares
    "Playa" -> "Selva" [label="sendero", color="saddlebrown", style=dashed];
    "Selva" -> "Caverna" [label="camino", color="black", style=solid];
    "Caverna" -> "Cima" [label="escalera", color="black", style=solid];
    "Embarcadero" -> "Playa" [label="lancha", color="blue", style=solid];
}
```

Listing 3: Ejemplo de archivo DOT generado

7.2. Implementación del Generador DOT

```
public class DotGenerator {

    public void generateDotFile(World world, String filePath) throws IOException {
        StringBuilder dotContent = new StringBuilder();

        // Inicio del archivo DOT
        dotContent.append("digraph \"").append(world.getName()).append("\" {\n");
        dotContent.append("  // Configuraci n general\n");
        dotContent.append("  graph [fontname=\"Arial\", rankdir=TB, overlap=false, splines=true];\n");
```

```
10          dotContent.append("  node [fontname=\"Arial\", style=
                filled];\n");
11          dotContent.append("  edge [fontname=\"Arial\"];\n\n");
12
13          // Generar nodos para lugares
14          dotContent.append("  // Lugares\n");
15          Map<String, Place> placeMap = new HashMap<>();
16          for (Place place : world.getPlaces()) {
17              placeMap.put(place.getName(), place);
18              dotContent.append("  \"").append(place.getName()).
                    append("\" [");
19              dotContent.append("shape=").append(place.getShape()).
                    append(", ");
20              dotContent.append("fillcolor=\"").append(place.
                    getFillColor()).append("\", ");
21              dotContent.append("label=\"").append(place.getName())
                    .append("\", ");
22              dotContent.append("pos=\"").append(place.getX()).
                    append(",").append(place.getY()).append("!\"];\n")
                    ;
23          }
24          dotContent.append("\n");
25
26          // Generar nodos para objetos con coordenadas
                espec ficas
27          dotContent.append("  // Objetos en coordenadas
                espec ficas\n");
28          for (MapObject object : world.getObjects()) {
29              if (!object.isAtPlace()) {
30                  String objectId = "obj_" + object.getName().
                        replaceAll("\\s+", "_");
31                  dotContent.append("  \"").append(objectId).append
                        ("\" [");
32                  dotContent.append("shape=").append(object.
                        getShape()).append(", ");
33                  dotContent.append("fillcolor=\"").append(object.
                        getFillColor()).append("\", ");
34                  dotContent.append("label=\"").append(object.
                        getEmoji()).append(" ").append(object.getName
                        ()).append("\", ");
35                  dotContent.append("pos=\"").append(object.getX())
                        .append(",").append(object.getY()).append("
                        !\"];\n");
36              }
37          }
38          dotContent.append("\n");
39
40          // Generar nodos para objetos en lugares
41          dotContent.append("  // Objetos en lugares\n");
42          for (MapObject object : world.getObjects()) {
43              if (object.isAtPlace()) {
```

```
44                  String objectId = "obj_" + object.getName().
                        replaceAll("\\s+", "_");
45                  dotContent.append("  \"").append(objectId).append
                        ("\" [");
46                  dotContent.append("shape=").append(object.
                        getShape()).append(", ");
47                  dotContent.append("fillcolor=\"").append(object.
                        getFillColor()).append("\", ");
48                  dotContent.append("label=\"").append(object.
                        getEmoji()).append(" ").append(object.getName
                        ()).append("\"];\n");
49
50                  // Conexi n entre el objeto y el lugar
51                  dotContent.append("  \"").append(objectId).append
                        ("\" -> \"").append(object.getPlaceId()).
                        append("\" [");
52                  dotContent.append("label=\"en\", dir=none, style=
                        dotted];\n");
53              }
54          }
55          dotContent.append("\n");
56
57          // Generar conexiones entre lugares
58          dotContent.append("  // Conexiones entre lugares\n");
59          for (Connection conn : world.getConnections()) {
60              dotContent.append("  \"").append(conn.getSource()).
                    append("\" -> \"").append(conn.getTarget()).append
                    ("\" [");
61              dotContent.append("label=\"").append(conn.getType()).
                    append("\", ");
62              dotContent.append("color=\"").append(conn.
                    getLineColor()).append("\", ");
63              dotContent.append("style=").append(conn.getLineStyle
                    ()).append("];\n");
64          }
65
66          // Fin del archivo DOT
67          dotContent.append("}\n");
68
69          // Escribir archivo
70          try (FileWriter writer = new FileWriter(filePath)) {
71              writer.write(dotContent.toString());
72          }
73      }
74  }
```

Listing 4: Fragmento de la clase DotGenerator

8. Generación de Reportes

8.1. Reporte de Tokens

El sistema genera un reporte HTML con la lista de tokens encontrados durante el análisis léxico.

