Requerimientos del sistema operativo:
Para el desarrollo de este programa se necesito con las siguiente características:
ProcesadorIntel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
RAM instalada 16.0 GB (15.8 GB utilizable)
Tipo de sistema Sistema operativo de 64 bits, procesador x64
Windows 11 última versión.
Debian GNU/Linux trixie/sid x86_64

Para las dependencias instaladas se utilizó:
Jflex
Cup
itextpdf
graphviz

Librerías generales:
```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JToolBar;
import javax.swing.filechooser.FileNameExtensionFilter;
```

Gramática del analizador léxico:

```
package Gramaticas;
import java_cup.runtime.*;
import Gramaticas.sym;
   import java.util.HashMap;
   import java.util.Map;
%%
%{
   private Symbol symbol(int type) {
      return new Symbol(type, yyline + 1, yycolumn + 1);
   }
   private Symbol symbol(int type, Object value) {
      return new Symbol(type, yyline + 1, yycolumn + 1, value);
   }
   private LinkedList<String> listaErrores;
public static Map<String, List<OperadorInfo>> operadoresInfo = new HashMap<>();
%}
%init{
   yyline = 1;
   yycolumn = 1;
   listaErrores = new LinkedList<>();
   yybegin(YYINITIAL);
%init}
%{
   private void addOperadorInfo(String operador, int linea, int columna) {
      operadoresInfo.computeIfAbsent(operador, k -> new ArrayList<>())
                    .add(new OperadorInfo(linea, columna, operador + " " + linea + ":" + columna));
   }
%}

%class AnalizadorLexico
%public
%unicode //define el conjunto de caracteres con los qiue trabajará el scanner
%standalone
%cup
%line
%column
%ignorecase

SaltosLinea     = \r|\n|\r\n
EspaciosBlancos = {SaltosLinea} | [ \t\f]
Identificador   = [a-zA-Z][a-zA-Z0-9_]*
Enteros         = 0 | ([1-9][0-9]*)
Decimal         = {Enteros}\.[0-9]+
Comentario      = "//".*
NumeroNegativo  =-({Enteros}|{Decimal})

%%
```

```
<YYINITIAL> {

//graficos
   "graficar"     { return symbol(sym.GRAFICAR); }
   "circulo"      { return symbol(sym.CIRCULO); }
   "cuadrado"     { return symbol(sym.CUADRADO); }
   "rectangulo"   { return symbol(sym.RECTANGULO); }
   "linea"        { return symbol(sym.LINEA); }
   "poligono"     { return symbol(sym.POLIGONO); }
   "animar"       { return symbol(sym.ANIMAR); }
   "objeto"       { return symbol(sym.OBJETO); }
   "anterior"     { return symbol(sym.ANTERIOR); }

 //colores
   "azul"         { return symbol(sym.COLOR, yytext()); }
   "rojo"         { return symbol(sym.COLOR, yytext()); }
   "amarillo"     { return symbol(sym.COLOR, yytext()); }
   "verde"        { return symbol(sym.COLOR, yytext()); }

//mis colores
   "morado"       { return symbol(sym.COLOR, yytext()); }
   "cafe"         { return symbol(sym.COLOR, yytext()); }
   "naranja"      { return symbol(sym.COLOR, yytext()); }
   "rosado"       { return symbol(sym.COLOR, yytext()); }
   "celeste"      { return symbol(sym.COLOR, yytext()); }

//animados
   "lineal"       { return symbol(sym.ANIM_TIPO_LINEAL, yytext()); }
   "curva"        { return symbol(sym.ANIM_TIPO_CURVA, yytext()); }

//OPERADORES
   "+"            { return symbol(sym.SUMA); }
   "-"            { return symbol(sym.RESTA); }
   "*"            { return symbol(sym.MULTIPLICACION); }
   "/"            { return symbol(sym.DIVISION); }
   "("            { return symbol(sym.PARENTESIS_ABIERTO); }
   ")"            { return symbol(sym.PARENTESIS_CERRADO); }
