Tercer Proyecto de Programación GeometricWall

Richard Eugenio Ávila Entenza Manuel Alejandro Rodriguez

Diciembre, 2023

Facultad de Matematica y Computacion.

Resumen

Como objetivo del Tercer Proyecto de Programación de la Facultad de Matemática y Computaciñon de la Universidad de la Habana se realizo el desarrollo del proyecto GeometricWall. Para el desarrollo de este proyecto se desarrolló una aplicación de Windows Presentation Foundation (WPF) con tecnología .Net 7.0 y C# como lenguaje de programación y XAML para implementar el diseño de la mismas. Se sugiere continuar con el desarrollo de la misma para ampliar el lenguaje e implementar en totalidad todas sus funciones.

Índice

1. Introducción

El proyecto se realiza con el objetivo de dar solución a la orientación del proyecto GeometricWall, para ello se hace uso de una aplicación de Windows Presentation Foundation (WPF), la misma se encarga de interpretar el lenguaje G# y servir los resultados del código que ha sido analizado previamente. La aplicación consta de un solo proyecto formado por un conjunto de bibliotecas de clases que serán las que se analizarán en este informe. Las bibliotecas de clases que serán abordadas en este informe son: Lexer, Parser, Interpreter y adicionalmente las bibliotecas Draw y Geometric las cuales no son necesarias abordar tanto

2. Desarrollo

La implementación de las funcionalidades de **GeometrciWall** está dividida en tres conjuntos de bibliotecas de clases:

- **Lexer:** encargado de procesar la instrucción escrita por el usuario y transformarla a una cadena de *Tokens*.

- **Parser:** encargado de procesar todos los Tokens enviados por el Lexer y construir un **Abstract Syntax Tree (AST)**.

-**Intérprete:** recorre y ejecuta el AST para evaluar este y devolver su valor.

A su vez se encuentran las bibliotecas de clases **Geometric** y **Draw**. Geometric contiene las cuales contienen las declaraciones de las figuras geométricas que están disponibles directamente mediante las instrucciones de G# y Draw se encarga de representar las funciones en la interfaz gráfica e interceptar las figuras para así obtener los puntos de intercepción de las mismas

2.1. Interfaz Gráfica.

La interfaz gráfica de GeometricWall presenta un diseño sencillo pero atractiva a la vez, se utiliza XAML para la construcción y estructura de la mismas. La interfaz cuenta con un panel de control en el lateral izquierdo en el cual se encuentran tres botones de arriba hacia abajo, Nuevo, Cargar y Guardar. En la parte superior de la interfaz se encuentran el título y el botón de salida de la aplicación. En el centro de la misma nos encontramos con tres estructuras de WPF, un TextBox, el botón de Interpretar y un Canvas.

2.1.1. Funcionalidad de los botones

- **Botón Nuevo:** Limpia todo el texto que tenemos en el TextBox esto sería un inicia limpio para comenzar a escribir códigos de G# una vez ya se tenía escrito algo en la misma, todo el código que se escribió anteriormente se perdería si no se tiene un respaldo del mismo.

- **Botón Cargar:** Como su nombre indica permite cargar un documento existente y cargarlo al TextBox, este se podría ejecutar satisfactoriamente en caso del mismo ser válido. Solamente pueden cargase documentos con la extensión .geo.

- **Botón Guardar:** Permite guardar el código del TextBox en la ubicación de su selección dicho documento se guardará con la extensión. geo. No es necesario ejecutar el código antes de guardar el mismo, aunque se recomienda hacerlo para verificar los resultados del mismo.

- **Botón Interpretar:** Ejecuta el proceso de interpretar el código existente en el TextBox. Botón Salir: Cierra la aplicación automáticamente.

2.1.2. Funcionalidad del TextBox y Canvas

Todo el código que se desee escribir para ser interpretado o guardado deberá ser escrito en el TextBox, no hay límite de línea o carácter a ser escritos el único requisito seria asegurar que el código que está siendo escrito sea un programa valido de G#. El Canvas se encarga de representar los resultados gráficos de los programas escritos en G#, su funcionalidad es similar a un eje de coordenadas, pero se trabaja en pixeles, el Canvas

utiliza tiene una dimensión de 548x548 siendo (0,0) en la esquina superior izquierda y el (548,548) en la esquina inferior izquierda.

2.2. Funcionalidad de la aplicación

La aplicación funciona de la siguiente forma, al ejecutar el botón de interpretar se inicializa una instancia de la clase Lexer que recibe un texto dicho texto es el código que se ha escrito es cual se guarda en la propiedad Text del TextBox, se inicializa una instancia de la clase Parser que recibe un Lexer, el mismo que ya ha sido inicializado, adicionalmente se inicializan las instancias de las clases SymbolTable y Draw, esta última recibe como referencia el Canvas de nuestra aplicación. Finalmente se crea una instancia de la clase Interpreter el cual recibe como parámetros las instancias de Parser, SymbolTable y Draw que han sido creadas. La clase Interpreter contiene el método Interpret el cual inicia la construcción de AST y evalúa el mismo.

2.2.1. Lexer.

La biblioteca de clases Lexer está compuesta por dos clases fundamentales Token y Lexer. En la clase Token se declaran la estructura y los tipos de token aceptados en el lenguaje, a su vez cada vez que una función es declarada se guarda en una lista inicializada en la misma clase de este modo se tendrá referencia a que tipo de Token de tipo ID es el llamado de una función que ha sido declarada.