```
public String generateTokensReport(ArrayList<Token> tokens,
    ArrayList<CharacterToken> charTokens) {
    StringBuilder html = new StringBuilder();

    // Encabezado HTML
    html.append("<!DOCTYPE html>\n");
    html.append("<html lang=\"es\">\n");
    html.append("<head>\n");
    html.append("  <meta charset=\"UTF-8\">\n");
    html.append("  <meta name=\"viewport\" content=\"width=device
        -width, initial-scale=1.0\">\n");
    html.append("  <title>Reporte de Tokens</title>\n");
    html.append("  <style>\n");
    html.append("    body { font-family: Arial, sans-serif;
        margin: 20px; }\n");
    html.append("    h1, h2 { color: #333; text-align: center; }\
        n");
    html.append("    table { width: 100%; border-collapse:
        collapse; margin-top: 20px; }\n");
    html.append("    th, td { padding: 8px; text-align: left;
        border: 1px solid #ddd; }\n");
    html.append("    th { background-color: #4CAF50; color: white
        ; }\n");
    html.append("    tr:nth-child(even) { background-color: #
        f2f2f2; }\n");
    html.append("    tr:hover { background-color: #ddd; }\n");
    html.append("    .token-count { text-align: center; margin-
        top: 20px; font-weight: bold; }\n");
    html.append("    .char-tokens th { background-color: #2196F3;
         }\n");
    html.append("  </style>\n");
    html.append("</head>\n");
    html.append("<body>\n");
    html.append("  <h1>Reporte de Tokens</h1>\n");

    // Resumen de tokens
    Map<TokenType, Long> tokenCounts = tokens.stream()
            .collect(Collectors.groupingBy(Token::getType,
                Collectors.counting()));

    html.append("  <div class=\"token-count\">Total de tokens: ")
        .append(tokens.size()).append("</div>\n");
    html.append("  <table style=\"width: 50%; margin: 20px auto
        ;\">\n");
    html.append("    <tr>\n");
```

```
33      html.append("      <th>Tipo de Token</th>\n");
34      html.append("      <th>Cantidad</th>\n");
35      html.append("    </tr>\n");
36              });
37
38      html.append("  </table>\n");
39
40      // Tabla de tokens
41      html.append("  <h2>Listado Detallado de Tokens</h2>\n");
42      html.append("  <table>\n");
43      html.append("    <tr>\n");
44      html.append("      <th>#</th>\n");
45      html.append("      <th>Token</th>\n");
46      html.append("      <th>Lexema</th>\n");
47      html.append("      <th>L nea </th>\n");
48      html.append("      <th>Columna</th>\n");
49      html.append("    </tr>\n");
50
51      // Filas de la tabla
52      int counter = 1;
53      for (Token token : tokens) {
54          html.append("    <tr>\n");
55          html.append("      <td>").append(counter++).append("</td
               >\n");
56          html.append("      <td>").append(token.getType().getValue
               ()).append("</td>\n");
57          html.append("      <td>").append(escapeHtml(token.
               getLexeme())).append("</td>\n");
58          html.append("      <td>").append(token.getLine()).append(
               "</td>\n");
59          html.append("      <td>").append(token.getColumn()).
               append("</td>\n");
60          html.append("    </tr>\n");
61      }
62
63      html.append("  </table>\n");
64
65      // Tabla de tokens a nivel de car cter
66      html.append("  <h2>An lisis Car cter por Car cter</h2>\n")
            ;
67      html.append("  <div class=\"token-count\">Total de caracteres
            analizados: ").append(charTokens.size()).append("</div>\n
            ");
68      html.append("  <table class=\"char-tokens\">\n");
69      html.append("    <tr>\n");
70      html.append("      <th>#</th>\n");
71      html.append("      <th>Car cter </th>\n");
72      html.append("      <th>Tipo</th>\n");
73      html.append("      <th>L nea </th>\n");
74      html.append("      <th>Columna</th>\n");
75      html.append("    </tr>\n");
```

