# Informe del Proyecto Pixel Wall-E

**Asignatura:** Programación  
**Curso:** 2024-2025  
**Institución:** MATCOM

## 1. Introducción

### 1.1 Descripción del Proyecto

El proyecto Pixel Wall-E consiste en el desarrollo de un compilador e intérprete para un lenguaje de programación especializado en pixel art. El sistema permite controlar un robot virtual llamado Wall-E que se mueve sobre un canvas cuadrado, ejecutando comandos de dibujo para crear arte pixel por pixel.

### 1.2 Objetivos

* **Objetivo Principal:** Implementar un compilador completo que incluya análisis léxico, sintáctico, semántico e interpretación de código.
* **Objetivos Específicos:**
  + Crear una interfaz gráfica intuitiva para edición y visualización
  + Implementar todas las instrucciones del lenguaje especificado
  + Desarrollar un sistema de depuración paso a paso
  + Manejar errores de manera robusta en todas las fases de compilación

### 1.3 Alcance

El proyecto abarca desde el análisis de código fuente hasta la generación visual de los resultados, incluyendo un editor de texto con numeración de líneas, un canvas interactivo y herramientas de depuración avanzadas.

## 2. Marco Teórico

### 2.1 Conceptos de Compiladores

El proyecto se basa en los fundamentos teóricos de construcción de compiladores:

* **Análisis Léxico:** Conversión del código fuente en tokens significativos
* **Análisis Sintáctico:** Construcción del Árbol de Sintaxis Abstracta (AST)
* **Análisis Semántico:** Verificación de reglas de contexto y tipos
* **Interpretación:** Ejecución directa del código validado

### 2.2 Arquitectura del Compilador

La arquitectura sigue el modelo clásico de compilador en fases:

Código Fuente → Lexer → Parser → Semantic Analyzer → Interpreter → Canvas

### 2.3 Tecnologías Utilizadas

* **Lenguaje:** C# con ASP.NET Core
* **Frontend:** HTML5, CSS3, JavaScript (Vanilla)
* **Patrón de Diseño:** MVC (Model-View-Controller)
* **Algoritmos:** Bresenham (círculos), Flood Fill (relleno)

## 3. Diseño e Implementación

### 3.1 Arquitectura del Sistema

#### 3.1.1 Componentes Principales

**Lexer** (Lexer.cs)

* + Análisis léxico mediante expresiones regulares
  + Reconocimiento de tokens: números, identificadores, operadores, palabras clave
  + Manejo de errores léxicos con reportes detallados

**Parser** (Parser.cs)

* + Análisis sintáctico descendente recursivo
  + Construcción del AST siguiendo la gramática del lenguaje
  + Manejo de precedencia de operadores

**Semantic Analyzer** (SemanticAnalyzer.cs)

* + Verificación de tipos y contexto
  + Validación de declaraciones de variables y etiquetas
  + Verificación de límites y restricciones del lenguaje

**Interpreter** (Interpreter.cs)

* + Ejecución directa del AST
  + Manejo del estado de Wall-E (posición, color, tamaño de pincel)
  + Implementación de todas las instrucciones de dibujo

#### 3.1.2 Estructura del AST

ASTNode (abstracto)

├── ProgramNode

├── Statement (abstracto)

│ ├── AssignmentNode

│ ├── LabelNode

│ ├── Instruction

│ └── FunctionCallStatement

└── ExpressionNode (abstracto)

├── ArithmeticExpression

├── BooleanExpression

└── StringExpression

### 3.2 Características Implementadas

#### 3.2.1 Instrucciones Básicas

* **Spawn(x, y):** Posicionamiento inicial de Wall-E
* **Color(string):** Cambio de color del pincel
* **Size(int):** Modificación del tamaño del pincel
* **DrawLine(dirX, dirY, distance):** Dibujo de líneas direccionales
* **DrawCircle(dirX, dirY, radius):** Dibujo de círculos
* **DrawRectangle(dirX, dirY, distance, width, height):** Dibujo de rectángulos
* **Fill():** Relleno por inundación (flood fill)

#### 3.2.2 Sistema de Variables y Expresiones

* Soporte para variables numéricas y booleanas
* Operaciones aritméticas: +, -, \*, /, \*\*, %
* Operaciones lógicas: &&, ||
* Operaciones de comparación: ==, >=, <=, >, <

#### 3.2.3 Control de Flujo

* Etiquetas para marcado de posiciones en código
* Saltos condicionales con GoTo [label] (condition)
* Evaluación de expresiones booleanas complejas

