1. Introducción

Pixel Wall-E es una aplicación de creación de *pixel-art* mediante un lenguaje de programación personalizado. Desarrollada en **C#** con el motor **Godot 4.4.1**, permite controlar un robot virtual (Wall-E) que pinta píxeles en un canvas cuadriculado. El proyecto integra un editor de código, ejecución de comandos, gestión de errores y exportación/importación de archivos (`.pw`).

2. Arquitectura y Tecnologías

- **Lenguaje**: C#
- **Motor**: Godot 4.4.1 (compatibilidad multiplataforma).

3. Implementación Clave

3.1. Lenguaje de Comandos

- **Sintaxis**: Secuencial, con instrucciones, asignaciones, funciones y saltos condicionales. Después de cada uno de estos comandos debe existir un salto de línea (a menos que sea la última línea por supuesto).
- **Ejemplo**:

```
•  Spawn(10, 10)           # Posición inicial
   Color(Blue)             # Pincel azul (puse los colores sin
comillas)
   Size(3)                 # Grosor 3 píxeles
   DrawLine(1, 0, 5)       # Línea horizontal
```

- **Validación**:
  - Errores léxicos (ej: @ es un caracter inesperado) detectado en tiempo de compilación.
  - Errores sintácticos (ej: `Spawn` duplicado) detectados en tiempo de compilación.
  - Errores en tiempo de ejecución (ej: coordenadas del Spawn fuera del canvas) reportados durante ejecución.

- **Canvas**:
  - Una sección cuadriculada. Cada casilla representa un pixel. El canvas se modifica en función del código cuando este se ejecuta.
  - Se usó el nodo de Godot TextureRect para generarlo.
  - Si el canvas ya estaba modificado por una ejecución anterior, la segunda ejecución comenzará con el canvas modificado.
  - Si se ejecutó un código válido sintáctica y semánticamente que dio un error en tiempo de ejecución, el código ejecutado antes del error se debe quedar aplicado en el canvas.
  - Cuando se cambia el tamaño del canvas se limpia este también. Todas las casillas comienzan en blanco de nuevo.
  - Si se le quitan las líneas verticales y horizontales al canvas y se establece cada píxel de la pantalla como un píxel del canvas Walle dibuja mucho más rápido.

- **Controles**:

  - *Botones*: Ejecutar, Guardar, Cargar `.pw`. Se usaron 3 nodos Button de Godot.
  - *SpinBox*: Redimensionar canvas. Se usó el nodo SpinBox de Godot.

3.3. Funcionalidades

- **Variables y Expresiones**:
  - Solo soporta asignación para operaciones aritméticas (`n <_ 5 + (3 * 2)`) o booleanas (`m <_ (3 < 4)`).
  - Sintaxis de asignación: variable <_ expresión. Variable es una cadena de texto formada por los 26 caracteres del alfabeto inglés (incluí mayúsculas y minúsculas) además puede contener caracteres numéricos y el símbolo _ pero no puede empezar con ninguno de estos y expresión es cualquier expresión aritmética o booleana.
- **Saltos Condicionales**:
  - Un salto condicional tiene la forma GoTo [label] (condition).
  - Un label es una etiqueta declarada en el código (puede ser antes o después del GoTo) y condition es una variable booleana. Si condition es true el código continua su ejecución en la línea de la etiqueta correspondiente.
- **Funciones Nativas**:
  - Las funciones retornan un valor numérico que se puede asignar a una variable. Reciben como entrada variables o literales (no otras funciones).
- **Instrucciones**:
  - Las instrucciones afectan a Wall-E y/o al canvas directamente. Reciben como entrada variables o literales.
  - Agregué la instrucción print que va acompañada de una expresión y lo que hace es mostrar el valor de dicha expresión, esto es muy útil pues permitió saber con mayor facilidad que mi intérprete estuviera funcionando correctamente.

4. Estructura

El proyecto se divide en varias clases separadas en distintos archivos .cs cada una con una función específica.

-Clases principales:

- Canvas: se encarga de crear el canvas en godot.
- Walle: en esta clase implemento todas las instrucciones que se encargan de dibujar sobre el canvas.
- Scanner: es la clase que se encarga de realizar el análisis léxico del código, tokenizando los caracteres válidos.
- Parser: su función es analizar sintácticamente el código, para ello construye un AST.
- Expr: define nodos para expresiones (todo lo que devuelve un valor).
- Stmt: define nodos para sentencias (acciones que no devuelven un valor).
- Interpreter: evalúa todas las expresiones e instrucciones del lenguaje.

- Program: se encarga de llamar al Scanner, Parser e Interpreter.
- MainScript: es la clase principal pues se encarga de unir la parte visual con la parte que se encarga de interpretar.

5. Conclusiones

Pixel Wall-E demuestra cómo un lenguaje minimalista puede generar arte digital complejo. La integración de Godot con C# permitió crear una UI fluida y un sistema de comandos robusto, mientras que la extensibilidad del código facilita añadir nuevas funcionalidades