```
76
77      // ... resto del c digo para generar el reporte de tokens
78
79      return html.toString();
80  }
```

Listing 5: Fragmento de la clase ReportGenerator para tokens

8.2. Reporte de Errores

El sistema también genera un reporte HTML con los errores léxicos y sintácticos encontrados durante el análisis.

```
1   public String generateErrorsReport(ArrayList<Token> errors) {
2       StringBuilder html = new StringBuilder();
3
4       // Encabezado HTML
5       html.append("<!DOCTYPE html>\n");
6       html.append("<html lang=\"es\">\n");
7       html.append("<head>\n");
8       html.append("  <meta charset=\"UTF-8\">\n");
9       html.append("  <meta name=\"viewport\" content=\"width=device
            -width, initial-scale=1.0\">\n");
10      html.append("  <title>Reporte de Errores</title>\n");
11      html.append("  <style>\n");
12      html.append("    body { font-family: Arial, sans-serif;
            margin: 20px; }\n");
13      html.append("    h1, h2 { color: #333; text-align: center; }\
            n");
14      html.append("    table { width: 100%; border-collapse:
            collapse; margin-top: 20px; }\n");
15      html.append("    th, td { padding: 8px; text-align: left;
            border: 1px solid #ddd; }\n");
16      html.append("    th { background-color: #f44336; color: white
            ; }\n");
17      html.append("    tr:nth-child(even) { background-color: #
            f2f2f2; }\n");
18      html.append("    tr:hover { background-color: #ddd; }\n");
19      html.append("    .lexical th { background-color: #E91E63; }\n
            ");
20      html.append("    .syntactic th { background-color: #9C27B0;
            }\n");
21      html.append("    .error-count { text-align: center; margin-
            top: 20px; font-weight: bold; }\n");
22      html.append("  </style>\n");
23      html.append("</head>\n");
24      html.append("<body>\n");
25      html.append("  <h1>Reporte de Errores</h1>\n");
26
27      // Resumen de errores
28      long lexicalErrors = errors.stream()
```