#### 3.2.4 Funciones Integradas

* **GetActualX():** Posición X actual de Wall-E
* **GetActualY():** Posición Y actual de Wall-E
* **GetCanvasSize():** Dimensiones del canvas
* **GetColorCount(color, x1, y1, x2, y2):** Conteo de píxeles por color
* **IsBrushColor(color):** Verificación del color actual
* **IsBrushSize(size):** Verificación del tamaño del pincel
* **IsCanvasColor(color, vertical, horizontal):** Verificación de color en posición relativa

### 3.3 Interfaz de Usuario

#### 3.3.1 Editor de Código

* Editor con numeración de líneas sincronizada
* Resaltado de línea actual durante depuración
* Resaltado de líneas con errores
* Funcionalidad de carga y guardado de archivos .pw

#### 3.3.2 Canvas Interactivo

* Renderización en tiempo real de píxeles
* Visualización de la posición de Wall-E
* Cuadrícula de referencia
* Redimensionamiento dinámico

#### 3.3.3 Sistema de Depuración

* Ejecución paso a paso
* Visualización del estado de variables
* Control de flujo (iniciar, pausar, continuar, detener)
* Panel de variables en tiempo real

### 3.4 Manejo de Errores

#### 3.4.1 Errores Léxicos

* Tokens inválidos con posición exacta
* Cadenas sin cerrar
* Caracteres no reconocidos

#### 3.4.2 Errores Sintácticos

* Estructuras gramaticales incorrectas
* Paréntesis y corchetes no balanceados
* Secuencias de tokens inválidas

#### 3.4.3 Errores Semánticos

* Variables no declaradas
* Tipos incompatibles
* Etiquetas duplicadas o no encontradas
* Parámetros incorrectos en funciones

#### 3.4.4 Errores de Ejecución

* Coordenadas fuera del canvas
* División por cero
* Límites de iteración excedidos

## 4. Algoritmos Implementados

### 4.1 Algoritmo de Bresenham para Círculos

Implementación optimizada para el dibujo de circunferencias mediante cálculo incremental:

int x = 0, y = radius;

int d = 3 - 2 \* radius;

while (y >= x) {

DrawCirclePoints(centerX, centerY, x, y);

if (d > 0) {

y--;

d = d + 4 \* (x - y) + 10;

} else {

d = d + 4 \* x + 6;

}

x++;

}

### 4.2 Algoritmo de Flood Fill Iterativo

Implementación no recursiva para evitar desbordamiento de pila:

var stack = new Stack<(int x, int y)>();

stack.Push((startX, startY));

while (stack.Count > 0) {

var (x, y) = stack.Pop();

if (canvas[y, x] == targetColor) {

canvas[y, x] = CurrentColor;

// Agregar píxeles adyacentes

}

}

### 4.3 Análisis Sintáctico Descendente Recursivo

Implementación de parser predictivo para la gramática del lenguaje:

private AdditiveExpression ParseAdditiveExpression() {

MultiplicativeExpression left = ParseMultiplicativeExpression();

while (currentToken.Type == Tipo.SUM || currentToken.Type == Tipo.SUB) {

Tipo operatorType = currentToken.Type;

Advance();

MultiplicativeExpression right = ParseMultiplicativeExpression();

return new AdditiveExpression(left, right, operatorType, line, column);

}

return new AdditiveExpression(left, line, column);

}

## 5. Resultados y Pruebas

### 5.1 Casos de Prueba Implementados

#### 5.1.1 Pruebas Básicas

* Dibujo de líneas en todas las direcciones
* Creación de círculos de diferentes tamaños
* Dibujo de rectángulos con dimensiones variables
* Cambio de colores y tamaños de pincel

#### 5.1.2 Pruebas de Control de Flujo

* Bucles con GoTo y etiquetas
* Condiciones complejas con operadores lógicos
* Anidamiento de estructuras de control

#### 5.1.3 Pruebas de Manejo de Errores

* Código con errores léxicos, sintácticos y semánticos
* Situaciones límite en coordenadas
* Manejo de memoria en operaciones intensivas

### 5.2 Rendimiento

* **Tiempo de compilación:** < 100ms para programas típicos
* **Capacidad del canvas:** Hasta 1000x1000 píxeles
* **Límite de ejecución:** 10,000 instrucciones para prevenir bucles infinitos

### 5.3 Funcionalidades Adicionales Implementadas

#### 5.3.1 Sistema de Depuración Avanzado

* Ejecución paso a paso con visualización en tiempo real
* Inspección de variables durante la ejecución
* Manejo de breakpoints implícitos

#### 5.3.2 Optimizaciones

* Agrupación de píxeles por color para renderización eficiente
* Validación temprana de límites del canvas
* Reutilización de estructuras de datos

## 6. Dificultades Encontradas y Soluciones

### 6.1 Desafíos Técnicos

#### 6.1.1 Sincronización del Scroll en el Editor

**Problema:** Mantener sincronizados los números de línea con el contenido del editor.  
**Solución:** Implementación de un contenedor único con scroll compartido y cálculo dinámico de alturas.

