## Portada

**Informe del Proyecto Pixel-Wall-E**

Autor: Lianet Tamarit Tejas

## 1. Introducción

* Proyecto académico de Ciencias de la Computación.
* Objetivo: desarrollar un intérprete de lenguaje personalizado que controle a Wall-E para dibujar en un canvas pixelado.

## 2. Arquitectura general

### 2.1 Visión global de los componentes

* **Lexer:** Convierte el código fuente en tokens.
* **Parser:** Toma los tokens y construye un AST (Árbol de Sintaxis Abstracta).
* **Evaluador:** Recorre el AST aplicando instrucciones sobre el estado (canvas).
* **Manejo de errores:** Captura y reporta errores léxicos, sintácticos y de ejecución.
* **WallState:** Representa el estado actual del canvas, posición y brocha.

### 2.2 Flujo de datos entre componentes

1. El usuario escribe código en el editor.
2. El **Lexer** produce una lista de tokens.
3. El **Parser** analiza tokens y genera un AST.
4. El **Evaluador**, usando el patrón Visitor, ejecuta el AST sobre el **WallState**.
5. Los cambios se reflejan en el canvas gráfico.

## 3. Componente Lexer

### 3.1 Propósito

* Analizar la cadena de entrada y clasificar lexemas en categorías (identificadores, números, operadores, palabras clave).

### 3.2 Estructura y TokenGeneral

* TokenTypeExtensions.TokenGeneral: Diccionario que mapea cada TokenType a su expresión regular.
* Incluye patrones para literales, operadores aritméticos (+,-,\*,/,\*\*,%,%), comparadores (<,>,<=,>=,==), símbolos ((,), [ ], ,) y palabras clave (GoTo).
* TokenTypeExtensions.ColorValue y Instructions definen valores válidos para color e instrucciones.

### 3.3 Clase Token

* **Type (TokenType)**: Categoría del token.
* **Lexeme (string)**: Texto reconocido.
* **Line (int)**: Número de línea en el código.
* **Position (int)**: Desplazamiento en la línea.

### 3.4 Proceso de tokenización (Lexer.Tokenize())

1. Recorre Input carácter a carácter.
2. Si encuentra espacio, actualiza Line y añade EOL.
3. Para cada posición, itera sobre TokenGeneral: aplica Regex y crea Token si hay coincidencia.
4. Avanza Position según longitud del lexema.
5. Si ningún patrón coincide, registra ErrorException por símbolo inválido.
6. Al finalizar, agrega TokenType.EOF.

### 3.5 Manejo de errores léxicos

* Lexer.Errores: Lista de ErrorException con mensaje, línea y posición.
* Al detectar símbolo no reconocido, crea y añade un ErrorException, luego continúa el análisis.

## 4. Componente Parser

### 4.1 Propósito

* Validar la secuencia de tokens y construir el AST.
* Identificar y reportar errores sintácticos y etiquetas mal referenciadas.

### 4.2 Estructura de la carpeta de AST

* Clases derivadas de ASTNode para cada instrucción:
  + Instrucciones: InstructionNode, GoTo, Label, Assignment, etc.
  + Expresiones: BinaryAritmethic, BinaryBoolean, UnaryExpression, GroupingExpr, FunctionCallNode, Number, StringNode, Variable.
* Cada nodo almacena OriginToken para rastreo de errores.

### 4.3 Métodos clave del Parser

* Parser.Parsind(): Punto de entrada. Verifica el primer token Spawn, itera construyendo nodos hasta EOF, sincroniza tras errores y valida etiquetas.
* ParseGenericInstruction(): Consume identificador, paréntesis, parámetros (Term()), genera InstructionNode o lanza ErrorException si inválido o Spawn repetido.
* ParseGoTo(): Procesa GoTo [etiqueta] (condición), cuenta referencias en gotoReferences.
* LabelParse(): Registra etiquetas en diccionario labels, evita duplicados.
* AssignmentParse(): Interpreta asignaciones de la forma identificador <- expresión.
* Análisis de expresiones: métodos encadenados Or(), And(), Equality(), Comparison(), Term(), Factor(), Pow(), Unary(), Primary().
* ValidateLabelsAndGoTos(): Comprueba que cada GoTo tenga etiqueta y marca uso; reporta etiquetas no referenciadas.
* Synchronize(): Avanza hasta EOL o EOF tras error para continuar parsing.