```
29              .filter(e -> e.getErrorType() == Token.ErrorType.
                    LEXICAL)
30              .count();
31      long syntacticErrors = errors.stream()
32              .filter(e -> e.getErrorType() == Token.ErrorType.
                    SYNTACTIC)
33              .count();
34
35      html.append("  <div class=\"error-count\">Total de errores: "
            ).append(errors.size()).append("</div>\n");
36      html.append("  <table style=\"width: 50%; margin: 20px auto
            ;\">\n");
37      html.append("    <tr>\n");
38      html.append("      <th>Tipo de Error</th>\n");
39      html.append("      <th>Cantidad</th>\n");
40      html.append("    </tr>\n");
41      html.append("    <tr>\n");
42      html.append("      <td>Errores L xicos</td>\n");
43      html.append("      <td>").append(lexicalErrors).append("</td
            >\n");
44      html.append("    </tr>\n");
45      html.append("    <tr>\n");
46      html.append("      <td>Errores Sint cticos</td>\n");
47      html.append("      <td>").append(syntacticErrors).append("</
            td>\n");
48      html.append("    </tr>\n");
49      html.append("  </table>\n");
50
51      // ... resto del c digo para generar el reporte de errores
52
53      return html.toString();
54  }
```

Listing 6: Fragmento de la clase ReportGenerator para errores

9. Diagrama de Clases

A continuación se presenta el diagrama de clases del sistema, mostrando las relaciones entre los componentes principales:

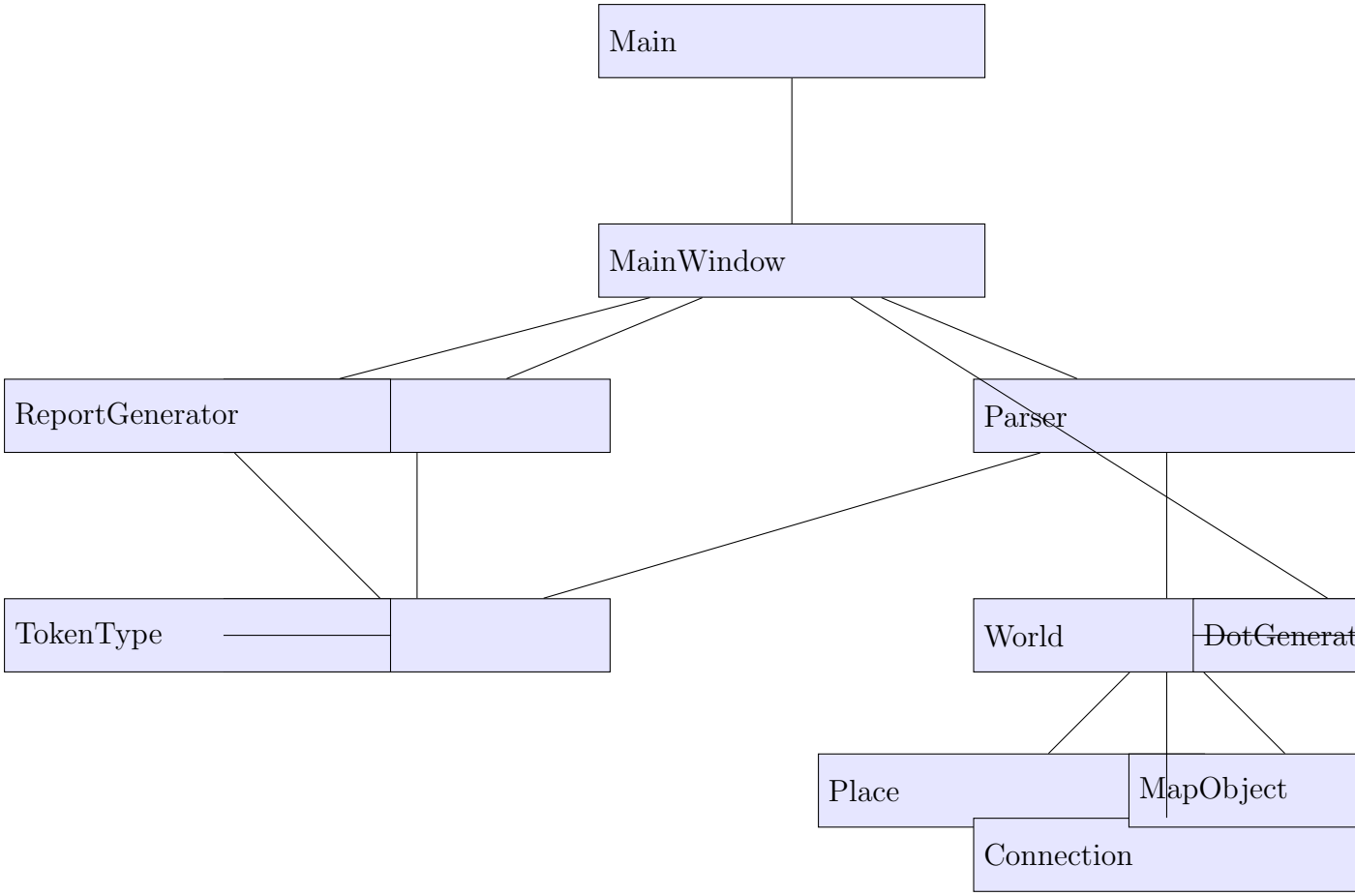

Figura 1: Diagrama de clases del sistema Generador Visual de Mapas Narrativos.

10. Diagrama de Flujo del Sistema

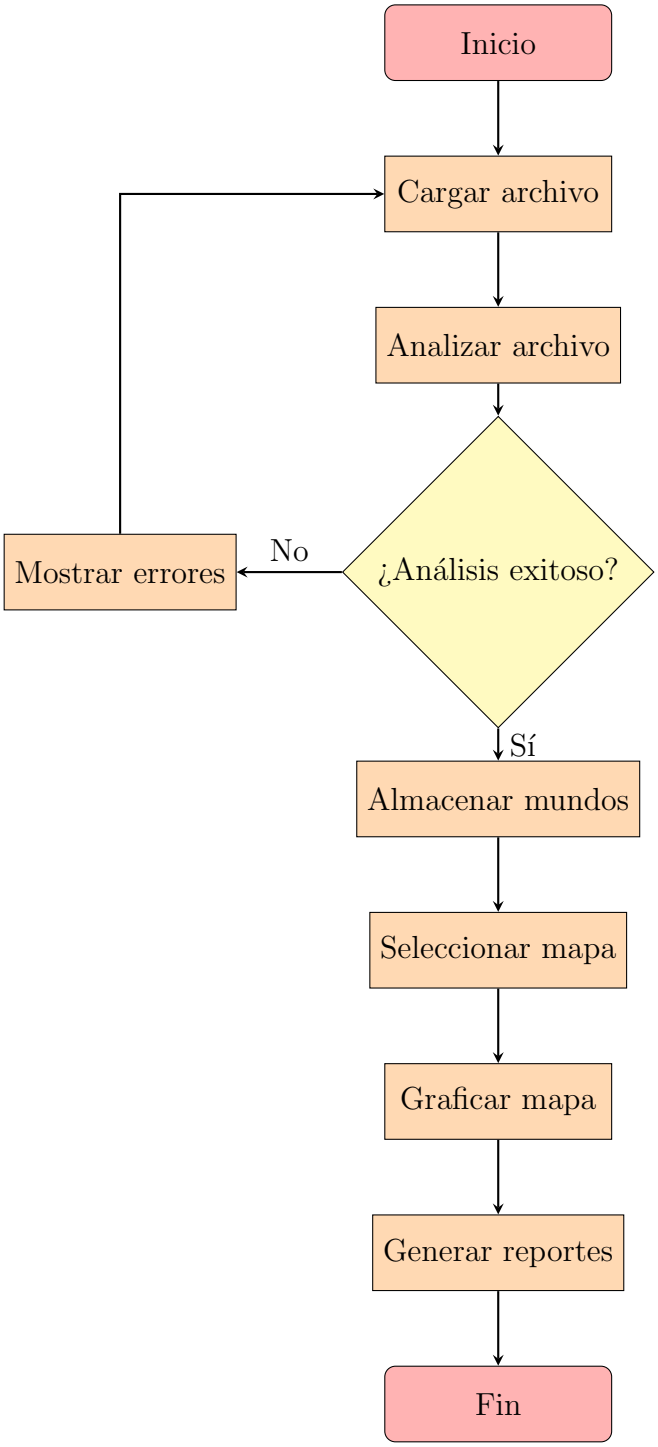

Figura 2: Diagrama de flujo del sistema Generador Visual de Mapas Narrativos.

11. Ejemplo de uso del sistema

11.1. Archivo de entrada

A continuación se muestra un ejemplo de archivo de entrada para el sistema:

```
world "Reino Encantado" {
    place Aldea:pueblo at (0,0)
    place Bosque:jungla at (2,1)
    place Montaña:montaña at (3,3)
    place Lago:río at (1,2)
    place Cueva:cueva at (4,1)

    connect Aldea to Bosque with "camino"
    connect Bosque to Montaña with "sendero"
    connect Aldea to Lago with "puente"
    connect Lago to Cueva with "lancha"
    connect Cueva to Montaña with "teleférico"

    object "Espada Mágica":arma at Cueva
    object "Llave Antigua":llave at (3,2)
    object "Libro de Hechizos":libro at Aldea
    object "Gema Brillante":gema at Montaña
},
world "Isla del Tesoro" {
    place Playa:playa at (0,0)
    place Selva:jungla at (1,1)
    place Cima:montaña at (3,2)
    place Caverna:cueva at (2,1)
    place Embarcadero:pueblo at (0,2)

    connect Playa to Selva with "sendero"
    connect Selva to Caverna with "camino"
    connect Caverna to Cima with "escalera"
    connect Embarcadero to Playa with "lancha"

    object "Cofre del Tesoro":tesoro at Caverna
    object "Mapa Antiguo":libro at Embarcadero
    object "Llave Dorada":llave at (2,2)
    object "Poción Curativa":poción at Selva
}
```

11.2. Visualización generada

El sistema procesará este archivo y generará visualizaciones gráficas como la siguiente:

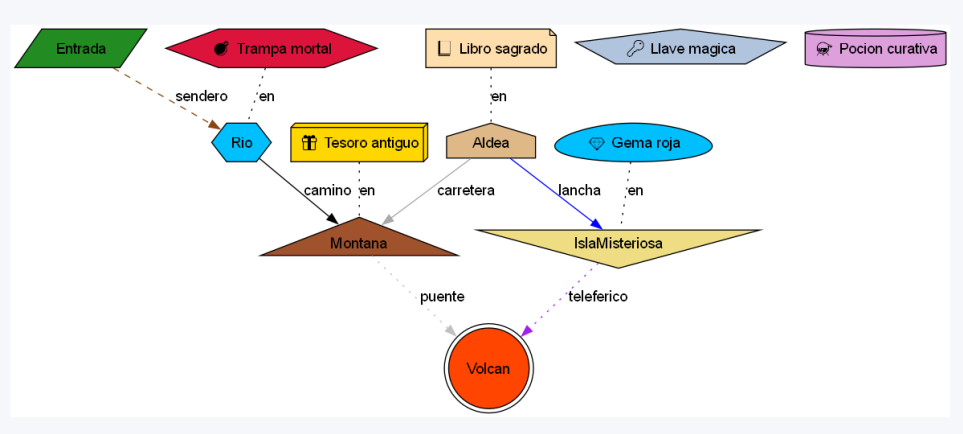

Figura 3: Ejemplo de mapa generado por el sistema.

12. Conclusiones

El desarrollo del Generador Visual de Mapas Narrativos ha permitido aplicar conceptos fundamentales de análisis léxico y sintáctico para el procesamiento de un lenguaje específico. Se ha implementado un sistema completo que permite:

- Realizar análisis léxico para identificar tokens en el archivo de entrada.
- Realizar análisis sintáctico para construir la estructura del mapa narrativo.
- Generar visualizaciones gráficas de los mapas utilizando Graphviz.
- Generar reportes detallados de tokens y errores.

El sistema demuestra la aplicación práctica de los conocimientos adquiridos en el curso de Lenguajes Formales y de Programación, permitiendo convertir descripciones textuales en representaciones visuales intuitivas.

13. Anexos

13.1. Tokens y Expresiones Regulares

13.2. Tabla Detallada de Tokens

13.3. Método de Construcción del Árbol de Análisis

13.4. Método Detallado de Construcción del Árbol de Análisis Sintáctico

13.4.1. Descripción General

El método de construcción del árbol de análisis sintáctico sigue un enfoque de análisis descendente (top-down parsing), donde se parte de la regla de producción más general (`<programa>`) y se descompone recursivamente en sus componentes.

Cuadro 2: Descripción Detallada de Tokens

Token	Expresión Regular	Descripción	Ejemplo
WORLD	`"world"`	Palabra reservada para definir mundo	`world "Reino"`
PLACE	`"place"`	Palabra reservada para definir lugar	`place Castillo`
CONNECT	`"connect"`	Palabra reservada para conexión	`connect A to B`
OBJECT	`"object"`	Palabra reservada para objeto	`object "Espada"`
AT	`"at"`	Indica posicionamiento	`at (0,0)`
TO	`"to"`	Indica destino	`to Castillo`
WITH	`"with"`	Indica tipo de conexión	`with "camino"`
IDENTIFIER	`[a-zA-Z][a-zA-Z0-9]*`	Nombre de lugares, objetos	`Castillo, Lugar1`
STRING	`".*"`	Cadena descriptiva	`"Reino Encantado`
NUMBER	`[0-9]+`	Coordenadas numéricas	`0, 123`
LBRACE	`"{"`	Llave de apertura	`{`
RBRACE	`"}"`	Llave de cierre	`}`
LPAREN	`"("`	Paréntesis de apertura	`(`
RPAREN	`")"`	Paréntesis de cierre	`)`
COMMA	`","`	Separador	`,`
COLON	`":"`	Delimitador	`:`
EOF	N/A	Fin de archivo	N/A
ERROR	N/A	Token no reconocido	`@`

13.4.2. Proceso de Construcción

El proceso de construcción del árbol de análisis sintáctico se realiza mediante las siguientes etapas:

1. **Inicialización del Árbol:**
   - Se crea un nodo raíz etiquetado como `<programa>`
   - Este nodo será el punto de entrada de toda la estructura
2. **Análisis de Mundos:**
   - Se recorren todos los tokens de entrada
   - Por cada token `"world"`, se crea un nodo hijo `<mundo>`
   - Cada nodo `<mundo>` se descompone en sus componentes:
     - Nodo de nombre del mundo
     - Nodo de `<lista_lugares>`
     - Nodo de `<lista_conexiones>`
     - Nodo de `<lista_objetos>`
3. **Análisis de Componentes:**
   - `<lista_lugares>`: Cada lugar genera un nodo con sus atributos
   - `<lista_conexiones>`: Cada conexión genera un nodo con origen, destino y tipo
   - `<lista_objetos>`: Cada objeto genera un nodo con nombre, tipo y ubicación

4. **Manejo de Errores:**
    - Si se encuentran tokens inesperados, se generan nodos de error
    - Los errores se registran para su posterior análisis y reporte

13.4.3. Ejemplo de Estructura de Árbol