#### 6.1.2 Manejo de Estado en Depuración

**Problema:** Preservar el estado entre pasos de depuración sin afectar el rendimiento.  
**Solución:** Implementación de una clase DebugInterpreter que extiende el intérprete básico con capacidades de pausa y reanudación.

#### 6.1.3 Precisión en el Posicionamiento de Wall-E

**Problema:** Discrepancia entre la especificación y la implementación inicial del posicionamiento final en DrawLine.  
**Solución:** Corrección para que Wall-E termine exactamente en el último píxel dibujado, no uno más allá.

### 6.2 Desafíos de Diseño

#### 6.2.1 Arquitectura Extensible

**Problema:** Crear un sistema que permita agregar nuevas instrucciones fácilmente.  
**Solución:** Uso de patrones de diseño como Strategy y Factory para separar la lógica de cada instrucción.

#### 6.2.2 Manejo de Errores Contextual

**Problema:** Proporcionar información de error útil con contexto de línea y columna.  
**Solución:** Propagación de información de posición a través de todos los nodos del AST.

## 7. Conclusiones

### 7.1 Objetivos Alcanzados

El proyecto ha cumplido exitosamente con todos los requisitos especificados:

* ✅ Implementación completa del compilador en todas sus fases
* ✅ Interfaz gráfica funcional y intuitiva
* ✅ Sistema de depuración avanzado
* ✅ Manejo robusto de errores
* ✅ Soporte para todas las instrucciones del lenguaje
* ✅ Funcionalidades adicionales que mejoran la experiencia del usuario

### 7.2 Aprendizajes Obtenidos

#### 7.2.1 Conceptos Teóricos

* Comprensión profunda de las fases de un compilador
* Aplicación práctica de algoritmos de parsing
* Implementación de máquinas de estado para análisis léxico
* Diseño y construcción de ASTs

#### 7.2.2 Habilidades Técnicas

* Programación orientada a objetos en C#
* Desarrollo web con ASP.NET Core
* Manipulación del DOM con JavaScript
* Algoritmos de gráficos por computadora

#### 7.2.3 Ingeniería de Software

* Importancia de la arquitectura modular
* Técnicas de depuración y testing
* Documentación y mantenibilidad del código
* Manejo de control de versiones con Git

### 7.3 Posibles Mejoras Futuras

#### 7.3.1 Extensiones del Lenguaje

* Soporte para funciones definidas por el usuario
* Estructuras de datos más complejas (arrays, objetos)
* Operaciones de manipulación de imágenes
* Importación y exportación de formatos de imagen estándar

#### 7.3.2 Optimizaciones de Rendimiento

* Compilación a código intermedio
* Optimizaciones de loop unrolling
* Paralelización de operaciones de dibujo
* Caching inteligente de resultados

#### 7.3.3 Mejoras de Interfaz

* Autocompletado de código
* Resaltado de sintaxis en tiempo real
* Sistema de themes/temas
* Exportación de animaciones

## 8. Reflexión Personal

Este proyecto ha representado una oportunidad invaluable para aplicar los conceptos teóricos de programación y compiladores en un contexto práctico y visual. La experiencia de construir un sistema completo desde cero ha proporcionado una comprensión profunda de:

**La complejidad inherente en el diseño de lenguajes:** Cada decisión de diseño tiene ramificaciones en todas las fases del compilador.

**La importancia de las pruebas sistemáticas:** Los bugs pueden aparecer en cualquier fase y su detección temprana es crucial.

**El valor de la documentación:** Un código bien documentado facilita enormemente el mantenimiento y la extensión.

**La satisfacción de crear algo funcional:** Ver el código transformarse en arte visual es tremendamente gratificante.

El proyecto Pixel Wall-E no solo ha sido un ejercicio académico, sino una demostración práctica de cómo los conceptos fundamentales de ciencias de la computación se combinan para crear herramientas útiles y expresivas. La experiencia adquirida será invaluable para futuros proyectos más complejos y especializados.