   ","            { return symbol(sym.COMA); }
   "="            { return symbol(sym.IGUAL); }
   ";"            { return symbol(sym.PUNTO_COMA); }
   "."            { return symbol(sym.PUNTO); }

 // Posiciones
   "posx"         { return symbol(sym.POSX); }
   "posy"         { return symbol(sym.POSY); }
   "radio"        { return symbol(sym.RADIO); }
   "ancho"        { return symbol(sym.ANCHO); }
   "alto"         { return symbol(sym.ALTO); }

   {Identificador} { return symbol(sym.IDENTIFICADOR, yytext()); }
   {Enteros}       { return symbol(sym.ENTEROS, Integer.parseInt(yytext())); }
   {Decimal}       { return symbol(sym.DECIMAL, Double.parseDouble(yytext())); }
```

```
    {EspaciosBlancos} { /* ignorar */ }

    {Comentario}     { /* ignore comments */ }
    {NumeroNegativo} { return symbol(sym.NUMERO, Double.parseDouble(yytext())); }

{EspaciosBlancos} { /* ignorar */ }
<<EOF>> { return symbol(sym.EOF); }

   [^] {
        String errorMsg = "Error lexico: Caracter invalido <" + yytext() + "> en linea " + (yyline+1) +
", columna " + (yycolumn+1);
        listaErrores.add(errorMsg);
        System.out.println(errorMsg);
    }
}
"+" {
    addOperadorInfo("+", yyline + 1, yycolumn + 1);
    return symbol(sym.SUMA);
}
"-" {
    addOperadorInfo("-", yyline + 1, yycolumn + 1);
    return symbol(sym.RESTA);
}
"*" {
    addOperadorInfo("*", yyline + 1, yycolumn + 1);
    return symbol(sym.MULTIPLICACION);
}
"/" {
    addOperadorInfo("/", yyline + 1, yycolumn + 1);
    return symbol(sym.DIVISION);
}
```

Gramática analizador sintáctico:

```
package Gramaticas;
import java_cup.runtime.*;
import java.util.ArrayList;
import java.awt.Color;
import Objetos.*;
parser code {:
    public static ArrayList<Figura> figuras = new ArrayList<>();
    public static ArrayList<Animacion> animaciones = new ArrayList<>();
    public static ArrayList<String> errores = new ArrayList<>();

    public void syntax_error(Symbol s) {
        errores.add("Error sintáctico en línea " + (s.left + 1) + ", columna " + (s.right + 1) + ": " +
s.value);
    }

    public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception {
        String errorMsg = "Error sintáctico irrecuperable en línea " + (s.left + 1) + ", columna " +
(s.right + 1) + ": " + s.value;
        errores.add(errorMsg);
        System.out.println(errorMsg);
    }
    public static ArrayList<String> getErrores() {
        return errores;
    }
  public static Map<String, List<OperadorInfo>> getOperadoresInfo() {
        return AnalizadorLexico.operadoresInfo;
    }
:}

action code {:
    private Color getColor(String colorName) {
        switch (colorName.toLowerCase()) {
            case "azul": return Color.BLUE;
            case "rojo": return Color.RED;
            case "amarillo": return Color.YELLOW;
            case "verde": return Color.GREEN;
            case "morado": return new Color(128, 0, 128);
            case "cafe": return new Color(139, 69, 19);
            case "naranja": return Color.ORANGE;
            case "rosado": return Color.PINK;
            case "celeste": return new Color(135, 206, 235);
            default: return Color.BLACK;
        }
    }
:}

// terminales: parte del alfabeto o lenguaje de nuestro programa
terminal GRAFICAR, CIRCULO, CUADRADO, RECTANGULO, LINEA, POLIGONO,
ANIMAR, OBJETO, ANTERIOR, OBJETO_ANTERIOR;
terminal String COLOR, ANIM_TIPO_LINEAL, ANIM_TIPO_CURVA;
```