La clase Lexer es la encargada de convertir las cadenas de texto a token, el método principal de la clase es GetNextToken() el cual es llamado desde el Parser haciendo uso la instancia del Lexer, este método indica al Lexer a retornar el próximo token. Adicionalmente se tienen métodos auxiliares para tomar avanzar al próximo carácter de la cadena de texto, eliminar los caracteres que corresponden con espacios en blanco y saltos de línea, así como identificar una cadena alfanumérica que tipo de token es el que le corresponde a dicha cadena.

2.2.2. Parser.

Utilizando parsing recursivo descendente se construye un Árbol de Sintaxis Abstracta (AST), siguiendo la jerarquía del lenguaje, un programa de G# es un árbol donde cada línea de código es una rama de la raíz, la raíz del árbol es un node de tipo ProgramAST.

- Jerarquía del Lenguaje:

ProgramAST: Lista de Statement.

Statement: DrawStatement, MeasureStatement, ConstantStatement, FunctionStatement, SecuenceStatement, IntersectStatement, LetInStatement, IfStatement o Expr.

DrawStatement: Lista de Var o GeometricDeclaration

MeasureStatement: Dos Var

ConstantStatement: Assign

FunctionStatement: Var y Statement

SecuenceStatement: Lista de Var

IntersectStatement: Dos Expr

LetInStatement: Lista de Statement y un Statement a evaluar.

IfStatement: Condicion: Statament y dos Statement a evaluar.
Expr: Term(+ | -)
Term: Factor(* | /)
Factor: Operadores Unarios, Number, GeometricDeclaration, Var o Expr
GeometricDeclaration: Declaraciones geométricas.
Assign: Var y un Statement

- **Nodos del lenguaje:**
Todos los nodos heredan de la clase abstracta AST.
ProgramAST: contiene una lista de AST.
DrawStatement: contiene una lista de figuras a graficar o la declaración de una.
MeasureStatement: contiene dos AST que serán las declaraciones de los puntos a ser medidos.
IntersectStatement: contiene dos AST que serán las declaraciones de las figuras a ser interceptadas.
Assign: Contiene la declaración de una variable.
Var: Llamado de una variable.
Secuence: Lista de AST.
LetIN: Contiene una lista de AST y un AST que será evaluado.
IfElse: Contiene una AST (condición) y los AST a ser evaluados.
FunctionDeclaration: Contiene la declaración de una función.
FunctionCall: Llamado de una function.
ORNode y ANDNode: Contiene los operadores logivos or y and respectivamente y dos AST
LogicOP: Contiene los operadores de comparación y dos AST.
PointAST, CircleAST, LineAST, SegmentAST, RayAST: Contiene las declaraciones de las figuras.
UnaryOP: Operador unario de los AST en caso de existir.
Num: Numero.
BinOp: Contiene las operaciones binarias.
Una vez contruido el árbol este es evaluado en el Interpreter.

2.2.3. Interpreter.

El **Interpreter** es el encargado de evaluar el AST obtenido del Parser, una vez se ha creado este se recorre el árbol *in-order* hasta obtener el valor de la raíz del árbol, si durante las visita a los nodos ocurre algun error semántico ya sea con tipos de datos, argumentos de funciones o mal uso de operadores se envía un excepción describiendo el error cometido.

El intérprete consta de tres bibliotecas de clases:

NodeVisitor se encarga de invocar el metodo de visita al node correspondiente, siempre que se reliza la visita de un nodo se llama el método Visit que recibe un nodo de tipo dynamic de NodeVisitor, segun el tipo de nodo que ha recibo el metodo se genera la visita correspondiente a dicho nodo. Por ejemplo si se envía un nodo de tipo **BinOP** se genera la visita **Visit_BinOP**.

SymbolTable tabla de símbolos encarga de guarda los valares y las expresiones de las variables y las funciones respectivamente para luego ser utilizadas, tambien se manejan

los ámbitos de declaraciones de la variables y los tipos de datos de las mismas. Destarcar de la tabla de símbolos dos aspectos principales acerca de los ámbitos de las variables, cada vez que se visita un nodo **LetInStatement**, se crea un nuevo ámbito de variables utilizando un **Stack**, de este modo en caso de existir en el nuevo ámbito una variable de igual nombre a otra que ya ha sido declarada, esta última, no sería la de mayor prioridad al llamarse, tambien al hacer un llamdo a una función se crea un nuevo ámbito temporal con los valores de los argumentos para así ejecutar la expresión de la función y devolver su valor, una vez devuelvo el resultado del llamado de la función este ámbito deja de existir.

Interpreter biblioteca donde se implementan las visitas correspondientes a cada nodo y contiene el método **Interpret** encargado de evaluar el AST. Ya que el árbol queda estructurado de manera jerarquica utilizando las reglas gramaticales aplicadas en el Parser, simplemente se encarga de visitar y devolver los resultados de las mismas.

3. Conclusiones

Finaliza el desarrollo de este proyecto se ha logrado implenetar una aplicación de GeometricWall y un interprete de G#, habíendose cumplido con el objetivo del Tercer Poyecto de Programación. Cabe destacar que aun quedan detalles por mejorar y optimizar en el proyecto, se espera en futuro realizar estas.