```
<programa>
 <mundo>
    "world"
    "Reino Encantado"
    <lista_lugares>
       <lugar>
           "place"
           "Castillo"
           "pueblo"
           <coordenadas>
               0
               0
    <lista_conexiones>
       <conexion>
           "connect"
           "Castillo"
           "Bosque"
           "camino"
    <lista_objetos>
        <objeto>
            "object"
            "Espada"
            "arma"
            "Castillo"
 ...

\subsubsection{Beneficios del Método}
\begin{itemize}
    \item Descomposición jerárquica del lenguaje
    \item Identificación clara de la estructura sintáctica
    \item Facilita la detección temprana de errores
    \item Permite una representación visual de la estructura del código
\end{itemize}
\begin{figure}[h]
\centering
\begin{verbatim}
función construirArbol(tokens):
    raiz = nuevoNodo(<programa>)
    mientras hayTokens():
        mundo = parsearMundo()
        raiz.agregarHijo(mundo)
    return raiz
```

```
función parsearMundo():
    nodoMundo = nuevoNodo(<mundo>)
    consumir("world")
    consumir(STRING)
    consumir("{")

    nodoMundo.agregarHijo(parsearLugares())
    nodoMundo.agregarHijo(parsearConexiones())
    nodoMundo.agregarHijo(parsearObjetos())

    consumir("}")
    return nodoMundo
```

13.5. Gramática Libre de Contexto

```
<programa>          ::= <mundo>+

<mundo>             ::= "world" STRING "{" <lista_lugares>
                        <lista_conexiones> <lista_objetos> "}"

<lista_lugares>   ::= <lugar>*
<lugar>           ::= "place" IDENTIFIER ":" IDENTIFIER
                        "at" "(" NUMBER "," NUMBER ")"

<lista_conexiones>::= <conexion>*
<conexion>        ::= "connect" IDENTIFIER "to" IDENTIFIER
                        "with" STRING

