

PixelScript: Uma Linguagem de Programação para Arte Procedural

Engenharia de Computação

Lógica da Computação 2025.2

> Raphael Cavalcanti Banov

Motivação do Projeto

Objetivo Principal: Teoria da Computação

Criar um compilador completo do zero para demonstrar o entendimento dos conceitos fundamentais de:

- Análise Léxica (Scanner com Flex)
- Análise Sintática (Parser com Bison)
- Geração de Código (Tradução para JavaScript)

O Compilador como Prova de Conceito.

Foco da Linguagem: Pixel Art Procedural

PixelScript foi concebida para a criação de Arte Procedural de Baixo Nível (Pixel Art), usando comandos simples de desenho.

O Desafio: Turing-Completeness

Implementar todas as estruturas de controle essenciais para garantir a funcionalidade completa da linguagem:

- Aritmética;
- Declaração de variáveis
- Laços de iteração
- Condicionais

Arquitetura do Compilador PixelScript



Características da Linguagem PixelScript

Estruturas de Controle

VAR nome = valor

Declara e inicializa uma variável

SE (condição) { ... }

Executa bloco se condição for verdadeira

SENAO { ... }

Executa bloco alternativo (com SE)

REPETIR (n) VEZES { ... }

Executa bloco n vezes em loop

Operadores: ==, !=, <, >, <=, >=

Comparação e aritmética (+, -, *, /)

Comandos de Desenho

CANVAS (largura, altura)

Define o tamanho da tela de desenho

COR (r, g, b)

Define a cor em RGB (0-255)

MOVER (x, y)

Move a "caneta" para a posição (x, y)

LINHA (x, y)

Desenha linha até a posição (x, y)

PONTO

Desenha um pixel na posição atual

RETANGULO (largura, altura)

Desenha retângulo preenchido

Curiosidade Técnica: O Problema do "Dangling Else"

O Problema

A ambiguidade do **SE/SENAO** é um clássico em compiladores:

```
SE (x > 0) SE (y > 0) PONTO SENA0 MOVER(0, 0)
```

Pergunta: O **SENAO** pertence ao primeiro **SE** ou ao segundo?

```
// Interpretação 1 (Errada): if (x > 0) { if (y > 0) { PONTO } else { MOVER(0, 0) } } // Interpretação 2  
(Correta): if (x > 0) { if (y > 0) { PONTO } else { MOVER(0, 0) } }
```

A Solução no Bison

Usamos regras de **comando_fechado** e **comando_aberto** para forçar a associação correta:

```
> comando_fechado: | SE (cond) { bloco } SENA0 { bloco } | comando_nao_se  
> comando_aberto: | SE (cond) { bloco }  
> comando_nao_se: | PONTO | LINHA | ...
```

Resultado: Um **SENAO** sempre se liga ao **SE** mais próximo, resolvendo a ambiguidade.

Exemplo Prático: Código PixelScript - Fibonacci.pixel

```
VAR tamanho_canvas = 200
CANVAS tamanho_canvas, tamanho_canvas

VAR centro_x = tamanho_canvas / 2
VAR centro_y = tamanho_canvas / 2
VAR x = centro_x
VAR y = centro_y
MOVER x, y // Inicia a caneta no centro

VAR escala = tamanho_canvas / 50
VAR direcao = 0
VAR passos = 1
VAR contador_passos = 0
VAR contador_giros = 0

// --- Loop Principal ---
REPETIR 100 VEZES {
    SE direcao == 0 { COR 255, 0, 0 } // Vermelho (Direita)
        SENO {
            SE direcao == 1 { COR 0, 255, 0 } // Verde (Cima)
            SENO {
                SE direcao == 2 { COR 0, 0, 255 } // Azul (Esquerda)
                SENO {
                    SE direcao == 3 { COR 255, 165, 0 } // Laranja (Baixo)
                }
            }
        }
}
```

```
VAR proximo_x = x      // Calcula o próximo ponto
VAR proximo_y = y

SE direcao == 0 { proximo_x = x + (passos * escala) } // Direita
SENAO {
    SE direcao == 1 { proximo_y = y - (passos * escala) } // Cima
    SENAOS {
        SE direcao == 2 { proximo_x = x - (passos * escala) } // Esquerda
        SENAOS {
            SE direcao == 3 { proximo_y = y + (passos * escala) } // Baixo
        }
    }
}
LINHA proximo_x, proximo_y // Desenha a linha para o próximo ponto
x = proximo_x               // Atualiza a posição da caneta para o final da linha
y = proximo_y
MOVER x, y

direcao = direcao + 1          // Lógica de giro
SE direcao > 3 {
    direcao = 0
}
contador_giros = contador_giros + 1
SE contador_giros == 2 {
    passos = passos + 1
    contador_giros = 0
}
```

Exemplo Prático: Resultado Visual

A Espiral de Fibonacci

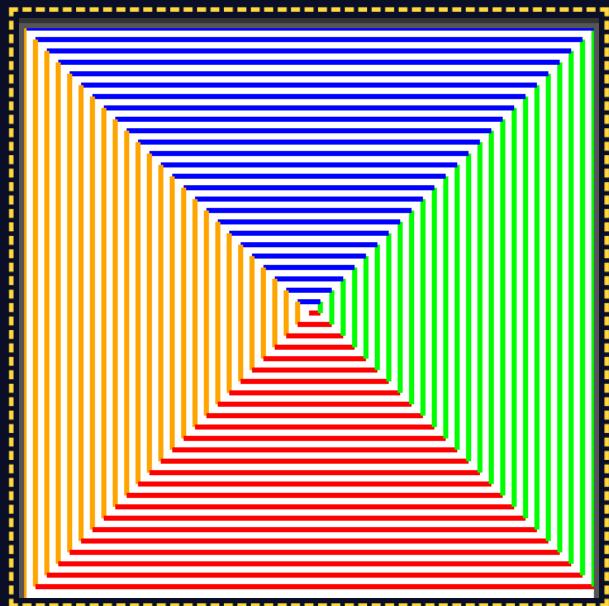
O resultado visual é uma espiral de Arquimedes desenhada usando a sequência de Fibonacci.

Características Visuais:

- Linhas de diferentes comprimentos seguindo a sequência de Fibonacci (1, 1, 2, 3, 5, 8, 13, ...).
- Cores que mudam a cada segmento, ilustrando o fluxo de execução.
- Estrutura espiral que cresce organicamente

Demonstração de Conceitos:

- Variáveis
- Loops
- Condicionais
- Comandos de desenho



Conclusão

- ✓ Compilador completo
- ✓ Linguagem Turing-completa com estruturas de controle essenciais
- ✓ Implementação prática de Flex (análise léxica) e Bison (análise sintática)
- ✓ Geração de código JavaScript e execução em VM (navegador)
- ✓ Resolução de ambiguidades clássicas (dangling else)

Obrigado!