### 4.4 Manejo de errores sintácticos

* Parser.Errores: Lista de ErrorException con detalles.
* Cada método Consume(...), ConsumeEOL() lanza ErrorException si el token no coincide con lo esperado.

## 5. Componente Evaluador

### 5.1 Aplicación del patrón Visitor

* Evaluate implementa IVisitor<object>, recorre AST mediante Accept.
* EvaluateProgram(): Recorre lista de ASTNode, maneja GoTo y ejecuta otros nodos.

### 5.2 Lógica de evaluación

* Visit(InstructionNode): Ejecuta cada instrucción (Spawn, Color, Size, DrawLine, etc.), valida tipos con ExpectTypes(...), maneja excepciones de coordenadas.
* Visit(...) para variables, asignaciones, expresiones aritméticas y booleanas, llamadas a funciones (GetActualX, GetColorCount, etc.).
* ExpectTypes(): Verifica número y tipo de argumentos, lanza ErrorException si difieren.

### 5.3 Manejo de errores en tiempo de ejecución

* ErroresEvaluacion: Lista de ErrorException capturados durante la ejecución.
* Captura condiciones como dibujo fuera del canvas o tipos inválidos, y acumula mensajes. Componente Parser ### 4.1 Propósito
* Validar la secuencia de tokens y construir el AST.
* Detectar errores sintácticos y etiquetas mal referenciadas.

### 4.2 Estructura de la carpeta de AST

* Clases derivadas de ASTNode para cada instrucción:
  + SpawnNode, ColorNode, DrawLineNode, etc.
  + AssignmentNode, GoToNode, LabelNode, etc.
* Cada nodo almacena información relevante (argumentos, posición).

## 5. Componente Evaluador

### 5.1 Aplicación del patrón Visitor

* Implementa una clase Evaluator que recorre el AST.
* Cada ASTNode acepta un visitante y llama al método específico (VisitDrawLine, VisitSpawn, etc.).

### 5.2 Flujo de ejecución

1. El Evaluator recibe el nodo raíz del AST.
2. Visita cada instrucción en orden.
3. Para cada nodo, invoca el método correspondiente y modifica el WallState.
4. En caso de GoTo, salta a la etiqueta indicada si la condición se cumple.
5. El canvas se actualiza en tiempo real mediante callbacks.

## 6. Manejo de Errores

### 6.1 Clase ErrorException

* Hereda de Exception.
* **Mensaje:** Descripción del error.
* **Línea:** Línea donde ocurrió.
* **Posición:** Posición dentro de la línea.
* Construye el mensaje completo: "<mensaje> (en línea <línea>, posición <posición>)".

### 6.2 Estrategia de captura y reporte

* El **Lexer** lanza errores por patrones inválidos.
* El **Parser** atrapa ErrorException, almacena en lista y sincroniza para continuar parsing.
* El **Evaluador** propaga excepciones para errores en tiempo de ejecución.
* Al final, se muestra al usuario la lista completa de errores.

## 7. Clase WallState

### 7.1 Representación del estado del canvas

* Matriz bidimensional string[,] Canvas con color de cada píxel.
* Tamaño del canvas configurable en el constructor.

### 7.2 Propiedades

* X, Y: Coordenadas actuales de Wall-E.
* BrushColor: Color activo.
* BrushSize: Grosor de la brocha.
* OnPixelPaint: Callback para actualizar la interfaz gráfica.

### 7.3 Métodos de dibujo

* SetPixel(x, y): Pinta un píxel con validación de límites.
* SetBrushPixels(px, py): Pinta un bloque según BrushSize.
* DrawLine(dx, dy, length): Dibuja línea usando desplazamientos.
* DrawRectangle(...): Dibuja rectángulo centrado en posición final.
* DrawCircle(...): Dibuja circunferencia usando algoritmo Midpoint.
* FillFrom(x, y, targetColor): Relleno recursivo tipo “cubeta”.
* IsInside(x, y): Valida coordenadas y lanza excepción si sale del canvas.