```
terminal SUMA, RESTA, MULTIPLICACION, DIVISION, PARENTESIS_ABIERTO,
PARENTESIS_CERRADO, COMA, IGUAL, PUNTO, PUNTO_COMA;
terminal String IDENTIFICADOR;
terminal Integer ENTEROS;
terminal Double DECIMAL, NUMERO;
terminal POSX, POSY, RADIO, ANCHO, ALTO;

// no terminales:
non terminal ArrayList<Figura> program;
non terminal Figura figure;
non terminal statement, animation;
non terminal Double expression;

// precedencia: orden gera
precedence left SUMA, RESTA;
precedence left MULTIPLICACION, DIVISION;
precedence left PARENTESIS_ABIERTO, PARENTESIS_CERRADO;

// espero que si sea la gramatica funcional
start with program;

program ::= statement program
        | statement
        {: RESULT = parser.figuras; :}
        ;


statement ::= figure:f
          {: parser.figuras.add(f); :}
          | animation
          ;

figure ::= GRAFICAR CIRCULO PARENTESIS_ABIERTO IDENTIFICADOR:id COMA
expression:x COMA expression:y COMA  expression:r COMA COLOR:c
PARENTESIS_CERRADO
       {: RESULT = new Circulo(id, (int)Math.round(x), (int)Math.round(y), (int)Math.round(r),
getColor(c)); :}
       | GRAFICAR CUADRADO PARENTESIS_ABIERTO IDENTIFICADOR:id COMA
expression:x COMA expression:y COMA expression:l COMA COLOR:c
PARENTESIS_CERRADO
       {: RESULT = new Cuadrado(id, (int)Math.round(x), (int)Math.round(y), (int)Math.round(l),
getColor(c)); :}
       | GRAFICAR RECTANGULO PARENTESIS_ABIERTO IDENTIFICADOR:id COMA
expression:x COMA expression:y COMA expression:w COMA expression:h COMA COLOR:c
PARENTESIS_CERRADO
       {: RESULT = new Rectangulo(id, (int)Math.round(x), (int)Math.round(y),
(int)Math.round(w), (int)Math.round(h), getColor(c)); :}
       | GRAFICAR LINEA PARENTESIS_ABIERTO IDENTIFICADOR:id COMA expression:x1
COMA expression:y1 COMA expression:x2 COMA expression:y2 COMA COLOR:c
PARENTESIS_CERRADO
       {: RESULT = new Linea(id, (int)Math.round(x1), (int)Math.round(y1), (int)Math.round(x2),
(int)Math.round(y2), getColor(c)); :}
```

```
        | GRAFICAR POLIGONO PARENTESIS_ABIERTO IDENTIFICADOR:id COMA
expression:x COMA expression:y COMA expression:s COMA expression:w COMA expression:h
COMA COLOR:c PARENTESIS_CERRADO
        {: RESULT = new Poligono(id, (int)Math.round(x), (int)Math.round(y), (int)Math.round(s),
(int)Math.round(w), (int)Math.round(h), getColor(c)); :}
        ;

animation ::= ANIMAR OBJETO ANTERIOR PARENTESIS_ABIERTO ANIM_TIPO_LINEAL:t
COMA expression:x COMA expression:y COMA expression:o PARENTESIS_CERRADO
          {: RESULT = new Animacion(null, "lineal", (int)Math.round(x), (int)Math.round(y),
(int)Math.round(o)); :}
          | ANIMAR OBJETO ANTERIOR PARENTESIS_ABIERTO ANIM_TIPO_CURVA:t
COMA expression:x COMA expression:y COMA expression:o PARENTESIS_CERRADO
          {: RESULT = new Animacion(null, "curva", (int)Math.round(x), (int)Math.round(y),
(int)Math.round(o)); :}
          ;

expression ::= NUMERO:n
           {: RESULT = n; :}
           | ENTEROS:e {: RESULT = e.doubleValue(); :}
           | DECIMAL:d {: RESULT = d; :}
           | expression:e1 SUMA expression:e2 {: RESULT = e1 + e2; :}
           | expression:e1 RESTA expression:e2 {: RESULT = e1 - e2; :}
           | expression:e1 MULTIPLICACION expression:e2 {: RESULT = e1 * e2; :}
           | expression:e1 DIVISION expression:e2 {: RESULT = e1 / e2; :}
           | PARENTESIS_ABIERTO expression:e PARENTESIS_CERRADO {: RESULT = e; :}
           ;
```