<lista_objetos>   ::= <objeto>*
<objeto>          ::= "object" STRING ":" IDENTIFIER "at" IDENTIFIER
                    | "object" STRING ":" IDENTIFIER
                      "at" "(" NUMBER "," NUMBER ")"
```

13.6. Gramática Libre de Contexto para Generador de Mapas Narrativos

13.6.1. Definición Formal de la Gramática

1. Símbolos Terminales (T):

- Palabras Reservadas: `world`, `place`, `connect`, `object`, `at`, `to`, `with`
- Literales: `IDENTIFIER`, `STRING`, `NUMBER`
- Símbolos: `{` `}` `(` `)` `:` `,`

2. Símbolos No Terminales (N):

- `<programa>`

- <mundo>
- <lista_lugares>
- <lugar>
- <lista_conexiones>
- <conexion>
- <lista_objetos>
- <objeto>
- <coordenadas>

3. Símbolo Inicial: <programa>
4. Conjunto de Producciones (P):

1. <programa> → <mundo>+
2. <mundo> → "world" STRING "{" <lista_lugares> <lista_conexiones> <lista_objetos> "
3. <lista_lugares> → <lugar> <lista_lugares> —
4. <lugar> → "place" IDENTIFIER ":" IDENTIFIER "at" <coordenadas>
5. <coordenadas> → "(" NUMBER "," NUMBER ")"
6. <lista_conexiones> → <conexion> <lista_conexiones> —
7. <conexion> → "connect" IDENTIFIER "to" IDENTIFIER "with" STRING
8. <lista_objetos> → <objeto> <lista_objetos> —
9. <objeto> → "object" STRING ":" IDENTIFIER "at" (IDENTIFIER ¡coordenadas¿)—

13.6.2. Explicación de las Producciones

- Producción 1: Un programa consta de uno o más mundos
- Producción 2: Un mundo tiene un nombre, lugares, conexiones y objetos
- Producción 3: Lista de lugares puede ser vacía o contener múltiples lugares
- Producción 4: Un lugar tiene nombre, tipo y coordenadas
- Producción 5: Coordenadas son un par de números entre paréntesis
- Producción 6: Lista de conexiones puede ser vacía o contener múltiples conexiones
- Producción 7: Conexión une dos lugares con un tipo de camino
- Producción 8: Lista de objetos puede ser vacía o contener múltiples objetos
- Producción 9: Un objeto puede estar en un lugar o en coordenadas específicas

13.6.3. Propiedades de la Gramática

- **Tipo:**Gramática Libre de Contexto (Tipo 2 en la Jerarquía de Chomsky)
- **Recursividad:** Permite múltiples mundos, lugares, conexiones y objetos
- **Derivación:** Análisis descendente (top-down parsing)

13.7. Ejemplo de Derivación

Entrada:

```
world "Reino" {
    place Castillo:pueblo at (0,0)
}
```

Derivación:

1. $< programa > \rightarrow < mundo >$
2. $< mundo > \rightarrow$ "world"STRING "{"$< lista\_lugares > < lista\_conexiones > < lista\_objetos >$ "}"
3. $< lista\_lugares > \rightarrow < lugar >$
4. $< lugar > \rightarrow$ "placeÏDENTIFIER ":ÏDENTIFIER .ªt("NUMBER ","NUMBER ")"
5. $< lista\_conexiones > \rightarrow \varepsilon$ (vacío)
6. $< lista\_objetos > \rightarrow \varepsilon$ (vacío)

13.8. Glosario de Términos

Token Unidad mínima con significado en el lenguaje

Expresión Regular Patrón que describe un conjunto de cadenas de texto

Árbol de Análisis Sintáctico Representación jerárquica de la estructura gramatical de un programa

Gramática Libre de Contexto Método formal para describir la estructura de un lenguaje

Parsing Proceso de análisis sintáctico que descompone un texto en sus componentes gramaticales

14. Referencias

- Oracle. (2023). Java Documentation. `https://docs.oracle.com/en/java/`
- Graphviz. (2023). Graphviz Documentation. `https://graphviz.org/documentation/`
- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley.
- Campos González, V. D. (2025). Material del curso Lenguajes Formales y de Programación. Universidad de San Carlos de Guatemala.